UNIVERSIDAD DE EL SALVADOR

FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE

DEPARTAMENTO DE INGENIERÍA Y ARQUITECTURA

TRABAJO DE GRADO

"ESTUDIO, DISEÑO Y DESARROLLO DE UNA SOLUCIÓN INFORMÁTICA BASADA EN BIOMETRÍA, PARA EL CONTROL DE ACCESO PEATONAL AL CAMPUS DE LA FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE."

PRESENTADO POR:

BR. VÍCTOR MANUEL RODRÍGUEZ UMAÑA

PARA OPTAR AL GRADO DE:

INGENIERO DE SISTEMAS INFORMÁTICOS

DOCENTE DIRECTOR:

ING. ERNESTO ALEXANDER CALDERÓN PERAZA

EL SALVADOR, CENTRO AMÉRICA

OCTUBRE DE 2011

UNIVERSIDAD DE EL SALVADOR

RECTOR:

MÁSTER RUFINO ANTONIO QUEZADA SÁNCHEZ

VICERRECTOR ACADÉMICO:

MÁSTER MIGUEL ÁNGEL PÉREZ RAMOS

VICERRECTOR ADMINISTRATIVO:

MÁSTER ÓSCAR NOÉ NAVARRETE

PRESIDENTE DE LA ASAMBLEA GENERAL UNIVERSITARIA:

DOCTOR LUIS GILBERTO PARADA GÓMEZ

FISCAL GENERAL:

DOCTOR RENÉ MADECADEL PERLA JIMÉNEZ

SECRETARIO GENERAL:

LICENCIADO DOUGLAS VLADIMIR ALFARO CHÁVEZ

FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE

DECANO:

LICDO. JORGE MAURICIO RIVERA

VICEDECANO:

LICDO. MSE. ELADIO EFRAÍN ZACARÍAS

ADMINISTRADOR ACADEMICO:

LICDO. GERBERT SALVADOR RIVAS

JEFE DEL DEPARTAMENTO DE INGENIERÍA Y ARQUITECTURA:

ING. RAÚL ERNESTO MARTÍNEZ BERMÚDEZ

TRABAJO DE GRADUACIÓN APROBADO POR:

_____

ING. ERNESTO ALEXANDER CALDERÓN

DOCENTE DIRECTOR

DEDICATORIA

- En primer lugar, quiero dedicar este trabajo a **Dios** todopoderoso, quien me dio la vida y me ha acompañado, guiado y fortalecido. Me ha permitido llegar hasta este momento y puedo asegurar que sin Él no habría sido posible.

- A **mis padres**, **Gustavo Rodríguez y Eva Umaña de Rodríguez.** En especial, agradezco a mi madre por su amor incondicional y constante, aún en las circunstancias más difíciles; por haber hecho su mejor esfuerzo y por ayudarme a llegar a la culminación de mi carrera universitaria.

- A **mis abuelos**, y en especial a **mis abuelos paternos**, **Víctor Rodríguez y Socorro Calderón de Rodríguez** (Q.E.P.D.), quienes me han apoyado incondicionalmente más que nadie para que esta meta se hiciera una realidad, aún cuando las cosas parecían inciertas.

- A **mi amada novia**, **Wendy Sorayda Paiz**, por ser alguien de apoyo en mi vida y estar conmigo en los momentos

difíciles, animándome e inspirándome a seguir adelante dando lo mejor de mí; así como **a su familia**, por ser tan especiales conmigo.

- A **mis tíos**, especialmente a **Antonia Rodríguez** (Q.E.P.D.), **Walter Umaña** y **Sara Elida Umaña**, quienes en más de algún momento fueron determinantes para que yo no abandonara este camino.

- A **mis hermanos**, **Oscar y Elenita Rodríguez**, por su cariño y aprecio; especialmente a mi hermano por ser además de un hermano, un gran amigo.

- A **mis primos**, por su cercanía; especialmente a **Luis Funes**, cuya ayuda fue crucial en uno de los momentos más difíciles de mi carrera.

- A los **esposos Joel Guzmán y Nohemí Zarceño**, quienes me animaron y apoyaron en muchos de los momentos que más lo necesité.

- A mis grandes amigos **Samuel Martínez** y **Marlon Rodríguez,** con quienes he podido compartir muchas de mis principales aflicciones, y en quienes he encontrado grandes consejos y palabras de ánimo y esperanza.

- A **mis compañeros y amigos de la universidad**; en particular a **Rodrigo Magaña** y **Roberto Minero**, junto a quienes tuve la oportunidad de trabajar, compartiendo conocimientos, ideas y preocupaciones, además de buenos momentos; así también a **Benjamín Rivera**, a quien considero un gran amigo incondicional, ganado en las situaciones adversas del comienzo de nuestra carrera.

- A mi maestro y amigo **Juan Carlos Peña**, con quien tuve la oportunidad de aprender mucho y pasar buenos momentos durante el cumplimiento del servicio social.

- A **mis compañeros de casa** de esta etapa estudiantil. Sé que dentro de poco también llegarán a un momento como éste.

- A **todos mis familiares, amigos y amigas** que en algún momento fueron parte de esta etapa que hoy llega a su fin.

AGRADECIMIENTOS

Tengo mucha gente a la cual agradecer tanto la culminación de mi carrera, como de este trabajo en particular; y a leer esto cada uno de ellos sabe que forma parte de ese grupo. Sin embargo me debo limitar a mencionar un pequeño grupo de esas personas, sin quienes no hubiera sido posible la culminación de este proyecto.

Sé que las palabras no son suficientes, pero quiero expresar mi sincero agradecimiento a:

- **Dios**, quien me dio la vida y la oportunidad de ver realizado este sueño que en muchos momentos parecía incierto. Gracias Dios, por haber estado a mi lado, guiándome, impulsándome a seguir adelante, siendo mi consuelo en los momentos difíciles y mi fortaleza en la debilidad, y por haber provisto todo lo necesario para llegar a este momento que muchas veces creí imposible.

- **Mi madre**, **Eva Magdalena de Rodríguez**, por su amor, comprensión, ayuda y todo lo demás. Simplemente puedo decir: ¡Gracias mamá!

- **Mi querida novia, Wendy Sorayda Paiz y su familia,** quienes han sido mucho más que especiales para mí y me han apoyado de muchas formas. Preciosa, te amo y te agradezco por todos los momentos especiales que hemos podido compartir; por animarme y creer en mí aún en los momento más difíciles; realmente sos una bendición. ¡Gracias por todo!

- **Mis abuelos paternos, Víctor Manuel Rodríguez y Socorro Calderón de Rodríguez** (Q.E.P.D.), por ser un ejemplo de superación para mí, por su cercanía, y por hacer de esta meta un sueño propio que me ayudaron a cumplir. No existe como pagarles, pero siempre los llevaré en el corazón y los recordaré como quienes Dios preparó para que transitar por este camino fuera posible. Mamita y papá Víctor, las palabras no son suficientes, pero sepan que les agradezco con la mayor sinceridad de mi corazón.

- **Mis abuelos maternos, Manuel Umaña y Marta Lemus de Umaña,** por sus constantes oraciones, por creer en mí y por motivarme a seguir adelante confiando en Dios. ¡Muchas gracias a ambos!

- **Mi hermano, Oscar Gustavo Rodríguez,** por ser un gran hermano y un gran amigo a la vez, y por ayudarme de diferentes maneras para la culminación de este proyecto. ¡Mil gracias hermano!

- **A FEDISAL, y particularmente al Programa de Becas FANTEL,** porque sin ellos no habría sido posible llegar hasta aquí. Gracias a su personal, que muy amablemente me ha atendido.

- **El docente director, Ing. Ernesto Alexander Calderón Peraza,** por interesarse en este proyecto, y por toda su ayuda y asesoría en aspectos concernientes al proyecto, y por prestarme la atención de un amigo cuando las cosas no marchaban bien. ¡Muchísimas gracias ingeniero!

- **El coordinador general del proyecto, Ing. Raúl Ernesto Martínez Bermúdez,** por ser un excelente coordinador y por su atención y comprensión siempre que fue requerida. ¡Infinitas gracias!

- **Los ingenieros Juan Carlos Peña y José Francisco Andaluz,** por asesorarme en algunos aspectos técnicos vinculados al proyecto. Gracias por compartir sus conocimientos conmigo.

- **Mis tíos Walter Umaña y Humberto Funes,** por su disposición a ayudarme en diferentes aspectos de manera incondicional. Tío Walter y tío Beto, ¡Muchísimas gracias!

- **La familia Villafuerte,** quienes amable y desinteresadamente me cedieron un lugar donde alojarme durante la etapa de mi carrera con mayores dificultades económicas. Mi agradecimiento es grande y deseo que Dios los siga bendiciendo.

- **Mis amigos Rodrigo Magaña, Marcos Salazar, Wilfredo Polanco y Marvin Tejada,** que me tendieron la mano para hacer posible la realización de este trabajo de grado. Les agradezco mucho, ¡Que Dios recompense lo que han hecho por ayudarme!

ÍNDICE

CAPÍTULO I: GENERALIDADES DEL PROYECTO.....1

1.1 INTRODUCCIÓN.....2

1.2 OBJETIVOS.....4

1.3 ANTECEDENTES.....6

1.4 PLANTEAMIENTO DEL PROBLEMA.....7

1.5 JUSTIFICACIÓN.....9

1.6 ALCANCES.....11

1.7 LIMITACIONES.....12

CAPÍTULO II: ASPECTOS TEÓRICOS.....13

2.1 INTRODUCCIÓN A LA BIOMETRÍA.....14

2.2 SISTEMAS DE RECONOCIMIENTO BIOMÉTRICO.....16

2.3 COMPARACIÓN DE SISTEMAS BIOMÉTRICOS.....21

2.4 RECONOCIMIENTO DE HUELLA DACTILAR.....26

MEJORA DE LA CALIDAD DE LA IMAGEN 26

EXTRACCIÓN DEL PATRÓN BIOMÉTRICO. 35

RECONOCIMIENTO DE UNA HUELLA DACTILAR 44

CAPÍTULO III: DISEÑO DEL PROYECTO.....54
3.1 DESCRIPCIÓN DE LOS ELEMENTOS DEL SISTEMA.....55
3.2 DISEÑO DE LA BASE DE DATOS.....59
3.3 DISEÑO DEL SISTEMA ADMINISTRATIVO.....60
3.4 DISEÑO DEL SISTEMA DEL PUNTO DE ACCESO.....62
3.5 DISEÑO DEL DISPOSITIVO DE ACCESO.....64
CAPÍTULO IV: DESARROLLO DEL SISTEMA.....69
4.1 ASPECTOS GENERALES.....70
4.2 DESARROLLO DEL SISTEMA ADMINISTRATIVO.....72
CONTROLES PERSONALIZADOS 73
VALIDADORES 116
UTILIDADES VARIAS 124
PERSISTENCIA DE DATOS 133
MÓDULO DE BIOMETRÍA 159
INTERFAZ DE USUARIO 167
CLASES GENERALES 217
ARCHIVOS DE CONFIGURACIÓN 220
4.3 DESARROLLO DEL SOFTWARE DEL PUNTO DE ACCESO.....224
ELEMENTOS DE FUNCIONALIDAD Y LÓGICA 224

ELEMENTOS DE INTERFAZ DE USUARIO 237
MÉTODO DE ARRANQUE 252
4.4 PROGRAMACIÓN DEL HARDWARE.....253
ASPECTOS TEÓRICOS 253
CODIFICACIÓN DEL DISPOSITIVO 254
CAPÍTULO V: MANUALES DE USUARIO.....259
5.1 MANUAL DE USUARIO PARA EL SISTEMA ADMINISTRATIVO....260
TABLA DE CONTENIDO 260
DEFINICIONES 261
ASPECTOS GENERALES 263
ADMINISTRACIÓN DEL SISTEMA 266
GENERACIÓN DE REPORTES 282
5.2 MANUAL DEL PROGRAMA DE CONTROL DE ACCESO.....286
TABLA DE CONTENIDO 286
DEFINICIONES 287
ASPECTOS GENERALES 289
ARRANQUE DEL PROGRAMA 292
CONFIGURACIÓN DEL PUNTO DE ACCESO 294
IDENTIFICACIÓN DE USUARIOS 296

CIERRE DEL PROGRAMA 297

CAPÍTULO VI: PLAN DE IMPLEMENTACIÓN 298

6.1 REQUERIMIENTOS OPERATIVOS 299

REQUERIMIENTOS DE SOFTWARE 299

REQUERIMIENTOS DE HARDWARE 301

OTROS REQUERIMIENTOS 302

6.2 INVERSIÓN REQUERIDA 307

6.3 PASOS DE IMPLEMENTACIÓN 311

CAPÍTULO VII: CONCLUSIONES Y RECOMENDACIONES 314

7.1 CONCLUSIONES 315

7.2 RECOMENDACIONES 316

7.3 REFERENCIAS BIBLIOGRÁFICAS 317

ANEXOS 319

CAPÍTULO I:

GENERALIDADES DEL PROYECTO

1.1 INTRODUCCIÓN

La Universidad de El Salvador es una institución de educación superior sobresaliente a nivel nacional, tanto por su historia como por el compromiso con la calidad educativa que le ha caracterizado en el transcurrir de la misma.

Este prestigio, sin embargo, se ha visto altamente opacado por diferentes factores extra académicos que se han presentado para hacerse notar cada vez más como una amenaza latente a la Universidad y a las personas vinculadas a la misma (estudiantes, docentes, administrativos, etc.)

Uno, y quizás el más grave de dichos problemas, es el relacionado a la seguridad dentro del las instalaciones de la Universidad. En este tema, la Facultad Multidisciplinaria de Occidente ha sido grandemente afectada durante los últimos años y el problema parece ser cada vez peor.

Estos problemas de seguridad tienen uno de sus grandes orígenes en el hecho de que no existe ninguna forma de control sobre el acceso de las personas a las instalaciones.

Es por esta razón que se ha vuelto necesario y aún urgente un control fiable de las personas que ingresan al campus.

El presente documento es el resultado del estudio de soluciones biométricas y el posterior diseño y desarrollo de un sistema de control de acceso automatizado para la Facultad Multidisciplinaria de Occidente de la Universidad de El Salvador. A través de sus páginas encontrará aspectos generales sobre el proyecto y el problema que soluciona, además de aspectos teóricos requeridos para el estudio y desarrollo de la solución. Podrá revisar, además, la documentación concerniente a dicha solución, partiendo de su diseño y llegando hasta los manuales de usuario de las aplicaciones del sistema de acceso, así como su plan de implementación.

Una de las partes esenciales de este documento lo conforman el diseño y desarrollo del sistema que aquí se presenta como solución para el control de acceso a la Facultad Multidisciplinaria de Occidente, la cual incluye el diseño de la base de dato; el diseño, código fuente y manual de referencia del programador para cada una de las aplicaciones del sistema; así como el diseño del dispositivo hardware que lo complementa.

1.2 OBJETIVOS

GENERAL:

Desarrollar una solución informática, basada en biometría, para controlar de manera fiable y automatizada el acceso peatonal al campus de la Universidad de El Salvador, Facultad Multidisciplinaria de Occidente.

ESPECÍFICOS:

- Diseñar una base de datos escalable que permita almacenar la información requerida para el sistema de control de acceso peatonal de manera centralizada.
- Desarrollar un programa computacional eficiente y fidedigno que permita identificar usuarios a partir de sus patrones biométricos.
- Desarrollar una aplicación de escritorio para la administración centralizada del sistema de control de acceso peatonal.

- Elaborar una interfaz electrónica para el control del dispositivo mecánico de acceso desde el programa de identificación biométrica.
- Elaborar un sistema que permita controlar el acceso de personas a zonas restringidas dentro del campus.
- Desarrollar, en términos generales, una solución cuya implementación pueda ser factible económica y técnicamente hablando.

1.3 ANTECEDENTES

Al día de hoy, en la Facultad Multidisciplinaria de Occidente se han realizado algunos esfuerzos por controlar aspectos como el ingreso de vehículos al campus y el ingreso y egreso de equipo perteneciente a terceros con alguna similitud a los bienes de la universidad para evitar robos, sin embargo no ha habido una forma de establecer un control sobre la totalidad de personas que ingresan al campus.

Se ha tratado también de restringir el acceso de personas a ciertas áreas como oficinas administrativas y departamentos docentes, sin embargo no ha sido posible poder controlar este factor de tal forma que solamente las personas indicadas puedan ingresar a las diferentes zonas del campus.

En resumen, la situación actual de la Facultad Multidisciplinaria de Occidente referente al control de acceso es tal, que a cualquier persona, aún siendo totalmente ajena a la Universidad, se le facilita ingresar de manera libre y desapercibida al campus y a muchas de las zonas dentro del mismo.

1.4 PLANTEAMIENTO DEL PROBLEMA

La Facultad Multidisciplinaria de Occidente es una institución altamente afectada por los altos niveles de delincuencia del país. En el interior de su campus se han perpetrado robos de diferentes tipos, tanto a la institución misma como a personas vinculadas a ella; aún bienes de la empresa privada han sido atacados dentro de sus instalaciones.

El problema de seguridad es altamente complejo, sin embargo algo que vuelve a la FMO más vulnerable y hace aún más dificultoso el poder lidiar con él, es el hecho de que exista vía libre para ingresar al campus y en gran medida a las diferentes zonas dentro del mismo.

Recientemente, estudiantes de la Facultad han sido asaltados a mano armada dentro del campus; ahora bien, esto no es en sí mismo el problema, sino la consecuencia del verdadero problema: el hecho de que alguien totalmente ajeno a la institución y con malas intenciones haya podido ingresar.

Por lo anterior, se puede asegurar que gran parte del problema de inseguridad que se vive en la Facultad

Multidisciplinaria de Occidente es debido a la falta de un control de acceso al campus y a las diversas zonas dentro del mismo.

Por otro lado, utilizar un mecanismo enteramente humano sería poco eficiente y altamente costoso. Además, no gozaría de la fiabilidad que en materia de seguridad puede brindar un sistema automatizado que no da lugar al criterio o el error humano.

Dado lo anterior, se propone la elaboración de un sistema de control de acceso centralizado y basado en biometría, que permita restringir el acceso al campus y a sus zonas, a usuarios no registrados o que no gozan de los permisos requeridos.

1.5 JUSTIFICACIÓN

En los tiempos de crisis social que estamos viviendo, la seguridad es un tema al que hay que prestar mucha atención. Las estadísticas hablan por sí solas: un reciente estudio reveló que cerca del 8% del presupuesto de las empresas privadas de El Salvador es destinado al área de seguridad.

Para la Universidad de El Salvador, sin embargo, y en caso particular para la Facultad Multidisciplinaria de Occidente, no es igualmente factible realizar este tipo de inversiones a tal magnitud. Con todo, la necesidad de brindar soluciones de seguridad se ha vuelto urgente, y más aún cuando las mismas limitantes de la institución la hacen más vulnerable.

Como ya se ha mencionado, una de las necesidades más urgentes es poder establecer un mecanismo de control de acceso al campus y a sus diferentes zonas, dado que es un primer gran paso para velar por la seguridad dentro de las instalaciones de la Facultad.

A esto se suma el hecho de que los procesos automatizados son más ágiles y confiables que los procesos manuales, y que

además tienen costos significativamente bajos a comparación de estos últimos.

Todo lo anterior hace notoria la necesidad de contar con un sistema de control de acceso automatizado y centralizado, por lo cual se propone el desarrollo de dicha solución basada en biometría.

1.6 ALCANCES

El sistema de acceso peatonal propuesto en este trabajo de grado está enmarcado por los siguientes alcances:

- Diseño y desarrollo de un programa computacional de control de acceso basado en el reconocimiento biométrico de usuarios, según la técnica de reconocimiento biométrico que más se adecúe a la situación de la FMO.
- Diseño y elaboración de una base de datos centralizada para el sistema de control de acceso.
- Diseño y desarrollo de una aplicación de escritorio, para la administración del sistema de control de acceso.
- Diseño y elaboración de un dispositivo electrónico, que permita que el dispositivo mecánico de acceso sea controlado por el programa del punto de acceso peatonal.
- Generar la documentación de referencia para desarrolladores de la base de datos, el sistema administrativo y el programa de control de acceso.
- Elaborar los manuales de usuario final tanto para el sistema administrativo como para el programa de control de acceso.

1.7 LIMITACIONES

La mayor limitante para el desarrollo del proyecto es la escasez existente en nuestro entorno de diversos tipos de recursos requeridos para el mismo, como son dispositivos de captura de patrones biométricos, circuitos integrados, etc.

A esto se suma la inexistencia de experiencias domésticas previas en el ramo; así como el alto costo de herramientas para el desarrollo de tecnologías biométricas, por el lado de las soluciones comerciales, y la escasa documentación de las herramientas libres.

CAPÍTULO II:

ASPECTOS TEÓRICOS

2.1 INTRODUCCIÓN A LA BIOMETRÍA

A través de la historia se ha tomado como regla primordial de seguridad el hecho de que solamente las personas indicadas puedan tener acceso a un lugar, servicio, privilegio, etc. Para ello ha sido necesario poder identificar a las personas que pretenden dicho acceso, y esta tarea se ha llevado a cabo básicamente a partir de: algo que la persona tiene (una llave, una tarjeta, etc.) o algo que la persona sabe (una palabra clave, una contraseña, etc.)

El reconocimiento biométrico, por su parte, es el que se efectúa a través de algo que uno es; es decir, algo que forma parte de uno mismo, como es el caso del rostro, la voz, la huella dactilar, etc.

Desde la antigüedad se ha identificado a las personas a partir de sus rasgos biométricos, y es que ¿cómo reconocemos a las personas si no es a través de ellos?

Esta capacidad, sin embargo, se ha desarrollado de manera automatizada hasta no hace muchos años, pero con su llegada ha ido ganando la batalla a los métodos tradicionales de reconocimiento dado que son más fiables. Por ejemplo, al

basarse en algo que se tiene se corre el riesgo de que un impostor sea aceptado por tener el objeto que se supone debería tener únicamente el usuario legítimo. Por otro lado, basarse en algo que se conoce puede dar lugar a denegar al usuario legítimo por haber olvidado o confundido lo que sabía; o bien podría darse el caso en que por voluntad propia, por error o por descuido, otra persona obtenga el conocimiento requerido y esto le permita hacerse pasar por el usuario legítimo.

La capacidad de identificación de las personas de manera automatizada a partir de patrones biométricos permite obtener la fiabilidad deseada y es la alternativa ideal para un sistema de control de acceso peatonal. *(Redacción propia)*

2.2 SISTEMAS DE RECONOCIMIENTO BIOMÉTRICO

Existen diferentes sistemas de reconocimiento biométrico, también llamados "técnicas de reconocimiento biométrico". En el caso particular de este proyecto se han contemplado los sistemas brevemente descritos a continuación:

- *Reconocimiento de huella dactilar:* Aún antes del auge que la tecnología informática ha tomado en las últimas décadas, las huellas dactilares ya se utilizaban como un medio de reconocimiento único de seres humanos. Esta técnica se basa en el hecho comprobado de que la forma descrita por las crestas y los valles de los dedos es única para cada ser humano. Para fines informáticos, dicha forma es capturada por medio de un sensor, mismo que puede ser de diversas naturalezas tecnológicas, para luego ser contrastado con una o varias capturas previas existentes dentro de una base de datos, comúnmente también denominada "banco de huellas". *(Resumen, Zorita 2003)*

- *Reconocimiento facial:* Los sistemas de reconocimiento facial autentican o identifican una persona a partir de una imagen digital o de un fotograma tomado de alguna fuente de video. Básicamente, el proceso consiste en comparar ciertos rasgos faciales de la imagen capturada frente a la almacenada en una base de datos. Pese a que muchos rasgos faciales son altamente cambiables con el paso del tiempo, el reconocimiento facial goza de una aceptación muy alta, además de altos niveles de fiabilidad; además, los avances tecnológicos han permitido el desarrollo de técnicas basadas en sensores de reconocimiento 3D, lo que abre una nueva dimensión para este sistema biométrico. *(Resumen, Zorita 2003)*

- *Geometría de la mano:* Este sistema biométrico se basa en la forma geométrica de la mano de una persona. Esta forma se define de manera particular por la estructura ósea y muscular de la misma. Actualmente, éste es uno de los sistemas biométricos más utilizados por varias razones entre las que destacan rapidez y una facilidad de uso relativamente alta.

Básicamente, su funcionamiento consiste en la captura de la dimensión y la forma de la mano y los dedos de la misma. Estos datos, al igual que en los otros sistemas biométricos, son contrastados con una base de datos. *(Resumen, Zorita 2003)*

- *Reconocimiento del iris:* La identificación de personas a partir del patrón del iris ocular ha experimentado un gran auge en los últimos años; esto debido a los excelentes resultados obtenidos y el gran interés mostrado por algunos sectores económicos para incorporar dicha técnica a sus sistemas de identificación. Una de las grandes ventajas de este sistema es la gran estabilidad presentada frente el paso del tiempo, e incluso, accidentes. Otra característica (según muchos, la más importante) de este sistema biométrico es su elevado nivel de unicidad. Según varios estudios, en el patrón del iris, frente a otras características físicas de las que se vale la biometría, existe más información que identifica unívocamente a un individuo. El insumo

para este tipo de sistema biométrico es una imagen del iris capturado desde el exterior a través de la córnea. *(Resumen, Zorita 2003)*

- *Reconocimiento de voz:* Las bases para el reconocimiento de voz data de alrededor del año 1960 como resultado del trabajo de la compañía Texas Instruments. Desde esos tiempos, la identificación a partir de la voz ha estado sujeta a numerosos estudios e investigaciones. Uno de los grandes problemas de este sistema biométrico es que el habla de un ser humano está determinada por elementos como hábitos lingüísticos y entonación, mismos que son altamente influenciados por su entorno y factores psicológicos. Esto conlleva a que la voz sea altamente cambiante en un tiempo relativamente corto. La variabilidad de una señal de voz es tanta, que se estima que un mismo locutor no puede repetir de forma exacta una misma frase o palabra. El sistema toma su decisión acerca de la identidad del locutor, a partir de señales de voz emitidas por este mismo. De estas señales se obtienen valores o patrones claves que son tomados como referencia y, por medio de un método para la obtención

de similitudes, comparados con patrones y valores obtenidos de una base de datos. *(Resumen, Zorita 2003)*

- *Análisis de escritura:* Pese a que en los últimos años, debido al avance tecnológico y las opciones que se presentan para sustituir diversas metodologías tradicionales, el uso de la escritura manual ha decrecido en gran manera; sin embargo, las mismas posibilidades tecnológicas han reavivado el interés por el uso de la escritura y su estudio con fines de identificación. La firma (o palabra clave) es capturada por medio de dispositivos digitales. A diferencia de la comparación de una firma escrita con anticipación, la captura de una firma por medio de dispositivos digitales y a tiempo real, permite tener información valiosa al momento de identificar a una persona; tal es el caso de la velocidad de escritura, la secuencia ordenada de trazos, la presión ejercida, etc. Luego de capturada la firma escrita y extraídos los patrones, éstos son analizados frente a los de la base de datos previamente existente, para identificar a una persona o determinar la autenticidad de la misma. *(Resumen, Zorita 2003)*

2.3 COMPARACIÓN DE SISTEMAS BIOMÉTRICOS

Todos los sistemas biométricos mencionados en el apartado anterior son funcionales, así también, todos poseen ventajas y desventajas, las cuales son un parámetro importante a tomar en cuenta cuando de elegir uno de ellos se trata.

Algunos aspectos fundamentales a tomar en cuenta son los siguientes:

- *Universalidad:* Existe para todos los individuos.
- *Unicidad:* Identifica unívocamente a cada individuo.
- *Permanencia:* Se mantiene invariable a corto plazo.
- *Mensurabilidad*: Es caracterizable cuantitativamente.
- *Rendimiento:* Nivel de rapidez, robustez y precisión.
- *Aceptabilidad:* Aceptado por la mayoría de las personas.
- *Invulnerabilidad:* Seguridad frente a fraudes.

En base a estudio y análisis de los diferentes sistemas biométricos y sus cualidades, a cada técnica se ha establecido valores para cada una de las cualidades mencionadas anteriormente. A continuación se muestra la tabla que recoge dicha información:

	Universalidad	Unicidad	Permanencia	Mensurabilidad	Rendimiento	Aceptabilidad	Invulnerabilidad
Reconocimiento de huella dactilar	M	A	A	M	A	M	M
Reconocimiento facial	A	B	M	A	B	A	A
Geometría de la mano	M	M	M	A	M	M	M
Reconocimiento del iris	A	A	A	M	A	B	B
Reconocimiento de la voz	M	B	B	M	B	A	A
Análisis de escritura	B	B	B	A	B	A	A

Tabla 2.1: Comparación de los sistemas biométricos.
B: Baja; M: Media; A: Alta. *(Zorita, 2003)*

Para seleccionar la alternativa más viable para el caso de la Facultad Multidisciplinaria de Occidente, se estiman las valoraciones para cada uno de los aspectos o criterios de evaluación, según las necesidades de la Institución. El orden de preferencia para cada uno de los aspectos de cada técnica de reconocimiento es el siguiente:

1) *Rendimiento*: El sistema manejará una gran cantidad de usuarios, por esta razón se requiere de un alto rendimiento.

2) *Unicidad*: Debido también a la alta cantidad de usuarios y a la necesidad de identificar inequívocamente a los usuarios, se requiere una alta unicidad.
3) *Universalidad*: El rasgo biométrico debe ser poseído por una cantidad de usuarios que alcance o se acerque a la totalidad.
4) *Aceptabilidad*: En la medida de lo posible, se espera que el sistema pueda ser implementado sin que genere molestias en los usuarios.
5) *Permanencia*: Como mínimo, se esperaría que los grandes grupos de usuarios, como es el caso de los estudiantes, no necesiten actualizar su información biométrica mientras formen parte de la institución.
6) *Invulnerabilidad*: Se requiere un sistema invulnerable, dentro de lo lógico; es decir que el costo de infringimiento supere los beneficios obtenidos de dicha acción.
7) *Mensurabilidad*: Este tipo de medición no es indispensable para el sistema de reconocimiento automatizado, siempre y cuando se conserve una alta tasa de rendimiento.

A estos criterios, cuyo orden de prioridad corresponde al orden en que son listados, han sido asignados los siguientes valores representativos de la misma:

CRITERIO	GRADO DE IMPORTANCIA
Rendimiento	0.20
Unicidad	0.20
Universalidad	0.17
Aceptabilidad	0.15
Permanencia	0.13
Invulnerabilidad	0.10
Mensurabilidad	0.05

Tabla 2.2: Importancia de las características de las técnicas biométricas para el sistema de control de acceso.

A cada uno de los criterios se asigna un valor relativo de conveniencia para cada técnica de reconocimiento biométrico, donde la importancia se multiplica por la descripción cualificativa de cada criterio, evaluado como: Bajo/a = 1; Medio/a = 2; Alto/a = 3.

La matriz de conveniencia correspondiente a la decisión de la técnica biométrica más apropiada para el control de acceso peatonal al campus de la FMO, queda así:

	Universalidad	Unicidad	Permanencia	Mensurabilidad	Rendimiento	Aceptabilidad	Invulnerabilidad	TOTAL
Reconocimiento de huella dactilar	0.34	0.60	0.39	0.10	0.60	0.30	0.20	2.53
Reconocimiento facial	0.51	0.20	0.26	0.20	0.20	0.45	0.30	2.12
Geometría de la mano	0.34	0.40	0.26	0.20	0.40	0.30	0.20	2.10
Reconocimiento del iris	0.51	0.60	0.39	0.10	0.60	0.15	0.10	2.45
Reconocimiento de la voz	0.34	0.20	0.13	0.10	0.20	0.45	0.30	1.72
Análisis de escritura	0.17	0.20	0.13	0.20	0.20	0.45	0.30	1.65

Tabla 2.3: Matriz de criterios sobre las técnicas biométricas y su conveniencia para el sistema de control de acceso.

Como se puede observar, la mayor conveniencia para el control de acceso peatonal del que trata este documento se obtiene haciendo uso del reconocimiento por huella dactilar. Además, existen aspectos técnicos y de implementación que favorecen este resultado. Tal es el caso de la facilidad de uso, el costo de los dispositivos de lectura, etc.

2.4 RECONOCIMIENTO DE HUELLA DACTILAR

El reconocimiento de una huella dactilar automatizado es altamente complejo debido a que la misma posee un bajo nivel de mensurabilidad, esto dado que no existe ningún patrón bien definido en la captura de la misma. Por esta razón requiere de un análisis minucioso de ella y sus características; este análisis es el que se describe a continuación.

Mejora de la calidad de la imagen

Dado que la calidad de la imagen puede verse comprometida por aspectos como la fuerza con que el dedo es colocado en el lector o el estado de este último, lo primero que se debe hacer es procesar la huella buscando un nivel de calidad que garantice un análisis correcto. Esto se lleva a cabo en los pasos mencionados a continuación:

Normalización

Esta etapa lo que persigue es, a través de la variación del rango de grises entre crestas y vallas, regular los valores del rango de luminancia de la imagen.

Para realizar este procedimiento, se obtiene la luminancia media de los pixeles que conforman la imagen de la huella y su varianza respectiva; luego los valores que sobrepasan dicho valor son oscurecidos y al resto se reduce la intensidad. *(Resumen, Zorita 2003)*

A continuación se muestra la imagen capturada de una huella (lado izquierdo) y la imagen resultante de aplicar el proceso de normalización sobre la primera. Como se puede observar, la procesada posee una mayor definición.

Imagen 2.1: Comparación entre imagen capturada y normalizada *(Zorita, 2003)*

Estimación del campo de orientación

Este paso permite conocer el ángulo de orientación de las crestas de la huella dactilar, dicha estimación se lleva a cabo con bloques de NxN pixeles, donde N suele tener un valor

entre 8 y 16, dependiendo de la resolución de la imagen a procesar y del grado de precisión requerido y del tiempo de respuesta deseado para el sistema.

Habiendo definido las dimensiones utilizada para los cuadros de análisis (el valor de N), se estima el ángulo de orientación local de las crestas para cada uno de dichos cuadros.

En muchos casos, debido al ruido de fondo la imagen captura, se vuelve necesaria una reestimación de del campo de orientación. En todo caso, esto dependerá de la calidad y el rendimiento deseados, así como de la calidad media de las huellas capturadas y el algoritmo específico utilizado. *(Resumen, Zorita 2003)*

Imagen 2.2: Comparación entre imagen capturada y normalizada *(Zorita, 2003)*

En la imagen 2.2, se muestra el resultado de este paso para la huella normalizada en el paso anterior.

Obtención de la región de interés

Dado que la imagen capturada posee ruido de fondo, es necesario seleccionar la región de interés para el análisis, dado que en caso contrario se podrían tomar en cuenta minucias ubicadas fuera del campo ocupado por la huella. Este paso, al igual que el anterior no es implementado por todos los algoritmos de análisis dactilar, pues aunque es de alta importancia, su necesidad también está relacionada con aspectos como el ambiente en que la huella es capturada, el tipo de escáner utilizado, etc.

A groso modo, lo que se pretende es diferenciar claramente entre la zona en que se encuentra la estructura de crestas y la zona en la que se encuentra el ruido de fondo. Una vez establecida esta separación, puede eliminarse la zona de la imagen correspondiente al ruido de fondo o ignorarla por completo para el proceso de análisis de las minucias.

Al igual que para los pasos anteriores, para este existen diversas técnicas; una de las más fiables, naturales y utilizadas es el análisis de la varianza de la intensidad de

grises en la dirección perpendicular a las crestas en cuadros de NxN. El criterio es que los cuadros que formen parte de la captura dactilar presentarán una varianza elevada del nivel de intensidad en la dirección perpendicular a la de la cresta y casi nula en la dirección de la misma; mientras tanto, los cuadros pertenecientes al ruido de fondo, al no tener ningún tipo de dependencia direccional, presentarán una varianza baja en todas las direcciones.

En base a lo anterior, se establece un umbral de decisión; los cuadros cuya varianza supere dicho umbral serán interpretados como parte de la región de interés, mientras que el resto se interpretarán como cuadros correspondientes al ruido de fondo de la captura. *(Resumen, Zorita 2003)*

En la imagen 2.3 se muestra la captura dactilar, una matriz de cuadros en escala de grises que representan la varianza de la que se ha hablado y por último, la separación entre la región de interés y la del ruido de la huella.

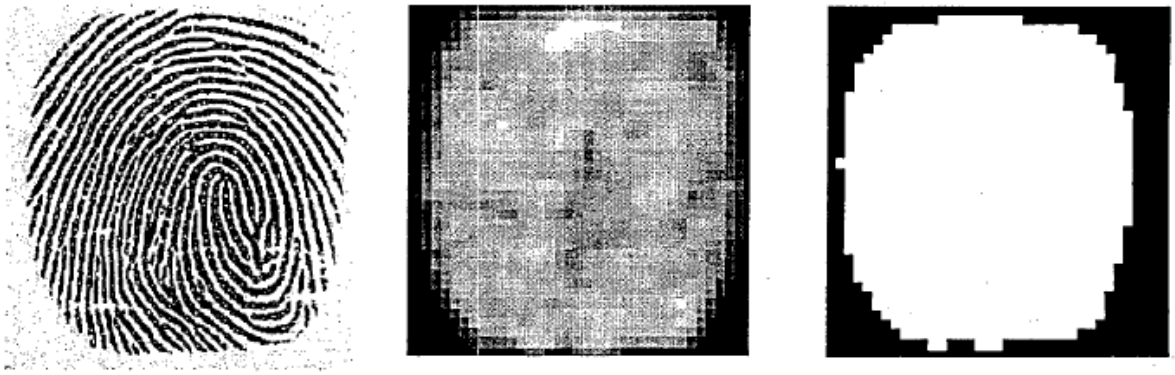


Imagen 2.3: Ilustración del establecimiento de la región de interés de una captura dactilar *(Zorita, 2003)*

Extracción de crestas

La extracción de crestas es fundamental para el análisis de una huella dactilar. Existen diversos algoritmos para tales fines y los mismos se basan en las propiedades estructurales de la imagen resultante de la captura.

El punto de partida para la extracción de crestas es el hecho de que la estructura de crestas y valles de una huella dactilar puede describirse como un flujo de líneas paralelas, donde los máximos y mínimos del nivel del luminancia se distribuyen a lo largo de la dirección normal al ángulo de orientación de cada bloque de la imagen.

Para preparar la imagen para la decisión de si un pixel forma parte de una cresta o un valle se filtra la imagen con cuadros capaces de aumentar el nivel de gris en la dirección

normal a la dirección de la cresta, llamados "máscaras adaptativas". La orientación de dichas máscaras se va adaptando a cada bloque de la imagen, tomando como parámetro el valor del ángulo local extraído del cuadro del campo de orientación correspondiente.

La convolución entre la imagen de la huella y cada máscara da como resultado una huella con las crestas reconstruidas en cierto grado, donde se han acentuado los niveles de grises en la dirección normal a cada una de ellas. Luego de la obtención de esta imagen, aún en escala de grises, se procede a convertirla en una imagen binarizada, es decir, con pixeles blancos y negros únicamente. Para ello, se establece un determinado umbral de luminancia; si el nivel de gris de un pixel sobrepasa dicho umbral, se considera como parte de una cresta, en caso contrario se interpretará como parte de un valle.

Otra de las ventajas de este procesamiento, como se puede observar en la siguiente imagen comparativa, es el hecho de que desaparecen casi en su totalidad los poros presentes en la captura. La imagen muestra la comparación entre la captura original y la huella binarizada. *(Resumen, Zorita 2003)*

Imagen 2.4: Demostración del resultado de extracción de crestas *(Zorita 2003)*

Perfilado de crestas

Luego de realizada la extracción de crestas, se obtiene la imagen binarizada de la huella dactilar. En miras al análisis de la huella, la imagen binarizada posee una calidad mucho mayor a la capturada, sin embargo esta calidad aún puede ser mejorada con el fin de simplificar el proceso de las siguientes etapas de análisis.

Lo que se persigue con este proceso es perfilar las crestas de la huella, realzando los bordes de las mismas; logrando también eliminar imperfecciones que puedan generar minucias espurias y hacer menos preciso el análisis. *(R., Zorita 2003)*

En la imagen 2.5 se puede apreciar una comparación entre la anterior huella binarizada y la resultante del proceso de perfilado de crestas; si se observa cuidadosamente se notará que la imagen de la derecha (imagen cuyas crestas han sido perfiladas) posee una mayor definición que la de la izquierda (primer imagen binarizada), y que en la imagen perfilada han desaparecido buena cantidad de las manchas de fondo presentes en la otra.

Imagen 2.5: Ilustración de huella binarizada con crestas perfiladas *(Zorita 2003)*

Con la realización de este paso y los anteriores, se logra obtener una huella dactilar lista para ejecutar sobre ella el proceso de análisis de su estructura y la extracción de su patrón biométrico, asunto abordado en el apartado siguiente.

Extracción del patrón biométrico.

La etapa de extracción del patrón biométrico es la encargada de obtener las características de la huella que permiten identificar de manera única al individuo al que pertenece. Este patrón es formado a partir de las minucias de la huella, entre las cuales se pueden encontrar:

a) *Bifurcación larga*: Dos crestas que se unen en una sola, o una cresta que se bifurca en dos, tratándose en ambos casos de crestas largas y con tendencia paralela.
b) *Final largo*: Final abrupto de una cresta larga.
c) *Isleta*: Dos bifurcaciones próximas compartiendo las mismas crestas.
d) *Puente*: Unión de dos crestas paralelas, mediante una tercera cresta con tendencia perpendicular.
e) *Cresta corta*: Cresta con principio y final muy próximos, generalmente situadas entre dos crestas largas.
f) *Cresta espuria*: Bifurcación de una cresta en una larga y una de final próximo.
g) *Pareja de bifurcaciones*: Dos crestas que confluyen en una sola, y a proximidad vuelven a bifurcarse en otras dos crestas.

h) *Minucia espuria*: Corresponde a todo aquel punto característico considerado como minucia ficticia.

Algunos algoritmos de análisis biométrico de huella dactilar no distinguen entre todos los tipos de minucias descritos a continuación, tanto por cuestiones de rendimiento, como por el hecho de que algunos tipos poco evidentes podrían aparecer únicamente como resultado de imperfecciones de adquisición, tal es el caso de las minucias tipo puente, cresta corta y cresta espuria; en estos casos, la minucia es interpretada como minucia espuria. *(Resumen, Zorita 2003)*

La imagen 2.6 muestra una ilustración de los tipos de minucia descritos con anterioridad:

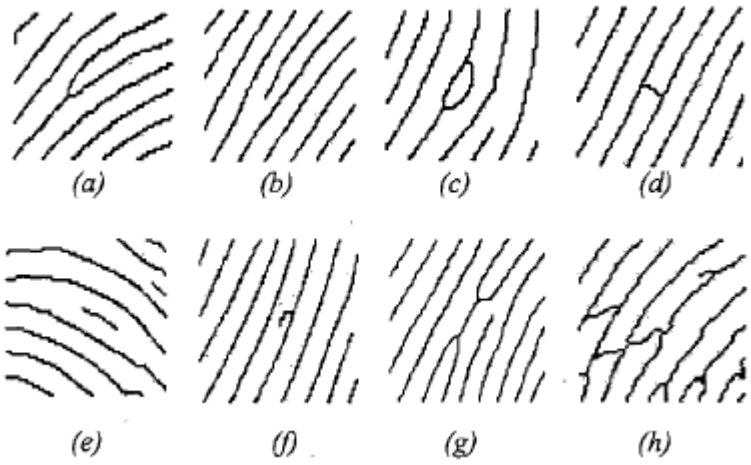


Imagen 2.6: Ilustración de minucias *(Zorita 2003)*

Para la obtención del patrón biométrico de una huella dactilar deben seguirse los pasos descritos a continuación:

Adelgazamiento de la imagen

Luego del proceso de mejora de la calidad de la imagen se obtiene una imagen visualmente clara, sin embargo, para fines de análisis automatizado se requiere una imagen clara para la ejecución del mismo. Esto se logra adelgazando la misma, de tal forma que las crestas se definan a un pixel de grosor. De esta manera los trazos de la huella se convierten en una fina línea que se aproxima a la línea central del trazo original, esto dado que generalmente se seleccionan los puntos que son equidistantes a los bordes de la región original.

La estrategia comúnmente utilizada para el proceso de adelgazamiento es la reducción del grosor en la imagen original, por capas de un pixel de grosor y empezando por el borde de cada trazo. Esto se logra a través de una operación de borrado condicional, que se realiza explorando la imagen pixel por pixel reiteradamente.

La decisión de si un pixel será borrado o seguirá formando parte de la imagen se toma analizándolo en torno a sus pixeles vecinos, por medio de una matriz de análisis de tres por tres. De esta manera, se busca eliminar todo aquel pixel cuya existencia no es necesaria para conservar la conectividad de la estructura, y que tampoco corresponde a un final de línea.

Nombrando los pixeles contendidos en la matriz como P_1, y el resto desde P_2 hasta P_9 en sentido anti horario a partir del pixel ubicado sobre P_1, se obtendría una matriz cuyos puntos podemos describir tal y como se muestra en la siguiente imagen:

P_3	P_2	P_9
P_4	P_1	P_8
P_5	P_6	P_7

Identificando un pixel blanco como con valor cero, y uno negro con valor uno; y llamando $NT(P_1)$ al número de transiciones entre blanco y negro en torno a P_1, y $NU(P_1)$ al número de pixeles vecinos iguales a uno P_1, se toma la decisión de borrar P_1 si se cumplen las siguientes condiciones:

$$6 \geq NU(P_1) \geq 2$$

y

$$NT(P_1) = P_2 + P_3 + P_4 + \ldots + P_9 = 1$$

y

$$P_2 \cdot P_4 \cdot P_8 = 0 \quad \text{ó} \quad NT(P_2) \neq 1$$

y

$$P_2 \cdot P_4 \cdot P_6 = 0 \quad \text{ó} \quad NT(P_4) \neq 1$$

Cada iteración se complementa repitiendo la exploración, pero sustituyendo las dos últimas condiciones por las siguientes:

$$P_2 \cdot P_6 \cdot P_8 = 0 \quad \text{ó} \quad NT(P_8) \neq 1$$

y

$$P_4 \cdot P_6 \cdot P_8 = 0 \quad \text{ó} \quad NT(P_6) \neq 1$$

El proceso completo de ambas exploraciones de la imagen se repite reiteradamente, hasta que ya no se produzcan cambios en la imagen resultante. Como resultado se tendrá una imagen donde las crestas de la huella estén trazadas en líneas de un pixel de grosor. *(Resumen, Zorita 2003)*

Eliminación de imperfecciones

El proceso de adelgazamiento no es suficiente para la preparación de la huella previa a ser sometida al análisis para la extracción de su patrón biométrico, puesto que la mayoría de imperfecciones resultantes de la adquisición de la huella quedan claramente reflejadas a la finalización de dicho proceso.

En el supuesto de que la imagen sea procesada para la obtención del patrón biométrico tal y como resulta del proceso de adelgazamiento, se tendría como consecuencia la disminución del rendimiento computacional para dicho proceso, así como la disminución de la fiabilidad del patrón biométrico extraído, dado que muchas de las minucias auténticas podrían ser eliminadas y muchas minucias espurias ser tomadas en cuenta como parte del patrón resultante.

Este proceso suele llevarse a cabo realizando, como primer paso, un análisis para la búsqueda de crestas espurias; luego, las crestas espurias encontradas son eliminadas y finalmente, se realiza la conexión entre crestas.

Los algoritmos son variados, y en todo caso, dependerán del balance entre rendimiento y fiabilidad requeridos, así como

del ambiente en que la huella es capturada y procesada. No obstante el algoritmo utilizado, a este punto ya se poseerá un huella bien definida y lista para ser procesada en busca de los puntos característicos o minucias. *(Resumen, Zorita 2003)*

Extracción de minucias

Esta es la etapa final para la generación del patrón biométrico de la huella dactilar. El análisis se ejecuta sobre la imagen resultante de la reconstrucción, el adelgazamiento y la eliminación de imperfecciones a partir de la imagen capturada.

Al igual que los pasos vistos anteriormente, para el proceso de extracción de minucias existen muchos algoritmos lógicos, así como diferentes APIs que los implementan. Dicho sea de paso, éstos también varían en cuanto a la información que es almacenada para cada huella.

Un procedimiento muy utilizado consiste en el análisis a partir de un cuadro de tres por tres, igual al del paso de adelgazamiento. En este caso se toma la consideración de que un pixel perteneciente a la cresta debe tener un valor de

uno, es decir, debe ser negro; en caso contrario, se entiende que el pixel no forma parte de una cresta de la huella.

A partir del criterio anterior y tomando P_i como un pixel iterante en que i toma los valores continuos del 2 al 9, se formula el siguiente análisis matemático:

- Un pixel pertenece a una cresta si tiene dos pixeles vecinos en negro, dentro de la ventana de análisis, es decir, cumple con que:

$$\sum_{i=2}^{9} P_i = 2$$

- Un pixel perteneciente a una cresta, corresponde al final de la misma si tiene únicamente un pixel vecino en negro, es decir:

$$\sum_{i=2}^{9} P_i = 1$$

- Un pixel perteneciente a una cresta, corresponde a una bifurcación si tiene tres pixeles vecinos en negro dentro de la ventana de análisis, es decir, cumple con que:

$$\sum_{i=2}^{9} P_i = 2$$

Los puntos encontrados son los que conforman el patrón biométrico, cada uno con su respectiva información; misma que, como se ha expresado, puede variar de un algoritmo a otro, pero generalmente incluye los siguientes aspectos:

- Coordenadas *x* e *y* del pixel que constituye la minucia.
- Ángulo de orientación local de la cresta a la que pertenece dicho pixel.
- Información adicional referente a la minucia según su tipo, es decir:
  - Cuando se trate de un final de cresta, coordenadas *x* e *y* del segmento muestreado de la cresta a la que pertenece dicho pixel.
  - Cuando se trate de una bifurcación de cresta, coordenadas *x* e *y* del segmento muestreado de una de las ramas de dicha bifurcación.

El arreglo en que se almacena esta información, además de alguna otra información de control dependiente de cada módulo de software, es el que se conoce como el patrón biométrico de la huella dactilar. *(Resumen, Zorita 2003)*

Reconocimiento de una huella dactilar

Anteriormente se ha tocado el tema sobre los inconvenientes en la calidad de la imagen capturada y de cómo estos deben ser resueltos, siguiendo una serie de pasos que a su vez se pueden realizar implementando diferentes algoritmos. Sin embargo, hasta la obtención del patrón biométrico, solamente se ha logrado extraer información altamente certera respecto de una huella dactilar, lo cual es un insumo para lo que en realidad interesa: poder reconocer la huella, a partir de sus patrones biométricos.

Es de hacer notar que el patrón biométrico posee probabilidades casi nulas de ser obtenido de manera idéntica en dos lecturas diferentes, aun cuando se tratase de la misma huella. La razón es que dicho patrón se ve afectado por aspectos originados en el ambiente o el usuario mismo, tal es el caso del estado de limpieza del lector, la posición en que el dedo sea colocado, la presión ejercida con el dedo sobre el lector, etc.

El problema básico es si los patrones biométricos de dos huellas extraídas en dos momentos diferentes, y probablemente en ambientes diferentes, corresponden a una misma huella

dactilar. Esta correspondencia existirá cuando el grado de similitud entre los dos patrones comparados supere un umbral preestablecido en base a diferentes aspectos, entre los que se destaca el balance entre la seguridad deseada y la tasa de aceptación mínima requerida.

En la etapa de reconocimiento de una huella a partir de su patrón biométrico, se distinguen dos pasos generales y primordiales. Los mismos se describen a continuación:

Alineación de patrones biométricos

Como ya se ha hecho mención, algo que se debe tomar muy en cuenta es el hecho de que una persona no siempre colocará su dedo exactamente de la misma manera sobre el lector, sea del tipo q sea; además, es muy probable que durante el proceso de preparación de la huella para el análisis, cambie ligeramente la cantidad de minucias detectadas, su posición, sus ángulos respectivos, entre otras cosas. Por la razón anterior se vuelve sumamente necesario alinear los patrones biométricos de ambas huellas.

Existe una solución teóricamente viable frente al problema de la no alineación de las huellas a comparar, y es la

comparación una a una de las minucias, utilizando una caja de tolerancia alrededor de ellas; en donde si la minucia correspondiente es encontrada dentro del rango de tolerancia, será tomada como legítima. En la práctica, sin embargo, la posición de las huellas puede ser tan variada que aún utilizando una caja de tolerancia no se establecería correspondencia entre dos minucias realmente equivalentes.

Una solución viable y fidedigna es la alineación de las crestas de la huella de prueba con las del patrón base (huella previamente almacenada que se va a capturar). De esta manera, se buscarán dos minucias que tengan la misma cresta asociada, llamadas "minucias de referencia". A partir de aquí, se deben calcular los parámetros de traslación y rotación entre dichas minucias, para posteriormente ser aplicados a las demás minucias del patrón prueba, de tal forma que quede alineado como el patrón base.

Para comprobar el resultado del paso anterior, se realiza una comparación de los dos patrones alineados, verificando si las coordenadas cartesianas y los ángulos de orientación de las minucias son coincidentes, dentro de un margen de tolerancia previamente definido. Si el número de minucias coincidentes supera el umbral previamente fijado, se valida el

alineamiento fijado; en caso contrario, se procede a la búsqueda de un nuevo par de minucias candidatas para ser referencia en el proceso de alineación, para repetir con ellas el proceso previamente descrito. Así sucesivamente, hasta que el número de correspondencias entre el patrón prueba y el patrón base supere el umbral, o bien que se supere la cantidad máxima de pruebas deseadas, misma que suele variar entre uno y tres.

Comparación de patrones biométricos

Ya se ha expuesto que las distancias relativas entre minucias, así como su aparición o no en el patrón biométrico, pueden verse comprometidas por diversos factores. Pese a ello, el algoritmo de reconocimiento debe dar un resultado certero sobre la coincidencia o no del patrón de prueba con respecto al patrón base.

Existen diversos algoritmos para tratar el problema de la comparación de cadenas, en su mayoría implementadas sobre símbolos o caracteres, pero aplicables para la comparación de las minucias que conforman el patrón biométrico. En este caso, altamente utilizado y a la vez muy práctico, el grado

de similitud entre los dos patrones, una vez finalizada la comparación entre ellos, vendrá dado por la distancia de edición mínima obtenida al establecer la correspondencia entre las minucias de las dos cadenas. La función de coste debe ser tolerante, para permitir un cierto margen de imprecisión.

El objetivo de la comparación de patrones basado en distancia de edición, también llamada "Distancia de Levenshtein", es "Determinar la serie de transformaciones de coste mínimo necesarias para convertir la cadena de minucias del patrón de prueba, o patrón a reconocer, A, en la cadena de minucias del patrón de la base de datos, B".

Para este proceso se dispone de tres operaciones de edición o transformaciones elementales, a saber:

1) *Sustitución*: La minucia *i*-ésima del patrón de prueba tiene correspondencia con la minucia *j*-ésima del patrón base. En términos de la distancia de edición, la minucia *i*-ésima del patrón de prueba es sustituida por la minucia *j*-ésima del patrón base para transformar una cadena en la otra.

2) *Destrucción*: La minucia *i*-ésima del patrón de prueba no tiene correspondencia con ninguna minucia del patrón base. En términos de la distancia de edición, la minucia *i*-ésima del patrón prueba es eliminada para transformar la cadena en la otra.
3) *Inserción*: La minucia *j*-ésima del patrón base no tiene correspondencia con ninguna minucia del patrón prueba. En términos de la distancia de edición, al patrón prueba le es añadida la minucia *j*-ésima del patrón base para transformar una cadena en la otra.

El algoritmo de conversión de una cadena en otra asocia un coste determinado a cada una de las transformaciones realizadas durante el proceso de comparación. Este coste será tanto mayor cuanto más difieran entre sí las minucias comparadas. Esto significa que para dos imágenes dactilares determinadas, el coste de transformación entre sus patrones será menor ante una mayor similitud y viceversa. *(Resumen, Zorita 2003)*

Gráficamente, los costos asignados para cada operación se representan en la imagen 2.7.

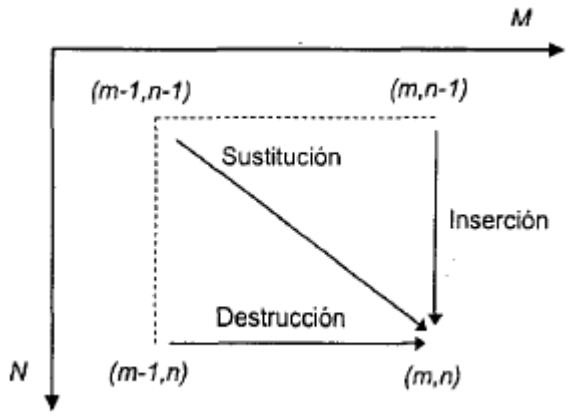


Imagen 2.7: Representación de costos de transformación *(Zorita 2003)*

A su vez, su formulación se especifica así:

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + \gamma(i,j) \\ D(i-1,j) + \gamma(i) \\ D(i,j-1) + \gamma(j) \end{cases}$$

Donde $D(i,j)$ representa el costo de hacer coincidir la minucia i del patrón de prueba con la minucia j del patrón base. Así, $\gamma(i)$ y $\gamma(j)$ representan los costos de destrucción y sustitución, respectivamente; mientras que $\gamma(i,j)$ simboliza el costo de sustitución. En el caso de los costos de destrucción e inserción, se pueden interpretar como 1; mientras tanto, el costo de sustitución será de 0 si la minucia es correspondiente y de 1 en caso contrario.

En la imagen 2.8 se presenta un ejemplo demostrativo con las cadenas de texto: "HOLA" como cadena base, y "HOLEA" como cadena de prueba.

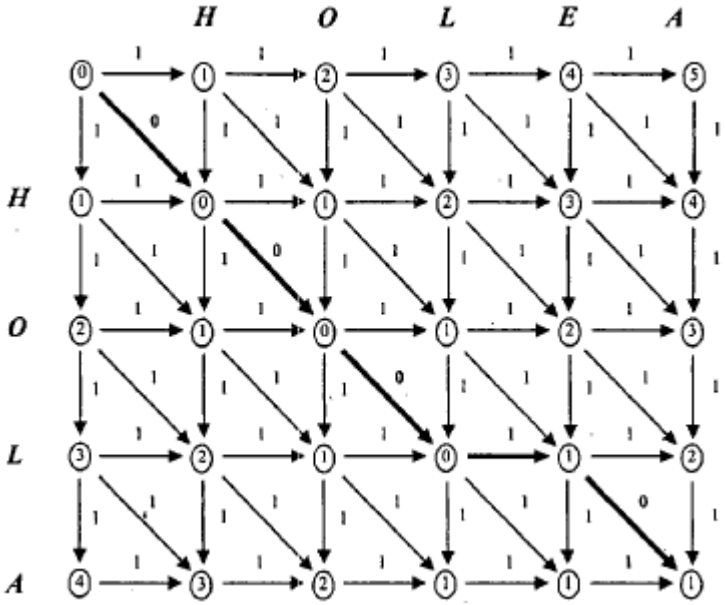


Imagen 2.8: Matriz de cálculo de costos de transformación *(Zorita 2003)*

En este ejemplo, para convertir la cadena de prueba en la cadena base se obtiene un coste mínimo de $D(5,4) = 1$.

A continuación se muestra una imagen que ilustra cómo, a partir del alineamiento de la captura de prueba con respecto a la captura base, se establece correspondencia entre sus dos patrones de minucias, dentro de un margen de tolerancia dado.

Imagen 2.9: Ilustración de la comparación de patrones de dos huellas alineadas *(Zorita 2003)*

En esta operación, se validará el reconocimiento si y solamente si el coste de coincidencia no supera el máximo establecido en la implementación del algoritmo, mismo que debe establecerse de manera experimental. *(Resumen, Zorita 2003)*

Este proceso de verificación de patrones biométricos, al ser ejecutado sobre toda una colección de huellas, permite identificar a una persona. Si la huella del sujeto es encontrada, se termina la búsqueda, entendiendo que se trata de un usuario registrado; mientras tanto, si al haber comparado con todas las huellas contenidas en la colección con la huella de prueba, se entenderá que no se trata de un usuario registrado.

CAPÍTULO III:

DISEÑO DEL PROYECTO

3.1 DESCRIPCIÓN DE LOS ELEMENTOS DEL SISTEMA

El sistema de control de acceso correspondiente a este proyecto y nombrado "UAccess" cuenta con diferentes elementos que le permiten ser completo, centralizado, administrable y objetivo. A continuación se describe brevemente cada uno de dichos elementos:

- *Base de datos:* La base de datos del sistema está concebida para ser una base de datos centralizada, a la cual se conectarán los equipos de los diferentes puntos de acceso peatonal, así como los que ejecuten el software administrativo para el sistema de control de acceso.

  La misma está hecha en MySQL, cuenta con 10 tablas y utiliza el motor de datos InnoDB.

- *Programa del punto de acceso:* El programa computacional del punto de acceso peatonal es el que se encarga de identificar a las personas y permitirles o denegarles el acceso al campus o una zona determinada. El mismo se conecta a la base de datos para consultar los usuarios

del sistema, su huella, sus permisos de acceso, su status, y en fin, toda la información que sea requerida para su funcionamiento. Cuando un usuario verificado es válido, envía la orden al torniquete de acceso para ser desbloqueado y permitir el paso del usuario. Se encarga además de insertar la información correspondiente en la bitácora de acceso cada vez que una persona es identificada correctamente y por ende, aceptada.

Este programa está desarrollado en C++ haciendo uso de las librerías de Qt para la interfaz de usuario, así como las librerías nativas libfprint y libusb, para el reconocimiento biométrico y la comunicación con el torniquete de acceso, respectivamente.

- *Sistema administrativo*: Es la aplicación de escritorio que contiene los módulos de software destinados para la administración del sistema desde una interfaz gráfica sencilla, funcional e intuitiva y sin requerir de altos conocimientos en informática.

  Esta aplicación permite administrar el sistema de acceso por completo, así como obtener información del mismo a través de los informes generados por éste.

Este programa está desarrollado en java; entre otras cosas, utiliza swing para las interfaces de usuario y trabaja sobre la base de datos centralizada del sistema de control de acceso por medio del API de persistencia JPA. Utiliza además JNA (Acceso Nativo de Java, por sus siglas en inglés) para las tareas de naturaleza biométrica, tal es el caso de la captura de la huella dactilar del usuario. Por seguridad, el sistema no almacena la imagen de la captura dactilar, sino solamente los datos principales de esta, requeridos para la identificación de los usuarios.

- *Dispositivo de acceso*: El dispositivo hardware que complementa al software de control de acceso está dividido en dos partes: El circuito de comunicación y control, y el torniquete mecánico de acceso. Esta segmentación permite flexibilidad en cuanto al dispositivo (torniquete) de acceso seleccionado.

  El hardware controlador contiene como elemento principal un PIC18F4550 de Microchip, el cual ha sido programado para comunicarse con el computador sobre el que corre el programa del punto de acceso a través del puerto USB.

Por su parte, el torniquete de acceso funciona a la orden recibida del hardware controlador. El mismo puede ser sustituido por otro tipo de dispositivo mecánico siempre y cuando permita el paso de una persona única y exclusivamente al recibir la orden del hardware controlador. *(Diseño propio)*

3.2 DISEÑO DE LA BASE DE DATOS

La base de datos del sistema de control de acceso peatonal posee diez tablas interrelacionadas tal y como se muestra en el Diagrama 3.1.

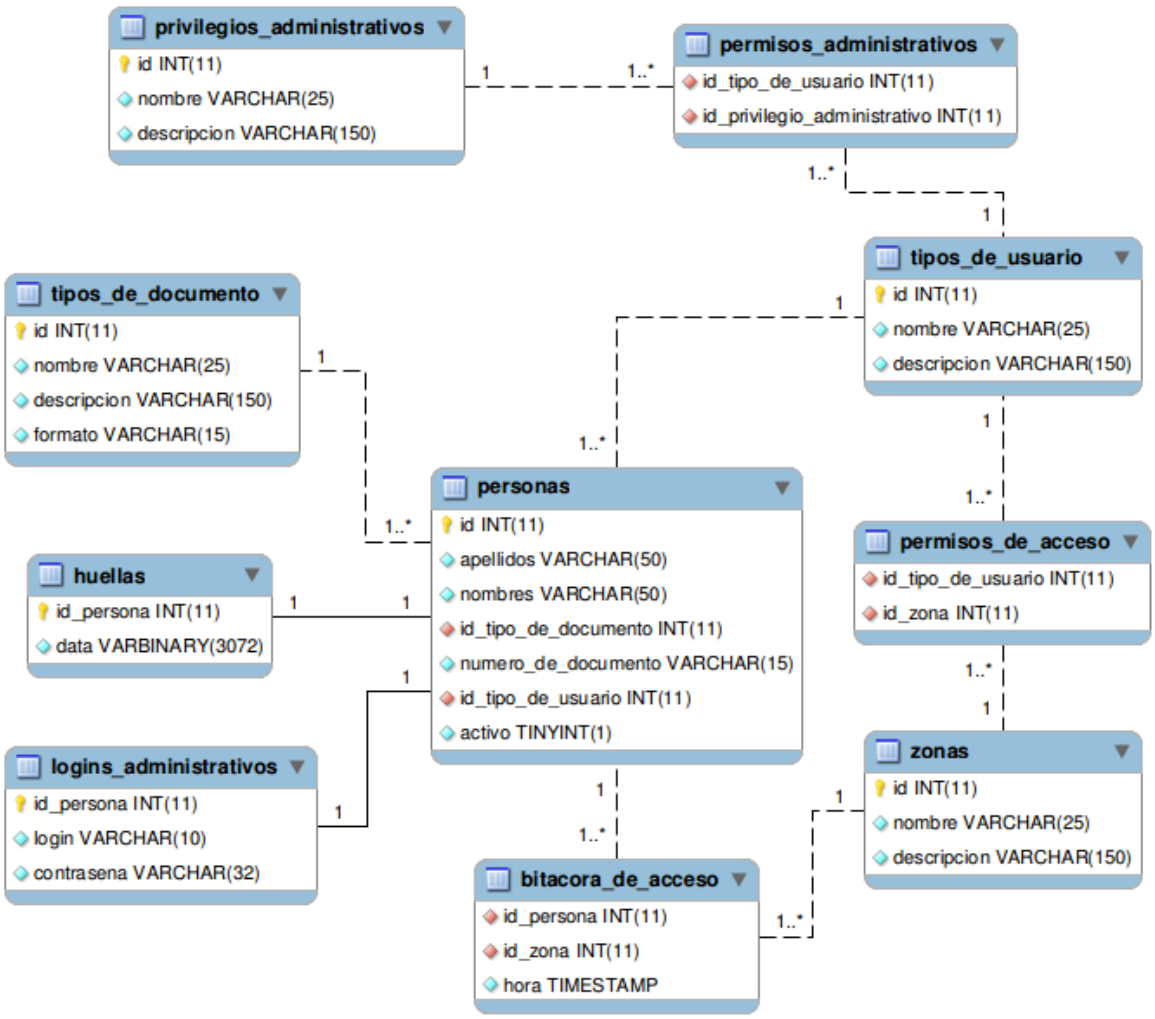


Diagrama 3.1: Diseño de la base de datos UAccess *(Diseño propio)*

Para ver el resto de la documentación revise el ANEXO 1.

3.3 DISEÑO DEL SISTEMA ADMINISTRATIVO

La aplicación administrativa se divide en diferentes módulos, representados cada uno por un paquete dentro del proyecto. Los mismos se describen a continuación *(Diseño propio)*:

- *Persistencia*: Se encarga del acceso a datos y la persistencia de los mismos para la aplicación.
- *Fprint*: Gestiona lo referente a las necesidades biométricas de la aplicación administrativa.
- *Utilidades*: Posee diferentes utilidades requeridas para el funcionamiento del sistema; por ejemplo, cargar reportes y encriptar contraseñas. A la vez posee los siguientes sub-módulos:
  - *Controles*: Posee controles personalizados utilizados en la interfaz gráfica de la aplicación.
  - *Validación*: Contiene utilidades para la validación de datos previo a su almacenamiento en la base de datos, y de usuarios previo a la ejecución de acciones por parte de éste.
- *GUI*: Módulo contenedor de los elementos de interfaz de usuario de la aplicación.

La jerarquía de paquetes y clases se encuentra descrita de manera gráfica en el Diagrama 3.2.

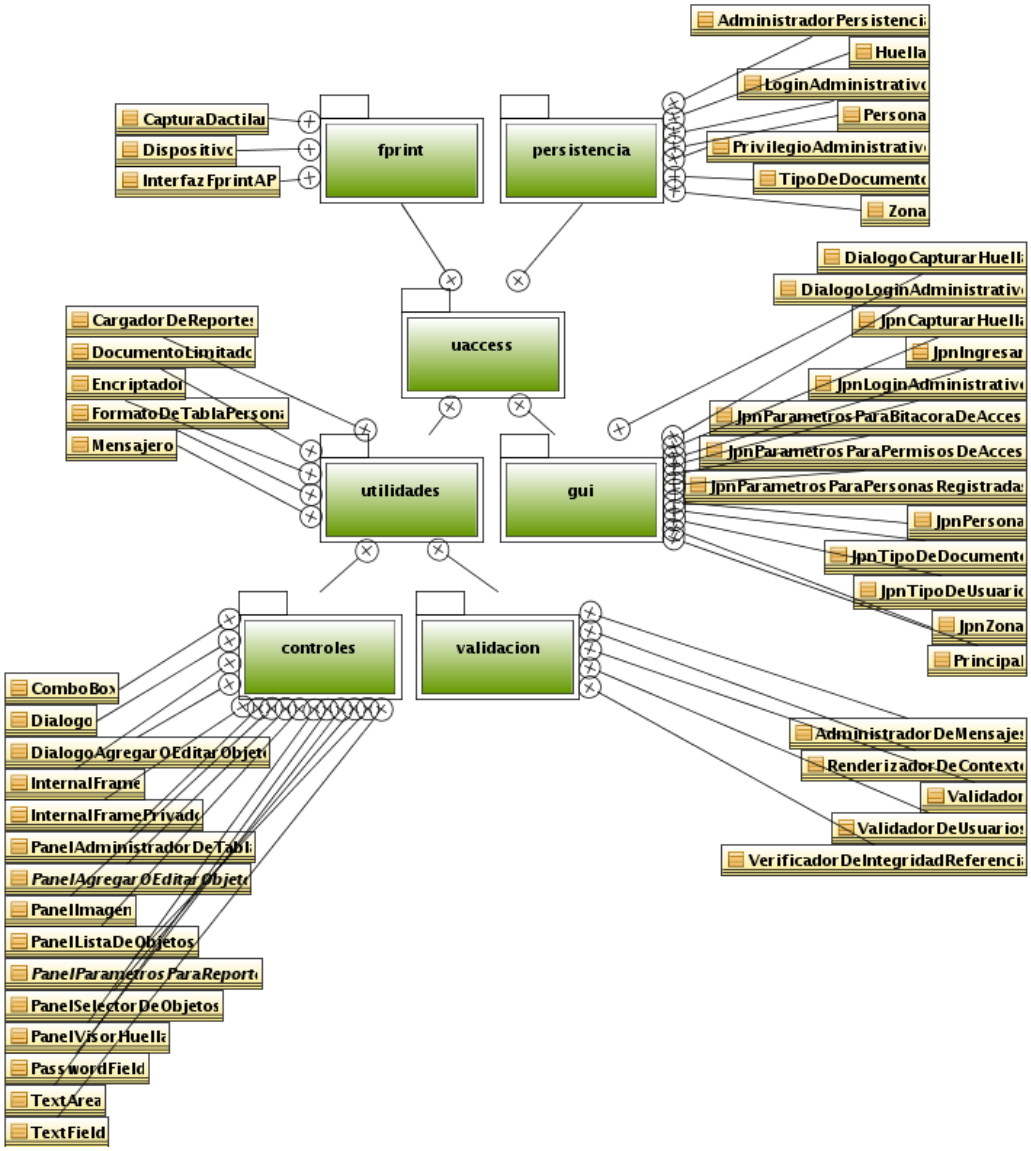


Diagrama 3.2: Clases del sistema administrativo *(Diseño propio)*

Para ver la documentación completa de cada una de las clases, revise el ANEXO 2.

3.4 DISEÑO DEL SISTEMA DEL PUNTO DE ACCESO

El programa del punto de acceso peatonal está diseñado para permitir únicamente el ingreso de usuarios identificados como válidos a partir de los patrones biométricos de su huella dactilar. Esta identificación se realiza en base a una lista de usuarios válidos que cambia constantemente para tenerla actualizada y no pasar por alto algún cambio como la desactivación de un usuario, entre otros probables.

Al momento de identificar a un usuario como válido, una señal es enviada al dispositivo de acceso para que el mismo se desbloquee, permitiéndole así el paso. Simultáneamente, el ingreso del usuario es registrado en la bitácora de acceso con la especificación del mismo, la zona a la que ingresó y el momento en que la acción se llevó a cabo.

Algo importante de destacar es el hecho de que las tareas de espera, captura y análisis de la huella, así como la comunicación con el torniquete de acceso, son gestionadas por un proceso separado al de interfaz gráfica. La comunicación entre dicho proceso y la interfaz de usuario se realiza por medio de señales desencadenadas por el primero e interpretadas y procesadas por el segundo. Dicho proceso es

pausado cuando el punto de acceso peatonal está siendo configurado y detenido cuando se solicita el cierre de la aplicación. *(Diseño propio)*

Desde la perspectiva de diseño, el funcionamiento anteriormente descrito está enmarcado por seis clases interrelacionadas de la manera en que se describen en el Diagrama 3.3.

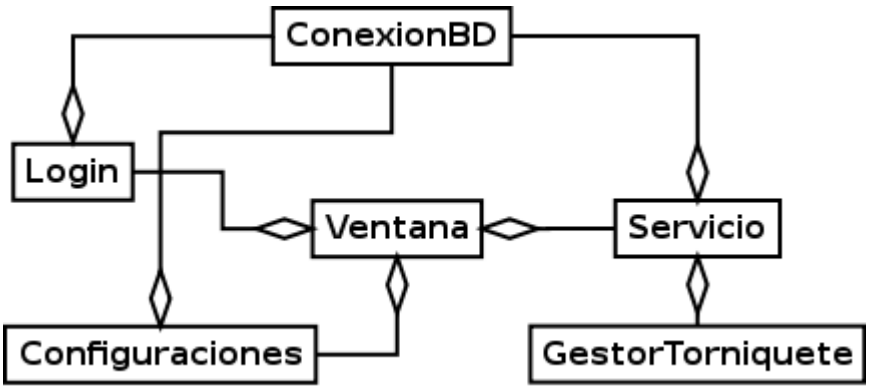


Diagrama 3.3: Clases del programa del punto de acceso peatonal *(Diseño propio)*

La documentación completa de las clases del punto de acceso peatonal y de sus respectivas variables y funciones miembros, puede ser revisada en el ANEXO 3.

3.5 DISEÑO DEL DISPOSITIVO DE ACCESO

Como ya se ha hecho mención, el dispositivo de acceso está dividido en dos partes fundamentales: El circuito de control al cual se hace referencia comúnmente como "dispositivo hardware", y el torniquete de acceso.

El dispositivo hardware es el circuito que establece un vínculo entre el programa del punto de acceso peatonal y el torniquete de acceso. El mismo se conecta por medio de un puerto USB al computador en que corre el programa.

Por otro lado, el torniquete de acceso es el dispositivo mecánico que permite el paso a las personas única y exclusivamente cuando es recibida la señal que le indica que un usuario válido desea ingresar.

En cuanto al diseño del dispositivo hardware, el mismo ha sido concebido para activar la salida al recibir la orden respectiva desde el puerto USB y desactivarla de manera similar. Para lograr esta funcionalidad, se hace uso de un microcontrolador PIC18F4550 programado para tales fines. La salida del mismo se conecta, a través de un optoacoplador y un transistor, a un relay que permite dar una salida final de

110VAC, con el cual alimentar al torniquete de acceso para su funcionamiento. *(Diseño propio)*

El diseño del circuito se presenta en el Diagrama 3.4. Por su parte, lo referente al programa del microcontrolador puede revisarse en el apartado 4.4.

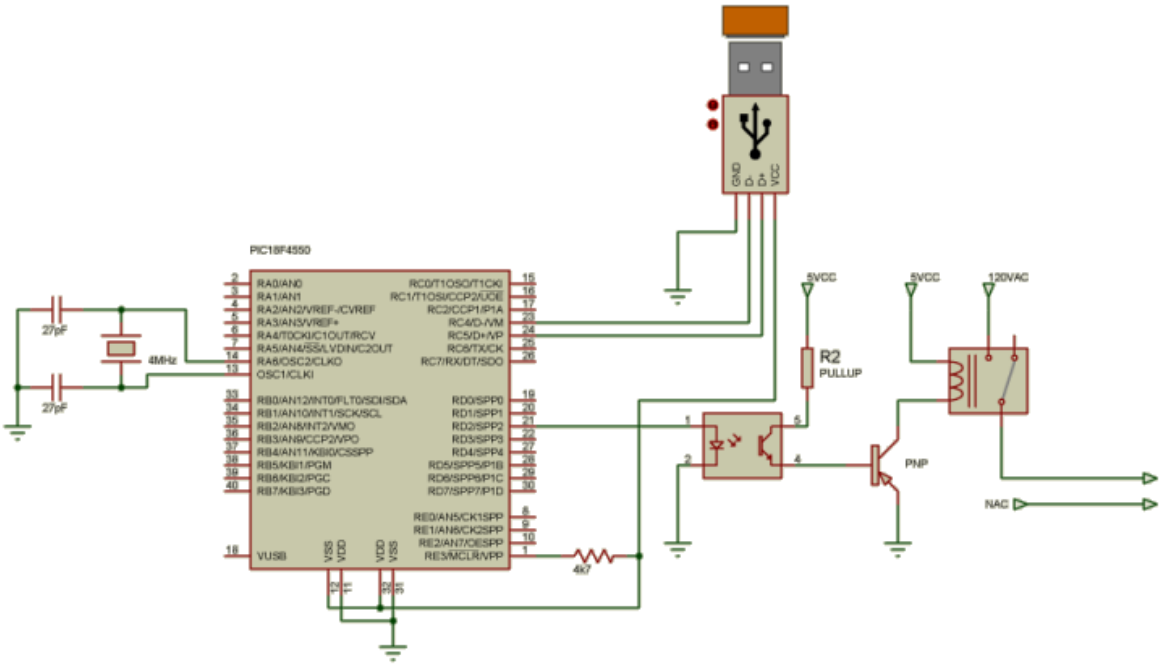


Diagrama 3.4: Diseño del circuito de control del torniquete de acceso *(Diseño propio)*

En cuanto al diseño del dispositivo mecánico, el sistema es adaptable a cualquier dispositivo que permita el acceso de las personas únicamente cuando la señal sea recibida.

Desde este punto de vista, se puede utilizar el tipo de torniquete de acceso que mejor se adecúe a las expectativas en base a seguridad, usabilidad, costos, espacio requerido, etc. *(Diseño propio)*

No obstante lo anterior, como parte del proyecto se ha realizado un prototipo de dispositivo mecánico que es desbloqueado para permitir el acceso únicamente a una persona cuando la señal es recibida.

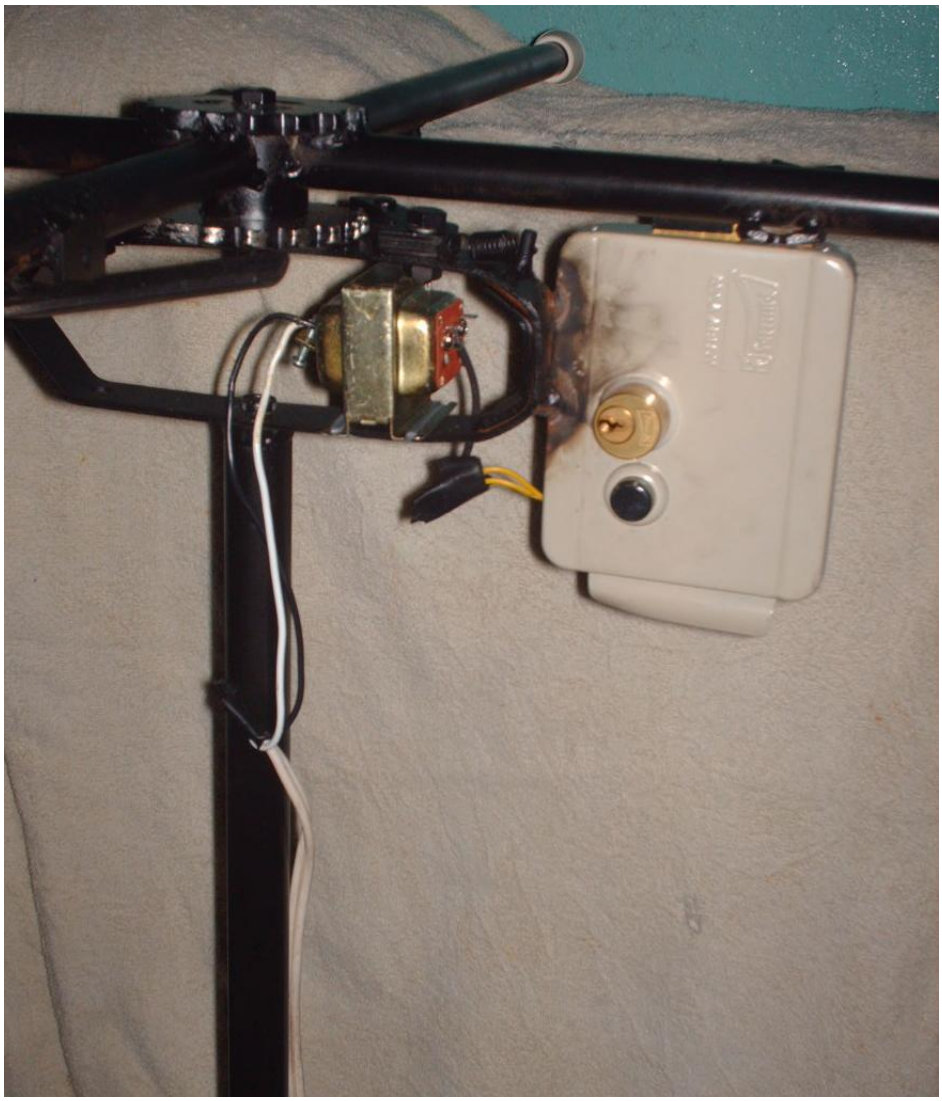
Dicho dispositivo es un torniquete de acceso de cuatro brazos horizontales que giran en torno a una rueda con cavidades de similitud a las de un engranaje, pero elaboradas para, en conjunto a dientes con movimiento regido por la tensión de resortes, girar en una sola dirección *(Diseño propio)*. Puede ver dicha pieza en la Fotografía 3.1.

Fotografía 3.1: Núcleo del torniquete de acceso.

El torniquete posee además una chapa eléctrica, que es desbloqueada al recibir la señal proveniente del dispositivo hardware. Esta chapa vuelve a bloquearse dese que el dispositivo comienza a girar, de esta manera luego de haber girado los 90° necesarios para el ingreso de una persona, el torniquete quedará nuevamente bloqueado al contacto de la chapa con piezas metálicas paralelas al brazo que marca el fin del giro*(Diseño propio)*. La Fotografía 3.2 muestra la manera en que el torniquete es bloqueado y desbloqueado.

Para una óptica completa del prototipo para el torniquete de acceso, puede ver la Fotografía 3.3.

Fotografía 3.2: Sistema de bloqueo y desbloqueo



Fotografía 3.3: Vista del prototipo del torniquete de acceso.

CAPÍTULO IV:

DESARROLLO DEL SISTEMA

4.1 ASPECTOS GENERALES

En el capítulo anterior se tocaron los aspectos de diseño del sistema y sus elementos o subsistemas. Este capítulo, por su parte, está dedicado al desarrollo de dichos elementos.

Concretamente, este capítulo aborda aspectos como la plataforma utilizada, lenguajes, IDES, librerías, etc., así como las versiones respectivas y otros aspectos técnicos.

Más adelante se abordan los aspectos propios de cada subsistema, sin embargo hay que comenzar con algunos aspectos generales del sistema:

- *Sistema Operativo:* Se ha utilizado Linux, específicamente Debian Squeeze con la versión de kernel 2.6.32-trunk-amd64.
- *Dispositivo de captura dactilar:* El scanner utilizado (aunque el sistema soporta otros fabricantes y modelos) ha sido el DigitalPersona U.are.U 4000B.
- *Libfprint:* Es una de las principales librerías utilizadas en el desarrollo del proyecto y es utilizada tanto por el programa de acceso como por el sistema administrativo. La versión utilizada es la 0.0.6.

- *Libusb:* Esta librería es directamente utilizada por el programa de control de acceso, y al mismo tiempo una dependencia de libfprint, que además se utiliza tanto en el software de acceso como en el administrativo. La versión utilizada es la 2:1.0.8-2.

Habiendo abordado el modelo de datos en el capítulo anterior, y los anteriores aspectos generales, podemos comenzar a examinar lo concerniente al desarrollo de cada uno de los subsistemas que forman parte de UAccess. *(Diseño propio)*

4.2 DESARROLLO DEL SISTEMA ADMINISTRATIVO

El aplicativo de software utilizado para la administración general del sistema ha sido desarrollado en java, utilizando específicamente el JDK versión 1.6.0_21.

Como entorno de desarrollo se ha utilizado NetBeans IDE, versión 6.9.1, complementado con iReport y JasperReports, ambos en su versión 4.1.1, para diseñar y utilizar los reportes.

Para el acceso a datos se ha hecho uso del driver de JDBC para MySQL, versión 5.1.6, y como módulo de persistencia, EclipseLink 2.0.2.

Otras librerías utilizadas son las siguientes:

- Oval 1.70, para validación de objetos.
- Glazedlists 1.8.0, para la manipulación de listas de objetos como tablas de datos a nivel de interfaz de usuario.
- Xstream 1.3.1, para leer archivos de configuración XML.
- JNA 3.3.0, para el uso de la librería nativa libfprint.
- JCalendar 1.3.3, para el uso de controles de selección fecha/hora en la interfaz de usuario.

- Substance 6.1, para el Look And Feel de la aplicación.
- Trident 1.3.0, requerido por el visor de reportes y las animaciones de Substance.

Habiendo abordado estos aspectos generales referente al desarrollo de la aplicación administrativa, podemos pasar al desarrollo de cada una de las clases.

A continuación se presentan cada uno de los paquetes del proyecto y sus clases, con su respectivo código fuente. Tanto los paquetes como las clases incluyen una breve descripción acerca de su naturaleza. De la misma manera, el código fuente se encuentra comentado con el formato estándar de javadoc en cada uno de sus métodos. Los únicos métodos no documentados son aquellos que, siento métodos que sobrescriben los de su clase madre, no requieren más comentario que el que se presenta en los métodos originales. *(Diseño propio)*

Controles Personalizados

El código aquí descrito es el correspondiente a controles personalizados utilizados para la interfaz gráfica de la

aplicación y corresponde a las clases de org.uaccess.utilidades.controles.

Clase ComboBox

Esta clase define un control tipo lista desplegable, con soporte para manejo de una lista de objetos.

```
package org.uaccess.utilidades.controles;

import ca.odell.glazedlists.GlazedLists;
import ca.odell.glazedlists.swing.AutoCompleteSupport;
import java.util.List;

/**
 * Control tipo lista desplegable con soporte para manejo de una lista
 * de objetos
 * @author victor
 */
public class ComboBox extends javax.swing.JComboBox {

    /**
     * Soporte de autocompletado instalado en el control
     */
    private AutoCompleteSupport soporteDeAutocompletado;

    /**
     * Crea un nuevo control ComboBox con soporte para trabajar con una
     * lista de objetos
     */
    public ComboBox() {
        super();
    }

    /**
     * Instala el contenido de la lista desplegable
     * @param lista lista a instalar en el control
     */
    public void instalar(List lista) {
        if (soporteDeAutocompletado != null){
            soporteDeAutocompletado.uninstall();
        }
```

```
        soporteDeAutocompletado = AutoCompleteSupport.install(this,
                GlazedLists.eventListOf(lista.toArray()));
        soporteDeAutocompletado.setStrict(true);
    }
}
```

Clase Dialogo

La clase "Dialogo" corresponde a una ventana tipo diálogo con soporte para contener un panel previamente creado.

```
package org.uaccess.utilidades.controles;

import java.awt.Component;
import java.awt.Frame;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
import javax.swing.GroupLayout;
import javax.swing.JButton;
import javax.swing.JPanel;
import org.uaccess.utilidades.Mensajero;

/**
 * Ventana tipo diálogo con soporte para contener un panel
 * previamente creado
 * @author victor
 */
public class Dialogo extends javax.swing.JDialog implements
        PropertyChangeListener {

    /**
     * Componente padre del diálogo
     */
    private static final Component padre =
            Mensajero.getInstancia().getPadre();

    /**
     * Crea un nuevo diálogo con tamaño por defecto
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
```

```
     */
    public Dialogo(JPanel panel, String titulo) {
        this(panel, titulo, panel.getPreferredSize().width,
                panel.getPreferredSize().height, null);
    }

    /**
     * Crea un nuevo diálogo con tamaño por defecto y los
     * parámetros indicados
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
     * @param botonPorDefecto botón por defecto del formulario
     * @param icono URL del ícono para la ventana
     */
    public Dialogo(JPanel panel, String titulo, JButton botonPorDefecto,
            URL icono) {
        this(panel, titulo, panel.getPreferredSize().width,
                panel.getPreferredSize().height, icono);
        panel.getRootPane().setDefaultButton(botonPorDefecto);
    }

    /**
     * Crea un nuevo diálogo con las dimensiones indicadas
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
     * @param ancho ancho indicado para la construcción del diálogo
     * @param alto altura indicada para la construcción del diálogo
     */
    public Dialogo(JPanel panel, String titulo, int ancho, int alto,
            URL icono) {
        super((Frame)padre, true);
        inicializar(panel, titulo, ancho, alto, icono);
    }

    /**
     * Inicializa el diálogo con las características indicadas
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
     * @param ancho ancho indicado para la construcción del diálogo
     * @param alto altura indicada para la construcción del diálogo
     */
    private void inicializar(JPanel panel, String titulo, int ancho,
            int alto, URL icono) {
        setTitle(titulo);
        if(icono != null){
            try {
                setIconImage(ImageIO.read(icono));
            } catch (IOException ex) {
                Logger.getLogger(Dialogo.class.getName())
                        .log(Level.SEVERE, null, ex);
```

```
            }
        }
        panel.addPropertyChangeListener(this);
        GroupLayout layout = new GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(layout.createParallelGroup(javax.swing
                .GroupLayout.Alignment.LEADING).addComponent(panel,
                javax.swing.GroupLayout.DEFAULT_SIZE, ancho - 10,
                Short.MAX_VALUE));
        layout.setVerticalGroup(layout.createParallelGroup(javax.swing
                .GroupLayout.Alignment.LEADING).addComponent(panel,
                javax.swing.GroupLayout.DEFAULT_SIZE, alto - 32,
                Short.MAX_VALUE));
        pack();
        setResizable(false);
        setLocationRelativeTo(null);
    }

    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        if (evt.getPropertyName().equals("close")) {
            this.dispose();
        }
    }
}
```

Clase DialogoAgregarOEditarObjeto

Esta clase define una ventana tipo diálogo con soporte para contener un panel estándar diseñado para la creación y edición de objetos.

```
package org.uaccess.utilidades.controles;

import java.awt.Component;
import java.awt.Frame;
import java.awt.Image;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
```

```
import javax.swing.GroupLayout;
import javax.swing.JPanel;
import org.uaccess.utilidades.Mensajero;

/**
 * Ventana tipo diálogo con soporte para contener un panel estándar
 * diseñado para la creación/edición de objetos
 * @author victor
 */
public class DialogoAgregarOEditarObjeto extends javax.swing.JDialog
        implements PropertyChangeListener {

    /*
     * Panel a ser mostrado en el diálogo
     */
    protected JPanel panel;
    /*
     * Variable tipo bandera que indica si la operación ha concluido
     * con éxito
     */
    private boolean operacionExitosa;
    /*
     * Componente padre del diálogo
     */
    private static final Component padre =
            Mensajero.getInstancia().getPadre();

    /**
     * Crea un nuevo diálogo con soporte para la creación/edición
     * de objetos
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
     * @param icono icono del diálogo
     */
    public DialogoAgregarOEditarObjeto(JPanel panel, String titulo,
            Image icono) {
        super((Frame)padre, true);
        this.panel = panel;
        initComponents();
        if(icono != null)
            setIconImage(icono);
        inicializar(titulo);
    }

    /**
     * Crea un nuevo diálogo con soporte para la creación/edición
     * de objetos
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
     * @param urlIcono URL del ícono del diálogo
```

```
     */
    public DialogoAgregarOEditarObjeto(JPanel panel, String titulo,
            URL urlIcono) {
        this(panel, titulo);
        Image icono;
        try {
            icono = ImageIO.read(urlIcono);
            setIconImage(icono);
        } catch (IOException ex) {
            icono = null;
        }
    }

    /**
     * Crea un nuevo diálogo con soporte para la creación/edición
     * de objetos
     * @param panel panel contenido en el diálogo
     * @param titulo título del diálogo
     */
    public DialogoAgregarOEditarObjeto(JPanel panel, String titulo) {
        this(panel, titulo, (Image)null);
    }

    /**
     * @deprecated Este método no es funcional. Para mostrar el
     * diálogo utilice mostrar()
     * @see #mostrar()
     * @param visible
     */
    @Override
    public void setVisible(boolean visible){
        Logger.getLogger(DialogoAgregarOEditarObjeto.class.getName())
                .log(Level.SEVERE, "DialogoAgregarOEditarObjeto"
                + ".setVisible(boolean) es un método disfuncional",
                (Object)null);
    }

    /**
     * Muestra el diálogo para agregar/editar un objeto
     * @return 'true' si la operación concluyó satisfactoriamente.
     * Caso contrario, retorna 'false'
     */
    public boolean mostrar(){
        if(panel.getClass().getSuperclass() ==
                PanelAgregarOEditarObjeto.class){
            ((PanelAgregarOEditarObjeto)panel)
                    .actualizarDatosRelacionados();
        }
        if(panel.getClass().getSuperclass() ==
                PanelAgregarOEditarObjeto.class){
```

```
            if(!((PanelAgregarOEditarObjeto)panel).isNuevoRegistro()){
                ((PanelAgregarOEditarObjeto)panel)
                        .sincronizarControlesConObjeto();
            }
        }
        operacionExitosa = false;
        super.setVisible(true);
        return operacionExitosa;
    }

    /**
     * Inicializa el diálogo con las carácterísticas respectivas
     * @param titulo título del diálogo
     */
    protected void inicializar(String titulo) {
        int ancho = panel.getPreferredSize().width;
        int alto = panel.getPreferredSize().height;
        setTitle(titulo);
        panel.addPropertyChangeListener(this);
        GroupLayout layout = new GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(layout.createParallelGroup(javax.swing
                .GroupLayout.Alignment.LEADING).addComponent(panel,
                javax.swing.GroupLayout.DEFAULT_SIZE, ancho - 10,
                Short.MAX_VALUE));
        layout.setVerticalGroup(layout.createParallelGroup(javax.swing
                .GroupLayout.Alignment.LEADING).addComponent(panel,
                javax.swing.GroupLayout.DEFAULT_SIZE, alto - 32,
                Short.MAX_VALUE));
        pack();
        setResizable(false);

        setLocationRelativeTo(null);
    }

     /**
     * Gestiona los eventos recibidos de los controles escuchados
     * @param evt Evento desencadenado por el componente emisor:
     * "operacionExitosa" u "operacionCancelada".
     * El primero indica que la operación finalizó de manera correcta;
     * el segundo que fue cancelada.
     * Ambos cierran el diálogo
     */
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        if (evt.getPropertyName().equals("operacionExitosa")) {
            operacionExitosa = true;
            this.dispose();
            return;
        }
```

```
        if (evt.getPropertyName().equals("operacionCancelada")) {
            operacionExitosa = false;
            this.dispose();
            return;
        }
    }
}
```

Clase InternalFrame

Esta clase corresponde a una ventana interna con soporte para contener un panel previamente creado.

```
package org.uaccess.utilidades.controles;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import javax.swing.GroupLayout;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JPanel;

/**
 * Ventana interna con soporte para contener un panel previamente creado
 * @author victor
 */
public class InternalFrame extends javax.swing.JInternalFrame implements
        PropertyChangeListener {

    /**
     * Panel contenido en el frame
     */
    private JPanel panel;


    /**
     * Crea un nuevo frame interno con los parámetros indicados
     * @param panel panel que será contenido en el frame
     */
    public InternalFrame(JPanel panel) {
        inicializar(panel, "", panel.getPreferredSize().width,
                panel.getPreferredSize().height, (Icon)null);
    }

    /**
```

```
     * Crea un nuevo frame interno con los parámetros indicados
     * @param panel panel contenido en el frame
     * @param titulo título del formulario
     */
    public InternalFrame(JPanel panel, String titulo) {
        inicializar(panel, titulo, panel.getPreferredSize().width,
                panel.getPreferredSize().height, (Icon)null);
    }

    /**
     * Crea un nuevo frame interno con los parámetros indicados
     * @param panel panel contenido en el frame
     * @param titulo título del formulario
     * @param icono ícono del frame interno
     */
    public InternalFrame(JPanel panel, String titulo, Icon icono) {
        inicializar(panel, titulo, panel.getPreferredSize().width,
                panel.getPreferredSize().height, icono);
    }

    /**
     * Crea un nuevo frame interno con los parámetros indicados
     * @param panel panel contenido en el frame
     * @param titulo título del formulario
     * @param ancho ancho para el formulario
     * @param alto altura para el formulario
     */
    public InternalFrame(JPanel panel, String titulo, int ancho,
            int alto) {
        inicializar(panel, titulo, ancho, alto, (Icon)null);
    }

    /**
     * Crea un nuevo frame interno con los parámetros indicados
     * @param panel panel contenido en el frame
     * @param titulo título del formulario
     * @param ancho ancho para el formulario
     * @param alto altura para el formulario
     * @param icono ícono para el frame
     */
    public InternalFrame(JPanel panel, String titulo, int ancho,
            int alto, String icono) {
        inicializar(panel, titulo, ancho, alto, icono);
    }

    /**
     * Obtiene el panel contenido en el frame
     * @return el panel contenido en el InternalFrame
     */
    public JPanel getPanel() {
```

```
        return panel;
    }

    /**
     * Inicializa el frame con los parámetro indicados
     * @param panel panel contenido en el frame
     * @param titulo título del formulario
     * @param ancho ancho para el formulario
     * @param alto altura para el formulario
     * @param icono ícono para el frame
     */
    private void inicializar(JPanel panel, String titulo, int ancho,
            int alto, String icono) {
        inicializar(panel, titulo, ancho, alto, new ImageIcon(getClass()
                .getResource("/images/16/" + icono)));
    }

        /**
     * Inicializa el frame con los parámetro indicados
     * @param panel panel contenido en el frame
     * @param titulo título del formulario
     * @param ancho ancho para el formulario
     * @param alto altura para el formulario
     * @param icono ícono para el frame
     */
    private void inicializar(JPanel panel, String titulo, int ancho,
            int alto, Icon icono) {
        setClosable(true);
        setIconifiable(true);
        setMaximizable(true);
        setResizable(true);
        setTitle(titulo);
        this.panel = panel;
        this.panel.addPropertyChangeListener("close", this);
        if (icono != null) {
            setFrameIcon(icono);
        }
        GroupLayout layout = new GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(layout.createParallelGroup(javax.swing
                .GroupLayout.Alignment.LEADING).addComponent(panel,
                javax.swing.GroupLayout.DEFAULT_SIZE, ancho - 10,
                Short.MAX_VALUE));
        layout.setVerticalGroup(layout.createParallelGroup(javax.swing
                .GroupLayout.Alignment.LEADING).addComponent(panel,
                javax.swing.GroupLayout.DEFAULT_SIZE, alto - 32,
                Short.MAX_VALUE));
        pack();
        setMinimumSize(getSize());
    }
```

```
    /**
     * Gestiona los eventos recibidos de los controles escuchados
     * @param evt Evento desencadenado por el componente emisor:
     * "close" para cerrar el frame
     */
    @Override
    public void propertyChange(PropertyChangeEvent evt) {
        if (evt.getPropertyName().equals("close")) {
            dispose();
        }
    }
}
```

Clase InternalFramePrivado

La clase "InternalFramePrivado" hereda de la clase "InternalFrame" y define una ventana interna con soporte para contener un panel previamente creado y mostrarlo únicamente cuando el usuario posee el permiso requerido.

```
package org.uaccess.utilidades.controles;

import java.beans.PropertyVetoException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.Icon;
import javax.swing.JDesktopPane;
import javax.swing.JPanel;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.validacion.ValidadorDeUsuarios;

/**
 * Ventana interna con soporte para contener un panel previamente creado
 * y ser mostrado únicamente cuando el usuario posee el permiso requerido
 * @author victor
 */
public class InternalFramePrivado extends InternalFrame {

    /**
     * Panel contenedor del internal frame
     */
    JDesktopPane desktopPane;
```

```
    /**
     * Privilegio requerido por el usuario para abrir el formulario
     */
    String privilegioRequerido;

    /**
     * Crea una nueva ventana privada con los parámetros indicados
     * @param desktopPane contendor de la ventana
     * @param panel panel contenido en el frame
     * @param titulo título de la ventana
     * @param permisoRequerido permiso requerido para ser mostrado
     * @param icono ícono de la ventana
     */
    public InternalFramePrivado(JDesktopPane desktopPane, JPanel panel,
            String titulo, String permisoRequerido, Icon icono) {
        super(panel, titulo, icono);
        this.desktopPane = desktopPane;
        this.privilegioRequerido = permisoRequerido;
    }

    /**
     * Crea una nueva ventana privada con los parámetros indicados
     * @param desktopPane contendor de la ventana
     * @param panel panel contenido en el frame
     * @param titulo título de la ventana
     * @param permisoRequerido permiso requerido para ser mostrado
     */
    public InternalFramePrivado(JDesktopPane desktopPane, JPanel panel,
            String titulo, String permisoRequerido){
        this(desktopPane, panel, titulo, permisoRequerido, null);
    }

    /**
     * Muestra la ventana, si y solamente si el usuario posee el
     * permiso requerido
     */
    public void mostrar(){
        if(ValidadorDeUsuarios.getInstancia()
                .verificarPermiso(privilegioRequerido)){
            try {
                desktopPane.add(this);
                super.setLocation((desktopPane.getWidth() / 2)
                        - (getWidth() / 2), (desktopPane.getHeight() / 2)
                        - (getHeight() / 2));
                super.setSelected(true);
                super.setVisible(true);
            } catch (PropertyVetoException ex) {
                Logger.getLogger(InternalFramePrivado.class
                        .getName()).log(Level.SEVERE, null, ex);
```

```
            }
        } else {
            Mensajero.getInstancia()
                    .mostrarError("Acceso denegado", null);
        }
    }

    /**
     * Método sobreescrito para ser funcional solamente cuando se
     * desea ocultar el frame,
     * caso contrario debe utilizarse mostrar()
     * @param visible
     * @see #mostrar()
     */
    @Override
    public void setVisible(boolean visible) {
        if(!visible)
            super.setVisible(visible);
    }
}
```

Clase PanelAdministradorDeTabla

Esta clase hereda de "JPanel" y define un panel con soporte para la administración de los objetos contenidos en una tabla.

```
package org.uaccess.utilidades.controles;

import ca.odell.glazedlists.gui.AdvancedTableFormat;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.DefaultComboBoxModel;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.validacion
            .VerificadorDeIntegridadReferencial;

/**
 * Panel con soporte para la administración de los objetos contenidos
 * en una tabla
 * @author victor
```

```
 */
public class PanelAdministradorDeTabla extends javax.swing.JPanel {

    /** Referencia al administrador de persistencia para la tabla */
    private AdministradorPersistencia administrador;
    /** La clase de los objetos contenidos en la tabla administrada */
    private Class claseDelObjeto;
    /** Consulta que devuelve los objetos contenidos en la
     * base de datos */
    private String consultaDeDatos;
    /** Panel utilizado para agregar y editar objetos */
    private PanelAgregarOEditarObjeto panelAgregarOEditar;
    /** Campos tomados en cuenta como "todos los campos" en las opciones
     * de filtrado */
    private int[] camposDeFiltrado;
    /** Campos que permiten filtrado individual siendo mostrados en las
     * opciones de filtrado */
    private int[] camposDeFiltradoIndividual;
    /** Icono mostrado en el diálogo de creación y edición de objetos */
    private Icon iconoDelDialogo;

    /**
     * Crea un nuevo panel para la administración de una tabla de objetos
     * con los parámetros indicados
     * @param claseDelObjeto clase correspondiente al tipo de objeto
     * administrado
     * @param consultaDeDatos consulta que devuelve los objetos
     * contenidos en la tabla
     * @param nombresDeCampos campos del objeto
     * @param encabezadosDeColumnas encabezados de las
     * columnas de la tabla
     * @param anchosDeColumnas anchos de las columnas de la tabla
     * @param formatoDeTabla formato de la tabla de objetos
     * @param camposDeFiltrado campos incluidos en el filtrado de objetos
     * por "todos los campos"
     * @param camposDeFiltradoIndividual campos que permiten filtrado
     * individual
     * @param panelAgregarOEditar panel utilizado para la creación y
     * edición de objetos
     * @param iconoDelDialogo ícono del diálogo de creación y edición
     * de objetos
     */
    public PanelAdministradorDeTabla(Class claseDelObjeto,
            String consultaDeDatos, String[] nombresDeCampos,
            String[] encabezadosDeColumnas, int[] anchosDeColumnas,
            AdvancedTableFormat formatoDeTabla, int[] camposDeFiltrado,
            int[] camposDeFiltradoIndividual, PanelAgregarOEditarObjeto
            panelAgregarOEditar, Icon iconoDelDialogo) {
        initComponents();
        DefaultComboBoxModel modeloJcb = new DefaultComboBoxModel(
```

```
                new Object[] {"Todos los campos"});

        administrador = AdministradorPersistencia.getInstancia();
        this.claseDelObjeto = claseDelObjeto;
        ploTabla.setNombresDeLasPropiedades(nombresDeCampos);
        ploTabla.setEncabezadosDeLasColumnas(encabezadosDeColumnas);
        ploTabla.setAnchosDeLasColumnas(anchosDeColumnas);
        ploTabla.setPropiedadesDeFiltrado(camposDeFiltrado);
        ploTabla.setFormatoDeTabla(formatoDeTabla);
        this.consultaDeDatos = consultaDeDatos;
        ploTabla.enlistar(administrador.obtenerLista(
                this.consultaDeDatos));

        this.camposDeFiltrado = camposDeFiltrado;
        this.camposDeFiltradoIndividual = camposDeFiltradoIndividual;
        for(int i = 0; i < this.camposDeFiltradoIndividual.length; i++)
            modeloJcb.addElement(encabezadosDeColumnas[this
                    .camposDeFiltradoIndividual[i]]);

        jcbCampoDeBusqueda.setModel(modeloJcb);
        this.panelAgregarOEditar = panelAgregarOEditar;
        this.iconoDelDialogo = iconoDelDialogo;
    }

    /**
     * Crea un nuevo panel para la administración de una tabla de
     * objetos con los parámetros indicados
     * * @param claseDelObjeto clase correspondiente al tipo de
     * objeto administrado
     * @param consultaDeDatos consulta que devuelve los objetos
     * contenidos en la tabla
     * @param nombresDeCampos campos del objeto
     * @param encabezadosDeColumnas encabezados de las columnas
     * de la tabla
     * @param formatoDeTabla formato de la tabla de objetos
     * @param camposDeFiltrado campos incluidos en el filtrado de
     * objetos por "todos los campos"
     * @param camposDeFiltradoIndividual campos que permiten filtrado
     * individual
     * @param panelAgregarOEditar panel utilizado para la creación
     * y edición de objetos
     * @param iconoDelDialogo ícono del diálogo de creación
     * y edición de objetos
     */
    public PanelAdministradorDeTabla(Class claseDelObjeto,
            String consultaDeDatos, String[] nombresDeCampos,
            String[] encabezadosDeColumnas, AdvancedTableFormat
            formatoDeTabla, int[] camposDeFiltrado, int[]
            camposDeFiltradoIndividual, PanelAgregarOEditarObjeto
            panelAgregarOEditar, Icon iconoDelDialogo) {
```

```
        this(claseDelObjeto, consultaDeDatos, nombresDeCampos,
                encabezadosDeColumnas, null, formatoDeTabla,
                camposDeFiltrado, camposDeFiltradoIndividual,
                panelAgregarOEditar, iconoDelDialogo);
    }

    /**
     * Crea un nuevo panel para la administración de una tabla
     * de objetos con los parámetros indicados
     * @param claseDelObjeto clase correspondiente al tipo de
     * objeto administrado
     * @param consultaDeDatos consulta que devuelve los objetos
     * contenidos en la tabla
     * @param nombresDeCampos campos del objeto
     * @param encabezadosDeColumnas encabezados de las columnas
     * de la tabla
     * @param anchosDeColumnas anchos de las columnas de la tabla
     * @param camposDeFiltrado campos incluidos en el filtrado de
     * objetos por "todos los campos"
     * @param camposDeFiltradoIndividual campos que permiten
     * filtrado individual
     * @param panelAgregarOEditar panel utilizado para la creación
     * y edición de objetos
     * @param iconoDelDialogo ícono del diálogo de creación y
     * edición de objetos
     */
    public PanelAdministradorDeTabla(Class claseDelObjeto,
            String consultaDeDatos, String[] nombresDeCampos,
            String[] encabezadosDeColumnas, int[] anchosDeColumnas,
            int[] camposDeFiltrado, int[] camposDeFiltradoIndividual,
            PanelAgregarOEditarObjeto panelAgregarOEditar,
            Icon iconoDelDialogo) {
        this(claseDelObjeto, consultaDeDatos, nombresDeCampos,
                encabezadosDeColumnas, anchosDeColumnas, null,
                camposDeFiltrado, camposDeFiltradoIndividual,
                panelAgregarOEditar, iconoDelDialogo);
    }

    /**
     * Crea un nuevo panel para la administración de una tabla de
     * objetos con los parámetros indicados
     * @param claseDelObjeto clase correspondiente al tipo de
     * objeto administrado
     * @param consultaDeDatos consulta que devuelve los objetos
     * contenidos en la tabla
     * @param nombresDeCampos campos del objeto
     * @param encabezadosDeColumnas encabezados de las columnas
     * de la tabla
     * @param camposDeFiltrado campos incluidos en el filtrado
     * de objetos por "todos los campos"
```

```
     * @param camposDeFiltradoIndividual campos que permiten
     * filtrado individual
     * @param panelAgregarOEditar panel utilizado para la creación
     * y edición de objetos
     * @param iconoDelDialogo ícono del diálogo de creación y
     * edición de objetos
     */
    public PanelAdministradorDeTabla(Class claseDelObjeto,
            String consultaDeDatos, String[] nombresDeCampos,
            String[] encabezadosDeColumnas, int[] camposDeFiltrado,
            int[] camposDeFiltradoIndividual, PanelAgregarOEditarObjeto
            panelAgregarOEditar, Icon iconoDelDialogo) {
        this(claseDelObjeto, consultaDeDatos, nombresDeCampos,
                encabezadosDeColumnas, null, null, camposDeFiltrado,
                camposDeFiltradoIndividual, panelAgregarOEditar,
                iconoDelDialogo);
    }

    /**
     * Refresca los datos contenidos en la tabla
     */
    private void refrescarDatos(){
        ploTabla.setElementos(administrador
                .obtenerLista(this.consultaDeDatos));
    }

    /**
     * Gestiona la creación de un nuevo registro (objeto) en la tabla
     */
    private void nuevoRegistro(){
        Object nuevoObjeto;
        try {
            nuevoObjeto = claseDelObjeto.newInstance();
        } catch (InstantiationException ex) {
            Logger.getLogger(PanelAdministradorDeTabla.class
                    .getName()).log(Level.SEVERE, null, ex);
            return;
        } catch (IllegalAccessException ex) {
            Logger.getLogger(PanelAdministradorDeTabla.class
                    .getName()).log(Level.SEVERE, null, ex);
            return;
        }
        boolean operacionEjecutada = false;
        panelAgregarOEditar.setObjeto(nuevoObjeto);
        panelAgregarOEditar.setNuevoRegistro(true);
        if(iconoDelDialogo != null)
            operacionEjecutada = new DialogoAgregarOEditarObjeto(
                    panelAgregarOEditar, "Nuevo registro",
                    ((ImageIcon)iconoDelDialogo).getImage()).mostrar();
        else
```

```
            operacionEjecutada = new DialogoAgregarOEditarObjeto(
                    panelAgregarOEditar, "Nuevo registro").mostrar();
        if(operacionEjecutada)
            ploTabla.situar(nuevoObjeto);
    }

    /**
     * Gestiona la edición de un registro (objeto) contenido en la tabla
     */
    private void editarRegistro(){
        if(ploTabla.getObjetoSeleccionado() == null){
            Mensajero.getInstancia().mostrarAdvertencia("No hay "
                    + "ningún registro a editar");
            return;
        }
        Object objeto = ploTabla.getObjetoSeleccionado();
        panelAgregarOEditar.setObjeto(objeto);
        panelAgregarOEditar.setNuevoRegistro(false);
        boolean operacionEjecutada = false;
        if(iconoDelDialogo != null)
            operacionEjecutada = new DialogoAgregarOEditarObjeto(
                    panelAgregarOEditar, "Editando objeto",
                    ((ImageIcon)iconoDelDialogo).getImage()).mostrar();
        else
            operacionEjecutada = new DialogoAgregarOEditarObjeto(
                    panelAgregarOEditar, "Editando objeto").mostrar();
        if(operacionEjecutada)
            ploTabla.reemplazar(ploTabla.getFilaSeleccionada(), objeto);
    }

    /**
     * Procesa la solicitud de eliminación de un nuevo registro
     * (objeto) contenido en la tabla
     */
    private void eliminarRegistro(){
        if(ploTabla.getObjetoSeleccionado() == null){
            Mensajero.getInstancia().mostrarAdvertencia("No hay ningún"
                    + " registro a eliminar");
            return;
        }
        Object objeto = ploTabla.getObjetoSeleccionado();
        if(VerificadorDeIntegridadReferencial.getInstancia()
                .verificarPrevioEliminacion(objeto)){
            if(Mensajero.getInstancia().hacerPregunta("¿Realmente desea"
                    + " eliminar el registro seleccionado?")){
                if(AdministradorPersistencia.getInstancia()
                        .eliminar(objeto))
                    ploTabla.remover(objeto);
                else
                    Mensajero.getInstancia().mostrarError("El objeto no"
```

```
                            + " ha podido ser eliminado", null);
            }
        } else {
            Mensajero.getInstancia().mostrarAdvertencia("El objeto"
                    + " seleccionado no puede ser eliminado porque"
                    + " está siendo utilizado");
        }
    }

    /**
     * Establece, en base al ítem seleccionado en la interfaz de usuario,
     * los campos de filtrado de la tabla contenedora de objetos
     */
    private void establecerCamposDeFiltrado(){
        if(jcbCampoDeBusqueda.getSelectedIndex() == 0)
            ploTabla.setPropiedadesDeFiltrado(camposDeFiltrado);
        else{
            ploTabla.setPropiedadesDeFiltrado(
                    camposDeFiltradoIndividual[jcbCampoDeBusqueda
                    .getSelectedIndex() - 1]);
        }
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtRefrescarActionPerformed(
            java.awt.event.ActionEvent evt) {
        refrescarDatos();
    }

    private void jbtAgregarActionPerformed(
            java.awt.event.ActionEvent evt) {
        nuevoRegistro();
    }

    private void jbtEditarActionPerformed(
            java.awt.event.ActionEvent evt) {
        editarRegistro();
    }

    private void jbtEliminarActionPerformed(
            java.awt.event.ActionEvent evt) {
        eliminarRegistro();
    }

    private void jcbCampoDeBusquedaActionPerformed(
            java.awt.event.ActionEvent evt) {
        establecerCamposDeFiltrado();
```

```
    }


    // Variables declaration - do not modify
    private javax.swing.JPanel jPanel1;
    private javax.swing.JToolBar.Separator jSeparator1;
    private javax.swing.JButton jbtAgregar;
    private javax.swing.JButton jbtEditar;
    private javax.swing.JButton jbtEliminar;
    private javax.swing.JButton jbtRefrescar;
    private javax.swing.JComboBox jcbCampoDeBusqueda;
    private javax.swing.JLabel jlbBuscar;
    private javax.swing.JLabel jlbEn;
    private javax.swing.JToolBar.Separator jspSeparador;
    private javax.swing.JToolBar jtbToolBar;
    private javax.swing.JTextField jtfBuscar;
    private org.uaccess.utilidades.controles
            .PanelListaDeObjetos ploTabla;
}
```

Clase PanelAgregarOEditarObjeto

Esta clase define un panel auxiliar estándar para agregar o editar un objeto desde un "PanelAdministradorDeTabla".

```
package org.uaccess.utilidades.controles;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;

/**
 * Panel auxiliar para agregar o editar un objeto desde
 * PanelAdministradorDeTabla
 * @author victor
 */
public abstract class PanelAgregarOEditarObjeto
        extends javax.swing.JPanel {
    /** Bandera que indica si el objeto que se está editando
     * es nuevo o no */
    protected boolean nuevoRegistro;
    /** Variable que almacena el objeto original rebibido para
     * la operación */
    protected Object objetoOriginal;
    /** Variable que almacena el objeto resultante de la operación */
```

```
    protected Object objeto;

    /**
     * Constructor para los elementos del tipo
     * PanelAgregarOEditarObjeto
     * @param registro registro original recibido por el panel
     * @param esNuevoRegistro indicador de si el registro recibido
     * es nuevo o no
     */
    public PanelAgregarOEditarObjeto(Object registro,
            boolean esNuevoRegistro) {
        inicializarComponentes();
        setNuevoRegistro(esNuevoRegistro);
        setObjeto(registro);
    }

    /**
     * Selecciona el valor de la bandera 'nuevoRegistro'
     * @param nuevoRegistro indica si se está editando un
     * registro nuevo o no
     */
    public void setNuevoRegistro(boolean nuevoRegistro) {
        this.nuevoRegistro = nuevoRegistro;
    }

    /**
     * Obtiene el valor de la bandera 'nuevoRegistro'
     * @return 'true' si se está creando un nuevo registro y 'false'
     * si se está editando uno existente
     */
    public boolean isNuevoRegistro() {
        return nuevoRegistro;
    }

    /**
     * Selecciona el objeto que será editado en el panel
     * @param objeto objeto a editar
     */
    public void setObjeto(Object objeto){
        this.objetoOriginal = objeto;
        this.objeto = objeto;
        sincronizarControlesConObjeto();
        try {
            this.objeto = objeto.getClass().newInstance();
            sincronizarObjetoConControles(this.objeto);
        } catch (InstantiationException ex) {
            Logger.getLogger(PanelAgregarOEditarObjeto.class
                    .getName()).log(Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            Logger.getLogger(PanelAgregarOEditarObjeto.class
```

```
                .getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Obtiene el objeto editado en el panel
     * @return objeto resultante del proceso de edición
     */
    public Object getObjeto() {
        sincronizarObjetoConControles(objeto);
        return objeto;
    }

    /**
     * Obtiene el botón por defecto del formulario
     * @return botón por defecto
     */
    public abstract JButton getBotonPorDefecto();

    /**
     * Actualiza datos de otras entidades relacionadas a la
     * agregada o editada
     */
    public abstract void actualizarDatosRelacionados();

    /**
     * Asigna los valores del objeto a los controles
     */
    public abstract void sincronizarControlesConObjeto();

    /**
     * Asigna los valores de los controles al objeto que le
     * sea indicado
     * @param objeto objeto al que se asignarán los valores,
     * generalmente objeto u objetoOriginal
     */
    public abstract void sincronizarObjetoConControles(
            Object objeto);

    /**
     * Ejecuta la validación del objeto que está siendo editado
     * @return
     */
    protected abstract boolean validar();

    /**
     * Ejecuta la inserción del objeto a la base de datos
     */
    protected abstract void ejecutarInsercion();
```

```
    /**
     * Ejecuta la actualización del objeto en la base de datos
     */
    protected abstract void ejecutarActualizacion();

    /**
     * Guarda el objeto en la base de datos, agregándolo si es nuevo
     * y actualizándolo si ya existe
     */
    protected void guardarObjeto(){
        if(nuevoRegistro)
            ejecutarInsercion();
        else
            ejecutarActualizacion();
    }

    /**
     * Inicializa los componentes de la interfaz gráfica
     */
    protected abstract void inicializarComponentes();
}
```

Clase PanelImagen

La clase "PanelImagen" es un "JPanel" con soporte para mostrar imágenes.

```
package org.uaccess.utilidades.controles;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;

/**
 * Panel con soporte para mostrar imágenes
 * @author victor
 */
public class PanelImagen extends javax.swing.JPanel {

    /** Imagen mostrada en el panel */
    private BufferedImage imagen;
```

```
    /**
     * Crea un nuevo panel con soporte para mostrar imágenes
     */
    public PanelImagen() {
        initComponents();
    }

    /**
     * Selecciona la imagen a ser mostrada en el panel
     * @param urlImagen URL de la imagen
     */
    public void setImagen(URL urlImagen) {
        try {
            imagen = ImageIO.read(urlImagen);
        } catch (IOException ex) {
            Logger.getLogger(PanelImagen.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
        repaint();
    }


    /**
     * Pinta la imagen sobre el panel
     * @param g componente graphics del panel
     */
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(imagen, 0, 0,this.getSize().width,
                this.getSize().height, null);
    }
}
```

Clase PanelListaDeObjetos

Esta clase corresponde a un panel con soporte para listar objetos en una tabla, donde cada fila contiene un objeto y las columnas corresponden a sus campos.

```
package org.uaccess.utilidades.controles;

import ca.odell.glazedlists.BasicEventList;
import ca.odell.glazedlists.EventList;
import ca.odell.glazedlists.FilterList;
import ca.odell.glazedlists.GlazedLists;
import ca.odell.glazedlists.SortedList;
import ca.odell.glazedlists.gui.AbstractTableComparatorChooser;
import ca.odell.glazedlists.gui.AdvancedTableFormat;
import ca.odell.glazedlists.swing.EventSelectionModel;
import ca.odell.glazedlists.swing.EventTableModel;
import ca.odell.glazedlists.swing.TableComparatorChooser;
import ca.odell.glazedlists.swing.TextComponentMatcherEditor;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.List;
import javax.swing.ListSelectionModel;
import javax.swing.SwingConstants;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.text.JTextComponent;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel con soporte para listar objetos en una tabla donde cada fila
 * contiene un objeto y las columnas son sus campos
 * @author victor
 */
public class PanelListaDeObjetos extends javax.swing.JPanel {

    /** Lista de objetos contenidos en el panel */
    private EventList lista;
    /** Encabezados de las columnas de la tabla */
    private String[] encabezadosDeLasColumnas;
    /** Propiedades utilizadas para filtrar objetos */
    private String[] propiedadesDeFiltrado;
    /** Nombres de las propiedades del tipo de objeto listado */
    private String[] nombresDeLasPropiedades;
    /** Ancho de las columnas de la tabla */
    private int[] anchosDeLasColumnas;
    /** Modelo de selección de la tabla */
    private EventSelectionModel modeloDeSeleccion;
    /** Modo de selección de objetos en la tabla */
    private int modoDeSeleccion;
    /** Formato de la tabla contenedora de objetos */
    private AdvancedTableFormat formatoDeTabla;
    /** Objeto vinculante entre el componente de filtrado y la tabla */
    private TextComponentMatcherEditor
            emparejadorDeComponenteDeFiltrado;
    /** Componente de texto que define el filtro aplicado a los
```

```
     * objetos contenidos */
    private JTextComponent componenteDeFiltrado;

    /**
     * Crea un nuevo panel con soporte para listar objetos
     */
    public PanelListaDeObjetos() {
        initComponents();
        tabla.addMouseListener(new java.awt.event.MouseAdapter() {
            @Override
            public void mouseClicked(java.awt.event.MouseEvent e) {
                if (e.getClickCount() == 1) {
                    firePropertyChange("selection", null,
                            getObjetosSeleccionados());
                }
            }
        });
        tabla.getActionMap().put("selectNextRowCell",
                new javax.swing.AbstractAction() {
            @Override
            public void actionPerformed(java.awt.event.ActionEvent e) {
                PanelListaDeObjetos.this.firePropertyChange("selection",
                        null, getObjetosSeleccionados());
            }
        });
        DefaultTableCellRenderer renderizador =
                (DefaultTableCellRenderer) tabla
                .getDefaultRenderer(Date.class);
        renderizador.setHorizontalAlignment(SwingConstants.CENTER);
        lista = new BasicEventList();
        modeloDeSeleccion = new EventSelectionModel(lista);
    }

    /**
     * Obtiene la lista de objetos del panel
     * @return la lista de objetos contenidos en el panel
     */
    public EventList getLista() {
        return lista;
    }

    /**
     * Selecciona los elementos contenidos en la lista de objetos
     * @param elementos los elementos a ser contenidos en la lista
     */
    public void setElementos(Collection elementos) {
        getLista().clear();  //remueve los elementos anteriores;
        getLista().addAll(elementos); //agrega los nuevos elementos;
    }
```

```
    /**
     * Obtiene los encabezados de las columnas
     * @return los encabezados de las columnas de la lista
     */
    public String[] getEncabezadosDeLasColumnas() {
        return encabezadosDeLasColumnas;
    }

    /**
     * Selecciona los encabezados de las columnas de la lista
     * @param encabezados para las columnas
     */
    public void setEncabezadosDeLasColumnas(String... encabezados) {
        this.encabezadosDeLasColumnas = encabezados;
    }

    /**
     * Obtiene las propiedades involucradas en el filtrado de objetos
     * @return los nombres de las propiedades de filtrado
     */
    public String[] getPropiedadesDeFiltrado() {
        return propiedadesDeFiltrado;
    }

    /**
     * Selecciona las propiedades involucradas en el
     * filtrado de objetos
     * @param propiedadesDeFiltrado los nombres de las propiedades
     * de filtrado
     */
    private void setPropiedadesDeFiltrado(
            String... propiedadesDeFiltrado) {
        this.propiedadesDeFiltrado = propiedadesDeFiltrado;
        if(emparejadorDeComponenteDeFiltrado != null && this
                .componenteDeFiltrado != null)
            emparejadorDeComponenteDeFiltrado.setFilterator(
                    GlazedLists.textFilterator(propiedadesDeFiltrado));
    }

    /**
     * Selecciona las propiedades involucradas en el
     * filtrado de objetos
     * @param propiedadesDeFiltrado los índices de las
     * propiedades de filtrado
     */
    public void setPropiedadesDeFiltrado(
            int... filterPropertiesIndexes) {
        String[] filterPropertiesNames =
                new String[filterPropertiesIndexes.length];
        for (int i = 0; i < filterPropertiesIndexes.length; i++)
```

```
            filterPropertiesNames[i] = nombresDeLasPropiedades[
                    filterPropertiesIndexes[i]];
        setPropiedadesDeFiltrado(filterPropertiesNames);
    }

    /**
     * Obtiene los nombre de las propiedades del tipo de objeto listado
     * @return los nombres de las propiedades
     */
    public String[] getNombresDeLasPropiedades() {
        return nombresDeLasPropiedades;
    }

    /**
     * Selecciona los nombres de las propiedades del tipo de
     * objeto listado
     * @param los nombres de las propiedades
     */
    public void setNombresDeLasPropiedades(
            String... nombresDeLasPropiedades) {
        this.nombresDeLasPropiedades = nombresDeLasPropiedades;
    }

    /**
     * Obtiene los anchos de las columnas que conforman la lista
     * @return los anchos de las columnas de la lista
     */
    public int[] getAnchosDeLasColumnas() {
        return anchosDeLasColumnas;
    }

    /**
     * Selecciona los anchos de las columnas que conforman la lista
     * @param anchosDeLasColumnas los anchos de las columnas de la lista
     */
    public void setAnchosDeLasColumnas(int... anchosDeLasColumnas) {
        this.anchosDeLasColumnas = anchosDeLasColumnas;
    }

    /**
     * Obtiene el formato de la tabla en que se listan los objetos
     * @return formato de la tabla
     */
    public AdvancedTableFormat getFormatoDeTabla() {
        return formatoDeTabla;
    }

    /**
     * Selecciona el formato de la talba en que se listan los objetos
     * @param formatoDeTabla formato de la tabla
```

```
     */
    public void setFormatoDeTabla(AdvancedTableFormat formatoDeTabla) {
        this.formatoDeTabla = formatoDeTabla;
    }

    /**
     * Obtiene la tabla contenida en el panel
     * @return la tabla del panel
     */
    public javax.swing.JTable getTabla() {
        return tabla;
    }

    /**
     * Selecciona el componente tipo texto que será utilizado
     * para filtrar los objetos de la lista
     * @param componente componente tipo texto utilizado
     * para el filtrado
     */
    public void setComponenteDeFiltrado(JTextComponent componente) {
        componenteDeFiltrado = componente;
    }

    /**
     * Obtiene el modo de selección en que trabaja la lista
     * @return el modo de selección de la lista
     */
    public int getModoDeSeleccion() {
        return modoDeSeleccion;
    }

    /**
     * Indica el modo de selección en que trabaja la lista
     * ListSelectionModel.SINGLE_SELECTION
     * ListSelectionModel.SINGLE_INTERVAL_SELECTION
     * ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
     * @param modoDeSeleccion el modo de selección a utilizar
     */
    public void setModoDeSeleccion(int modoDeSeleccion) {
        this.modoDeSeleccion = modoDeSeleccion;
    }

    /**
     * Obtiene los objetos seleccionados en la lista
     * @return los objetos seleccionados
     */
    public List getObjetosSeleccionados() {
        return modeloDeSeleccion.getSelected();
    }
```

```
    /**
     * Obtiene el objeto seleccionado en la lista. Si hay varios
     * seleccionados, retorna el primero que forma parte de la selección
     * @return el objeto seleccionado
     */
    public Object getObjetoSeleccionado() {
        Object objetoSeleccionado = null;
        if (!modeloDeSeleccion.isSelectionEmpty()) {
            objetoSeleccionado = modeloDeSeleccion.getSelected().get(0);
        }
        return objetoSeleccionado;
    }

    /**
     * Obtiene el índice de la fila seleccionada. -1 indica que
     * no hay ninguna
     * @return el índice de la primera fila fila seleccionada
     */
    public int getFilaSeleccionada() {
        return tabla.getSelectedRow();
    }

    /**
     * Instala los objetos a ser listados en el panel
     * @param elementos elementos que se desea enlistar
     */
    public void enlistar(Collection elementos) {
        if (elementos == null) {
            elementos = new ArrayList();
        }
        try {
            lista.clear();
            lista.addAll(elementos);

            emparejadorDeComponenteDeFiltrado =
                    new TextComponentMatcherEditor(componenteDeFiltrado,
                    GlazedLists.textFilterator(propiedadesDeFiltrado));
            FilterList elementosFiltrados = new FilterList(
                    lista, emparejadorDeComponenteDeFiltrado);
            SortedList elementosOrdenados = new SortedList(
                    elementosFiltrados, null);

            if(formatoDeTabla == null){
                formatoDeTabla = (AdvancedTableFormat) GlazedLists
                        .tableFormat(nombresDeLasPropiedades,
                        encabezadosDeLasColumnas);
            }

            EventTableModel modeloDeTabla = new EventTableModel(
                    elementosOrdenados, formatoDeTabla);
```

```
            tabla.setModel(modeloDeTabla);

            TableComparatorChooser.install(tabla, elementosOrdenados,
                    AbstractTableComparatorChooser.SINGLE_COLUMN);

            modeloDeSeleccion = new EventSelectionModel(
                    elementosOrdenados);
            modeloDeSeleccion.setSelectionMode(modoDeSeleccion >= 0
                    && modoDeSeleccion < 3 ? modoDeSeleccion :
                        ListSelectionModel.SINGLE_SELECTION);
            tabla.setSelectionModel(modeloDeSeleccion);

            if (anchosDeLasColumnas != null) {
                for (int i = 0; i < anchosDeLasColumnas.length; i++) {
                    if (anchosDeLasColumnas[i] > 0) {
                        tabla.getColumnModel().getColumn(i)
                                .setPreferredWidth(
                                anchosDeLasColumnas[i]);
                    }
                }
            }

        } catch (Exception ex) {
            Mensajero.getInstancia().mostrarError(
                    "Error al listar los registros", ex);
        }
    }

    /**
     * Agregar un nuevo objeto a la lista
     * @param objeto objeto a ser agregado
     * @return 'true' si el objeto se ha agregado correctamente,
     * caso contrario 'false'
     */
    public boolean agregar(Object objeto) {
        boolean add = lista.add(objeto);
        return add;
    }

    /**
     * Remueve un objeto existente en la lista
     * @param objeto objeto a ser removido
     * @return 'true'si el objeto se ha removido, caso contrario 'false'
     */
    public boolean remover(Object objeto) {
        return lista.remove(objeto);
    }

    /**
```

```
     * Remueve un objeto existente en la lista
     * @param indice índice del objeto a remover
     * @return 'true'si el objeto se ha removido, caso contrario 'false'
     */
    public Object removerPorIndice(int indice) {
        return lista.remove(indice);
    }

    /**
     * Sitúa un nuevo objeto dentro de la lista
     * @param objeto el objeto que se desea situar en la lista
     * @return el elemento ya posicionado si se situó
     * satisfactoriamente, caso contrario 'null'
     */
    public Object situar(Object objeto) {
        Object set = null;
        int index = lista.indexOf(objeto);
        if (index > -1) {
            set = lista.set(index, objeto);
        } else {
            agregar(objeto);
        }
        return set;
    }

    /**
     * Reemplaza un objeto contenido en la lista en base a su
     * identificador
     * @param objeto objeto a reemplazar
     * @return el elemento ya reemplazado si se realizó
     * satisfactoriamente, caso contrario 'null'
     */
    public Object reemplazar(Object objeto) {
        Object set = null;
        set = lista.set(lista.indexOf(objeto), objeto);
        return set;
    }

    /**
     * Reemplaza un objeto contenido en la lista en base a su índice
     * @param indice índice del objeto a reemplazar
     * @param objeto objeto por el que será reemplazado
     * @return el elemento ya reemplazado si se realizó
     * satisfactoriamente, caso contrario 'null'
     */
    public Object reemplazar(int indice, Object objeto) {
        Object set;
        set = lista.set(indice, objeto);
        return set;
    }
```

```
    /**
     * Obtiene el índice del objeto especificado dentro de la lista
     * @param objeto objeto del que se desea obtener el índice
     * @return índice del objeto
     */
    public int obetenerIndice(Object objeto) {
        return lista.indexOf(objeto);
    }
}
```

Clase PanelParametrosParaReporte

Esta clase corresponde a la abstracción de un panel genérico para la selección de parámetros para la generación de reportes.

```
package org.uaccess.utilidades.controles;

import javax.swing.JButton;

/**
 * Panel genérico para la selección de parámetros para la
 * generación de reportes
 * @author victor
 */
public abstract class PanelParametrosParaReporte
        extends javax.swing.JPanel {

    /**
     * Contructor para los componentes del tipo
     * 'PanelParametrosParaReporte'
     */
    public PanelParametrosParaReporte() {
        inicializarComponentes();
        establecerValoresIniciales();
    }

    /**
     * Obtiene el botón por defecto del panel
     * @return botón por defecto
     */
    public abstract JButton getBotonPorDefecto();
```

```
    /**
     * Establece los valores iniciales de los controles del panel
     */
    protected abstract void establecerValoresIniciales();

    /**
     * Valida los parámetros establecidos en los controles
     * @return 'true' si los parámetros se han especificado de
     * manera correcta,
     * 'false' en caso contrario
     */
    protected abstract boolean validarParametros();

   /**
     * Genera el reporte a partir de los parámetros indicados
     */
    protected abstract void generarReporte();

    /**
     * Inicializa los componentes de la interfaz gráfica
     */
    protected abstract void inicializarComponentes();

     /**
      * Cancela la generación del reporte y emite la orden para
      * que la ventana
      * contenedora del panel sea cerrada
      */
    protected void cancelar(){
        firePropertyChange("close", true, null);
    }
}
```

Clase PanelSelectorDeObjetos

"PanelSelectorDeObjetos" define un panel diseñado para la selección de múltiples objetos.

```
package org.uaccess.utilidades.controles;

import java.util.Arrays;
import javax.swing.DefaultListModel;
import javax.swing.JList;

/**
```

```
 * Panel diseñado para la selección de múltiples objetos
 * @author victor
 */
public class PanelSelectorDeObjetos extends javax.swing.JPanel {

    /**
     * Crea un nuevo panel con soporte para la selección de
     * múltiples objetos
     */
    public PanelSelectorDeObjetos() {
        initComponents();
    }

    /**
     * Selecciona el encabezado de la sección de objetos
     * disponibles del panel
     * @param encabezado encabezado de los objetos disponibles
     */
    public void setEncabezadoDisponibles(String encabezado){
        jlbObjetosDisponibles.setText(encabezado);
    }

    /**
     * Selecciona el encabezado de la sección de objetos
     * seleccionados del panel
     * @param encabezado encabezado de los objetos seleccionados
     */
    public void setEncabezadoSeleccionados(String encabezado){
        jlbObjetosSeleccionados.setText(encabezado);
    }

    /**
     * Establece los objetos listados en el control
     * @param objetos objetos contenidos en el panel
     */
    public void setObjetos(Object[] objetos){
        jlsObjetosSeleccionados.setModel(new DefaultListModel());
        seleccionarYOrdenarObjetos(jlsObjetosDisponibles, objetos);
    }

    /**
     * Obtiene los objetos que han sido seleccionados en el panel
     * @return lista de objetos seleccionados
     */
    public Object[] getObjetosSeleccionados(){
        Object objetosSeleccionados[] = new Object[
                jlsObjetosSeleccionados.getModel().getSize()];
        for(int i=0; i<objetosSeleccionados.length; i++)
            objetosSeleccionados[i] = jlsObjetosSeleccionados
                    .getModel().getElementAt(i);
```

```
        return objetosSeleccionados;
    }

    /**
     * Establece los objetos seleccionados del control
     * @param objetos objetos que se desea seleccionar
     */
    public void setObjetosSeleccionados(Object[] objetos){
        DefaultListModel lmDisponibles = new DefaultListModel();
        DefaultListModel lmSeleccionados = new DefaultListModel();
        Object objeto;
        Arrays.sort(objetos);
        int totalDeObjetos = jlsObjetosDisponibles.getModel().getSize()
                + jlsObjetosSeleccionados.getModel().getSize();
        for(int i=0; i<totalDeObjetos; i++){
            if(i<jlsObjetosDisponibles.getModel().getSize())
                objeto = jlsObjetosDisponibles.getModel()
                        .getElementAt(i);
            else
                objeto = jlsObjetosSeleccionados.getModel()
                        .getElementAt(i - jlsObjetosDisponibles
                        .getModel().getSize());
            if(Arrays.binarySearch(objetos, objeto) < 0)
                lmDisponibles.add(lmDisponibles.getSize(), objeto);
            else
                lmSeleccionados.add(lmSeleccionados.getSize(), objeto);
        }
        jlsObjetosDisponibles.setModel(lmDisponibles);
        jlsObjetosSeleccionados.setModel(lmSeleccionados);
    }

    /**
     * Establece los objetos de la lista y los ordena
     * @param lista lista en la que serán contenidos los objetos
     * @param objetos objetos a ser contenidos en la lista
     */
    private void seleccionarYOrdenarObjetos(JList lista,
            Object[] objetos){
        DefaultListModel listModel = new DefaultListModel();
        Arrays.sort(objetos);
        for(int i=0; i<objetos.length; i++)
            listModel.add(i, objetos[i]);
        lista.setModel(listModel);
    }

    /**
     * Desplaza los objetos seleccionados en una lista hacia otra
     * @param jlsOrigen lista de donde se moverán los objetos
     * @param jlsDestino lista hacia donde se moverán los objetos
     */
```

```
    private void desplazarObjetosSeleccionados(JList jlsOrigen,
            JList jlsDestino){
        Object objetosDestino[] = new Object[jlsDestino.getModel()
                .getSize() + jlsOrigen.getSelectedValues().length];
        Object objetosOrigen[] = new Object[jlsOrigen.getModel()
                .getSize() - jlsOrigen.getSelectedValues().length];
        int iObj = 0;
        for(int i=0; i<jlsOrigen.getModel().getSize(); i++){
            if(!jlsOrigen.isSelectedIndex(i)){
                objetosOrigen[iObj] = jlsOrigen.getModel()
                        .getElementAt(i);
                iObj++;
            }
        }
        for(int i=0; i<objetosDestino.length; i++){
            if(i < jlsDestino.getModel().getSize())
                objetosDestino[i] = jlsDestino.getModel()
                        .getElementAt(i);
            else{
                objetosDestino[i] = jlsOrigen.getSelectedValues()[
                        i - jlsDestino.getModel().getSize()];
            }
        }
        seleccionarYOrdenarObjetos(jlsDestino, objetosDestino);
        seleccionarYOrdenarObjetos(jlsOrigen, objetosOrigen);
    }

    /**
     * Desplaza todos los objetos contenidos en una lista hacia otra
     * @param jlsOrigen lista de donde se moverán los objetos
     * @param jlsDestino lista hacia donde se moverán los objetos
     */
    private void desplazarObjetos(JList jlsOrigen, JList jlsDestino){
        if(jlsOrigen.getModel().getSize() < 1)
            return;
        jlsOrigen.setSelectionInterval(0, jlsOrigen.getModel()
                .getSize() - 1);
        desplazarObjetosSeleccionados(jlsOrigen, jlsDestino);
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAgregarActionPerformed(
            java.awt.event.ActionEvent evt) {
        desplazarObjetosSeleccionados(jlsObjetosDisponibles,
                jlsObjetosSeleccionados);
    }
```

```
    private void jbtQuitarActionPerformed(
            java.awt.event.ActionEvent evt) {
        desplazarObjetosSeleccionados(jlsObjetosSeleccionados,
                jlsObjetosDisponibles);
    }

    private void jbtAgregarTodosActionPerformed(
            java.awt.event.ActionEvent evt) {
        desplazarObjetos(jlsObjetosDisponibles,
                jlsObjetosSeleccionados);
    }

    private void jbtQuitarTodosActionPerformed(
            java.awt.event.ActionEvent evt) {
        desplazarObjetos(jlsObjetosSeleccionados,
                jlsObjetosDisponibles);
    }


    // Variables declaration - do not modify
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JButton jbtAgregar;
    private javax.swing.JButton jbtAgregarTodos;
    private javax.swing.JButton jbtQuitar;
    private javax.swing.JButton jbtQuitarTodos;
    private javax.swing.JLabel jlbObjetosDisponibles;
    private javax.swing.JLabel jlbObjetosSeleccionados;
    private javax.swing.JList jlsObjetosDisponibles;
    private javax.swing.JList jlsObjetosSeleccionados;
    // End of variables declaration
}
```

Clase PanelVisorHuella

Esta clase hereda de "JPanel" y define un panel con soporte para mostrar la imagen de una captura dactilar.

```
package org.uaccess.utilidades.controles;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.BufferedImage;
import org.uaccess.fprint.CapturaDactilar;
```

```
/**
 * Panel con soporte para mostrar la imagen de una captura dactilar
 * @author victor
 */
public class PanelVisorHuella extends javax.swing.JPanel {

    /** Captura dactilar mostrada en el panel */
    private CapturaDactilar capturaDactilar;
    /**Bandera que indica si la huella se mostrará binarizada (blanco
     * y negro) o no, en cuyo caso se mostrará en escala de grises */
    private boolean binarizada;

    /**
     * Crea un nuevo panel con soporte para mostrar una
     * captura dactilar
     */
    public PanelVisorHuella() {
        initComponents();
        binarizada = false;
    }

    /**
     * Selecciona la captura dactilar mostrada en el panel
     * @param captura captura dactilar a mostrar
     */
    public void setCaptura(CapturaDactilar captura) {
        this.capturaDactilar = captura;
        repaint();
    }

    /**
     * Obtiene la captura dactilar que está siendo mostrada en el panel
     * @return captura dactilar contenida en el panel
     */
    public CapturaDactilar getCaptura() {
        return capturaDactilar;
    }

    /**
     * Establece el modo en que se muestra la captura dactilar
     * @param binarizada 'true' indica que la huella será mostrada
     * binarizada (blanco y negro), 'false' indica que se mostrará
     * en escala de grises
     */
    public void setBinarizada(boolean binarizada) {
        this.binarizada = binarizada;
        repaint();
    }

    /**
```

```
     * Construye la imagen a partir de sus datos en bytes. Los
     * mismos se obtienen de la imagen binarizada o en escala
     * de grises de la captura, según sea el caso
     * @param binarizada indica si la imagen se contruirá binarizada o no
     * @return la imagen contruída
     */
    private Image construirImagen(boolean binarizada){
        byte bytesImagen[];
        if(binarizada)
            bytesImagen = capturaDactilar.getBytesImagenBinarizada();
        else
            bytesImagen = capturaDactilar.getBytesImagen();
        byte bufferImagen[] = new byte[bytesImagen.length * 3];
        int desplazamiento = 0;
        for(int i = 0; i < bytesImagen.length; i++){
            bufferImagen[desplazamiento] = bytesImagen[i];
            bufferImagen[desplazamiento + 1] = bytesImagen[i];
            bufferImagen[desplazamiento + 2] = bytesImagen[i];
            desplazamiento += 3;
        }

        int ancho = capturaDactilar.getAnchoImagen();
        int altura = capturaDactilar.getAlturaImagen();
        BufferedImage bImagen = new BufferedImage(ancho, altura,
                BufferedImage.TYPE_INT_RGB);
        for(int y = 0; y < altura; y++){
            for(int x = 0; x < ancho; x++){
                int pix = bytesImagen[y * ancho + x] & 0xFF;
                int rgb = (pix << 16) | (pix << 8) | pix;
                bImagen.setRGB(x, y, rgb);
            }
        }
        Image imagen = bImagen;
        Graphics gImagen = bImagen.createGraphics();
        gImagen.drawImage(imagen, 0, 0, this);

        return imagen;
    }
}
```

Clase PasswordField

Clase correspondiente a un campo de texto monolínea, diseñado para la edición de contraseñas con límite de longitud.

```
package org.uaccess.utilidades.controles;

import org.uaccess.utilidades.DocumentoLimitado;

/**
 * Campo de texto diseñado para la edición de contraseñas con
 * límite de longitud
 * @author victor
 */
public class PasswordField extends javax.swing.JPasswordField{

    /**
     * Crea un nuevo campo de texto con soporte para la edición
     * de contraseñas
     */
    public PasswordField() {
        super();
    }

    /**
     * Establece la longitud máxima del campo de texto
     * @param length longitud máxima del texto
     */
    public void setLength(int length) {
        setDocument(new DocumentoLimitado(length));
    }
}
```

Clase TextArea

Esta clase define un campo de texto multilínea con soporte para definir longitud máxima.

```
package org.uaccess.utilidades.controles;

import org.uaccess.utilidades.DocumentoLimitado;

/**
 * Campo de texto multilínea con soporte para definir longitud máxima
 * @author victor
 */
public class TextArea extends javax.swing.JTextArea {

    /**
     * Crea un nuevo campo de texto multilínea
```

```
     */
    public TextArea() {
        super();
        setLineWrap(true);
    }

    /**
     * Selecciona la longitud máxima del texto contenido en el control
     * @param length longitud máxima del texto
     */
    public void setLength(int length) {
        setDocument(new DocumentoLimitado(length));
    }

    /**
     * Obtiene el texto contenido en el control
     * @return texto contenido en el control. Si está vacío
     * retorna 'null'
     */
    @Override
    public String getText() {
        String text = super.getText();
        return text.isEmpty() ? null : text;
    }
}
```

Clase TextField

Esta clase hereda de "JTextField" y corresponde a un campo de texto monolínea con soporte para definir la longitud máxima.

```
package org.uaccess.utilidades.controles;

import org.uaccess.utilidades.DocumentoLimitado;

/**
 * Campo de texto monolínea con soporte para definir la longitud máxima
 * @author victor
 */
public class TextField extends javax.swing.JTextField {

    /**
     * Establece la longitud máxima del texto contenido en el control
     * @param length longitud máxima del texto
     */
```

```
    public void setLength(int length) {
        setDocument(new DocumentoLimitado(length));
    }

    /**
     * Obtiene el texto contenido en el control
     * @return texto contenido en el control. Si está vacío
     * retorna 'null'
     */
    @Override
    public String getText() {
        String text = super.getText();
        return text.isEmpty() ? null : text;
    }
}
```

Validadores

El módulo de validadores corresponde a las clases contenidas en el paquete "org.uaccess.utilidades.validacion". Las mismas están destinadas a las tareas de validación de objetos previo a su inserción o actualización en la base de datos, así como a la validación de usuarios del sistema, según sus permisos administrativos, para ingresar a ciertos recursos del mismo.

Clases AdministradorDeMensajes

Esta clase es la implementación del "MessageResolver" de Oval para la aplicación, y se encarga de la administración de los mensajes para los diferentes tipos de violaciones encontradas por el validador.

```
package org.uaccess.utilidades.validacion;

/**
 * Administrador de los mensajes resultantes de la validación de oval
 * @author victor
 */
public class AdministradorDeMensajes implements net.sf.oval
        .localization.message.MessageResolver {

    /**
     * Obtiene el mensaje acorde al tipo de violación
     * @param key clave del tipo de violación
     * @return el mensaje para el tipo de violación especificado
     */
    @Override
    public String getMessage(String key) {
        String message = key;
        if (key.indexOf("NotNull") != -1) {
            message = "{context} no puede estar vacío";
        } else if (key.indexOf("NotEmpty") != -1) {
            message = "{context} no posee ningún valor válido";
        } else if (key.indexOf("NotBlank") != -1) {
            message = "{context} no puede estar en blanco";
        } else if (key.indexOf("NotNegative") != -1) {
            message = "{context} no puede ser negativo";
        } else if (key.indexOf("NotEqual") != -1) {
            message = "{context} no puede ser igual a {invalidValue}";
        } else if (key.indexOf("Size") != -1) {
            message = "El tamaño de {context} no es válido";
        } else if (key.indexOf("MaxLength") != -1) {
            message = "{context} es demasiado largo";
        } else if (key.indexOf("Min") != -1) {
            message = "{context} es inferior al mínimo permitido";
        } else if (key.indexOf("Max") != -1) {
            message = "{context} excede el máximo permitido";
        } else if (key.indexOf("MinSize") != -1) {
            message = "{context} no cumple con el tamaño mínimo";
        } else if (key.indexOf("MinLength") != -1) {
            message = "{context} es demasiado corto";
        } else if (key.indexOf("Range") != -1) {
            message = "{context} está fuera del rango permitido";
        } else if (key.indexOf("Unique") != -1) {
            message = "{context} ya existe";
        }
        return message;
    }
}
```

Clase RenderizadorDeContexto

Esta clase define el renderizador del mensaje en el que se muestran al usuario las violaciones encontradas.

```
package org.uaccess.utilidades.validacion;

import net.sf.oval.context.OValContext;
import net.sf.oval.localization.context.OValContextRenderer;

/**
 * Renderizador del mensaje en el que se muestran las
 * violaciones al usuario
 * @author victor
 */
public class RenderizadorDeContexto implements OValContextRenderer {

    /**
     * Da formato al mensaje en que se muestran las violaciones
     * del objeto verificado
     * @param context contexto de validación
     * @return el mensaje que informa sobre las validaciones
     * ya renderizado
     */
    @Override
    public String render(OValContext context) {
        String render = context.toString();
        int index = render.lastIndexOf(".");
        render = getCampoFormateado(render.substring(index + 1));
        String first = render.substring(0, 1);
        String upper = first.toUpperCase();
        render = "\"" + render.replaceFirst(first, upper) + "\"";
        return render;
    }

    /**
     * Obtiene un campo formateado según los casos definidos y
     * el campo tal como es recibido si para el mismo no
     * existe una regla de formato
     * @param nombreDelCampo nombre del campo que se desea formatear
     * @return campo formateado según su definición
     */
    private String getCampoFormateado(String nombreDelCampo){
        if(nombreDelCampo.equals("descripcion")){
            return "descripción";
        } else if(nombreDelCampo.equals("tipoDeDocumento")){
            return "tipo de documento";
```

```
        } else if(nombreDelCampo.equals("numeroDeDocumento")){
            return "número de documento";
        } else if(nombreDelCampo.equals("tipoDeUsuario")){
            return "tipo de usuario";
        } else if(nombreDelCampo.equals("contrasena")){
            return "contraseña";
        } else {
            return nombreDelCampo;
        }

    }
}
```

Clase Validador

Esta clase corresponde al validador de objetos; mismo que opera según las reglas de validación definidas.

```
package org.uaccess.utilidades.validacion;

import java.util.List;
import net.sf.oval.ConstraintViolation;
import net.sf.oval.Validator;
import net.sf.oval.configuration.xml.XMLConfigurer;

/**
 * Validador de objetos según las reglas de validación definidas
 * @author victor
 */
public class Validador extends Validator {

    /** Instancial del validador */
    private static Validador instancia;

    /**
     * Crea un nuevo validador de objetos
     */
    private Validador() {
        super();
        setMessageResolver(new AdministradorDeMensajes());
        setContextRenderer(new RenderizadorDeContexto());
    }

    /**
     * Obtiene la instancia del validador de objetos para la aplicación
```

```
     * @return instancia del validador
     */
    public static Validador getInstancia() {
        if (instancia == null) {
            instancia = new Validador();
        }
        return instancia;
    }

    /**
     * Valida el objeto recibido y, de existir, obtiene sus violaciones
     * @param objeto objeto a validar
     * @return Cadena de texto que contiene las violaciones encontradas.
     * Si no se encuentra ninguna se retorna una cadena vacía
     */
    public String obtenerViolaciones(Object objeto) {
        List<ConstraintViolation> violations = validate(objeto);
        StringBuilder message = new StringBuilder(200);
        if (violations.size() > 0) {
            for (ConstraintViolation v : violations) {
                message.append(v.getMessage()).append('\n');
            }
        }
        return message.toString();
    }

    /**
     * Registra un archivo XML que contiene reglas de validación
     * @param archivo ruta relativa del archivo XML
     * @return 'true' si el archivo se ha agregado correctamente,
     * caso contrario 'false'
     */
    public boolean agregarXML(String archivo) {
        XMLConfigurer configuradorXML = new XMLConfigurer();
        try {
            configuradorXML.fromXML(getClass().getResource(archivo)
                    .toURI().toURL().openStream());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return getConfigurers().add(configuradorXML);
    }
}
```

Clase ValidadorDeUsuarios

Clase correspondiente al validador de usuarios del sistema según sus permisos.

```
package org.uaccess.utilidades.validacion;

import org.uaccess.GestorDeDatosGlobales;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.Persona;
import org.uaccess.persistencia.PrivilegioAdministrativo;

/**
 * Validador de usuarios del sistema según sus permisos
 * @author victor
 */
public class ValidadorDeUsuarios {

    /** Instancia del validador de permisos de usuarios */
    private static ValidadorDeUsuarios instancia;

    /**
     * Crea un nuevo validador de usuarios
     */
    public ValidadorDeUsuarios() {
    }

    /**
     * Obtiene la instancia del validador de usuarios de la aplicación
     * @return instancia del validador de usuarios
     */
    public static ValidadorDeUsuarios getInstancia() {
        if(instancia==null)
            instancia = new ValidadorDeUsuarios();
        return instancia;
    }

    /**
     * Verifica si el usuario recibido posee asigando el
     * privilegio indicado
     * @param usuario usuario a verificar
     * @param nombrePrivilegio privilegio consultado
     * @return 'true' si el usuario posee asignado el privilegio,
     * caso contrario 'false'
     */
    public boolean verificarPermiso(Persona usuario,
            String nombrePrivilegio){
```

```
        PrivilegioAdministrativo privilegio =
                (PrivilegioAdministrativo) AdministradorPersistencia
                .getInstancia().obtenerObjeto("PrivilegioAdministrativo"
                + ".findByNombre", nombrePrivilegio);
        if(privilegio == null)
            return false;
        AdministradorPersistencia.getInstancia().refrescar(privilegio);
        if(!privilegio.getTipoDeUsuarioCollection().contains(usuario
                .getTipoDeUsuario()))
            return false;
        return true;
    }

    /**
     * Verifica si el usuario actual del sistema posee asignado
     * el privilegio indicado
     * @param nombrePrivilegio privilegio consultado
     * @return 'true' si el usuario posee asignado el privilegio,
     * caso contrario 'false'
     */
    public boolean verificarPermiso(String nombrePrivilegio){
        return verificarPermiso(GestorDeDatosGlobales.getInstacia()
                .getUsuario(), nombrePrivilegio);
    }
}
```

Clase VerificadorDeIntegridadReferencial

Esta clase define un validador que se encarga de velar por la integridad referencial de los datos.

```
package org.uaccess.utilidades.validacion;

import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.Huella;
import org.uaccess.persistencia.LoginAdministrativo;
import org.uaccess.persistencia.Persona;
import org.uaccess.persistencia.PrivilegioAdministrativo;
import org.uaccess.persistencia.TipoDeDocumento;
import org.uaccess.persistencia.TipoDeUsuario;
import org.uaccess.persistencia.Zona;

/**
 * Validador que vela por la integridad referencial de los datos
 * @author victor
```

```
 */
public class VerificadorDeIntegridadReferencial {

    /** Instancia del verificador de integridad referencial */
    private static VerificadorDeIntegridadReferencial instancia;

    /**
     * Crea un nuevo verificador de integridad referencial
     */
    private VerificadorDeIntegridadReferencial(){
    }

    /**
     * Obtiene la instancia del verificador de integridad referencial
     * de la aplicación
     * @return
     */
    public static VerificadorDeIntegridadReferencial getInstancia() {
        if(instancia == null){
            instancia = new VerificadorDeIntegridadReferencial();
        }
        return instancia;
    }

    /**
     * Verifica que la eliminación de un objeto no dañe la integridad
     * referencial de la Base de Datos
     * @param objeto objeto a ser analizado previo a su eliminación
     * @return 'true' si el objeto puede ser eliminido sin atentar
     * contra la integridad referencial; caso contrario, 'false'
     */
    public boolean verificarPrevioEliminacion(Object objeto){
        if(objeto.getClass() == Huella.class){
            return true;
        }
        if(objeto.getClass() == LoginAdministrativo.class){
            return true;
        }
        if(objeto.getClass() == Persona.class){
            String consulta = "SELECT a.id_zona, a.hora FROM "
                    + "bitacora_de_acceso a WHERE id_persona = "
                    + ((Persona)objeto).getId().toString();
            if(AdministradorPersistencia.getInstancia()
                    .ejecutarConsultaNativa(consulta,
                    new Object[]{}).size() < 1)
                return true;
        }
        if(objeto.getClass() == PrivilegioAdministrativo.class){
            AdministradorPersistencia.getInstancia()
                    .refrescar(objeto);
```

```
            if(((PrivilegioAdministrativo)objeto)
                    .getTipoDeUsuarioCollection().size() < 1)
                return true;
        }
        if(objeto.getClass() == TipoDeDocumento.class){
            AdministradorPersistencia.getInstancia()
                    .refrescar(objeto);
            if(((TipoDeDocumento)objeto).getPersonaCollection()
                    .size() < 1)
                return true;
        }
        if(objeto.getClass() == TipoDeUsuario.class){
            AdministradorPersistencia.getInstancia()
                    .refrescar(objeto);
            if(((TipoDeUsuario)objeto).getPersonaCollection()
                    .size() < 1)
                return true;
        }
        if(objeto.getClass() == Zona.class){
            AdministradorPersistencia.getInstancia()
                    .refrescar(objeto);
            if(((Zona)objeto).getTipoDeUsuarioCollection()
                    .size() < 1)
                return true;
        }
        return false;
    }
}
```

Utilidades varias

Este paquete corresponde al conjunto de utilitarios o clases multipropósito contenidas en "org.uaccess.utilidades".

Clase CargadorDeReportes

La clase "CargadorDeReportes" corresponde al tipo de objeto encargado de llenar y mostrar reportes de JasperReports.

```
package org.uaccess.utilidades;

import java.io.InputStream;
import java.sql.Connection;
import java.util.HashMap;
import java.util.Map;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.engine.util.JRLoader;
import net.sf.jasperreports.view.JasperViewer;

/**
 * Cargador de reportes de jasper reports
 * @author victor
 */
public class CargadorDeReportes {

    /** Conexión a la base de datos utilizada para poblar el reporte */
    private Connection conexion;
    /** Instancia del cargador de reportes */
    private static CargadorDeReportes instancia;

    /**
     * Establece la conexión a la base de datos utilizada para poblar
     * los reportes
     * @param conexionBD conexión a la base de datos
     */
    public void setConexion(Connection conexionBD) {
        this.conexion = conexionBD;
    }

    /**
     * Obtiene la conexión a la base de datos utilizada por el cargador
     * @return conexión a la base de datos
     */
    public Connection getConexion() {
        return conexion;
    }

    /**
     * Obtiene la instancia del cargador de reportes de la aplicación
     * @return instancia del cargador de reportes
     */
    public static CargadorDeReportes getInstancia() {
        if(instancia == null)
            instancia = new CargadorDeReportes();
        return instancia;
    }
```

```
    /**
     * Muestra un reporte en el entorno de escritorio y basado en
     * los parámetros
     * @param stream stream del fichero .jasper correspondiente al
     * reporte
     * @param nombresParametros nombres de los parámetros del
     * reporte a mostrar
     * @param valoresParametros valores de los parámetros con que
     * se cargará el reporte
     * @param titulo título del reporte a mostrar
     */
    public void mostrarReporte(InputStream stream,
            String[] nombresParametros, Object[] valoresParametros,
            String titulo) throws Exception{
        Map parametros = new HashMap();
        try {
            JasperReport reporteMaestro = (JasperReport) JRLoader
                    .loadObject(stream);
            for(int i=0; i<nombresParametros.length; i++)
                parametros.put(nombresParametros[i],
                        valoresParametros[i]);
            JasperPrint reporte = JasperFillManager
                    .fillReport(reporteMaestro, parametros, conexion);
            JasperViewer visor = new JasperViewer(reporte, false);
            visor.setTitle(titulo);
            visor.setVisible(true);
        } catch (Exception ex) {
            throw ex;
        }
    }
}
```

Clase DocumentoLimitado

"DocumentoLimitado" define un documento plano con límite de longitud.

```
package org.uaccess.utilidades;

import java.awt.Toolkit;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.PlainDocument;

/**
```

```
 * Documento con límite de longitud
 * @author victor
 */
public class DocumentoLimitado extends PlainDocument {

    /** Limite del documento */
    private int limite;

    /**
     * Crea un nuevo documento con límite de longitud
     * @param limite
     */
    public DocumentoLimitado(int limite) {
        this.limite = limite;
    }

    /**
     * Inserta una cadena de texto en el documento según es
     * especificado por los parámetros si es que ésta no excede
     * el límite de longitud
     * @param offset desplazamiento del inicio del texto
     * @param s cadena a insertar
     * @param a atributos para el contexto de inserción
     */
    @Override
    public void insertString(int offset, String s, AttributeSet a)
            throws BadLocationException {
        if (getLength() + s.length() <= limite) {
            super.insertString(offset, s, a);
        } else {
            Toolkit.getDefaultToolkit().beep();
        }
    }
}
```

Clase Encriptador

"Encriptador", tal y como sugiere su nombre, corresponde a un encriptador de cadenas de texto.

```
package org.uaccess.utilidades;

import java.math.BigInteger;
import java.security.MessageDigest;
```

```
/**
 * Encriptador de cadenas de texto
 * @author victor
 */
public class Encriptador {

    /**
     * Encripta una cadena de texto
     * @param algoritmo algoritmo a utilizar para la encriptación
     * @param texto texto que se desea cifrar
     * @return texto ya cifrado. Si se recibe una cadena vacía u
     * ocurre un error
     * en el proceso de filtrado, retornará 'null'
     */
    public static String cifrar(String algoritmo, String texto){
        if (texto == null) {
            return null;
        }
        try {
            final MessageDigest textoProcesado = MessageDigest
                    .getInstance(algoritmo);
            textoProcesado.update(texto.getBytes("UTF-8"));
            StringBuilder hash = new StringBuilder(new BigInteger(
                    1, textoProcesado.digest()).toString(16));
            while (hash.length() % 2 != 0) {
                hash.insert(0, "0");
            }
            return hash.toString();
        } catch (Exception ex) {
            return null;
        }
    }
}
```

Clase FormatoDeTablaPersona

Esta clase define el formato de la tabla que lista los objetos tipo "Persona" para GlazedLists.

```
package org.uaccess.utilidades;

import ca.odell.glazedlists.gui.AdvancedTableFormat;
import java.util.Comparator;
import org.uaccess.persistencia.Persona;
```

```
/**
 * Definidor de formato de la tabla 'personas' y su correspondiente
 * tipo de objeto: 'Persona'
 * @author victor
 */
public class FormatoDeTablaPersona implements
        AdvancedTableFormat<Persona>,
        ca.odell.glazedlists.gui.TableFormat<Persona> {

    /** Columnas de la tabla */
    String columnas[];

    /**
     * Crea un nuevo definidor de formato para la tabla 'personas'
     * @param columnas columnas de la tabla
     */
    public FormatoDeTablaPersona(String[] columnas) {
        this.columnas = columnas;
    }

    /**
     * Obtiene la clase correspondiente al tipo de objeto de
     * la columa especificada
     * @param i índice de la columna
     * @return la clase que define el tipo de objeto de la columna
     */
    public Class getColumnClass(int i) {
        if(i==5)
            return Boolean.class;
        return Object.class;
    }

    /**
     * Obtiene el comparador de objetos para la columa especificada
     * @param i índice de la columna
     * @return comparador de objetos
     */
    public Comparator getColumnComparator(int i) {
        return new org.apache.commons.collections.comparators
                .ComparableComparator();
    }

    /**
     * Obtiene la cantidad de columnas de la tabla
     * @return cantidad de columnas
     */
    public int getColumnCount() {
        return columnas.length;
    }
```

```
    /**
     * Obtiene el nombre de la columna especificada
     * @param i índice de la columna
     * @return nombre de la columna
     */
    public String getColumnName(int i) {
        return columnas[i];
    }

    /**
     * Obtiene el valor de la columna especificada para el
     * objeto recibido
     * @param p persona de la que se desea obtener el valor de
     * una columna
     * @param i índice de la columna que posee el valor deseado
     * @return el valor del campo
     */
    public Object getColumnValue(Persona p, int i) {
        switch (i) {
            case 0:
                return p.getApellidos();
            case 1:
                return p.getNombres();
            case 2:
                return p.getTipoDeDocumento();
            case 3:
                return p.getNumeroDeDocumento();
            case 4:
                return p.getTipoDeUsuario();
            case 5:
                return p.getActivo();
            default:
                return null;
        }
    }
}
```

Clase Mensajero

"Mensajero" es la clase que define al gestor de mensajes de usuario utilizado por la aplicación para informar de los diferentes sucesos.

```
package org.uaccess.utilidades;

import java.awt.Component;
import javax.swing.JOptionPane;

/**
 * Objeto encargado de mostrar al usuario los mensajes
 * desencadenados durante el uso de la aplicación
 * @author victor
 */
public class Mensajero {

    /** Instancia del mensajero de la aplicación */
    private static Mensajero instancia;
    /** Componennte padre de las ventanas de mensajes*/
    private Component padre;

    /** Crea un nuevo objeto mensajero */
    private Mensajero() {
    }

    /**
     * Obtiene la instancia del objeto mensajero de la aplicación
     * @return mensajero de la aplicación
     */
    public static Mensajero getInstancia() {
        if (instancia == null) {
            instancia = new Mensajero();
        }
        return instancia;
    }

    /**
     * Obtiene el componente padre para los mensajes mostrados
     * @return componente padre
     */
    public Component getPadre() {
        return padre;
    }

    /**
     * Establece el componente padre para los mensajes mostrados
     * @param padre the parent to set
     */
    public void setPadre(Component padre) {
        this.padre = padre;
    }

    /**
     * Muestra un mensaje de advertencia al usuario
```

```
     * @param advertencia advertencia a mostrar
     */
    public void mostrarAdvertencia(Object advertencia) {
        JOptionPane.showMessageDialog(padre, advertencia,
                "Advertencia", JOptionPane.WARNING_MESSAGE);
    }

    /**
     * Muestra un mensaje de información al usuario
     * @param informacion información a mostrar
     */
    public void mostrarInformacion(Object informacion) {
        JOptionPane.showMessageDialog(padre, informacion,
                "Información", JOptionPane.INFORMATION_MESSAGE);
    }

    /**
     * Muestra un mensaje de error al usuario
     * @param error mensaje de error a mostrar
     * @param excepcion excepción ocurrida
     */
    public void mostrarError(String error, Throwable excepcion) {
        JOptionPane.showMessageDialog(padre, error, "Error",
                JOptionPane.ERROR_MESSAGE);
    }

    /**
     * Muestra un diálogo de pregunta al usuario
     * @param pregunta pregunta a mostrar
     * @return 'true' si el usuario responde afirmativamente,
     * caso contrario 'false'
     */
    public boolean hacerPregunta(Object pregunta) {
        return (JOptionPane.showConfirmDialog(padre, pregunta,
                "Confirmación", JOptionPane.YES_NO_OPTION) ==
                JOptionPane.YES_OPTION) ? true : false;
    }
}
```

Persistencia de datos

El módulo de persistencia es el encargado del acceso a datos y la gestión de los mismos de una manera persistente para la aplicación.

Clase AdministradorPersistencia

Esta clase define el administrador de datos basado en el API de persistencia de java (JPA).

```
package org.uaccess.persistencia;

import java.sql.Connection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.persistence.Parameter;
import javax.persistence.Persistence;
import javax.persistence.Query;
import org.eclipse.persistence.config.QueryHints;

/**
 * Administrador de datos basado en el API de persistencia de java
 * @author victor
 */
public class AdministradorPersistencia {

    /** Instancia del administrador de persistencia de la aplicación */
    private static AdministradorPersistencia instancia;
    /** Administrador JPA de entidades */
    private EntityManager administradorDeEntidades;
    /** Conexión JDBC a la base de datos */
    private Connection conexion;

    /**
```

```
     * Crea un nuevo administrador de persistencia
     */
    private AdministradorPersistencia() {
        Map propiedades = new HashMap();
        administradorDeEntidades = Persistence
                .createEntityManagerFactory("UAccessPU",
                propiedades).createEntityManager();
        administradorDeEntidades.getTransaction().begin();
        conexion = administradorDeEntidades.unwrap(Connection.class);
        administradorDeEntidades.getTransaction().commit();
    }

    /**
     * Obtiene la instancia del administrador de persistencia
     * de la aplicación
     * @return el administrador de persistencia de la aplicación
     */
    public static AdministradorPersistencia getInstancia() {
        if (instancia == null) {
            instancia = new AdministradorPersistencia();
        }
        return instancia;
    }

    /**
     * Obtiene la conexión a la base de datos utilizada por el
     * administrador de persistencia
     * @return conexión JDBC utilizada por el administrador
     */
    public Connection getConexion() {
        return conexion;
    }

    /**
     * Obtiene la lista de objetos resultante de la consulta y
     * sus parámetros
     * @param nombreConsulta cadena de caracteres que identifica
     * a la consulta
     * @param parametros parámetros que complementan a la consulta
     * @return lista resultante de la ejecución de la consulta
     */
    public List obtenerLista(final String nombreConsulta,
            final Object... parametros) {
        Query consulta = crearConsulta(nombreConsulta);
        for (int i = 0; i < parametros.length; i++) {
            consulta.setParameter(((Parameter)consulta.getParameters()
                    .toArray()[i]).getName(), parametros[i]);
        }
        return consulta.getResultList();
    }
```

```
    /**
     * Obtiene un objeto individual resultante de la consulta y
     * sus parámetros
     * @param nombreConsulta cadena de caracteres que identifica a
     * la consulta
     * @param parametros parámetros que complementan a la consulta
     * @return objeto resultante de la ejecución de la consulta.
     * Si ocurre un error o ningún objeto satisface la consulta
     * retorna 'null'
     */
    public Object obtenerObjeto(final String nombreConsulta,
            final Object... parametros) {
        Query consulta = crearConsulta(nombreConsulta);
        for (int i = 0; i < parametros.length; i++) {
            consulta.setParameter(((Parameter)consulta.getParameters()
                    .toArray()[i]).getName(), parametros[i]);
        }
        try{
            return consulta.getSingleResult();
        } catch (Exception ex) {
            return null;
        }
    }

    /**
     * Crea la instancia de una consulta en base a su nombre
     * @param nombre nombre de la consulta
     * @return la instancia de la consulta
     */
    private Query crearConsulta(final String nombre) {
        Query consulta = administradorDeEntidades
                .createNamedQuery(nombre);
        consulta.setHint(QueryHints.RESULT_COLLECTION_TYPE,
                java.util.ArrayList.class);
        return consulta;
    }

    /**
     * Ejecuta una consulta nativa (consulta SQL)
     * @param consultaSql consulta nativa
     * @param parametros parámetros de la consulta
     * @return Lista de registros resultantes de la consulta
     */
    public List ejecutarConsultaNativa(String consultaSql,
            Object... parametros) {
        Query consulta = administradorDeEntidades
                .createNativeQuery(consultaSql);
        for (int i = 0; i < parametros.length; i++) {
            consulta.setParameter(((Parameter)consulta.getParameters()
```

```
                    .toArray()[i]).getName(), parametros[i]);
        }
        return consulta.getResultList();
    }

    /**
     * Inicia una transacción sobre los datos
     */
    public void iniciarTransaccion() {
        EntityTransaction transaccion = administradorDeEntidades
                .getTransaction();
        if (!transaccion.isActive()) {
            transaccion.begin();
        }
    }

    /**
     * Crea una instancia administrada y persistente del objeto
     * @param objeto objeto a persistir
     */
    private void persistir(Object objeto) {
        administradorDeEntidades.persist(objeto);
    }

    /**
     * Combina el estado del objeto en el contexto de
     * persistencia actual.
     * @param objeto objeto a fusionar
     */
    private void fusionar(Object objeto) {
        administradorDeEntidades.merge(objeto);
    }

    /**
     * Remueve un objeto del administrador de entidades
     * @param objeto objeto a remover
     */
    private void remover(Object objeto) {
        objeto = administradorDeEntidades.merge(objeto);
        administradorDeEntidades.remove(objeto);
    }

    /**
     * Refresca un objeto del administrador de entidades
     * @param objeto objeto a refrescar
     */
    public void refrescar(Object objeto) {
        administradorDeEntidades.refresh(objeto);
    }
```

```
    /**
     * Obtiene la transacción en proceso y hace un rollback
     * de la misma
     */
    public void deshacer() {
        EntityTransaction transaccion = administradorDeEntidades
                .getTransaction();
        if (transaccion.isActive()) {
            transaccion.rollback();
        }
    }

    /**
     * Ejecuta la transacción en proceso
     */
    public void ejecutar() {
        administradorDeEntidades.getTransaction().commit();
    }

    /**
     * Persiste un nuevo objeto y lo inserta en la base de datos
     * @param objeto objeto a insertar
     * @return 'true' si el objeto se ha insertado correctamente,
     * 'false' en caso contrario
     */
    public boolean insertar(Object objeto) {
        boolean insertado = false;
        try {
            iniciarTransaccion();
            persistir(objeto);
            ejecutar();
            insertado = true;
        } catch (Exception ex) {
            deshacer();
        }
        return insertado;
    }

    /**
     * Actualiza un objeto previamente existente
     * @param objeto objeto a actualizar
     * @return 'true' si el objeto se ha actualizado correctamente,
     * 'false' en caso contrario
     */
    public boolean actualizar(Object objeto) {
        boolean actualizado = false;
        try {
            iniciarTransaccion();
            fusionar(objeto);
            ejecutar();
```

```
            actualizado = true;
        } catch (Exception ex) {
            deshacer();
        }
        return actualizado;
    }

    /**
     * Elimina un objeto del administrador de entidades y su
     * respectivo registro en la base de datos
     * @param objeto objeto a eliminar
     * @return 'true' si el objeto se ha eliminado correctamente,
     * 'false' en caso contrario
     */
    public boolean eliminar(Object objeto) {
        boolean eliminado = false;
        try {
            iniciarTransaccion();
            remover(objeto);
            ejecutar();
            eliminado = true;
        } catch (Exception ex) {
            deshacer();
        }
        return eliminado;
    }

}
```

Clase Huella

Clase correspondiente a la entidad "Huella" del modelo de datos.

```
package org.uaccess.persistencia;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
```

```
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Huella' y su tabla respectiva en el
 * modelo de datos. Para mayor información sobre ésta, diríjase a la
 * documentación de la base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "huellas")
@NamedQueries({
    @NamedQuery(name = "Huella.findAll", query = "SELECT h "
    + "FROM Huella h"),
    @NamedQuery(name = "Huella.findByPersona", query = "SELECT h "
    + "FROM Huella h WHERE h.persona = :persona")})
public class Huella implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @JoinColumn(name = "id_persona", referencedColumnName = "id")
    private Persona persona;
    @Basic(optional = false)
    @Lob
    @Column(name = "data")
    private byte[] data;

    public Huella() {
    }

    public Huella(Persona persona) {
        this.persona = persona;
    }

    public Huella(Persona persona, byte[] data) {
        this.persona = persona;
        this.data = data;
    }

    public byte[] getData() {
        return data;
    }

    public void setData(byte[] data) {
        this.data = data;
    }

    public Persona getPersona() {
        return persona;
    }

    public void setPersona(Persona persona) {
```

```
        this.persona = persona;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (persona != null ? persona.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Huella)) {
            return false;
        }
        Huella other = (Huella) object;
        if ((this.persona == null && other.persona != null)
                || (this.persona != null
                && !this.persona.equals(other.persona))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "org.uaccess.persistencia.Huella[persona="
                + persona.toString() + "]";
    }
}
```

Clase LoginAdministrativo

Clase correspondiente a la entidad "Login administrativo" del modelo de datos.

```
package org.uaccess.persistencia;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.NamedQueries;
```

```
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Login administrativo' y su tabla
 * respectiva en el modelo de datos. Para mayor información sobre ésta,
 * diríjase a la documentación de la base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "logins_administrativos")
@NamedQueries({
    @NamedQuery(name = "LoginAdministrativo.findAll", query = "SELECT l"
    + " FROM LoginAdministrativo l"),
    @NamedQuery(name = "LoginAdministrativo.findByPersona", query
    = "SELECT l FROM LoginAdministrativo l WHERE l.persona = :persona"),
    @NamedQuery(name = "LoginAdministrativo.findByLogin", query
    = "SELECT l FROM LoginAdministrativo l WHERE l.login = :login"),
    @NamedQuery(name = "LoginAdministrativo.findByContrasena", query
    = "SELECT l FROM LoginAdministrativo l WHERE "
            + "l.contrasena = :contrasena"),
    @NamedQuery(name = "LoginAdministrativo.findByLoginAndContrasena",
    query = "SELECT l FROM LoginAdministrativo l WHERE "
            + "l.login = :login AND l.contrasena = :contrasena")})
public class LoginAdministrativo implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @JoinColumn(name = "id_persona", referencedColumnName = "id")
    private Persona persona;
    @Basic(optional = false)
    @Column(name = "login", unique = true)
    private String login;
    @Basic(optional = false)
    @Column(name = "contrasena")
    private String contrasena;

    public LoginAdministrativo() {
    }

    public LoginAdministrativo(Persona persona) {
        this.persona = persona;
    }

    public LoginAdministrativo(Persona persona, String login,
            String contrasena) {
        this.persona = persona;
        this.login = login;
        this.contrasena = contrasena;
    }
```

```
    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getContrasena() {
        return contrasena;
    }

    public void setContrasena(String contrasena) {
        this.contrasena = contrasena;
    }

    public Persona getPersona() {
        return persona;
    }

    public void setPersona(Persona persona) {
        this.persona = persona;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (persona != null ? persona.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof LoginAdministrativo)) {
            return false;
        }
        LoginAdministrativo other = (LoginAdministrativo) object;
        if ((this.persona == null && other.persona != null)
                || (this.persona != null &&
                !this.persona.equals(other.persona))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "org.uaccess.persistencia.LoginAdministrativo[persona="
                + persona.toString() + "]";
    }}
```

Clase Persona

Clase correspondiente a la entidad "Persona" del modelo de datos.

```
package org.uaccess.persistencia;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Persona' y su tabla respectiva en
 * el modelo de datos. Para mayor información sobre ésta, diríjase a
 * la documentación de la base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "personas")
@NamedQueries({
    @NamedQuery(name = "Persona.findAll", query =
    "SELECT p FROM Persona p"),
    @NamedQuery(name = "Persona.findById", query =
    "SELECT p FROM Persona p WHERE p.id = :id"),
    @NamedQuery(name = "Persona.findByApellidos", query =
    "SELECT p FROM Persona p WHERE p.apellidos = :apellidos"),
    @NamedQuery(name = "Persona.findByNombres", query =
    "SELECT p FROM Persona p WHERE p.nombres = :nombres"),
    @NamedQuery(name = "Persona.findByNumeroDeDocumento", query =
    "SELECT p FROM Persona p WHERE "
            + "p.numeroDeDocumento = :numeroDeDocumento"),
    @NamedQuery(name = "Persona.findByActivo", query =
    "SELECT p FROM Persona p WHERE p.activo = :activo"),
    @NamedQuery(name = "Persona.findByTipoYNumeroDeDocumento",
    query = "SELECT p FROM Persona p WHERE "
            + "p.tipoDeDocumento = :tipoDeDocumento AND "
            + "p.numeroDeDocumento = :numeroDeDocumento")})
```

```
public class Persona implements Serializable, Comparable<Persona> {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "apellidos")
    private String apellidos;
    @Basic(optional = false)
    @Column(name = "nombres")
    private String nombres;
    @Basic(optional = false)
    @Column(name = "numero_de_documento")
    private String numeroDeDocumento;
    @Basic(optional = false)
    @Column(name = "activo")
    private boolean activo;
    @JoinColumn(name = "id_tipo_de_documento",
    referencedColumnName = "id")
    @ManyToOne(optional = false, fetch = FetchType.LAZY)
    private TipoDeDocumento tipoDeDocumento;
    @JoinColumn(name = "id_tipo_de_usuario",
    referencedColumnName = "id")
    @ManyToOne(optional = false, fetch = FetchType.LAZY)
    private TipoDeUsuario tipoDeUsuario;

    public Persona() {
    }

    public Persona(Integer id) {
        this.id = id;
    }

    public Persona(Integer id, String apellidos, String nombres,
            String numeroDeDocumento, boolean activo) {
        this.id = id;
        this.apellidos = apellidos;
        this.nombres = nombres;
        this.numeroDeDocumento = numeroDeDocumento;
        this.activo = activo;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
```

```
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    public String getNombres() {
        return nombres;
    }

    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    public String getNumeroDeDocumento() {
        return numeroDeDocumento;
    }

    public void setNumeroDeDocumento(String numeroDeDocumento) {
        this.numeroDeDocumento = numeroDeDocumento;
    }

    public boolean getActivo() {
        return activo;
    }

    public void setActivo(boolean activo) {
        this.activo = activo;
    }

    public TipoDeDocumento getTipoDeDocumento() {
        return tipoDeDocumento;
    }

    public void setTipoDeDocumento(TipoDeDocumento tipoDeDocumento) {
        this.tipoDeDocumento = tipoDeDocumento;
    }

    public TipoDeUsuario getTipoDeUsuario() {
        return tipoDeUsuario;
    }

    public void setTipoDeUsuario(TipoDeUsuario tipoDeUsuario) {
        this.tipoDeUsuario = tipoDeUsuario;
    }
```

```
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Persona)) {
            return false;
        }
        Persona other = (Persona) object;
        if ((this.id == null && other.id != null) ||
                (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return apellidos + ", " + nombres;
    }

    public int compareTo(Persona p) {
        return toString().compareTo(p.toString());
    }
}
```

Clase PrivilegioAdministrativo

Clase correspondiente a la entidad "Privilegio administrativo" del modelo de datos.

```
package org.uaccess.persistencia;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Privilegio administrativo' y su tabla
 * respectiva en el modelo de datos. Para mayor información sobre ésta,
 * diríjase a la documentación de la base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "privilegios_administrativos")
@NamedQueries({
    @NamedQuery(name = "PrivilegioAdministrativo.findAll", query =
    "SELECT p FROM PrivilegioAdministrativo p"),
    @NamedQuery(name = "PrivilegioAdministrativo.findById", query =
    "SELECT p FROM PrivilegioAdministrativo p WHERE p.id = :id"),
    @NamedQuery(name = "PrivilegioAdministrativo.findByNombre", query =
    "SELECT p FROM PrivilegioAdministrativo p WHERE p.nombre = :nombre"),
    @NamedQuery(name = "PrivilegioAdministrativo.findByDescripcion",
    query = "SELECT p FROM PrivilegioAdministrativo p WHERE "
            + "p.descripcion = :descripcion")})
public class PrivilegioAdministrativo implements Serializable,
                  Comparable<PrivilegioAdministrativo> {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "nombre")
    private String nombre;
    @Basic(optional = false)
    @Column(name = "descripcion")
    private String descripcion;
    @ManyToMany(mappedBy="permisosAdministrativos",
    fetch = FetchType.LAZY)
    private Collection<TipoDeUsuario> tipoDeUsuarioCollection;

    public PrivilegioAdministrativo() {
    }

    public PrivilegioAdministrativo(Integer id) {
        this.id = id;
    }

    public PrivilegioAdministrativo(Integer id, String nombre,
```

```
            String descripcion) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public Collection<TipoDeUsuario> getTipoDeUsuarioCollection() {
        return tipoDeUsuarioCollection;
    }

    public void setTipoDeUsuarioCollection(
            Collection<TipoDeUsuario> tipoDeUsuarioCollection) {
        this.tipoDeUsuarioCollection = tipoDeUsuarioCollection;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof PrivilegioAdministrativo)) {
            return false;
```

```
        }
        PrivilegioAdministrativo other =
                (PrivilegioAdministrativo) object;
        if ((this.id == null && other.id != null) ||
                (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    public int compareTo(PrivilegioAdministrativo p) {
        return toString().compareTo(p.toString());
    }

    @Override
    public String toString() {
        return nombre;
    }
}
```

Clase TipoDeDocumento

Clase correspondiente a la entidad "Tipo de documento" del modelo de datos.

```
package org.uaccess.persistencia;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Tipo de documento' y su tabla
 * respectiva en el modelo de datos. Para mayor información sobre ésta,
```

```
 * diríjase a la documentación de la base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "tipos_de_documento")
@NamedQueries({
    @NamedQuery(name = "TipoDeDocumento.findAll", query =
    "SELECT t FROM TipoDeDocumento t"),
    @NamedQuery(name = "TipoDeDocumento.findById", query =
    "SELECT t FROM TipoDeDocumento t WHERE t.id = :id"),
    @NamedQuery(name = "TipoDeDocumento.findByNombre", query =
    "SELECT t FROM TipoDeDocumento t WHERE t.nombre = :nombre"),
    @NamedQuery(name = "TipoDeDocumento.findByDescripcion", query =
    "SELECT t FROM TipoDeDocumento t WHERE "
            + "t.descripcion = :descripcion"),
    @NamedQuery(name = "TipoDeDocumento.findByFormato", query =
    "SELECT t FROM TipoDeDocumento t WHERE t.formato = :formato")})
public class TipoDeDocumento implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "nombre", unique = true)
    private String nombre;
    @Basic(optional = false)
    @Column(name = "descripcion")
    private String descripcion;
    @Basic(optional = false)
    @Column(name = "formato")
    private String formato;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "tipoDeDocumento",
    fetch = FetchType.LAZY)
    private Collection<Persona> personaCollection;

    public TipoDeDocumento() {
    }

    public TipoDeDocumento(Integer id) {
        this.id = id;
    }

    public TipoDeDocumento(Integer id, String nombre,
            String descripcion, String formato) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.formato = formato;
```

```
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public String getFormato() {
        return formato;
    }

    public void setFormato(String formato) {
        this.formato = formato;
    }

    public Collection<Persona> getPersonaCollection() {
        return personaCollection;
    }

    public void setPersonaCollection(
            Collection<Persona> personaCollection) {
        this.personaCollection = personaCollection;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }
```

```
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof TipoDeDocumento)) {
            return false;
        }
        TipoDeDocumento other = (TipoDeDocumento) object;
        if ((this.id == null && other.id != null) ||
                (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return nombre;
    }
}
```

Clase TipoDeUsuario

Clase correspondiente a la entidad "Tipo de usuario" del modelo de datos.

```
import java.io.Serializable;
import java.util.Arrays;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Tipo de usuario' y su tabla
```

```
 * respectiva en el modelo de datos. Para mayor información sobre ésta,
 * diríjase a la documentación de la base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "tipos_de_usuario")
@NamedQueries({
    @NamedQuery(name = "TipoDeUsuario.findAll", query =
    "SELECT t FROM TipoDeUsuario t"),
    @NamedQuery(name = "TipoDeUsuario.findById", query =
    "SELECT t FROM TipoDeUsuario t WHERE t.id = :id"),
    @NamedQuery(name = "TipoDeUsuario.findByNombre", query =
    "SELECT t FROM TipoDeUsuario t WHERE t.nombre = :nombre"),
    @NamedQuery(name = "TipoDeUsuario.findByDescripcion", query =
    "SELECT t FROM TipoDeUsuario t WHERE "
            + "t.descripcion = :descripcion")})
public class TipoDeUsuario implements Serializable,
                    Comparable<TipoDeUsuario> {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "nombre", unique = true)
    private String nombre;
    @Basic(optional = false)
    @Column(name = "descripcion")
    private String descripcion;
    @ManyToMany
    @JoinTable(name="permisos_de_acceso",
        joinColumns={@JoinColumn(name="id_tipo_de_usuario")},
        inverseJoinColumns={@JoinColumn(name="id_zona")}
    )
    private Collection<Zona> zonasPermitidas;
    @ManyToMany
    @JoinTable(name="permisos_administrativos",
        joinColumns={@JoinColumn(name="id_tipo_de_usuario")},
        inverseJoinColumns={@JoinColumn(
                name="id_privilegio_administrativo")}
    )
    private Collection<PrivilegioAdministrativo> permisosAdministrativos;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "tipoDeUsuario",
    fetch = FetchType.LAZY)
    private Collection<Persona> personaCollection;

    public TipoDeUsuario() {
        this.zonasPermitidas = Arrays.asList(new Zona[0]);
        this.permisosAdministrativos = Arrays.asList(
```

```
                new PrivilegioAdministrativo[0]);
    }

    public TipoDeUsuario(Integer id) {
        this.id = id;
    }

    public TipoDeUsuario(Integer id, String nombre,
            String descripcion) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
    public Collection<Zona> getZonasPermitidas() {
        return zonasPermitidas;
    }

    public void setZonasPermitidas(
            Collection<Zona> zonasPermitidas) {
        this.zonasPermitidas = zonasPermitidas;
    }

    public void setZonasPermitidas(Zona[] zonasPermitidas) {
        limpiarZonasPermitidas();
        this.zonasPermitidas = Arrays.asList(zonasPermitidas);
    }
```

```
    public Collection<PrivilegioAdministrativo>
            getPermisosAdministrativos() {
        return permisosAdministrativos;
    }

    public void setPermisosAdministrativos(
            Collection<PrivilegioAdministrativo>
            permisosAdministrativos) {
        this.permisosAdministrativos = permisosAdministrativos;
    }

    public void setPermisosAdministrativos(
            PrivilegioAdministrativo[] permisosAdministrativos) {
        limpiarPermisosAdministrativos();
        this.permisosAdministrativos = Arrays.asList(
                permisosAdministrativos);
    }

    public Collection<Persona> getPersonaCollection() {
        return personaCollection;
    }

    public void setPersonaCollection(
            Collection<Persona> personaCollection) {
        this.personaCollection = personaCollection;
    }

        private void limpiarZonasPermitidas(){
        try{
            this.zonasPermitidas.clear();
        }catch(Exception ex){
        }
    }

    private void limpiarPermisosAdministrativos(){
        try{
            this.permisosAdministrativos.clear();
        }catch(Exception ex){
        }
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
```

```
    public boolean equals(Object object) {
        if (!(object instanceof TipoDeUsuario)) {
            return false;
        }
        TipoDeUsuario other = (TipoDeUsuario) object;
        if ((this.id == null && other.id != null) ||
                (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return nombre;
    }

    public int compareTo(TipoDeUsuario t) {
        return toString().compareTo(t.toString());
    }
}
```

Clase Zona

Clase correspondiente a la entidad "Zona" del modelo de datos.

```
package org.uaccess.persistencia;

import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * Objeto que define la entidad 'Zona' y su tabla respectiva
```

```
 * en el modelo de datos. Para mayor información sobre ésta, diríjase
 * a la documentación de la
 * base de datos del sistema
 * @author victor
 */
@Entity
@Table(name = "zonas")
@NamedQueries({
    @NamedQuery(name = "Zona.findAll", query =
    "SELECT z FROM Zona z"),
    @NamedQuery(name = "Zona.findById", query =
    "SELECT z FROM Zona z WHERE z.id = :id"),
    @NamedQuery(name = "Zona.findByNombre", query =
    "SELECT z FROM Zona z WHERE z.nombre = :nombre"),
    @NamedQuery(name = "Zona.findByDescripcion", query =
    "SELECT z FROM Zona z WHERE z.descripcion = :descripcion")})
public class Zona implements Serializable, Comparable<Zona> {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "nombre")
    private String nombre;
    @Basic(optional = false)
    @Column(name = "descripcion")
    private String descripcion;
    @ManyToMany(mappedBy="zonasPermitidas", fetch = FetchType.LAZY)
    private Collection<TipoDeUsuario> tipoDeUsuarioCollection;

    public Zona() {
    }

    public Zona(Integer id) {
        this.id = id;
    }

    public Zona(Integer id, String nombre, String descripcion) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
```

```
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public Collection<TipoDeUsuario> getTipoDeUsuarioCollection() {
        return tipoDeUsuarioCollection;
    }

    public void setTipoDeUsuarioCollection(
            Collection<TipoDeUsuario> tipoDeUsuarioCollection) {
        this.tipoDeUsuarioCollection = tipoDeUsuarioCollection;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Zona)) {
            return false;
        }
        Zona other = (Zona) object;
        if ((this.id == null && other.id != null) ||
                (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    public int compareTo(Zona z) {
        return toString().compareTo(z.toString());
```

```
    }

    @Override
    public String toString() {
        return nombre;
    }
}
```

Módulo de biometría

Este paquete contiene las clases encargadas de la gestión de la adquisición y procesamiento de las capturas de huellas dactilares.

Clase CapturaDactilar

La clase "CapturaDactilar" corresponde al resultado de la captura de una huella dactilar.

```
package org.uaccess.fprint;

import com.sun.jna.Pointer;

/**
 * Captura de una huella dactilar
 * @author victor
 */
public class CapturaDactilar {

    /** Puntero a los datos de la captura realizada */
    private Pointer pDatos;
    /** Puntero a la imagen en escala de grises de la huella */
    private Pointer pImagen;
    /** Puntero a la imagen binarizada de la huella */
    private Pointer pBinarizada;
```

```
    /**
     * Crea un nuevo objeto a partir de los parámetros recibidos
     * @param pDatos puntero a los datos de la captura
     * @param pImagen puntero a la imagen de la huella
     * @param pBinarizada puntero a la imagen binarizada de la huella
     */
    public CapturaDactilar(Pointer pDatos, Pointer pImagen,
            Pointer pBinarizada) {
        this.pDatos = pDatos;
        this.pImagen = pImagen;
        this.pBinarizada = pBinarizada;
    }

    /**
     * Obtiene el puntero a los datos de la captura dactilar
     * @return puntero a los datos de la captura
     */
    public Pointer getPDatos() {
        return pDatos;
    }

    /**
     * Obtiene el puntero a la imagen en escala de grises de la
     * huella capturada
     * @return puntero a la imagen de la huella
     */
    public Pointer getPImagen() {
        return pImagen;
    }

    /**
     * Obtiene el puntero a la imagen binarizada de la huella capturada
     * @return puntero a la imagen de la huella
     */
    public Pointer getPBinarizada() {
        return pBinarizada;
    }

    /**
     * Establece los punteros correspondientes a la captura dactilar
     * @param pDatos puntero a los datos
     * @param pImagen puntero a la imagen
     * @param pBinarizada puntero a la imagen binarizada
     */
    public void setPunteros(Pointer pDatos, Pointer pImagen,
            Pointer pBinarizada){
        this.pDatos = pDatos;
        this.pImagen = pImagen;
        this.pBinarizada = pBinarizada;
    }
```

```
    /**
     * Obtiene el ancho de la imagen
     * @return ancho de la imagen en pixeles
     */
    public int getAnchoImagen(){
        return InterfazFprintAPI.getInstancia()
                .ObtenerAnchoImagen(pImagen);
    }

    /**
     * Obtiene la altura de la imagen
     * @return altura de la imagen en pixeles
     */
    public int getAlturaImagen(){
        return InterfazFprintAPI.getInstancia()
                .ObtenerAlturaImagen(pImagen);
    }

    /**
     * Obtiene los bytes correspondientes a la imagen original
     * (escala de grises)
     * @return imagen original en bytes
     */
    public byte[] getBytesImagen(){
        return InterfazFprintAPI.getInstancia()
                .ObtenerBytesImagen(pImagen);
    }

    /**
     * Obtiene los bytes correspondientes a la imagen binarizada
     * @return imagen binarizada en bytes
     */
    public byte[] getBytesImagenBinarizada(){
        return InterfazFprintAPI.getInstancia()
                .ObtenerBytesImagen(pBinarizada);
    }

    /**
     * Obtiene los bytes correspondientes a los datos resultantes
     * de la captura
     * @return datos de la captura dactilar en bytes
     */
    public byte[] getBytesDatos(){
        return InterfazFprintAPI.getInstancia()
                .ObtenerDatosCaptura(pDatos);
    }
}
```

Clase Dispositivo

Esta clase define un dispositivo de captura dactilar o escáner de huellas dactilares.

```
package org.uaccess.fprint;

import com.sun.jna.Pointer;

/**
 * Dispositivo lector de huellas dactilares
 * @author victor
 */
public class Dispositivo {
    /** puntero al dispositivo */
    private Pointer puntero;
    /** nombre del dispositivo */
    private String nombre;
    /** nombre del controlador del dispositivo */
    private String nombreControlador;

    /**
     * Crea un nuevo dispositivo a partir del puntero hacia el mismo
     * @param pDispositivo puntero al dispositivo
     */
    public Dispositivo(Pointer pDispositivo){
        this.puntero = pDispositivo;
        this.nombre = InterfazFprintAPI.getInstancia()
                .ObtenerNombreDispositivo(pDispositivo);
        this.nombreControlador = InterfazFprintAPI.getInstancia()
                .ObtenerNombreControlador(pDispositivo);
    }

    /**
     * Obtiene el puntero al dispositivo de captura
     * @return puntero al dispositivo
     */
    public Pointer getPuntero() {
        return puntero;
    }

    /**
     * Obtiene el nombre del dispositivo de captura
     * @return nombre del dispositivo
     */
    public String getNombre() {
        return nombre;
```

```
    }

    /**
     * Obtiene el nombre del controlador del dispositivo de captura
     * @return nombre del controlador del dispositivo
     */
    public String getNombreControlador() {
        return nombreControlador;
    }

    @Override
    public String toString() {
        return nombre;
    }
}
```

Clase InterfazFprintAPI

La clase "InterfazFprintAPI" es la encargada de crear un vínculo entre la aplicación administrativa y la librería nativa de fines biométricos, libfprint. Dicho funcionamiento está enmarcado por la librería de Acceso Nativo de Java (JNA, por sus siglas en inglés).

```
package org.uaccess.fprint;

import com.sun.jna.Library;
import com.sun.jna.Native;
import com.sun.jna.Pointer;
import com.sun.jna.ptr.PointerByReference;

/**
 * Interfaz entre el API de fprint y la aplicación
 * @author victor
 */
public class InterfazFprintAPI {

    /** Instancia de la interfaz */
    private static InterfazFprintAPI instacia;
    /** Bandera que indica si el API ya ha sido inicializado */
    private boolean apiInicializado;
```

```
    /** Vínculo de acceso a la librería nativa de fprint */
    private FPrint fprint;

    private interface FPrint extends Library {

        public int fp_init();

        public void fp_exit();

        public Pointer[] fp_discover_devs();

        public Pointer fp_dscv_dev_get_driver(Pointer dev);

        public String fp_driver_get_name(Pointer drv);

        public String fp_driver_get_full_name(Pointer drv);

        public Pointer fp_dev_open(Pointer ddev);

        public void fp_dev_close(Pointer dev);

        public int fp_enroll_finger_img(Pointer dev,
                PointerByReference print_data, PointerByReference img);

        public Pointer fp_img_get_data(Pointer img);

        public int fp_img_get_height(Pointer img);

        public int fp_img_get_width(Pointer img);

        public void fp_img_standardize(Pointer img);

        public Pointer fp_img_binarize(Pointer img);

        public int fp_print_data_get_data(Pointer data,
                PointerByReference ret);

    }

    /**
     * Crea una interfaz para fprint
     */
    private InterfazFprintAPI(){
        fprint = (FPrint) Native.loadLibrary("fprint", FPrint.class);
        if(fprint.fp_init() == 0)
            apiInicializado = true;
        else
            apiInicializado = false;
    }
```

```
    /**
     * Obtiene la instancia de la interfaz de acceso a fprint
     * @return instancia de la interfaz
     */
    public static InterfazFprintAPI getInstancia() {
        if (instacia == null) {
            instacia = new InterfazFprintAPI();
        }
        return instacia;
    }

    /**
     * Ubica los dispositivos de captura dactilar disponibles
     * @return dispositivos disponibles
     */
    public Dispositivo[] DescubrirDispositivos() throws Exception {
        if(apiInicializado){
            Pointer[] pDispositivos = fprint.fp_discover_devs();
            Dispositivo dispositivos[] =
                    new Dispositivo[pDispositivos.length];
            for(int i = 0; i < dispositivos.length; i++)
                dispositivos[i] = new Dispositivo(pDispositivos[i]);
            return dispositivos;
        }
        else
            throw new Exception(
                    "El API de fprint aún no ha sido inicializado");
    }

    /**
     * Obtiene el nombre del dispositivo indicado
     * @param pDispositivo puntero al dispositivo
     * @return nombre del dispositivo
     */
    public String ObtenerNombreDispositivo(Pointer pDispositivo){
        return fprint.fp_driver_get_full_name(
                fprint.fp_dscv_dev_get_driver(pDispositivo));
    }

    /**
     * Obtiene el nombre del controlador del dispositivo indicado
     * @param pDispositivo puntero al dispositivo
     * @return nombre del controlador del dispositivo
     */
    public String ObtenerNombreControlador(Pointer pDispositivo){
        return fprint.fp_driver_get_name(
                fprint.fp_dscv_dev_get_driver(pDispositivo));
    }

    /**
```

```
     * Obtiene el ancho de la imagen indicada
     * @param pImagen puntero a la imagen
     * @return ancho de la imagen en pixeles
     */
    public int ObtenerAnchoImagen(Pointer pImagen){
        return fprint.fp_img_get_width(pImagen);
    }

    /**
     * Obtiene la altura de la imagen indicada
     * @param pImagen puntero a la imagen
     * @return altura de la imagen en pixeles
     */
    public int ObtenerAlturaImagen(Pointer pImagen){
        return fprint.fp_img_get_height(pImagen);
    }

    /**
     * Obtiene los bytes correspondientes a la imagen indicada
     * @param pImagen puntero a la imagen
     * @return bytes de la imagen
     */
    public byte[] ObtenerBytesImagen(Pointer pImagen){
        Pointer imgData = fprint.fp_img_get_data(pImagen);
        byte[] imgBytes = imgData.getByteArray(
                0, ObtenerAnchoImagen(pImagen)
                * ObtenerAlturaImagen(pImagen));
        return imgBytes;
    }

    /**
     * Ejecuta la binarización de la imagen recibida
     * @param pImagen puntero a la imagen a binarizar
     * @return puntero a la imagen binarizada
     */
    public Pointer BinarizarImagen(Pointer pImagen){
        return fprint.fp_img_binarize(pImagen);
    }

    /**
     * Obtiene los bytes correspondientes a los datos de la
     * captura dactilar recibida
     * @param pDatosCaptura puntero a los datos de la captura
     * @return bytes correspondientes a los datos
     */
    public byte[] ObtenerDatosCaptura(Pointer pDatosCaptura){
        PointerByReference rDatosCaptura = new PointerByReference();
        int tamCamptura = fprint.fp_print_data_get_data(pDatosCaptura,
                rDatosCaptura);
        byte[] bytesCaptura = rDatosCaptura.getValue().getByteArray(0,
```

```
                tamCamptura);
        return bytesCaptura;
    }

    /**
     * Ejecuta la captura de una huella dactilar desde el
     * dispositivo indicado
     * @param lector dispositivo lector de huellas
     * @return captura dactilar resultante
     */
    public CapturaDactilar CapturarHuella(Dispositivo lector)
            throws Exception{
        if(!apiInicializado){
            throw new Exception(
                    "El API de fprint aún no ha sido inicializado");
        }
        Pointer disp;
        PointerByReference rDatos = new PointerByReference();
        PointerByReference rImagen = new PointerByReference();
        Pointer pBinarizada = Pointer.NULL;
        CapturaDactilar captura = null;
        disp = fprint.fp_dev_open(lector.getPuntero());
        if (fprint.fp_enroll_finger_img(disp, rDatos, rImagen) == 1){
            fprint.fp_img_standardize(rImagen.getValue());
            pBinarizada = fprint.fp_img_binarize(rImagen.getValue());
            captura = new CapturaDactilar(rDatos.getValue(),
                    rImagen.getValue(), pBinarizada);
        }
        fprint.fp_dev_close(disp);
        return captura;
    }
}
```

Interfaz de usuario

El módulo de interfaz de usuario es el que posee los elementos de interfaz gráfica de la aplicación. Los mismos se encuentran en el paquete “org.uaccess.gui”.

Clase DialogoCapturarHuella

"DialogoCapturarHuella" define una clase extendida de "DialogoAgregarOEditarObjeto", diseñada para la captura de huellas dactilares en modo diálogo.

```
package org.uaccess.gui;

import java.net.URL;
import org.uaccess.fprint.CapturaDactilar;
import org.uaccess.utilidades.controles.DialogoAgregarOEditarObjeto;

/**
 * Ventana tipo diálogo utilizada para la captura de huellas dactilares
 * @author victor
 */
public class DialogoCapturarHuella
        extends DialogoAgregarOEditarObjeto {

    /**
     * Crea un nuevo diólgo con soporte para captura de
     * huellas dactilares
     * @param icono URL del ícono del diálogo
     */
    public DialogoCapturarHuella(URL icono) {
        super(new JpnCapturarHuella(), "Capturar huella", icono);
    }

    /**
     * Muestra el diálogo al usuario
     * @param c captura dactilar a mostrar en el panel contenido
     * @return captura dactilar contenida al momento de cerrar el
     * diálogo
     */
    public CapturaDactilar mostrar(CapturaDactilar c) {
        CapturaDactilar captura;
        if(super.mostrar())
            captura = ((JpnCapturarHuella)panel).obtenerCaptura();
        else
            captura = null;
        return captura;
    }
}
```

Clase DialogoLoginAdministrativo

"DialogoLoginAdministrativo" define una clase extendida de "DialogoAgregarOEditarObjeto", diseñada para la edición de logins administrativos en modo diálogo.

```
package org.uaccess.gui;

import java.net.URL;
import org.uaccess.persistencia.LoginAdministrativo;
import org.uaccess.utilidades.controles.DialogoAgregarOEditarObjeto;

/**
 * Ventana tipo diálogo utilizada para la edición de
 * logins administrativos
 * @author victor
 */
public class DialogoLoginAdministrativo
        extends DialogoAgregarOEditarObjeto {

    /**
     * Crea un nuevo diálogo con soporte para la edición de
     * logins administrativos
     * @param icono URL del ícono del diálogo
     */
    public DialogoLoginAdministrativo(URL icono) {
        super(new JpnLoginAdministrativo(),
                "Login administrativo", icono);
    }

    /**
     * Muestra la ventana y como valor inicial de sus controles
     * el login recibido
     * @param login login recibido
     * @return login al momento de cerrar el diálogo
     */
    public LoginAdministrativo mostrar(LoginAdministrativo login) {
        ((JpnLoginAdministrativo)panel).setDatos(login);
        if(super.mostrar())
            login = ((JpnLoginAdministrativo)panel).getDatos();
        return login;
    }
}
```

Clase JpnCapturarHuella

Esta clase corresponde al panel diseñado para la captura de huellas dactilares.

```
package org.uaccess.gui;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import org.uaccess.fprint.CapturaDactilar;
import org.uaccess.fprint.Dispositivo;
import org.uaccess.fprint.InterfazFprintAPI;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel diseñado para la captura de huellas dactilares
 * @author victor
 */
public class JpnCapturarHuella extends javax.swing.JPanel {

    /**
     * Crea un nuevo panel para captura de huellas
     */
    public JpnCapturarHuella() {
        initComponents();
        poblarDispositivos();
    }

    /**
     * Obtiene el botón por defecto del panel
     * @return botón por defecto
     */
    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    /**
     * Muestra los dispositivos de captura disponibles en la
     * interfaz de usuario
     */
    private void poblarDispositivos(){
        try {
            DefaultComboBoxModel modeloDispositivos =
                    new DefaultComboBoxModel(InterfazFprintAPI
```

```
                .getInstancia().DescubrirDispositivos());
            jcbDispositivos.setModel(modeloDispositivos);
        } catch (Exception ex) {
            Mensajero.getInstancia().mostrarError(ex.getMessage(), ex);
        }
    }

    /**
     * Ejecuta la captura dactilar desde el dispositivo seleccionado
     */
    private void ejecutarCaptura(){
        if(jcbDispositivos.getSelectedItem() == null){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "No hay ningún dispositivo disponible");
            return;
        }
        try {
            pvhVisorHuella.setCaptura(
                    InterfazFprintAPI.getInstancia().CapturarHuella(
                    (Dispositivo) jcbDispositivos.getSelectedItem()));
        } catch (Exception ex) {
            Mensajero.getInstancia().mostrarError(ex.getMessage(), ex);
        }
    }

    /**
     * Obtiene la captura dactilar resultante de la ejecución de la
     * captura y correspondiente a la imagen del panel visor de la huella
     * @return captura dactilar resultante
     */
    public CapturaDactilar obtenerCaptura(){
        return pvhVisorHuella.getCaptura();
    }

    /**
     * Acepta la captura realizada. De haberse realizado
     * satisfactoriamente informa a sus escuchantes y cierra la ventana
     */
    private void aceptar(){
        if(pvhVisorHuella.getCaptura() == null){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "No hay ninguna captura dactilar por guardar");
            return;
        }
        firePropertyChange("operacionExitosa", true, null);
    }

    /**
     * Cancela el proceso de captura de la huella e informa a sus
     * escuchantes para que la ventana sea cerrada
```

```
     */
    private void cancelar(){
        firePropertyChange("operacionCancelada", true, null);
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtRefrescarDispositivosActionPerformed(
            java.awt.event.ActionEvent evt) {
        poblarDispositivos();
    }

    private void jbtEjecutarCapturaActionPerformed(
            java.awt.event.ActionEvent evt) {
        ejecutarCaptura();
    }

    private void jrbEscalaDeGrisesActionPerformed(
            java.awt.event.ActionEvent evt) {
        pvhVisorHuella.setBinarizada(false);
    }

    private void jrbBinarizadaActionPerformed(
            java.awt.event.ActionEvent evt) {
        pvhVisorHuella.setBinarizada(true);
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        aceptar();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        cancelar();
    }


    // Variables declaration - do not modify
    private javax.swing.ButtonGroup bgrModoVista;
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JButton jbtEjecutarCaptura;
    private javax.swing.JButton jbtRefrescarDispositivos;
    private javax.swing.JComboBox jcbDispositivos;
    private javax.swing.JLabel jlbDispositivo;
    private javax.swing.JLabel jlbModoVista;
    private javax.swing.JRadioButton jrbBinarizada;
```

```
    private javax.swing.JRadioButton jrbEscalaDeGrises;
    private org.uaccess.utilidades.controles
            .PanelVisorHuella pvhVisorHuella;
    // End of variables declaration
}
```

Clase JpnIngresar

Esta clase define el panel diseñado para el ingreso de usuarios a la aplicación.

```
package org.uaccess.gui;

import javax.swing.JButton;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.LoginAdministrativo;
import org.uaccess.persistencia.Persona;
import org.uaccess.utilidades.Encriptador;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel diseñado para la autenticación de usuarios que desean
 * ingresar al sistema
 * @author victor
 */
public class JpnIngresar extends javax.swing.JPanel {

    /** Persona que está ingresando al sistema */
    private Persona usuario;

    /**
     * Crea un nuevo panel para el ingreso de usuarios
     */
    public JpnIngresar() {
        initComponents();
        pimImagen.setImagen(getClass().getResource(
                "/org/uaccess/recursos/imagenes/banner.png"));
        usuario = null;
    }

    /**
     * Obtiene el botón por defecto del panel
     * @return botón por defecto
     */
    public JButton getBotonPorDefecto(){
```

```
        return jbtAceptar;
    }

    /**
     * Obtiene el usuario que está ingresando al sistema por
     * medio del panel
     * @return usuario que está ingresando al sistema
     */
    public Persona getUsuario() {
        return usuario;
    }

    /**
     * Valida que los datos hayan sido ingresados correctamente
     * @return 'true' si los datos se han ingresado de manera
     * correcta, 'false' en caso contrario
     */
    private boolean validar(){
        if(jtfUsuario.getText() == null || String.valueOf(
                jtfContrasena.getPassword()).equals("")){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe introducir el nombre de usuario"
                    + " y la contraseña");
            return false;
        }
        return true;
    }

    /**
     * Ejecuta la autenticación del usuario según los datos
     * ingresados en el panel
     */
    private void autenticar(){
        if(!validar())
            return;
        String consulta =
                "LoginAdministrativo.findByLoginAndContrasena";
        Object parametros[] = new Object[]{jtfUsuario.getText(),
        Encriptador.cifrar("MD5", String.valueOf(
                jtfContrasena.getPassword()))};
        LoginAdministrativo login = (LoginAdministrativo)
                AdministradorPersistencia.getInstancia()
                .obtenerObjeto(consulta, parametros);
        if(login != null){
            usuario = login.getPersona();
            cerrar();
        }
        else{
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Usuario no válido");
```

```
        }
    }

    /**
     * Cancela el proceso de autenticación cerrando la ventana y
     * finalizando la aplicación
     */
    private void cancelar(){
        cerrar();
    }

    /**
     * Emite la orden para cerrar la ventana contenedora
     */
    private void cerrar(){
        firePropertyChange("close", true, null);
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        autenticar();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        cancelar();
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbContrasena;
    private javax.swing.JLabel jlbUsuario;
    private org.uaccess.utilidades.controles
            .PasswordField jtfContrasena;
    private org.uaccess.utilidades.controles.TextField jtfUsuario;
    private org.uaccess.utilidades.controles.PanelImagen pimImagen;
    // End of variables declaration

}
```

Clase JpnLoginAdministrativo

La clase "JpnLoginAdministrativo" corresponde al panel diseñado para la edición de los logins administrativos asignados a las personas.

```
package org.uaccess.gui;

import java.util.Arrays;
import javax.swing.JButton;
import org.uaccess.persistencia.LoginAdministrativo;
import org.uaccess.utilidades.Encriptador;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel diseñado para la edición de los logins administrativos
 * asignados a las personas
 * @author victor
 */
public class JpnLoginAdministrativo extends javax.swing.JPanel {

    /**
     * Crea un nuevo panel para la edición de logins administrativos
     */
    public JpnLoginAdministrativo() {
        initComponents();
    }

    /**
     * Obtiene el botón por defecto del panel
     * @return botón por defecto
     */
    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    /**
     * Establece los datos del login a partir del objeto recibido
     * @param login login a mostrar en el panel
     */
    public void setDatos(LoginAdministrativo login){
        if(login != null){
            jtfLogin.setText(login.getLogin());
            jtfContrasena.setText("*******");
            jtfConfContrasena.setText("*******");
        }
```

```
    }

    /**
     * Obtiene los datos del login del panel
     * @return login actual del panel
     */
    public LoginAdministrativo getDatos(){
        try{
            return new LoginAdministrativo(null, jtfLogin.getText(),
                    Encriptador.cifrar("MD5", String.valueOf(
                    jtfContrasena.getPassword())));
        }catch (Exception ex){
            return null;
        }
    }

    /**
     * Verifica que los datos del login se hayan
     * ingresado correctamente
     * @return 'true' si los datos se han ingresado correctamente,
     * 'false' si no
     */
    private boolean validar(){
        if(jtfLogin.getText() == null){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "No ha indicado el nombre para inicio de sesión");
            return false;
        }
        if(jtfContrasena.getPassword().length < 1){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe ingresar la contraseña");
            return false;
        }
        if(jtfConfContrasena.getPassword().length < 1){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe ingresar la confirmación de la contraseña");
            return false;
        }
        if(!Arrays.equals(jtfContrasena.getPassword(),
                jtfConfContrasena.getPassword())){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "La contraseña y su confirmación deben coincidir");
            return false;
        }
        return true;
    }

    /**
     * Acepta la operación de edición del login administrativo
     */
```

```
    private void aceptar(){
        if(validar())
            firePropertyChange("operacionExitosa", true, null);
    }

    /**
     * Cancela la operación de edición y lo hace saber a su escuchante
     * para que la ventana sea cerrada
     */
    private void cancelar(){
        firePropertyChange("operacionCancelada", true, null);
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        aceptar();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        cancelar();
    }

    // Variables declaration - do not modify
    private javax.swing.ButtonGroup bgrModoVista;
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbContrasena;
    private javax.swing.JLabel jlbContrasena1;
    private javax.swing.JLabel jlbLogin;
    private org.uaccess.utilidades.controles
            .PasswordField jtfConfContrasena;
    private org.uaccess.utilidades.controles
            .PasswordField jtfContrasena;
    private org.uaccess.utilidades.controles.TextField jtfLogin;
    // End of variables declaration
}
```

Clase JpnParametrosParaBitacoraDeAcceso

Esta clase corresponde al formulario utilizado para seleccionar los parámetros para el reporte que muestra la bitácora de acceso peatonal.

```
package org.uaccess.gui;

import java.io.InputStream;
import java.sql.Timestamp;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Vector;
import javax.swing.JButton;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.Persona;
import org.uaccess.persistencia.TipoDeUsuario;
import org.uaccess.persistencia.Zona;
import org.uaccess.utilidades.CargadorDeReportes;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.controles.PanelParametrosParaReporte;
import org.uaccess.utilidades.controles.PanelSelectorDeObjetos;

/**
 * Panel diseñado para la selección de parámetros para el informe
 * "Bitácora de acceso"
 * @author victor
 */
public class JpnParametrosParaBitacoraDeAcceso
        extends PanelParametrosParaReporte {

    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    protected void establecerValoresIniciales(){
        Date fechaInicial = new Date();
        fechaInicial.setHours(0);
        fechaInicial.setMinutes(0);
        jdcDesde.setDate(fechaInicial);
        jdcHasta.setDate(new Date());
        psoZonas.setObjetos(AdministradorPersistencia.getInstancia()
                .obtenerLista("Zona.findAll").toArray());
        psoTiposDeUsuario.setObjetos(AdministradorPersistencia
                .getInstancia().obtenerLista(
```

```
                "TipoDeUsuario.findAll").toArray());
        psoPersonas.setObjetos(AdministradorPersistencia.getInstancia()
                .obtenerLista("Persona.findAll").toArray());
    }

    protected boolean validarParametros(){
        if(jdcDesde.getDate() == null || jdcHasta.getDate() == null
                || jdcHasta.getDate().compareTo(
                jdcDesde.getDate()) < 0){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe introducir el rango temporal correctamente");
            return false;
        }
        if(psoZonas.getObjetosSeleccionados().length < 1){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe especificar al menos una zona");
            return false;
        }
        if(psoTiposDeUsuario.getObjetosSeleccionados().length < 1
                && psoPersonas.getObjetosSeleccionados().length < 1){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe especificar uno o más tipos de usuario"
                    + "\n y/o una o más personas");
            return false;
        }
        return true;
    }

    /**
     * Obtiene los identificadores de los objetos seleccionados en
     * la lista especificada
     * @param psoLista panel contendor de la lista de objetos
     * @return lista correspondiente a los objetos seleccionados en
     * el panel indicado
     */
    private List IDsDeObjetosEnLista(
            PanelSelectorDeObjetos psoLista){
        Object objetos[] = psoLista.getObjetosSeleccionados();
        Vector<Integer> vIds = new Vector<Integer>(objetos.length);
        for(int i=0; i<objetos.length; i++){
            if(objetos[i].getClass() == Zona.class)
                vIds.add(((Zona)objetos[i]).getId());
            if(objetos[i].getClass() == TipoDeUsuario.class)
                vIds.add(((TipoDeUsuario)objetos[i]).getId());
            if(objetos[i].getClass() == Persona.class)
                vIds.add(((Persona)objetos[i]).getId());
        }
        return vIds;
    }
```

```
    protected void generarReporte(){
        if(!validarParametros())
            return;
        try {
            InputStream stream = getClass().getResourceAsStream(
                    "/org/uaccess/reportes/BitacoraDeAcceso.jasper");
            String nParametros[] = new String[]{"hora_inicial",
            "hora_final", "zonas", "tipos_de_usuario", "personas",
            "logo"};
            Object vParametros[] = new Object[nParametros.length];
            vParametros[0] = new Timestamp(
                    jdcDesde.getDate().getTime());
            vParametros[1] = new Timestamp(
                    jdcHasta.getDate().getTime());
            vParametros[2] = IDsDeObjetosEnLista(psoZonas);
            List tiposDeUsuario = IDsDeObjetosEnLista(
                    psoTiposDeUsuario);
            if(tiposDeUsuario.size() > 0)
                vParametros[3] = tiposDeUsuario;
            else
                vParametros[3] = new Vector<Integer>(
                        Collections.singleton(-1));
            List personas = IDsDeObjetosEnLista(psoPersonas);
            if(personas.size() > 0)
                vParametros[4] = personas;
            else
                vParametros[4] = new Vector<Integer>(
                        Collections.singleton(-1));
            vParametros[5] = getClass().getResourceAsStream(
                    "/org/uaccess/reportes/logo.png");
            CargadorDeReportes.getInstancia().mostrarReporte(
                    stream, nParametros, vParametros,
                    "UAccess - Bitácora de acceso");
        } catch (Exception ex) {
            Mensajero.getInstancia().mostrarError("Error al intentar "
                    + "mostrar el reporte:\n" + ex.getMessage(), ex);
        }
    }

    protected void inicializarComponentes() {
        initComponents();
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        generarReporte();
```

```
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        cancelar();
    }


    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private com.toedter.calendar.JDateChooser jdcDesde;
    private com.toedter.calendar.JDateChooser jdcHasta;
    private javax.swing.JLabel jlbDesde;
    private javax.swing.JLabel jlbHasta;
    private javax.swing.JLabel jlbPersonas;
    private javax.swing.JLabel jlbTiposDeUsuario;
    private javax.swing.JLabel jlbZonas;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoPersonas;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoTiposDeUsuario;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoZonas;
    // End of variables declaration
}
```

Clase JpnParametrosParaPermisosDeAcceso

Esta clase define el formulario utilizado para seleccionar los parámetros para el reporte que muestra los permisos de acceso definidos en el sistema.

```
package org.uaccess.gui;

import java.io.InputStream;
import java.util.List;
import java.util.Vector;
import javax.swing.JButton;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.Zona;
import org.uaccess.utilidades.CargadorDeReportes;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.controles.PanelParametrosParaReporte;
```

```
import org.uaccess.utilidades.controles.PanelSelectorDeObjetos;

/**
 * Panel diseñado para la selección de parámetros para el informe
 * "Permisos de acceso"
 * @author victor
 */
public class JpnParametrosParaPermisosDeAcceso
        extends PanelParametrosParaReporte {

    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    protected void establecerValoresIniciales(){
        psoZonas.setObjetos(AdministradorPersistencia.getInstancia()
                .obtenerLista("Zona.findAll").toArray());
    }

    protected boolean validarParametros(){
        if(psoZonas.getObjetosSeleccionados().length < 1){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe especificar al menos una zona");
            return false;
        }
        return true;
    }

    /**
     * Obtiene los identificadores de los objetos seleccionados en
     * la lista especificada
     * @param psoLista panel contendor de la lista de objetos
     * @return lista correspondiente a los objetos seleccionados en
     * el panel indicado
     */
    private List IDsDeObjetosEnLista(PanelSelectorDeObjetos psoLista){
        Object objetos[] = psoLista.getObjetosSeleccionados();
        Vector<Integer> vIds = new Vector<Integer>(objetos.length);
        for(int i=0; i<objetos.length; i++){
            if(objetos[i].getClass() == Zona.class)
                vIds.add(((Zona)objetos[i]).getId());
        }
        return vIds;
    }

    protected void generarReporte(){
        if(!validarParametros())
            return;
        try {
            InputStream stream = getClass().getResourceAsStream(
```

```
                    "/org/uaccess/reportes/PermisosDeAcceso.jasper");
            String nParametros[] = new String[]{"zonas", "logo"};
            Object vParametros[] = new Object[nParametros.length];
            vParametros[0] = IDsDeObjetosEnLista(psoZonas);
            vParametros[1] = getClass().getResourceAsStream(
                    "/org/uaccess/reportes/logo.png");

            CargadorDeReportes.getInstancia().mostrarReporte(stream,
                    nParametros, vParametros,
                    "UAccess - Permisos de acceso");
        } catch (Exception ex) {
            Mensajero.getInstancia().mostrarError(
                    "Error al intentar mostrar el reporte:\n"
                    + ex.getMessage(), ex);
        }
    }

    protected void inicializarComponentes() {
        initComponents();
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        generarReporte();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        cancelar();
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbTiposDeUsuario;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoZonas;
    // End of variables declaration

}
```

Clase JpnParametrosParaPersonasRegistradas

Esta clase corresponde al formulario utilizado para seleccionar los parámetros para el reporte que muestra las personas registradas en el sistema.

```
package org.uaccess.gui;

import java.io.InputStream;
import java.util.List;
import java.util.Vector;
import javax.swing.JButton;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.TipoDeUsuario;
import org.uaccess.utilidades.CargadorDeReportes;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.controles.PanelParametrosParaReporte;
import org.uaccess.utilidades.controles.PanelSelectorDeObjetos;

/**
 * Panel diseñado para la selección de parámetros para el informe
 * "Permisos de acceso"
 * @author victor
 */
public class JpnParametrosParaPersonasRegistradas
        extends PanelParametrosParaReporte {

    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    protected void establecerValoresIniciales(){
        psoTiposDeUsuario.setObjetos(AdministradorPersistencia
                .getInstancia().obtenerLista(
                "TipoDeUsuario.findAll").toArray());
    }

    protected boolean validarParametros(){
        if(psoTiposDeUsuario.getObjetosSeleccionados().length < 1){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Debe especificar al menos un tipo de usuario");
            return false;
        }
        return true;
    }
```

```
    /**
     * Obtiene los identificadores de los objetos seleccionados en
     * la lista especificada
     * @param psoLista panel contendor de la lista de objetos
     * @return lista correspondiente a los objetos seleccionados en
     * el panel indicado
     */
    private List IDsDeObjetosEnLista(PanelSelectorDeObjetos psoLista){
        Object objetos[] = psoLista.getObjetosSeleccionados();
        Vector<Integer> vIds = new Vector<Integer>(objetos.length);
        for(int i=0; i<objetos.length; i++){
            if(objetos[i].getClass() == TipoDeUsuario.class)
                vIds.add(((TipoDeUsuario)objetos[i]).getId());
        }
        return vIds;
    }

    protected void generarReporte(){
        if(!validarParametros())
            return;
        try {
            InputStream stream = getClass().getResourceAsStream(
                    "/org/uaccess/reportes/PersonasRegistradas.jasper");
            String nParametros[] = new String[]{
                "tipos_de_usuario", "logo"};
            Object vParametros[] = new Object[nParametros.length];
            vParametros[0] = IDsDeObjetosEnLista(psoTiposDeUsuario);
            vParametros[1] = getClass().getResourceAsStream(
                    "/org/uaccess/reportes/logo.png");

            CargadorDeReportes.getInstancia().mostrarReporte(
                    stream, nParametros, vParametros,
                    "UAccess - Personas registradas");
        } catch (Exception ex) {
            Mensajero.getInstancia().mostrarError(
                    "Error al intentar mostrar el reporte:\n"
                    + ex.getMessage(), ex);
        }
    }

    protected void inicializarComponentes() {
        initComponents();
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
```

```
        generarReporte();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        cancelar();
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbTiposDeUsuario;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoTiposDeUsuario;
    // End of variables declaration

}
```

Clase JpnPersona

La clase "JpnPersona" corresponde al panel diseñado para la edición de personas.

```
package org.uaccess.gui;

import java.text.ParseException;
import java.util.List;
import org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto;
import com.sun.jna.Pointer;
import javax.swing.JButton;
import javax.swing.text.DefaultFormatterFactory;
import javax.swing.text.MaskFormatter;
import org.uaccess.fprint.CapturaDactilar;
import org.uaccess.utilidades.validacion.Validador;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.Huella;
import org.uaccess.persistencia.LoginAdministrativo;
import org.uaccess.persistencia.Persona;
import org.uaccess.persistencia.TipoDeDocumento;
import org.uaccess.persistencia.TipoDeUsuario;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.validacion.ValidadorDeUsuarios;

/**
 * Panle diseñado para la edición de personas
```

```
 * @author victor
 */
public class JpnPersona extends PanelAgregarOEditarObjeto {

    /** Huella dactilar correspondiente a la persona que
     * está siendo editada */
    private Huella capturaDactilar;
    /** Login administrativo correspondiente a la persona que
     * está siendo editada */
    private LoginAdministrativo login;

    /**
     * Crea un nuevo panel para la edición de una nueva persona
     */
    public JpnPersona() {
        super(new Persona(), true);
    }

    /**
     * Crea un nuevo panel a partir de los parámetros especificados
     * @param registro registro correspondiente a la persona a editar
     * @param esNuevoRegistro bandera que indica si es nuevo registro
     */
    public JpnPersona(Object registro, boolean esNuevoRegistro) {
        super(registro, esNuevoRegistro);
        initComponents();
    }

    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    @Override
    public void setObjeto(Object objeto) {
        super.setObjeto(objeto);
        setCapturaDactilar(obtenerHuellaRegistrada());
        setLogin(obtenerLoginRegistrado());
    }

    /**
     * Establece la captura dactilar correspondiente a la persona
     * @param capturaDactilar captura dactilar de la persona
     */
    private void setCapturaDactilar(Huella capturaDactilar) {
        this.capturaDactilar = capturaDactilar;
        sincronizarControlesConStatusHuella();
    }

    /**
     * Establece el login administrativo correspondiente a la persona
```

```
     * @param login login administrativo de la persona
     */
    private void setLogin(LoginAdministrativo login){
        this.login = login;
        sincronizarControlesConStatusLogin();
    }

    public void sincronizarControlesConObjeto() {
        this.jtfApellidos.setText(((Persona)objeto).getApellidos());
        this.jtfNombres.setText(((Persona)objeto).getNombres());
        if(((Persona)objeto).getTipoDeDocumento() != null)
            this.cmbTipoDocumento.setSelectedItem(((Persona)objeto)
                    .getTipoDeDocumento());
        this.jtfNumeroDocumento.setText(((Persona)objeto)
                .getNumeroDeDocumento());
        if(((Persona)objeto).getTipoDeUsuario() != null)
            this.cmbTipoUsuario.setSelectedItem(((Persona)objeto)
                    .getTipoDeUsuario());
        if(((Persona)objeto).getActivo() || nuevoRegistro)
            this.jrbActivo.setSelected(true);
        else
            this.jrbInactivo.setSelected(true);
    }

    public void sincronizarObjetoConControles(Object objeto) {
        ((Persona)objeto).setApellidos(jtfApellidos.getText());
        ((Persona)objeto).setNombres(jtfNombres.getText());
        ((Persona)objeto).setTipoDeDocumento(
                (TipoDeDocumento)cmbTipoDocumento.getSelectedItem());
        if(jtfNumeroDocumento.isEditValid()){
            try {
                jtfNumeroDocumento.commitEdit();
                ((Persona)objeto).setNumeroDeDocumento(
                        jtfNumeroDocumento.getText());
            } catch (ParseException ex) {
                ((Persona)objeto).setNumeroDeDocumento(null);
            }
        } else {
            ((Persona)objeto).setNumeroDeDocumento(null);
        }
        ((Persona)objeto).setTipoDeUsuario(
                (TipoDeUsuario)cmbTipoUsuario.getSelectedItem());
        ((Persona)objeto).setActivo(jrbActivo.isSelected());
    }

    /**
     * Sincroniza los controles respectivos con el estatus de
     * la captura dactilar de la persona
     */
    public void sincronizarControlesConStatusHuella(){
```

```
        if(capturaDactilar == null){
            jlbStatusCapturaDactilar.setText("<html><font color=red>"
                    + "Huella dactilar no registrada</font></html>");
            jbtEliminarCaptura.setEnabled(false);
        }
        else{
            jlbStatusCapturaDactilar.setText("<html><font color=green>"
                    + "Huella dactilar registrada</font></html>");
            jbtEliminarCaptura.setEnabled(true);
        }
    }

    /**
     * Sincroniza los controles respectivos con el estatus del
     * login administrativo de la persona
     */
    public void sincronizarControlesConStatusLogin(){
        if(login == null){
            jlbStatusLogin.setText("<html><font color=red>"
                    + "Login no registrado</font></html>");
            jbtEditarLogin.setText("Crear...");
            jbtEliminarLogin.setEnabled(false);
        }
        else{
            jlbStatusLogin.setText("<html><font color=green>"
                    + "Login registrado</font></html>");
            jbtEditarLogin.setText("Editar...");
            jbtEliminarLogin.setEnabled(true);
        }
    }

    protected void ejecutarInsercion(){
        if(validar()){
            sincronizarObjetoConControles(objetoOriginal);
            objeto = objetoOriginal;
            if(AdministradorPersistencia.getInstancia()
                    .insertar(getObjeto())){
                String mensaje = "La persona se ha guardado"
                        + " correctamente,\nsin embargo no"
                        + " se ha podido almacenar:";
                boolean error = false;
                if(capturaDactilar != null){
                    capturaDactilar.setPersona((Persona)objeto);
                    if(!AdministradorPersistencia.getInstancia()
                            .insertar(capturaDactilar)){
                        mensaje += "\n\t-Su huella dactilar";
                        error = true;
                    }
                }
                if(login != null){
```

```
                    login.setPersona((Persona)objeto);
                    if(!AdministradorPersistencia.getInstancia()
                            .insertar(login)){
                        mensaje += "\n\t-Su login administrativo";
                        error = true;
                    }
                }
                if(error)
                    Mensajero.getInstancia()
                            .mostrarAdvertencia(mensaje);
                firePropertyChange("operacionExitosa", true, null);
            }
            else
                Mensajero.getInstancia().mostrarError(
                        "La persona no ha podido ser guardada", null);
        }
    }

    protected void ejecutarActualizacion(){
        if(validar()){
            sincronizarObjetoConControles(objetoOriginal);
            objeto = objetoOriginal;
            if(AdministradorPersistencia.getInstancia()
                    .actualizar(getObjeto())){
                String mensaje = "La persona se ha actualizado "
                        + "correctamente,"
                        + "\nsin embargo no se ha podido:";
                if(!(actualizarHuella(mensaje)
                        && actualizarLogin(mensaje)))
                    Mensajero.getInstancia().mostrarAdvertencia(mensaje);
                firePropertyChange("operacionExitosa", true, null);
            }
            else
                Mensajero.getInstancia().mostrarError(
                        "La persona no ha podido ser actualizada", null);
        }
    }

    /**
     * Actualiza la huella correspondiente a la persona
     * @param mensaje mensaje resultante de la ejecución de
     * la actualización
     * @return 'true' si la huella se actualizó correctamente,
     * 'false' en caso contrario
     */
    protected boolean  actualizarHuella(String mensaje){
        boolean error = false;
        if(huellaExisteEnBD()){
            if(capturaDactilar != null){
                capturaDactilar.setPersona((Persona)objeto);
```

```
                if(!AdministradorPersistencia.getInstancia()
                        .actualizar(capturaDactilar)){
                    mensaje += "\n\t-Actualizar su captura dactilar";
                    error = true;
                }
            }
            else{
                capturaDactilar = (Huella) AdministradorPersistencia
                        .getInstancia().obtenerObjeto(
                        "Huella.findByPersona", (Persona)objeto);
                if(!AdministradorPersistencia.getInstancia()
                        .eliminar(capturaDactilar)){
                    mensaje += "\n\t-Eliminar la"
                            + " captura dactilar previa";
                    error = true;
                }
            }
        }
        else{
            if(capturaDactilar != null){
                capturaDactilar.setPersona((Persona)objeto);
                if(!AdministradorPersistencia.getInstancia()
                        .insertar(capturaDactilar)){
                    mensaje += "\n\t-Guardar su captura dactilar";
                    error = true;
                }
            }
        }
        return !error;
    }

    /**
     * Actualiza el login administrativo de la persona
     * @param mensaje mensaje resultante de la ejecución de
     * la actualización
     * @return 'true' si el login se actualizó correctamente,
     * 'false' en caso contrario
     */
    protected boolean  actualizarLogin(String mensaje){
        boolean error = false;
        if(loginExisteEnBD()){
            if(login != null){
                login.setPersona((Persona)objeto);
                if(!AdministradorPersistencia.getInstancia()
                        .actualizar(login)){
                    mensaje += "\n\t-Actualizar su"
                            + " login administrativo";
                    error = true;
                }
            }
```

```
            else{
                login = (LoginAdministrativo) AdministradorPersistencia
                        .getInstancia().obtenerObjeto(
                        "LoginAdministrativo.findByPersona",
                        (Persona)objeto);
                if(!AdministradorPersistencia.getInstancia()
                        .eliminar(login)){
                    mensaje += "\n\t-Eliminar su "
                            + "login administrativo previo";
                    error = true;
                }
            }
        }
        else{
            if(login != null){
                login.setPersona((Persona)objeto);
                if(!AdministradorPersistencia.getInstancia()
                        .insertar(login)){
                    mensaje += "\n\t-Guardar su login administrativo";
                    error = true;
                }
            }
        }
        return !error;
    }

    protected boolean validar(){
        String violaciones;
        violaciones = Validador.getInstancia()
                .obtenerViolaciones(getObjeto());
        if(!violaciones.isEmpty()){
            Mensajero.getInstancia()
                    .mostrarAdvertencia(violaciones);
            return false;
        }
        List coincidentes = AdministradorPersistencia.getInstancia()
                .obtenerLista("Persona.findByTipoYNumeroDeDocumento",
                ((Persona)objeto).getNumeroDeDocumento(),
                ((Persona)objeto).getTipoDeDocumento());
        if(coincidentes.size() > 0 && ((Persona)coincidentes.get(0))
                .getId() != ((Persona)objetoOriginal).getId()){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Ya existe una persona con el documento"
                    + " especificado\n(Combinación de tipo"
                    + " y número de documento)");
            return false;
        }
        if(capturaDactilar == null && login == null){
            if(!Mensajero.getInstancia().hacerPregunta(
                    "No hay ninguna captura dactilar ni login"
```

```
                    + " administrativo correspondiente a esta persona."
                    + "\nLa persona no será un usuario válido"
                    + " del sistema. ¿Desea continuar?"))
                return false;
        } else {
            if(capturaDactilar != null){
                String violacionesHuella;
                violacionesHuella = Validador.getInstancia()
                        .obtenerViolaciones(capturaDactilar);
                if(!violacionesHuella.isEmpty()){
                    Mensajero.getInstancia()
                            .mostrarAdvertencia(violacionesHuella);
                    return false;
                }
            }
            if(login != null){
                String violacionesLogin;
                violacionesLogin = Validador.getInstancia()
                        .obtenerViolaciones(login);
                if(!violacionesLogin.isEmpty()){
                    Mensajero.getInstancia().mostrarAdvertencia(
                            violacionesLogin);
                    return false;
                }
                List loginsCoincidentes = AdministradorPersistencia
                        .getInstancia().obtenerLista(
                        "LoginAdministrativo.findByLogin",
                        login.getLogin());
                if(loginsCoincidentes.size() > 0 &&
                        ((LoginAdministrativo)loginsCoincidentes
                        .get(0)).getPersona().getId() !=
                        ((Persona)objetoOriginal).getId()){
                    Mensajero.getInstancia().mostrarAdvertencia(
                            "El login asignado ya está"
                            + " siendo utilizado");
                    return false;
                }
            }
        }
        return true;
    }

    /**
     * Verifica si en la base de datos existe la huella correspondiente
     * a la persona editada
     * @return 'true' si la huella es encontrada,
     * 'false' en caso contrario
     */
    private boolean huellaExisteEnBD(){
        if(((Persona)objetoOriginal).getId() == null){
```

```
            return false;
        }
        Object[] huellasRegistradas = AdministradorPersistencia
                .getInstancia().obtenerLista("Huella.findByPersona",
                ((Persona)objetoOriginal)).toArray();
        if(huellasRegistradas.length > 0)
            return true;
        else
            return false;
    }

    /**
     * Obtiene de la base de datos la huella correspondiente
     * a la persona
     * @return huella registrada correspondiente a la persona,
     * 'null' si no existe tal huella
     */
    private Huella obtenerHuellaRegistrada(){
        if(huellaExisteEnBD())
            return (Huella)AdministradorPersistencia.getInstancia()
                    .obtenerObjeto("Huella.findByPersona",
                    (Persona)objetoOriginal);
        else
            return null;
    }

    /**
     * Verifica si en la base de datos existe el login
     * administrativo correspondiente a la persona editada
     * @return 'true' si el login es encontrado,
     * 'false' en caso contrario
     */
    private boolean loginExisteEnBD(){
        if(((Persona)objetoOriginal).getId() == null){
            return false;
        }
        Object[] loginsRegistrados = AdministradorPersistencia
                .getInstancia().obtenerLista(
                "LoginAdministrativo.findByPersona",
                ((Persona)objetoOriginal)).toArray();
        if(loginsRegistrados.length > 0)
            return true;
        else
            return false;
    }

    /**
     * Obtiene de la base de datos el login administrativo
     * correspondiente a la persona
     * @return login registrado correspondiente a la persona,
```

```
     * 'null' si no existe tal login
     */
    private LoginAdministrativo obtenerLoginRegistrado(){
        if(loginExisteEnBD())
            return (LoginAdministrativo)AdministradorPersistencia
                    .getInstancia().obtenerObjeto(
                    "LoginAdministrativo.findByPersona",
                    (Persona)objetoOriginal);
        else
            return null;
    }

    /**
     * Inicia el registro de la huella dactilar correspondiente
     * a la persona
     */
    private void registrarHuella(){
        if(!ValidadorDeUsuarios.getInstancia()
                .verificarPermiso("Admon de huellas")){
            Mensajero.getInstancia().mostrarError(
                    "Acceso denegado", null);
            return;
        }
        CapturaDactilar captura = new DialogoCapturarHuella(
                getClass().getResource("/org/uaccess/recursos/"
                + "imagenes/16/persona.png")).mostrar(null);
        if(captura != null){
            if(captura.getPDatos() == Pointer.NULL)
                Mensajero.getInstancia().mostrarInformacion(
                        "captura.getPDatos() == Pointer.NULL");
            capturaDactilar = new Huella(null,
                    captura.getBytesDatos());
        }
        sincronizarControlesConStatusHuella();
    }

    /**
     * Elimina la huella dactilar correspondiente a la persona
     */
    private void eliminarHuellaRegistrada(){
        if(!ValidadorDeUsuarios.getInstancia()
                .verificarPermiso("Admon de huellas")){
            Mensajero.getInstancia().mostrarError(
                    "Acceso denegado", null);
            return;
        }
        if(Mensajero.getInstancia().hacerPregunta("¿Realmente desea"
                + " eliminar la captura dactilar registrada?"
                + "\n(Los cambios aún podrán ser cancelados)")){
            capturaDactilar = null;
```

```
            sincronizarControlesConStatusHuella();
        }
    }

    /**
     * Inicia la edición del login administrativo correspondiente
     * a la persona
     */
    private void editarLoginAdministrativo(){
        if(!ValidadorDeUsuarios.getInstancia()
                .verificarPermiso("Admon de logins")){
            Mensajero.getInstancia().mostrarError(
                    "Acceso denegado", null);
            return;
        }
        login = new DialogoLoginAdministrativo(getClass()
                .getResource("/org/uaccess/recursos/imagenes"
                + "/16/persona.png")).mostrar(login);
        sincronizarControlesConStatusLogin();
    }

    /**
     * Elimina el login administrativo correspondiente a la persona
     */
    private void eliminarLoginAdministrativo(){
        if(!ValidadorDeUsuarios.getInstancia()
                .verificarPermiso("Admon de logins")){
            Mensajero.getInstancia().mostrarError(
                    "Acceso denegado", null);
            return;
        }
        if(Mensajero.getInstancia().hacerPregunta(
                "¿Realmente desea eliminar el login administrativo"
                + " asignado?\n(Los cambios aún podrán ser"
                + " cancelados)")){
            login = null;
            sincronizarControlesConStatusLogin();
        }
    }

    /**
     * Carga los datos correspondientes a las entidades relacionadas
     */
    private void cargarDatosExternos(){
        cmbTipoDocumento.instalar(AdministradorPersistencia
                .getInstancia().obtenerLista(
                "TipoDeDocumento.findAll"));
        cmbTipoUsuario.instalar(AdministradorPersistencia
                .getInstancia().obtenerLista("TipoDeUsuario.findAll"));
    }
```

```
/**
 * Sincroniza el formato del documento con el tipo seleccionado
 */
private void sincronizarTipoYFormatoDeDocumento(){
    if(cmbTipoDocumento.getSelectedItem() != null){
        try {
            jtfNumeroDocumento.setValue(null);
            jtfNumeroDocumento.setFormatterFactory(
                    new DefaultFormatterFactory(new MaskFormatter(
                    ((TipoDeDocumento) cmbTipoDocumento
                    .getSelectedItem()).getFormato())));
            jtfNumeroDocumento.setEnabled(true);
        } catch (ParseException ex) {
            Mensajero.getInstancia().mostrarError(
                    "El formato para el tipo de documento"
                    + " no está definido correctamente.", ex);
            jtfNumeroDocumento.setEnabled(false);
        }
    }
    else{
        jtfNumeroDocumento.setEnabled(false);
    }
}

protected void inicializarComponentes(){
    initComponents();
    cargarDatosExternos();
}

@Override
public void actualizarDatosRelacionados() {
    cargarDatosExternos();
}


private void initComponents() {
    //Código de inicialización de la UI generado por el IDE
}

private void jbtAceptarActionPerformed(
        java.awt.event.ActionEvent evt) {
    guardarObjeto();
}

private void jbtCancelarActionPerformed(
        java.awt.event.ActionEvent evt) {
    firePropertyChange("operacionCancelada", true, null);
}
```

```
    private void jbtCapturarHuellaActionPerformed(
            java.awt.event.ActionEvent evt) {
        registrarHuella();
    }

    private void jbtEliminarCapturaActionPerformed(
            java.awt.event.ActionEvent evt) {
        eliminarHuellaRegistrada();
    }

    private void jbtEliminarLoginActionPerformed(
            java.awt.event.ActionEvent evt) {
        eliminarLoginAdministrativo();
    }

    private void cmbTipoDocumentoActionPerformed(
            java.awt.event.ActionEvent evt) {
        sincronizarTipoYFormatoDeDocumento();
    }

    private void jbtEditarLoginActionPerformed(
            java.awt.event.ActionEvent evt) {
        editarLoginAdministrativo();
    }


    // Variables declaration - do not modify
    private javax.swing.ButtonGroup bgrActivo;
    private org.uaccess.utilidades.controles.ComboBox cmbTipoDocumento;
    private org.uaccess.utilidades.controles.ComboBox cmbTipoUsuario;
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JButton jbtCapturarHuella;
    private javax.swing.JButton jbtEditarLogin;
    private javax.swing.JButton jbtEliminarCaptura;
    private javax.swing.JButton jbtEliminarLogin;
    private javax.swing.JLabel jlbApellidos;
    private javax.swing.JLabel jlbCapturaDactilar;
    private javax.swing.JLabel jlbLoginAdministrativo;
    private javax.swing.JLabel jlbNombres;
    private javax.swing.JLabel jlbNumeroDocumento;
    private javax.swing.JLabel jlbStatus;
    private javax.swing.JLabel jlbStatusCapturaDactilar;
    private javax.swing.JLabel jlbStatusLogin;
    private javax.swing.JLabel jlbTipoDocumento;
    private javax.swing.JLabel jlbTipoUsuario;
    private javax.swing.JRadioButton jrbActivo;
    private javax.swing.JRadioButton jrbInactivo;
    private org.uaccess.utilidades.controles.TextField jtfApellidos;
    private org.uaccess.utilidades.controles.TextField jtfNombres;
```

```
    private javax.swing.JFormattedTextField jtfNumeroDocumento;
    // End of variables declaration

}
```

Clase JpnTipoDeDocumento

Esta clase define el panel diseñado para la edición de los tipos de documento de identidad.

```
package org.uaccess.gui;

import java.util.List;
import javax.swing.JButton;
import org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto;
import org.uaccess.utilidades.validacion.Validador;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.TipoDeDocumento;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel diseñado para la edición de tipos de documento
 * @author victor
 */
public class JpnTipoDeDocumento extends PanelAgregarOEditarObjeto {

    /**
     * Crea un nuevo panel para la edición de un nuevo
     * tipo de documento
     */
    public JpnTipoDeDocumento() {
        super(new TipoDeDocumento(), true);
    }

    /**
     * Crea un nuevo panel para la edición de tipos de objeto con los
     * parámetros indicados
     * @param registro registro correspondiente al tipo de documento
     * @param esNuevoRegistro bandera que indica si es nuevo registro
     */
    public JpnTipoDeDocumento(
            Object registro, boolean esNuevoRegistro) {
        super(registro, esNuevoRegistro);
        initComponents();
    }
```

```
public JButton getBotonPorDefecto(){
    return jbtAceptar;
}

public void sincronizarControlesConObjeto() {
    this.jtfNombre.setText(
            ((TipoDeDocumento)objeto).getNombre());
    this.jtaDescripcion.setText(
            ((TipoDeDocumento)objeto).getDescripcion());
    this.jtfFormato.setText(
            ((TipoDeDocumento)objeto).getFormato());
}

public void sincronizarObjetoConControles(Object objeto) {
    ((TipoDeDocumento)objeto).setNombre(
            this.jtfNombre.getText());
    ((TipoDeDocumento)objeto).setDescripcion(
            this.jtaDescripcion.getText());
    ((TipoDeDocumento)objeto).setFormato(
            this.jtfFormato.getText());
}

protected void ejecutarInsercion(){
    if(validar()){
        sincronizarObjetoConControles(objetoOriginal);
        objeto = objetoOriginal;
        if(AdministradorPersistencia.getInstancia()
                .insertar(getObjeto())){
            firePropertyChange("operacionExitosa", true, null);
        }
        else
            Mensajero.getInstancia().mostrarError(
                    "La inserción no ha podido ser realizada",
                    null);
    }
}

protected void ejecutarActualizacion(){
    if(validar()){
        sincronizarObjetoConControles(objetoOriginal);
        objeto = objetoOriginal;
        if(AdministradorPersistencia.getInstancia()
                .actualizar(getObjeto())){
            firePropertyChange("operacionExitosa", true, null);
        }
        else
            Mensajero.getInstancia().mostrarError(
                    "La actualización no ha podido ser realizada",
                    null);
```

```
        }
    }

    protected boolean validar(){
        String violaciones;
        violaciones = Validador.getInstancia()
                .obtenerViolaciones(getObjeto());
        if(!violaciones.isEmpty()){
            Mensajero.getInstancia().mostrarAdvertencia(violaciones);
            return false;
        }
        List coincidentes = AdministradorPersistencia.getInstancia()
                .obtenerLista("TipoDeDocumento.findByNombre",
                ((TipoDeDocumento)objeto).getNombre());
        if(coincidentes.size() > 0 && ((TipoDeDocumento)coincidentes
                .get(0)).getId() != ((TipoDeDocumento)objetoOriginal)
                .getId()){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Ya existe un tipo de documento con el"
                    + " nombre indicado");
            return false;
        }
        return true;
    }

    protected void inicializarComponentes(){
        initComponents();
    }

    @Override
    public void actualizarDatosRelacionados() {
        //No hay entidades de las que el objeto dependa
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        guardarObjeto();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        firePropertyChange("operacionCancelada", true, null);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
```

```
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbDescripcion;
    private javax.swing.JLabel jlbFormato;
    private javax.swing.JLabel jlbNombre;
    private javax.swing.JScrollPane jspDescripcion;
    private org.uaccess.utilidades.controles.TextArea jtaDescripcion;
    private org.uaccess.utilidades.controles.TextField jtfFormato;
    private org.uaccess.utilidades.controles.TextField jtfNombre;
    // End of variables declaration

}
```

Clase JpnTipoDeUsuario

Esta clase corresponde al formulario para la edición de los tipos de usuario del sistema.

```
package org.uaccess.gui;

import java.util.List;
import javax.swing.JButton;
import org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto;
import org.uaccess.utilidades.validacion.Validador;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.PrivilegioAdministrativo;
import org.uaccess.persistencia.TipoDeUsuario;
import org.uaccess.persistencia.Zona;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel diseñado para la edición de tipos de usuario
 * @author victor
 */
public class JpnTipoDeUsuario extends PanelAgregarOEditarObjeto {

    /**
     * Crea un nuevo panel para la edición de un nuevo tipo de usuario
     */
    public JpnTipoDeUsuario() {
        super(new TipoDeUsuario(), true);
    }

    /**
     * Crea un nuevo panel para la edición de tipos de usuario con
     * los parámetros indicados
```

```
     * @param registro registro correspondiente al tipo de usuario
     * a editar
     * @param esNuevoRegistro bandera que indica si es nuevo registro
     * o no
     */
    public JpnTipoDeUsuario(
            Object registro, boolean esNuevoRegistro) {
        super(registro, esNuevoRegistro);
        initComponents();
    }

    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    public void sincronizarControlesConObjeto() {
        this.jtfNombre.setText(((TipoDeUsuario)objeto).getNombre());
        this.jtaDescripcion.setText(
                ((TipoDeUsuario)objeto).getDescripcion());
        if(((TipoDeUsuario)objeto).getZonasPermitidas() != null)
            this.psoPermisosDeAcceso.setObjetosSeleccionados(
                    ((TipoDeUsuario)objeto).getZonasPermitidas()
                    .toArray());
        else
            this.psoPermisosDeAcceso.setObjetosSeleccionados(
                    new Object[0]);
        if(((TipoDeUsuario)objeto)
                .getPermisosAdministrativos() != null)
            this.psoPermisosAdministrativos
                    .setObjetosSeleccionados(((TipoDeUsuario)objeto)
                    .getPermisosAdministrativos().toArray());
        else
            this.psoPermisosAdministrativos
                    .setObjetosSeleccionados(new Object[0]);
    }

    public void sincronizarObjetoConControles(Object objeto) {
        ((TipoDeUsuario)objeto).setNombre(this.jtfNombre.getText());
        ((TipoDeUsuario)objeto).setDescripcion(
                this.jtaDescripcion.getText());
        sincronizarObjetoConZonasPermitidas();
        sincronizarObjetoConPermisosAdministrativos();
    }

    /**
     * Asigna al tipo de usuario, las zonas seleccionadas
     * como permitidas
     */
    public void sincronizarObjetoConZonasPermitidas(){
        Object zonasSeleccionadas[] = psoPermisosDeAcceso
```

```
                .getObjetosSeleccionados();
        Zona zonasPermitidas[] = new Zona[zonasSeleccionadas.length];
        for(int i=0; i<zonasPermitidas.length; i++)
            zonasPermitidas[i] = (Zona)zonasSeleccionadas[i];
        ((TipoDeUsuario)objeto).setZonasPermitidas(zonasPermitidas);
    }

    /**
     * Asigna al tipo de usuario los privilegios
     * administrativos seleccionados
     */
    public void sincronizarObjetoConPermisosAdministrativos(){
        Object privilegiosSeleccionados[] =
                psoPermisosAdministrativos.getObjetosSeleccionados();
        PrivilegioAdministrativo permisosAsignados[] =
                new PrivilegioAdministrativo[
                privilegiosSeleccionados.length];
        for(int i=0; i<permisosAsignados.length; i++)
            permisosAsignados[i] = (PrivilegioAdministrativo)
                    privilegiosSeleccionados[i];
        ((TipoDeUsuario)objeto).setPermisosAdministrativos(
                permisosAsignados);
    }

    protected void ejecutarInsercion(){
        if(validar()){
            sincronizarObjetoConControles(objetoOriginal);
            objeto = objetoOriginal;
            if(AdministradorPersistencia.getInstancia()
                    .insertar(getObjeto())){
                firePropertyChange("operacionExitosa", true, null);
            }
            else
                Mensajero.getInstancia().mostrarError(
                        "La inserción no ha podido ser realizada",
                        null);
        }
    }

    protected void ejecutarActualizacion(){
        if(validar()){
            sincronizarObjetoConControles(objetoOriginal);
            objeto = objetoOriginal;
            if(AdministradorPersistencia.getInstancia()
                    .actualizar(getObjeto())){
                firePropertyChange("operacionExitosa", true, null);
            }
            else
                Mensajero.getInstancia().mostrarError(
                        "La actualización no ha podido ser realizada",
```

```
                        null);;
        }
    }

    protected boolean validar(){
        String violaciones;
        violaciones = Validador.getInstancia().obtenerViolaciones(
                getObjeto());
        if(!violaciones.isEmpty()){
            Mensajero.getInstancia().mostrarAdvertencia(violaciones);
            return false;
        }
        List coincidentes = AdministradorPersistencia.getInstancia()
                .obtenerLista("TipoDeUsuario.findByNombre",
                ((TipoDeUsuario)objeto).getNombre());
        if(coincidentes.size() > 0 && ((TipoDeUsuario)coincidentes
                .get(0)).getId() != ((TipoDeUsuario)objetoOriginal)
                .getId()){
            Mensajero.getInstancia().mostrarAdvertencia("Ya existe"
                    + " un tipo de usuario con el nombre indicado");
            return false;
        }
        if(((TipoDeUsuario)getObjeto()).getPermisosAdministrativos()
                .size() < 1 && ((TipoDeUsuario)getObjeto())
                .getZonasPermitidas().size() < 1){
            if(!Mensajero.getInstancia().hacerPregunta("No ha"
                    + " otorgado ninguna zona de acceso ni permisos"
                    + " administrativos al tipo de usuario."
                    + "\nLos usuarios de este tipo no podrán"
                    + " ser utilizados. ¿Desea continuar?"))
                return false;
        }
        return true;
    }

    /**
     * Carga los datos de las entidades relacionadas en los
     * controles correspondientes
     */
    private void cargarDatosExternos(){
        psoPermisosDeAcceso.setObjetos(AdministradorPersistencia
                .getInstancia().obtenerLista("Zona.findAll").toArray());
        psoPermisosAdministrativos.setObjetos(
                AdministradorPersistencia.getInstancia().obtenerLista(
                "PrivilegioAdministrativo.findAll").toArray());
    }

    protected void inicializarComponentes(){
        initComponents();
        cargarDatosExternos();
```

```
    }

    @Override
    public void actualizarDatosRelacionados() {
        cargarDatosExternos();
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        guardarObjeto();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        firePropertyChange("operacionCancelada", true, null);
    }


    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbDescripcion;
    private javax.swing.JLabel jlbNombre;
    private javax.swing.JLabel jlbPermisosDeAcceso;
    private javax.swing.JLabel jlbPermisosDeAcceso1;
    private javax.swing.JScrollPane jspDescripcion;
    private org.uaccess.utilidades.controles.TextArea jtaDescripcion;
    private org.uaccess.utilidades.controles.TextField jtfNombre;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoPermisosAdministrativos;
    private org.uaccess.utilidades.controles
            .PanelSelectorDeObjetos psoPermisosDeAcceso;
    // End of variables declaration

}
```

Clase JpnZona

El panel "JpnZona" corresponde al formulario utilizado para la edición y creación de las zonas geográficas.

```
package org.uaccess.gui;

import java.util.List;
import javax.swing.JButton;
import org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto;
import org.uaccess.utilidades.validacion.Validador;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.persistencia.Zona;
import org.uaccess.utilidades.Mensajero;

/**
 * Panel diseñado para la edición de zonas
 * @author victor
 */
public class JpnZona extends PanelAgregarOEditarObjeto {

    /**
     * Crea un nuevo panel para la edición de una nueva zona
     */
    public JpnZona() {
        super(new Zona(), true);
    }

    /**
     * Crea un nuevo panel para la edición de zonas con los
     * parámetros recibidos
     * @param registro registro correspondiente a la zona a editar
     * @param esNuevoRegistro bandera que indica si es nuevo registro
     */
    public JpnZona(Object registro, boolean esNuevoRegistro) {
        super(registro, esNuevoRegistro);
        initComponents();
    }

    public JButton getBotonPorDefecto(){
        return jbtAceptar;
    }

    public void sincronizarControlesConObjeto() {
        this.jtfNombre.setText(((Zona)objeto).getNombre());
        this.jtaDescripcion.setText(((Zona)objeto).getDescripcion());
    }

    public void sincronizarObjetoConControles(Object objeto) {
        ((Zona)objeto).setNombre(this.jtfNombre.getText());
        ((Zona)objeto).setDescripcion(this.jtaDescripcion.getText());
    }

    protected void ejecutarInsercion(){
        if(validar()){
```

```
            sincronizarObjetoConControles(objetoOriginal);
            objeto = objetoOriginal;
            if(AdministradorPersistencia.getInstancia().insertar(
                    getObjeto())){
                firePropertyChange("operacionExitosa", true, null);
            }
            else
                Mensajero.getInstancia().mostrarError(
                        "La inserción no ha podido ser realizada",
                        null);;
        }
    }

    protected void ejecutarActualizacion(){
        if(validar()){
            sincronizarObjetoConControles(objetoOriginal);
            objeto = objetoOriginal;
            if(AdministradorPersistencia.getInstancia().actualizar(
                    getObjeto())){
                firePropertyChange("operacionExitosa", true, null);
            }
            else
                Mensajero.getInstancia().mostrarError(
                        "La actualización no ha podido ser realizada",
                        null);;
        }
    }

    protected boolean validar(){
        String violaciones;
        violaciones = Validador.getInstancia().obtenerViolaciones(
                getObjeto());
        if(!violaciones.isEmpty()){
            Mensajero.getInstancia().mostrarAdvertencia(violaciones);
            return false;
        }
        List coincidentes = AdministradorPersistencia.getInstancia()
                .obtenerLista("Zona.findByNombre",
                ((Zona)objeto).getNombre());
        if(coincidentes.size() > 0 && ((Zona)coincidentes.get(0))
                .getId() != ((Zona)objetoOriginal).getId()){
            Mensajero.getInstancia().mostrarAdvertencia(
                    "Ya existe una zona con el nombre indicado");
            return false;
        }
        return true;
    }

    protected void inicializarComponentes(){
        initComponents();
```

```
    }

    public void actualizarDatosRelacionados() {
        //No hay entidades de las cuales dependa el objeto
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    private void jbtAceptarActionPerformed(
            java.awt.event.ActionEvent evt) {
        guardarObjeto();
    }

    private void jbtCancelarActionPerformed(
            java.awt.event.ActionEvent evt) {
        firePropertyChange("operacionCancelada", true, null);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jbtAceptar;
    private javax.swing.JButton jbtCancelar;
    private javax.swing.JLabel jlbDescripcion;
    private javax.swing.JLabel jlbNombre;
    private javax.swing.JScrollPane jspDescripcion;
    private org.uaccess.utilidades.controles.TextArea jtaDescripcion;
    private org.uaccess.utilidades.controles.TextField jtfNombre;
    // End of variables declaration

}
```

Clase Principal

Esta clase corresponde a la ventana principal del sistema administrativo.

```
package org.uaccess.gui;

import java.io.IOException;
import org.uaccess.utilidades.controles.PanelAdministradorDeTabla;
import java.beans.PropertyVetoException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
```

```
import org.uaccess.persistencia.Persona;
import org.uaccess.persistencia.TipoDeDocumento;
import org.uaccess.persistencia.TipoDeUsuario;
import org.uaccess.persistencia.Zona;
import org.uaccess.utilidades.FormatoDeTablaPersona;
import org.uaccess.utilidades.controles.InternalFrame;
import org.uaccess.utilidades.controles.InternalFramePrivado;

/**
 * Ventana principal del sistema
 * @author victor
 */
public class Principal extends javax.swing.JFrame {
    /** Ventana de administración de tipos de usuario */
    InternalFramePrivado ifpTiposDeUsuario;
    /** Ventana de administración de tipos de documento */
    InternalFramePrivado ifpTiposDeDocumento;
    /** Ventana de administración de personas */
    InternalFramePrivado ifpPersonas;
    /** Ventana de administración de zonas */
    InternalFramePrivado ifpZona;
    /** Ventana de selección de parámetros y carga del
     * reporte 'Bitácora de acceso' */
    InternalFramePrivado ifpParametrosParaBitacoraDeAcceso;
    /** Ventana de selección de parámetros y carga del
     * reporte 'Personas registradas' */
    InternalFramePrivado ifpParametrosParaPersonasRegistradas;
    /** Ventana de selección de parámetros y carga del
     * reporte 'Permisos de acceso' */
    InternalFramePrivado ifpParametrosParaPermisosDeAcceso;

    /**
     * Crea una nueva ventana
     */
    public Principal() {
        initComponents();
        try {
            this.setIconImage(ImageIO.read(getClass().getResource(
                    "/org/uaccess/recursos/imagenes/icono.png")));
        } catch (IOException ex) {
            Logger.getLogger(Principal.class.getName())
                    .log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Indica si una ventana interna se encuentra abierta
     * @param frame ventana a evaluar
     * @return 'true' si está abierta, 'false' en caso contrario
     */
```

```
    private boolean  internalFrameAbierto(InternalFrame frame){
        if(frame == null)
            return false;
        if(frame.isVisible())
            return true;
        return false;
    }

    private void initComponents() {
        //Código de inicialización de la UI generado por el IDE
    }

    /**
     * Evento ActionPerformed sobre jmiTiposDeUsuario.
     * Carga la ventana de administración de los tipos de usuario
     */
    private void jmiTiposDeUsuarioActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(ifpTiposDeUsuario))
                ifpTiposDeUsuario.setSelected(true);
            else{
                String[] campos = new String[] {"nombre","descripcion"};
                String[] encabezadosDeColumnas = new String[] {
                    "Nombre","Descripción"};
                int[] anchosDeColumnas = new int[] {100, 400};
                int[] camposDeFiltrado = new int[] {0, 1};
                int[] camposDeFiltradoIndividual = new int[] {0, 1};
                PanelAdministradorDeTabla panel =
                        new PanelAdministradorDeTabla(
                        TipoDeUsuario.class, "TipoDeUsuario.findAll",
                        campos, encabezadosDeColumnas, anchosDeColumnas,
                        camposDeFiltrado, camposDeFiltradoIndividual,
                        new JpnTipoDeUsuario(),
                        jmiTiposDeUsuario.getIcon());
                ifpTiposDeUsuario = new InternalFramePrivado(
                        desktopPane, panel, "Tipos de usuario",
                        "Admon de tipos de usuario",
                        jmiTiposDeUsuario.getIcon());
                ifpTiposDeUsuario.mostrar();
            }

        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }

    /**
     * Evento ActionPerformed sobre jmiTiposDeDocumento.
```

```
     * Carga la ventana de administración de los tipos de documento
     */
    private void jmiTiposDeDocumentoActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(ifpTiposDeDocumento))
                ifpTiposDeDocumento.setSelected(true);
            else{
                String[] campos = new String[] {
                    "nombre", "descripcion", "formato"};
                String[] encabezadosDeColumnas = new String[] {
                    "Nombre", "Descripción", "Formato"};
                int[] anchosDeColumnas = new int[] {80, 350, 70};
                int[] camposDeFiltrado = new int[] {0, 1, 2};
                int[] camposDeFiltradoIndividual = new int[] {0, 1};
                PanelAdministradorDeTabla panel =
                        new PanelAdministradorDeTabla(
                        TipoDeDocumento.class, "TipoDeDocumento.findAll",
                        campos, encabezadosDeColumnas, anchosDeColumnas,
                        camposDeFiltrado, camposDeFiltradoIndividual,
                        new JpnTipoDeDocumento(),
                        jmiTiposDeDocumento.getIcon());
                ifpTiposDeDocumento = new InternalFramePrivado(
                        desktopPane, panel, "Tipos de documento",
                        "Admon de tipos de docto",
                        jmiTiposDeDocumento.getIcon());
                ifpTiposDeDocumento.mostrar();
            }
        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }

    /**
     * Evento ActionPerformed sobre jmiPersonas.
     * Carga la ventana de administración de las personas
     */
    private void jmiPersonasActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(ifpPersonas))
                ifpPersonas.setSelected(true);
            else{
                String[] campos = new String[] {"apellidos", "nombres",
                "tipoDeDocumento", "numeroDeDocumento",
                "tipoDeUsuario", "activo"};
                String[] encabezadosDeColumnas = new String[] {
                    "Apellidos", "Nombres", "Tipo de doc.",
                    "No. de doc.", "Tipo de usuario", "Activo"};
```

```
                int[] anchosDeColumnas = new int[] {
                    60, 60, 60, 40, 50, 10};
                int[] camposDeFiltrado = new int[] {0, 1, 2, 3, 4};
                int[] camposDeFiltradoIndividual = new int[] {0, 1, 3};
                PanelAdministradorDeTabla panel =
                        new PanelAdministradorDeTabla(Persona.class,
                        "Persona.findAll", campos, encabezadosDeColumnas,
                        anchosDeColumnas, new FormatoDeTablaPersona(
                        encabezadosDeColumnas), camposDeFiltrado,
                        camposDeFiltradoIndividual, new JpnPersona(),
                        jmiPersonas.getIcon());
                ifpPersonas = new InternalFramePrivado(desktopPane,
                        panel, "Personas", "Admon de personas",
                        jmiPersonas.getIcon());
                ifpPersonas.mostrar();
            }
        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }

    /**
     * Evento ActionPerformed sobre jmiZonas.
     * Carga la ventana de administración de las zonas
     */
    private void jmiZonasActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(ifpZona))
                ifpZona.setSelected(true);
            else{
                String[] campos = new String[] {"nombre","descripcion"};
                String[] encabezadosDeColumnas = new String[] {
                    "Nombre","Descripción"};
                int[] anchosDeColumnas = new int[] {100, 400};
                int[] camposDeFiltrado = new int[] {0, 1};
                int[] camposDeFiltradoIndividual = new int[] {0, 1};
                PanelAdministradorDeTabla panel =
                        new PanelAdministradorDeTabla(Zona.class,
                        "Zona.findAll", campos, encabezadosDeColumnas,
                        anchosDeColumnas, camposDeFiltrado,
                        camposDeFiltradoIndividual, new JpnZona(),
                        jmiZonas.getIcon());
                ifpZona = new InternalFramePrivado(desktopPane, panel,
                        "Zonas", "Admon de zonas", jmiZonas.getIcon());
                ifpZona.mostrar();
            }
        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
```

```
                    Level.SEVERE, null, ex);
        }
    }

    /**
     * Evento ActionPerformed sobre jmiBitacoraDeAcceso.
     * Carga la ventana de parámetros para generar el
     * reporte "Bitácora de acceso"
     */
    private void jmiBitacoraDeAccesoActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(ifpParametrosParaBitacoraDeAcceso))
                ifpParametrosParaBitacoraDeAcceso.setSelected(true);
            else{
                ifpParametrosParaBitacoraDeAcceso =
                        new InternalFramePrivado(desktopPane,
                        new JpnParametrosParaBitacoraDeAcceso(),
                        "Bitácora de acceso", "Rep bitacora de acceso",
                        jmiBitacoraDeAcceso.getIcon());
                ifpParametrosParaBitacoraDeAcceso.mostrar();
            }
        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }

    /**
     * Evento ActionPerformed sobre jmiPersonasRegistradas.
     * Carga la ventana de parámetros para generar el
     * reporte "Personas registradas"
     */
    private void jmiPersonasRegistradasActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(
                    ifpParametrosParaPersonasRegistradas))
                ifpParametrosParaPersonasRegistradas
                .setSelected(true);
            else{
                ifpParametrosParaPersonasRegistradas =
                        new InternalFramePrivado(desktopPane,
                        new JpnParametrosParaPersonasRegistradas(),
                        "Personas registradas", "Rep personas regis"
                        + "tradas", jmiPersonasRegistradas.getIcon());
                ifpParametrosParaPersonasRegistradas.mostrar();
            }
        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
```

```
                    Level.SEVERE, null, ex);
        }
    }

    /**
     * Evento ActionPerformed sobre jmiPermisosDeAcceso.
     * Carga la ventana de parámetros para generar el
     * reporte "Permisos de acceso"
     */
    private void jmiPermisosDeAccesoActionPerformed(
            java.awt.event.ActionEvent evt) {
        try {
            if(internalFrameAbierto(ifpParametrosParaPermisosDeAcceso))
                ifpParametrosParaPermisosDeAcceso.setSelected(true);
            else{
                ifpParametrosParaPermisosDeAcceso =
                        new InternalFramePrivado(desktopPane,
                        new JpnParametrosParaPermisosDeAcceso(),
                        "Permisos de acceso", "Rep permisos de acceso",
                        jmiPermisosDeAcceso.getIcon());
                ifpParametrosParaPermisosDeAcceso.mostrar();
            }
        } catch (PropertyVetoException ex) {
            Logger.getLogger(Principal.class.getName()).log(
                    Level.SEVERE, null, ex);
        }
    }

    // Variables declaration - do not modify
    private javax.swing.JDesktopPane desktopPane;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jmiBitacoraDeAcceso;
    private javax.swing.JMenuItem jmiPermisosDeAcceso;
    private javax.swing.JMenuItem jmiPersonas;
    private javax.swing.JMenuItem jmiPersonasRegistradas;
    private javax.swing.JMenuItem jmiTiposDeDocumento;
    private javax.swing.JMenuItem jmiTiposDeUsuario;
    private javax.swing.JMenuItem jmiZonas;
    private javax.swing.JMenu jmnAdministracion;
    private javax.swing.JMenu jmnReportes;
    // End of variables declaration

}
```

Clases generales

En este módulo se encuentran las clases generales del sistema administrativo.

Clase GestorDeDatosGlobales

Esta clase define al objeto encargado de gestionar información global del sistema, como por ejemplo, la información referente al usuario actual del sistema.

```
package org.uaccess;

import org.uaccess.persistencia.Persona;

/**
 * Objeto encargado de almacenar y gestionar los datos globales
 * @author victor
 */
public class GestorDeDatosGlobales {

    /** Instancia del gestor de datos globales de la aplicación */
    private static GestorDeDatosGlobales instacia;
    /** Usuario de la aplicación */
    private Persona usuario;

    /**
     * Crea un nuevo gestor de datos globales
     */
    private GestorDeDatosGlobales(){
    }

    /**
     * Obtiene el gestor de datos globales de la aplicación
     * @return
     */
    public static GestorDeDatosGlobales getInstacia() {
        if(instacia == null)
```

```
            instacia = new GestorDeDatosGlobales();
        return instacia;
    }

    /**
     * Establece la persona que está usando la aplicación
     * @param usuario usuario de la aplicación
     */
    public void setUsuario(Persona usuario) {
        this.usuario = usuario;
    }

    /**
     * Obtiene el usuario actual de la aplicación
     * @return usuario de la aplicación
     */
    public Persona getUsuario() {
        return usuario;
    }

}
```

Clase Main

Esta es la clase principal del sistema, es decir la que se ejecuta y toma el control cuando el sistema inicia.

```
package org.uaccess;

import javax.swing.UIDefaults;
import javax.swing.UIManager;
import org.pushingpixels.lafwidget.LafWidget;
import org.pushingpixels.substance.api.SubstanceConstants;
import org.pushingpixels.substance.api.SubstanceLookAndFeel;
import org.pushingpixels.substance.api.fonts.FontPolicy;
import org.pushingpixels.substance.api.fonts.FontSet;
import org.pushingpixels.substance.api.skin.CremeSkin;
import org.uaccess.gui.JpnIngresar;
import org.uaccess.gui.Principal;
import org.uaccess.persistencia.AdministradorPersistencia;
import org.uaccess.utilidades.CargadorDeReportes;
import org.uaccess.utilidades.validacion.Validador;
import org.uaccess.utilidades.Mensajero;
import org.uaccess.utilidades.controles.Dialogo;
```

```
/**
 * Clase principal de la aplicación
 * @author victor
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                establecerTema();
                JpnIngresar panelDeIngreso = new JpnIngresar();
                Dialogo login = new Dialogo(panelDeIngreso, "UAccess",
                        panelDeIngreso.getBotonPorDefecto(), getClass()
                        .getResource("/org/uaccess/recursos/imagenes/"
                        + "icono.png"));

                Mensajero.getInstancia().setPadre(login);
                login.setVisible(true);
                GestorDeDatosGlobales.getInstacia().setUsuario(
                        panelDeIngreso.getUsuario());
                if(GestorDeDatosGlobales.getInstacia()
                        .getUsuario() != null){
                    Principal formularioPrincipal;
                    Validador.getInstancia().agregarXML(
                            "/META-INF/oval.xml");
                    formularioPrincipal = new Principal();
                    Mensajero.getInstancia().setPadre(
                            formularioPrincipal);
                    formularioPrincipal.setVisible(true);
                    CargadorDeReportes.getInstancia().setConexion(
                            AdministradorPersistencia.getInstancia()
                            .getConexion());
                }
            }
        });
    }

    /**
     * Establece el tema (Look And Feel) de la aplicación
     */
    public static void establecerTema() {
        SubstanceLookAndFeel.setSkin(new CremeSkin());

        final FontSet substanceCoreFontSet = SubstanceLookAndFeel
                .getFontPolicy().getFontSet("Substance", null);
        FontPolicy fontPolicy = new FontPolicy() {
            @Override
```

```
            public FontSet getFontSet(String arg0, UIDefaults arg1) {
                return substanceCoreFontSet;
            }
        };

        SubstanceLookAndFeel.setFontPolicy(fontPolicy);
        UIManager.put(LafWidget.TEXT_FLIP_SELECT_ON_ESCAPE, true);
        UIManager.put(LafWidget.TEXT_SELECT_ON_FOCUS, true);
        UIManager.put(LafWidget.TEXT_EDIT_CONTEXT_MENU, true);
        UIManager.put(LafWidget.HAS_LOCK_ICON, true);
        UIManager.put(SubstanceLookAndFeel.SHOW_EXTRA_WIDGETS, true);
        UIManager.put(SubstanceLookAndFeel
                .TABBED_PANE_CONTENT_BORDER_KIND, SubstanceConstants
                .TabContentPaneBorderKind.SINGLE_PLACEMENT);
        UIManager.put(SubstanceLookAndFeel
                .TABBED_PANE_CONTENT_BORDER_KIND, SubstanceConstants
                .TabContentPaneBorderKind.SINGLE_PLACEMENT);
    }

}
```

Archivos de configuración

Esta sección define los archivos de configuración contenidos en la aplicación y que enmarcan su comportamiento.

Oval

El fichero "oval.xml" es el que define las reglas de validación de objetos para Oval.

```
<?xml version="1.0" ?>
<oval
  xmlns="http://oval.sf.net/oval-configuration"
```

```
  xmlns:xsi="http://http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://oval.sf.net/oval-configuration
  http://oval.sourceforge.net/oval-configuration.xsd"
>
    <class type="org.uaccess.persistencia.TipoDeUsuario"
    overwrite="false">
        <field name="nombre">
            <notNull/>
            <notBlank/>
        </field>
        <field name="descripcion">
            <notNull/>
            <notBlank/>
        </field>
    </class>
    <class type="org.uaccess.persistencia.TipoDeDocumento"
    overwrite="false">
        <field name="nombre">
            <notNull/>
            <notBlank/>
        </field>
        <field name="descripcion">
            <notNull/>
            <notBlank/>
        </field>
        <field name="formato">
            <notNull/>
            <notBlank/>
        </field>
    </class>
    <class type="org.uaccess.persistencia.Persona" overwrite="false">
        <field name="apellidos">
            <notNull/>
            <notBlank/>
        </field>
        <field name="nombres">
            <notNull/>
            <notBlank/>
        </field>
        <field name="tipoDeDocumento">
            <notNull/>
            <notBlank/>
        </field>
        <field name="numeroDeDocumento">
            <notNull/>
            <notBlank/>
        </field>
        <field name="tipoDeUsuario">
            <notNull/>
            <notBlank/>
```

```
        </field>
        <field name="activo">
            <notNull/>
            <notBlank/>
        </field>
    </class>
    <class type="org.uaccess.persistencia.Huella" overwrite="false">
        <!--El ID de la persona se establece automaticamente-->
        <field name="data">
            <notNull/>
            <notBlank/>
        </field>
    </class>
    <class type="org.uaccess.persistencia.LoginAdministrativo"
    overwrite="false">
        <!--El ID de la persona se establece automaticamente-->
        <field name="login">
            <notNull/>
            <notBlank/>
        </field>
        <field name="contrasena">
            <notNull/>
            <notBlank/>
        </field>
    </class>
     <class type="org.uaccess.persistencia.Zona" overwrite="false">
        <field name="nombre">
            <notNull/>
            <notBlank/>
        </field>
        <field name="descripcion">
            <notNull/>
            <notBlank/>
        </field>
    </class>
</oval>
```

Persistencia

Este fichero contiene la información que regirá al módulo de acceso a datos y persistencia de los mismos.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="UAccessPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>org.uaccess.persistencia.Huella</class>
    <class>org.uaccess.persistencia.Persona</class>
    <class>org.uaccess.persistencia.Acceso</class>
    <class>org.uaccess.persistencia.TipoDeDocumento</class>
    <class>org.uaccess.persistencia.TipoDeUsuario</class>
    <class>org.uaccess.persistencia.Zona</class>
    <class>org.uaccess.persistencia.LoginAdministrativo</class>
    <class>org.uaccess.persistencia.PrivilegioAdministrativo</class>
    <shared-cache-mode>ALL</shared-cache-mode>
    <properties>
      <property name="javax.persistence.jdbc.url"
      value="jdbc:mysql://localhost:3306/uaccess"/>
      <property name="javax.persistence.jdbc.password"
      value="biodev"/>
      <property name="javax.persistence.jdbc.driver"
      value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.user"
      value="biodev"/>
    </properties>
  </persistence-unit>
</persistence>
```

4.3 DESARROLLO DEL SOFTWARE DEL PUNTO DE ACCESO

En esta sección se tratan los aspectos referentes al programa del punto de acceso peatonal, y antes de abordar sus elementos, definiremos los aspectos generales respecto a su desarrollo.

El programa de control de acceso peatonal ha sido desarrollado utilizando C++, así como las librerías de Qt4 para la interfaz gráfica y otros aspectos como el acceso a datos y el multiprocesamiento.

La versión específica de Qt utilizada, ha sido la 4.6.3; y como entorno de desarrollo se ha hecho uso de Qt Creator, en su versión 1.3.1.

Elementos de funcionalidad y lógica

Los elementos aquí descritos son los que conforman el núcleo de la aplicación y definen su lógica de funcionamiento.

Clase ConexionBD

Esta clase define la información para la conexión a la base de datos del sistema.

Archivo conexionbd.h

```
#ifndef CONEXIONBD_H
#define CONEXIONBD_H
#include <QString>
#include <QtSql>

/**
 * Gestor de la conexión a la base de datos.
 *
 * Esta clase trabaja con el objeto QSqlDatabase de la aplicación.
 * Proporciona funcionalidad para establecer los parámetros de la
 * conexión y conectarse a la base de datos.
 */
class ConexionBD
{
public:
    /**
     * Crea un nuevo objeto para la administración de la conexión
     * a la base de datos sin parámetros establecidos.
     */
    ConexionBD();
    /**
     * Crea un nuevo objeto para la administración de la conexión
     * a la base a partir de los parámetros recibidos.
     * @param nombreHost Nombre del host servidor de la base de datos.
     * @param nombreBD Nombre de la base de datos.
     * @param nombreUsuario Nombre del usuario de la base de datos.
     * @param contrasenaUsuario Contraseña del usuario que establece
     * la conexión.
     */
    ConexionBD(QString nombreHost, QString nombreBD,
               QString nombreUsuario, QString contrasenaUsuario);
    /**
     * Establece la conexión a la base de datos.
     * @return 'true' si la conexión se establece de manera correcta.
     * 'false' en caso contrario.
     */
    bool conectar();
private:
    /** Instancia del objeto de conexión a la base de datos */
    QSqlDatabase bd;
};
```

```
#endif // CONEXIONBD_H
```

Archivo conexionbd.cpp

```
#include "conexionbd.h"
#include <QString>

ConexionBD::ConexionBD()
{
    ConexionBD("", "", "", "");
};

ConexionBD::ConexionBD(QString nombreHost, QString nombreBD,
                       QString nombreUsuario, QString contrasenaUsuario)
{
    this->bd = QSqlDatabase::addDatabase("QMYSQL");
    this->bd.setHostName(nombreHost);
    this->bd.setDatabaseName(nombreBD);
    this->bd.setUserName(nombreUsuario);
    this->bd.setPassword(contrasenaUsuario);
}

bool ConexionBD::conectar()
{
    if(this->bd.open())
        return true;
    else
        return false;
}
```

Clase GestorTorniquete

La clase "GestorTorniquete" corresponde al objeto encargado de interactuar con el dispositivo hardware del punto de acceso peatonal.

Archivo gestortorniquete.h

```
#ifndef GESTORTORNIQUETE_H
#define GESTORTORNIQUETE_H

extern "C"
{
    #include <usb.h>
}

/**
 * Gestor del torniquete de acceso.
 *
 * Esta clase es la que define al objeto encargado de gestionar el
 * torniquete del punto de acceso peatonal. Dicho dispositivo debe
 * estar conectado a uno de los puertos USB de la computadora en la
 * que corre el software del punto de acceso. La comunicación entre
 * estos dos elementos vía USB es la gestionada por "GestorTorniquete"
 */
class GestorTorniquete
{
public:
    /**
     * Crea un nuevo gestor para el torniquete de acceso conectado a la
     * computadora a partir del VID (identificador del fabricante) y el
     * PID (identificador del producto) del dispositivo USB conectado.
     */
    GestorTorniquete(int vid, int pid);
    /**
     * Indica si el dispositivo de acceso ya ha sido encontrado en el PC.
     * @return 'true' si el dispositivo está presente. 'false' en caso
     * contrario.
     */
    bool dispositivoEstaPresente();
    /**
     * Indica si hay comunicación abierta entre el dispositivo y el PC.
     * @return 'true' si el dispositivo está abierto. 'false' en caso
     * contrario.
     */
    bool dispositivoEstaAbierto();
    /**
     * Verifica si el dispositivo está conectado al PC haciendo una
     * búsqueda por el identificador estándar USB del dispositivo
     * (combinación de VID y PID).
     * @return 'true' si el dispositivo es encontrado. 'false' en caso
     * contrario.
     */
    bool buscarDispositivo();
    /**
     * Abre la comunicación al torniquete de acceso si y solo si el
```

```
     * dispositivo se encuentra presente.
     * @return 'true' la comunicación con el dispositivo se ha abierto
     * satisfactiamente. 'false' en caso contrario.
     */
    bool abrirDispositivo();
    /**
     * Envía una orden al dispositivo si y solamente si hay comunicación
     * abierta entre este y el PC.
     * @return 'true' si los datos fuenron enviados con éxito. 'false'
     * en caso contrario.
     */
    bool enviarOrden(const char *datos);
    /**
     * Cierra la comunicación entre el PC y el torniquete de acceso.
     * @return 'true' si el dispositivo ha sido cerrado. 'false' en caso
     * contrario.
     */
    bool cerrarDispositivo();

private:
    /** VID del dispositivo de acceso peatonal */
    int vidDispositivo;
    /** PID del dispositivo de acceso peatonal */
    int pidDispositivo;
    /** Puntero al dispositivo USB encontrado */
    struct usb_device *dispositivoEncontrado;
    /** Puntero al dispositivo USB abierto */
    usb_dev_handle *dispositivoAbierto;

};

#endif // GESTORTORNIQUETE_H
```

Archivo gestortorniquete.cpp

```
#include "gestortorniquete.h"

GestorTorniquete::GestorTorniquete(int vid, int pid)
{
    vidDispositivo = vid;
    pidDispositivo = pid;
    dispositivoEncontrado = NULL;
    dispositivoAbierto = NULL;
}

bool GestorTorniquete::dispositivoEstaPresente()
{
    if(dispositivoEncontrado){
```

```
        return true;
    } else {
        return false;
    }
}

bool GestorTorniquete::dispositivoEstaAbierto()
{
    if(dispositivoAbierto){
        return true;
    } else {
        return false;
    }
}

bool GestorTorniquete::buscarDispositivo()
{
    struct usb_bus *bus;
    struct usb_device *dev;

    usb_find_busses();
    usb_find_devices();
    for (bus = usb_get_busses(); bus; bus = bus->next)
    {
        for (dev = bus->devices; dev; dev = dev->next)
        {
            if (dev->descriptor.idVendor == vidDispositivo
                && dev->descriptor.idProduct == pidDispositivo)
            {
                dispositivoEncontrado = dev;
                return true;
            }
        }
    }
    dispositivoEncontrado = NULL;
    return false;
}

bool GestorTorniquete::abrirDispositivo()
{
    cerrarDispositivo();
    if(dispositivoEncontrado){
        dispositivoAbierto = usb_open(dispositivoEncontrado);
        if(dispositivoAbierto){
            return true;
        }
    }
    return false;
}
```

```
bool GestorTorniquete::enviarOrden(const char *datos)
{
    if(dispositivoAbierto){
        if(usb_bulk_write(dispositivoAbierto, 1,
                          datos, 1, 1000) > 0){
            return true;
        }
    }
    return false;
}

bool GestorTorniquete::cerrarDispositivo()
{
    if(dispositivoAbierto){
        usb_release_interface(dispositivoAbierto, 0);
        usb_close(dispositivoAbierto);
    }
    return true;
}
```

Clase Servicio

Esta clase corresponde al servicio que se encarga del reconocimiento de usuarios y de permitir o denegar el paso a los mismos, según sea el caso. Esta clase funciona en un hilo de procesamiento diferente al del resto de la aplicación.

Archivo servicio.h

```
#ifndef SERVICIO_H
#define SERVICIO_H

#include <QThread>
#include <QSettings>
#include "conexionbd.h"
#include "gestortorniquete.h"

extern "C"
{
    #include <libfprint/fprint.h>
```

```
}

/**
 * Servicio de reconocimiento de usuarios.
 *
 * Esta clase define al servicio de reconocimiento de usuarios
 * en base a su captura dactilar. Es servicio de reconocimiento
 * biométrico funciona en un hilo de procesamiento distinto al
 * del resto de la aplicación y utiliza librerías nativas
 * libfprint y libusb. El servicio de autenticación incluye
 * además el desbloqueo del dispositivo de acceso para los
 * usuarios aceptados. Si las configuraciones del sistema no
 * pueden ser leídas, así como si el lector de huellas
 * dactilares o el dispositivo de acceso no se encuentran
 * disponibles el proceso se detiene.\n
 * El servicio no maneja elementos de interfaz gráfica. Lo
 * que sucede durante el que hacer del mismo es informado al
 * hilo de procesamiento principal del programa por medio de
 * señales desencadenadas a la ocurrencia de cambios de estado
 * u eventos relevantes; de esta manera, es el proceso principal
 * de la aplicación, a través de los componentes de interfaz
 * gráfica que corren sobre él, el encargado de mostrar la
 * ocurrencia de los diferentes acontecimientos.
 */
class Servicio : public QThread
{
    Q_OBJECT

public:
    /**
     * Crea el hilo de procesamiento encargado de brindar el
     * servicio de verificación de usuarios y de bloquear o
     * aceptar el paso de los mismos, según sea el caso.
     */
    Servicio();
    /**
     * Detiene el servicio de autenticación de usuarios.
     */
    void stop();
    /**
     * Almacena el identificador de la zona a la que corresponde
     * el punto de acceso peatonal.
     */
    int idZona;
    /**
     * Almacena el identificador del dispositivo de captura
     * dactilar utilizado por el proceso.
     */
    int idScanner;
    /**
```

```
     * Almacena el tiempo (en milisegundos) que debe transcurrir
     * entre la aceptación de un usuario y la reanudación del
     * servicio.
     */
    int postAceptacion;
    /**
     * Almacena el tiempo (en milisegundos) que debe transcurrir
     * entre la denegación de un usuario
     * y la reanudación del servicio.
     */
    int postDenegacion;
    /**
     * Puntero al dispositivo de captura dactilar utilizado por
     * el programa.
     */
    struct fp_dscv_dev *scanner;


protected:
    /**
     * Inicia el servicio de autenticación de usuarios.
     */
    void run();

private:
    /**
     * Bandera que indica si el servicio debe detenerse.
     */
    volatile bool stopped;
    /**
     * Puntero al gestor del dispositivo de acceso.
     */
    GestorTorniquete *gestorTorniquete;
    /**
     * Método encargado de leer las configuraciones del punto
     * de acceso peatonal y cargar los valores en las variables
     * correspondientes.
     * @return 'true' si las configuraciones se han cargado
     * correctamente. 'false' en caso contrario.
     */
    bool leerConfiguraciones();
    /**
     * Busca el dispositivo de captura de huellas dactilares
     * cuyo identificador corresponda al de las configuraciones.
     * @return 'true' si el dispositivo es encontrado.
     * 'false' en caso contrario.
     */
    bool buscarDispositivo();
    /**
     * Verifica que se encuentren conectados al PC y disponibles
```

```
     * para su uso tanto el dispositivo de lectura de huellas
     * como el torniquete de acceso.
     * @return 'true' si ambos dispositivos se encuentran
     * disponibles. 'false' en otro caso.
     */
    bool verificarDispositivos();
    /**
     * Obtiene las huellas registradas en la base de datos,
     * correspondientes a usuarios activos y con permiso de
     * acceso a la zona a la que corresponde el punto de
     * acceso peatonal.
     * @return puntero a las huellas registradas.
     */
    struct fp_print_data** getHuellasRegistradas(int **personas);

signals:
    /**
     * Señal que informa la aceptación de un usuario.
     * @param id Identificador de la persona aceptada.
     * @param id_zona Identificador de la zona a la que la
     * persona ha ingresado.
     */
    void personaAceptada(int id, int id_zona);
    /**
     * Señal que informa la denegación de un usuario.
     */
    void personaDenegada();
    /**
     * Señal que informa la reanudación del servicio luedo de
     * la aceptación o denegación de una persona.
     */
    void reintentando();
    /**
     * Señal que informa la ocurrencia de un error durante la
     * ejecución del servicio
     * @param mensaje Mensaje descriptivo del error ocurrido.
     */
    void error(QString mensaje);

};

#endif // SERVICIO_H
```

Archivo servicio.cpp

```
#include "servicio.h"

Servicio::Servicio()
{
    stopped = false;
}

void Servicio::stop()
{
    stopped = true;
}

void Servicio::run()
{
    gestorTorniquete = new GestorTorniquete(0x04d8, 0x0011);

    if(fp_init() !=0){
        emit error("<font>No se ha podido inicializar el "
                   + "m&oacute;dulo de biometr&iacute;a</font>");
        stopped = false;
        return;
    }

    if(!leerConfiguraciones()){
        emit error("<font>No se han podido cargar "
                   + "las configuraciones</font>");
        stopped = false;
        return;
    }

    if(!buscarDispositivo()){
        emit error("<font>El dispositivo de lectura "
                   + "dactilar no ha sido encontrado</font>");
        stopped = false;
        return;
    }

    fp_dev *dispositivo = fp_dev_open(scanner);
    int *idsPersonas;
    while(!stopped && verificarDispositivos()){
        size_t indice = -1;
        int resultado = fp_identify_finger(dispositivo,
                        getHuellasRegistradas(&idsPersonas), &indice);
        switch(resultado){
        case FP_VERIFY_MATCH:
            char orden;
            emit personaAceptada(idsPersonas[(int)indice], idZona);
            orden = 1 &0xff;
```

```
            gestorTorniquete->enviarOrden(&orden);
            msleep(500);
            orden = 2 &0xff;
            gestorTorniquete->enviarOrden(&orden);
            msleep(postAceptacion - 1000);
            emit reintentando();
            msleep(500);
            break;
        case FP_VERIFY_NO_MATCH:
            emit personaDenegada();
            msleep(postDenegacion - 500);
            emit reintentando();
            msleep(500);
            break;
        default:
            emit reintentando();
        }
    }
    fp_dev_close(dispositivo);
    gestorTorniquete->cerrarDispositivo();
    stopped = false;
}

bool Servicio::leerConfiguraciones()
{
    QSettings configuraciones("UESOcc", "UAccess");
    idZona = configuraciones.value("id_zona", -1).toInt();
    idScanner = configuraciones.value("scanner", -1).toInt();
    postAceptacion = configuraciones.value("esp_acp", -1).toInt();
    postDenegacion = configuraciones.value("esp_dng", -1).toInt();
    if(idZona > 0 && idScanner > 0
       && postAceptacion > 0 && postDenegacion > 0){
        return true;
    } else{
        return false;
    }
}

bool Servicio::buscarDispositivo()
{
    fp_dscv_dev** dispositivos = fp_discover_devs();
    for(int i=0; dispositivos[i]; i++){
        if(fp_dscv_dev_get_driver_id(dispositivos[i])
            == idScanner){
            scanner = dispositivos[i];
            return true;
        }
    }
    return false;
}
```

```
bool Servicio::verificarDispositivos()
{
    if(!buscarDispositivo()){
        emit error("<font>El dispositivo de lectura dactilar "
                   + "no ha sido encontrado</font>");
        return false;
    }
    if(gestorTorniquete->buscarDispositivo()){
        if(!gestorTorniquete->abrirDispositivo()){
            emit error("<font>No se ha podido establecer "
                       + "comunicación con el torniquete</font>");
            return false;
        }
    } else {
        emit error("<font>No se ha encontrado el "
                   + "torniquete de acceso</font>");
        return false;
    }
    return true;
}

struct fp_print_data** Servicio::getHuellasRegistradas(
                                int **personas)
{
    QString consulta;
    consulta = "SELECT id_persona, data ";
    consulta += "FROM huellas h ";
    consulta += "INNER JOIN personas p ON h.id_persona = p.id ";
    consulta += "WHERE p.activo = 1 ";
    consulta += "AND h.id_persona ";
    consulta += "IN (SELECT p.id ";
    consulta += "FROM personas p ";
    consulta += "WHERE p.id_tipo_de_usuario IN (";
    consulta += "SELECT a.id_tipo_de_usuario FROM ";
    consulta += "permisos_de_acceso a WHERE a.id_zona = ";
    consulta += QString::number(idZona) + "))";

    QSqlQuery gestorConsulta;

    gestorConsulta.exec(consulta);


    *personas = (int*) malloc(sizeof(int)
                              * gestorConsulta.size());
    fp_print_data **huellas = (fp_print_data **)
                              malloc(sizeof(*huellas) *
                                     (gestorConsulta.size() + 1));
    int h = 0;
```

```
    while(gestorConsulta.next()){
        QByteArray bytes;
        int tamanio;
        unsigned char *datos;
        struct fp_print_data *datosHuella = NULL;
        (*personas)[h] = gestorConsulta.value(0).toInt();
        bytes = gestorConsulta.value(1).toByteArray();
        tamanio = 2414;//bytes.size();
        datos = new unsigned char[tamanio];
        for(int i = 0; i<tamanio; i++)
            datos[i] = (unsigned char)bytes.at(i);
        datosHuella = fp_print_data_from_data(datos, tamanio);
        huellas[h] = datosHuella;
        h++;
    }
    huellas[h] = NULL;

    return huellas;
}
```

Elementos de interfaz de usuario

En esta sección se encuentran definidos los elementos de interfaz de usuario del programa de control del punto de acceso peatonal.

Formulario Configuraciones

Este formulario es el diseñado para establecer las configuraciones propias del punto de acceso peatonal.

Archivo configuraciones.h

```
#ifndef CONFIGURACIONES_H
#define CONFIGURACIONES_H

#include <QDialog>
#include <QSettings>

extern "C"
{
    #include <libfprint/fprint.h>
}

namespace Ui {
    class configuraciones;
}

/**
 * Formulario diseñado para la edición de configuraciones.
 *
 * El formulario al que corresponde esta clase es el que permite
 * establecer las configuraciones del punto de acceso peatonal.
 */
class Configuraciones : public QDialog {
    Q_OBJECT
public:
    /**
     * Crea una nueva ventana para la administración de las
     * configuraciones del punto de acceso peatonal.
     */
    Configuraciones(QWidget *parent = 0);
    /**
     * Destruye la ventana y libera la memoria utilizada por la misma.
     */
    ~Configuraciones();

protected:
    /**
     * Controlador de cambios de estado de Qt
     */
    void changeEvent(QEvent *e);
    /**
     * Controlador del evento desencadenado cuando el formulario es
     * mostrado. A la ocurrencia de este evento se establecen el
     * contenido y valor inicial de los controles.
     */
    void showEvent(QShowEvent *);

private:
    /**
```

```
     * Puntero al diseño de la interfaz gráfica del formulario.
     */
    Ui::configuraciones *ui;
    /**
     * Carga las zonas de la base de datos y las muestra en la
     * lista desplegable correspondiente.
     */
    void mostrarZonas();
    /**
     * Carga los dispositivos de captura dactilar descubiertos por el API
     * de fprint y los muestra en la lista desplegable correspondiente.
     */
    void mostrarDispositivos();
    /**
     * Lee las configuraciones del punto de acceso peatonal y las carga
     * en la interfaz de usuario. Si el archivo de configuración o alguno
     * de sus elementos no es encontrado, se establecen valores por
     * defecto en la interfaz de usuario.
     */
    void leerConfiguraciones();
    /**
     * Efectúa la validación de las configuraciones establecidas por el
     * usuario en la interfaz gráfica.
     * @param infraccion Referencia a una cadena de caracteres en la que
     * se escribirá la infracción encontrada en caso de existir.
     * @return 'true' si las configuraciones cumplen con los criterios de
     * validación. 'false' en caso contrario.
     */
    bool validarConfiguraciones(QString &infraccion);
    /**
     * Guarda las configuraciones para el punto de acceso peatonal que
     * han sido seleccionadas por el usuario.
     */
    void guardarConfiguraciones();

public slots:
    /**
     * Evento que indica que el usuario ha terminado de editar las
     * configuraciones y desea guardarlas. A la ocurrencia de este
     * evento se validan las configuraciones seleccionadas y si cumplen
     * con los criterios de validación se guardan y cierra la ventana;
     * si no cumplen, se informa al usuario sobre las infracciones
     * encontradas y la ventana sigue abierta.
     */
    void on_btnAceptar_clicked();
    /**
     * Evento que indica que el usuario desea cancelar la edición de
     * las configuraciones. A la ocurrencia de este evento se cierra
     * la ventana ignorando los cambios realizados.
     */
```

```
    void on_btnCancelar_clicked();
};

#endif // CONFIGURACIONES_H
```

Archivo configuraciones.cpp

```
#include "configuraciones.h"
#include "ui configuraciones.h"
#include <QtSql>
#include <QMessageBox>

Configuraciones::Configuraciones(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::configuraciones)
{
    ui->setupUi(this);
}

Configuraciones::~Configuraciones()
{
    delete ui;
}

void Configuraciones::changeEvent(QEvent *e)
{
    QDialog::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

void Configuraciones::showEvent(QShowEvent *)
{
    mostrarZonas();
    mostrarDispositivos();
    leerConfiguraciones();
}

void Configuraciones::mostrarZonas()
{
    QSqlQuery consultaZonas("SELECT * FROM zonas");
    consultaZonas.exec();
    ui->cmbZona->clear();
```

```
    while(consultaZonas.next()){
        ui->cmbZona->addItem(consultaZonas.value(1).toString(),
                             consultaZonas.value(0).toInt());
    }
}

void Configuraciones::mostrarDispositivos()
{
    fp_dscv_dev **dispositivos = fp_discover_devs();
    ui->cmbLector->clear();
    for(int i=0; dispositivos[i]; i++){
        fp_driver *controlador = fp_dscv_dev_get_driver(dispositivos[i]);
        ui->cmbLector->addItem(QString(fp_driver_get_full_name(
                controlador)), fp_driver_get_driver_id(controlador));
    }
}

void Configuraciones::leerConfiguraciones()
{
    QSettings configuraciones("UESOcc", "UAccess");
    if(configuraciones.value("id_zona", -1) != -1)
        ui->cmbZona->setCurrentIndex(ui->cmbZona->findData(
                configuraciones.value("id_zona")));
    if(configuraciones.value("scanner", -1) != -1)
        ui->cmbLector->setCurrentIndex(ui->cmbLector->findData(
                configuraciones.value("scanner")));
    if(configuraciones.value("esp_acp", -1) != -1)
        ui->spbTiempoPostAceptacion->setValue(
                configuraciones.value("esp_acp").toInt());
    if(configuraciones.value("esp_dng", -1) != -1)
        ui->spbTiempoPostDenegacion->setValue(
                configuraciones.value("esp_dng").toInt());
}

bool Configuraciones::validarConfiguraciones(QString &infraccion)
{
    if(ui->cmbZona->currentIndex() == -1){
        infraccion = "No hay ninguna zona por guardar";
        return false;
    }
    if(ui->cmbLector->currentIndex() == -1){
        infraccion =
                "<font>No hay ning&uacute;n lector por guardar</font>";
        return false;
    }
    return true;
}

void Configuraciones::guardarConfiguraciones()
{
```

```
    QSettings configuraciones("UESOcc", "UAccess");
    configuraciones.setValue("id_zona", ui->cmbZona->itemData(
            ui->cmbZona->currentIndex()));
    configuraciones.setValue("scanner", ui->cmbLector->itemData(
            ui->cmbLector->currentIndex()));
    configuraciones.setValue("esp_acp",
                             ui->spbTiempoPostAceptacion->value());
    configuraciones.setValue("esp_dng",
                             ui->spbTiempoPostDenegacion->value());
}

//EVENTOS

void Configuraciones::on_btnAceptar_clicked()
{
    QString error;
    QMessageBox *mensaje = new QMessageBox(this);
    if(validarConfiguraciones(error)){
        guardarConfiguraciones();
        this->close();
    } else{
        mensaje->setText(error);
        mensaje->show();
    }

}

void Configuraciones::on_btnCancelar_clicked()
{
    this->close();
}
```

Formulario Login

El formulario correspondiente a esta clase es el utilizado para la autenticación de usuarios del sistema.

Archivo login.h

```
#ifndef LOGIN_H
#define LOGIN_H
```

```
#include <QDialog>

namespace Ui {
    class Login;
}

/**
 * Formulario diseñado para la verificación de usuarios del sistema.
 *
 * El formulario al que corresponde esta clase es el que permite
 * autenticar los usuarios del sistema a partir de un privilegio
 * administrativo indicado para tales fines.
 */
class Login : public QDialog {
    Q_OBJECT
public:
    /**
     * Crea una nueva ventana destinada a la autenticación de usuarios.
     */
    Login(QWidget *parent = 0);
    /**
     * Destruye la ventana y libera la memoria utilizada por la misma.
     */
    ~Login();
    /**
     * Muestra el formulario y ejecuta la autenticación del usuario.
     * @return 'true' si el usuario es válido. 'false' en caso contrario.
     */
    bool autenticarUsuario(QString privilegioRequerido);

protected:
    /**
     * Controlador de cambios de estado de Qt
     */
    void changeEvent(QEvent *e);
    /**
     * Controlador del evento desencadenado cuando el formulario es
     * mostrado. A la ocurrencia de este evento se establece el valor
     * inicial de los controles.
     */
    void showEvent(QShowEvent *);

private:
    /**
     * Puntero al diseño de la interfaz gráfica del formulario.
     */
    Ui::Login *ui;
    /**
     * Privilegio administrativo que requiere el usuario para ser
     * autenticado.
```

```
     */
    QString privilegioVerificado;
    /**
     * Almacena el resultado de la última autenticación realizada.
     * 'true' indica que la última autenticación se realizó por un
     * usuario válido. 'false' indica que el último usuario en
     * logearse no era válido.
     */
    bool ultimoResultado;
    /**
     * Verifica el usuario en la base de datos.
     */
    bool verificarUsuarioEnBD();
public slots:
    /**
     * Evento que indica que el usuario ha ingresado sus datos y
     * desea autenticarse. A la ocurrencia de este evento se
     * verifica la autenticidad del usuario; si el usuario es
     * válido establece el último resultado a 'true', si no a 'false'.
     * En ambos casos se cierra la ventana y en el caso particular
     * de que el usuario no sea válido se muestra un mensaje de
     * error antes de dicha acción.
     */
    void on_btnAceptar_clicked();
    /**
     * Evento que indica que el usuario desea cancelar el proceso
     * de autenticación.
     * A la ocurrencia de este evento se cierra la ventana.
     */
    void on_btnCancelar_clicked();
};

#endif // LOGIN_H
```

Archivo login.cpp

```
#include "login.h"
#include "ui_login.h"
#include <QtSql>
#include <QMessageBox>

Login::Login(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Login)
{
    ui->setupUi(this);
}
```

```
Login::~Login()
{
    delete ui;
}

void Login::changeEvent(QEvent *e)
{
    QDialog::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

void Login::showEvent(QShowEvent *)
{
    ultimoResultado = false;
    ui->txtUsuario->setText("");
    ui->txtContrasena->setText("");
    ui->txtUsuario->setFocus();
}

bool Login::autenticarUsuario(QString privilegioRequerido)
{
    this->privilegioVerificado = privilegioRequerido;
    this->exec();
    return this->ultimoResultado;
}

bool Login::verificarUsuarioEnBD()
{
    QString hashContrasenia = QString(QCryptographicHash::hash(
            ui->txtContrasena->text().toUtf8(),
            QCryptographicHash::Md5).toHex());
    QString consulta;
    consulta = "SELECT id_persona ";
    consulta += "FROM logins_administrativos ";
    consulta += "WHERE (login = '" + ui->txtUsuario->text();
    consulta += "' AND contrasena = '" + hashContrasenia + "') ";
    consulta += "AND id_persona IN (";
    consulta += "SELECT id FROM personas WHERE id_tipo_de_usuario IN (";
    consulta += "SELECT id FROM tipos_de_usuario WHERE id IN (";
    consulta += "SELECT id_tipo_de_usuario ";
    consulta += "FROM permisos_administrativos ";
    consulta += "WHERE id_privilegio_administrativo = (";
    consulta += "SELECT id FROM privilegios_administrativos ";
    consulta += "WHERE nombre = '" + privilegioVerificado + "' ";
```

```
    consulta += "))))";
    QSqlQuery gestorConsulta;
    gestorConsulta.exec(consulta);
    if(gestorConsulta.size() == 1)
        return true;
    else
        return false;
}

//EVENTOS

void Login::on_btnAceptar_clicked()
{
    QMessageBox *mensaje = new QMessageBox(this);
    ultimoResultado = verificarUsuarioEnBD();
    if(ultimoResultado == false){
        mensaje->setText("<font>Usuario no v&aacute;lido</font>");
        mensaje->show();
    }
    this->close();
}

void Login::on_btnCancelar_clicked()
{
    this->close();
}
```

Clase Ventana

Esta clase define a la ventana principal del programa de control de acceso.

Archivo ventana.h

```
#ifndef VENTANA_H
#define VENTANA_H

#include <QWidget>
#include <QCloseEvent>
#include "servicio.h"
#include "login.h"
#include "configuraciones.h"
```

```
extern "C"
{
    #include <libfprint/fprint.h>
}

namespace Ui {
    class Ventana;
}

/**
 * Formulario principal del punto de acceso peatonal.
 *
 * El formulario al que corresponde esta clase es el que será
 * mostrado casi la totalidad del tiempo en la pantalla del punto
 * de acceso peatonal. Éste es con el que las personas que
 * desean ingresar al sistema interactúan mostrando los
 * mensajes dirigidos al usuario.
 */
class Ventana : public QWidget {
    Q_OBJECT
public:
    /**
     * Crea una nueva ventana principal.
     */
    Ventana(QWidget *parent = 0);
    /**
     * Destruye la ventana y libera la memoria utilizada por
     * la misma.
     */
    ~Ventana();

protected:
    /**
     * Controlador de cambios de estado de Qt
     */
    void changeEvent(QEvent *e);
    /**
     * Controlador del evento desencadenado cuando el formulario
     * es mostrado. A la ocurrencia de este evento se inicia el
     * servicio para el reconocimiento de personas y, según sea
     * el caso, permitirles o denegarles el acceso.
     */
    void showEvent(QShowEvent *);
    /**
     * Controlador del evento desencadenado cuando el formulario
     * es cerrado. Esta operación solamente puede ser realizada
     * por un usuario autorizado, puesto que detiene el servicio
     * de reconocimiento y cierra por completo la aplicación.
     * Cuando se procede a cerrar la ventana y con ello finalizar
```

```
     * el programa, antes de realizar esta acción se ordena al
     * servicio que se detenga y se espera a que finalice con
     * las tareas en ejecución.
     */
    void closeEvent(QCloseEvent *event);
    /**
     * Ordena el cierre de la aplicación.
     * Antes de realizar esta acción se ordena al servicio que se
     * detenga y se espera a que finalice con las tareas en
     * ejecución.
     */
    void salir();
    /**
     * Inicia el servicio de control de acceso peatonal.
     */
    void iniciarServicio();
    /**
     * Detiene el servicio de control de acceso
     */
    void detenerServicio();

private:
    /**
     * Puntero al diseño de la interfaz gráfica de la ventana.
     */
    Ui::Ventana *ui;
    /**
     * Instancia del servicio de control de acceso.
     */
    Servicio servicio;
    /**
     * Puntero a la instancia del diálogo para solicitar login
     * en tareas administrativas.
     */
    Login *dlgLogin;
    /**
     * Puntero a la instancia del diálogo para editar las
     * configuraciones del punto de acceso.
     */
    Configuraciones *dlgConfiguraciones;

public slots:
    /**
     * Muestra al usuario el error recibido.
     * @param error a mostrar.
     */
    void mostrarError(QString mensaje);
    /**
     * Solicita al usuario la lectura de su huella dactilar.
     */
```

```
    void solicitarHuella();
    /**
     * Informa al usuario que el acceso le ha sido denegado.
     */
    void denegarAcceso();
    /**
     * Informa al usuario que el acceso le ha sido concedido.
     * Posteriormente, almacena el ingreso del usuario en la
     * bitácora de acceso.
     */
    void permitirAcceso(int idPersona, int idZona);
    /**
     * Evento que indica que el usuario desea cerrar la aplicación.
     * A la ocurrencia de este evento se invoca el método 'salir()'.
     */
    void on_btnSalir_clicked();
    /**
     * Evento que indica que el usuario desea configurar el
     * punto de acceso. A la ocurrencia de este evento se solicita
     * al usuario sus credenciales de acceso. Si es un usuario
     * auténtico y posee los permisos para configurar el punto de
     * acceso se abre el diálogo para la edición de las
     * configuraciones. Si no, se muestra un mensaje de error y
     * se ignora la solicitud.
     */
    void on_btnConfigurar_clicked();

};

#endif // VENTANA_H
```

Archivo ventana.cpp

```
#include "ventana.h"
#include "ui_ventana.h"
#include <QMessageBox>
#include <QtSql>
#include "stdio.h"

Ventana::Ventana(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Ventana)
{
    ui->setupUi(this);
    connect(&servicio, SIGNAL(error(QString)), this,
            SLOT(mostrarError(QString)));
    connect(&servicio, SIGNAL(reintentando()), this,
            SLOT(solicitarHuella()));
```

```
    connect(&servicio, SIGNAL(personaDenegada()), this,
            SLOT(denegarAcceso()));
    connect(&servicio, SIGNAL(personaAceptada(int, int)), this,
            SLOT(permitirAcceso(int, int)));
    dlgLogin = new Login();
    dlgConfiguraciones = new Configuraciones();
}

Ventana::~Ventana()
{
    delete ui;
}

void Ventana::changeEvent(QEvent *e)
{
    QWidget::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

void Ventana::salir()
{
    if(dlgLogin->autenticarUsuario("Cierre de sist de acceso")){
        servicio.stop();
        servicio.wait();
        QApplication::quit();
    }
}

void Ventana::iniciarServicio()
{
    servicio.start();
}

void Ventana::detenerServicio()
{
    servicio.stop();
    servicio.wait();
}

void Ventana::mostrarError(QString mensaje)
{
    ui->lblMensaje->setText("<font color=black>Error: "
                            + mensaje + "</font>");
}
```

```
void Ventana::denegarAcceso()
{
    ui->lblMensaje->setText(
            "<font color=red>&iexcl;ACCESO DENEGADO!</font>");
}

void Ventana::permitirAcceso(int idPersona, int idZona)
{
    ui->lblMensaje->setText("<font color=green>"
            + "&iexcl;&iexcl;&iexcl;BIENVENIDO!!!</font>");
    QSqlQuery sentenciaDeInsercion;
    sentenciaDeInsercion.prepare("INSERT INTO bitacora_de_acceso"
            + "(id_persona, id_zona) VALUES (:id_persona, :id_zona)");
    sentenciaDeInsercion.bindValue(":id_persona", idPersona);
    sentenciaDeInsercion.bindValue(":id_zona", idZona);
    sentenciaDeInsercion.exec();
}

void Ventana::solicitarHuella()
{
    ui->lblMensaje->setText("<font color=black>"
                            + "Coloque su dedo en el lector...</font>");
}

//EVENTOS

void Ventana::showEvent(QShowEvent *)
{
    iniciarServicio();
}

void Ventana::closeEvent(QCloseEvent *event)
{
    event->ignore();
    salir();
}

void Ventana::on_btnSalir_clicked()
{
    salir();
}

void Ventana::on_btnConfigurar_clicked()
{
    if(dlgLogin->autenticarUsuario("Config de sist de acceso")){
        detenerServicio();
        dlgConfiguraciones->exec();
        iniciarServicio();
}}
```

Método de arranque

En esta sección se define únicamente el método "main", utilizado para la ejecución del software.

```
#include <QtGui/QApplication>
#include <QErrorMessage>
#include "stdio.h"
#include "ventana.h"
#include "conexionbd.h"
#include "gestortorniquete.h"

extern "C"
{
    #include <libfprint/fprint.h>
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Ventana *v = new Ventana;
    ConexionBD *conexionBD = new ConexionBD(
            "localhost", "uaccess", "biodev", "biodev");
    int r = 1;
    r = fp_init();
    if(r != 0){
        printf("Error: No se ha podido inicializar libfprint");
        exit(1);
    }
    if(!conexionBD->conectar()){
        printf("Error: No se ha podido realizar "
               + "la conexión a la base de datos");
        exit(1);
    }
    v->showFullScreen();
    r = a.exec();
    fp_exit();
    return r;
}
```

4.4 PROGRAMACIÓN DEL HARDWARE

En el capítulo anterior se abordó el diseño del dispositivo hardware a nivel de circuito. En esta sección se trata lo referente al programa contenido en el microcontrolador, que es el que lo hace funcional.

Aspectos teóricos

El PIC18F4550 es un microcontrolador multipropósito diseñado pensando en los estándares del puerto USB. El mismo necesita ser programado para funcionar de acuerdo a las necesidades particulares.

Existen librerías estándar provistas por los fabricantes de software programadores de microcontroladores, de tal forma que solamente sea necesario programar la funcionalidad deseada y no aspectos generales que bien se podrían llamar estándares.

En este caso particular, se ha utilizado el PCWHD Compiler de CCS Inc., en su versión 4.104 y sus librerías.

Codificación del dispositivo

La codificación del dispositivo se divide básicamente en dos archivos de código fuente. Los mismos se describen y muestran a continuación.

Descriptor de configuraciones

Este archivo define las configuraciones generales del dispositivo, requeridos por el estándar del Bus de Serie Universal (USB, por sus siglas en inglés).

```
#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

/////////////////////////////////////////////////////
///
///   Inicializacion del descriptor
///
/////////////////////////////////////////////////////

   #DEFINE USB_TOTAL_CONFIG_LEN      32

   char const USB_CONFIG_DESC[] = {
         USB_DESC_CONFIG_LEN,
         USB_DESC_CONFIG_TYPE,
         USB_TOTAL_CONFIG_LEN,0,
         1,
         0x01,
         0x00,
         0xC0,
         0x32,
```

```
        USB_DESC_INTERFACE_LEN,
        USB_DESC_INTERFACE_TYPE,
        0x00,
        0x00,
        2,
        0xFF,
        0xFF,
        0xFF,
        0x00,

        USB_DESC_ENDPOINT_LEN,
        USB_DESC_ENDPOINT_TYPE,
        0x81,
        0x02,
        USB_EP1_TX_SIZE & 0xFF,USB_EP1_TX_SIZE >> 8,
        0x01,

        USB_DESC_ENDPOINT_LEN,
        USB_DESC_ENDPOINT_TYPE,
        0x01,
        0x02,
        USB_EP1_RX_SIZE & 0xFF,USB_EP1_RX_SIZE >> 8,
        0x01,

};

 #define USB_NUM_HID_INTERFACES   0

 #define USB_MAX_NUM_INTERFACES   1

 const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

 #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
    #error USB_TOTAL_CONFIG_LEN not defined correctly
 #endif


 char const USB_DEVICE_DESC[] ={
       USB_DESC_DEVICE_LEN,
       0x01,
       0x10,0x01,
       0x00,
       0x00,
       0x00,
       USB_MAX_EP0_PACKET_LENGTH,
       0xD8,0x04, //VID de Microchip
       0x11,0x00,
       0x00,0x01,
       0x01,      //cadena 0
```

```
         0x02,      //cadena 1
         0x00,      //cadena 2
         USB_NUM_CONFIGURATIONS
   };


////////////////////////////////////////////////////
///
///   Cadenas de descripcion del sipositivo
///
////////////////////////////////////////////////////

const char USB_STRING_DESC_OFFSET[]={0,42,38};

#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)

char const USB_STRING_DESC[]={
   //cadena 0
         4,
         USB_DESC_STRING_TYPE,
         0xD8,0x04,
   //cadena 1
         42,
         USB_DESC_STRING_TYPE,
         'T',0,
         'o',0,
         'r',0,
         'n',0,
         'i',0,
         'q',0,
         'u',0,
         'e',0,
         't',0,
         'e',0,
         ' ',0,
         'd',0,
         'e',0,
         ' ',0,
         'a',0,
         'c',0,
         'c',0,
         'e',0,
         's',0,
         'o',0,
   //cadena 2
         38,
         USB_DESC_STRING_TYPE,
         'T',0,
         'o',0,
         'r',0,
```

```
         'n',0,
         'i',0,
         'q',0,
         'u',0,
         'e',0,
         't',0,
         'e',0,
         ' ',0,
         'U',0,
         'A',0,
         'c',0,
         'c',0,
         'e',0,
         's',0,
         's',0,

};


#ENDIF
```

Codificación del funcionamiento

Este archivo es el que define el funcionamiento del microcontrolador, basado en el descriptor de configuraciones y los ficheros ".h" multipropósito incluidos por el compilador.

```
#include <18F4550.h>
#fuses XTPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,
       USBDIV,PLL1,CPUDIV1,VREGEN,NOPBADEN
#use delay(clock=48000000)

#define USB_HID_DEVICE     FALSE
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_TX_SIZE    5
#define USB_EP1_RX_SIZE    5

#include <pic18_usb.h>
#include "usb_desc_scope.h"
#include <usb.c>
```

```
#define TUA       PIN_D2
#define SGN_ON    output_high
#define SGN_OFF   output_low


void main(void) {

   int8 comando;

   SGN_OFF(SGN);

   usb_init();

   usb_task();
   usb_wait_for_enumeration();

   while (TRUE)
   {
      if(usb_enumerated())
      {
         if (usb_kbhit(1))
         {
            usb_get_packet(1, &comando, 1);

            switch (comando)
            {
               case 1:
               SGN_ON(SGN);
               break;

               case 2:
               SGN_OFF(SGN);
               break;
            }

         }
   }
}
```

CAPÍTULO V:

MANUALES DE USUARIO

5.1 MANUAL DE USUARIO PARA EL SISTEMA ADMINISTRATIVO

Tabla de contenido

- Definiciones
- Aspectos generales
  - Ingreso al sistema
  - Ventana principal
- Administración del sistema
  - Uso de la ventana administrativa
  - Creación y edición de zonas
  - Creación y edición de tipos de usuario
  - Creación y edición de tipos de documento
  - Creación y edición de personas
- Generación de reportes
  - Bitácora de acceso
  - Informe de personas registradas
  - Informe de permisos de acceso
  - Estructura general de los reportes
  - Interfaz del visor de reportes

Definiciones

- *UAccess*: El nombre "UAccess" hace referencia al sistema de control de acceso peatonal basado en biometría creado para la Facultad Multidisciplinaria de Occidente.

- *Sistema administrativo*: Término que, al igual que "aplicación administrativa" y "módulo administrativo", se utiliza para hacer referencia a la aplicación de escritorio utilizada para la administración general del sistema UAccess.

- *Zona*: Se utiliza para referirse a una zona geográfica determinada dentro del campus, e incluso al campus mismo.

- *Tipo de usuario*: Se refiere a los tipos de usuario utilizados para clasificar las personas registradas en el sistema.

- *Tipo de documento*: Término utilizado para referirse a un tipo de documento de identidad.

- *Persona*: Hace referencia a las personas registradas en el sistema, indistintamente si éstas utilizan únicamente el sistema administrativo o son usuarios del control de acceso peatonal, o si poseen ambos privilegios o ninguno.

- *Huella dactilar*: Término utilizado, al igual que "captura dactilar" o simplemente "huella", para referirse a la información resultante del proceso de captura dactilar.

- *Login administrativo*: Se refiere a la combinación de nombre de usuario y contraseña otorgado a un usuario para tareas administrativas. También suele ser nombrado simplemente como "login".

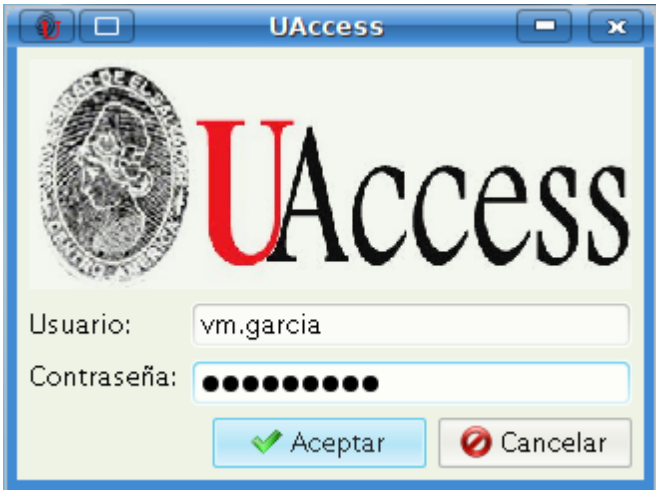
Aspectos generales

En esta sección se abordan aspectos generales sobre el uso de la aplicación administrativa del sistema "UAccess".

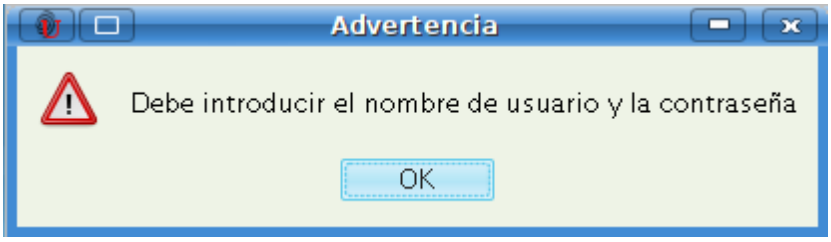
Ingreso al sistema

Para ingresar al sistema se debe poseer un login administrativo válido asignado.

Esta acción se lleva a cabo en la ventana inicial del sistema, donde debe especificarse el nombre de usuario o seudónimo del usuario que desea ingresar, así como su contraseña respectiva.



Si no ha ingresado los datos requeridos, se le mostrará un mensaje de error como el de la imagen mostrada abajo.

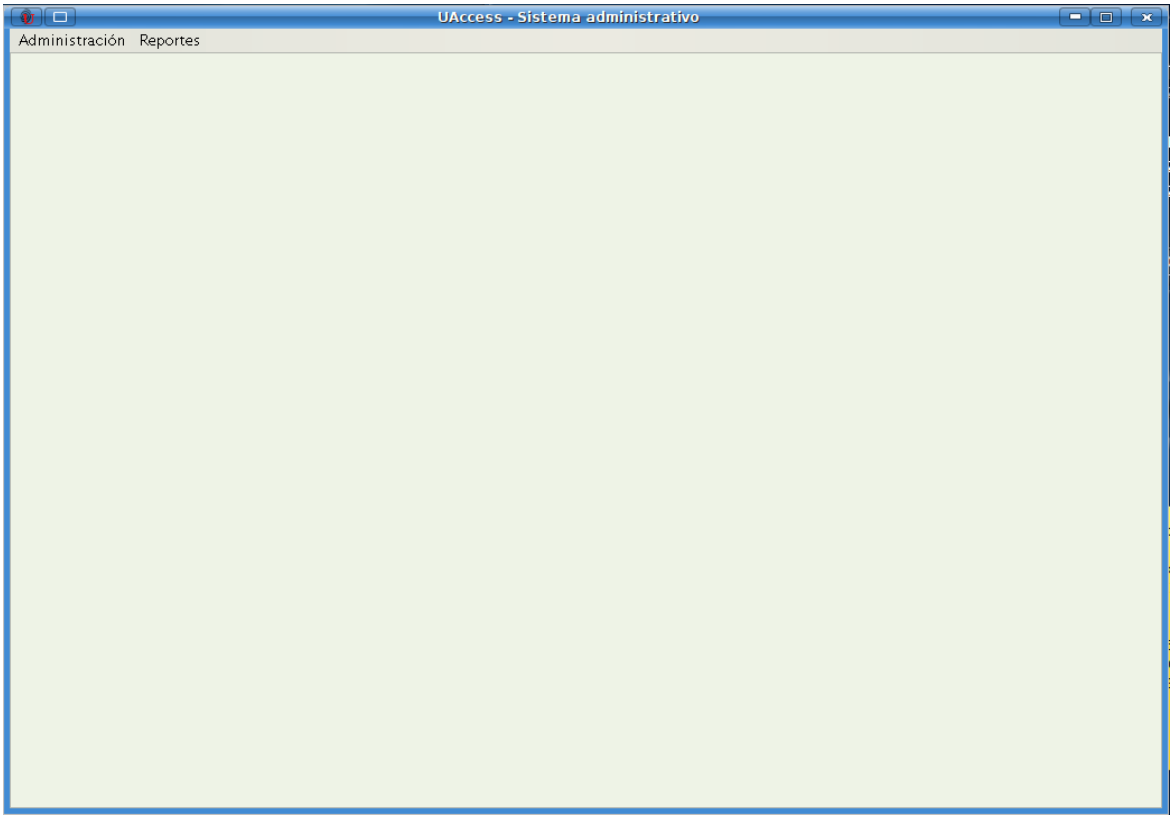


Si ya ha ingresado la información requerida, presione la tecla "Enter" o de click en el botón "Aceptar". Si el usuario es válido se le permitirá el acceso a la interfaz administrativa; caso contrario, se le denegará el acceso y se le mostrará un mensaje de error indicándoselo.

Ventana principal

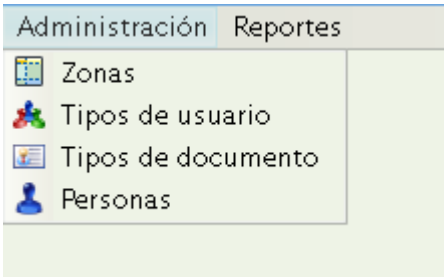
La ventana principal del programa administrativo es la que se muestra inmediatamente después del ingreso de un usuario válido del sistema.

Dicha ventana posee la barra de menús, donde son desplegadas las opciones para administración y generación de reportes. También posee un espacio de trabajo donde se muestran las ventanas internas del sistema.



Administración del sistema

En este apartado se abordan los aspectos referentes a la administración del sistema de acceso "UAccess" por medio de la aplicación administrativa. Particularmente, se enfoca en los elementos del menú "Administración" de la misma.



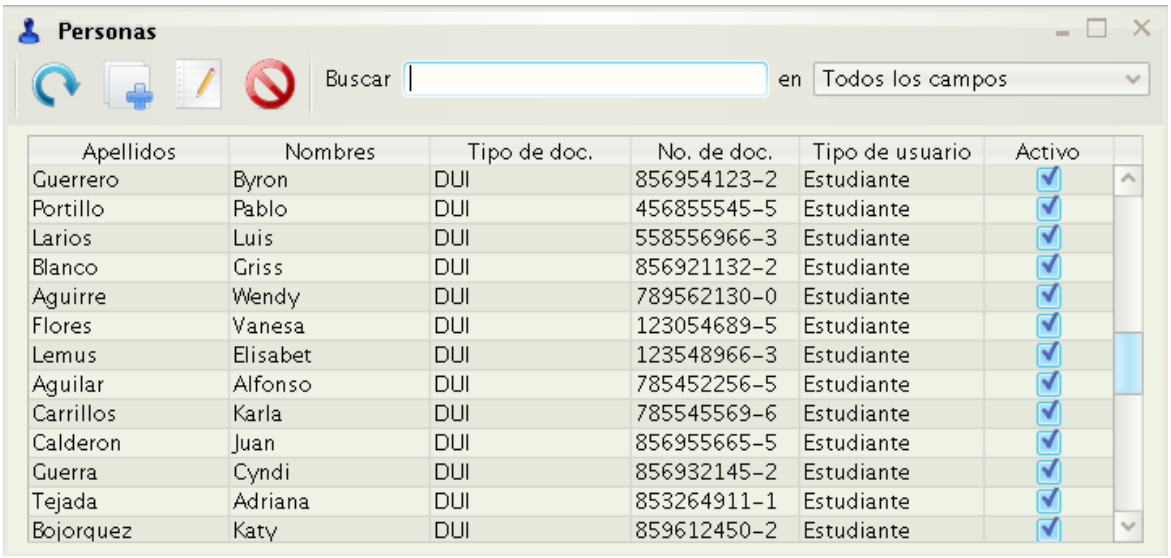
Uso de la ventana administrativa

Para la administración de los diferentes elementos del sistema se utiliza una ventana administrativa estándar que permite agregar, editar y eliminar objetos, según sea necesario.

La ventana administrativa lista los objetos respectivos mostrando sus atributos en una tabla. Puede seleccionarse un objeto seleccionando la fila en que este se encuentra contenido. La ventana posee también una barra de opciones que permite operar sobre toda la tabla acciones como refresco de

datos y búsquedas, así como acciones de carácter individual, tal es el caso de la creación, edición y eliminación de elementos.

La ventana administrativa y sus diferentes elementos pueden apreciarse en la siguiente imagen:



Apellidos	Nombres	Tipo de doc.	No. de doc.	Tipo de usuario	Activo
Guerrero	Byron	DUI	856954123-2	Estudiante	☑
Portillo	Pablo	DUI	456855545-5	Estudiante	☑
Larios	Luis	DUI	558556966-3	Estudiante	☑
Blanco	Griss	DUI	856921132-2	Estudiante	☑
Aguirre	Wendy	DUI	789562130-0	Estudiante	☑
Flores	Vanesa	DUI	123054689-5	Estudiante	☑
Lemus	Elisabet	DUI	123548966-3	Estudiante	☑
Aguilar	Alfonso	DUI	785452256-5	Estudiante	☑
Carrillos	Karla	DUI	785545569-6	Estudiante	☑
Calderon	Juan	DUI	856955665-5	Estudiante	☑
Guerra	Cyndi	DUI	856932145-2	Estudiante	☑
Tejada	Adriana	DUI	853264911-1	Estudiante	☑
Bojorquez	Katy	DUI	859612450-2	Estudiante	☑

En la imagen pueden observarse:

- Los botones de control de la ventana, los cuales permiten minimizarla, maximizarla y cerrarla.

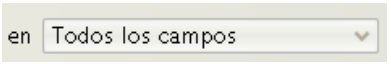
- El título de la ventana y su ícono, que se encargan de mostrar el tipo de objeto que se está administrando en la misma.

- La tabla de datos correspondiente al tipo de objeto con el que se está trabajando.

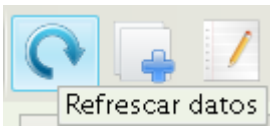
Apellidos	Nombres	Tipo de doc.	No. de doc.	Tipo de usuario	Activo
Guerrero	Byron	DUI	856954123-2	Estudiante	☑
Portillo	Pablo	DUI	456855545-5	Estudiante	☑
Larios	Luis	DUI	558556966-3	Estudiante	☑
Blanco	Griss	DUI	856921132-2	Estudiante	☑
Aguirre	Wendy	DUI	789562130-0	Estudiante	☑
Flores	Vanesa	DUI	123054689-5	Estudiante	☑
Lemus	Elisabet	DUI	123548966-3	Estudiante	☑
Aguilar	Alfonso	DUI	785452256-5	Estudiante	☑
Carrillos	Karla	DUI	785545569-6	Estudiante	☑
Calderon	Juan	DUI	856955665-5	Estudiante	☑
Guerra	Cyndi	DUI	856932145-2	Estudiante	☑
Tejada	Adriana	DUI	853264911-1	Estudiante	☑

- El campo de texto que permite establecer el filtro para la búsqueda de objetos.

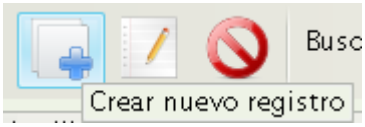
- La lista desplegable que indica el o los campos a los que se aplicará el criterio de búsqueda.



- El botón de refresco de datos, que permite obtener los datos más recientes en la tabla, dado que el sistema trabaja con una base de datos centralizada y los mismos bien podrían haber sido modificados por otro usuario desde el momento en que el formulario está siendo mostrado en pantalla.

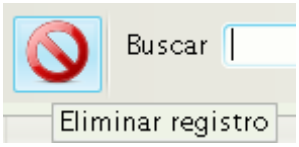


- El botón para la creación de un nuevo objeto, mismo que inicia dicho proceso mostrando una ventana secundaria para tal fin.

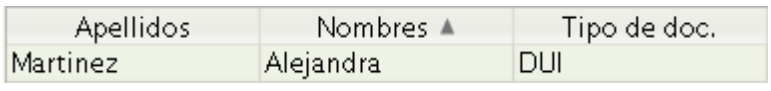


- El botón para la edición de un objeto existente, el cual permite iniciar el proceso de edición del objeto correspondiente a la fila seleccionada, en una ventana secundaria.

- El botón para la eliminación de un objeto, el cual permite eliminar el elemento correspondiente a la fila seleccionada en la tabla.



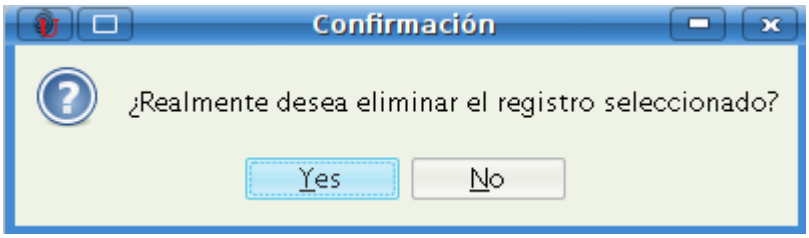
- El encabezado de las columnas de la tabla, el cual permite ordenar los objetos listados en la misma, de manera ascendente o descendente, en base al contenido de la columna respectiva.



La ventana posee además los siguientes aspectos de comportamiento frente a la interacción:

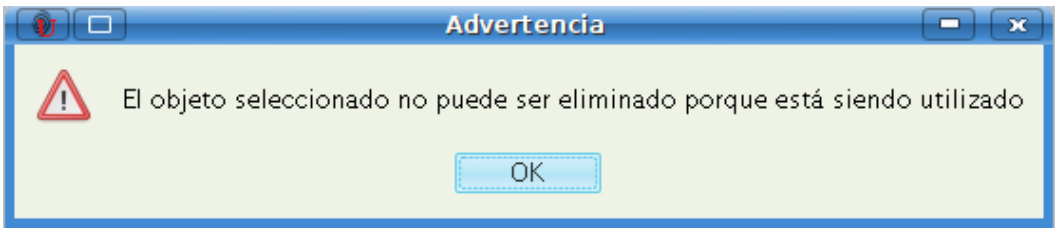
- Cuando se utilizan los botones del menú que operan sobre un objeto existente (botón "Editar" y botón "Eliminar") sin que haya un objeto seleccionado en la tabla, se recibirá un mensaje indicando dicho error de operación.

- Previo a la eliminación de un registro, aparecerá un mensaje de diálogo y la operación será realizada solamente si se responde afirmativamente a dicha solicitud de confirmación.



- Cuando se solicita la eliminación de un objeto, el sistema verifica que dicha acción no atente contra la integridad referencial de los datos (p.e., que se desee eliminar un tipo de usuario del cual existen personas registradas, una persona a la que se haga referencia en

la bitácora de acceso, etc.). Si esta es la situación, el registro no podrá ser eliminado. Se mostrará un mensaje de error indicándoselo, y si aún desea eliminarlo, deberá comunicarse con el administrador de la base de datos del sistema.



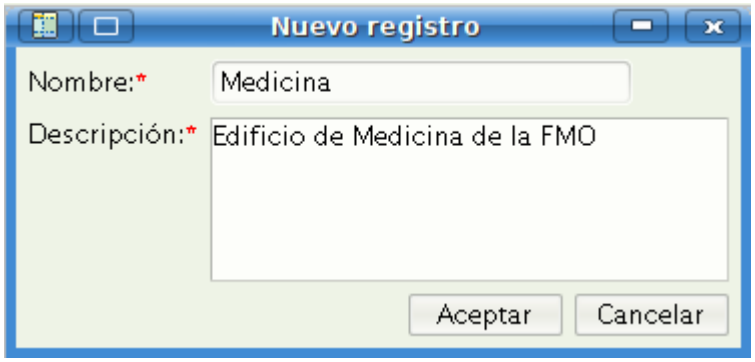
El esquema de trabajo es similar para todas las ventanas administrativas, la única diferencia significativa se encuentra en el diálogo utilizado para la creación y edición de objetos. Los formularios de dichos diálogos son abordados a continuación.

Creación y edición de zonas

Para la creación o edición de una zona geográfica, se deben especificar los siguientes atributos de la misma:

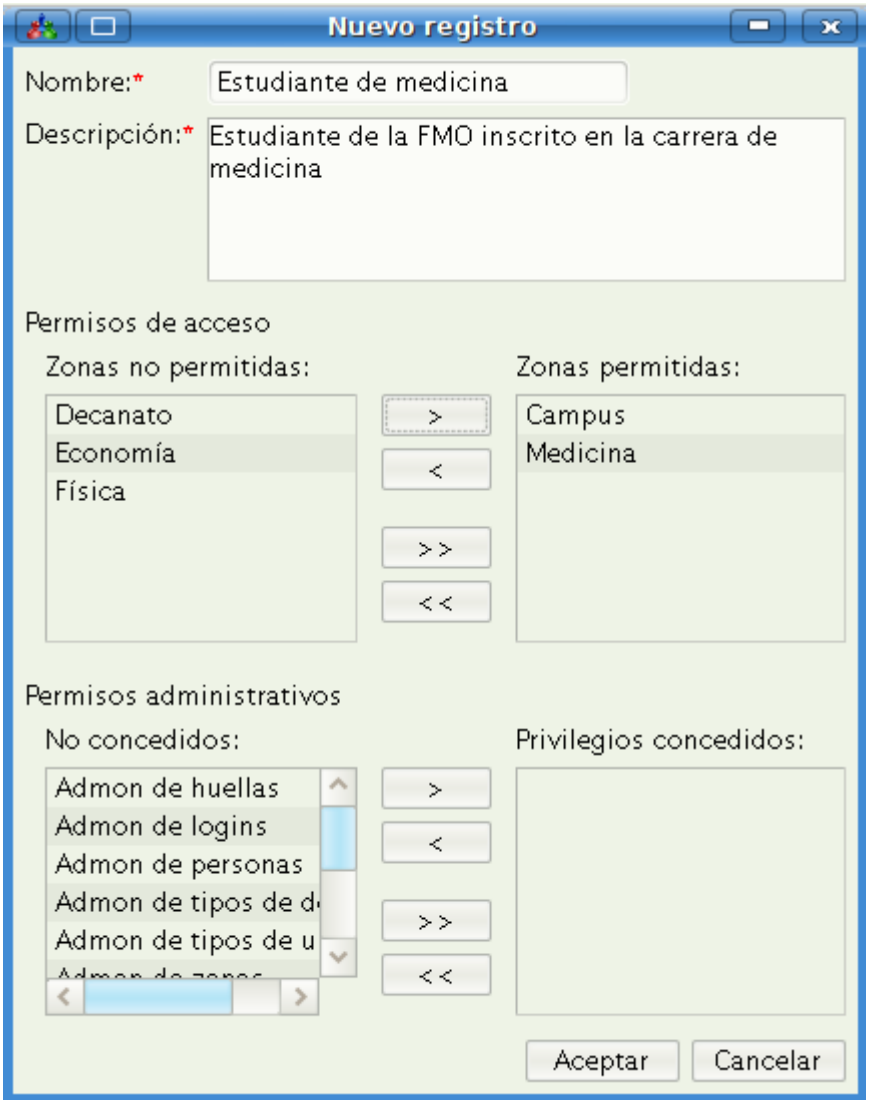
- *Nombre*: Nombre corto con el que se identificará a la zona. Este nombre debe ser único y poseer entre 1 y 25 caracteres alfanuméricos.
- *Descripción*: Descripción de la zona geográfica que permita identificarla de manera inequívoca. Esta descripción puede poseer hasta 150 caracteres alfanuméricos.

A continuación se muestra el formulario de creación y edición de zonas:



Creación y edición de tipos de usuario

Los tipos de usuario del sistema son definidos en la aplicación administrativa a partir de los parámetros notorios en la ventana:



- *Nombre*: Nombre corto con el que se identificará el tipo de usuario. El mismo no puede ser asignado a más de un tipo de usuario y estará formado por entre 1 y 25 caracteres alfanuméricos.
- *Descripción*: Definición ampliada (en comparación al nombre) del tipo de usuario que se está creando o

editando. Esta descripción puede poseer hasta 150 caracteres alfanuméricos.

- *Zonas permitidas*: Indica las zonas geográficas a las que los usuario del tipo que está siendo creado o editado tendrán acceso.
- *Permisos administrativos*: Especifica los privilegios administrativos asignados al tipo de usuario en cuestión. La asignación de estos privilegios es heredada por cada uno de los usuarios pertenecientes a dicho tipo.

Si los usuarios del tipo en cuestión no poseen acceso a ninguna zona ni ningún privilegio administrativo se muestra un mensaje de error indicándolo; dado que en este caso, aunque el tipo de usuario puede ser creado, los usuarios pertenecientes a dicho grupo no podrían ingresar a ninguna zona ni tener acceso a ningún módulo administrativo del sistema.

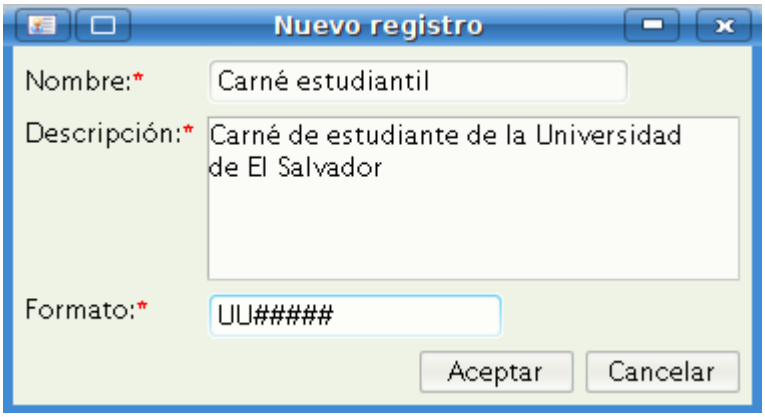
Creación y edición de tipos de documento

Los tipos de documento de identidad son creados y editados a partir de los siguientes atributos del mismo:

- *Nombre*: Define un nombre único de entre 1 y 25 caracteres alfanuméricos, con que se identificará inequívocamente el tipo de documento.
- *Descripción*: Descripción extensa (en comparación al nombre) del tipo de documento. Este campo puede poseer hasta 150 caracteres alfanuméricos.
- *Formato*: Formato que debe respetar el número de documento, cuando se especifique que el tipo de documento es el que se está agregando o editando. Dicho formato debe especificarse utilizando la siguiente connotación:

Caracter	Descripción
#	Un número.
`	Caracter de escape para utilizar los caracteres aquí listados.
U	Letra mayúscula.
L	Letra minúscula.
A	Cualquier caracter alfanumérico.
?	Cualquier caracter no numérico.
*****	Cualquier caracter.
H	Un dígito hexadecimal.

A continuación se muestra la imagen del formulario para la creación y edición de tipos de documento:



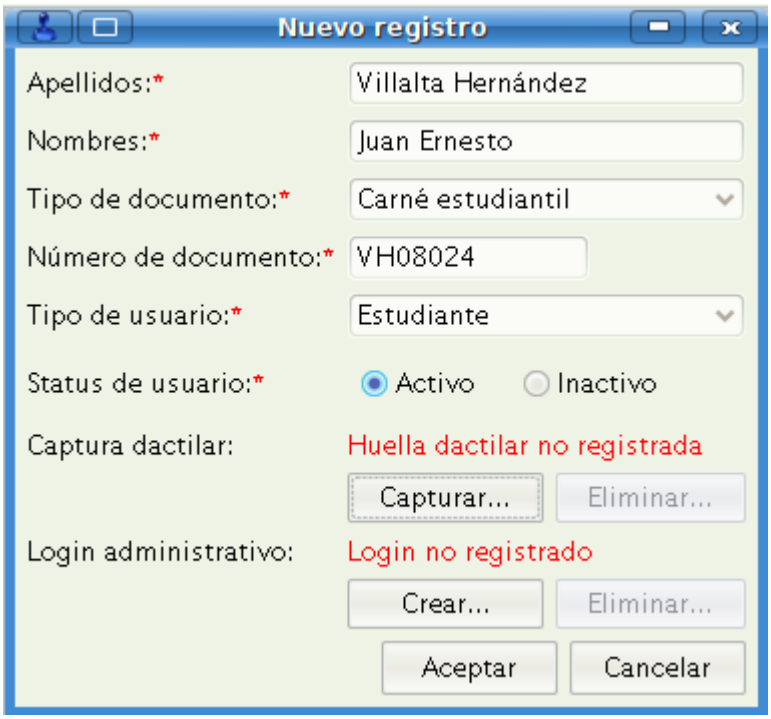
Creación y edición de personas

Las personas registradas en el sistema son creadas o editadas especificando los siguientes atributos:

- *Apellidos*: Apellidos de la persona, según su documento de identidad.
- *Nombres*: Nombres de la persona, según su documento.
- *Tipo de documento*: Tipo de documento que le identifica.
- *Número de documento*: Número del documento de identidad.
- *Tipo de usuario*: Tipo de usuario al que pertenece.
- *Status*: Estado actual del usuario para el control de acceso peatonal (activo o inactivo).

- *Captura dactilar*: Captura dactilar de la persona. Misma a través de la cual será reconocido en el sistema de acceso.
- *Login administrativo*: Login administrativo que ha sido asignado al usuario, si éste es el caso.

El formulario de creación y edición de usuarios se muestra a continuación:



El formulario está vinculado además a dos ventas de diálogo adicionales, que complementan la selección de parámetros de la persona que se está agregando o editando.

La primera, utilizada para capturar la huella dactilar, se muestra a continuación:

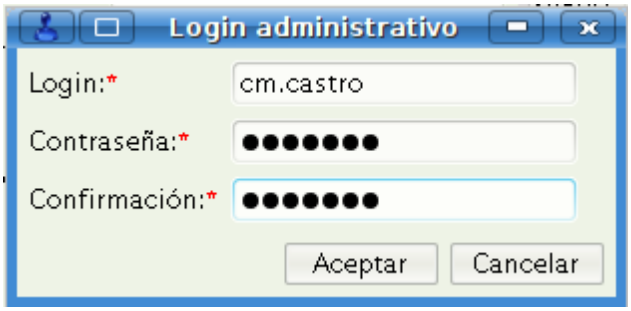
Como se puede observar, la ventana posee una lista desplegable en la que se establece el dispositivo de captura dactilar que será utilizado.

La captura se efectúa al momento de presionar el botón "Ejecutar captura". El modo de vista de la huella puede ser cambiado en la parte baja del panel, donde se puede escoger

entre “Escala de grises”, que muestra la imagen tal y como fue capturada, y “Binarizada”, que muestra la captura en blanco y negro, haciendo la huella mucho más evidente.

Se recomienda no guardar la huella sin antes haber sido visualizada en modo binarizado, puesto que así se facilita establecer un criterio sobre la nitidez con que la misma fue capturada.

La otra ventana de diálogo utilizada para completar la información de la persona es la siguiente:



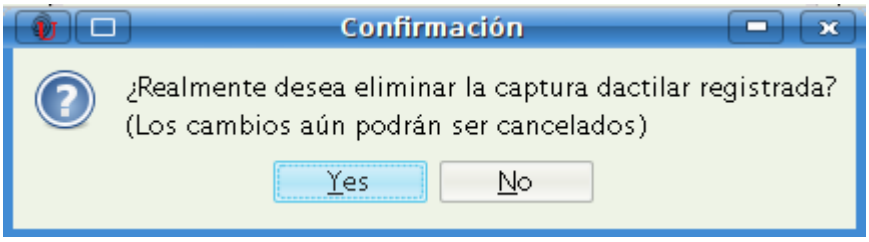
En ella se establece la información del login administrativo asignado a la persona, si es que éste sea el caso.

Tanto cuando se ha registrado la huella dactilar de la persona, como cuando se le ha asignado un login

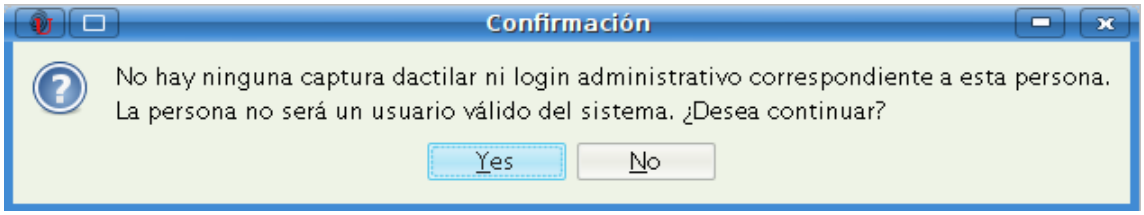
administrativo, el mensaje que aparece en el formulario general de parámetros de la misma cambia. La siguiente imagen ilustra dicho cambio.

Si la persona posee su huella registrada o un login administrativo asignado, los mismos pueden ser eliminados utilizando el botón "Eliminar" de la sección correspondiente. Esta operación debe ser confirmada, aunque será guardada hasta que se acepten los cambios hechos sobre la persona, por lo que aún podría ser revertida.

El mensaje de confirmación, que a la vez informa lo expuesto anteriormente, puede verse a continuación:



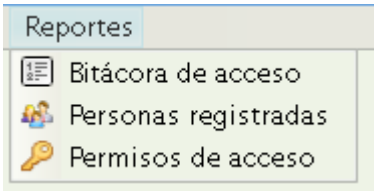
Si la persona no posee ninguna captura dactilar registrada ni un login administrativo asignado, la misma no podrá hacer uso de ninguno de los elementos del sistema. Por esta razón, si este es el caso al momento de intentar guardar la persona, se muestra la siguiente advertencia, que a su vez es una solicitud de confirmación para almacenar al usuario sin dichos elementos:



Generación de reportes

En este apartado se trata lo referente a la generación de reportes y la selección de parámetros para la realización de esta tarea.

Esta acción se inicia en el menú "Reportes", donde se debe seleccionar el reporte que se desea generar.



Bitácora de acceso

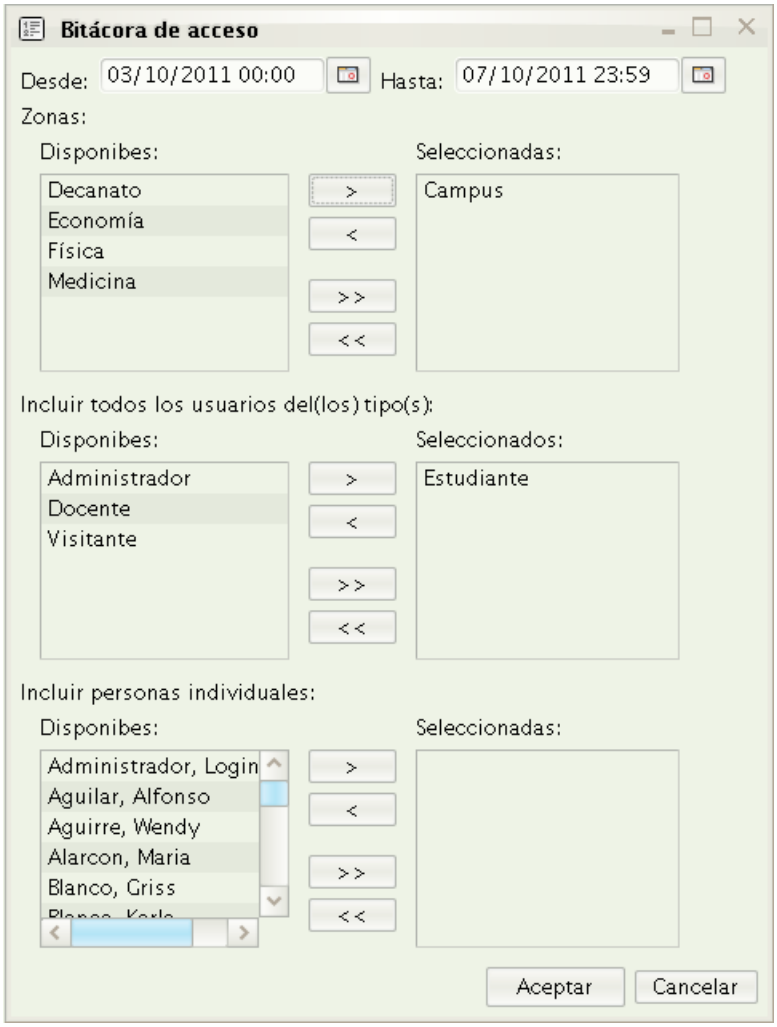
La bitácora de acceso es la que muestra los ingresos a las zonas seleccionadas, por parte de los usuarios indicados, entre un rango temporal determinado.

Específicamente, los parámetros a seleccionar para la generación del reporte son los siguientes:

- *Rango temporal*: Establece desde y hasta que fecha y hora será generado el reporte.

- *Zonas seleccionadas*: Indica las zonas que serán incluidas.
- *Tipos de usuario*: Todos los usuarios del tipo seleccionado serán tomados en cuanta.
- *Personas*: Personas individuales a ser tomadas en cuenta.

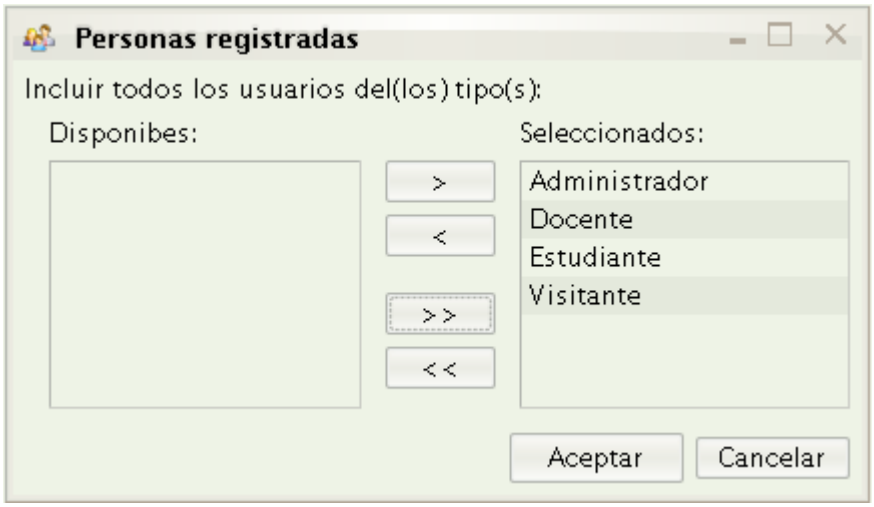
A continuación se muestra el formulario de selección de parámetros:



Informe de personas registradas

Este informe muestra, como su nombre lo indica, el listado de las personas registradas en el sistema. Además de mostrar parte de su información personal, indica su status y si posee huella y login administrativo.

La ventana de selección de parámetros para este reporte se muestra a continuación:



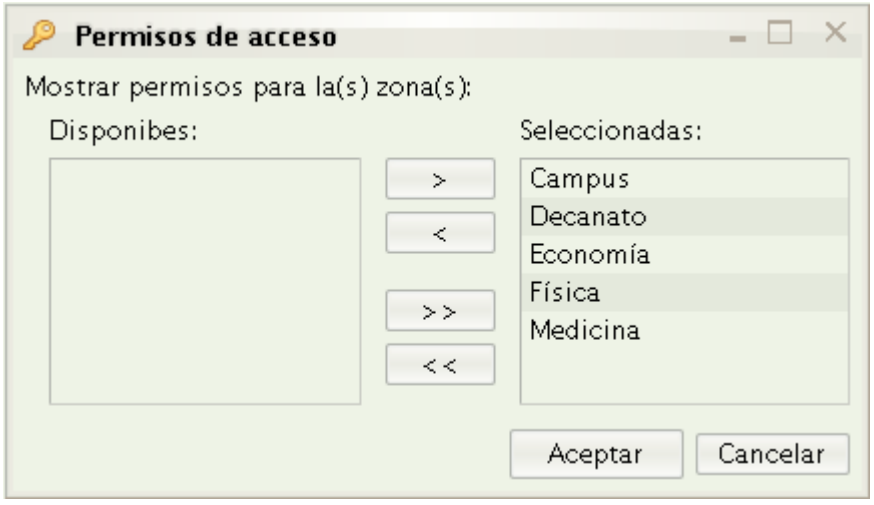
Como se puede observar en la imagen anterior, lo único que se debe indicar para generar el informe de personas registradas, es los tipos de usuarios de los cuales se desea listar las personas.

Informe de permisos de acceso

El informe de permisos de acceso muestra un listado de las zonas geográficas y los tipos de usuario que tiene acceso a cada una de ellas.

El único parámetro que debe ser indicado para tales fines, es el listado de zonas que serán tomadas en cuenta para la generación del reporte.

A continuación se muestra el formulario de la selección de parámetros para la generación de este reporte



5.2 MANUAL DEL PROGRAMA DE CONTROL DE ACCESO

Tabla de contenido

- Definiciones
- Aspectos generales
- Arranque del programa
- Configuración del punto de acceso
- Identificación de usuarios
- Cierre del programa

Definiciones

- *UAccess*: El nombre "UAccess" hace referencia al sistema de control de acceso peatonal basado en biometría creado para la Facultad Multidisciplinaria de Occidente.

- *Punto de acceso peatonal*: Término que se utiliza para hacer referencia a un punto de acceso al campus o una zona determinada. También se utiliza de manera indistinta "Punto de acceso".

- *Programa de control de acceso*: Se refiere a la aplicación software que corre en el computador del punto de acceso peatonal, y que se encarga de identificar usuarios para permitirles o denegarles el paso. También es llamado "Programa de acceso", "Programa de acceso peatonal", "Programa del punto de acceso" o "Software del punto de acceso".

- *Zona*: Se utiliza para referirse a una zona geográfica determinada dentro del campus, e incluso al campus mismo.

- *Persona*: Hace referencia a cada una de las personas registradas en el sistema, indistintamente si éstas utilizan únicamente el sistema administrativo o son usuarios del control de acceso peatonal, o si poseen ambos privilegios o ninguno.

- *Huella dactilar*: Término utilizado, al igual que "captura dactilar" o simplemente "huella", para referirse a la información resultante del proceso de captura dactilar.

- *Login administrativo*: Se refiere a la combinación de nombre de usuario y contraseña otorgado a un usuario para tareas administrativas. También suele ser nombrado simplemente como "login".

- *Torniquete de acceso*: Término utilizado, al igual que "Dispositivo de acceso" o simplemente "Torniquete", para hacer referencia al conjunto del dispositivo mecánico y su circuito controlador.

Aspectos generales

El programa del punto de acceso trabaja esperando constantemente por la colocación de la huella dactilar en el lector, interpretando esta acción como la solicitud del usuario para ingresar a la zona a la que corresponde el punto de acceso.

Si un usuario que solicita el ingreso está registrado y activo, y posee permiso de acceso a la zona, el ingreso a la misma le es concedido; caso contrario, el ingreso le es denegado.

El programa puede ser configurado por personas que poseen dicho privilegio administrativo, para definir los parámetros de funcionamiento del mismo. Para esta acción, al igual que para cerrar la aplicación, es necesario identificarse con un login administrativo válido.

La interfaz de usuario es muy sencilla, dado que el funcionamiento de la aplicación gira en torno a una sola pantalla, casi en su totalidad.

A continuación se presenta dicha pantalla y los elementos que la conforman:

Los elementos que forman parte de dicha pantalla pueden ser visualizados en la imagen anterior y se describen a continuación:

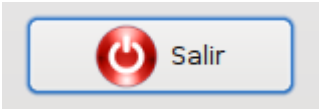
- *Logo de presentación*: Este elemento es estático, y no posee una funcionalidad más allá de la presentación del software.

- *Campo de mensajes*: Es un campo de texto o etiqueta en la que serán mostrados los mensajes originados por la ocurrencia de los diversos sucesos de la aplicación. El

contenido de dicho campo de texto será modificado cada vez que sea necesario mostrar un nuevo mensaje.

- *Botón "Configurar"*: Inicia el proceso de configuración del punto de acceso peatonal.

- *Botón "Salir"*: Inicia el proceso para salir del programa de control de acceso.



Arranque del programa

Cuando el programa arranca hace una serie de verificaciones referente a los elementos involucrados, las cuales se describen a continuación:

- *Conexión a la base de datos no establecida*: Si no se logra establecer la conexión a la base de datos, la aplicación no será iniciada. Como la interfaz gráfica no habrá sido iniciada, dicha situación se informará imprimiendo un mensaje de error en la terminal en que corres el programa.
- *API de libfprint no inicializado*: Si libfprint no es encontrada o no puede ser inicializada, no arrancará la aplicación. Esta situación será informada exclusivamente por medio de un mensaje de error en la terminal correspondiente.
- *Lector de huellas no encontrado*: Si luego de inicializado el sistema y cargada la interfaz gráfica no se encuentra el dispositivo de captura dactilar, se muestra el siguiente mensaje de error:

Error: El dispositivo de lectura dactilar no ha sido encontrado

- *Torniquete de acceso no encontrado*: Si luego de inicializado el sistema y cargada la interfaz gráfica no se encuentra el dispositivo electromecánico, se muestra el siguiente mensaje de error:

  Error: No se ha encontrado el torniquete de acceso

Si las comprobaciones mencionadas anteriormente no encuentran ninguna dificultad, el programa es iniciado correctamente y de manera instantánea se comienza a solicitar al usuario, la colocación de la huella en el lector.

Configuración del punto de acceso

El software del punto de acceso peatonal permite seleccionar desde la interfaz de usuario, y únicamente a administradores del sistema con los permisos requeridos, sus parámetros de funcionamiento.

A continuación se muestra el formulario en que se realiza esta acción:

Los parámetros de funcionamiento establecidos en este formulario son los siguientes:

- *Zona*: Zona geográfica a la que corresponde el punto de acceso peatonal.
- *Lector de huellas*: Tipo de scanner de huellas que será utilizado para la captura de la imagen dactilar de los usuarios que desean ingresar a la zona.

- *Tiempo de espera post-aceptación*: Tiempo, en milisegundos, que se esperará para continuar solicitando una nueva huella, luego de haber permitido el acceso a un usuario.
- *Tiempo de espera post-denegación*: Tiempo, en milisegundos, que se esperará para continuar solicitando una nueva huella, luego de haber denegado el acceso a una persona.

Para poder establecer estos parámetros, el usuario debe ser un administrador del sistema perteneciente a un tipo de usuario con permisos para configurar el punto de acceso.

Identificación de usuarios

Cada vez que una huella es colocada en el lector, el sistema interpreta esta acción como la solicitud por parte de un usuario para ingresar a la zona. En este momento, la huella capturada es analizada; si el usuario es un usuario válido del sistema, con status activo y con permiso de acceso a la zona en cuestión, el acceso le es concedido; en caso contrario, el acceso le es denegado:

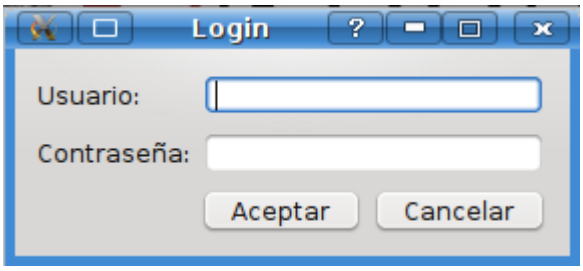
Cuando un usuario es aceptado, se ordena al torniquete que permita el paso de éste y se almacena en la base de datos el ingreso del usuario a la zona respectiva. Al usuario le es informado que ha sido aceptado por medio del mensaje que se muestra en la siguiente imagen:

Por otro lado, cuando el usuario no es reconocido, le es negado el acceso a la zona, y le es informado por el software de la siguiente manera:

Cierre del programa

Para iniciar el cierre del programa, se debe dar un click en el botón "Salir". Para ejecutar esta acción se necesita ser un usuario con login administrativo y poseer el permiso correspondiente asignado.

Por la razón anterior, al clickear sobre el botón, lo primero que el programa hará es solicitar las credenciales del usuario por medio de la siguiente ventana:



Si, y solamente si, el usuario es válido, se procederá a cerrar la aplicación. En dicho caso, primeramente se terminará con las tareas en ejecución por parte del servicio de reconocimiento de usuarios; luego se detendrán los otros servicios utilizados y se cerrarán los APIs de los que hace uso el programa. Al haber realizado estas acciones, el programa será cerrado.

CAPÍTULO VI:

PLAN DE IMPLEMENTACIÓN

6.1 REQUERIMIENTOS OPERATIVOS

Como cualquier proyecto de informática, para obtener el funcionamiento deseado es necesario cumplir con los requerimientos especificados.

En este apartado se presentan los requerimientos operativos del sistema UAccess clasificados en: requerimientos de software, que especifica el software necesario para hacer uso del sistema y sus diferentes módulos; y requerimientos de hardware, que define las necesidades de hardware.

Requerimientos de software

Requerimientos generales de software

- El sistema "UAccess" necesita un servidor de bases de datos MySQL con la base de datos del sistema instalada.

- En cada uno de los equipos de cómputo utilizados, ya sea para el sistema administrativo o para el programa de control de acceso, se requiere de un sistema operativo. Por compatibilidad, se recomienda el uso de Linux,

Debian, con kernel 2.6.32 o superior. El sistema administrativo es multiplataforma, sin embargo en una plataforma distinta a Linux no se dispone de libfprint, por lo que no estaría disponible el módulo biométrico de la misma.

- Para el funcionamiento biométrico y la comunicación USB de las aplicaciones, se requiere de las librerías libfprint 0.0.6 y libusb 2:1.0.8 o superior.

Requerimientos para el sistema administrativo

- La aplicación administrativa, desarrollada en Java SE, requiere del JRE versión 1.6 o superior.

- Para el acceso a la librería nativa de libfprint se requiere de la librería JNA compatible con la arquitectura del sistema operativo. La de 64 bits está incluida en el software.

Requerimientos para la aplicación de acceso

- La aplicación de acceso requiere tener instalado Qt4, específicamente la versión 4.6.3 o superior.

Requerimientos de hardware

Pese a que el sistema es funcional con bajos recursos de hardware, esto comprometería el rendimiento y la estabilidad del mismo. Para un funcionamiento total y acorde a las expectativas, se recomienda como mínimo el equipo hardware listado en la siguiente tabla:

Requerimiento	Sistema administrativo	Sistema de control de acceso
Procesador	Intel Celeron D de 3.06 GHz.	Intel Core 2 Duo de 2.2 GHz.
Memoria RAM	2 GB	2 GB
Disco duro	80 GB	40 GB
Tarjeta de red	FastEthernet	FastEthernet
Ambiente de red	1+ Mbps	1+ Mbps
Puerto USB	1.1 o superior	2.0

Adicionalmente, se requiere:

- Un lector de huellas dactilares por cada punto de acceso y uno por cada sistema administrativo que requiera del

registro de huellas. Se recomienda el scanner DigitalPersona U.are.U 4000B.

- Un circuito controlador para el torniquete de acceso por cada punto de acceso peatonal.

Para el punto de acceso se recomienda además seleccionar equipos de bajo calentamiento y/o un excelente sistema de enfriamiento.

Otros requerimientos

Para el control de acceso peatonal, un elemento indispensable no contemplado como parte del hardware por ser un dispositivo meramente mecánico es el torniquete de acceso.

Dicho torniquete requiere funcionar al recibimiento de la señal del sistema, por lo que se puede optar por cualquiera de las siguientes opciones:

- Reproducir el prototipo en condiciones apropiadas para el funcionamiento exigido por el sistema de control de acceso.

- Modificar un torniquete prefabricado para activar el paso ante el recibimiento de la señal emitida por el hardware controlador.
- Comprar torniquetes diseñados para su activación a partir de la señal emitida por el sistema, ya sea el pulso eléctrico o un código digital en particular; en cuyo caso habría que modificar el hardware controlador.

Tanto para el caso de la adquisición y posterior modificación de un torniquete de acceso prefabricado, como para la adquisición de uno que trabaje en base al recibimiento de un pulso eléctrico o una señal electrónica; e incluso para la fabricación de un torniquete con las características mecánicas de uno prefabricado, fusionado al funcionamiento clave del prototipo incluido con este trabajo, se puede seleccionar entre diferentes tipos de torniquetes de acceso, cuyas características están vinculadas a diferentes necesidades. A continuación se muestran los tipos recomendados para el control de acceso en la Facultad Multidisciplinaria de Occidente:

Torniquete tipo T

Ha sido el torniquete tradicionalmente utilizado, tiene la ventaja de ser de fácil fabricación en comparación a los demás. Por la misma razón, suele tener un precio más accesible. Adicionalmente, son de fácil modificación; si es que se opta por la adquisición y modificación de torniquetes prefabricados.

Torniquete tipo trípode

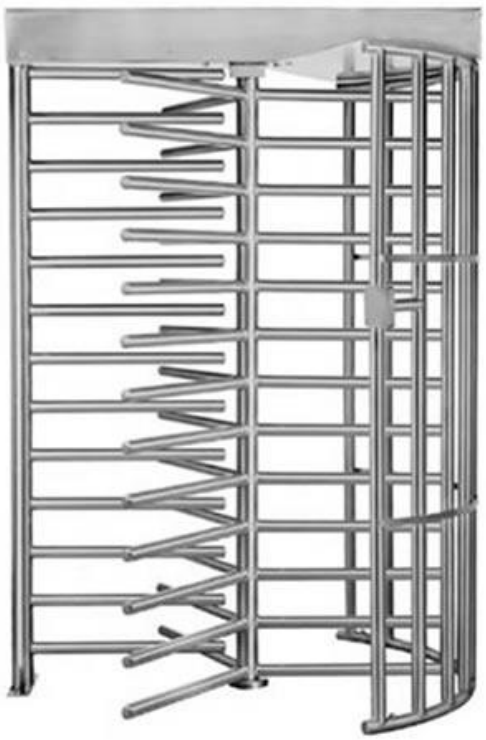
La principal ventaja de este tipo de torniquete es el ahorro de espacio, así como la versatilidad para su uso. Es muy recomendable para el paso rápido de altos volúmenes de usuarios y sería muy recomendable para la Facultad. Dentro de

sus desventajas podemos mencionar el hecho de que su fabricación o modificación sería más complejas y costosas, así como su adquisición, en comparación al torniquete tipo T.

Torniquete de cuerpo entero

El torniquete de cuerpo entero es la mejor alternativa en lo que a seguridad respecta, puesto que hace un cierre completo del paso a través de él; es decir, no posee la vulnerabilidad que presentan los anteriores en que las personas podrían pasar por debajo o por sobre de él. Dentro de sus desventajas se puede destacar el aspecto de costos, puesto que su valor es muy superior a los anteriores, tanto para su adquisición y

modificación, como para su fabricación. Otro aspecto a tomar en cuenta es el hecho de que el tiempo medio de uso de este tipo de torniquetes es mínimamente superior al de los dos anteriores, lo que podría afectar significativamente la fluidez de usuarios, principalmente en las horas pico.



6.2 INVERSIÓN REQUERIDA

Dado que la universidad ya cuenta con equipo hardware para la instalación de la base de datos, y equipos de cómputo utilizables para la administración del sistema "UAccess", la inversión requerida se mide en base a cada punto de acceso deseado.

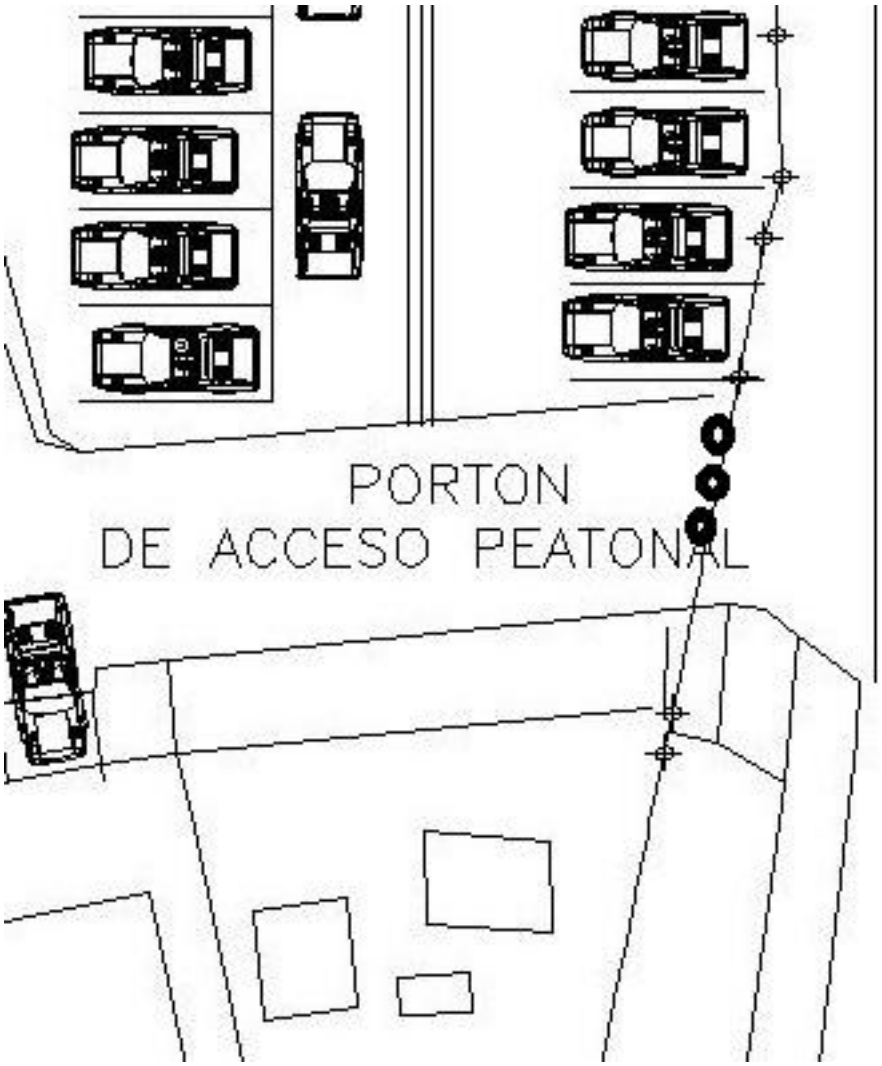
En la siguiente tabla se muestra la inversión requerida para cada punto de acceso peatonal instalado:

Descripción del requerimiento	**Monto (US$)**
Equipo de cómputo	$420.00
Lector de huellas dactilares	$43.58
Circuito de control del torniquete	$26.87
Torniquete de acceso	$375.00
Modificación del torniquete de acceso	$33.65
Módulo de protección del equipo de cómputo	$18.00
Inversión total por punto de acceso	**$917.10**

Para una primera etapa de implementación, se sugiere la colocación de cinco puntos de acceso en las entradas principales del campus de la facultad: tres en el portón de

acceso peatonal y en la acera peatonal contigua al portón de salida vehicular, según se ilustra a continuación:

Ubicación para los puntos de acceso del portón de acceso peatonal:



Ubicación de los puntos de acceso para la acera peatonal contigua a la salida vehicular:

Para esta primera etapa de implementación, la inversión requerida sería el resultado de multiplicar la cantidad de

puntos de acceso por el monto individual de cada uno de ellos, obteniendo costo total de $4,585.50.

Se deben tomar muy en cuenta las siguientes consideraciones adicionales, respecto al costo de implementación:

- El costo total mostrado anteriormente corresponde únicamente al de los puntos de acceso, dado que la Facultad Multidisciplinaria de Occidente cuenta con los recursos materiales y humanos requeridos para la implementación.
- Para posteriores etapas, el costo sería el producto del costo individual de cada punto de acceso por la cantidad de los mismos que haya sido implementada.
- El costo individual de cada punto de acceso puede variar dependiendo los tipos de equipo de cómputo y torniquetes de acceso que se opte por utilizar
- El costo de implementación de cada punto de acceso varía en relación al precio de mercado de cada uno de los componentes que lo conforman.

6.3 PASOS DE IMPLEMENTACIÓN

En este apartado se describen los pasos requeridos para la implementación del sistema "UAccess". Estos pasos se presentan a continuación en orden cronológico:

- *Instalación de la base de datos en servidor*: El primero paso es tener la base de datos centralizada instalada en el servidor. Se debe crear también en la base de datos el primer usuario administrativo. Aquí vale aclarar que aunque la base de datos ha sido realizada en MySQL, si a la hora de implementar el sistema se desea utilizar otro servidor, tanto el sistema administrativo como el programa de control de acceso se pueden adaptar fácilmente.

- *Instalación del sistema administrativo*: Como segundo paso, se debe proceder a instalar el sistema administrativo en los equipos de cómputo requeridos.

- *Creación de los usuarios administrativos*: Ya teniendo el sistema administrativo instalado en los equipos que así lo requieran, se debe proceder a la creación de los

usuarios administrativos del sistema y la asignación de privilegios administrativos a los mismos, dado que éstos tomarán parte activa en los pasos siguientes.

- *Identificación y creación de las zonas*: En este paso se identificarán las zonas geográficas para las que se usará el sistema. Éstas deberán ser creadas a nivel de sistema.

- *Identificación y creación de los tipos de usuario*: Luego de identificadas las zonas, habrá que identificar los tipos de usuario no administrativos y los permisos de acceso para las personas pertenecientes a dichos tipos de usuario.

- *Registro de los usuarios del sistema*: Este paso consiste en registrar a cada uno de los usuarios que hará uso del sistema de acceso peatonal, con su respectiva huella dactilar. Este es el paso que más tiempo y trabajo exigirá, y requiere de especial cuidado porque un error aquí puede generar comportamientos no deseados, y que aún podrían pasar desapercibidos durante mucho tiempo.

- *Instalación de los puntos de acceso peatonal*: Como paso final para la implementación del sistema "UAccess", se debe instalar cada uno de los puntos de acceso peatonal de las diferentes zonas geográficas y configurarse adecuadamente.

Luego de implementado, se sugiere el constante monitoreo que permita detectar cualquier comportamiento indeseado y/o identificar cambios que puedan mejorar el funcionamiento del sistema.

CAPÍTULO VII:

CONCLUSIONES Y RECOMENDACIONES

7.1 CONCLUSIONES

- En la Facultad Multidisciplinaria de Occidente se requiere de la implementación de un sistema de control de acceso fiable y eficiente, que garantice que solamente las personas indicadas pueden ingresar a sus diferentes zonas geográficas.

- El sistema "UAccess" se perfila como la principal solución ante la falta de un mecanismo de control de acceso en la UES-FMO.

- Implementar el sistema "UAccess" sería mucho más viable económicamente para la Facultad en comparación a soluciones propietarias y prefabricadas del mercado.

- La Facultad Multidisciplinaria de Occidente posee la capacidad tanto operativa como económica para la implementación del sistema "UAccess", y de esta manera resolver en gran medida los problemas de seguridad que últimamente se le han presentado.

7.2 RECOMENDACIONES

- Llevar a cabo la implementación del sistema biométrico de control de acceso peatonal, "UAccess".

- Como parte de la implementación del sistema, definir políticas claras en cuanto al acceso al campus y a sus zonas, que complementen el trabajo del sistema de control de acceso automatizado.

- Definir, previo a la implementación del sistema, reglas de comportamiento para casos contingenciales en los que el funcionamiento del sistema automatizado si vea afectado parcial o totalmente.

- Llevar a cabo una campaña informativa y de concientización a cada uno de los usuarios del sistema de acceso antes y durante la implementación del mismo.

- Impulsar la implementación del sistema "UAccess" para el resto de la Universidad de El Salvador.

7.3 REFERENCIAS BIBLIOGRÁFICAS

- Reconocimiento Automático Mediante Patrones Biométricos de Huella Dactilar

  Danilo Simón Zorita (Zorita), UPM, Año 2003

- Como Programar en Java

  Deitel & Deitel, 5ª Edición

- Pro EJB 3: Java Persistence API

  Mike Keith, Jason Haley and Merrick Schincariol

- C/C++. Edición revisada y actualizada 2010

  Miguel Ángel Acera García

- C++ GUI Programming with Qt 4

  Jasmin Blanchette & Mark Summerfield

- PIC18F2455/2550/4455/4550 Data Sheet

  Microchip Technology Inc.

- Qt Reference Documentation

  http://doc.qt.nokia.com/4.6/index.html

- Libfprint API Reference

  http://www.reactivated.net/fprint/api/

- Libusb Developers Guide

  http://libusb.sourceforge.net/doc/

- JNA API Documentation

  http://jna.java.net/javadoc/overview-summary.html

- Programador PIPO2

  http://213.97.130.124/progs/pipo2/pipo2.htm

- PICC CCS Compiler Manual

  http://www.ccsinfo.com/downloads/ccs_c_manual.pdf

ANEXOS

ANEXO 1: DOCUMENTACIÓN DE LA BASE DE DATOS

Database uaccess

Table structure for table bitacora_de_acceso

Field	Type	Null	Default	Comments
id_persona	int(11)	No		Identificador de la persona que ingresó
id_zona	int(11)	No		Identificador de la zona a la que ingresó la persona
hora	timestamp	No	CURRENT_ TIMESTAMP	Momento en que ocurrió la acción

Table structure for table bitacora_de_acceso

Field	Type	Null	Default	Comments
id_persona	int(11)	No		Identificador de la persona que ingresó
id_zona	int(11)	No		Identificador de la zona a la que ingresó la persona
hora	timestamp	No	CURRENT_ TIMESTAMP	Momento en que ocurrió la acción

Table structure for table huellas

Field	Type	Null	Default	Comments
id_persona	int(11)	No		Identificador de la persona a la que corresponde la captura dactilar
data	var(3072)	No		Información de la captura dactilar en bytes

Table structure for table huellas

Field	Type	Null	Default	Comments
id_persona	int(11)	No		Identificador de la persona a la que corresponde la captura dactilar
data	var(3072)	No		Información de la captura dactilar en bytes

Table structure for table logins_administrativos

Field	Type	Null	Default	Comments
id_persona	int(11)	No		Identificador de la persona a la que corresponde el login administrativo
login	varchar(10)	No		Nombre de usuario para el inicio de sesión
contrasena	varchar(32)	No		Contraseña encriptada del usuario

Table structure for table logins_administrativos

Field	Type	Null	Default	Comments
id_persona	int(11)	No		Identificador de la persona a la que corresponde el login administrativo
login	varchar(10)	No		Nombre de usuario para el inicio de sesión
contrasena	varchar(32)	No		Contraseña encriptada del usuario

Table structure for table permisos_administrativos

Field	Type	Null	Default	Comments
id_tipo_de_usuario	int(11)	No		Identificador del tipo de usuario al que corresponde el permiso
id_privilegio_administrativo	int(11)	No		Identificador del privilegio administrativo asignado

Table structure for table permisos_administrativos

Field	Type	Null	Default	Comments
id_tipo_de_usuario	int(11)	No		Identificador del tipo de usuario al que corresponde el permiso
id_privilegio_administrativo	int(11)	No		Identificador del privilegio administrativo asignado

Table structure for table permisos_de_acceso

Field	Type	Null	Default	Comments
id_tipo_de_usuario	int(11)	No		Identificador del tipo de usuario al que corresponde el permiso
id_zona	int(11)	No		Identificador de la zona a la que se ha concedido acceso

Table structure for table permisos_de_acceso

Field	Type	Null	Default	Comments
id_tipo_de_usuario	int(11)	No		Identificador del tipo de usuario al que corresponde el

				permiso
id_zona	int(11)	No		Identificador de la zona a la que se ha concedido acceso

Table structure for table personas

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador inequívoco de la persona registrada
apellidos	varchar(50)	No		Apellidos de la persona
nombres	varchar(50)	No		Nombres de la persona
id_tipo_de_documento	int(11)	No		Identificador del tipo de documento con que la persona se identifica
numero_de_documento	varchar(15)	No		Número de documento de identidad de la persona
id_tipo_de_usuario	int(11)	No		Identificador del tipo de usuario al que la persona corresponde
activo	tinyint(1)	No		

Table structure for table personas

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador inequívoco de la persona registrada
apellidos	varchar(50)	No		Apellidos de la persona
nombres	varchar(50)	No		Nombres de la persona
id_tipo_de_documento	int(11)	No		Identificador del tipo de

				documento con que la persona se identifica
numero_de_documento	varchar(15)	No		Número de documento de identidad de la persona
id_tipo_de_usuario	int(11)	No		Identificador del tipo de usuario al que la persona corresponde
activo	tinyint(1)	No		

Table structure for table privilegios_administrativos

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador del privilegio administrativo
nombre	varchar(25)	No		Nombre único del privilegio
descripcion	varchar(150)	No		Descripción del privilegio

Table structure for table privilegios_administrativos

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador del privilegio administrativo
nombre	varchar(25)	No		Nombre único del privilegio
descripcion	varchar(150)	No		Descripción del privilegio

Table structure for table tipos_de_documento

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador del tipo de documento de identificación
nombre	varchar(25)	No		Nombre único del tipo de documento

descripcion	varchar(150)	No		Descipción del tipo de documento
formato	varchar(15)	No		Formato del tipo de documento

Table structure for table tipos_de_documento

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador del tipo de documento de identificación
nombre	varchar(25)	No		Nombre único del tipo de documento
descripcion	varchar(150)	No		Descipción del tipo de documento
formato	varchar(15)	No		Formato del tipo de documento

Table structure for table tipos_de_usuario

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador del tipo de usuario del sistema
nombre	varchar(25)	No		Nombre único del tipo de usuario
descripcion	varchar(150)	No		Descripción del tipo de usuario

Table structure for table tipos_de_usuario

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador del tipo de usuario del sistema
nombre	varchar(25)	No		Nombre único del tipo de

				usuario
descripcion	varchar(150)	No		Descripción del tipo de usuario

Table structure for table zonas

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador de la zona
nombre	varchar(25)	No		Nombre único de la zona
descripcion	varchar(150)	No		Descripción de la zona

Table structure for table zonas

Field	Type	Null	Default	Comments
id	int(11)	No		Identificador de la zona
nombre	varchar(25)	No		Nombre único de la zona
descripcion	varchar(150)	No		Descripción de la zona

ANEXO 2: MANUAL DE REFERENCIA DEL PROGRAMADOR PARA EL SISTEMA ADMINISTRATIVO

Package org.uaccess

Class Summary	
GestorDeDatosGlobales	Objeto encargado de almacenar y gestionar los datos globales
Main	Clase principal de la aplicación

org.uaccess

Class GestorDeDatosGlobales

```
java.lang.Object
  └ org.uaccess.GestorDeDatosGlobales
```

```
public class GestorDeDatosGlobales
extends java.lang.Object
```

Objeto encargado de almacenar y gestionar los datos globales

Method Summary	
`static` GestorDeDatosGlobales	**getInstacia**`()` Obtiene el gestor de datos globales de la aplicación
Persona	**getUsuario**`()` Obtiene el usuario actual de la aplicación
`void`	**setUsuario**`(`Persona `usuario)` Establece la persona que está usando la aplicación

Method Detail

getInstacia

```
public static GestorDeDatosGlobales getInstacia()
```

Obtiene el gestor de datos globales de la aplicación

Returns:

---

setUsuario

`public void` **`setUsuario`**`(Persona usuario)`

Establece la persona que está usando la aplicación

Parameters:

`usuario` - usuario de la aplicación

---

getUsuario

`public Persona` **`getUsuario`**`()`

Obtiene el usuario actual de la aplicación

Returns:

usuario de la aplicación

org.uaccess

Class Main

```
java.lang.Object
  └ org.uaccess.Main
```

```
public class Main
extends java.lang.Object
```

`Clase` principal de la aplicación

Constructor Summary

Main`()`	

Method Summary

`static void`	**establecerTema**`()` Establece el tema (Look And Feel) de la aplicación
`static void`	**main**`(java.lang.String[] args)`

Constructor Detail

Main

```
public Main()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:
`args` - the command line arguments

---

establecerTema

```
public static void establecerTema()
```

Establece el tema (Look And Feel) de la aplicación

Package org.uaccess.fprint

Class Summary

CapturaDactilar	Captura de una huella dactilar
Dispositivo	Dispositivo lector de huellas dactilares
InterfazFprintAPI	Interfaz entre el API de fprint y la aplicación

org.uaccess.fprint

Class CapturaDactilar

```
java.lang.Object
  └ org.uaccess.fprint.CapturaDactilar
```

```
public class CapturaDactilar
extends java.lang.Object
```

Captura de una huella dactilar

Constructor Summary

CapturaDactilar(com.sun.jna.Pointer pDatos, com.sun.jna.Pointer pImagen, com.sun.jna.Pointer pBinarizada) Crea un nuevo objeto a partir de los parámetros recibidos	

Method Summary

int	**getAlturaImagen**() Obtiene la altura de la imagen
int	**getAnchoImagen**() Obtiene el ancho de la imagen
byte[]	**getBytesDatos**() Obtiene los bytes correspondientes a los datos resultantes de la captura
byte[]	**getBytesImagen**() Obtiene los bytes correspondientes a la imagen original (escala de grises)
byte[]	**getBytesImagenBinarizada**() Obtiene los bytes correspondientes a la imagen binarizada
com.sun.jna.Pointer	**getPBinarizada**() Obtiene el puntero a la imagen binarizada de la huella capturada
com.sun.jna.Pointer	**getPDatos**() Obtiene el puntero a los datos de la captura dactilar
com.sun.jna.Pointer	**getPImagen**() Obtiene el puntero a la imagen en escala de grises de la huella capturada
void	**setPunteros**(com.sun.jna.Pointer pDatos, com.sun.jna.Pointer pImagen, com.sun.jna.Pointer pBinarizada) Establece los punteros correspondientes a la captura dactilar

Constructor Detail

CapturaDactilar

```
public CapturaDactilar(com.sun.jna.Pointer pDatos,
                       com.sun.jna.Pointer pImagen,
                       com.sun.jna.Pointer pBinarizada)
```

Crea un nuevo objeto a partir de los parámetros recibidos

Parameters:

`pDatos` - puntero a los datos de la captura

`pImagen` - puntero a la imagen de la huella

`pBinarizada` - puntero a la imagen binarizada de la huella

Method Detail

getPDatos

```
public com.sun.jna.Pointer getPDatos()
```

Obtiene el puntero a los datos de la captura dactilar

Returns:

puntero a los datos de la captura

---

getPImagen

```
public com.sun.jna.Pointer getPImagen()
```

Obtiene el puntero a la imagen en escala de grises de la huella capturada

Returns:

puntero a la imagen de la huella

---

getPBinarizada

```
public com.sun.jna.Pointer getPBinarizada()
```

Obtiene el puntero a la imagen binarizada de la huella capturada

Returns:

puntero a la imagen de la huella

---

setPunteros

```
public void setPunteros(com.sun.jna.Pointer pDatos,
                        com.sun.jna.Pointer pImagen,
```

com.sun.jna.Pointer pBinarizada)

Establece los punteros correspondientes a la captura dactilar

Parameters:

`pDatos` - puntero a los datos

`pImagen` - puntero a la imagen

`pBinarizada` - puntero a la imagen binarizada

---

getAnchoImagen

`public int` **`getAnchoImagen`**`()`

Obtiene el ancho de la imagen

Returns:

ancho de la imagen en pixeles

---

getAlturaImagen

`public int` **`getAlturaImagen`**`()`

Obtiene la altura de la imagen

Returns:

altura de la imagen en pixeles

---

getBytesImagen

`public byte[]` **`getBytesImagen`**`()`

Obtiene los bytes correspondientes a la imagen original (escala de grises)

Returns:

imagen original en bytes

---

getBytesImagenBinarizada

`public byte[]` **`getBytesImagenBinarizada`**`()`

Obtiene los bytes correspondientes a la imagen binarizada

Returns:

imagen binarizada en bytes

---

getBytesDatos

`public byte[]` **`getBytesDatos`**`()`

Obtiene los bytes correspondientes a los datos resultantes de la captura

Returns:
datos de la captura dactilar en bytes

org.uaccess.fprint

Class Dispositivo

```
java.lang.Object
  └ org.uaccess.fprint.Dispositivo
```

```
public class Dispositivo
extends java.lang.Object
```

Dispositivo lector de huellas dactilares

Constructor Summary

Dispositivo(com.sun.jna.Pointer pDispositivo) Crea un nuevo dispositivo a partir del puntero hacia el mismo	

Method Summary

java.lang.String	**getNombre**() Obtiene el nombre del dispositivo de captura
java.lang.String	**getNombreControlador**() Obtiene el nombre del controlador del dispositivo de captura
com.sun.jna.Pointer	**getPuntero**() Obtiene el puntero al dispositivo de captura
java.lang.String	**toString**()

Constructor Detail

Dispositivo

```
public Dispositivo(com.sun.jna.Pointer pDispositivo)
```

Crea un nuevo dispositivo a partir del puntero hacia el mismo

Parameters:

`pDispositivo` - puntero al dispositivo

Method Detail

getPuntero

```
public com.sun.jna.Pointer getPuntero()
```

Obtiene el puntero al dispositivo de captura

Returns:

puntero al dispositivo

---

getNombre

```
public java.lang.String getNombre()
```

Obtiene el nombre del dispositivo de captura

Returns:

nombre del dispositivo

---

getNombreControlador

```
public java.lang.String getNombreControlador()
```

Obtiene el nombre del controlador del dispositivo de captura

Returns:

nombre del controlador del dispositivo

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

org.uaccess.fprint

Class InterfazFprintAPI

```
java.lang.Object
  └ org.uaccess.fprint.InterfazFprintAPI
```

```
public class InterfazFprintAPI
extends java.lang.Object
```

Interfaz entre el API de fprint y la aplicación

Method Summary

com.sun.jna.Pointer	**BinarizarImagen**(com.sun.jna.Pointer pImagen) Ejecuta la binarización de la imagen recibida
CapturaDactilar	**CapturarHuella**(Dispositivo lector) Ejecuta la captura de una huella dactilar desde el dispositivo indicado
Dispositivo[]	**DescubrirDispositivos**() Ubica los dispositivos de captura dactilar disponibles
static InterfazFprintAPI	**getInstancia**() Obtiene la instancia de la interfaz de acceso a fprint
int	**ObtenerAlturaImagen**(com.sun.jna.Pointer pImagen) Obtiene la altura de la imagen indicada
int	**ObtenerAnchoImagen**(com.sun.jna.Pointer pImagen) Obtiene el ancho de la imagen indicada
byte[]	**ObtenerBytesImagen**(com.sun.jna.Pointer pImagen) Obtiene los bytes correspondientes a la imagen indicada
byte[]	**ObtenerDatosCaptura**(com.sun.jna.Pointer pDatosCaptura) Obtiene los bytes correspondientes a los datos de la captura dactilar recibida
java.lang.String	**ObtenerNombreControlador**(com.sun.jna.Pointer pDispositivo) Obtiene el nombre del controlador del dispositivo indicado
java.lang.String	**ObtenerNombreDispositivo**(com.sun.jna.Pointer pDispositivo) Obtiene el nombre del dispositivo indicado

Method Detail

getInstancia

```
public static InterfazFprintAPI getInstancia()
```

Obtiene la instancia de la interfaz de acceso a fprint

Returns:
instancia de la interfaz

DescubrirDispositivos

```
public Dispositivo[] DescubrirDispositivos()
                                        throws java.lang.Exception
```

Ubica los dispositivos de captura dactilar disponibles

Returns:
dispositivos disponibles

Throws:
`java.lang.Exception`

ObtenerNombreDispositivo

```
public java.lang.String
ObtenerNombreDispositivo(com.sun.jna.Pointer pDispositivo)
```

Obtiene el nombre del dispositivo indicado

Parameters:
`pDispositivo` - puntero al dispositivo

Returns:
nombre del dispositivo

ObtenerNombreControlador

```
public java.lang.String
ObtenerNombreControlador(com.sun.jna.Pointer pDispositivo)
```

Obtiene el nombre del controlador del dispositivo indicado

Parameters:
`pDispositivo` - puntero al dispositivo

Returns:
nombre del controlador del dispositivo

ObtenerAnchoImagen

```
public int ObtenerAnchoImagen(com.sun.jna.Pointer pImagen)
```

Obtiene el ancho de la imagen indicada

Parameters:
`pImagen` - puntero a la imagen

Returns:

ancho de la imagen en pixeles

ObtenerAlturaImagen

```
public int ObtenerAlturaImagen(com.sun.jna.Pointer pImagen)
```

Obtiene la altura de la imagen indicada

Parameters:

`pImagen` - puntero a la imagen

Returns:

altura de la imagen en pixeles

ObtenerBytesImagen

```
public byte[] ObtenerBytesImagen(com.sun.jna.Pointer pImagen)
```

Obtiene los bytes correspondientes a la imagen indicada

Parameters:

`pImagen` - puntero a la imagen

Returns:

bytes de la imagen

BinarizarImagen

```
public com.sun.jna.Pointer BinarizarImagen(com.sun.jna.Pointer pImagen)
```

Ejecuta la binarización de la imagen recibida

Parameters:

`pImagen` - puntero a la imagen a binarizar

Returns:

puntero a la imagen binarizada

ObtenerDatosCaptura

```
public byte[] ObtenerDatosCaptura(com.sun.jna.Pointer pDatosCaptura)
```

Obtiene los bytes correspondientes a los datos de la captura dactilar recibida

Parameters:

`pDatosCaptura` - puntero a los datos de la captura

Returns:

bytes correspondientes a los datos

CapturarHuella

```
public CapturaDactilar CapturarHuella(Dispositivo lector)
                                throws java.lang.Exception
```

Ejecuta la captura de una huella dactilar desde el dispositivo indicado

Parameters:

`lector` - dispositivo lector de huellas

Returns:

captura dactilar resultante

Throws:

`java.lang.Exception`

Package org.uaccess.gui

Class Summary

DialogoCapturarHuella	Ventana tipo diálogo utilizada para la captura de huellas dactilares
DialogoLoginAdministrativo	Ventana tipo diálogo utilizada para la edición de logins administrativos
JpnCapturarHuella	Panel diseñado para la captura de huellas dactilares
JpnIngresar	Panel diseñado para la autenticación de usuarios que desean ingresar al sistema
JpnLoginAdministrativo	Panel diseñado para la edición de los logins administrativos asignados a las personas
JpnParametrosParaBitacoraDeAcceso	Panel diseñado para la selección de parámetros para el informe "Bitácora de acceso"
JpnParametrosParaPermisosDeAcceso	Panel diseñado para la selección de parámetros para el informe "Permisos de acceso"
JpnParametrosParaPersonasRegistradas	Panel diseñado para la selección de parámetros para el informe "Permisos de acceso"
JpnPersona	Panle diseñado para la edición de personas
JpnTipoDeDocumento	Panel diseñado para la edición de tipos de documento

JpnTipoDeUsuario	Panel diseñado para la edición de tipos de usuario
JpnZona	Panel diseñado para la edición de zonas
Principal	Ventana principal del sistema

org.uaccess.gui

Class DialogoCapturarHuella

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Dialog
                  └ javax.swing.JDialog
                      └
org.uaccess.utilidades.controles.DialogoAgregarOEditarObjeto
                          └ org.uaccess.gui.DialogoCapturarHuella
```

```
public class DialogoCapturarHuella
extends DialogoAgregarOEditarObjeto
```

Ventana tipo diálogo utilizada para la captura de huellas dactilares

See Also:

Serialized Form

Constructor Summary

DialogoCapturarHuella(java.net.URL icono) Crea un nuevo diólgo con soporte para captura de huellas dactilares

Method Summary

CapturaDactilar	**mostrar**(CapturaDactilar c) Muestra el diálogo al usuario

Constructor Detail

DialogoCapturarHuella

`public` **`DialogoCapturarHuella`**`(java.net.URL icono)`

Crea un nuevo diólgo con soporte para captura de huellas dactilares

Parameters:

`icono` - URL del ícono del diálogo

Method Detail

mostrar

`public` CapturaDactilar **`mostrar`**`(`CapturaDactilar `c)`

Muestra el diálogo al usuario

Parameters:

`c` - captura dactilar a mostrar en el panel contenido

Returns:

captura dactilar contenida al momento de cerrar el diálogo

org.uaccess.gui

Class DialogoLoginAdministrativo

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
org.uaccess.utilidades.controles.DialogoAgregarOEditarObjeto
                          └org.uaccess.gui.DialogoLoginAdministrativo
```

`public class` **`DialogoLoginAdministrativo`**
`extends` DialogoAgregarOEditarObjeto

Ventana tipo diálogo utilizada para la edición de logins administrativos

See Also:

Serialized Form

Constructor Summary

DialogoLoginAdministrativo(java.net.URL icono) Crea un nuevo diálogo con soporte para la edición de logins administrativos

Method Summary

LoginAdministrativo	**mostrar**(LoginAdministrativo login) Muestra la ventana y como valor inicial de sus controles el login recibido

Constructor Detail

DialogoLoginAdministrativo

public **DialogoLoginAdministrativo**(java.net.URL icono)

Crea un nuevo diálogo con soporte para la edición de logins administrativos

Parameters:

icono - URL del ícono del diálogo

Method Detail

mostrar

public LoginAdministrativo **mostrar**(LoginAdministrativo login)

Muestra la ventana y como valor inicial de sus controles el login recibido

Parameters:

login - login recibido

Returns:

login al momento de cerrar el diálogo

org.uaccess.gui

Class JpnCapturarHuella

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └org.uaccess.gui.JpnCapturarHuella
```

```
public class JpnCapturarHuella
extends javax.swing.JPanel
```

Panel diseñado para la captura de huellas dactilares

See Also:
Serialized Form

Constructor Summary

JpnCapturarHuella()	Crea un nuevo panel para captura de huellas

Method Summary

javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del panel
CapturaDactilar	**obtenerCaptura**() Obtiene la captura dactilar resultante de la ejecución de la captura y correspondiente a la imagen del panel visor de la huella

Constructor Detail

JpnCapturarHuella

```
public JpnCapturarHuella()
```

Crea un nuevo panel para captura de huellas

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Obtiene el botón por defecto del panel

Returns:
botón por defecto

obtenerCaptura

public CapturaDactilar **obtenerCaptura**()

Obtiene la captura dactilar resultante de la ejecución de la captura y correspondiente a la imagen del panel visor de la huella

Returns:

captura dactilar resultante

org.uaccess.gui

Class JpnIngresar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └org.uaccess.gui.JpnIngresar
```

```
public class JpnIngresar
extends javax.swing.JPanel
```

Panel diseñado para la autenticación de usuarios que desean ingresar al sistema

See Also:

Serialized Form

Constructor Summary

JpnIngresar() Crea un nuevo panel para el ingreso de usuarios

Method Summary

javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del panel
Persona	**getUsuario**() Obtiene el usuario que está ingresando al sistema por medio del panel

Constructor Detail

JpnIngresar

```
public JpnIngresar()
```

Crea un nuevo panel para el ingreso de usuarios

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Obtiene el botón por defecto del panel

Returns:

botón por defecto

---

getUsuario

```
public Persona getUsuario()
```

Obtiene el usuario que está ingresando al sistema por medio del panel

Returns:

usuario que está ingresando al sistema

org.uaccess.gui

Class JpnLoginAdministrativo

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └org.uaccess.gui.JpnLoginAdministrativo
```

```
public class JpnLoginAdministrativo
extends javax.swing.JPanel
```

Panel diseñado para la edición de los logins administrativos asignados a las personas

See Also:

Serialized Form

Constructor Summary

JpnLoginAdministrativo()	Crea un nuevo panel para la edición de logins administrativos

Method Summary

javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del panel
LoginAdministrativo	**getDatos**() Obtiene los datos del login del panel
void	**setDatos**(LoginAdministrativo login) Establece los datos del login a partir del objeto recibido

Constructor Detail

JpnLoginAdministrativo

public **JpnLoginAdministrativo**()

Crea un nuevo panel para la edición de logins administrativos

Method Detail

getBotonPorDefecto

public javax.swing.JButton **getBotonPorDefecto**()

Obtiene el botón por defecto del panel

Returns:

botón por defecto

---

setDatos

public void **setDatos**(LoginAdministrativo login)

Establece los datos del login a partir del objeto recibido

Parameters:

`login` - login a mostrar en el panel

---

getDatos

`public` [LoginAdministrativo](#) **`getDatos`**`()`

Obtiene los datos del login del panel

Returns:

login actual del panel

org.uaccess.gui

Class JpnParametrosParaBitacoraDeAcceso

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelParametrosParaReporte
                      └org.uaccess.gui.JpnParametrosParaBitacoraDeAcceso
```

`public class` **`JpnParametrosParaBitacoraDeAcceso`**
`extends` [PanelParametrosParaReporte](#)

Panel diseñado para la selección de parámetros para el informe "Bitácora de acceso"

Constructor Summary

JpnParametrosParaBitacoraDeAcceso`()`

Method Summary

`protected void`	**establecerValoresIniciales**`()` Establece los valores iniciales de los controles del panel
`protected void`	**generarReporte**`()` Genera el reporte a partir de los parámetros indicados
`javax.swing.JButton`	**getBotonPorDefecto**`()`

	Obtiene el botón por defecto del panel
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
protected boolean	**validarParametros**() Valida los parámetros establecidos en los controles

Constructor Detail

JpnParametrosParaBitacoraDeAcceso

```
public JpnParametrosParaBitacoraDeAcceso()
```

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Description copied from class: PanelParametrosParaReporte
Obtiene el botón por defecto del panel
Specified by:
getBotonPorDefecto in class PanelParametrosParaReporte
Returns:
botón por defecto

---

establecerValoresIniciales

```
protected void establecerValoresIniciales()
```

Description copied from class: PanelParametrosParaReporte
Establece los valores iniciales de los controles del panel
Specified by:
establecerValoresIniciales in class PanelParametrosParaReporte

---

validarParametros

```
protected boolean validarParametros()
```

Description copied from class: PanelParametrosParaReporte
Valida los parámetros establecidos en los controles
Specified by:

validarParametros in class PanelParametrosParaReporte

Returns:

'true' si los parámetros se han especificado de manera correcta, 'false' en caso contrario

---

generarReporte

```
protected void generarReporte()
```

Description copied from class: PanelParametrosParaReporte

Genera el reporte a partir de los parámetros indicados

Specified by:

generarReporte in class PanelParametrosParaReporte

---

inicializarComponentes

```
protected void inicializarComponentes()
```

Description copied from class: PanelParametrosParaReporte

Inicializa los componentes de la interfaz gráfica

Specified by:

inicializarComponentes in class PanelParametrosParaReporte

org.uaccess.gui

Class JpnParametrosParaPermisosDeAcceso

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelParametrosParaReporte
                      └org.uaccess.gui.JpnParametrosParaPermisosDeAcces
```

```
public class JpnParametrosParaPermisosDeAcceso
extends PanelParametrosParaReporte
```

Panel diseñado para la selección de parámetros para el informe "Permisos de acceso"

Constructor Summary

JpnParametrosParaPermisosDeAcceso()

Method Summary

protected void	**establecerValoresIniciales**() Establece los valores iniciales de los controles del panel
protected void	**generarReporte**() Genera el reporte a partir de los parámetros indicados
javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del panel
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
protected boolean	**validarParametros**() Valida los parámetros establecidos en los controles

Constructor Detail

JpnParametrosParaPermisosDeAcceso

```
public JpnParametrosParaPermisosDeAcceso()
```

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Description copied from class: PanelParametrosParaReporte
Obtiene el botón por defecto del panel
Specified by:
getBotonPorDefecto in class PanelParametrosParaReporte
Returns:
botón por defecto

---

establecerValoresIniciales

```
protected void establecerValoresIniciales()
```

Description copied from class: PanelParametrosParaReporte
Establece los valores iniciales de los controles del panel
Specified by:
establecerValoresIniciales in class PanelParametrosParaReporte

validarParametros

`protected boolean` **`validarParametros`**`()`

Description copied from class: PanelParametrosParaReporte
Valida los parámetros establecidos en los controles
Specified by:
validarParametros in class PanelParametrosParaReporte
Returns:
'true' si los parámetros se han especificado de manera correcta, 'false' en caso contrario

---

generarReporte

`protected void` **`generarReporte`**`()`

Description copied from class: PanelParametrosParaReporte
Genera el reporte a partir de los parámetros indicados
Specified by:
generarReporte in class PanelParametrosParaReporte

---

inicializarComponentes

`protected void` **`inicializarComponentes`**`()`

Description copied from class: PanelParametrosParaReporte
Inicializa los componentes de la interfaz gráfica
Specified by:
inicializarComponentes in class PanelParametrosParaReporte

---

org.uaccess.gui

Class JpnParametrosParaPersonasRegistradas

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelParametrosParaReporte
                      └
org.uaccess.gui.JpnParametrosParaPersonasRegistradas
```

```
public class JpnParametrosParaPersonasRegistradas
extends PanelParametrosParaReporte
```

Panel diseñado para la selección de parámetros para el informe "Permisos de acceso"

See Also:

Serialized Form

Constructor Summary

JpnParametrosParaPersonasRegistradas()

Method Summary

protected void	**establecerValoresIniciales**() Establece los valores iniciales de los controles del panel
protected void	**generarReporte**() Genera el reporte a partir de los parámetros indicados
javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del panel
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
protected boolean	**validarParametros**() Valida los parámetros establecidos en los controles

Constructor Detail

JpnParametrosParaPersonasRegistradas

```
public JpnParametrosParaPersonasRegistradas()
```

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Description copied from class: PanelParametrosParaReporte

Obtiene el botón por defecto del panel

Specified by:
getBotonPorDefecto in class PanelParametrosParaReporte
Returns:
botón por defecto

---

establecerValoresIniciales

```
protected void establecerValoresIniciales()
```

Description copied from class: PanelParametrosParaReporte
Establece los valores iniciales de los controles del panel
Specified by:
establecerValoresIniciales in class PanelParametrosParaReporte

---

validarParametros

```
protected boolean validarParametros()
```

Description copied from class: PanelParametrosParaReporte
Valida los parámetros establecidos en los controles
Specified by:
validarParametros in class PanelParametrosParaReporte
Returns:
'true' si los parámetros se han especificado de manera correcta, 'false' en caso contrario

---

generarReporte

```
protected void generarReporte()
```

Description copied from class: PanelParametrosParaReporte
Genera el reporte a partir de los parámetros indicados
Specified by:
generarReporte in class PanelParametrosParaReporte

---

inicializarComponentes

```
protected void inicializarComponentes()
```

Description copied from class: PanelParametrosParaReporte
Inicializa los componentes de la interfaz gráfica
Specified by:
inicializarComponentes in class PanelParametrosParaReporte

org.uaccess.gui

Class JpnPersona

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto
                      └org.uaccess.gui.JpnPersona
```

```
public class JpnPersona
extends PanelAgregarOEditarObjeto
```

Panle diseñado para la edición de personas

See Also:

Serialized Form

Constructor Summary

JpnPersona()	Crea un nuevo panel para la edición de una nueva persona
JpnPersona(java.lang.Object registro, boolean esNuevoRegistro)	Crea un nuevo panel a partir de los parámetros especificados

Method Summary

void	**actualizarDatosRelacionados**() Actualiza datos de otras entidades relacionadas a la agregada o editada
protected boolean	**actualizarHuella**(java.lang.String mensaje) Actualiza la huella correspondiente a la persona
protected boolean	**actualizarLogin**(java.lang.String mensaje) Actualiza el login administrativo de la persona
protected void	**ejecutarActualizacion**() Ejecuta la actualización del objeto en la base de datos
protected void	**ejecutarInsercion**() Ejecuta la inserción del objeto a la base de datos

javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del formulario
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
void	**setObjeto**(java.lang.Object objeto) Selecciona el objeto que será editado en el panel
void	**sincronizarControlesConObjeto**() Asigna los valores del objeto a los controles
void	**sincronizarControlesConStatusHuella**() Sincroniza los controles respectivos con el estatus de la captura dactilar de la persona
void	**sincronizarControlesConStatusLogin**() Sincroniza los controles respectivos con el estatus del login administrativo de la persona
void	**sincronizarObjetoConControles**(java.lang.Object objeto) Asigna los valores de los controles al objeto que le sea indicado
protected boolean	**validar**() Ejecuta la validación del objeto que está siendo editado

Constructor Detail

JpnPersona

```
public JpnPersona()
```

Crea un nuevo panel para la edición de una nueva persona

JpnPersona

```
public JpnPersona(java.lang.Object registro,
                  boolean esNuevoRegistro)
```

Crea un nuevo panel a partir de los parámetros especificados

Parameters:

`registro` - registro correspondiente a la persona a editar

`esNuevoRegistro` - bandera que indica si es nuevo registro o no

Method Detail

getBotonPorDefecto

`public javax.swing.JButton` **`getBotonPorDefecto`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Obtiene el botón por defecto del formulario
Specified by:
getBotonPorDefecto in class PanelAgregarOEditarObjeto
Returns:
botón por defecto

---

setObjeto

`public void` **`setObjeto`**`(java.lang.Object objeto)`

Description copied from class: PanelAgregarOEditarObjeto
Selecciona el objeto que será editado en el panel
Overrides:
setObjeto in class PanelAgregarOEditarObjeto
Parameters:
`objeto` - objeto a editar

---

sincronizarControlesConObjeto

`public void` **`sincronizarControlesConObjeto`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Asigna los valores del objeto a los controles
Specified by:
sincronizarControlesConObjeto in class PanelAgregarOEditarObjeto

---

sincronizarObjetoConControles

`public void` **`sincronizarObjetoConControles`**`(java.lang.Object objeto)`

Description copied from class: PanelAgregarOEditarObjeto
Asigna los valores de los controles al objeto que le sea indicado
Specified by:
sincronizarObjetoConControles in class PanelAgregarOEditarObjeto
Parameters:
`objeto` - objeto al que se asignarán los valores, generalmente objeto u objetoOriginal

sincronizarControlesConStatusHuella

`public void` **`sincronizarControlesConStatusHuella`**`()`

Sincroniza los controles respectivos con el estatus de la captura dactilar de la persona

---

sincronizarControlesConStatusLogin

`public void` **`sincronizarControlesConStatusLogin`**`()`

Sincroniza los controles respectivos con el estatus del login administrativo de la persona

---

ejecutarInsercion

`protected void` **`ejecutarInsercion`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la inserción del objeto a la base de datos
Specified by:
ejecutarInsercion in class PanelAgregarOEditarObjeto

---

ejecutarActualizacion

`protected void` **`ejecutarActualizacion`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la actualización del objeto en la base de datos
Specified by:
ejecutarActualizacion in class PanelAgregarOEditarObjeto

---

actualizarHuella

`protected boolean` **`actualizarHuella`**`(java.lang.String mensaje)`

Actualiza la huella correspondiente a la persona
Parameters:
`mensaje` - mensaje resultante de la ejecución de la actualización
Returns:
'true' si la huella se actualizó correctamente, 'false' en caso contrario

---

actualizarLogin

```
protected boolean actualizarLogin(java.lang.String mensaje)
```

Actualiza el login administrativo de la persona

Parameters:

`mensaje` - mensaje resultante de la ejecución de la actualización

Returns:

'true' si el login se actualizó correctamente, 'false' en caso contrario

---

validar

```
protected boolean validar()
```

Description copied from class: PanelAgregarOEditarObjeto

Ejecuta la validación del objeto que está siendo editado

Specified by:

validar in class PanelAgregarOEditarObjeto

Returns:

---

inicializarComponentes

```
protected void inicializarComponentes()
```

Description copied from class: PanelAgregarOEditarObjeto

Inicializa los componentes de la interfaz gráfica

Specified by:

inicializarComponentes in class PanelAgregarOEditarObjeto

---

actualizarDatosRelacionados

```
public void actualizarDatosRelacionados()
```

Description copied from class: PanelAgregarOEditarObjeto

Actualiza datos de otras entidades relacionadas a la agregada o editada

Specified by:

actualizarDatosRelacionados in class PanelAgregarOEditarObjeto

org.uaccess.gui

Class JpnTipoDeDocumento

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
```

```
        └javax.swing.JComponent
            └javax.swing.JPanel
                └
org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto
                    └org.uaccess.gui.JpnTipoDeDocumento
```

```
public class JpnTipoDeDocumento
extends PanelAgregarOEditarObjeto
```

Panel diseñado para la edición de tipos de documento

See Also:
Serialized Form

Constructor Summary

JpnTipoDeDocumento()	Crea un nuevo panel para la edición de un nuevo tipo de documento
JpnTipoDeDocumento(java.lang.Object registro, boolean esNuevoRegistro)	Crea un nuevo panel para la edición de tipos de objeto con los parámetros indicados

Method Summary

void	**actualizarDatosRelacionados**() Actualiza datos de otras entidades relacionadas a la agregada o editada
protected void	**ejecutarActualizacion**() Ejecuta la actualización del objeto en la base de datos
protected void	**ejecutarInsercion**() Ejecuta la inserción del objeto a la base de datos
javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del formulario
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
void	**sincronizarControlesConObjeto**() Asigna los valores del objeto a los controles
void	**sincronizarObjetoConControles**(java.lang.Object objeto) Asigna los valores de los controles al objeto que le sea indicado

protected boolean	**validar**() Ejecuta la validación del objeto que está siendo editado

Constructor Detail

JpnTipoDeDocumento

```
public JpnTipoDeDocumento()
```

Crea un nuevo panel para la edición de un nuevo tipo de documento

---

JpnTipoDeDocumento

```
public JpnTipoDeDocumento(java.lang.Object registro,
                          boolean esNuevoRegistro)
```

Crea un nuevo panel para la edición de tipos de objeto con los parámetros indicados

Parameters:

`registro` - registro correspondiente al tipo de documento a editar

`esNuevoRegistro` - bandera que indica si es nuevo registro o no

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Description copied from class: PanelAgregarOEditarObjeto

Obtiene el botón por defecto del formulario

Specified by:

getBotonPorDefecto in class PanelAgregarOEditarObjeto

Returns:

botón por defecto

---

sincronizarControlesConObjeto

```
public void sincronizarControlesConObjeto()
```

Description copied from class: PanelAgregarOEditarObjeto

Asigna los valores del objeto a los controles

Specified by:

sincronizarControlesConObjeto in class PanelAgregarOEditarObjeto

---

sincronizarObjetoConControles

```
public void sincronizarObjetoConControles(java.lang.Object objeto)
```

Description copied from class: PanelAgregarOEditarObjeto
Asigna los valores de los controles al objeto que le sea indicado
Specified by:
sincronizarObjetoConControles in class PanelAgregarOEditarObjeto
Parameters:
`objeto` - objeto al que se asignarán los valores, generalmente objeto u objetoOriginal

---

ejecutarInsercion

```
protected void ejecutarInsercion()
```

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la inserción del objeto a la base de datos
Specified by:
ejecutarInsercion in class PanelAgregarOEditarObjeto

---

ejecutarActualizacion

```
protected void ejecutarActualizacion()
```

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la actualización del objeto en la base de datos
Specified by:
ejecutarActualizacion in class PanelAgregarOEditarObjeto

---

validar

```
protected boolean validar()
```

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la validación del objeto que está siendo editado
Specified by:
validar in class PanelAgregarOEditarObjeto
Returns:

---

inicializarComponentes

```
protected void inicializarComponentes()
```

Description copied from class: PanelAgregarOEditarObjeto

Inicializa los componentes de la interfaz gráfica
Specified by:
inicializarComponentes in class PanelAgregarOEditarObjeto

---

actualizarDatosRelacionados

`public void` **`actualizarDatosRelacionados`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Actualiza datos de otras entidades relacionadas a la agregada o editada
Specified by:
actualizarDatosRelacionados in class PanelAgregarOEditarObjeto

org.uaccess.gui

Class JpnTipoDeUsuario

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto
                      └org.uaccess.gui.JpnTipoDeUsuario
```

`public class` **`JpnTipoDeUsuario`**
`extends` PanelAgregarOEditarObjeto

Panel diseñado para la edición de tipos de usuario

See Also:
Serialized Form

Constructor Summary

JpnTipoDeUsuario`()` Crea un nuevo panel para la edición de un nuevo tipo de usuario
JpnTipoDeUsuario`(java.lang.Object registro, boolean esNuevoRegistro)` Crea un nuevo panel para la edición de tipos de usuario con los parámetros indicados

Method Summary

void	**actualizarDatosRelacionados**() Actualiza datos de otras entidades relacionadas a la agregada o editada
protected void	**ejecutarActualizacion**() Ejecuta la actualización del objeto en la base de datos
protected void	**ejecutarInsercion**() Ejecuta la inserción del objeto a la base de datos
javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del formulario
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
void	**sincronizarControlesConObjeto**() Asigna los valores del objeto a los controles
void	**sincronizarObjetoConControles**(java.lang.Object objeto) Asigna los valores de los controles al objeto que le sea indicado
void	**sincronizarObjetoConPermisosAdministrativos**() Asigna al tipo de usuario los privilegios administrativos seleccionados
void	**sincronizarObjetoConZonasPermitidas**() Asigna al tipo de usuario las zonas seleccionadas como permitidas
protected boolean	**validar**() Ejecuta la validación del objeto que está siendo editado

Constructor Detail

JpnTipoDeUsuario

```
public JpnTipoDeUsuario()
```

Crea un nuevo panel para la edición de un nuevo tipo de usuario

---

JpnTipoDeUsuario

```
public JpnTipoDeUsuario(java.lang.Object registro,
```

boolean esNuevoRegistro)

Crea un nuevo panel para la edición de tipos de usuario con los parámetros indicados

Parameters:

`registro` - registro correspondiente al tipo de usuario a editar

`esNuevoRegistro` - bandera que indica si es nuevo registro o no

Method Detail

getBotonPorDefecto

`public javax.swing.JButton` **`getBotonPorDefecto`**`()`

Description copied from class: PanelAgregarOEditarObjeto

Obtiene el botón por defecto del formulario

Specified by:

getBotonPorDefecto in class PanelAgregarOEditarObjeto

Returns:

botón por defecto

---

sincronizarControlesConObjeto

`public void` **`sincronizarControlesConObjeto`**`()`

Description copied from class: PanelAgregarOEditarObjeto

Asigna los valores del objeto a los controles

Specified by:

sincronizarControlesConObjeto in class PanelAgregarOEditarObjeto

---

sincronizarObjetoConControles

`public void` **`sincronizarObjetoConControles`**`(java.lang.Object objeto)`

Description copied from class: PanelAgregarOEditarObjeto

Asigna los valores de los controles al objeto que le sea indicado

Specified by:

sincronizarObjetoConControles in class PanelAgregarOEditarObjeto

Parameters:

`objeto` - objeto al que se asignarán los valores, generalmente objeto u objetoOriginal

---

sincronizarObjetoConZonasPermitidas

`public void` **`sincronizarObjetoConZonasPermitidas`**`()`

Asigna al tipo de usuario las zonas seleccionadas como permitidas

---

sincronizarObjetoConPermisosAdministrativos

`public void` **`sincronizarObjetoConPermisosAdministrativos`**`()`

Asigna al tipo de usuario los privilegios administrativos seleccionados

---

ejecutarInsercion

`protected void` **`ejecutarInsercion`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la inserción del objeto a la base de datos
Specified by:
ejecutarInsercion in class PanelAgregarOEditarObjeto

---

ejecutarActualizacion

`protected void` **`ejecutarActualizacion`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la actualización del objeto en la base de datos
Specified by:
ejecutarActualizacion in class PanelAgregarOEditarObjeto

---

validar

`protected boolean` **`validar`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la validación del objeto que está siendo editado
Specified by:
validar in class PanelAgregarOEditarObjeto
Returns:

---

inicializarComponentes

`protected void` **`inicializarComponentes`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Inicializa los componentes de la interfaz gráfica
Specified by:
inicializarComponentes in class PanelAgregarOEditarObjeto

actualizarDatosRelacionados

`public void` **`actualizarDatosRelacionados`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Actualiza datos de otras entidades relacionadas a la agregada o editada

Specified by:
actualizarDatosRelacionados in class PanelAgregarOEditarObjeto

org.uaccess.gui

Class JpnZona

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto
                      └org.uaccess.gui.JpnZona
```

`public class` **`JpnZona`**
`extends` PanelAgregarOEditarObjeto

Panel diseñado para la edición de zonas

See Also:
Serialized Form

Constructor Summary

JpnZona`()`	Crea un nuevo panel para la edición de una nueva zona
JpnZona`(java.lang.Object registro, boolean esNuevoRegistro)`	Crea un nuevo panel para la edición de zonas con los parámetros recibidos

Method Summary

`void`	**actualizarDatosRelacionados**`()` Actualiza datos de otras entidades relacionadas a la agregada o editada

protected void	**ejecutarActualizacion**() Ejecuta la actualización del objeto en la base de datos
protected void	**ejecutarInsercion**() Ejecuta la inserción del objeto a la base de datos
javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del formulario
protected void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
void	**sincronizarControlesConObjeto**() Asigna los valores del objeto a los controles
void	**sincronizarObjetoConControles**(java.lang.Object objeto) Asigna los valores de los controles al objeto que le sea indicado
protected boolean	**validar**() Ejecuta la validación del objeto que está siendo editado

Constructor Detail

JpnZona

```
public JpnZona()
```

Crea un nuevo panel para la edición de una nueva zona

JpnZona

```
public JpnZona(java.lang.Object registro,
               boolean esNuevoRegistro)
```

Crea un nuevo panel para la edición de zonas con los parámetros recibidos

Parameters:

`registro` - registro correspondiente a la zona a editar

`esNuevoRegistro` - bandera que indica si es nuevo registro o no

Method Detail

getBotonPorDefecto

```
public javax.swing.JButton getBotonPorDefecto()
```

Description copied from class: PanelAgregarOEditarObjeto

Obtiene el botón por defecto del formulario
Specified by:
getBotonPorDefecto in class PanelAgregarOEditarObjeto
Returns:
botón por defecto

---

sincronizarControlesConObjeto

`public void` **`sincronizarControlesConObjeto`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Asigna los valores del objeto a los controles
Specified by:
sincronizarControlesConObjeto in class PanelAgregarOEditarObjeto

---

sincronizarObjetoConControles

`public void` **`sincronizarObjetoConControles`**`(java.lang.Object objeto)`

Description copied from class: PanelAgregarOEditarObjeto
Asigna los valores de los controles al objeto que le sea indicado
Specified by:
sincronizarObjetoConControles in class PanelAgregarOEditarObjeto
Parameters:
`objeto` - objeto al que se asignarán los valores, generalmente objeto u objetoOriginal

---

ejecutarInsercion

`protected void` **`ejecutarInsercion`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la inserción del objeto a la base de datos
Specified by:
ejecutarInsercion in class PanelAgregarOEditarObjeto

---

ejecutarActualizacion

`protected void` **`ejecutarActualizacion`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la actualización del objeto en la base de datos
Specified by:
ejecutarActualizacion in class PanelAgregarOEditarObjeto

validar

`protected boolean` **`validar`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Ejecuta la validación del objeto que está siendo editado
Specified by:
validar in class PanelAgregarOEditarObjeto
Returns:

---

inicializarComponentes

`protected void` **`inicializarComponentes`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Inicializa los componentes de la interfaz gráfica
Specified by:
inicializarComponentes in class PanelAgregarOEditarObjeto

---

actualizarDatosRelacionados

`public void` **`actualizarDatosRelacionados`**`()`

Description copied from class: PanelAgregarOEditarObjeto
Actualiza datos de otras entidades relacionadas a la agregada o editada
Specified by:
actualizarDatosRelacionados in class PanelAgregarOEditarObjeto

org.uaccess.gui

Class Principal

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └org.uaccess.gui.Principal
```

```
public class Principal
extends javax.swing.JFrame
```

Ventana principal del sistema

See Also:
Serialized Form

Constructor Summary

Principal() Crea una nueva ventana

Constructor Detail

Principal

```
public Principal()
```

Crea una nueva ventana

Package org.uaccess.persistencia

Class Summary

AdministradorPersistencia	Administrador de datos basado en el API de persistencia de java
Huella	Objeto que define la entidad 'Huella' y su tabla respectiva en el modelo de datos.
LoginAdministrativo	Objeto que define la entidad 'Login administrativo' y su tabla respectiva en el modelo de datos.
Persona	Objeto que define la entidad 'Persona' y su tabla respectiva en el modelo de datos.
PrivilegioAdministrativo	Objeto que define la entidad 'Privilegio administrativo' y su tabla respectiva en el modelo de datos.
TipoDeDocumento	Objeto que define la entidad 'Tipo de documento' y su tabla respectiva en el modelo de datos.
TipoDeUsuario	Objeto que define la entidad 'Tipo de usuario' y su tabla respectiva en el modelo de datos.
Zona	Objeto que define la entidad 'Zona' y su tabla respectiva en el modelo de datos.

org.uaccess.persistencia

Class AdministradorPersistencia

```
java.lang.Object
  └ org.uaccess.persistencia.AdministradorPersistencia
```

```
public class AdministradorPersistencia
extends java.lang.Object
```

Administrador de datos basado en el API de persistencia de java

---

Method Summary

boolean	**actualizar**(java.lang.Object objeto) Actualiza un objeto previamente existente
void	**deshacer**() Obtiene la transacción en proceso y hace un rollback de la misma
void	**ejecutar**() Ejecuta la transacción en proceso
java.util.List	**ejecutarConsultaNativa**(java.lang.String consultaSql, java.lang.Object... parametros) Ejecuta una consulta nativa (consulta SQL)
boolean	**eliminar**(java.lang.Object objeto) Elimina un objeto del administrador de entidades y su respectivo registro en la base de datos
java.sql.Connection	**getConexion**() Obtiene la conexión a la base de datos utilizada por el administrador de persistencia
static AdministradorPersistencia	**getInstancia**() Obtiene la instancia del administrador de persistencia de la aplicación
void	**iniciarTransaccion**() Inicia una transacción sobre los datos
boolean	**insertar**(java.lang.Object objeto) Persiste un nuevo objeto y lo inserta en la base de datos
java.util.List	**obtenerLista**(java.lang.String nombreConsulta,

	java.lang.Object... parametros) Obtiene la lista de objetos resultante de la consulta y sus parámetros
java.lang.Object	**obtenerObjeto**(java.lang.String nombreConsulta, java.lang.Object... parametros) Obtiene un objeto individual resultante de la consulta y sus parámetros
void	**refrescar**(java.lang.Object objeto) Refresca un objeto del administrador de entidades

Method Detail

getInstancia

```
public static AdministradorPersistencia getInstancia()
```

Obtiene la instancia del administrador de persistencia de la aplicación

Returns:

el administrador de persistencia de la aplicación

---

getConexion

```
public java.sql.Connection getConexion()
```

Obtiene la conexión a la base de datos utilizada por el administrador de persistencia

Returns:

conexión JDBC utilizada por el administrador

---

obtenerLista

```
public java.util.List obtenerLista(java.lang.String nombreConsulta,
                                   java.lang.Object... parametros)
```

Obtiene la lista de objetos resultante de la consulta y sus parámetros

Parameters:

`nombreConsulta` - cadena de caracteres que identifica a la consulta

`parametros` - parámetros que complementan a la consulta

Returns:

lista resultante de la ejecución de la consulta

---

obtenerObjeto

```
public java.lang.Object obtenerObjeto(java.lang.String nombreConsulta,
                                       java.lang.Object... parametros)
```

Obtiene un objeto individual resultante de la consulta y sus parámetros

Parameters:

`nombreConsulta` - cadena de caracteres que identifica a la consulta

`parametros` - parámetros que complementan a la consulta

Returns:

objeto resultante de la ejecución de la consulta. Si ocurre un error o ningún objeto satisface la consulta retorna 'null'

---

ejecutarConsultaNativa

```
public java.util.List
ejecutarConsultaNativa(java.lang.String consultaSql,

java.lang.Object... parametros)
```

Ejecuta una consulta nativa (consulta SQL)

Parameters:

`consultaSql` - consulta nativa

`parametros` - parámetros de la consulta

Returns:

Lista de registros resultantes de la consulta

---

iniciarTransaccion

```
public void iniciarTransaccion()
```

Inicia una transacción sobre los datos

---

refrescar

```
public void refrescar(java.lang.Object objeto)
```

Refresca un objeto del administrador de entidades

Parameters:

`objeto` - objeto a refrescar

---

deshacer

```
public void deshacer()
```

Obtiene la transacción en proceso y hace un rollback de la misma

---

ejecutar

```
public void ejecutar()
```

Ejecuta la transacción en proceso

---

insertar

```
public boolean insertar(java.lang.Object objeto)
```

Persiste un nuevo objeto y lo inserta en la base de datos

Parameters:

`objeto` - objeto a insertar

Returns:

'true' si el objeto se ha insertado correctamente, 'false' en caso contrario

---

actualizar

```
public boolean actualizar(java.lang.Object objeto)
```

Actualiza un objeto previamente existente

Parameters:

`objeto` - objeto a actualizar

Returns:

'true' si el objeto se ha actualizado correctamente, 'false' en caso contrario

---

eliminar

```
public boolean eliminar(java.lang.Object objeto)
```

Elimina un objeto del administrador de entidades y su respectivo registro en la base de datos

Parameters:

`objeto` - objeto a eliminar

Returns:

'true' si el objeto se ha eliminado correctamente, 'false' en caso contrario

org.uaccess.persistencia

Class Huella

```
java.lang.Object
  └ org.uaccess.persistencia.Huella
```

All Implemented Interfaces:

java.io.Serializable

```
@Entity
public class Huella
extends java.lang.Object
implements java.io.Serializable
```

Objeto que define la entidad 'Huella' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:

Serialized Form

Constructor Summary

Huella()
Huella(Persona persona)
Huella(Persona persona, byte[] data)

Method Summary

boolean	**equals**(java.lang.Object object)
byte[]	**getData**()
Persona	**getPersona**()
int	**hashCode**()
void	**setData**(byte[] data)
void	**setPersona**(Persona persona)

java.lang.String	**toString**()

Constructor Detail

Huella

public **Huella**()

Huella

public **Huella**(Persona persona)

Huella

public **Huella**(Persona persona,
byte[] data)

Method Detail

getData

public byte[] **getData**()

---

setData

public void **setData**(byte[] data)

---

getPersona

public Persona **getPersona**()

---

setPersona

public void **setPersona**(Persona persona)

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

`equals` in class `java.lang.Object`

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

org.uaccess.persistencia

Class LoginAdministrativo

```
java.lang.Object
  └ org.uaccess.persistencia.LoginAdministrativo
```

All Implemented Interfaces:

java.io.Serializable

```
@Entity
public class LoginAdministrativo
extends java.lang.Object
implements java.io.Serializable
```

Objeto que define la entidad 'Login administrativo' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:

Serialized Form

Constructor Summary

LoginAdministrativo()
LoginAdministrativo(Persona persona)
LoginAdministrativo(Persona persona, java.lang.String login, java.lang.String contrasena)

Method Summary

boolean	**equals**(java.lang.Object object)
java.lang.String	**getContrasena**()
java.lang.String	**getLogin**()
Persona	**getPersona**()
int	**hashCode**()
void	**setContrasena**(java.lang.String contrasena)
void	**setLogin**(java.lang.String login)
void	**setPersona**(Persona persona)
java.lang.String	**toString**()

Constructor Detail

LoginAdministrativo

public **LoginAdministrativo**()

LoginAdministrativo

```
public LoginAdministrativo(Persona persona)
```

LoginAdministrativo

```
public LoginAdministrativo(Persona persona,
                           java.lang.String login,
                           java.lang.String contrasena)
```

Method Detail

getLogin

```
public java.lang.String getLogin()
```

---

setLogin

```
public void setLogin(java.lang.String login)
```

---

getContrasena

```
public java.lang.String getContrasena()
```

---

setContrasena

```
public void setContrasena(java.lang.String contrasena)
```

---

getPersona

```
public Persona getPersona()
```

---

setPersona

```
public void setPersona(Persona persona)
```

---

hashCode

```
public int hashCode()
```

Overrides:
`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:
`equals` in class `java.lang.Object`

---

toString

```
public java.lang.String toString()
```

Overrides:
`toString` in class `java.lang.Object`

org.uaccess.persistencia

Class Persona

```
java.lang.Object
  └ org.uaccess.persistencia.Persona
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable<Persona>

```
@Entity
public class Persona
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable<Persona>
```

Objeto que define la entidad 'Persona' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:
Serialized Form

Constructor Summary

Persona()
Persona(java.lang.Integer id)

Persona(java.lang.Integer id, java.lang.String apellidos, java.lang.String nombres, java.lang.String numeroDeDocumento, boolean activo)

Method Summary

int	**compareTo**(Persona p)
boolean	**equals**(java.lang.Object object)
boolean	**getActivo**()
java.lang.String	**getApellidos**()
java.lang.Integer	**getId**()
java.lang.String	**getNombres**()
java.lang.String	**getNumeroDeDocumento**()
TipoDeDocumento	**getTipoDeDocumento**()
TipoDeUsuario	**getTipoDeUsuario**()
int	**hashCode**()
void	**setActivo**(boolean activo)
void	**setApellidos**(java.lang.String apellidos)
void	**setId**(java.lang.Integer id)
void	**setNombres**(java.lang.String nombres)
void	**setNumeroDeDocumento**(java.lang.String numeroDeDocumen

	to)
void	**setTipoDeDocumento**(TipoDeDocumento tipoDeDocumento)
void	**setTipoDeUsuario**(TipoDeUsuario tipoDeUsuario)
java.lang.String	**toString**()

Constructor Detail

Persona

```
public Persona()
```

Persona

```
public Persona(java.lang.Integer id)
```

Persona

```
public Persona(java.lang.Integer id,
               java.lang.String apellidos,
               java.lang.String nombres,
               java.lang.String numeroDeDocumento,
               boolean activo)
```

Method Detail

getId

```
public java.lang.Integer getId()
```

---

setId

```
public void setId(java.lang.Integer id)
```

---

getApellidos

```
public java.lang.String getApellidos()
```

---

setApellidos

```
public void setApellidos(java.lang.String apellidos)
```

---

getNombres

```
public java.lang.String getNombres()
```

---

setNombres

```
public void setNombres(java.lang.String nombres)
```

---

getNumeroDeDocumento

```
public java.lang.String getNumeroDeDocumento()
```

---

setNumeroDeDocumento

```
public void setNumeroDeDocumento(java.lang.String numeroDeDocumento)
```

---

getActivo

```
public boolean getActivo()
```

---

setActivo

```
public void setActivo(boolean activo)
```

---

getTipoDeDocumento

```
public TipoDeDocumento getTipoDeDocumento()
```

---

setTipoDeDocumento

```
public void setTipoDeDocumento(TipoDeDocumento tipoDeDocumento)
```

---

getTipoDeUsuario

```
public TipoDeUsuario getTipoDeUsuario()
```

---

setTipoDeUsuario

```
public void setTipoDeUsuario(TipoDeUsuario tipoDeUsuario)
```

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

`equals` in class `java.lang.Object`

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

---

compareTo

```
public int compareTo(Persona p)
```

Specified by:

`compareTo` in interface `java.lang.Comparable<Persona>`

org.uaccess.persistencia

Class PrivilegioAdministrativo

```
java.lang.Object
  └ org.uaccess.persistencia.PrivilegioAdministrativo
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<PrivilegioAdministrativo>

```
@Entity
public class PrivilegioAdministrativo
extends java.lang.Object
implements java.io.Serializable,
java.lang.Comparable<PrivilegioAdministrativo>
```

Objeto que define la entidad 'Privilegio administrativo' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:

Serialized Form

Constructor Summary

PrivilegioAdministrativo()
PrivilegioAdministrativo(java.lang.Integer id)
PrivilegioAdministrativo(java.lang.Integer id, java.lang.String nombre, java.lang.String descripcion)

Method Summary

int	**compareTo**(PrivilegioAdministrativo p)
boolean	**equals**(java.lang.Object object)
java.lang.String	**getDescripcion**()
java.lang.Integer	**getId**()

java.lang.String	**getNombre**()
java.util.Collection <TipoDeUsuario>	**getTipoDeUsuarioCollection**()
int	**hashCode**()
void	**setDescripcion**(java.lang.String descripcion)
void	**setId**(java.lang.Integer id)
void	**setNombre**(java.lang.String nombre)
void	**setTipoDeUsuarioCollection**(java.util.Collection<TipoDeUsuario> tipoDeUsuarioCollection)
java.lang.String	**toString**()

Constructor Detail

PrivilegioAdministrativo

```
public PrivilegioAdministrativo()
```

PrivilegioAdministrativo

```
public PrivilegioAdministrativo(java.lang.Integer id)
```

PrivilegioAdministrativo

```
public PrivilegioAdministrativo(java.lang.Integer id,
                                java.lang.String nombre,
                                java.lang.String descripcion)
```

Method Detail

getId

public java.lang.Integer **getId**()

---

setId

public void **setId**(java.lang.Integer id)

---

getNombre

public java.lang.String **getNombre**()

---

setNombre

public void **setNombre**(java.lang.String nombre)

---

getDescripcion

public java.lang.String **getDescripcion**()

---

setDescripcion

public void **setDescripcion**(java.lang.String descripcion)

---

getTipoDeUsuarioCollection

public java.util.Collection<TipoDeUsuario> **getTipoDeUsuarioCollection**()

---

setTipoDeUsuarioCollection

public void
setTipoDeUsuarioCollection(java.util.Collection<TipoDeUsuario> tipoDeUsuarioCollection)

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

`equals` in class `java.lang.Object`

---

compareTo

```
public int compareTo(PrivilegioAdministrativo p)
```

Specified by:

`compareTo` in interface `java.lang.Comparable<PrivilegioAdministrativo>`

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

org.uaccess.persistencia

Class TipoDeDocumento

```
java.lang.Object
  └ org.uaccess.persistencia.TipoDeDocumento
```

All Implemented Interfaces:

java.io.Serializable

```
@Entity
public class TipoDeDocumento
extends java.lang.Object
implements java.io.Serializable
```

Objeto que define la entidad 'Tipo de documento' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:

Serialized Form

Constructor Summary

TipoDeDocumento()
TipoDeDocumento(java.lang.Integer id)
TipoDeDocumento(java.lang.Integer id, java.lang.String nombre, java.lang.String descripcion, java.lang.String formato)

Method Summary

boolean	**equals**(java.lang.Object object)
java.lang.String	**getDescripcion**()
java.lang.String	**getFormato**()
java.lang.Integer	**getId**()
java.lang.String	**getNombre**()
java.util.Collection<Persona>	**getPersonaCollection**()
int	**hashCode**()
void	**setDescripcion**(java.lang.String descripcion)
void	**setFormato**(java.lang.String formato)
void	**setId**(java.lang.Integer id)
void	**setNombre**(java.lang.String nombre)
void	**setPersonaCollection**(java.util.Collection<Persona> personaCollection)

java.lang.String	**toString**()

Constructor Detail

TipoDeDocumento

```
public TipoDeDocumento()
```

TipoDeDocumento

```
public TipoDeDocumento(java.lang.Integer id)
```

TipoDeDocumento

```
public TipoDeDocumento(java.lang.Integer id,
                       java.lang.String nombre,
                       java.lang.String descripcion,
                       java.lang.String formato)
```

Method Detail

getId

```
public java.lang.Integer getId()
```

---

setId

```
public void setId(java.lang.Integer id)
```

---

getNombre

```
public java.lang.String getNombre()
```

---

setNombre

```
public void setNombre(java.lang.String nombre)
```

---

getDescripcion

```
public java.lang.String getDescripcion()
```

---

setDescripcion

```
public void setDescripcion(java.lang.String descripcion)
```

---

getFormato

```
public java.lang.String getFormato()
```

---

setFormato

```
public void setFormato(java.lang.String formato)
```

---

getPersonaCollection

```
public java.util.Collection<Persona> getPersonaCollection()
```

---

setPersonaCollection

```
public void
setPersonaCollection(java.util.Collection<Persona> personaCollection)
```

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

`equals` in class `java.lang.Object`

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

org.uaccess.persistencia

Class TipoDeUsuario

```
java.lang.Object
  └org.uaccess.persistencia.TipoDeUsuario
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<TipoDeUsuario>

```
@Entity
public class TipoDeUsuario
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable<TipoDeUsuario>
```

Objeto que define la entidad 'Tipo de usuario' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:

Serialized Form

Constructor Summary

TipoDeUsuario()
TipoDeUsuario(java.lang.Integer id)
TipoDeUsuario(java.lang.Integer id, java.lang.String nombre, java.lang.String descripcion)

Method Summary

int	**compareTo**(TipoDeUsuario t)
boolean	**equals**(java.lang.Object object)

java.lang.String	**getDescripcion**()
java.lang.Integer	**getId**()
java.lang.String	**getNombre**()
java.util.Collection<PrivilegioAdministrativo>	**getPermisosAdministrativos**()
java.util.Collection<Persona>	**getPersonaCollection**()
java.util.Collection<Zona>	**getZonasPermitidas**()
int	**hashCode**()
void	**setDescripcion**(java.lang.String descripcion)
void	**setId**(java.lang.Integer id)
void	**setNombre**(java.lang.String nombre)
void	**setPermisosAdministrativos**(java.util.Collection<PrivilegioAdministrativo> permisosAdministrativos)
void	**setPermisosAdministrativos**(PrivilegioAdministrativo[] permisosAdministrativos)
void	**setPersonaCollection**(java.util.Collection<Persona> personaCollection)
void	**setZonasPermitidas**(java.util.Collection<Zona> zonasPermitidas)
void	**setZonasPermitidas**(Zona[] zonasPermitidas)
java.lang.String	**toString**()

Constructor Detail

TipoDeUsuario

```
public TipoDeUsuario()
```

TipoDeUsuario

```
public TipoDeUsuario(java.lang.Integer id)
```

TipoDeUsuario

```
public TipoDeUsuario(java.lang.Integer id,
                     java.lang.String nombre,
                     java.lang.String descripcion)
```

Method Detail

getId

```
public java.lang.Integer getId()
```

---

setId

```
public void setId(java.lang.Integer id)
```

---

getNombre

```
public java.lang.String getNombre()
```

---

setNombre

```
public void setNombre(java.lang.String nombre)
```

---

getDescripcion

```
public java.lang.String getDescripcion()
```

---

setDescripcion

```
public void setDescripcion(java.lang.String descripcion)
```

---

getZonasPermitidas

```
public java.util.Collection<Zona> getZonasPermitidas()
```

---

setZonasPermitidas

```
public void
setZonasPermitidas(java.util.Collection<Zona> zonasPermitidas)
```

---

setZonasPermitidas

```
public void setZonasPermitidas(Zona[] zonasPermitidas)
```

---

getPermisosAdministrativos

```
public java.util.Collection<PrivilegioAdministrativo>
getPermisosAdministrativos()
```

---

setPermisosAdministrativos

```
public void
setPermisosAdministrativos(java.util.Collection<PrivilegioAdministrativo>
 permisosAdministrativos)
```

---

setPermisosAdministrativos

```
public void
setPermisosAdministrativos(PrivilegioAdministrativo[] permisosAdministrat
ivos)
```

---

getPersonaCollection

```
public java.util.Collection<Persona> getPersonaCollection()
```

---

setPersonaCollection

```
public void
setPersonaCollection(java.util.Collection<Persona> personaCollection)
```

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

`equals` in class `java.lang.Object`

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

---

compareTo

```
public int compareTo(TipoDeUsuario t)
```

Specified by:

`compareTo` in interface `java.lang.Comparable<TipoDeUsuario>`

org.uaccess.persistencia

Class Zona

`java.lang.Object`

└ **org.uaccess.persistencia.Zona**

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<Zona>

```
@Entity
public class Zona
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable<Zona>
```

Objeto que define la entidad 'Zona' y su tabla respectiva en el modelo de datos. Para mayor información sobre ésta, diríjase a la documentación de la base de datos del sistema

See Also:

Serialized Form

Constructor Summary

Zona()
Zona(java.lang.Integer id)
Zona(java.lang.Integer id, java.lang.String nombre, java.lang.String descripcion)

Method Summary

int	**compareTo**(Zona z)
boolean	**equals**(java.lang.Object object)
java.lang.String	**getDescripcion**()
java.lang.Integer	**getId**()
java.lang.String	**getNombre**()
java.util.Collection<TipoDeUsuario>	**getTipoDeUsuarioCollection**()
int	**hashCode**()

void	**setDescripcion**(java.lang.String descripcion)
void	**setId**(java.lang.Integer id)
void	**setNombre**(java.lang.String nombre)
void	**setTipoDeUsuarioCollection**(java.util.Collection<TipoDeUsuario> tipoDeUsuarioCollection)
java.lang.String	**toString**()

Constructor Detail

Zona

```
public Zona()
```

Zona

```
public Zona(java.lang.Integer id)
```

Zona

```
public Zona(java.lang.Integer id,
            java.lang.String nombre,
            java.lang.String descripcion)
```

Method Detail

getId

```
public java.lang.Integer getId()
```

---

setId

```
public void setId(java.lang.Integer id)
```

---

getNombre

```
public java.lang.String getNombre()
```

---

setNombre

```
public void setNombre(java.lang.String nombre)
```

---

getDescripcion

```
public java.lang.String getDescripcion()
```

---

setDescripcion

```
public void setDescripcion(java.lang.String descripcion)
```

---

getTipoDeUsuarioCollection

```
public java.util.Collection<TipoDeUsuario> getTipoDeUsuarioCollection()
```

---

setTipoDeUsuarioCollection

```
public void
setTipoDeUsuarioCollection(java.util.Collection<TipoDeUsuario> tipoDeUsua
rioCollection)
```

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `java.lang.Object`

---

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

`equals` in class `java.lang.Object`

---

compareTo

```
public int compareTo(Zona z)
```

Specified by:

`compareTo` in interface `java.lang.Comparable<Zona>`

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

Package org.uaccess.utilidades

Class Summary	
CargadorDeReportes	Cargador de reportes de jasper reports
DocumentoLimitado	Documento con límite de longitud
Encriptador	Encriptador de cadenas de texto
FormatoDeTablaPersona	Definidor de formato de la tabla 'personas' y su correspondiente tipo de objeto: 'Persona'
Mensajero	Objeto encargado de mostrar al usuario los mensajes desencadenados durante el uso de la aplicación

org.uaccess.utilidades

Class CargadorDeReportes

```
java.lang.Object
  └org.uaccess.utilidades.CargadorDeReportes
```

```
public class CargadorDeReportes
extends java.lang.Object
```

Cargador de reportes de jasper reports

Constructor Summary

CargadorDeReportes()

Method Summary

java.sql.Connection	**getConexion**() Obtiene la conexión a la base de datos utilizada por el cargador
static CargadorDeReportes	**getInstancia**() Obtiene la instancia del cargador de reportes de la aplicación
void	**mostrarReporte**(java.lang.String rutaReporte, java.lang.String[] nombresParametros, java.lang.Object[] valoresParametros, java.lang.String titulo) Muestra un reporte en el entorno de escritorio y basado en los parámetros
void	**setConexion**(java.sql.Connection conexionBD) Establece la conexión a la base de datos utilizada para poblar los reportes

Constructor Detail

CargadorDeReportes

public **CargadorDeReportes**()

Method Detail

setConexion

public void **setConexion**(java.sql.Connection conexionBD)

Establece la conexión a la base de datos utilizada para poblar los reportes

Parameters:

conexionBD - conexión a la base de datos

---

getConexion

```
public java.sql.Connection getConexion()
```

Obtiene la conexión a la base de datos utilizada por el cargador

Returns:

conexión a la base de datos

---

getInstancia

```
public static CargadorDeReportes getInstancia()
```

Obtiene la instancia del cargador de reportes de la aplicación

Returns:

instancia del cargador de reportes

---

mostrarReporte

```
public void mostrarReporte(java.lang.String rutaReporte,
                           java.lang.String[] nombresParametros,
                           java.lang.Object[] valoresParametros,
                           java.lang.String titulo)
                    throws java.lang.Exception
```

Muestra un reporte en el entorno de escritorio y basado en los parámetros

Parameters:

`rutaReporte` - ruta del fichero .jasper correspondiente al reporte

`nombresParametros` - nombres de los parámetros del reporte a mostrar

`valoresParametros` - valores de los parámetros con que se cargará el reporte

`titulo` - título del reporte a mostrar

Throws:

`java.lang.Exception`

org.uaccess.utilidades

Class DocumentoLimitado

```
java.lang.Object
  └javax.swing.text.AbstractDocument
      └javax.swing.text.PlainDocument
          └org.uaccess.utilidades.DocumentoLimitado
```

All Implemented Interfaces:

java.io.Serializable, javax.swing.text.Document

```
public class DocumentoLimitado
```

```
extends javax.swing.text.PlainDocument
```

Documento con límite de longitud

See Also:

Serialized Form

Constructor Summary

DocumentoLimitado(int limite) Crea un nuevo documento con límite de longitud	

Method Summary

void	**insertString**(int offset, java.lang.String s, javax.swing.text.AttributeSet a) Inserta una cadena de texto en el documento según es especificado por los parámetros si es que ésta no excede el límite de longitud

Constructor Detail

DocumentoLimitado

```
public DocumentoLimitado(int limite)
```

Crea un nuevo documento con límite de longitud

Parameters:

`limite` -

Method Detail

insertString

```
public void insertString(int offset,
                         java.lang.String s,
                         javax.swing.text.AttributeSet a)
                  throws javax.swing.text.BadLocationException
```

Inserta una cadena de texto en el documento según es especificado por los parámetros si es que ésta no excede el límite de longitud

Specified by:

`insertString` in interface `javax.swing.text.Document`

Overrides:

`insertString` in class `javax.swing.text.PlainDocument`

Parameters:

`offset` - desplazamiento del inicio del texto

`s` - cadena a insertar

`a` - atributos para el contexto de inserción

Throws:

`javax.swing.text.BadLocationException`

org.uaccess.utilidades

Class Encriptador

```
java.lang.Object
  └org.uaccess.utilidades.Encriptador
```

```
public class Encriptador
extends java.lang.Object
```

Encriptador de cadenas de texto

Constructor Summary

Encriptador()

Method Summary

static java.lang.String	**cifrar**(java.lang.String algoritmo, java.lang.String texto) Encripta una cadena de texto

Constructor Detail

Encriptador

```
public Encriptador()
```

Method Detail

cifrar

```
public static java.lang.String cifrar(java.lang.String algoritmo,
                                       java.lang.String texto)
```

Encripta una cadena de texto

Parameters:

`algoritmo` - algoritmo a utilizar para la encriptación

`texto` - texto que se desea cifrar

Returns:

texto ya cifrado. Si se recibe una cadena vacía u ocurre un error en el proceso de filtrado, retornará 'null'

org.uaccess.utilidades

Class FormatoDeTablaPersona

```
java.lang.Object
  └ org.uaccess.utilidades.FormatoDeTablaPersona
```

All Implemented Interfaces:

ca.odell.glazedlists.gui.AdvancedTableFormat<Persona>,
ca.odell.glazedlists.gui.TableFormat<Persona>

```
public class FormatoDeTablaPersona
extends java.lang.Object
implements ca.odell.glazedlists.gui.AdvancedTableFormat<Persona>,
ca.odell.glazedlists.gui.TableFormat<Persona>
```

Definidor de formato de la tabla 'personas' y su correspondiente tipo de objeto: 'Persona'

Constructor Summary

FormatoDeTablaPersona`(java.lang.String[] columnas)`	Crea un nuevo definidor de formato para la tabla 'personas'

Method Summary

`java.lang.Class`	**getColumnClass**`(int i)` Obtiene la clase correspondiente al tipo de objeto de la columa especificada

java.util.Comparator	**getColumnComparator**(int i) Obtiene el comparador de objetos para la columa especificada
int	**getColumnCount**() Obtiene la cantidad de columnas de la tabla
java.lang.String	**getColumnName**(int i) Obtiene el nombre de la columna especificada
java.lang.Object	**getColumnValue**(Persona p, int i) Obtiene el valor de la columna especificada para el objeto recibido

Constructor Detail

FormatoDeTablaPersona

```
public FormatoDeTablaPersona(java.lang.String[] columnas)
```

Crea un nuevo definidor de formato para la tabla 'personas'

Parameters:

`columnas` - columnas de la tabla

Method Detail

getColumnClass

```
public java.lang.Class getColumnClass(int i)
```

Obtiene la clase correspondiente al tipo de objeto de la columa especificada

Specified by:

`getColumnClass` in interface `ca.odell.glazedlists.gui.AdvancedTableFormat<Persona>`

Parameters:

`i` - índice de la columna

Returns:

la clase que define el tipo de objeto de la columna

---

getColumnComparator

```
public java.util.Comparator getColumnComparator(int i)
```

Obtiene el comparador de objetos para la columa especificada

Specified by:

`getColumnComparator` in interface
`ca.odell.glazedlists.gui.AdvancedTableFormat<Persona>`

Parameters:

`i` - índice de la columna

Returns:

comparador de objetos

---

getColumnCount

```
public int getColumnCount()
```

Obtiene la cantidad de columnas de la tabla

Specified by:

`getColumnCount` in interface
`ca.odell.glazedlists.gui.TableFormat<Persona>`

Returns:

cantidad de columnas

---

getColumnName

```
public java.lang.String getColumnName(int i)
```

Obtiene el nombre de la columna especificada

Specified by:

`getColumnName` in interface `ca.odell.glazedlists.gui.TableFormat<Persona>`

Parameters:

`i` - índice de la columna

Returns:

nombre de la columna

---

getColumnValue

```
public java.lang.Object getColumnValue(Persona p,
                                       int i)
```

Obtiene el valor de la columna especificada para el objeto recibido

Specified by:

`getColumnValue` in interface
`ca.odell.glazedlists.gui.TableFormat<Persona>`

Parameters:

`p` - persona de la que se desea obtener el valor de una columna

`i` - índice de la columna que posee el valor deseado

Returns:

el valor del campo

org.uaccess.utilidades

Class Mensajero

```
java.lang.Object
  └org.uaccess.utilidades.Mensajero
```

```
public class Mensajero
extends java.lang.Object
```

Objeto encargado de mostrar al usuario los mensajes desencadenados durante el uso de la aplicación

---

Method Summary

static Mensajero	**getInstancia**() Obtiene la instancia del objeto mensajero de la aplicación
java.awt.Component	**getPadre**() Obtiene el componente padre para los mensajes mostrados
boolean	**hacerPregunta**(java.lang.Object pregunta) Muestra un diálogo de pregunta al usuario
void	**mostrarAdvertencia**(java.lang.Object advertencia) Muestra un mensaje de advertencia al usuario
void	**mostrarError**(java.lang.String error, java.lang.Throwable excepcion) Muestra un mensaje de error al usuario
void	**mostrarInformacion**(java.lang.Object informacion) Muestra un mensaje de información al usuario
void	**setPadre**(java.awt.Component padre) Establece el componente padre para los mensajes mostrados

Method Detail

getInstancia

```
public static Mensajero getInstancia()
```

Obtiene la instancia del objeto mensajero de la aplicación
Returns:
mensajero de la aplicación

---

getPadre

```
public java.awt.Component getPadre()
```

Obtiene el componente padre para los mensajes mostrados
Returns:
componente padre

---

setPadre

```
public void setPadre(java.awt.Component padre)
```

Establece el componente padre para los mensajes mostrados
Parameters:
`padre` - the parent to set

---

mostrarAdvertencia

```
public void mostrarAdvertencia(java.lang.Object advertencia)
```

Muestra un mensaje de advertencia al usuario
Parameters:
`advertencia` - advertencia a mostrar

---

mostrarInformacion

```
public void mostrarInformacion(java.lang.Object informacion)
```

Muestra un mensaje de información al usuario
Parameters:
`informacion` - información a mostrar

---

mostrarError

```
public void mostrarError(java.lang.String error,
                         java.lang.Throwable excepcion)
```

Muestra un mensaje de error al usuario
Parameters:
`error` - mensaje de error a mostrar

`excepcion` - excepción ocurrida

---

hacerPregunta

`public boolean` **`hacerPregunta`**`(java.lang.Object pregunta)`

Muestra un diálogo de pregunta al usuario

Parameters:

`pregunta` - pregunta a mostrar

Returns:

'true' si el usuario responde afirmativamente, caso contrario 'false'

Package org.uaccess.utilidades.controles

Class Summary

ComboBox	Control tipo lista desplegable con soporte para manejo de una lista de objetos
Dialogo	Ventana tipo diálogo con soporte para contener un panel previamente creado
DialogoAgregarOEditarObjeto	Ventana tipo diálogo con soporte para contener un panel estándar diseñado para la creación/edición de objetos
InternalFrame	Ventana interna con soporte para contener un panel previamente creado
InternalFramePrivado	Ventana interna con soporte para contener un panel previamente creado y ser mostrado únicamente cuando el usuario posee el permiso requerido
PanelAdministradorDeTabla	Panel con soporte para la administración de los objetos contenidos en una tabla
PanelAgregarOEditarObjeto	Panel auxiliar para agregar o editar un objeto desde PanelAdministradorDeTabla
PanelImagen	Panel con soporte para mostrar imágenes
PanelListaDeObjetos	Panel con soporte para listar objetos en una tabla donde cada fila contiene un objeto y las columnas son sus campos
PanelParametrosParaReporte	Panel genérico para la selección de parámetros para la generación de reportes

PanelSelectorDeObjetos	Panel diseñado para la selección de múltiples objetos
PanelVisorHuella	Panel con soporte para mostrar la imagen de una captura dactilar
PasswordField	Campo de texto diseñado para la edición de contraseñas con límite de longitud
TextArea	Campo de texto multilínea con soporte para definir longitud máxima
TextField	Campo de texto monolínea con soporte para definir la longitud máxima

org.uaccess.utilidades.controles

Class ComboBox

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JComboBox
                  └org.uaccess.utilidades.controles.ComboBox
```

```
public class ComboBox
extends javax.swing.JComboBox
```

Control tipo lista desplegable con soporte para manejo de una lista de objetos

See Also:

Serialized Form

Constructor Summary

ComboBox() Crea un nuevo control ComboBox con soporte para trabajar con una lista de objetos

Method Summary

void	**instalar**(java.util.List lista) Instala el contenido de la lista desplegable

Constructor Detail

ComboBox

`public` **`ComboBox`**`()`

Crea un nuevo control ComboBox con soporte para trabajar con una lista de objetos

Method Detail

instalar

`public void` **`instalar`**`(java.util.List lista)`

Instala el contenido de la lista desplegable

Parameters:

`lista` - lista a instalar en el control

org.uaccess.utilidades.controles

Class Dialogo

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └org.uaccess.utilidades.controles.Dialogo
```

```
public class Dialogo
extends javax.swing.JDialog
implements java.beans.PropertyChangeListener
```

Ventana tipo diálogo con soporte para contener un panel previamente creado

See Also:

Serialized Form

Constructor Summary

`Dialogo``(javax.swing.JPanel panel, java.lang.String titulo)` Crea un nuevo diálogo con tamaño por defecto
`Dialogo``(javax.swing.JPanel panel, java.lang.String titulo, int ancho, int alto, java.net.URL icono)` Crea un nuevo diálogo con las dimensiones indicadas
`Dialogo``(javax.swing.JPanel panel, java.lang.String titulo,`

javax.swing.JButton botonPorDefecto, java.net.URL icono)
Crea un nuevo diálogo con tamaño por defecto y los parámetros indicados

Method Summary

void	**propertyChange**(java.beans.PropertyChangeEvent evt)

Constructor Detail

Dialogo

```
public Dialogo(javax.swing.JPanel panel,
               java.lang.String titulo)
```

Crea un nuevo diálogo con tamaño por defecto

Parameters:

`panel` - panel contenido en el diálogo

`titulo` - título del diálogo

Dialogo

```
public Dialogo(javax.swing.JPanel panel,
               java.lang.String titulo,
               javax.swing.JButton botonPorDefecto,
               java.net.URL icono)
```

Crea un nuevo diálogo con tamaño por defecto y los parámetros indicados

Parameters:

`panel` - panel contenido en el diálogo

`titulo` - título del diálogo

`botonPorDefecto` - botón por defecto del formulario

`icono` - URL del ícono para la ventana

Dialogo

```
public Dialogo(javax.swing.JPanel panel,
               java.lang.String titulo,
               int ancho,
               int alto,
               java.net.URL icono)
```

Crea un nuevo diálogo con las dimensiones indicadas

Parameters:

`panel` - panel contenido en el diálogo
`titulo` - título del diálogo
`ancho` - ancho indicado para la construcción del diálogo
`alto` - altura indicada para la construcción del diálogo

Method Detail

propertyChange

```
public void propertyChange(java.beans.PropertyChangeEvent evt)
```

Specified by:
`propertyChange` in interface `java.beans.PropertyChangeListener`

org.uaccess.utilidades.controles

Class DialogoAgregarOEditarObjeto

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
org.uaccess.utilidades.controles.DialogoAgregarOEditarObjeto
```

```
public class DialogoAgregarOEditarObjeto
extends javax.swing.JDialog
implements java.beans.PropertyChangeListener
```

Ventana tipo diálogo con soporte para contener un panel estándar diseñado para la creación/edición de objetos

See Also:
Serialized Form

Constructor Summary

DialogoAgregarOEditarObjeto`(javax.swing.JPanel panel, java.lang.String titulo)` Crea un nuevo diálogo con soporte para la creación/edición de objetos
DialogoAgregarOEditarObjeto`(javax.swing.JPanel panel, java.lang.String titulo, java.awt.Image icono)` Crea un nuevo diálogo con soporte para la creación/edición de objetos

DialogoAgregarOEditarObjeto(javax.swing.JPanel panel, java.lang.String titulo, java.net.URL urlIcono) Crea un nuevo diálogo con soporte para la creación/edición de objetos	

Method Summary

protected void	**inicializar**(java.lang.String titulo) Inicializa el diálogo con las carácterísticas respectivas
boolean	**mostrar**() Muestra el diálogo para agregar/editar un objeto
void	**propertyChange**(java.beans.PropertyChangeEvent evt) Gestiona los eventos recibidos de los controles escuchados
void	**setVisible**(boolean visible) **Deprecated.** *Este método no es funcional. Para mostrar el diálogo utilice mostrar()*

Field Detail

panel

```
protected javax.swing.JPanel panel
```

Constructor Detail

DialogoAgregarOEditarObjeto

```
public DialogoAgregarOEditarObjeto(javax.swing.JPanel panel,
                                   java.lang.String titulo,
                                   java.awt.Image icono)
```

Crea un nuevo diálogo con soporte para la creación/edición de objetos

Parameters:

`panel` - panel contenido en el diálogo

`titulo` - título del diálogo

`icono` - icono del diálogo

DialogoAgregarOEditarObjeto

```
public DialogoAgregarOEditarObjeto(javax.swing.JPanel panel,
                                   java.lang.String titulo,
                                   java.net.URL urlIcono)
```

Crea un nuevo diálogo con soporte para la creación/edición de objetos

Parameters:

`panel` - panel contenido en el diálogo

`titulo` - título del diálogo

`urlIcono` - URL del ícono del diálogo

DialogoAgregarOEditarObjeto

```
public DialogoAgregarOEditarObjeto(javax.swing.JPanel panel,
                                   java.lang.String titulo)
```

Crea un nuevo diálogo con soporte para la creación/edición de objetos

Parameters:

`panel` - panel contenido en el diálogo

`titulo` - título del diálogo

Method Detail

setVisible

```
public void setVisible(boolean visible)
```

Deprecated. *Este método no es funcional. Para mostrar el diálogo utilice mostrar()*

Overrides:

`setVisible` in class `java.awt.Dialog`

Parameters:

`visible` -

See Also:

`mostrar()`

---

mostrar

```
public boolean mostrar()
```

Muestra el diálogo para agregar/editar un objeto

Returns:

'true' si la operación concluyó satisfactoriamente. Caso contrario, retorna 'false'

---

inicializar

```
protected void inicializar(java.lang.String titulo)
```

Inicializa el diálogo con las carácterísticas respectivas

Parameters:

`titulo` - título del diálogo

propertyChange

`public void` **`propertyChange`**`(java.beans.PropertyChangeEvent evt)`

Gestiona los eventos recibidos de los controles escuchados

Specified by:

`propertyChange` in interface `java.beans.PropertyChangeListener`

Parameters:

`evt` - Evento desencadenado por el componente emisor: "operacionExitosa" u "operacionCancelada". El primero indica que la operación finalizó de manera correcta; el segundo que fue cancelada. Ambos cierran el diálogo

org.uaccess.utilidades.controles

Class InternalFrame

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JInternalFrame
                  └org.uaccess.utilidades.controles.InternalFrame
```

```
public class InternalFrame
extends javax.swing.JInternalFrame
implements java.beans.PropertyChangeListener
```

Ventana interna con soporte para contener un panel previamente creado

See Also:

Serialized Form

Constructor Summary

InternalFrame`(javax.swing.JPanel panel)` Crea un nuevo frame interno con los parámetros indicados
InternalFrame`(javax.swing.JPanel panel, java.lang.String titulo)` Crea un nuevo frame interno con los parámetros indicados
InternalFrame`(javax.swing.JPanel panel, java.lang.String titulo, javax.swing.Icon icono)` Crea un nuevo frame interno con los parámetros indicados

InternalFrame(javax.swing.JPanel panel, java.lang.String titulo, int ancho, int alto) Crea un nuevo frame interno con los parámetros indicados	
InternalFrame(javax.swing.JPanel panel, java.lang.String titulo, int ancho, int alto, java.lang.String icono) Crea un nuevo frame interno con los parámetros indicados	

Method Summary

javax.swing.JPanel	**getPanel**() Obtiene el panel contenido en el frame
void	**propertyChange**(java.beans.PropertyChangeEvent evt) Gestiona los eventos recibidos de los controles escuchados

Constructor Detail

InternalFrame

```
public InternalFrame(javax.swing.JPanel panel)
```

Crea un nuevo frame interno con los parámetros indicados

Parameters:

`panel` - panel que será contenido en el frame

InternalFrame

```
public InternalFrame(javax.swing.JPanel panel,
                     java.lang.String titulo)
```

Crea un nuevo frame interno con los parámetros indicados

Parameters:

`panel` - panel contenido en el frame

`titulo` - título del formulario

InternalFrame

```
public InternalFrame(javax.swing.JPanel panel,
                     java.lang.String titulo,
                     javax.swing.Icon icono)
```

Crea un nuevo frame interno con los parámetros indicados

Parameters:

`panel` - panel contenido en el frame
`titulo` - título del formulario
`icono` - ícono del frame interno

InternalFrame

```
public InternalFrame(javax.swing.JPanel panel,
                     java.lang.String titulo,
                     int ancho,
                     int alto)
```

Crea un nuevo frame interno con los parámetros indicados
Parameters:
`panel` - panel contenido en el frame
`titulo` - título del formulario
`ancho` - ancho para el formulario
`alto` - altura para el formulario

InternalFrame

```
public InternalFrame(javax.swing.JPanel panel,
                     java.lang.String titulo,
                     int ancho,
                     int alto,
                     java.lang.String icono)
```

Crea un nuevo frame interno con los parámetros indicados
Parameters:
`panel` - panel contenido en el frame
`titulo` - título del formulario
`ancho` - ancho para el formulario
`alto` - altura para el formulario
`icono` - ícono para el frame

Method Detail

getPanel

```
public javax.swing.JPanel getPanel()
```

Obtiene el panel contenido en el frame
Returns:
el panel contenido en el InternalFrame

---

propertyChange

`public void` **`propertyChange`**`(java.beans.PropertyChangeEvent evt)`

Gestiona los eventos recibidos de los controles escuchados

Specified by:

`propertyChange` in interface `java.beans.PropertyChangeListener`

Parameters:

`evt` - Evento desencadenado por el componente emisor: "close" para cerrar el frame

org.uaccess.utilidades.controles

Class InternalFramePrivado

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JInternalFrame
                  └org.uaccess.utilidades.controles.InternalFrame
                      └
org.uaccess.utilidades.controles.InternalFramePrivado
```

`public class` **`InternalFramePrivado`**
`extends` `InternalFrame`

Ventana interna con soporte para contener un panel previamente creado y ser mostrado únicamente cuando el usuario posee el permiso requerido

See Also:

Serialized Form

Constructor Summary

InternalFramePrivado`(javax.swing.JDesktopPane desktopPane, javax.swing.JPanel panel, java.lang.String titulo, java.lang.String permisoRequerido)` Crea una nueva ventana privada con los parámetros indicados
InternalFramePrivado`(javax.swing.JDesktopPane desktopPane, javax.swing.JPanel panel, java.lang.String titulo, java.lang.String permisoRequerido, javax.swing.Icon icono)` Crea una nueva ventana privada con los parámetros indicados

Method Summary

void	**mostrar**() Muestra la ventana, si y solamente si el usuario posee el permiso requerido
void	**setVisible**(boolean visible) Método sobreescrito para ser funcional solamente cuando se desea ocultar el frame, caso contrario debe utilizarse mostrar()

Methods inherited from class org.uaccess.utilidades.controles.InternalFrame
getPanel, propertyChange

Constructor Detail

InternalFramePrivado

```
public InternalFramePrivado(javax.swing.JDesktopPane desktopPane,
                            javax.swing.JPanel panel,
                            java.lang.String titulo,
                            java.lang.String permisoRequerido,
                            javax.swing.Icon icono)
```

Crea una nueva ventana privada con los parámetros indicados

Parameters:

`desktopPane` - contendor de la ventana

`panel` - panel contenido en el frame

`titulo` - título de la ventana

`permisoRequerido` - permiso requerido para ser mostrado

`icono` - ícono de la ventana

InternalFramePrivado

```
public InternalFramePrivado(javax.swing.JDesktopPane desktopPane,
                            javax.swing.JPanel panel,
                            java.lang.String titulo,
                            java.lang.String permisoRequerido)
```

Crea una nueva ventana privada con los parámetros indicados

Parameters:

`desktopPane` - contendor de la ventana

`panel` - panel contenido en el frame

`titulo` - título de la ventana

`permisoRequerido` - permiso requerido para ser mostrado

Method Detail

mostrar

```
public void mostrar()
```

Muestra la ventana, si y solamente si el usuario posee el permiso requerido

---

setVisible

```
public void setVisible(boolean visible)
```

Método sobreescrito para ser funcional solamente cuando se desea ocultar el frame, caso contrario debe utilizarse mostrar()

Overrides:

`setVisible` in class `javax.swing.JComponent`

Parameters:

`visible` -

See Also:

`mostrar()`

org.uaccess.utilidades.controles

Class PanelAdministradorDeTabla

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelAdministradorDeTabla
```

```
public class PanelAdministradorDeTabla
extends javax.swing.JPanel
```

Panel con soporte para la administración de los objetos contenidos en una tabla

See Also:

Serialized Form

Constructor Summary

PanelAdministradorDeTabla(java.lang.Class claseDelObjeto, java.lang.String consultaDeDatos, java.lang.String[] nombresDeCampos, java.lang.String[] encabezadosDeColumnas, ca.odell.glazedlists.gui.AdvancedTableFormat formatoDeTabla, int[] camposDeFiltrado, int[] camposDeFiltradoIndividual, PanelAgregarOEditarObjeto panelAgregarOEditar, javax.swing.Icon iconoDelDialogo) Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados * @param claseDelObjeto clase correspondiente al tipo de objeto administrado
PanelAdministradorDeTabla(java.lang.Class claseDelObjeto, java.lang.String consultaDeDatos, java.lang.String[] nombresDeCampos, java.lang.String[] encabezadosDeColumnas, int[] anchosDeColumnas, ca.odell.glazedlists.gui.AdvancedTableFormat formatoDeTabla, int[] camposDeFiltrado, int[] camposDeFiltradoIndividual, PanelAgregarOEditarObjeto panelAgregarOEditar, javax.swing.Icon iconoDelDialogo) Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados
PanelAdministradorDeTabla(java.lang.Class claseDelObjeto, java.lang.String consultaDeDatos, java.lang.String[] nombresDeCampos, java.lang.String[] encabezadosDeColumnas, int[] anchosDeColumnas, int[] camposDeFiltrado, int[] camposDeFiltradoIndividual, PanelAgregarOEditarObjeto panelAgregarOEditar, javax.swing.Icon iconoDelDialogo) Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados
PanelAdministradorDeTabla(java.lang.Class claseDelObjeto, java.lang.String consultaDeDatos, java.lang.String[] nombresDeCampos, java.lang.String[] encabezadosDeColumnas, int[] camposDeFiltrado, int[] camposDeFiltradoIndividual, PanelAgregarOEditarObjeto panelAgregarOEditar, javax.swing.Icon iconoDelDialogo) Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados

Constructor Detail

PanelAdministradorDeTabla

```
public PanelAdministradorDeTabla(java.lang.Class claseDelObjeto,
                                 java.lang.String consultaDeDatos,
                                 java.lang.String[] nombresDeCampos,
```

```
java.lang.String[] encabezadosDeColumnas,
                                    int[] anchosDeColumnas,

ca.odell.glazedlists.gui.AdvancedTableFormat formatoDeTabla,
                                    int[] camposDeFiltrado,
                                    int[] camposDeFiltradoIndividual,

PanelAgregarOEditarObjeto panelAgregarOEditar,
                                    javax.swing.Icon iconoDelDialogo)
```

Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados

Parameters:

`claseDelObjeto` - clase correspondiente al tipo de objeto administrado

`consultaDeDatos` - consulta que devuelve los objetos contenidos en la tabla

`nombresDeCampos` - campos del objeto

`encabezadosDeColumnas` - encabezados de las columnas de la tabla

`anchosDeColumnas` - anchos de las columnas de la tabla

`formatoDeTabla` - formato de la tabla de objetos

`camposDeFiltrado` - campos incluidos en el filtrado de objetos por "todos los campos"

`camposDeFiltradoIndividual` - campos que permiten filtrado individual

`panelAgregarOEditar` - panel utilizado para la creación y edición de objetos

`iconoDelDialogo` - ícono del diálogo de creación y edición de objetos

PanelAdministradorDeTabla

```
public PanelAdministradorDeTabla(java.lang.Class claseDelObjeto,
                                    java.lang.String consultaDeDatos,
                                    java.lang.String[] nombresDeCampos,

java.lang.String[] encabezadosDeColumnas,

ca.odell.glazedlists.gui.AdvancedTableFormat formatoDeTabla,
                                    int[] camposDeFiltrado,
                                    int[] camposDeFiltradoIndividual,

PanelAgregarOEditarObjeto panelAgregarOEditar,
                                    javax.swing.Icon iconoDelDialogo)
```

Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados * @param claseDelObjeto clase correspondiente al tipo de objeto administrado

Parameters:

`consultaDeDatos` - consulta que devuelve los objetos contenidos en la tabla

`nombresDeCampos` - campos del objeto

`encabezadosDeColumnas` - encabezados de las columnas de la tabla

`formatoDeTabla` - formato de la tabla de objetos
`camposDeFiltrado` - campos incluidos en el filtrado de objetos por "todos los campos"
`camposDeFiltradoIndividual` - campos que permiten filtrado individual
`panelAgregarOEditar` - panel utilizado para la creación y edición de objetos
`iconoDelDialogo` - ícono del diálogo de creación y edición de objetos

PanelAdministradorDeTabla

```
public PanelAdministradorDeTabla(java.lang.Class claseDelObjeto,
                                 java.lang.String consultaDeDatos,
                                 java.lang.String[] nombresDeCampos,

java.lang.String[] encabezadosDeColumnas,
                                 int[] anchosDeColumnas,
                                 int[] camposDeFiltrado,
                                 int[] camposDeFiltradoIndividual,

PanelAgregarOEditarObjeto panelAgregarOEditar,
                                 javax.swing.Icon iconoDelDialogo)
```

Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados

Parameters:

`claseDelObjeto` - clase correspondiente al tipo de objeto administrado
`consultaDeDatos` - consulta que devuelve los objetos contenidos en la tabla
`nombresDeCampos` - campos del objeto
`encabezadosDeColumnas` - encabezados de las columnas de la tabla
`anchosDeColumnas` - anchos de las columnas de la tabla
`camposDeFiltrado` - campos incluidos en el filtrado de objetos por "todos los campos"
`camposDeFiltradoIndividual` - campos que permiten filtrado individual
`panelAgregarOEditar` - panel utilizado para la creación y edición de objetos
`iconoDelDialogo` - ícono del diálogo de creación y edición de objetos

PanelAdministradorDeTabla

```
public PanelAdministradorDeTabla(java.lang.Class claseDelObjeto,
                                 java.lang.String consultaDeDatos,
                                 java.lang.String[] nombresDeCampos,
                                java.lang.String[] encabezadosDeColumnas,
                                 int[] camposDeFiltrado,
                                 int[] camposDeFiltradoIndividual,

PanelAgregarOEditarObjeto panelAgregarOEditar,
```

javax.swing.Icon iconoDelDialogo)

Crea un nuevo panel para la administración de una tabla de objetos con los parámetros indicados

Parameters:

`claseDelObjeto` - clase correspondiente al tipo de objeto administrado

`consultaDeDatos` - consulta que devuelve los objetos contenidos en la tabla

`nombresDeCampos` - campos del objeto

`encabezadosDeColumnas` - encabezados de las columnas de la tabla

`camposDeFiltrado` - campos incluidos en el filtrado de objetos por "todos los campos"

`camposDeFiltradoIndividual` - campos que permiten filtrado individual

`panelAgregarOEditar` - panel utilizado para la creación y edición de objetos

`iconoDelDialogo` - ícono del diálogo de creación y edición de objetos

org.uaccess.utilidades.controles

Class PanelAgregarOEditarObjeto

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelAgregarOEditarObjeto
```

```
public abstract class PanelAgregarOEditarObjeto
extends javax.swing.JPanel
```

Panel auxiliar para agregar o editar un objeto desde PanelAdministradorDeTabla

See Also:

Serialized Form

Constructor Summary

PanelAgregarOEditarObjeto(java.lang.Object registro, boolean esNuevoRegistro) Constructor para los elementos del tipo PanelAgregarOEditarObjeto

Method Summary

abstract void	**actualizarDatosRelacionados**()

	Actualiza datos de otras entidades relacionadas a la agregada o editada
protected abstract void	**ejecutarActualizacion**() Ejecuta la actualización del objeto en la base de datos
protected abstract void	**ejecutarInsercion**() Ejecuta la inserción del objeto a la base de datos
abstract javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del formulario
java.lang.Object	**getObjeto**() Obtiene el objeto editado en el panel
protected void	**guardarObjeto**() Guarda el objeto en la base de datos, agregándolo si es nuevo y actualizándolo si ya existe
protected abstract void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
boolean	**isNuevoRegistro**() Obtiene el valor de la bandera 'nuevoRegistro'
void	**setNuevoRegistro**(boolean nuevoRegistro) Selecciona el valor de la bandera 'nuevoRegistro'
void	**setObjeto**(java.lang.Object objeto) Selecciona el objeto que será editado en el panel
abstract void	**sincronizarControlesConObjeto**() Asigna los valores del objeto a los controles
abstract void	**sincronizarObjetoConControles**(java.lang.Object objeto) Asigna los valores de los controles al objeto que le sea indicado
protected abstract boolean	**validar**() Ejecuta la validación del objeto que está siendo editado

Field Detail

nuevoRegistro

```
protected boolean nuevoRegistro
```

Bandera que indica si el objeto que se está editando es nuevo o no

---

objetoOriginal

```
protected java.lang.Object objetoOriginal
```

Variable que almacena el objeto original rebibido para la operación

---

objeto

```
protected java.lang.Object objeto
```

Variable que almacena el objeto resultante de la operación

Constructor Detail

PanelAgregarOEditarObjeto

```
public PanelAgregarOEditarObjeto(java.lang.Object registro,
                                 boolean esNuevoRegistro)
```

Constructor para los elementos del tipo PanelAgregarOEditarObjeto

Parameters:

`registro` - registro original recibido por el panel

`esNuevoRegistro` - indicador de si el registro recibido es nuevo o no

Method Detail

setNuevoRegistro

```
public void setNuevoRegistro(boolean nuevoRegistro)
```

Selecciona el valor de la bandera 'nuevoRegistro'

Parameters:

`nuevoRegistro` - indica si se está editando un registro nuevo o no

---

isNuevoRegistro

```
public boolean isNuevoRegistro()
```

Obtiene el valor de la bandera 'nuevoRegistro'

Returns:

'true' si se está creando un nuevo registro y 'false' si se está editando uno existente

---

setObjeto

`public void` **`setObjeto`**`(java.lang.Object objeto)`

Selecciona el objeto que será editado en el panel

Parameters:

`objeto` - objeto a editar

---

getObjeto

`public java.lang.Object` **`getObjeto`**`()`

Obtiene el objeto editado en el panel

Returns:

objeto resultante del proceso de edición

---

getBotonPorDefecto

`public abstract javax.swing.JButton` **`getBotonPorDefecto`**`()`

Obtiene el botón por defecto del formulario

Returns:

botón por defecto

---

actualizarDatosRelacionados

`public abstract void` **`actualizarDatosRelacionados`**`()`

Actualiza datos de otras entidades relacionadas a la agregada o editada

---

sincronizarControlesConObjeto

`public abstract void` **`sincronizarControlesConObjeto`**`()`

Asigna los valores del objeto a los controles

---

sincronizarObjetoConControles

`public abstract void`
`sincronizarObjetoConControles``(java.lang.Object objeto)`

Asigna los valores de los controles al objeto que le sea indicado

Parameters:

`objeto` - objeto al que se asignarán los valores, generalmente objeto u objetoOriginal

---

validar

```
protected abstract boolean validar()
```

Ejecuta la validación del objeto que está siendo editado

Returns:

---

ejecutarInsercion

```
protected abstract void ejecutarInsercion()
```

Ejecuta la inserción del objeto a la base de datos

---

ejecutarActualizacion

```
protected abstract void ejecutarActualizacion()
```

Ejecuta la actualización del objeto en la base de datos

---

guardarObjeto

```
protected void guardarObjeto()
```

Guarda el objeto en la base de datos, agregándolo si es nuevo y actualizándolo si ya existe

---

inicializarComponentes

```
protected abstract void inicializarComponentes()
```

Inicializa los componentes de la interfaz gráfica

org.uaccess.utilidades.controles

Class PanelImagen

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └org.uaccess.utilidades.controles.PanelImagen
```

```
public class PanelImagen
extends javax.swing.JPanel
```

Panel con soporte para mostrar imágenes

See Also:
Serialized Form

Constructor Summary

PanelImagen()	Crea un nuevo panel con soporte para mostrar imágenes

Method Summary

protected void	**paintComponent**(java.awt.Graphics g) Pinta la imagen sobre el panel
void	**setImagen**(java.net.URL urlImagen) Selecciona la imagen a ser mostrada en el panel

Constructor Detail

PanelImagen

public **PanelImagen**()
Crea un nuevo panel con soporte para mostrar imágenes

Method Detail

setImagen

public void **setImagen**(java.net.URL urlImagen)
Selecciona la imagen a ser mostrada en el panel
Parameters:
urlImagen - URL de la imagen

---

paintComponent

protected void **paintComponent**(java.awt.Graphics g)
Pinta la imagen sobre el panel
Overrides:
paintComponent in class javax.swing.JComponent
Parameters:

g - componente graphics del panel

org.uaccess.utilidades.controles

Class PanelListaDeObjetos

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └org.uaccess.utilidades.controles.PanelListaDeObjetos
```

```
public class PanelListaDeObjetos
extends javax.swing.JPanel
```

Panel con soporte para listar objetos en una tabla donde cada fila contiene un objeto y las columnas son sus campos

See Also:

Serialized Form

Constructor Summary

PanelListaDeObjetos()	Crea un nuevo panel con soporte para listar objetos

Method Summary

boolean	**agregar**(java.lang.Object objeto) Agregar un nuevo objeto a la lista
void	**enlistar**(java.util.Collection elementos) Instala los objetos a ser listados en el panel
int[]	**getAnchosDeLasColumnas**() Obtiene los anchos de las columnas que conforman la lista
java.lang.String[]	**getEncabezadosDeLasColumnas**() Obtiene los encabezados de las columnas
int	**getFilaSeleccionada**() Obtiene el índice de la fila seleccionada.
ca.odell.glazedlists.gui.AdvancedTableFormat	**getFormatoDeTabla**() Obtiene el formato de la tabla en que se listan los

	objetos
ca.odell.glazedlists.EventList	**getLista**() Obtiene la lista de objetos del panel
int	**getModoDeSeleccion**() Obtiene el modo de selección en que trabaja la lista
java.lang.String[]	**getNombresDeLasPropiedades**() Obtiene los nombre de las propiedades del tipo de objeto listado
java.lang.Object	**getObjetoSeleccionado**() Obtiene el objeto seleccionado en la lista.
java.util.List	**getObjetosSeleccionados**() Obtiene los objetos seleccionados en la lista
java.lang.String[]	**getPropiedadesDeFiltrado**() Obtiene las propiedades involucradas en el filtrado de objetos
javax.swing.JTable	**getTabla**() Obtiene la tabla contenida en el panel
int	**obetenerIndice**(java.lang.Object objeto) Obtiene el índice del objeto especificado dentro de la lista
java.lang.Object	**reemplazar**(int indice, java.lang.Object objeto) Reemplaza un objeto contenido en la lista en base a su índice
java.lang.Object	**reemplazar**(java.lang.Object objeto) Reemplaza un objeto contenido en la lista en base a su identificador
boolean	**remover**(java.lang.Object objeto) Remueve un objeto existente en la lista
java.lang.Object	**removerPorIndice**(int indice) Remueve un objeto existente en la lista
void	**setAnchosDeLasColumnas**(int... anchosDeLasColumnas) Selecciona los anchos de las columnas que conforman la lista
void	**setComponenteDeFiltrado**(javax.swing.text.JTextComponent componente) Selecciona el componente tipo texto que será utilizado para filtrar los objetos de la lista

void	**setElementos**(java.util.Collection elementos) Selecciona los elementos contenidos en la lista de objetos
void	**setEncabezadosDeLasColumnas**(java.lang.String... encabezados) Selecciona los encabezados de las columnas de la lista
void	**setFormatoDeTabla**(ca.odell.glazedlists.gui.AdvancedTableFormat formatoDeTabla) Selecciona el formato de la talba en que se listan los objetos
void	**setModoDeSeleccion**(int modoDeSeleccion) Indica el modo de selección en que trabaja la lista ListSelectionModel.SINGLE_SELECTION ListSelectionModel.SINGLE_INTERVAL_SELECTION ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
void	**setNombresDeLasPropiedades**(java.lang.String... nombresDeLasPropiedades) Selecciona los nombres de las propiedades del tipo de objeto listado
void	**setPropiedadesDeFiltrado**(int... filterPropertiesIndexes) Selecciona las propiedades involucradas en el filtrado de objetos
java.lang.Object	**situar**(java.lang.Object objeto) Sitúa un nuevo objeto dentro de la lista

Constructor Detail

PanelListaDeObjetos

```
public PanelListaDeObjetos()
```

Crea un nuevo panel con soporte para listar objetos

Method Detail

getLista

```
public ca.odell.glazedlists.EventList getLista()
```

Obtiene la lista de objetos del panel
Returns:
la lista de objetos contenidos en el panel

---

setElementos

`public void` **`setElementos`**`(java.util.Collection elementos)`

Selecciona los elementos contenidos en la lista de objetos
Parameters:
`elementos` - los elementos a ser contenidos en la lista

---

getEncabezadosDeLasColumnas

`public java.lang.String[]` **`getEncabezadosDeLasColumnas`**`()`

Obtiene los encabezados de las columnas
Returns:
los encabezados de las columnas de la lista

---

setEncabezadosDeLasColumnas

`public void` **`setEncabezadosDeLasColumnas`**`(java.lang.String... encabezados)`

Selecciona los encabezados de las columnas de la lista
Parameters:
`encabezados` - para las columnas

---

getPropiedadesDeFiltrado

`public java.lang.String[]` **`getPropiedadesDeFiltrado`**`()`

Obtiene las propiedades involucradas en el filtrado de objetos
Returns:
los nombres de las propiedades de filtrado

---

setPropiedadesDeFiltrado

`public void` **`setPropiedadesDeFiltrado`**`(int... filterPropertiesIndexes)`

Selecciona las propiedades involucradas en el filtrado de objetos
Parameters:
`propiedadesDeFiltrado` - los índices de las propiedades de filtrado

---

getNombresDeLasPropiedades

`public java.lang.String[] **getNombresDeLasPropiedades**()`

Obtiene los nombre de las propiedades del tipo de objeto listado

Returns:

los nombres de las propiedades

---

setNombresDeLasPropiedades

`public void **setNombresDeLasPropiedades**(java.lang.String... nombresDeLasPropiedades)`

Selecciona los nombres de las propiedades del tipo de objeto listado

Parameters:

`los` - nombres de las propiedades

---

getAnchosDeLasColumnas

`public int[] **getAnchosDeLasColumnas**()`

Obtiene los anchos de las columnas que conforman la lista

Returns:

los anchos de las columnas de la lista

---

setAnchosDeLasColumnas

`public void **setAnchosDeLasColumnas**(int... anchosDeLasColumnas)`

Selecciona los anchos de las columnas que conforman la lista

Parameters:

`anchosDeLasColumnas` - los anchos de las columnas de la lista

---

getFormatoDeTabla

`public ca.odell.glazedlists.gui.AdvancedTableFormat **getFormatoDeTabla**()`

Obtiene el formato de la tabla en que se listan los objetos

Returns:

formato de la tabla

---

setFormatoDeTabla

```
public void
setFormatoDeTabla(ca.odell.glazedlists.gui.AdvancedTableFormat formatoDeT
abla)
```

Selecciona el formato de la talba en que se listan los objetos

Parameters:

`formatoDeTabla` - formato de la tabla

---

getTabla

```
public javax.swing.JTable getTabla()
```

Obtiene la tabla contenida en el panel

Returns:

la tabla del panel

---

setComponenteDeFiltrado

```
public void
setComponenteDeFiltrado(javax.swing.text.JTextComponent componente)
```

Selecciona el componente tipo texto que será utilizado para filtrar los objetos de la lista

Parameters:

`componente` - componente tipo texto utilizado para el filtrado

---

getModoDeSeleccion

```
public int getModoDeSeleccion()
```

Obtiene el modo de selección en que trabaja la lista

Returns:

el modo de selección de la lista

---

setModoDeSeleccion

```
public void setModoDeSeleccion(int modoDeSeleccion)
```

Indica el modo de selección en que trabaja la lista
ListSelectionModel.SINGLE_SELECTION
ListSelectionModel.SINGLE_INTERVAL_SELECTION
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION

Parameters:

`modoDeSeleccion` - el modo de selección a utilizar

---

getObjetosSeleccionados

`public java.util.List` **`getObjetosSeleccionados`**`()`

Obtiene los objetos seleccionados en la lista

Returns:

los objetos seleccionados

---

getObjetoSeleccionado

`public java.lang.Object` **`getObjetoSeleccionado`**`()`

Obtiene el objeto seleccionado en la lista. Si hay varios seleccionados, retorna el primero que forma parte de la selección

Returns:

el objeto seleccionado

---

getFilaSeleccionada

`public int` **`getFilaSeleccionada`**`()`

Obtiene el índice de la fila seleccionada. -1 indica que no hay ninguna

Returns:

el índice de la primera fila fila seleccionada

---

enlistar

`public void` **`enlistar`**`(java.util.Collection elementos)`

Instala los objetos a ser listados en el panel

Parameters:

`elementos` - elementos que se desea enlistar

---

agregar

`public boolean` **`agregar`**`(java.lang.Object objeto)`

Agregar un nuevo objeto a la lista

Parameters:

`objeto` - objeto a ser agregado

Returns:

'true' si el objeto se ha agregado correctamente, caso contrario 'false'

---

remover

```
public boolean remover(java.lang.Object objeto)
```

Remueve un objeto existente en la lista

Parameters:

`objeto` - objeto a ser removido

Returns:

'true'si el objeto se ha removido, caso contrario 'false'

---

removerPorIndice

```
public java.lang.Object removerPorIndice(int indice)
```

Remueve un objeto existente en la lista

Parameters:

`indice` - índice del objeto a remover

Returns:

'true'si el objeto se ha removido, caso contrario 'false'

---

situar

```
public java.lang.Object situar(java.lang.Object objeto)
```

Sitúa un nuevo objeto dentro de la lista

Parameters:

`objeto` - el objeto que se desea situar en la lista

Returns:

el elemento ya posicionado si se situó satisfactoriamente, caso contrario 'null'

---

reemplazar

```
public java.lang.Object reemplazar(java.lang.Object objeto)
```

Reemplaza un objeto contenido en la lista en base a su identificador

Parameters:

`objeto` - objeto a reemplazar

Returns:

el elemento ya reemplazado si se realizó satisfactoriamente, caso contrario 'null'

---

reemplazar

```
public java.lang.Object reemplazar(int indice,
                                   java.lang.Object objeto)
```

Reemplaza un objeto contenido en la lista en base a su índice
Parameters:
`indice` - índice del objeto a reemplazar
`objeto` - objeto por el que será reemplazado
Returns:
el elemento ya reemplazado si se realizó satisfactoriamente, caso contrario 'null'

---

obetenerIndice

```
public int obetenerIndice(java.lang.Object objeto)
```

Obtiene el índice del objeto especificado dentro de la lista
Parameters:
`objeto` - objeto del que se desea obtener el índice
Returns:
índice del objeto

org.uaccess.utilidades.controles

Class PanelParametrosParaReporte

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
org.uaccess.utilidades.controles.PanelParametrosParaReporte
```

```
public abstract class PanelParametrosParaReporte
extends javax.swing.JPanel
```

Panel genérico para la selección de parámetros para la generación de reportes

See Also:
Serialized Form

Constructor Summary

PanelParametrosParaReporte()	Contructor para los componentes del tipo 'PanelParametrosParaReporte'

Method Summary

protected void	**cancelar**() Cancela la generación del reporte y emite la orden para que la ventana contenedora del panel sea cerrada
protected abstract void	**establecerValoresIniciales**() Establece los valores iniciales de los controles del panel
protected abstract void	**generarReporte**() Genera el reporte a partir de los parámetros indicados
abstract javax.swing.JButton	**getBotonPorDefecto**() Obtiene el botón por defecto del panel
protected abstract void	**inicializarComponentes**() Inicializa los componentes de la interfaz gráfica
protected abstract boolean	**validarParametros**() Valida los parámetros establecidos en los controles

Constructor Detail

PanelParametrosParaReporte

```
public PanelParametrosParaReporte()
```

Contructor para los componentes del tipo 'PanelParametrosParaReporte'

Method Detail

getBotonPorDefecto

```
public abstract javax.swing.JButton getBotonPorDefecto()
```

Obtiene el botón por defecto del panel

Returns:

botón por defecto

---

establecerValoresIniciales

```
protected abstract void establecerValoresIniciales()
```

Establece los valores iniciales de los controles del panel

---

validarParametros

```
protected abstract boolean validarParametros()
```

Valida los parámetros establecidos en los controles

Returns:

'true' si los parámetros se han especificado de manera correcta, 'false' en caso contrario

---

generarReporte

```
protected abstract void generarReporte()
```

Genera el reporte a partir de los parámetros indicados

---

inicializarComponentes

```
protected abstract void inicializarComponentes()
```

Inicializa los componentes de la interfaz gráfica

---

cancelar

```
protected void cancelar()
```

Cancela la generación del reporte y emite la orden para que la ventana contenedora del panel sea cerrada

org.uaccess.utilidades.controles

Class PanelSelectorDeObjetos

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
```

```
        └javax.swing.JComponent
            └javax.swing.JPanel
                └
```

org.uaccess.utilidades.controles.PanelSelectorDeObjetos

```
public class PanelSelectorDeObjetos
extends javax.swing.JPanel
```

Panel diseñado para la selección de múltiples objetos

See Also:

Serialized Form

Constructor Summary

PanelSelectorDeObjetos()	Crea un nuevo panel con soporte para la selección de múltiples objetos

Method Summary

java.lang.Object[]	**getObjetosSeleccionados**() Obtiene los objetos que han sido seleccionados en el panel
void	**setEncabezadoDisponibles**(java.lang.String encabezado) Selecciona el encabezado de la sección de objetos disponibles del panel
void	**setEncabezadoSeleccionados**(java.lang.String encabezado) Selecciona el encabezado de la sección de objetos seleccionados del panel
void	**setObjetos**(java.lang.Object[] objetos) Establece los objetos listados en el control
void	**setObjetosSeleccionados**(java.lang.Object[] objetos) Establece los objetos seleccionados del control

Constructor Detail

PanelSelectorDeObjetos

```
public PanelSelectorDeObjetos()
```

Crea un nuevo panel con soporte para la selección de múltiples objetos

Method Detail

setEncabezadoDisponibles

`public void` **`setEncabezadoDisponibles`**`(java.lang.String encabezado)`

Selecciona el encabezado de la sección de objetos disponibles del panel

Parameters:

`encabezado` - encabezado de los objetos disponibles

---

setEncabezadoSeleccionados

`public void` **`setEncabezadoSeleccionados`**`(java.lang.String encabezado)`

Selecciona el encabezado de la sección de objetos seleccionados del panel

Parameters:

`encabezado` - encabezado de los objetos seleccionados

---

setObjetos

`public void` **`setObjetos`**`(java.lang.Object[] objetos)`

Establece los objetos listados en el control

Parameters:

`objetos` - objetos contenidos en el panel

---

getObjetosSeleccionados

`public java.lang.Object[]` **`getObjetosSeleccionados`**`()`

Obtiene los objetos que han sido seleccionados en el panel

Returns:

lista de objetos seleccionados

---

setObjetosSeleccionados

`public void` **`setObjetosSeleccionados`**`(java.lang.Object[] objetos)`

Establece los objetos seleccionados del control

Parameters:

`objetos` - objetos que se desea seleccionar

org.uaccess.utilidades.controles

Class PanelVisorHuella

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └org.uaccess.utilidades.controles.PanelVisorHuella
```

```
public class PanelVisorHuella
extends javax.swing.JPanel
```

Panel con soporte para mostrar la imagen de una captura dactilar

See Also:

Serialized Form

Constructor Summary

PanelVisorHuella()	Crea un nuevo panel con soporte para mostrar una captura dactilar

Method Summary

CapturaDactilar	**getCaptura**() Obtiene la captura dactilar que está siendo mostrada en el panel
protected void	**paintComponent**(java.awt.Graphics g) Pinta la imagen sobre el panel.
void	**setBinarizada**(boolean binarizada) Establece el modo en que se muestra la captura dactilar
void	**setCaptura**(CapturaDactilar captura) Selecciona la captura dactilar mostrada en el panel

Constructor Detail

PanelVisorHuella

```
public PanelVisorHuella()
```

Crea un nuevo panel con soporte para mostrar una captura dactilar

Method Detail

setCaptura

`public void` **`setCaptura`**`(CapturaDactilar captura)`

Selecciona la captura dactilar mostrada en el panel

Parameters:

`captura` - captura dactilar a mostrar

---

getCaptura

`public CapturaDactilar` **`getCaptura`**`()`

Obtiene la captura dactilar que está siendo mostrada en el panel

Returns:

captura dactilar contenida en el panel

---

setBinarizada

`public void` **`setBinarizada`**`(boolean binarizada)`

Establece el modo en que se muestra la captura dactilar

Parameters:

`binarizada` - 'true' indica que la huella será mostrada binarizada (blanco y negro), 'false' indica que se mostrará en escala de grises

---

paintComponent

`protected void` **`paintComponent`**`(java.awt.Graphics g)`

Pinta la imagen sobre el panel. Si no hay una captura dactilar disponible, muestra un mensaje que lo indica.

Overrides:

`paintComponent` in class `javax.swing.JComponent`

Parameters:

`g` - componente graphics del panel

org.uaccess.utilidades.controles

Class PasswordField

`java.lang.Object`

```
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.text.JTextComponent
                  └javax.swing.JTextField
                      └javax.swing.JPasswordField
                          └
```

org.uaccess.utilidades.controles.PasswordField

```
public class PasswordField
extends javax.swing.JPasswordField
```

Campo de texto diseñado para la edición de contraseñas con límite de longitud

See Also:

Serialized Form

Constructor Summary

PasswordField()	Crea un nuevo campo de texto con soporte para la edición de contraseñas

Method Summary

void	**setLength**(int length) Establece la longitud máxima del campo de texto

Constructor Detail

PasswordField

```
public PasswordField()
```

Crea un nuevo campo de texto con soporte para la edición de contraseñas

Method Detail

setLength

```
public void setLength(int length)
```

Establece la longitud máxima del campo de texto

Parameters:

`length` - longitud máxima del texto

org.uaccess.utilidades.controles

Class TextArea

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.text.JTextComponent
                  └javax.swing.JTextArea
                      └org.uaccess.utilidades.controles.TextArea
```

```
public class TextArea
extends javax.swing.JTextArea
```

Campo de texto multilínea con soporte para definir longitud máxima

See Also:

Serialized Form

Constructor Summary

TextArea()	Crea un nuevo campo de texto multilínea

Method Summary

java.lang.String	**getText**() Obtiene el texto contenido en el control
void	**setLength**(int length) Selecciona la longitud máxima del texto contenido en el control

Constructor Detail

TextArea

```
public TextArea()
```

Crea un nuevo campo de texto multilínea

Method Detail

setLength

```
public void setLength(int length)
```

Selecciona la longitud máxima del texto contenido en el control

Parameters:

`length` - longitud máxima del texto

---

getText

```
public java.lang.String getText()
```

Obtiene el texto contenido en el control

Overrides:

`getText` in class `javax.swing.text.JTextComponent`

Returns:

texto contenido en el control. Si está vacío retorna 'null'

org.uaccess.utilidades.controles

Class TextField

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.text.JTextComponent
                  └javax.swing.JTextField
                      └org.uaccess.utilidades.controles.TextField
```

```
public class TextField
extends javax.swing.JTextField
```

Campo de texto monolínea con soporte para definir la longitud máxima

See Also:

Serialized Form

Constructor Summary

TextField()

Method Summary

java.lang.String	**getText**() Obtiene el texto contenido en el control
void	**setLength**(int length) Establece la longitud máxima del texto contenido en el control

Constructor Detail

TextField

```
public TextField()
```

Method Detail

setLength

```
public void setLength(int length)
```

Establece la longitud máxima del texto contenido en el control

Parameters:

`length` - longitud máxima del texto

---

getText

```
public java.lang.String getText()
```

Obtiene el texto contenido en el control

Overrides:

`getText` in class `javax.swing.text.JTextComponent`

Returns:

texto contenido en el control. Si está vacío retorna 'null'

ANEXO 3: MANUAL DE REFERENCIA DEL PROGRAMADOR PARA EL SOFTWARE DEL PUNTO DE ACCESO

ConexionBD Class Reference

#include <conexionbd.h>

List of all members.

• Public Member Functions

ConexionBD ()
ConexionBD (QString nombreHost, QString nombreBD, QString nombreUsuario, QString contrasenaUsuario)
bool conectar ()

• Detailed Description

Gestor de la conexión a la base de datos.

Esta clase trabaja con el objeto QSqlDatabase de la aplicación. Proporciona funcionalidad para establecer los parámetros de la conexión y conectarse a la base de datos.

• Constructor & Destructor Documentation

ConexionBD::ConexionBD ( )

Crea un nuevo objeto para la administración de la conexión a la base de datos sin parámetros establecidos.

ConexionBD::ConexionBD (QString *nombreHost*,
QString *nombreBD*,
QString *nombreUsuario*,
QString *contrasenaUsuario*
)

Crea un nuevo objeto para la administración de la conexión a la base a partir de los parámetros recibidos.

Parameters:

nombreHost	Nombre del host servidor de la base de datos.
nombreBD	Nombre de la base de datos.
nombreUsuario	Nombre del usuario de la base de datos.
contrasenaUsuario	Contraseña del usuario que establece la conexión.

• Member Function Documentation

bool ConexionBD::conectar ( )

Establece la conexión a la base de datos.

Returns:

'true' si la conexión se establece de manera correcta. 'false' en caso contrario.

The documentation for this class was generated from the following files:

- UAccess/conexionbd.h
- UAccess/conexionbd.cpp

Configuraciones Class Reference

#include <configuraciones.h>

List of all members.

• Public Slots

void on_btnAceptar_clicked ()
void on_btnCancelar_clicked ()

• Public Member Functions

Configuraciones (QWidget *parent=0)
~Configuraciones ()

Protected Member Functions

void changeEvent (QEvent *e)
void showEvent (QShowEvent *)

Detailed Description

Formulario diseñado para la edición de configuraciones.

El formulario al que corresponde esta clase es el que permite establecer las configuraciones del punto de acceso peatonal.

Constructor & Destructor Documentation

Configuraciones::Configuraciones (QWidget * *parent* = 0)

Crea una nueva ventana para la administración de las configuraciones del punto de acceso peatonal.

Configuraciones::~Configuraciones ( )

Destruye la ventana y libera la memoria utilizada por la misma.

Member Function Documentation

void Configuraciones::changeEvent (QEvent * *e*) [protected]

Controlador de cambios de estado de Qt

void Configuraciones::on_btnAceptar_clicked ( ) [slot]

Evento que indica que el usuario ha terminado de editar las configuraciones y desea guardarlas. A la ocurrencia de este evento se validan las configuraciones seleccionadas y si cumplen con los criterios de validación se guardan y cierra la ventana; si no cumplen, se informa al usuario sobre las infracciones encontradas y la ventana sigue abierta.

void Configuraciones::on_btnCancelar_clicked ( ) [slot]

Evento que indica que el usuario desea cancelar la edición de las configuraciones. A la ocurrencia de este evento se cierra la ventana ignorando los cambios realizados.

void Configuraciones::showEvent (QShowEvent *) [protected]

Controlador del evento desencadenado cuando el formulario es mostrado. A la ocurrencia de este evento se establecen el contenido y valor inicial de los controles.

The documentation for this class was generated from the following files:

- UAccess/configuraciones.h
- UAccess/configuraciones.cpp

GestorTorniquete Class Reference

#include <gestortorniquete.h>

List of all members.

Public Member Functions

	GestorTorniquete (int vid, int pid)
bool	dispositivoEstaPresente ()
bool	dispositivoEstaAbierto ()
bool	buscarDispositivo ()
bool	abrirDispositivo ()
bool	enviarOrden (const char *datos)
bool	cerrarDispositivo ()

Detailed Description

Gestor del torniquete de acceso.

Esta clase es la que define al objeto encargado de gestionar el torniquete del punto de acceso peatonal. Dicho dispositivo debe estar conectado a uno de los puertos USB de la computadora en la que corre el software del punto de acceso. La comunicación entre estos dos elementos vía USB es la gestionada por "GestorTorniquete"

• Constructor & Destructor Documentation

GestorTorniquete::GestorTorniquete (int *vid*
, int *pid*
)

Crea un nuevo gestor para el torniquete de acceso conectado a la computadora a partir del VID (identificador del fabricante) y el PID (identificador del producto) del dispositivo USB conectado.

• Member Function Documentation

bool GestorTorniquete::abrirDispositivo ( )

Abre la comunicación al torniquete de acceso si y solo si el dispositivo se encuentra presente.

Returns:
'true' la comunicación con el dispositivo se ha abierto satisfactiamente. 'false' en caso contrario.

bool GestorTorniquete::buscarDispositivo ( )

Verifica si el dispositivo está conectado al PC haciendo una búsqueda por el identificador estándar USB del dispositivo (combinación de VID y PID).

Returns:
'true' si el dispositivo es encontrado. 'false' en caso contrario.

bool GestorTorniquete::cerrarDispositivo ( )

Cierra la comunicación entre el PC y el torniquete de acceso.

Returns:
'true' si el dispositivo ha sido cerrado. 'false' en caso contrario.

bool GestorTorniquete::dispositivoEstaAbierto ( )

Indica si hay comunicación abierta entre el dispositivo y el PC.

Returns:
'true' si el dispositivo está abierto. 'false' en caso contrario.

bool GestorTorniquete::dispositivoEstaPresente ( )

Indica si el dispositivo de acceso ya ha sido encontrado en el PC.

Returns:
'true' si el dispositivo está presente. 'false' en caso contrario.

bool GestorTorniquete::enviarOrden (const char * *datos*)

Envía una orden al dispositivo si y solamente si hay comunicación abierta entre este y el PC.

Returns:
'true' si los datos fuenron enviados con éxito. 'false' en caso contrario.

The documentation for this class was generated from the following files:

- UAccess/gestortorniquete.h
- UAccess/gestortorniquete.cpp

Login Class Reference

#include <login.h>

List of all members.

Public Slots

void on_btnAceptar_clicked ()
void on_btnCancelar_clicked ()

Public Member Functions

Login (QWidget *parent=0)
~Login ()

bool autenticarUsuario (QString privilegioRequerido)

Protected Member Functions

void changeEvent (QEvent *e)
void showEvent (QShowEvent *)

Detailed Description

Formulario diseñado para la verificación de usuarios del sistema.

El formulario al que corresponde esta clase es el que permite autenticar los usuarios del sistema a partir de un privilegio administrativo indicado para tales fines.

Constructor & Destructor Documentation

Login::Login (QWidget * *parent* = 0)

Crea una nueva ventana destinada a la autenticación de usuarios.

Login::~Login ( )

Destruye la ventana y libera la memoria utilizada por la misma.

Member Function Documentation

bool Login::autenticarUsuario (QString *privilegioRequerido*)

Muestra el formulario y ejecuta la autenticación del usuario.

Returns:
'true' si el usuario es válido. 'false' en caso contrario.

void Login::changeEvent (QEvent * *e*) [protected]

Controlador de cambios de estado de Qt

void Login::on_btnAceptar_clicked ( ) [slot]

Evento que indica que el usuario ha ingresado sus datos y desea autenticarse. A la ocurrencia de este evento se verifica la autenticidad del usuario; si el usuario es válido establece el último resultado a 'true', si no a 'false'. En ambos casos se cierra la ventana y en el caso particular de que el usuario no sea válido se muestra un mensaje de error antes de dicha acción.

void Login::on_btnCancelar_clicked ( ) [slot]

Evento que indica que el usuario desea cancelar el proceso de autenticación. A la ocurrencia de este evento se cierra la ventana.

void Login::showEvent (QShowEvent *) [protected]

Controlador del evento desencadenado cuando el formulario es mostrado. A la ocurrencia de este evento se establece el valor inicial de los controles.

The documentation for this class was generated from the following files:

- UAccess/login.h
- UAccess/login.cpp

Servicio Class Reference

#include <servicio.h>

List of all members.

Signals

void personaAceptada (int id, int id_zona)
void personaDenegada ()
void reintentando ()
void error (QString mensaje)

Public Member Functions

Servicio ()

void stop ()

Public Attributes

int idZona
int idScanner
int postAceptacion
int postDenegacion
struct fp_dscv_dev * scanner

Protected Member Functions

void run ()

Detailed Description

Servicio de reconocimiento de usuarios.

Esta clase define al servicio de reconocimiento de usuarios en base a su captura dactilar. Es servicio de reconocimiento biométrico funciona en un hilo de procesamiento distinto al del resto de la aplicación y utiliza librerías nativas libfprint y libusb. El servicio de autenticación incluye además el desbloqueo del dispositivo de acceso para los usuarios aceptados. Si las configuraciones del sistema no pueden ser leídas, así como si el lector de huellas dactilares o el dispositivo de acceso no se encuentran disponibles el proceso se detiene.
El servicio no maneja elementos de interfaz gráfica. Lo que sucede durante el que hacer del mismo es informado al hilo de procesamiento principal del programa por medio de señales desencadenadas a la ocurrencia de cambios de estado u eventos relevantes; de esta manera, es el proceso principal de la aplicación, a través de los componentes de interfaz gráfica que corren sobre él, el encargado de mostrar la ocurrencia de los diferentes acontecimientos.

Constructor & Destructor Documentation

Servicio::Servicio ( )

Crea el hilo de procesamiento encargado de brindar el servicio de verificación de usuarios y de bloquear o aceptar el paso de los mismos, según sea el caso.

• Member Function Documentation

void Servicio::error (QString *mensaje*) [signal]

Señal que informa la ocurrencia de un error durante la ejecución del servicio

Parameters:

mensaje	Mensaje descriptivo del error ocurrido.

void Servicio::personaAceptada (int *id*, int *id_zona*) [signal]

Señal que informa la aceptación de un usuario.

Parameters:

id	Identificador de la persona aceptada.
id_zona	Identificador de la zona a la que la persona ha ingresado.

void Servicio::personaDenegada ( ) [signal]

Señal que informa la denegación de un usuario.

void Servicio::reintentando ( ) [signal]

Señal que informa la reanudación del servicio luedo de la aceptación o denegación de una persona.

void Servicio::run ( ) [protected]

Inicia el servicio de autenticación de usuarios.

void Servicio::stop ( )

Detiene el servicio de autenticación de usuarios.

• Member Data Documentation

int Servicio::idScanner

Almacena el identificador del dispositivo de captura dactilar utilizado por el proceso.

int Servicio::idZona

Almacena el identificador de la zona a la que corresponde el punto de acceso peatonal.

int Servicio::postAceptacion

Almacena el tiempo (en milisegundos) que debe transcurrir entre la aceptación de un usuario y la reanudación del servicio.

int Servicio::postDenegacion

Almacena el tiempo (en milisegundos) que debe transcurrir entre la denegación de un usuario y la reanudación del servicio.

struct fp_dscv_dev* Servicio::scanner

Puntero al dispositivo de captura dactilar utilizado por el programa.

The documentation for this class was generated from the following files:

- UAccess/servicio.h
- UAccess/moc_servicio.cpp
- UAccess/servicio.cpp

Ventana Class Reference

#include <ventana.h>

List of all members.

Public Slots

void mostrarError (QString mensaje)
void solicitarHuella ()
void denegarAcceso ()

void permitirAcceso (int idPersona, int idZona)
void on_btnSalir_clicked ()
void on_btnConfigurar_clicked ()

Public Member Functions

Ventana (QWidget *parent=0)
~Ventana ()

Protected Member Functions

void changeEvent (QEvent *e)
void showEvent (QShowEvent *)
void closeEvent (QCloseEvent *event)
void salir ()
void iniciarServicio ()
void detenerServicio ()

Detailed Description

Formulario principal del punto de acceso peatonal.

El formulario al que corresponde esta clase es el que será mostrado casi la totalidad del tiempo en la pantalla del punto de acceso peatonal. Éste es con el que las personas que desean ingresar al sistema interactúan mostrando los mensajes dirigidos al usuario.

Constructor & Destructor Documentation

Ventana::Ventana (QWidget * *parent* = `0`)

Crea una nueva ventana principal.

Ventana::~Ventana ( )

Destruye la ventana y libera la memoria utilizada por la misma.

- # Member Function Documentation

void Ventana::changeEvent (QEvent * *e*) [protected]

Controlador de cambios de estado de Qt

void Ventana::closeEvent (QCloseEvent * *event*) [protected]

Controlador del evento desencadenado cuando el formulario es cerrado. Esta operación solamente puede ser realizada por un usuario autorizado, puesto que detiene el servicio de reconocimiento y cierra por completo la aplicación. Cuando se procede a cerrar la ventana y con ello finalizar el programa, antes de realizar esta acción se ordena al servicio que se detenga y se espera a que finalice con las tareas en ejecución.

void Ventana::denegarAcceso ( ) [slot]

Informa al usuario que el acceso le ha sido denegado.

void Ventana::detenerServicio ( ) [protected]

Detiene el servicio de control de acceso

void Ventana::iniciarServicio ( ) [protected]

Inicia el servicio de control de acceso peatonal.

void Ventana::mostrarError (QString *mensaje*) [slot]

Muestra al usuario el error recibido.

Parameters:

error a mostrar.

void Ventana::on_btnConfigurar_clicked ( ) [slot]

Evento que indica que el usuario desea configurar el punto de acceso. A la ocurrencia de este evento se solicita al usuario sus credenciales de acceso. Si es un usuario auténtico y posee los permisos para configurar el punto de acceso se abre el diálogo para la edición de las configuraciones. Si no, se muestra un mensaje de error y se ignora la solicitud.

void Ventana::on_btnSalir_clicked ( ) [slot]

Evento que indica que el usuario desea cerrar la aplicación. A la ocurrencia de este evento se invoca el método 'salir()'.

void Ventana::permitirAcceso (int *idPersona*
, int *idZona*
) [slot]

Informa al usuario que el acceso le ha sido concedido. Posteriormente, almacena el ingreso del usuario en la bitácora de acceso.

void Ventana::salir ( ) [protected]

Ordena el cierre de la aplicación. Antes de realizar esta acción se ordena al servicio que se detenga y se espera a que finalice con las tareas en ejecución.

void Ventana::showEvent (QShowEvent *) [protected]

Controlador del evento desencadenado cuando el formulario es mostrado. A la ocurrencia de este evento se inicia el servicio para el reconocimiento de personas y, según sea el caso, permitirles o denegarles el acceso.

void Ventana::solicitarHuella ( ) [slot]

Solicita al usuario la lectura de su huella dactilar.

The documentation for this class was generated from the following files:

- UAccess/ventana.h
- UAccess/ventana.cpp