UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROGRAMACIÓN DE APLICACIONES PARA DISPOSITIVOS MÓVILES

PRESENTADO POR:

NELSON EDUARDO HERNÁNDEZ GERMÁN

PARA OPTAR AL TITULO DE:

INGENIERO DE SISTEMAS INFORMÁTICOS

CIUDAD UNIVERSITARIA, DICIEMBRE DE 2006

UNIVERSIDAD DE EL SALVADOR

RECTORA :

DRA. MARÍA ISABEL RODRÍGUEZ

SECRETARIA GENERAL :

LICDA. ALICIA MARGARITA RIVAS DE RECINOS

FACULTAD DE INGENIERÍA Y ARQUITECTURA

DECANO :

ING. MARIO ROBERTO NIETO LOVO

SECRETARIO :

ING. OSCAR EDUARDO MARROQUÍN HERNÁNDEZ

ESCUELA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

DIRECTOR :

ING. JULIO ALBERTO PORTILLO

UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Trabajo de Graduación previo a la opción al Grado de:

INGENIERO DE SISTEMAS INFORMÁTICOS

Título :

PROGRAMACIÓN DE APLICACIONES PARA DISPOSITIVOS MÓVILES

Presentado por :

NELSON EDUARDO HERNÁNDEZ GERMÁN

Trabajo de Graduación aprobado por :

Docente Director :

ING. JULIO ALBERTO PORTILLO

San Salvador, diciembre de 2006

Trabajo de Graduación Aprobado por:

Docente Director :

ING. JULIO ALBERTO PORTILLO

Mas a Dios gracias, el cual nos lleva siempre en triunfo
en Cristo Jesús, y por medio de nosotros manifiesta
en todo lugar el olor de su conocimiento.
2 Corintios 2:14

Indice General

Introducción i
Objetivos ii
Objetivo General ii
Objetivos Específicos ii
Alcance iii
Limitaciones iv
1. Dispositivos Móviles 1
1.1. Sistemas Operativos para Dispositivos Móviles 1
1.2. Emuladores de Dispositivos Móviles 3
1.3. Desarrollo de Aplicaciones para Dispositivos Móviles 3
1.3.1. Lenguajes de Programación 4
1.3.2. Gestores de Bases de Datos 7
2. Desarrollo de la Aplicación Prototipo 9
2.1. Descripción de la Aplicación 9
2.1.1. Objetivos de la Aplicación 9
2.2. Requerimientos 9
2.2.1. Funciones de la Aplicación 9
2.2.2. Atributos de la Aplicación 10
2.2.3. Identificación de Casos de Uso 10
2.2.4. Descripción de Procesos 11
2.2.5. Diagrama de Casos de Uso 15
2.2.6. Casos de Uso Narrados 16
2.2.7. Modelo Conceptual 26
2.2.8. Glosario Conceptual 27
2.3. Análisis 28
2.3.1. Diagramas de Secuencia 29
2.3.2. Contratos de Operación 36
2.3.3. Diagramas de Estado 50
2.3.4. Diagramas de Actividades 53
2.3.5. Modelo Conceptual Refinado 59
2.4. Diseño 74

2.4.1. Descripción de Casos de Uso Reales 75
2.4.2. Diagramas de Colaboración 85
2.4.3. Diagrama de Clases 94
2.4.4. Diseño de la Base de Datos 95
2.4.5. Arquitectura de la Aplicación 102
2.5. Diseño de Datos 104
2.6. Diseño de la Interfaz de Usuario 130
2.7. Codificación y Pruebas 138
2.7.1. Tecnologías Empleadas en el Desarrollo de la Aplicación 138
2.7.2. Herramientas Utilizadas en el Proceso de Desarrollo 139
2.7.3. Descripción de Archivos 139
2.7.4. Pruebas de la Aplicación 140
2.7.5. Código Fuente 140
3. Manual de Usuario 194
3.1. Iniciando MovilMed 195
3.2. Menú Expediente 196
3.2.1. Nuevo Expediente 197
3.2.2. Editar Expediente 199
3.3. Menú Consulta 203
3.3.1. Nueva Consulta 203
3.3.2. Editar Consulta 203
3.4. Menú Evolución 204
3.4.1. Nueva Evolución 204
3.4.2. Editar Evolución 204
3.5. Menú Listados 205
3.6. Menú Sincronizar 205
3.7. Menú Administrar 206
3.7.1. Inicializar DB 207
3.7.2. Cambiar Clave 207
4. Manual de Instalación 209
4.1. Requerimientos de instalación 210
4.2. Instalación a Través de Internet 210
4.3. Instalación Directa 211

4.4. Desinstalación 212
Conclusiones y Trabajo Futuro 213
Conclusiones 213
Trabajo Futuro 213
Apéndice A 214
Tecnologías de programación de aplicaciones con J2ME 214
Arquitectura de J2ME 217
J2SDK 225
J2ME Wireless Toolkit 229
JEDIT 231
Midlets 232
Consideraciones en el desarrollo de midlets 233
Ciclo de vida de un midlet 234
Creación del Midlet HolaMundo 235
Explicación del Midlet HolaMundo 243
Apéndice B 248
PointBase Micro 248
Descarga 248
Instalación 248
Acceso a la Base de Datos con J2ME 249
Glosario 257
Referencias Bibliográficas 263
Índice Alfabético 269

Indice de Figuras y Tablas

Figuras

Figura 2.1 Diagrama de casos de uso 15
Figura 2.2 Modelo conceptual 26
Figura 2.3 Modelo conceptual refinado 59
Figura 2.4 Diagrama de clases 94
Figura 2.5 Diagrama Entidad-Relación 99
Figura 2.6 Modelo fisico de la base de datos 101
Figura 2.7 Arquitectura de la aplicación de fichas médicas 102
Figura 2.8 Ventana_crearExpediente 130
Figura 2.9 Ventana_consulta 131
Figura 2.10 Ventana_menuAntecedentes 131
Figura 2.11 Ventana_antecedentesPersonales 132
Figura 2.12 Ventana_antecedentesFamiliares 132
Figura 2.13 Ventana_menuTratamiento 133
Figura 2.14 Ventana_tratamiento 133
Figura 2.15 Ventana_citas 134
Figura 2.16 Ventana_citasProgramadas 134
Figura 2.17 Ventana_consultaEvolucion 135
Figura 2.18 Ventana_menuExamen 135
Figura 2.19 Ventana_examen 136
Figura 2.20 Ventana_tipoListado 136
Figura 2.21 Ventana_listado 137
Figura 2.22 Ventana_sincronizacion 137
Figura 3.1 Ingreso de clave de la aplicación 195
Figura 3.2 Inicio de la aplicación. (a) Icono de la aplicación, (b) Menú principal 196
Figura 3.3 Pantalla de error por clave incorrecta 196
Figura 3.4 Menú de adición y modificación 197
Figura 3.5 Pantalla de adición de expediente 197
Figura 3.6 Selección de fecha 198

Figura 3.7 Pantalla 2 de adición de expediente 198
Figura 3.8 Mensaje de error en ingreso de datos 199
Figura 3.9 Mensaje de registro guardado 199
Figura 3.10 Pantalla de selección de expediente 200
Figura 3.11 Pantalla de edición de expediente 201
Figura 3.12 Pantalla 2 de edición de expediente 201
Figura 3.13 Mensaje de eliminación de registro 202
Figura 3.14 Lista de expedientes 202
Figura 3.15 Pantalla de adición de consulta 203
Figura 3.16 Pantalla de adición de evolución 204
Figura 3.17 Pantalla fechas listado 205
Figura 3.18 Pantalla de sincronización 206
Figura 3.19 Pantalla de administración 206
Figura 3.20 Confirmación de inicialización de la base de datos 207
Figura 3.21 Pantalla para cambiar la clave 208
Figura A. 1 Ediciones de Java 2 215
Figura A. 2 Tecnología Java 216
Figura A. 3 Arquitectura J2ME 217
Figura A. 4 Variables de entorno configuradas 227
Figura A. 5 Variables de entorno no configuradas 227
Figura A. 6 Configuración de variables de entorno 228
Figura A. 7 Selección de directorio para extracción de archivos 231
Figura A. 8 Confirmación de reemplazo de archivos 231
Figura A. 9 Cuerpo de un midlet 233
Figura A. 10 Ciclo de vida de un midlet 235
Figura A. 11 Entorno KToolbar 236
Figura A. 12 Nombres del proyecto y clase 236
Figura A. 13 Configuración del proyecto 237
Figura A. 14 Mensajes de creación del proyecto 237
Figura A. 15 Estructura de directorios del proyecto 238
Figura A. 16 (a) Menú para ejecutar la aplicación, (b) Aplicación en ejecución 241
Figura B. 1 Directorios de PointBase 249

Figura B. 2 Proyecto eisi 250
Figura B. 3 Clases del proyecto eisi 253
Figura B. 4 Propiedades del archivo esi.jar 254
Figura B. 5 Tamaño del archivo eisi.jar 255

Tablas

Tabla 2.1 Categorías de las funciones 9
Tabla 2.2 Funciones de la aplicación 10
Tabla 2.3 Atributos de la aplicación 10
Tabla 2.4 Glosario conceptual 27
Tabla 2.5 Descripción de archivos con conforman la aplicación 140
Tabla A. 1 Paquetes de los perfiles MID 224
Tabla A. 2 Descripción de atributos del archivo .jar 243

Introducción

Los grandes avances en la ciencia y la tecnología han contribuido a la evolución de los sistemas computarizados, los dispositivos electrónicos se vuelven cada vez más pequeños y con mayores capacidades a la vez que su costo disminuye, volviéndose asequibles para la mayoría de la población. Hoy en día existen dispositivos dotados de capacidades de procesamiento de datos que caben en la palma de la mano, mismos que son conocidos como dispositivos móviles. Entre estos dispositivos se destacan los PDA (ayudante personal digital por sus siglas en inglés) y los *SmartPhones*.

Los dispositivos móviles constituyen poderosas herramientas, que aunadas con recurso humano calificado permiten lograr eficiencia y efectividad en las actividades de cualquier tipo de organización, ya que tienden a reducir el tiempo y esfuerzo en la captura de datos lo que finalmente se traduce en reducción de costos.

Con el propósito de aportar tanto conocimiento en las, relativamente incipientes, tecnologías relacionadas con el desarrollo de aplicaciones para dispositivos móviles; como nuevas soluciones a nuestra sociedad, se ha desarrollado una aplicación de tipo corporativa. Dicha aplicación está destinada a médicos, de forma tal, que un galeno pueda llevar un registro de los datos personales y características del paciente, el motivo de la consulta, antecedentes personales y familiares relacionados con el padecimiento, así como las distintas etapas del tratamiento.

Se han determinado los requerimientos para la aplicación, se ha elaborado el análisis y diseño y se ha codificado empleado *Java 2 Micro Edition* (J2ME), el cual es un lenguaje de programación que permite una alta portabilidad, es decir, que las aplicaciones desarrolladas pueden correr en diversos dispositivos. Así mismo se ha elaborado la memoria del proyecto siguiendo los lineamientos establecidos en la norma española UNE 157001:2002 según lo establece el Reglamento General para la Realización de los Proyectos de Fin de Carrera de las Titulaciones de la Escuela Superior de Ingeniería de Cádiz.

Objetivos

Objetivo General

Realizar una investigación y aplicación de las tecnologías disponibles relacionadas con el desarrollo de aplicaciones para dispositivos móviles.

Objetivos Específicos

- Recabar información sobre las tecnologías para el desarrollo de aplicaciones para dispositivos móviles.
- Determinar los requerimientos para la aplicación prototipo.
- Realizar el análisis de los requerimientos y el diseño de la aplicación prototipo.
- Codificar la aplicación prototipo empleando las tecnologías de desarrollo investigadas.
- Probar la aplicación desarrollada, en dispositivos móviles.
- Elaborar manuales de instalación y de usuario de la aplicación.

Alcance

Para el desarrollo de aplicaciones para dispositivos móviles existen tecnologías propietarias no liberadas y tecnologías liberadas de uso comercial, la investigación se enmarcará en estas últimas ya que es sobre ellas que está disponible información para su explotación.

La codificación de la aplicación se hará empleando el lenguaje de programación que sea más flexible, es decir aquel lenguaje que sea más independiente del hardware, esto con el propósito de garantizar la mayor portabilidad de las aplicaciones de un dispositivo a otro.

Es importante recalcar que existen diferentes tipos de aplicaciones para dispositivos móviles, a saber: compras en contexto, mapas y navegación, búsqueda y acceso a Internet, comunicaciones y chat, juegos, y aplicaciones corporativas; no obstante la aplicación que se desarrollará para demostrar el uso de las tecnologías investigadas se centrará en el último tipo, es decir aplicaciones corporativas, tal es el caso de la aplicación de fichas médicas.

Limitaciones

Para el desarrollo del proyecto se econtró la siguiente limitante.

Los gestores de bases de datos disponibles en el mercado para dispositivos móviles no proveen un subconjunto de interfaces de programación que permita establecer conexiones desde las aplicaciones a la base de datos; excepto el gestor PointBase Micro, el cual en su versión de evaluación limita a 25 el número de registros por tabla.

1. Dispositivos Móviles

Cuando se habla de dispositivos móviles, se está refiriendo principalmente a los PDA (asistentes personales digitales por sus siglas en inglés) y los *SmartPhones* o teléfonos inteligentes.

Un PDA es una especie de computadora de mano que inicialmente fue diseñada con el propósito de ser una agenda electrónica, pero actualmente es mucho más que una agenda, ya que en estos dispositivos es posible navegar en la web, ver y editar documentos, ver películas, entre otras cosas. Fue en el año de 1992 cuando Apple presentó el primer PDA, sin embargo este fue un fracaso; el éxito estaba reservado para la empresa Palm, la cual presentó su versión de PDA en el año de 1995. Actualmente existen PDAs que corren el sistema operativo de Palm, otros que corren el sistema Pocket PC de Microsoft y algunos que funcionan bajo el sistema operativo Symbian.

Por otro lado los *SmartPhones* son dispositivos que integran la funcionalidad de un teléfono móvil y un PDA, algunos de estos aparatos corren el sistema operativo Symbian.

Conforme transcurre el tiempo, salen al mercado nuevos dispositivos móviles que superan en gran medida a sus antecesores en lo concerniente a capacidades de procesamiento y almacenamiento. Así mismo, en los últimos años se han liberado interfaces de programación que facilitan el desarrollo de aplicaciones que corran en estos dispositivos.

1.1. Sistemas Operativos para Dispositivos Móviles

A continuación se hace una breve descripción de los principales sistemas operativos que corren en dispositivos móviles.

Palm OS[1]

[1] Ver referencias: [WIK06-3], [PAL06-1], [PAL06-2], [PAL06-3]

Palm OS es un sistema operativo desarrollado por la compañía PalmSource, y es uno de los más populares debido a su simplicidad y a que fue la versión de PDA de esta compañía la primera en alcanzar el éxito.

Este sistema operativo es el que corre en los PDAs de la marca Palm, sin embargo existen dispositivos de otras marcas que también lo emplean, entre ellos están los de las marcas: Sony, IBM, Samsung, Handera y Kyocera.

Actualmente el desarrollo de Palm se divide en dos entornos principales: Palm OS Garnet (Versión 5) y Palm OS Cobalt (Versión 6).

Entre las características de este sistema operativo se destacan: sistema de 32 bits, enteramente multihilo y multitarea, multimedia, telefonía, protocolos de comunicación, criptografía y gestión de certificados.

Para garantizar la compatibilidad con programas antiguos, las aplicaciones se ejecutan en un entorno emulado conocido como PACE (Entorno de compatibilidad de aplicaciones Palm, por sus siglas en inglés).

El sistema operativo Palm hace uso de un código de 4 caracteres alfanuméricos conocido como identificador de autor o Creador ID para identificar una aplicación. Los desarrolladores pueden registrar su Creador ID en spp.palmos.com/iws.

Windows Mobile[2]

Windows Mobile o WM es un sistema operativo desarrollado por Microsoft y está basado en l tecnología Windows CE O/S. WM 5.0 está basado en Windows CE versión 5. Microsoft ha sacado al mercado diferentes versiones de sistema operativo para dispositivos móviles: Microsoft Windows CE 1.0, Pocket PC 2002. Pocket PC 2002, Windows Mobile 2003, Windows Mobile 5.0, entre otros.

La versión 5.0 de WM incluye: controlador para ethernet, Wi-Fi, soporte para WPA, soporte para VPN, API para criptografía (basada en RSA), telefonía, soporte XML.

[2] Ver referencias: [WIK06-1], [WIK06-2], [MIK06], [MSC06-1], [PMX06],

Symbian OS[3]

Symbian OS es un potente sistema operativo, no obstante, por razones más bien de marketing no alcanzado la popularidad debida. Por ahora, Nokia emplea Symbian para sus teléfonos celulares, sin embargo existen en el mercado otros dispositivos que también utilizan este sistema, tal es el caso de Siemens.

La versión 9.3 del sistema operativo Symbian posee un kernel multi tarea, soporte de telefonía integrado, multimedia, protocolos de comunicación, manejo de datos, soporte de gráficos en 3D, amplio soporte de Java, criptografía y manejo de certificados, envío de música a auriculares a través de tecnología inalámbrica.

1.2. Emuladores de Dispositivos Móviles

Un punto clave al momento de desarrollar aplicaciones para dispositivos móviles es la existencia de aplicaciones conocidas como emuladores, ya que, por lo general no es posible contar con todos los modelos de dispositivos móviles para probar los desarrollos realizados. Estas aplicaciones están disponibles para sistemas de escritorio y emulan tanto el sistema operativo como la funcionalidad del dispositivo. Existen entornos de desarrollo o IDEs que incorporan emuladores de dispositivos móviles; con estos IDEs no solo es posible la edición y depuración del código fuente, sino que también la previsualización en un dispositivo simulado de la ejecución del código.

Algunos fabricantes de dispositivos móviles también han liberado emuladores que corren en computadoras de escritorio, como ocurre con el emulador oficial de Palm llamado POSE (Palm OS Emulator), o con Microsoft Device Emulator 1.0 que emula el Windows Mobile.

1.3. Desarrollo de Aplicaciones para Dispositivos Móviles

Actualmente, existen en el mercado diversos lenguajes de programación para dispositivos móviles, que van desde C y C++ hasta Basic, pasando por Java. Las aplicaciones pueden

[3] Ver referencias: [SYM06]

ser cómodamente desarrolladas en sistemas de escritorio y probadas empleando emuladores, para finalmente instalarlas en los dispositivos reales.

En lo que al almacenamiento respecta, existe un mecanismo propio de los dispositivos móviles conocido como *Record Management System* (RMS), es semejante a una base de datos, pero no soporta instrucciones SQL.

Por otro lado, existen servidores de bases de datos relacionales para estos dispositivos, entre ellos: HSQL Database Engine, SQL Anywhere Studio, IBM DB2 Everyplace, Oracle9i Lite y PointBase Micro Edition.

1.3.1. Lenguajes de Programación[4]

El desarrollo de aplicaciones para dispositivos móviles usualmente se lleva a cabo en plataformas para sistemas de escritorio, siendo posible el desarrollo bajo Windows, Mac OS, Unix y Linux. Así como es posible desarrollar en diversas plataformas, también es posible desarrollar en diversos lenguajes de programación; a continuación se hace una breve descripción de los principales lenguajes disponibles en el mercado para el desarrollo de aplicaciones para dispositivos móviles.

Basic

Para aquellos desarrolladores que programan en Basic existen diversas posibilidades, entre ellas se destacan las que se presentan a continuación.

NS Basic/Palm. Con esta herramienta es posible desarrollar todo tipo de aplicaciones para Palm dentro de un entrono de desarrollo visual.

Mobile Visual Basic. Desarrollado por AppForge, permite extender la funcionalidad de Visual Basic, ya que al instalarlo se añade una opción para poder generar proyectos del tipo AppForge, que incorpora además de los controles estándares una serie de elementos específicos para el desarrollo de aplicaciones para Palm.

---

[4] Ver referencias: [MSC06-2], [PCW06], [LNX06], [RIS06], [HAW06], [WIK06-5]

Satellite Forms MobileApp Designer. Es un entorno de desarrollo basado en Visual Basic, que genera código ejecutable ya sea en Palm OS o en Pocket PC 2002.

Scoutbuilder. Basado en el lenguaje de programación Basic con el que se pueden desarrollar aplicaciones para Palm OS de manera rápida. Utiliza Intellisense para ayudar a completar líneas de código automáticamente.

Palm OS Software Development Kit (SDK)

Exclusivamente para desarrollo de aplicaciones que corran sobre Palm OS, es un paquete de desarrollo compuesto, básicamente, de archivos cabecera, librerías con funciones y utilidades básicas. Es importante recalcar que no incluye herramienta de desarrollo. Palm OS SDK está disponible para Windows, Mac OS y Linux.

CodeWarrior

Desarrollado por la compañía Metrowerks, realmente no es un lenguaje de programación, si no, un entorno de desarrollo. CodeWarrior está disponible para Windows y para Mac OS. Con él pueden crearse aplicaciones para todas las versiones del Palm OS. El lenguaje de programación puede ser C o C++.

Java 2 Micro Edition

También abreviado J2ME, fue desarrollado como una respuesta para una plataforma de dispositivos móviles. J2ME es en realidad un subconjunto de Java 2 Standard Edition (J2SE) dirigido a dispositivos con recursos limitados principalmente en lo concerniente a capacidad de procesamiento, memoria y resoluciones de pantalla.

HawHaw

HAWHAW está basado en el lenguaje PHP y permite la publicación de páginas WAP que también son accesible por navegadores HTML estándares. HAWHAW automáticamente determina las capacidades del aparato que está haciendo la petición y crea el código de marcado apropiado.

SuperWaba

Es un lenguaje de programación basado en Java, optimizado para su uso en dispositivos con pantallas pequeñas. Auque incluye su propio conjunto de librerías, los programas desarrollados en SuperWaba pueden ser ejecutados en cualquier plataforma que interprete Java.

SuperWaba es la continuación del proyecto Waba, planteado como una alternativa frente a *Java Micro Edition*.

PRC-Tools

Es un producto de código abierto que consiste en un conjunto de herramientas que, haciendo uso de una versión modificada del SDK, permite el desarrollo de aplicaciones empleando el compilador GCC, ya sea usando C o C++.

Visual Studio .NET

Es un entorno de desarrollo, que emplea una tecnología llamada framework, que es semejante a la Java Virtual Machine (JVM). El framework también está disponible en una versión más liviana conocida como .NET Compact Framework, dirigida principalmente a dispositivos móviles.

PDA ToolBox

Es un entorno de desarrollo gráfico que permite crear aplicaciones para Palm, contiene una gama de elementos de formularios.

CASL

CASL es el acrónimo de Compact Application Solution Language, es un entorno de desarrollo visual que genera código para Palm OS o Pocket PC, aunque las versiones más completas se dirigen a la primera plataforma. El lenguaje empleado en CASL es orientado a objetos y permite incluir segmentos de código C.

Microsoft eMbedded Visual Tools

Consiste en una serie de herramientas proporcionadas por Microsoft para el desarrollo de aplicaciones para dispositivos móviles, e incluye: Pocket PC SDK, Handheld PC SDK y

Palm PC SDK. Los lenguajes de programación disponibles son Visual C++ y Visual Basic, ambos en ediciones embedded.

HS Pascal

Con HS Pascal es posible producir programas ejecutables directamente sobre Palm OS. En el mercado existen entornos de desarrollo que soportan HS Pascal, tal como Pythia y Poivre.

1.3.2. Gestores de Bases de Datos[5]

Un aspecto de vital importancia en el desarrollo de aplicaciones corporativas es el almacenamiento, la mayoría de aplicaciones de este tipo operan sobre datos, almacenándolos, recuperándolos, actualizándolos, manipulándolos y eliminándolos.

Por defecto, los dispositivos móviles soportan un tipo de almacenamiento denominado *Record Management System* (RMS), el cual es semejante a una base de datos, sin embargo no soporta sentencias SQL. En RMS la unidad mínima de almacenamiento es el registro, y su manipulación, en el caso de J2ME, se logra a través de una serie de métodos propios de un paquete denominado rms, el cual permite para operar directamente sobre los registros.

Una mejor opción para almacenar datos en dispositivos móviles, lo constituyen los gestores de bases de datos propios para dichos dispositivos y actualmente existen varias alternativas, a continuación se describen brevemente las principales.

HSQLDB

Es un motor de base de datos relacional escrito en Java, cuenta con un driver JDBC, soporta ANSI-92 SQL. Ofrece bases de datos pequeñas y rápidas con tablas basadas ya sea en memoria o disco. Solamente necesita 170 KB de memoria para correr.

SQL Anywhere Studio

Es una plataforma de base de datos y sincronización desarrollada por iAnywhere Solutions Inc. (Una subsidiaria de Sybase Inc.). Soporta SQL, proporciona manejo de datos y

[5] Ver referencias: [SNS06], [OST06], [IBM06-1], [CWD06], [IBM06-2], [PED06-1], [ORC06-1], [ORC06-2], [PED06-2], [MIC06], [WIK06-4], [SYB06], [SWH06], [IBM06-3]

sincronización de datos corporativos. La edición UltraLite está diseñada para dispositivos con limitaciones de memoria (alrededor de 150 KB), además provee integridad referencial, procesamiento de transacciones y encriptación.

DB2 Everyplace

Propiedad de IBM, es un servidor de base de datos relacional, permite la sincronización de datos, puede ser usada como una base de datos local independiente en un dispositivo móvil, y opcionalmente, se puede contar con un servidor de sincronización empresarial. 200 KB de footprint.

Oracle9i Lite

Es una edición desarrollada específicamente para dispositivos móviles, maneja bases de datos relacionales y ofrece sincronización de datos. Es posible acceder a la base de datos empleando ODBC, ADO, JDBC o Web-to-Go.

PointBase Micro Edition

Motor de base de datos relacional, con menos de 45KB de footprint. PointBase provee una utilería llamada PointBase UniSync, la cual permite sincronizar bidireccionalmente los datos entre la base de datos en un dispositvo móvil y una base de datos corporativa, incluendo PointBase Embedded, Oracle y SQL Server. Soporta encriptación de datos y manejo de base de datos en memoria.

2. Desarrollo de la Aplicación Prototipo

2.1. Descripción de la Aplicación

Se desarrollará una aplicación para dispositivos móviles consistente en fichas médicas, para que un galeno pueda llevar un registro de los datos personales y características del paciente, el motivo de la consulta, antecedentes personales y familiares relacionados con el padecimiento, así como las distintas etapas del tratamiento. Para el modelado de la aplicación se empleará un enfoque orientado a objetos, haciendo uso del lenguaje de modelado unificado UML.

2.1.1. Objetivos de la Aplicación

- Mejorar la atención a los pacientes
- Llevar un control consistente y actualizado de los expedientes de los pacientes
- Obtener información de manera oportuna
- Disminuir errores en el registro de datos

2.2. Requerimientos

2.2.1. Funciones de la Aplicación

Las funciones principales de la aplicación se describen en la tabla 2.2, y se clasifican de acuerdo a las categorías mostradas en la tabla 2.1.

CATEGORIA	DESCRIPCION
Externa	Estas funciones son las que al ser ejecutadas, pueden ser observadas por el usuario.
Interna	La ejecución de esta categoría de funciones puede que no sea visible para el usuario.

Tabla 2.1 Categorías de las funciones

REFERENCIA	FUNCION	CATEGORIA
R-001	Asignar número de expediente	Interna
R-002	Registrar datos personales del paciente	Externa
R-003	Registrar motivo de la consulta	Externa
R-004	Registrar datos del resultado del examen físico del paciente	Externa
R-005	Almacenar diagnóstico de la enfermedad	Externa

R-006	Almacenar tratamiento	Externa
R-007	Recuperar citas próximas	Interna
R-008	Almacenar evolución del paciente	Externa
R-009	Registrar datos del resultado de exámenes de laboratorio	Externa
R-010	Recuperar y mostrar datos del paciente	Externa
R-011	Mostrar listado de citas	Externa
R-012	Mostrar listado de pacientes	Externa
R-013	Sincronizar datos entre dispositivo móvil y computadora de escritorio	Interna

Tabla 2.2 Funciones de la aplicación

2.2.2. Atributos de la Aplicación

En la Tabla 2.3 se describen los atributos de la aplicación

ATRIBUTO	DETALLES Y RESTRICCIONES
Tiempo de respuesta	(Restricción de frontera) Una vez ingresada la información necesaria para realizar las acciones solicitadas, la aplicación deberá proporcionar la salida correspondiente en un periodo no mayor de cinco segundos.
Interfaz de usuario	(Detalle)La aplicación utilizará un entorno gráfico para interactuar con el usuario.
Facilidad de aprendizaje	(Detalle) La aplicación deberá ser fácilmente entendible, de modo que el tiempo necesario para que los galenos aprendan a utilizarla sea mínimo.
Tolerancia a fallos	(Restricción de frontera) La aplicación tratará de reponerse automáticamente cuando ocurran errores generados por su mal uso, reportando al usuario la causa aparente del error.
Confiabilidad	(Restricción de frontera) La información proporcionada por la aplicación debe ser precisa, verificable y exacta.
Capacidad de mantenimiento	(Detalle) El mantenimiento de la aplicación así como futuras extensiones de funciones debe realizarse con facilidad.
Plataforma del dispositivo móvil	(Restricción de frontera) Debe soportar el perfil MIDP 2.0, con cualquier sistema operativo.

Tabla 2.3 Atributos de la aplicación

2.2.3. Identificación de Casos de Uso

Identificación de actores

Los actores que interactúan con la aplicación son:

- Paciente
- Médico

- Aplicación en computadora de escritorio (ACE)

Paciente y casos de uso con los que se relaciona

- Crear expediente
- Iniciar consulta
- Realizar examen físico
- Diagnosticar enfermedad
- Determinar evolución de la enfermedad

Médico y casos de uso con los que se relaciona

- Crear expediente
- Iniciar consulta
- Realizar examen físico
- Diagnosticar enfermedad
- Prescribir tratamiento
- Asignar cita
- Determinar evolución de la enfermedad
- Obtener listado
- Sincronizar datos

ACE y casos de uso con los que se relaciona

- Sincronizar datos

2.2.4. Descripción de Procesos

Crear expediente

Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Si es primera vez que el médico atiende al paciente, entonces solicita los siguientes datos: apellidos, nombres, sexo, fecha de nacimiento, estado

civil, dirección, número de teléfono fijo, número de teléfono móvil, dirección de correo electrónico, tipo y número de documento de identidad personal, ocupación, nombre del padre, nombre de la madre, nombre del cónyuge, nombre del responsable, dirección del responsable, teléfono del responsable, nombre de la persona que proporciona los datos, parentesco de la persona que proporciona los datos, tipo y número de documento de identidad de la persona que proporciona los datos.

Iniciar consulta

Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente, entonces el médico solicita los siguientes datos: apellidos, nombres y/o número de documento de identidad personal. Este caso de uso presupone que el expediente del paciente ya fue creado. El médico indaga sobre el motivo de la consulta.

Realizar examen físico

Este caso de uso se presenta cuando el médico mide el peso, estatura y las constantes vitales del paciente: presión arterial, pulso, frecuencia respiratoria y temperatura.

Diagnosticar enfermedad

Este caso de uso sucede cuando el médico indaga sobre los síntomas que presenta el paciente, y los antecedentes personales y familiares del mismo. Y formula su diagnóstico.

Prescribir tratamiento

Este caso de uso se da cuando el médico prescribe los medicamentos y su dosificación. E indica la realización de exámenes de laboratorio.

Asignar cita

Este caso de uso sucede cuando el médico le asigna al paciente la fecha y hora de la próxima cita.

Determinar evolución de la enfermedad

Este caso de uso inicia cuando el paciente se ha presentado al consultorio del médico o cuando el médico ha realizado una visita al paciente, con el propósito de darle seguimiento

a una enfermedad previamente consultada, entonces el médico indaga sobre los resultados del tratamiento.

Obtener listado

Este caso de uso se presenta cuando el médico requiere un listado de las citas para una fecha determinada, o un listado de sus pacientes.

Sincronizar datos

Este caso de uso se da cuando el médico quiere sincronizar los datos entre el dispositivo móvil y la computadora de escritorio.

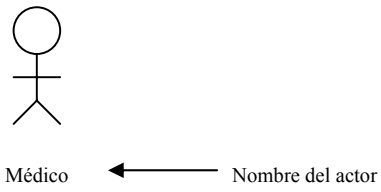
DETALLE DE LA TECNICA

DIAGRAMA DE CASOS DE USO

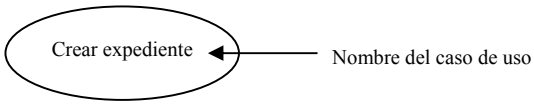
Un caso de uso representa un proceso común e importante. Los casos de uso constituyen una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización software. Un diagrama de Casos de Uso muestra las distintas operaciones que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios u otras aplicaciones).

SIMBOLOGIA

Actor (usuario)



Casos de uso



CASOS DE USO NARRADOS

Aquí se hace una descripción más detallada de los procesos o casos de uso, indicando la interacción a través de acciones de los actores y respuestas del sistema o aplicación.

2.2.5. Diagrama de Casos de Uso

En la figura 2.1 se presenta el diagrama de casos de uso de la aplicación de fichas médicas.

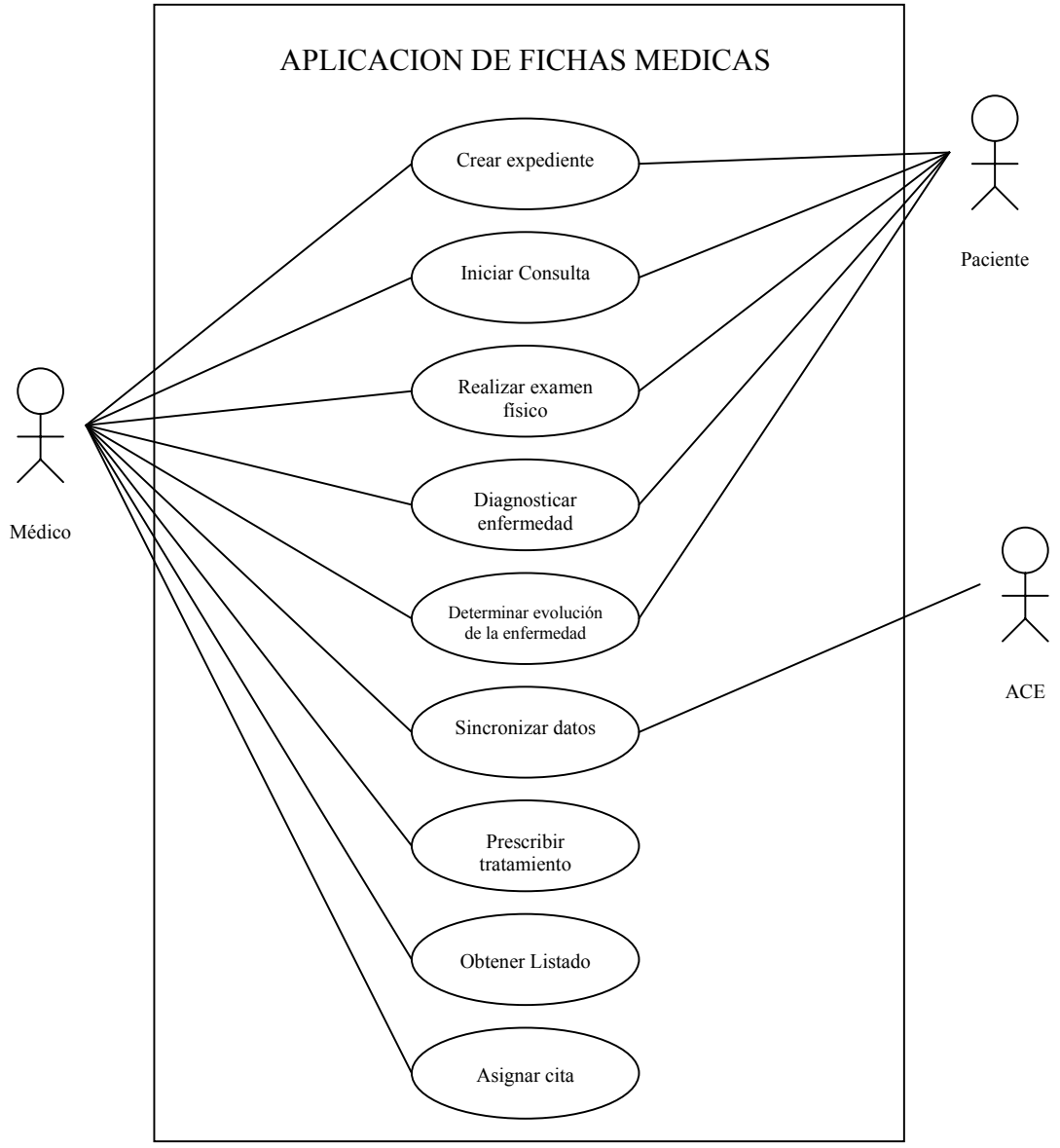


Figura 2.1 Diagrama de casos de uso

2.2.6. Casos de Uso Narrados

CASO DE USO	Crear expediente
ACTORES	Paciente, Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Y es primera vez que el médico atiende al paciente.

CASO DE USO	Iniciar consulta
ACTORES	Paciente, Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Y el médico indaga sobre el motivo de la consulta.

CASO DE USO	Realizar examen físico
ACTORES	Paciente, Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso se presenta cuando el médico mide el peso, estatura y las constantes vitales del paciente: presión arterial, pulso, frecuencia respiratoria y temperatura.

CASO DE USO	Diagnosticar enfermedad
ACTORES	Paciente, Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso sucede cuando el médico indaga sobre los síntomas que presenta el paciente, y los antecedentes personales y familiares del mismo. Y formula su diagnóstico.

CASO DE USO	Prescribir tratamiento
ACTORES	Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso se da cuando el médico prescribe los medicamentos y su dosificación.

CASO DE USO	Asignar cita
ACTORES	Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso sucede cuando el médico le asigna al paciente la fecha y hora de la próxima cita.

CASO DE USO	Determinar evolución de la enfermedad
ACTORES	Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso inicia cuando el paciente se ha presentado al consultorio del médico o cuando el médico ha realizado una visita al paciente, con el propósito de darle seguimiento a una enfermedad previamente consultada, entonces el médico indaga sobre los resultados del tratamiento.

CASO DE USO	Obtener listado
ACTORES	Médico
TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso se presenta cuando el médico requiere un listado de las citas para una fecha determinada, o un listado de sus pacientes.

CASO DE USO	Sincronizar datos
ACTORES	Médico, ACE

TIPO	Primario y esencial
DESCRIPCIÓN	Este caso de uso se da cuando el médico quiere sincronizar los datos entre el dispositivo móvil y la computadora de escritorio.

Casos de uso extendidos

CASO DE USO	Crear expediente
ACTORES	Paciente, Médico
PROPOSITO	Crear un expediente para un nuevo paciente.
VISION	Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Y es primera vez que el médico atiende al paciente.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-001, R-002

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El paciente se presenta ante el médico. 2. El médico indica a la aplicación que ingresará los datos de un nuevo paciente. 4. El médico ingresa los siguientes datos: apellidos, nombres, sexo, fecha de nacimiento, estado civil, dirección, número de teléfono fijo, número de teléfono móvil, dirección de correo electrónico, tipo y número de documento de identidad personal, ocupación, nombre del padre, nombre de la madre, nombre del cónyuge, nombre del responsable, dirección del responsable, teléfono del responsable, nombre de la persona que proporciona los datos, parentesco de la persona que proporciona los datos, tipo y número de documento de identidad de la persona que proporciona los datos.	3. La aplicación inicializa los componentes involucrados en el registro del nuevo paciente. 5. La aplicación almacena los datos del nuevo paciente.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Iniciar consulta
ACTORES	Paciente, Médico
PROPOSITO	Registrar una nueva consulta médica.
VISION	Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Y el médico indaga sobre el motivo de la consulta.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-003

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El paciente se presenta ante el médico. 2. El médico indica a la aplicación que ingresará los datos de una consulta. 4. El médico ingresa el motivo de la consulta	3. La aplicación inicializa los componentes involucrados en el registro de una consulta. 5. La aplicación almacena el motivo de la consulta.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Realizar examen físico
ACTORES	Paciente, Médico
PROPOSITO	Registrar resultados del examen físico.
VISION	Este caso de uso se presenta cuando el médico mide el peso, estatura y las constantes vitales del paciente: presión arterial, pulso, frecuencia respiratoria y temperatura.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-004

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico realiza un chequeo de las constantes	

vitales del paciente: presión arterial, pulso, frecuencia respiratoria y temperatura; y mide su peso y estatura. 2. El médico ingresa los resultados del examen físico.	4. La aplicación almacena los resultados del examen físico.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Diagnosticar enfermedad
ACTORES	Paciente, Médico
PROPOSITO	Registrar el resultado del diagnóstico de la enfermedad.
VISION	Este caso de uso sucede cuando el médico indaga sobre los síntomas que presenta el paciente, y los antecedentes personales y familiares del mismo. Y formula su diagnóstico.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-005
CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico pide al paciente que describa los síntomas y en base a ellos pregunta sobre los antecedentes personales y familiares. 2. El paciente describe los síntomas y relata sobre antecedentes personales y familiares relacionados con la enfermedad. 3. El médico ingresa datos sobre los antecedentes personales y familiares. 11. El médico realiza el diagnóstico de la enfermedad y lo ingresa en la aplicación	5. La aplicación almacena los antecedentes personales y familiares. 6. La aplicación almacena los resultados del diagnóstico.
CURSOS ALTERNATIVOS	
Paso 3. El médico realiza diagnóstico de la enfermedad y refiere al paciente a un hospital, e ingresa los datos respectivos en la aplicación.	La aplicación almacena los resultados del diagnóstico y los datos del hospital.

CASO DE USO	Prescribir tratamiento
ACTORES	Médico
PROPOSITO	Registrar el tratamiento al que se someterá el paciente.
VISION	Este caso de uso se da cuando el médico prescribe los medicamentos y su dosificación.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-006, R-009
CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico prescribe los medicamentos y la dosificación respectiva, e ingresa esos datos en la aplicación. 3. El médico indica al paciente la realización de exámenes de laboratorio.	2. La aplicación almacena los datos del tratamiento prescrito.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Asignar cita
ACTORES	Médico
PROPOSITO	Asignar fecha y hora de próxima cita de un paciente.
VISION	Este caso de uso sucede cuando el médico le asigna al paciente la fecha y hora de la próxima cita.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-007
CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico indica a la aplicación que desea reservar fecha y hora para una cita. 3. El médico ingresa fecha y hora de la nueva cita.	2. La aplicación muestra una lista de citas programadas cuya fecha sea mayor a la actual. 4. La aplicación almacena la fecha y hora de la

	cita.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Determinar evolución de la enfermedad
ACTORES	Médico, Paciente
PROPOSITO	Registrar los resultados que ha tenido el paciente después del tratamiento.
VISION	Este caso de uso inicia cuando el paciente se ha presentado al consultorio del médico o cuando el médico ha realizado una visita al paciente, con el propósito de darle seguimiento a una enfermedad previamente consultada, entonces el médico indaga sobre los resultados del tratamiento.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-008, R-009, R-010
CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El paciente presenta el resultado de los exámenes de laboratorio. 2. El médico pregunta al paciente respecto de los resultados que ha observado después del tratamiento. 3. El paciente describe los resultados. 6. El médico ingresa los datos de la evolución de la enfermedad y los resultados de los exámenes de laboratorio.	5. La aplicación almacena los datos de la evolución de la enfermedad y resultado de exámenes.
CURSOS ALTERNATIVOS	
Paso 4. El médico ingresa los datos de la evolución de la enfermedad, los resultados de los exámenes de laboratorio y refiere al paciente a un hospital.	La aplicación almacena los datos de la evolución de la enfermedad, resultado de exámenes y datos del hospital.

CASO DE USO	Obtener listado
ACTORES	Médico

PROPOSITO	Obtener un listado: de citas o de pacientes que atiende.
VISION	Este caso de uso se presenta cuando el médico requiere un listado de las citas para una fecha determinada, o un listado de sus pacientes.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-011, R-012

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico indica a la aplicación que desea obtener un listado. 3. El médico selecciona el tipo de listado. 7. El médico ingresa los parámetros respectivos.	2. La aplicación muestra las opciones: listado de citas, y listado de pacientes. 4. La aplicación muestra las opciones para el tipo de listado elegido. 6. La aplicación muestra el listado.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Sincronizar datos
ACTORES	Médico, ACE
PROPOSITO	Sincronizar la base de datos de la computadora de escritorio y la base de datos del dispositivo móvil.
VISION	Este caso de uso se da cuando el médico quiere sincronizar los datos entre el dispositivo móvil y la computadora de escritorio.
TIPO	Primario y esencial
REFERENCIAS	Funciones: R-013

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico inicia el servicio de sincronización en la computadora de escritorio. 2. La aplicación en la computadora de escritorio (ACE) "escucha" las peticiones de sincronización provenientes del dispositivo móvil.	

3. El médico inicia la sincronización en el dispositivo móvil. 5. ACE realiza la sincronización.	4. La aplicación hace la petición de sincronización. 6. La aplicación indica que el proceso de sincronización ha finalizado.
CURSOS ALTERNATIVOS	
---	---

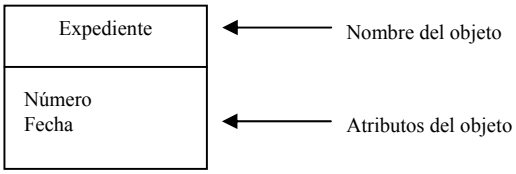
DETALLE DE LA TECNICA

MODELO CONCEPTUAL

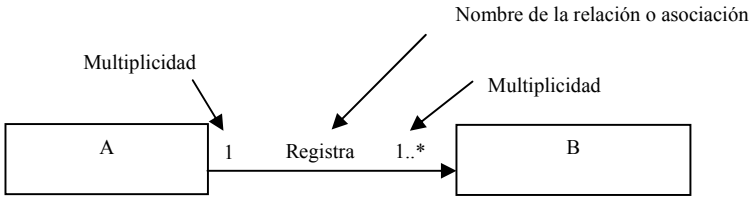
El modelo concetual representa de manera gráfica la relación entre los diferentes conceptos u objetos encontrados a partir de la descripción de los casos de uso.

SIMBOLOGIA

Objeto



Asociaciones



Un objeto A registra 1 o más objetos B

Un objeto B es registrado por 1 objeto A

2.2.7. Modelo Conceptual

En la figura 2.2 se representa el modelo conceptual.

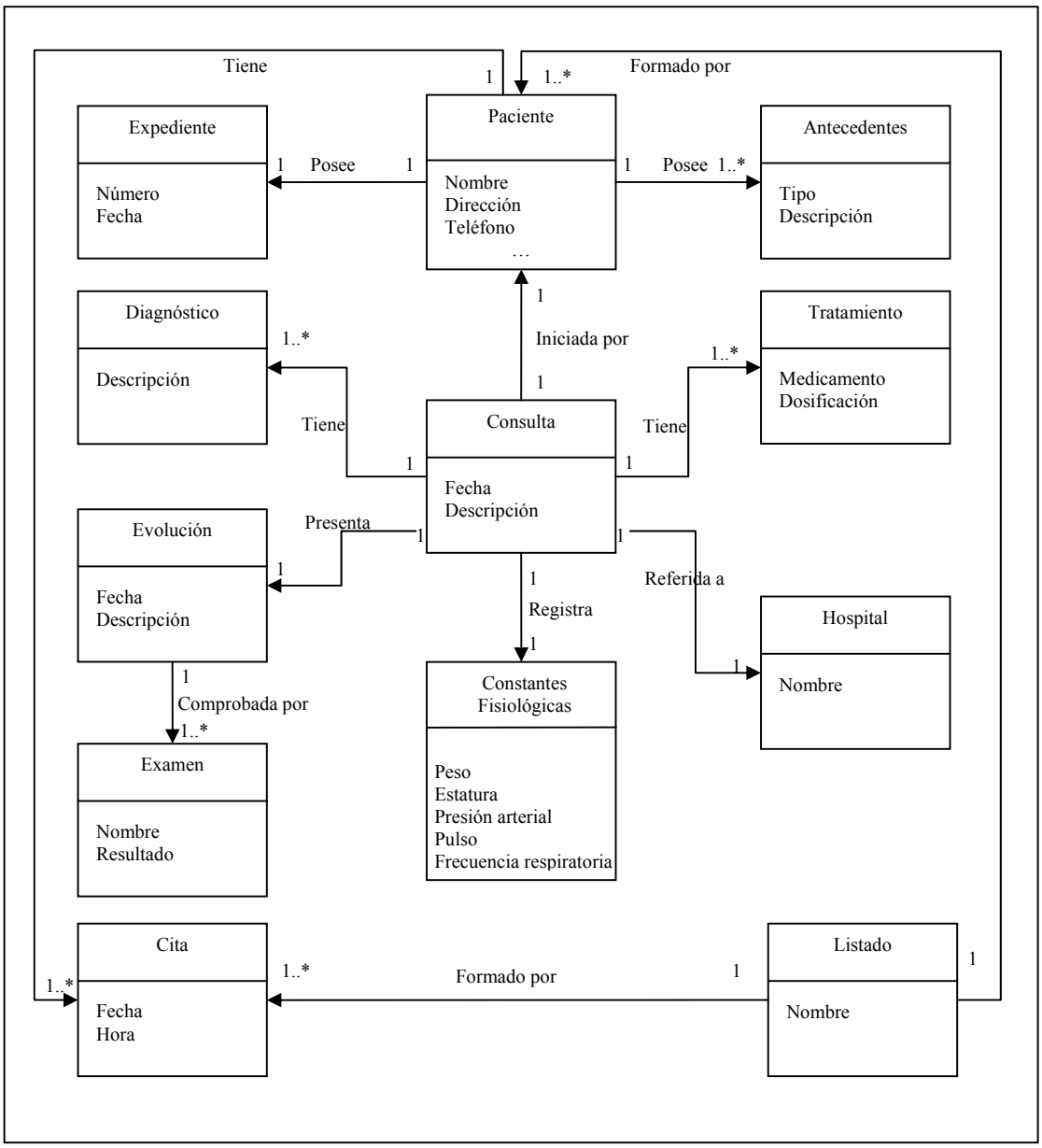


Figura 2.2 Modelo conceptual

2.2.8. Glosario Conceptual

TERMINO	DESCRIPCION
Paciente	Persona que adolece una enfermedad y que se presenta al médico para que éste lo ausculte.
Expediente	Información relativa a un paciente, que incluye sus datos personales y la fecha de la primera consulta.
Antecedentes	Contiene los antecedentes personales del paciente y los antecedentes familiares relacionados con la enfermedad que padece.
Diagnóstico	Consiste en el dictamen emitido por el médico respecto de la enfermedad que padece el paciente.
Tratamiento	Formado por la prescripción de medicamentos con su respectiva dosificación.
Consulta	Contiene una descripción del motivo por el cual el paciente se presenta ante el médico.
Evolución	Está formada por las diversas etapas que atraviesa el paciente, después de seguir el tratamiento ordenado por el médico.
Examen	Resultado de los análisis de laboratorio ordenados por el médico.
Constantes Fisiológicas	Contiene el peso, estatura y constantes vitales (presión arterial, pulso, frecuencia respiratoria y temperatura) del paciente.
Hospital	Centro de atención de salud al cual es referido un paciente para un tratamiento exhaustivo.
Cita	Formada por la fecha y hora de la próxima entrevista con el paciente.
Listado	Puede estar formado por una lista de citas o por una lista de pacientes.

Tabla 2.4 Glosario conceptual

2.3. Análisis

DETALLE DE LA TECNICA

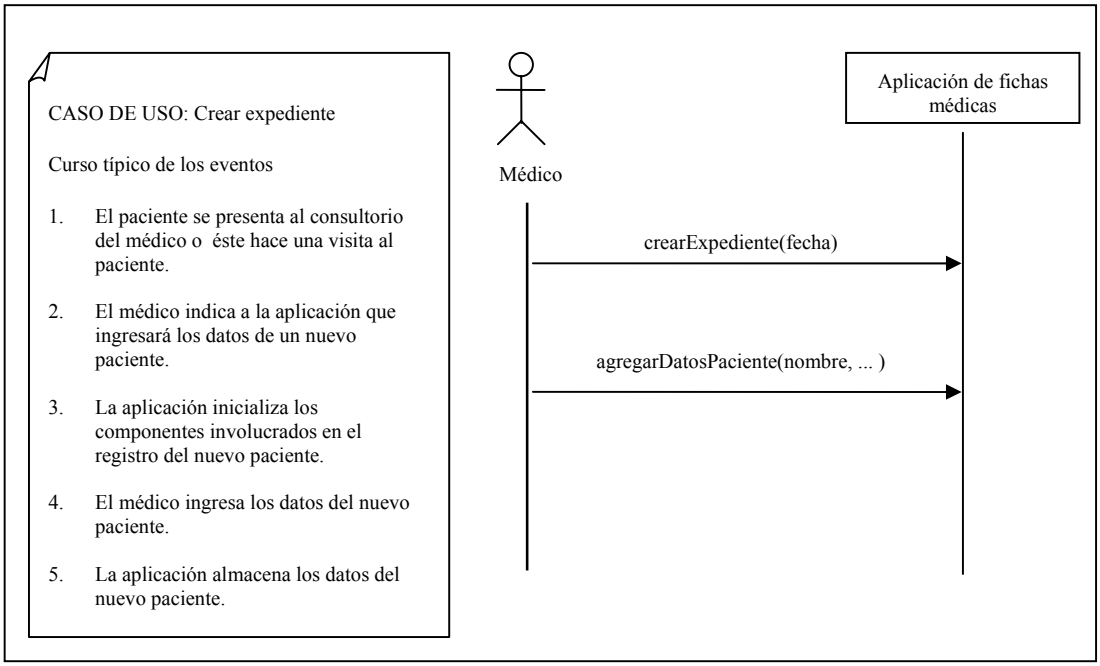
DIAGRAMAS DE SECUENCIA

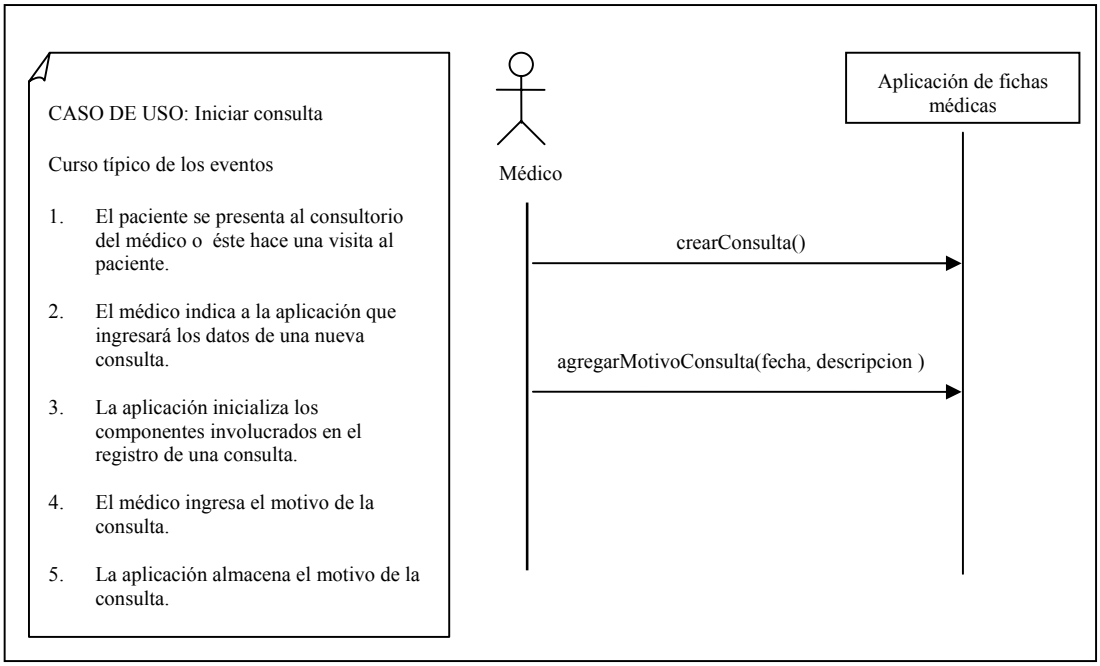
Muestran de forma gráfica los eventos que se producen en la interacción entre los actores y el sistema o aplicación, o entre objetos.

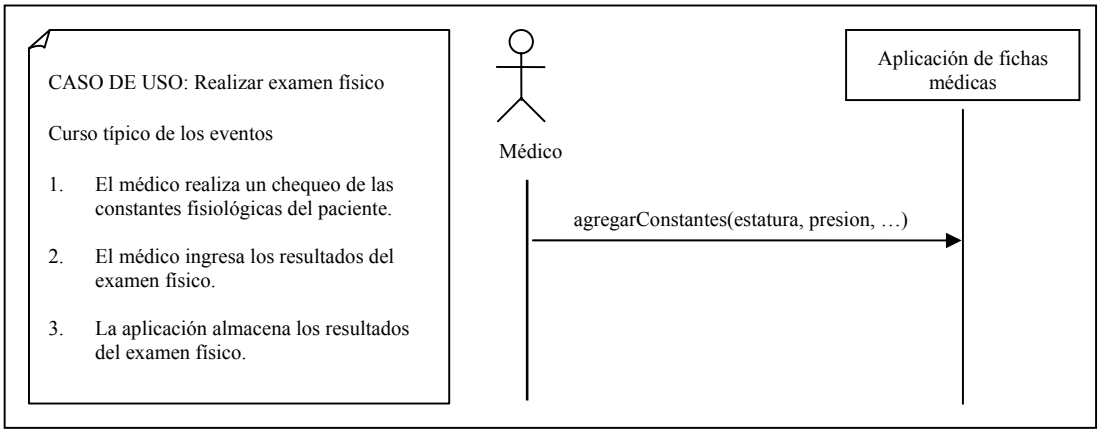
Los diagramas de secuencia muestran el intercambio de mensajes y ponen especial énfasis en el orden y el momento en que se envían los mensajes a los objetos.

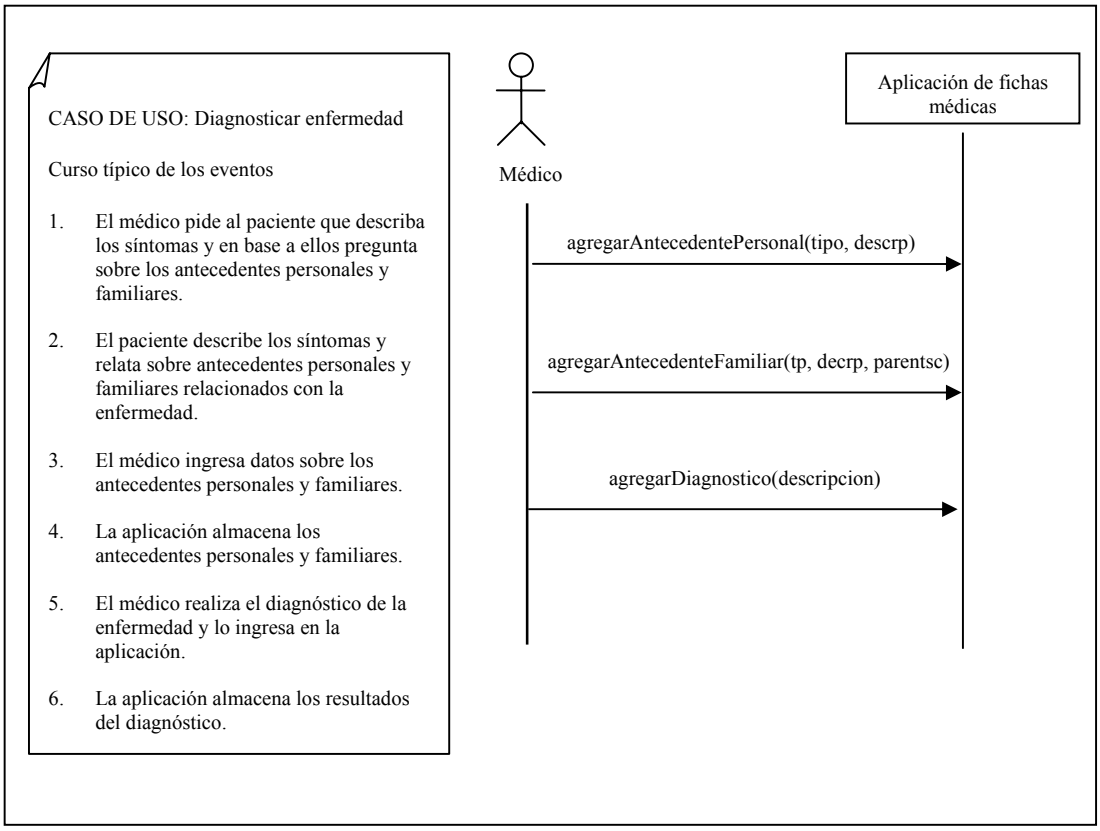
En los diagramas de secuencia, los objetos están representados por rectángulos, el eje de tiempo es vertical, incrementándose hacia abajo, de forma que los mensajes son enviados de un objeto a otro en forma de flechas con los nombres de la operación y los parámetros.

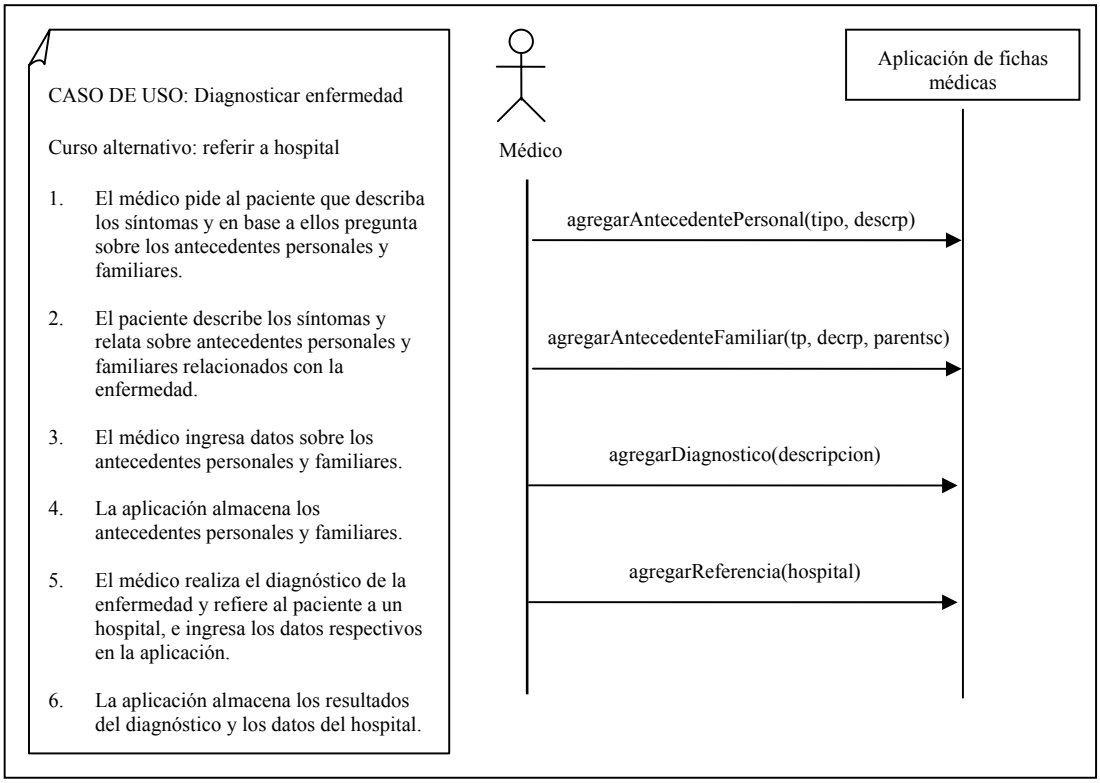
2.3.1. Diagramas de Secuencia

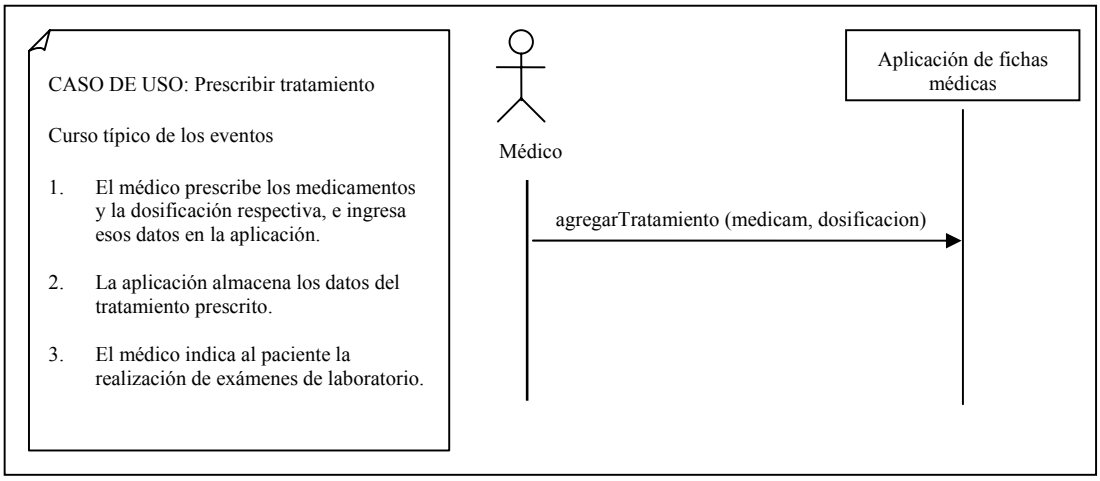


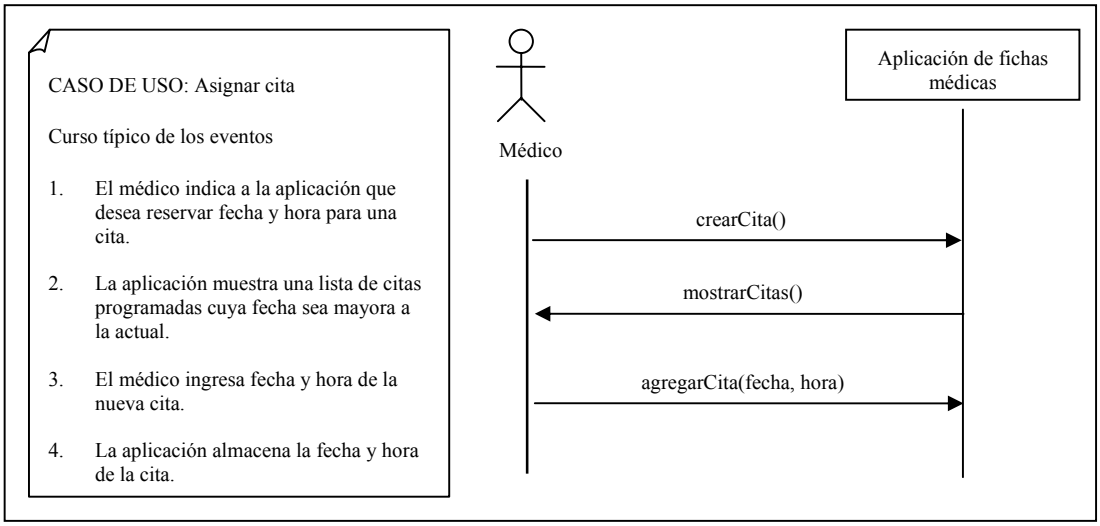


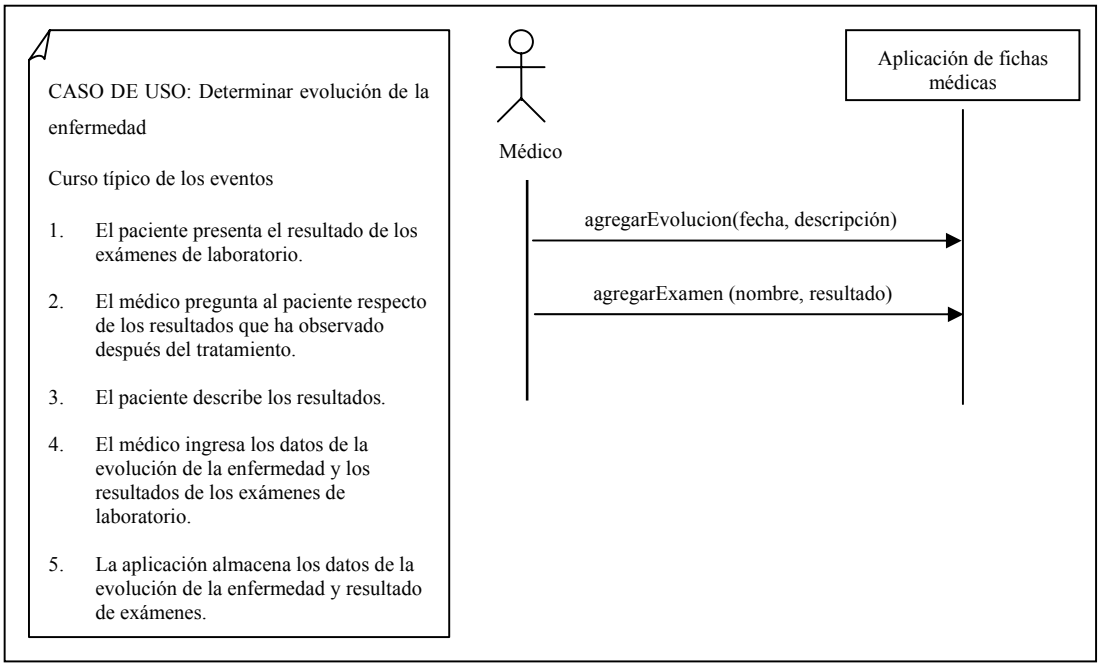

CASO DE USO: Realizar examen físico
Curso típico de los eventos
1. El médico realiza un chequeo de las constantes fisiológicas del paciente.
2. El médico ingresa los resultados del examen físico.
3. La aplicación almacena los resultados del examen físico.
Médico
Aplicación de fichas médicas
agregarConstantes(estatura, presion, …)


CASO DE USO: Diagnosticar enfermedad
Curso típico de los eventos
1. El médico pide al paciente que describa los síntomas y en base a ellos pregunta sobre los antecedentes personales y familiares.
2. El paciente describe los síntomas y relata sobre antecedentes personales y familiares relacionados con la enfermedad.
3. El médico ingresa datos sobre los antecedentes personales y familiares.
4. La aplicación almacena los antecedentes personales y familiares.
5. El médico realiza el diagnóstico de la enfermedad y lo ingresa en la aplicación.
6. La aplicación almacena los resultados del diagnóstico.
Médico
Aplicación de fichas médicas
agregarAntecedentePersonal(tipo, descrp)
agregarAntecedenteFamiliar(tp, decrp, parentsc)
agregarDiagnostico(descripcion)


CASO DE USO: Diagnosticar enfermedad
Curso alternativo: referir a hospital
1. El médico pide al paciente que describa los síntomas y en base a ellos pregunta sobre los antecedentes personales y familiares.
2. El paciente describe los síntomas y relata sobre antecedentes personales y familiares relacionados con la enfermedad.
3. El médico ingresa datos sobre los antecedentes personales y familiares.
4. La aplicación almacena los antecedentes personales y familiares.
5. El médico realiza el diagnóstico de la enfermedad y refiere al paciente a un hospital, e ingresa los datos respectivos en la aplicación.
6. La aplicación almacena los resultados del diagnóstico y los datos del hospital.
Médico
Aplicación de fichas médicas
agregarAntecedentePersonal(tipo, descrp)
agregarAntecedenteFamiliar(tp, decrp, parentsc)
agregarDiagnostico(descripcion)
agregarReferencia(hospital)


CASO DE USO: Prescribir tratamiento
Curso típico de los eventos
1. El médico prescribe los medicamentos y la dosificación respectiva, e ingresa esos datos en la aplicación.
2. La aplicación almacena los datos del tratamiento prescrito.
3. El médico indica al paciente la realización de exámenes de laboratorio.
Médico
Aplicación de fichas médicas
agregarTratamiento (medicam, dosificacion)

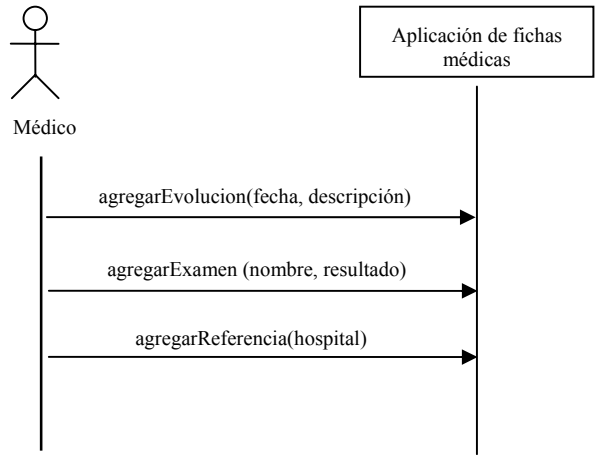

CASO DE USO: Asignar cita
Curso típico de los eventos
1. El médico indica a la aplicación que desea reservar fecha y hora para una cita.
2. La aplicación muestra una lista de citas programadas cuya fecha sea mayora a la actual.
3. El médico ingresa fecha y hora de la nueva cita.
4. La aplicación almacena la fecha y hora de la cita.
Médico
Aplicación de fichas médicas
crearCita()
mostrarCitas()
agregarCita(fecha, hora)


CASO DE USO: Determinar evolución de la enfermedad
Curso típico de los eventos
1. El paciente presenta el resultado de los exámenes de laboratorio.
2. El médico pregunta al paciente respecto de los resultados que ha observado después del tratamiento.
3. El paciente describe los resultados.
4. El médico ingresa los datos de la evolución de la enfermedad y los resultados de los exámenes de laboratorio.
5. La aplicación almacena los datos de la evolución de la enfermedad y resultado de exámenes.
Médico
Aplicación de fichas médicas
agregarEvolucion(fecha, descripción)
agregarExamen (nombre, resultado)

CASO DE USO: Determinar evolución de la enfermedad

Curso alternativo: referir a hospital

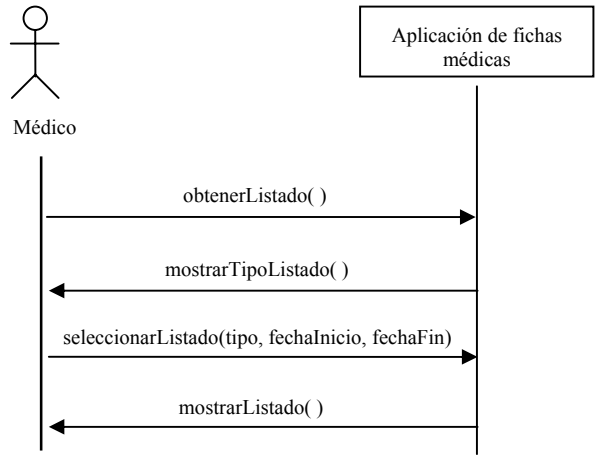
1. El paciente presenta el resultado de los exámenes de laboratorio.
2. El médico pregunta al paciente respecto de los resultados que ha observado después del tratamiento.
3. El paciente describe los resultados.
4. El médico ingresa los datos de la evolución de la enfermedad, los resultados de los exámenes de laboratorio y refiere al paciente a un hospital.
5. La aplicación almacena los datos de la evolución de la enfermedad, resultado de exámenes y datos del hospital.



CASO DE USO: Obtener listado

Curso típico de los eventos

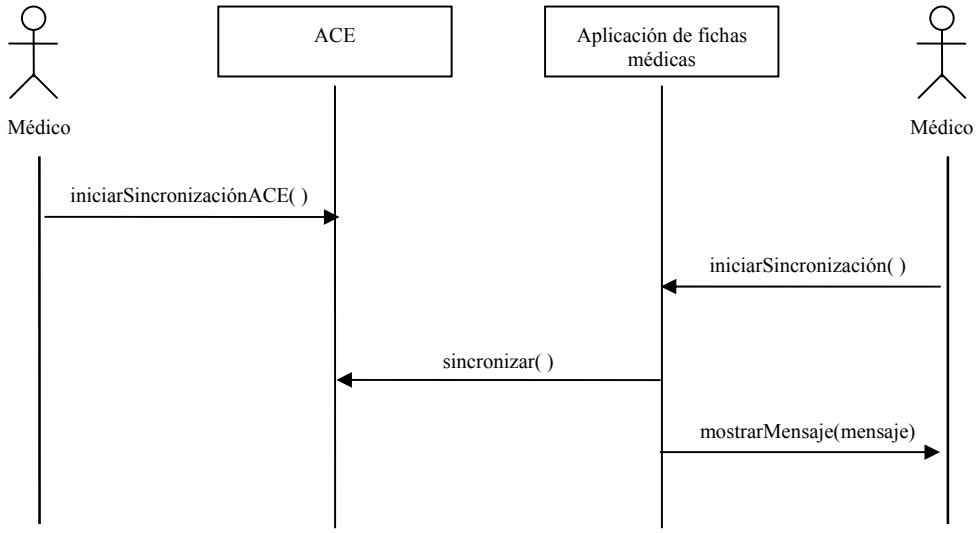
1. El médico indica a la aplicación que desea obtener un listado.
2. La aplicación muestra las opciones: listados de citas, y listado de pacientes.
3. El médico selecciona el tipo de listado.
4. La aplicación muestra las opciones para el tipo de listado elegido.
5. El médico ingresa los parámetros respectivos.
6. La aplicación muestra el listado.



CASO DE USO: Sincronizar datos

Curso típico de los eventos

1. El médico inicia el servicio de sincronización en la computadora de escritorio.
2. La aplicación en la computadora de escritorio (ACE) “escucha” las peticiones de sincronización provenientes del dispositivo móvil.
3. El médico inicia la sincronización en el dispositivo móvil.
4. La aplicación hace la petición de sincronización.
5. ACE realiza la sincronización.
6. La aplicación indica que el proceso de sincronización ha finalizado.



DETALLE DE LA TECNICA

CONTRATOS DE OPERACION

Los contratos de operación contribuyen a definir el sistema o aplicación, ya que, describen el efecto que tienen las operaciones sobre el sistema.

2.3.2. Contratos de Operación

Lista de contratos de operación

- crearExpediente(fecha)
- agregarDatosPaciente(nombre,…)
- crearConsulta()
- agregarMotivoConsulta(fecha, descripción)
- agregarConstantes(estatura, presion, …)
- agregarAntecedentePersonal(tipo, descripción)
- agregarAntecedenteFamiliar(tipo, descripción, parentesco)
- agregarDiagnostico(descripción)
- agregarReferencia(hospital)
- agregarTratamiento(medicamento, dosificación)
- crearCita()
- mostrarCitas()
- agregarCita(fecha, hora)
- agregarEvolución(fecha, descripción)
- agregarExamen(nombre, resultado)
- obtenerListado()
- mostrarTipoListado()
- seleccionarListado(tipo, fechaInicio, fechaFin)
- mostrarListado()
- iniciarSincronizacionACE()
- iniciarSincronizacion()
- sincronizar()

- mostrarMensaje(mensaje)

NOMBRE	crearExpediente(fecha)
RESPONSABILIDADES	Iniciar los elementos necesarios para el registro de un nuevo paciente
REFERENCIAS	Casos de uso: Crear expediente
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	

POST-CONDICIONES	
▪ Se ha creado una instancia de expediente ▪ Se ha asignado un correlativo al expediente ▪ Se ha modificado el atributo fecha del expediente	

NOMBRE	agregarDatosPaciente(apellidos, nombres, sexo, fechaNac, estadoCivil, direccion, telefono, movil, email, tipoDoc, numeroDoc, ocupacion, padre, madre, conyuge, responsable, direccionResp, telefonoResp, nombrePDatos, parentescoPDatos, tipoDocPDatos, numeroDocPDatos)
RESPONSABILIDADES	Registrar los datos de un nuevo paciente
REFERENCIAS	Casos de uso: Crear expediente
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente	
POST-CONDICIONES	

- Se ha creado una instancia de paciente
- Se han modificado los atributos del paciente: apellidos, nombres, sexo, fechaNac, estadoCivil, direccion, telefono, movil, email, tipoDoc, numeroDoc, ocupacion, padre, madre, conyuge, responsable, direccionResp, telefonoResp, nombrePDatos, parentescoPDatos, tipoDocPDatos, numeroDocPDatos.
- El paciente se ha asociado con el expediente

NOMBRE	crearConsulta()
RESPONSABILIDADES	Iniciar los elementos necesarios para el registro de una nueva consulta
REFERENCIAS	Casos de uso: Iniciar consulta
NOTAS	---
EXCEPCIONES	---
SALIDA	---

PRE-CONDICIONES

- Se ha creado el expediente para el paciente
- Se han registrado los datos del paciente

POST-CONDICIONES

- Se ha creado una instancia de consulta
- La consulta se ha asociado con el paciente

NOMBRE	agregarMotivoConsulta(fecha, descripción)
RESPONSABILIDADES	Registrar el motivo de la consulta
REFERENCIAS	Casos de uso: Iniciar consulta
NOTAS	---
EXCEPCIONES	---
SALIDA	---

PRE-CONDICIONES

- Se ha creado el expediente para el paciente

▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta
POST-CONDICIONES
▪ Se han modificado los atributos fecha y descripción de consulta

NOMBRE	agregarConstantes(estatura, peso, presionArterial, pulso, frecuenciaRespiratoria)
RESPONSABILIDADES	Registrar el resultado del examen físico del paciente
REFERENCIAS	Casos de uso: Realizar examen físico
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de constantesFisiologicas ▪ Se han modificado los atributos de constantesFisiologicas: estatura, peso, presionArterial, pulso, recuenciaRespiratoria ▪ Las constantesFisiologicas se han asociado con consulta	

NOMBRE	agregarAntecedentesPersonales(tipo, descripción)
RESPONSABILIDADES	Registrar antecedentes personales del paciente
REFERENCIAS	Casos de uso: Diagnosticar enfermedad
NOTAS	---
EXCEPCIONES	---

SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de antecedentesPersonales ▪ Se han modificado los atributos tipo y descripción de antecedentesPersonales ▪ Los antecedentesPersonales se han asociado con paciente	

NOMBRE	agregarAntecedenteFamiliar(tipo, descripción, parentesco)
RESPONSABILIDADES	Registrar antecedentes familiares del paciente, relacionados con el padecimiento
REFERENCIAS	Casos de uso: Diagnosticar enfermedad
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de antecedentesFamiliares ▪ Se han modificado los atributos tipo, descripción y parentesco de antecedentesFamiliares ▪ Los antecedentesFamiliares se han asociado con paciente	

NOMBRE	agregarDiagnostico(descripción)
RESPONSABILIDADES	Registrar el diagnóstico de la enfermedad

REFERENCIAS	Casos de uso: Diagnosticar enfermedad
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de diagnóstico ▪ Se ha modificado el atributo descripción de diagnóstico ▪ El diagnóstico se ha asociado con consulta	

NOMBRE	agregarReferencia(hospital)
RESPONSABILIDADES	Registrar el hospital al cual se refiere un paciente
REFERENCIAS	Casos de uso: Diagnosticar enfermedad
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de referencia ▪ Se ha asociado la referencia con hospital ▪ La referencia se ha asociado con consulta	

NOMBRE	agregarTratamiento(medicamento, dosificación)
RESPONSABILIDADES	Registrar el tratamiento prescrito al paciente
REFERENCIAS	Casos de uso: Prescribir tratamiento
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de tratamiento ▪ Se han modificado los atributos medicamento y dosificación de tratamiento ▪ Se ha asociado el tratamiento con consulta	

NOMBRE	crearCita()
RESPONSABILIDADES	Iniciar los elementos necesarios para el registro de una nueva cita
REFERENCIAS	Casos de uso: Asignar cita
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de cita	

NOMBRE	mostrarCitas()
RESPONSABILIDADES	Mostrar las citas programadas cuya fecha sea mayor a la actual
REFERENCIAS	Casos de uso: Asignar cita
NOTAS	---
EXCEPCIONES	---
SALIDA	Se muestra una lista de las citas programadas
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	

NOMBRE	agregarCita(fecha, hora)
RESPONSABILIDADES	Registrar una nueva cita
REFERENCIAS	Casos de uso: Asignar cita
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos ▪ Se ha creado una instancia de cita	
POST-CONDICIONES	
▪ Se han modificado los atributos fecha y hora de cita	

▪ Se ha asociado cita con paciente

NOMBRE	agregarEvolución(fecha, descripción)
RESPONSABILIDADES	Registrar la evolución de la enfermedad
REFERENCIAS	Casos de uso: Determinar evolución de la enfermedad
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	
POST-CONDICIONES	
▪ Se ha creado una instancia de evolucion ▪ Se han modificado los atributos fecha y descripcion de evolucion ▪ Se ha asociado evolución con consulta	

NOMBRE	agregarExamen(nombre, resultado)
RESPONSABILIDADES	Registrar el resultado de los exámenes de laboratorio
REFERENCIAS	Casos de uso: Determinar evolución de la enfermedad
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado el expediente para el paciente ▪ Se han registrado los datos del paciente ▪ Se ha creado una nueva consulta y modificado sus atributos	

▪ Se ha registrado la evolución de la enfermedad
POST-CONDICIONES
▪ Se ha creado una instancia de examen ▪ Se han modificado los atributos nombre y resultado de examen ▪ Se ha asociado examen con evolución

NOMBRE	obtenerListado()
RESPONSABILIDADES	Iniciar los elementos necesarios para generar un listado
REFERENCIAS	Casos de uso: Obtener listado
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	

POST-CONDICIONES	
▪ Se ha creado una instancia de listado	

NOMBRE	mostrarTipoListado()
RESPONSABILIDADES	Mostrar los tipos de listado disponibles
REFERENCIAS	Casos de uso: Obtener listado
NOTAS	---
EXCEPCIONES	---
SALIDA	Se muestra los tipos de listado disponibles
PRE-CONDICIONES	

POST-CONDICIONES	

NOMBRE	seleccionarListado(tipo, fechaInicio, fechaFin)
RESPONSABILIDADES	Seleccionar un tipo de listado
REFERENCIAS	Casos de uso: Obtener listado
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha creado una instancia de listado	
POST-CONDICIONES	
▪ El listado Se ha asociado con cita o con paciente (según el atributo tipo) ▪ Se han modificado los atributos fechaInicio y fechaFin de listado	

NOMBRE	mostrarListado()
RESPONSABILIDADES	Mostrar un listado de citas o de pacientes (según el tipo elegido)
REFERENCIAS	Casos de uso: Obtener listado
NOTAS	---
EXCEPCIONES	---
SALIDA	Se muestra un listado de citas o de pacientes (según el tipo elegido)
PRE-CONDICIONES	
▪ Se ha creado una instancia de listado ▪ Se han modificado los atributos fechaInicio y fechaFin de listado	
POST-CONDICIONES	

NOMBRE	iniciarSincronizacionACE()
RESPONSABILIDADES	Iniciar el servicio de sincronización en la computadora de escritorio
REFERENCIAS	Casos de uso: Sincronizar datos
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	

POST-CONDICIONES	

NOMBRE	iniciarSincronizacion()
RESPONSABILIDADES	Iniciar el proceso de sincronización
REFERENCIAS	Casos de uso: Sincronizar datos
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha iniciado el servicio de sincronización en la computadora de escritorio	
POST-CONDICIONES	

NOMBRE	sincronizar()
RESPONSABILIDADES	Sincronizar los datos entre la base de datos en la computadora de escritorio y la base de datos en el dispositivo móvil

REFERENCIAS	Casos de uso: Sincronizar datos
NOTAS	---
EXCEPCIONES	---
SALIDA	---
PRE-CONDICIONES	
▪ Se ha iniciado el servicio de sincronización en la computadora de escritorio ▪ Se ha iniciado el proceso de sincronización en el dispositivo móvil	
POST-CONDICIONES	
▪ Se han sincronizado los datos entre la base de datos en la computadora de escritorio y la base de datos en el dispositivo móvil	

NOMBRE	mostrarMensaje(mensaje)
RESPONSABILIDADES	Mostrar un mensaje de texto
REFERENCIAS	Casos de uso: Sincronizar datos
NOTAS	---
EXCEPCIONES	---
SALIDA	Muestra un mensaje de texto
PRE-CONDICIONES	

POST-CONDICIONES	

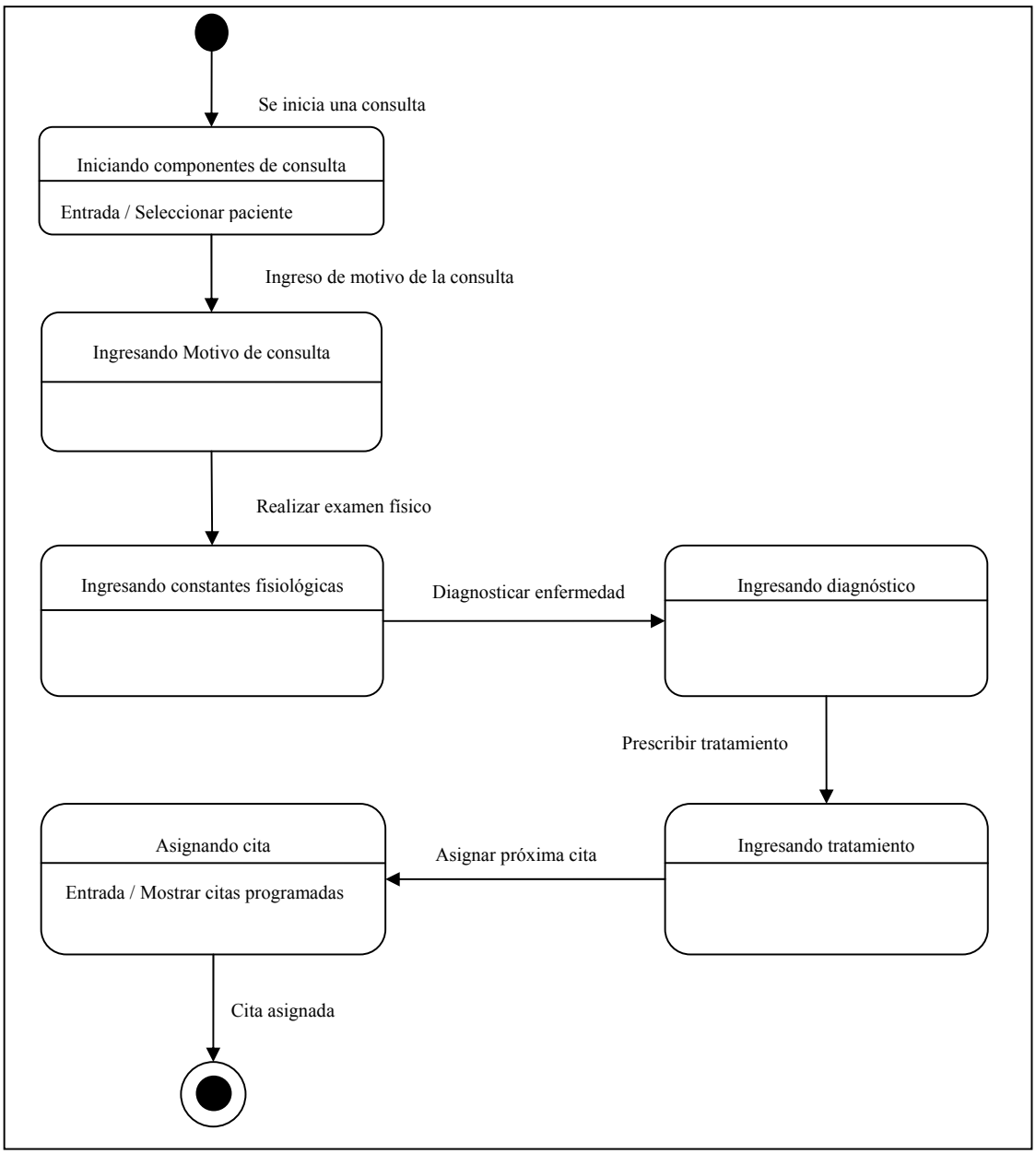
DETALLE DE LA TECNICA

DIAGRAMAS DE ESTADO

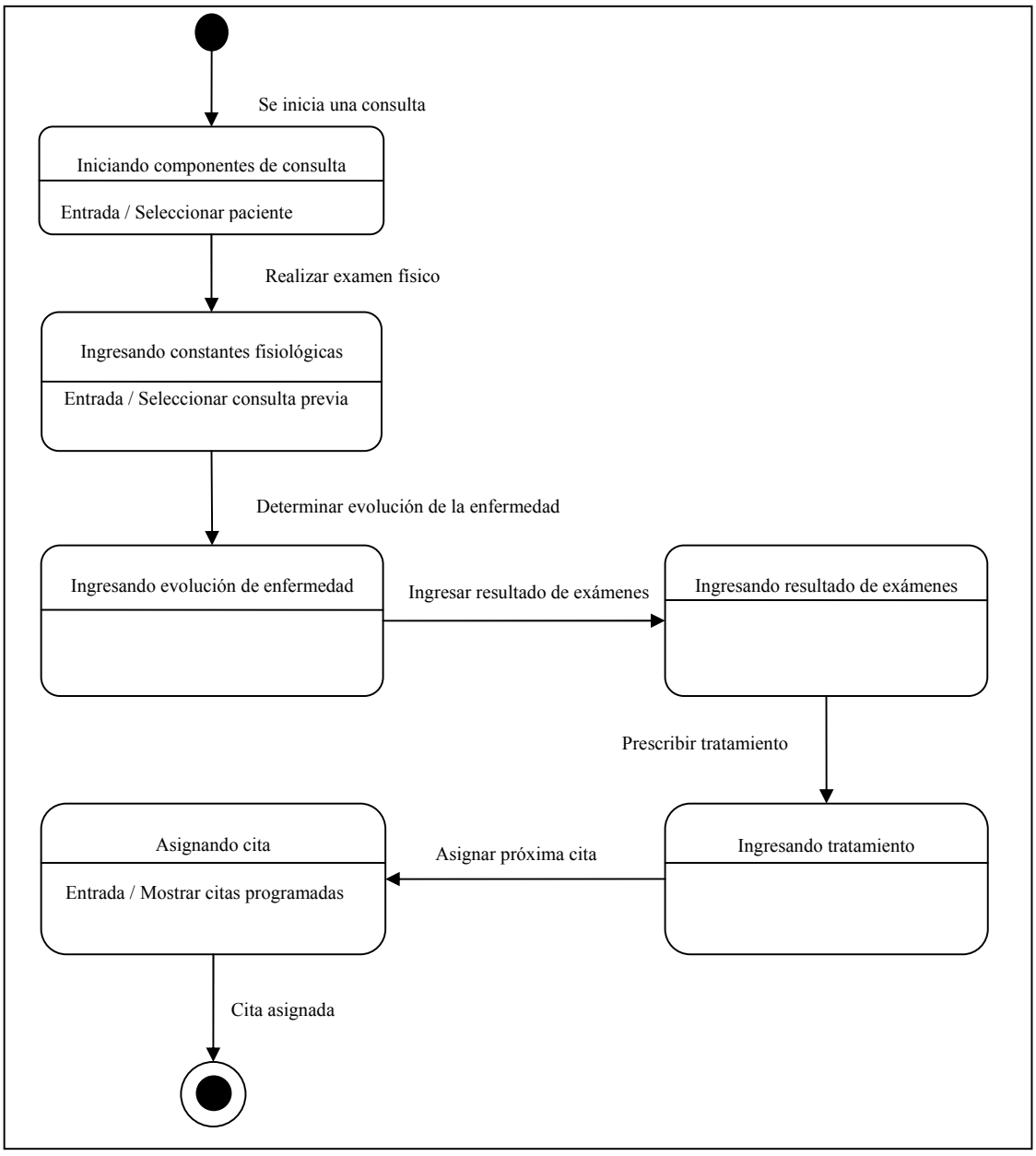
Los diagramas de estado se emplean para representar el estado y transisciones del sistema o estados de los objetos durante su vida en una aplicación.

2.3.3. Diagramas de Estado

Realizar primera consulta



Realizar consulta de seguimiento

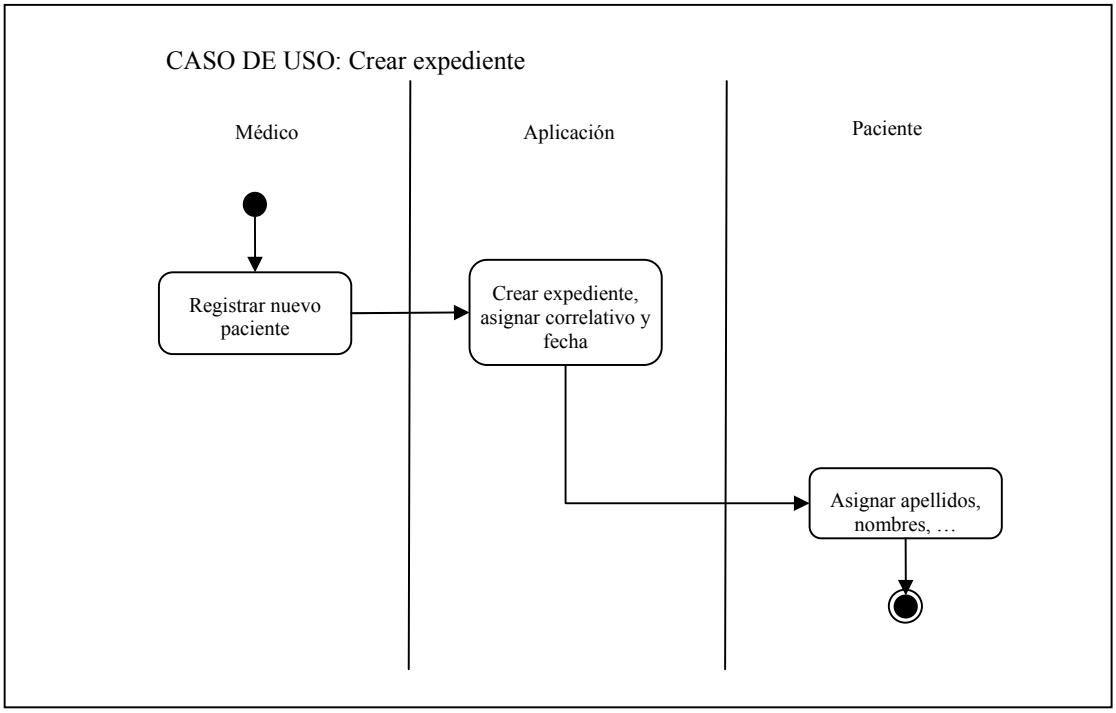


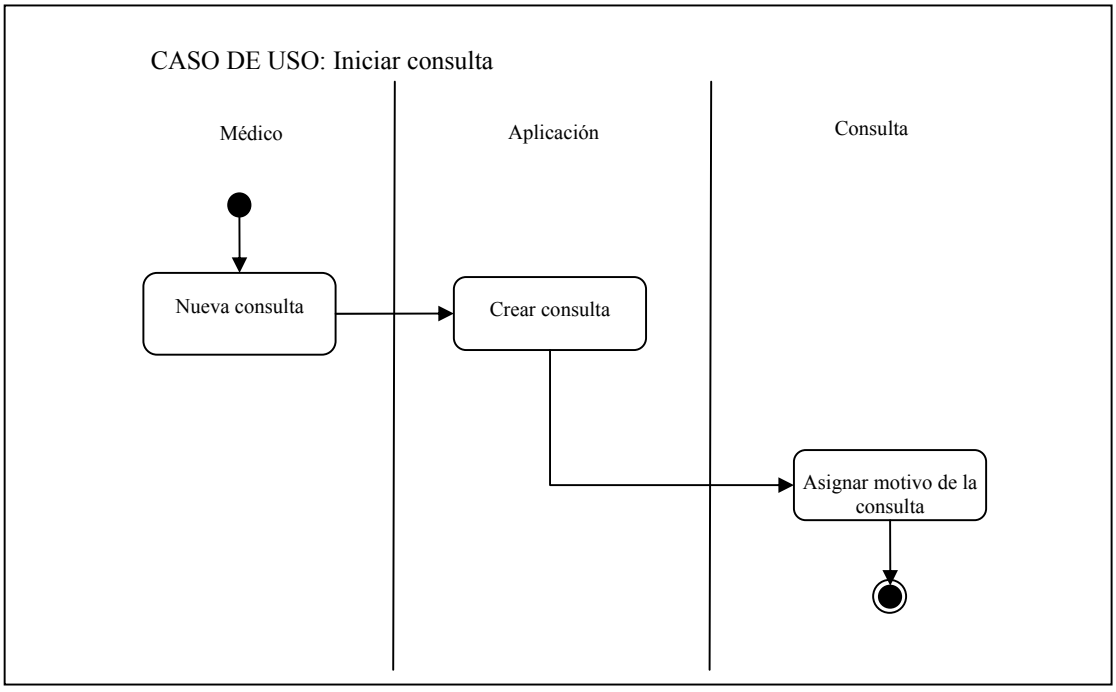
DETALLE DE LA TECNICA

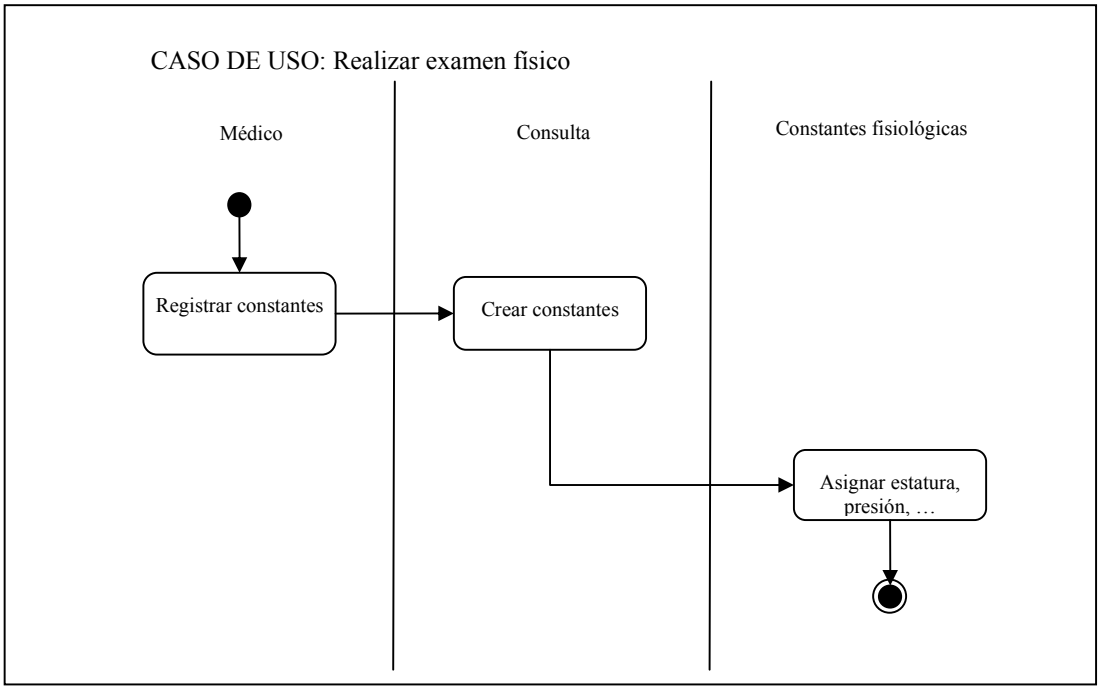
DIAGRAMAS DE ACTIVIDADES

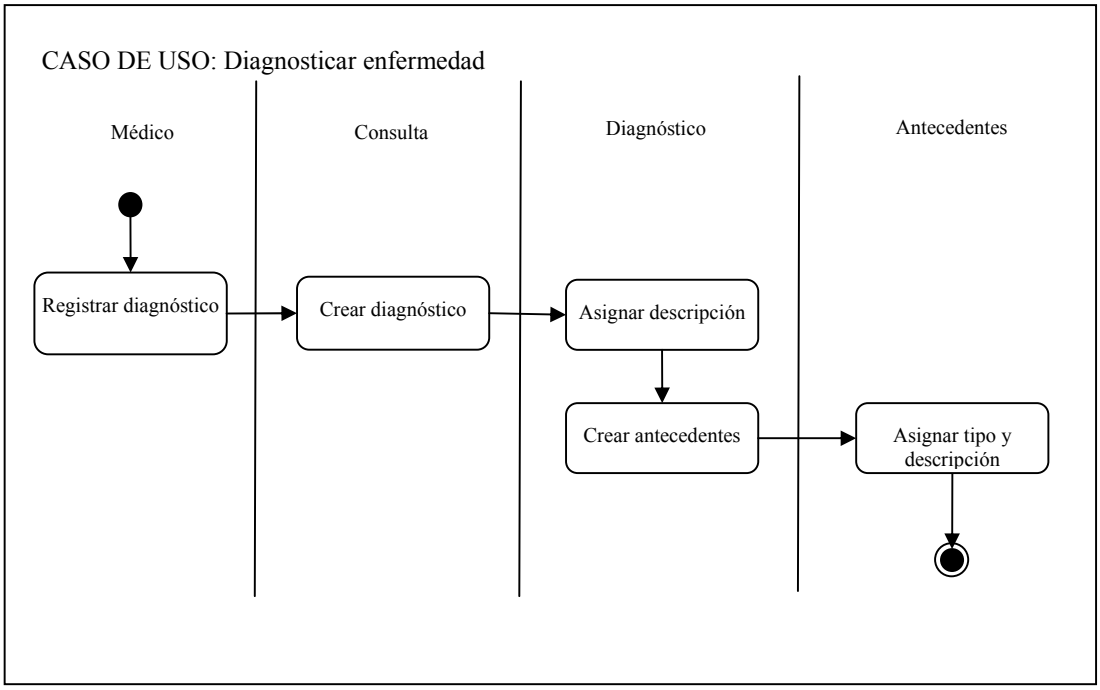
Los diagramas de actividades describen la secuencia de actividades en un sistema o aplicación. Las actividades se representan por medio de rectángulos y la secuencia es descrita por medio de flechas.

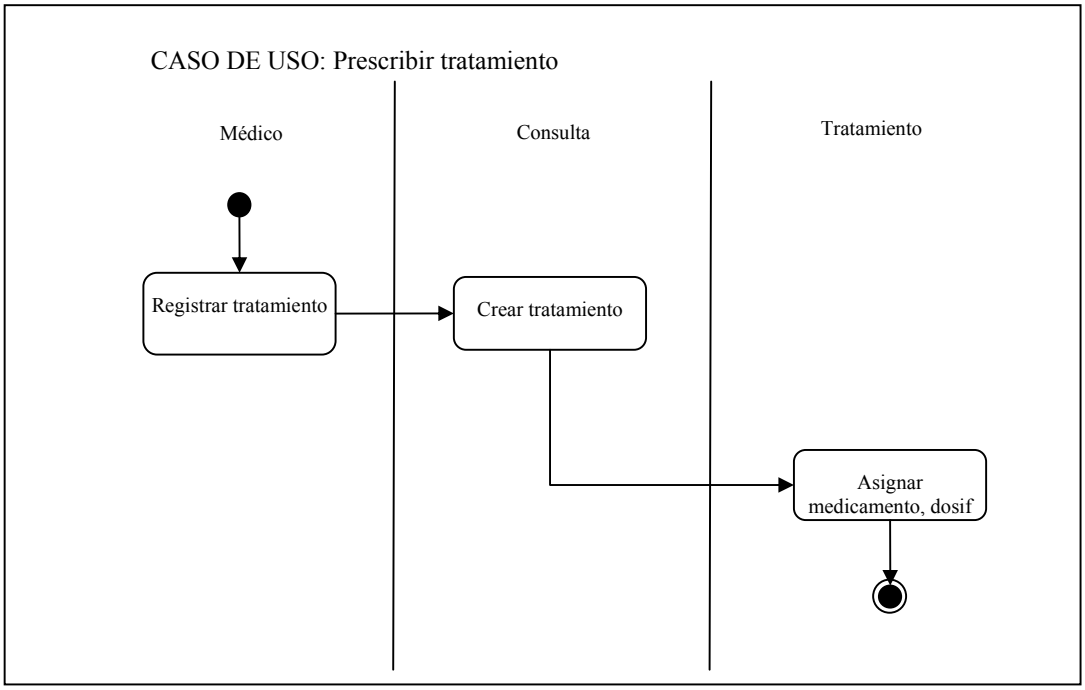
2.3.4. Diagramas de Actividades

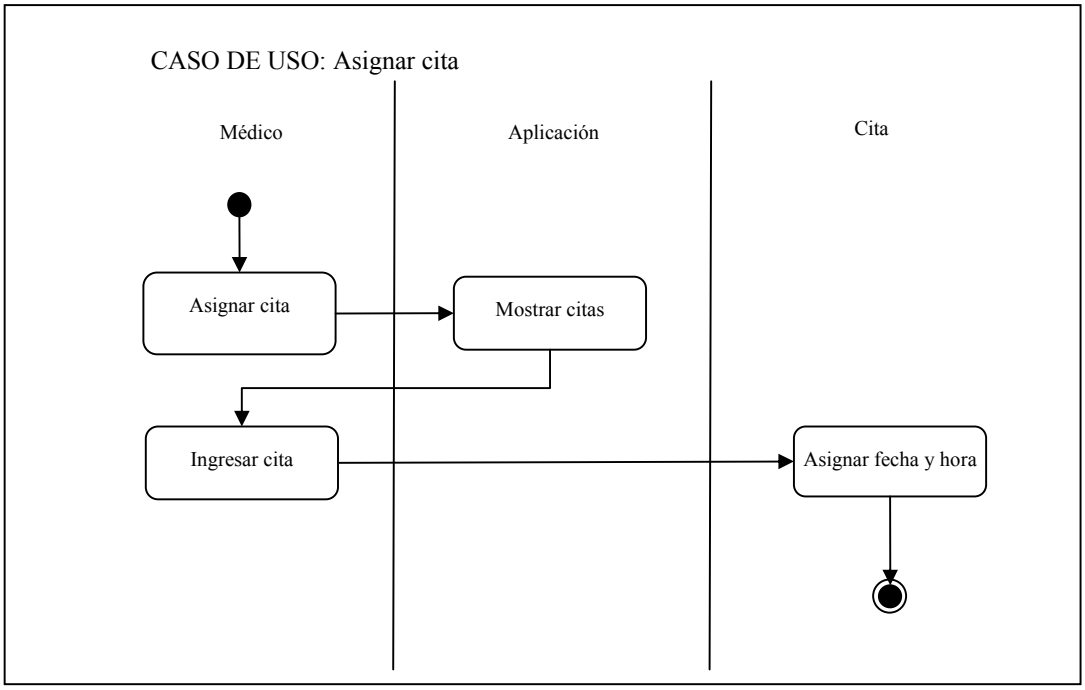


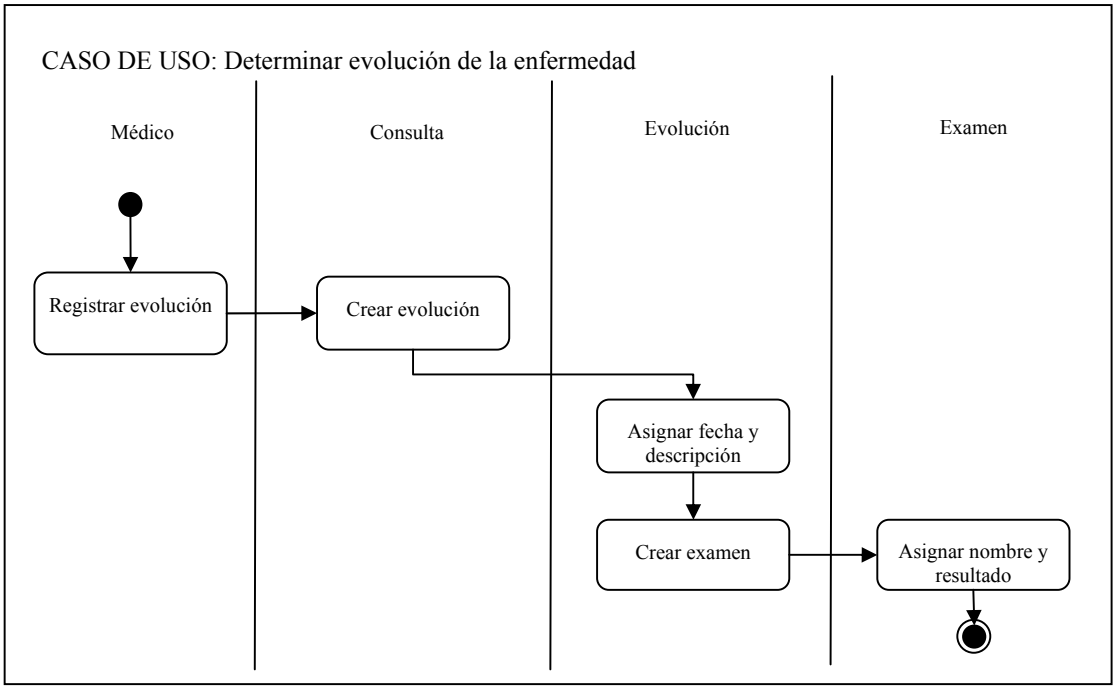


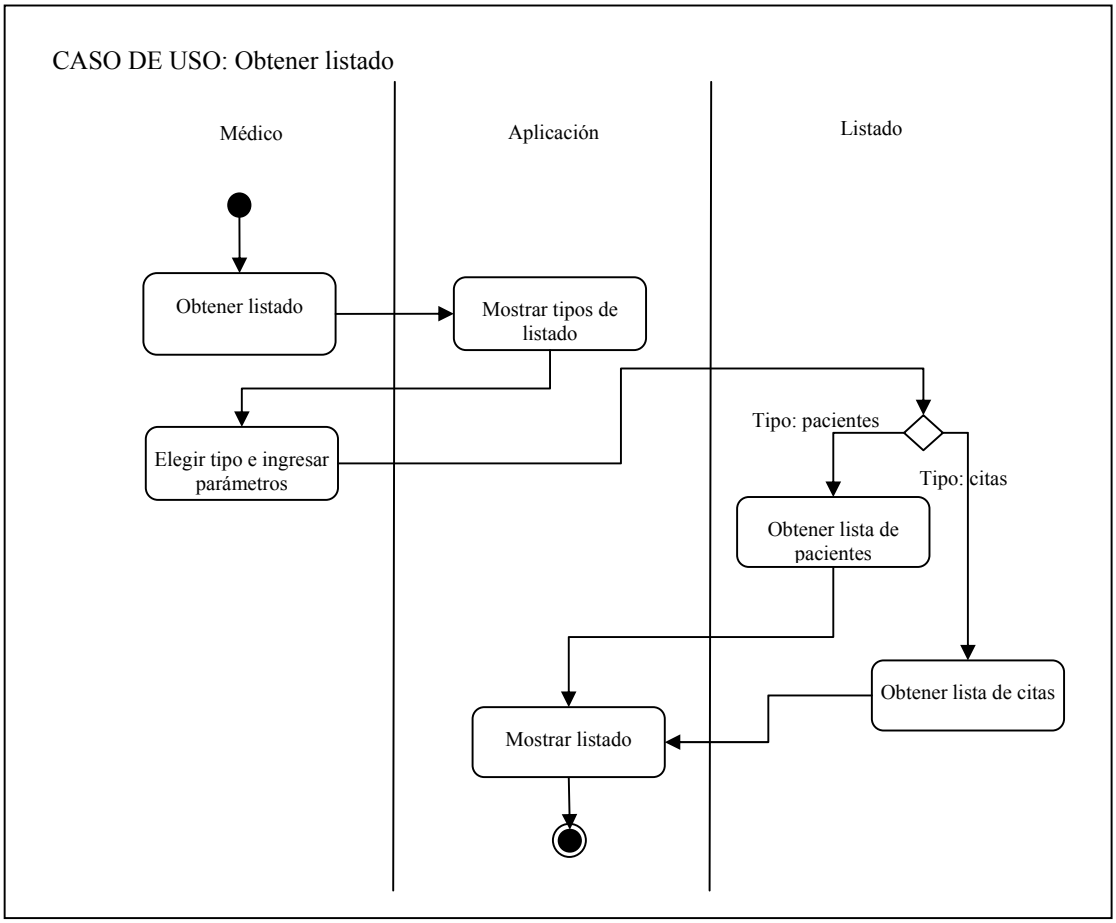

CASO DE USO: Realizar examen fisico
Médico
Consulta
Constantes fisiológicas
Registrar constantes
Crear constantes
Asignar estatura, presión, …

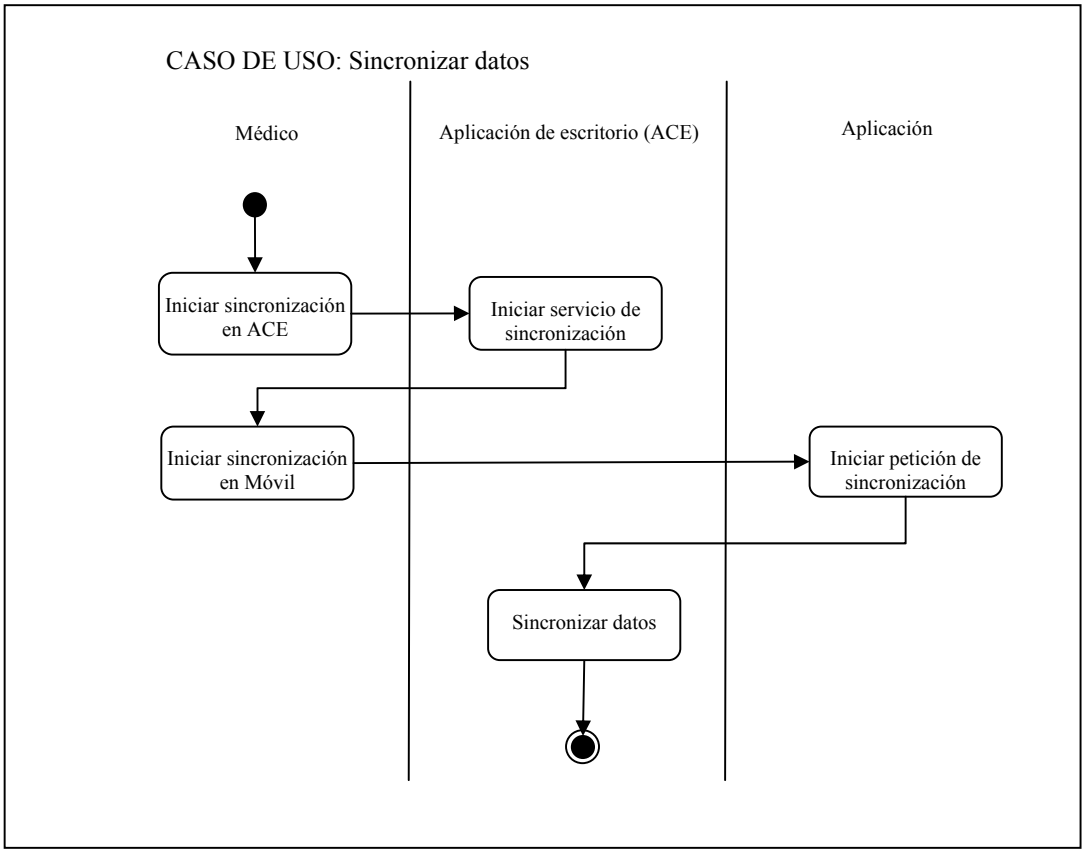

CASO DE USO: Diagnosticar enfermedad
Médico
Consulta
Diagnóstico
Antecedentes
Registrar diagnóstico
Crear diagnóstico
Asignar descripción
Crear antecedentes
Asignar tipo y descripción


CASO DE USO: Prescribir tratamiento
Médico
Consulta
Tratamiento
Registrar tratamiento
Crear tratamiento
Asignar
medicamento, dosif


CASO DE USO: Asignar cita
Médico
Aplicación
Cita
Asignar cita
Mostrar citas
Ingresar cita
Asignar fecha y hora


CASO DE USO: Determinar evolución de la enfermedad
Médico
Consulta
Evolución
Examen
Registrar evolución
Crear evolución
Asignar fecha y descripción
Crear examen
Asignar nombre y resultado


CASO DE USO: Obtener listado
Médico
Aplicación
Listado
Obtener listado
Mostrar tipos de listado
Elegir tipo e ingresar parámetros
Tipo: pacientes
Tipo: citas
Obtener lista de pacientes
Obtener lista de citas
Mostrar listado


CASO DE USO: Sincronizar datos
Médico
Aplicación de escritorio (ACE)
Aplicación
Iniciar sincronización en ACE
Iniciar servicio de sincronización
Iniciar sincronización en Móvil
Iniciar petición de sincronización
Sincronizar datos

DETALLE DE LA TECNICA

MODELO CONCEPTUAL REFINADO

Después de desarrollar las técnicas correspondientes al análisis, se debe realizar un refinamiento del modelo conceptual, ya que, se tiene una mejor comprensión del dominio del problema. Es posible identificar nuevos objetos, nuevas asociaciones, nuevos atributos o determinar que algunos objetos pueden ser representados de diferente forma.

2.3.5. Modelo Conceptual Refinado

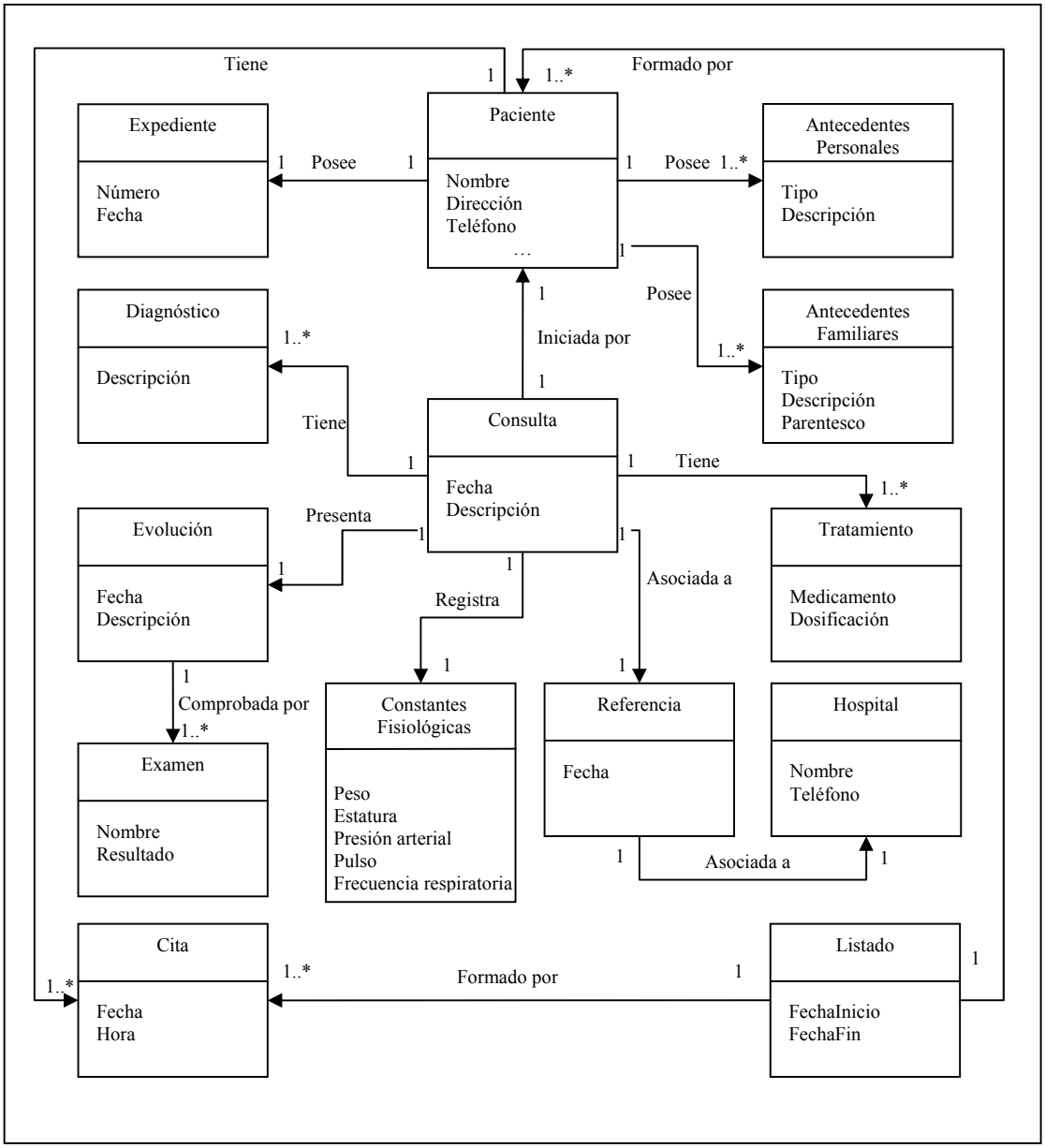


Figura 2.3 Modelo conceptual refinado

Descripción de atributos

OBJETO	expediente	
DESCRIPCION	Contiene el número de expediente asignado al paciente	
ATRIBUTOS		
numero	Sirve para identificar a un paciente, es un número correlativo, según la creación del expediente.	
	TIPO	Numérico
	LONGITUD	4 caracteres
	VALOR POR DEFECTO	Correlativo
	FORMATO	####
fecha	Almacena la fecha de creación del expediente del paciente.	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	Toma la fecha del sistema
	FORMATO	dd/mm/aa

OBJETO	paciente	
DESCRIPCION	Contiene los datos de un paciente	
ATRIBUTOS		
apellidos	Contiene los apellidos del paciente	
	TIPO	Alfabético
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
nombres	Contiene los nombres del paciente	
	TIPO	Alfabético

	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
sexo	Almacena el sexo del paciente	
	TIPO	Alfanumérico
	LONGITUD	1 caracter
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: 0, 1 (0:Masculino, 1:Femenino)
fechaNac	Almacena la fecha de nacimiento del paciente	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	N/A
	FORMATO	dd/mm/aa
estadoCivil	Contiene el estado civil del paciente	
	TIPO	Alfabético
	LONGITUD	10 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: soltero, casado, viudo, acompañado.
direccion	Almacena la dirección del paciente	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A

	FORMATO	Posibles valores: A-Z, a-z, 0-9 y caracteres especiales.
telefono	Almacena el número de teléfono fijo del paciente	
	TIPO	Alfanumérico
	LONGITUD	8 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	########
movil	Almacena el número de teléfono móvil del paciente	
	TIPO	Alfanumérico
	LONGITUD	8 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	##########
email	Contiene la dirección de correo electrónico del paciente	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9 y caracteres especiales.
tipoDoc	Contiene el tipo de documento presentado por el paciente	
	TIPO	Alfanumérico
	LONGITUD	9 caracteres
	VALOR POR DEFECTO	DUI
	FORMATO	Posibles valores: Carnet, DUI, DNI, pasaporte.

numeroDoc	Contiene el número del documento presentado por el paciente	
	TIPO	Alfanumérico
	LONGITUD	20 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.
ocupacion	Almacena la ocupación del paciente	
	TIPO	Alfanumérico
	LONGITUD	30 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: jornalero, estudiante, comerciante, empleado, desempleado.
padre	Almacena el primer nombre y primer apellido del padre del paciente	
	TIPO	Alfabético
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
madre	Almacena el primer nombre y apellido de soltera de la madre del paciente	
	TIPO	Alfabético
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
conyuge	Almacena el primer nombre y primer apellido del cónyuge del paciente	
	TIPO	Alfabético

	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
responsable	Almacena el primer nombre y primer apellido del responsable del paciente	
	TIPO	Alfabético
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
direccionResp	Almacena la dirección del responsable	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9 y caracteres especiales.
telefonoResp	Almacena el teléfono de contacto del responsable	
	TIPO	Alfanumérico
	LONGITUD	8 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	########
nombrePDatos	Contiene el nombre de la persona que proporcionó los datos del paciente	
	TIPO	Alfabético
	LONGITUD	50 caracteres

	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z
parentescoPDatos	Contiene el parentesco de la persona que proporciona los datos y el paciente	
	TIPO	Alfabético
	LONGITUD	10 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: progenitor, hermano, abuelo, cónyuge, amigo.
tipoDocPDatos	Contiene el tipo de documento de la persona que proporciona los datos	
	TIPO	Alfanumérico
	LONGITUD	9 caracteres
	VALOR POR DEFECTO	DUI
	FORMATO	Posibles valores: Carnet, DUI, DNI, pasaporte.
numeroDocPDatos	Contiene el número del documento presentado por la persona que proporcionó los datos del paciente	
	TIPO	Alfanumérico
	LONGITUD	20 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	antecedentesPersonales
DESCRIPCION	Contiene datos relativos a los antecedentes personales del paciente.
ATRIBUTOS	

tipo	Almacena el tipo de antecedente que posee el paciente	
	TIPO	Alfabético
	LONGITUD	13 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: alérgicos, tóxicos, fisiológicos, patológicos, ginecológicos, quirúrgicos
descripción	Almacena una descripción del tipo de antecedente del paciente	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	antecedentesFamiliares	
DESCRIPCION	Contiene datos relativos a los antecedentes familiares del paciente, relacionados con la enfermedad.	
ATRIBUTOS		
tipo	Almacena el tipo de antecedente familiar que posee el paciente	
	TIPO	Alfabético
	LONGITUD	13 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: alérgicos, fisiológicos, patológicos.
descripción	Almacena una descripción del tipo de antecedente del paciente	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A

	FORMATO	Posibles valores: A-Z, a-z, 0-9.
parentesco	Almacena el parentesco entre el paciente y el familiar cuyo antecedente se está describiendo.	
	TIPO	Alfabético
	LONGITUD	6 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: padre, madre, abuelo, abuela.

OBJETO	diagnostico	
DESCRIPCION	Contiene datos relativos al diagnóstico de la enfermedad que padece el paciente.	
ATRIBUTOS		
descripción	Almacena una descripción del diagnóstico de la enfermedad	
	TIPO	Alfanumérico
	LONGITUD	200 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	consulta	
DESCRIPCION	Contiene datos relativos al motivo de la consulta	
ATRIBUTOS		
fecha	Almacena la fecha de la consulta	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	Toma la fecha del sistema

	FORMATO	dd/mm/aa
descripción	Almacena una descripción del motivo de la consulta	
	TIPO	Alfanumérico
	LONGITUD	100 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	evolucion	
DESCRIPCION	Contiene datos relativos a la evolución de la enfermedad	
ATRIBUTOS		
fecha	Almacena la fecha de la consulta en la que se verifica la evolución de la enfermedad	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	Toma la fecha del sistema
	FORMATO	dd/mm/aa
descripción	Almacena una descripción de la evolución de la enfermedad	
	TIPO	Alfanumérico
	LONGITUD	200 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	tratamiento
DESCRIPCION	Contiene datos relativos al tratamientos prescrito
ATRIBUTOS	

medicamento	Almacena el nombre del medicamento prescrito	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.
dosificacion	Almacena la dosificación respectiva del medicamento prescrito	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	examen	
DESCRIPCION	Contiene el resultado de los exámenes de laboratorio ordenados al paciente	
ATRIBUTOS		
nombre	Almacena el nombre del examen de laboratorio	
	TIPO	Alfanumérico
	LONGITUD	25 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.
resultado	Almacena el resultado del examen de laboratorio	
	TIPO	Alfanumérico
	LONGITUD	25 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.

OBJETO	constantesFisiologicas	
DESCRIPCION	Contiene los resultados del examen físico realizado al paciente	
ATRIBUTOS		
peso	Almacena el peso (libras) del paciente	
	TIPO	numérico
	LONGITUD	3 dígitos
	VALOR POR DEFECTO	N/A
	FORMATO	###, Valores admitidos: Mayor que 0 y menor que 600
estatura	Almacena la estatura (centímetros) del paciente	
	TIPO	numérico
	LONGITUD	3 dígitos
	VALOR POR DEFECTO	N/A
	FORMATO	###, Valores admitidos: Mayor que 0 y menor que 300
presionArterialSistólica	Almacena la presión arterial (mmHg: milímetros de mercurio) sistólica del paciente	
	TIPO	numérico
	LONGITUD	3 dígitos
	VALOR POR DEFECTO	N/A
	FORMATO	###, Valores admitidos: Mayor o igual que 90 y menor o igual que 130
presionArterialDiastólica	Almacena la presión arterial (mmHg: milímetros de mercurio) diastólica del paciente	
	TIPO	numérico
	LONGITUD	2 dígitos
	VALOR POR DEFECTO	N/A

	FORMATO	##, Valores admitidos: Mayor o igual que 60 y menor o igual que 90
pulso	Almacena el pulso (latidos por minuto) del paciente	
	TIPO	Numérico
	LONGITUD	3 dígitos
	VALOR POR DEFECTO	N/A
	FORMATO	###, Valores admitidos: Mayor 70 y menor que 100
frecuenciaRespiratoria	Almacena la frecuencia respiratoria (respiraciones por minuto) del paciente	
	TIPO	numérico
	LONGITUD	2 dígitos
	VALOR POR DEFECTO	N/A
	FORMATO	##, Valores admitidos: Mayor 15 y menor que 25

OBJETO	referencia	
DESCRIPCION	Contiene la fecha en que se refiere un paciente a un hospital	
ATRIBUTOS		
fecha	Almacena el nombre del medicamento prescrito	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	Toma la fecha del sistema
	FORMATO	dd/mm/aa

OBJETO	hospital
DESCRIPCION	Contiene datos del hospital al que se refiere un paciente

ATRIBUTOS		
nombre	Almacena el nombre del hospital	
	TIPO	Alfanumérico
	LONGITUD	50 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	Posibles valores: A-Z, a-z, 0-9.
telefono	Almacena el número de teléfono del hospital	
	TIPO	Alfanumérico
	LONGITUD	8 caracteres
	VALOR POR DEFECTO	N/A
	FORMATO	########

OBJETO	cita	
DESCRIPCION	Contiene datos relativos a una cita con un paciente	
ATRIBUTOS		
fecha	Almacena la fecha de la cita	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	N/A
	FORMATO	dd/mm/aa
hora	Almacena la hora de la cita	
	TIPO	Tiempo
	LONGITUD	N/A
	VALOR POR DEFECTO	N/A
	FORMATO	hh:mm

OBJETO	listado	
DESCRIPCION	Contiene la fecha de inicio y la fecha de fin del listado solicitado	
ATRIBUTOS		
fechaInicio	Almacena la fecha de inicio del listado	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	N/A
	FORMATO	dd/mm/aa
fechaFin	Almacena la fecha fin del listado	
	TIPO	Fecha
	LONGITUD	N/A
	VALOR POR DEFECTO	N/A
	FORMATO	dd/mm/aa

2.4. Diseño

DETALLE DE LA TECNICA

DESCRIPCION DE CASOS DE USO REALES

Aquí se hace una descripción de los casos de uso, e incluye la interacción con las interfaces de usuario que tendrá el sistema o aplicación.

2.4.1. Descripción de Casos de Uso Reales

Se presentará a continuación la descripción de casos de uso acompañados de los formularios respectivos; es importante aclarar que dichos formularios son genéricos, es decir, que principalmente representan los controles y viñetas, no así el diseño final de los mismos.

CASO DE USO	Crear expediente
ACTORES	Paciente, Médico
PROPOSITO	Crear un expediente para un nuevo paciente.
VISION	Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Y es primera vez que el médico atiende al paciente.
TIPO	Primario y real
REFERENCIAS	Funciones: R-001, R-002

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El paciente se presenta ante el médico.	
2. El médico indica a la aplicación que ingresará los datos de un nuevo paciente.	3. La aplicación presenta la pantalla Ventana_crearExpediente (figura 2.20 a).
4. El médico ingresa los siguientes datos: apellidos, nombres, sexo, fecha de nacimiento, estado civil, dirección, número de teléfono fijo, número de teléfono móvil, dirección de correo electrónico, tipo y número de documento de identidad personal y ocupación, en Ventana_crearExpediente (figura 2.20 a) y da clic en el botón aceptar.	5. La aplicación almacena temporalmente los datos del nuevo paciente y presenta la pantalla Ventana_crearExpediente (figura 2.20 b).
6. El médico ingresa los siguientes datos: nombre del padre, nombre de la madre, nombre del cónyuge, nombre del responsable, dirección del responsable, teléfono del responsable, nombre de la persona que proporciona los datos, parentesco de la persona que proporciona los datos, tipo y número de documento de identidad de la persona que proporciona los datos, en Ventana_crearExpediente (figura 2.20 b) y	7. La aplicación almacena los datos del nuevo paciente.

selecciona el botón aceptar.	
CURSOS ALTERNATIVOS	
Paso 4: El médico cancela el ingreso de datos, para lo cual selecciona el botón cancelar de la pantalla Ventana_crearExpediente (figura 2.20 a). Paso 6: El médico cancela el ingreso de datos, para lo cual selecciona el botón cancelar de la pantalla Ventana_crearExpediente (figura 2.20 b).	La aplicación elimina los datos almacenados temporalmente.

CASO DE USO	Iniciar consulta
ACTORES	Paciente, Médico
PROPOSITO	Registrar una nueva consulta médica.
VISION	Este caso de uso inicia cuando el paciente se presenta al consultorio del médico o cuando el médico hace una visita al paciente. Y el médico indaga sobre el motivo de la consulta.
TIPO	Primario y real
REFERENCIAS	Funciones: R-003

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El paciente se presenta ante el médico. 2. El médico indica a la aplicación que ingresará los datos de una consulta. 4. El médico ingresa el motivo de la consulta en la caja de texto E de Ventana_consulta (figura 2.21).	3. La aplicación presenta la pantalla Ventana_consulta (figura 2.21).
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Realizar examen físico
ACTORES	Paciente, Médico
PROPOSITO	Registrar resultados del examen físico.
VISION	Este caso de uso se presenta cuando el médico mide el peso, estatura y las constantes vitales del paciente: presión arterial, pulso, frecuencia respiratoria y temperatura.
TIPO	Primario y real
REFERENCIAS	Funciones: R-004

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico realiza un chequeo de las constantes vitales del paciente: presión arterial, pulso, frecuencia respiratoria y temperatura; y mide su peso y estatura. 3. El médico ingresa los resultados del examen físico en las cajas de texto F, G, H, I, J de Ventana_consulta (figura 2.21).	
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Diagnosticar enfermedad
ACTORES	Paciente, Médico
PROPOSITO	Registrar el resultado del diagnóstico de la enfermedad.
VISION	Este caso de uso sucede cuando el médico indaga sobre los síntomas que presenta el paciente, y los antecedentes personales y familiares del mismo. Y formula su diagnóstico.
TIPO	Primario y real
REFERENCIAS	Funciones: R-005

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico pide al paciente que describa los	

síntomas y en base a ellos pregunta sobre los antecedentes personales y familiares. 2. El paciente describe los síntomas y relata sobre antecedentes personales y familiares relacionados con la enfermedad. 3. El médico selecciona el botón anteced de Ventana_consulta (figura 2.21).	4. La aplicación muestra la pantalla Ventana_menuAntecedentes (figura 2.22).
5. El médico selecciona el botón personales de Ventana_menuAntecedentes (figura 2.22).	6. La aplicación muestra la pantalla Ventana_antecedentesPersonales (figura 2.17).
7. El médico selecciona el tipo de antecedente de la lista desplegable A e ingresa la descripción en B de Ventana_antecedentesPersonales (figura 2.17) y selecciona el botón aceptar.	8. La aplicación almacena temporalmente el antecedente y muestra nuevamente la pantalla Ventana_menuAntecedentes (figura 2.22).
9. El médico selecciona el botón salir de Ventana_menuAntecedentes (figura 2.22).	10. La aplicación muestra nuevamente la pantalla Ventana_consulta (figura 2.21).
11. El médico realiza el diagnóstico de la enfermedad y lo ingresa en la caja de texto K de Ventana_consulta (figura 2.21).	

CURSOS ALTERNATIVOS

Paso 5-7:

El médico selecciona el botón familiares de Ventana_menuAntecedentes (figura 2.22).

La aplicación muestra la pantalla Ventana_antecedentesFamiliares (figura 2.18).

El médico selecciona el parentesco de la lista despligable A, el tipo de antecedente de la lista desplegable B e ingresa la descripción en C de Ventana_antecedentesFamiliares (figura 2.18) y selecciona el botón aceptar.

Paso 11:

El médico realiza diagnóstico de la enfermedad y refiere al paciente a un hospital, e ingresa el diagnóstico en la caja de texto K de Ventana_consulta (figura 2.21) y selecciona el hospital al que referirá al paciente de la lista desplegable L.

CASO DE USO	Prescribir tratamiento
ACTORES	Médico
PROPOSITO	Registrar el tratamiento al que se someterá el paciente.

VISION	Este caso de uso se da cuando el médico prescribe los medicamentos y su dosificación.
TIPO	Primario y real
REFERENCIAS	Funciones: R-006, R-009

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico prescribe los medicamentos y la dosificación respectiva.	
2. El médico selecciona el botón tratam de Ventana_consulta (figura 2.21).	3. La aplicación muestra la pantalla Ventana_menuTratamiento (figura 2.19).
4. El médico selecciona el botón agregar de Ventana_menuTratamiento (figura 2.19).	5. La aplicación muestra la pantalla Ventana_tratamiento (figura 2.20).
6. El médico ingresa el medicamento en la caja de texto A y la dosificación en la caja de texto B de Ventana_tratamiento (figura 2.20) y selecciona el botón aceptar.	7. La aplicación almacena temporalmente los datos del tratamiento y muestra nuevamente la pantalla Ventana_menuTratamiento (figura 2.19).
8. El médico selecciona el botón salir Ventana_menuTratamiento (figura 2.19).	9. La aplicación muestra nuevamente la pantalla Ventana_consulta (figura 2.21).
10. El médico indica al paciente la realización de exámenes de laboratorio y selecciona el botón aceptar de Ventana_consulta (figura 2.21).	11. La aplicación almacena los datos temporales (antecedentes y tratamiento) y los ingresados en Ventana_consulta (figura 2.21) y muestra la pantalla Ventana_citas (figura 2.21).
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Asignar cita
ACTORES	Médico
PROPOSITO	Asignar fecha y hora de próxima cita de un paciente.
VISION	Este caso de uso sucede cuando el médico le asigna al paciente la fecha y hora de la próxima cita.

TIPO	Primario y real
REFERENCIAS	Funciones: R-007
CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico selecciona asignar cita de Ventana_citas (figura 2.21).	2. La aplicación muestra una lista de citas programadas cuya fecha sea mayor a la actual en Ventana_citasProgramadas (figura 2.22).
3. El médico ingresa fecha y hora de la nueva cita en Ventana_citasProgramadas (figura 2.22) y selecciona el botón aceptar.	4. La aplicación almacena la fecha y hora de la cita.
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Determinar evolución de la enfermedad
ACTORES	Médico, Paciente
PROPOSITO	Registrar los resultados que ha tenido el paciente después del tratamiento.
VISION	Este caso de uso inicia cuando el paciente se ha presentado al consultorio del médico o cuando el médico ha realizado una visita al paciente, con el propósito de darle seguimiento a una enfermedad previamente consultada, entonces el médico indaga sobre los resultados del tratamiento.
TIPO	Primario y real
REFERENCIAS	Funciones: R-008, R-009, R-010
CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El paciente presenta el resultado de los exámenes de laboratorio. 2. El médico pregunta al paciente respecto de los resultados que ha observado después del tratamiento. 3. El paciente describe los resultados. 4. El médico indica a la aplicación que ingresará la	5. La aplicación muestra la pantalla

evolución de la enfermedad. 6. El médico ingresa los datos de la evolución de la enfermedad en la caja de texto K. 7. El médico selecciona el botón examen de Ventana_consultaEvolución (figura 2.17). 9. El médico selecciona el botón agregar de Ventana_menuExamen (figura 2.18). 11. El médico selecciona el tipo de examen de la lista desplegable A e ingresa el resultado en la caja de texto B de Ventana_examen (figura 2.19) y selecciona el botón aceptar. 13. El médico selecciona el botón salir de Ventana_menuExamen (figura 2.18). 15. El médico selecciona el botón aceptar de Ventana_consultaEvolución (figura 2.17).	Ventana_consultaEvolución (figura 2.17). 8. La aplicación muestra la pantalla Ventana_menuExamen (figura 2.18). 10. La aplicación muestra la pantalla Ventana_examen (figura 2.19). 12. La aplicación almacena temporalmente el resultado del examen y muestra nuevamente la pantalla Ventana_menuExamen (figura 2.18). 14. La aplicación muestra nuevamente la pantalla Ventana_consultaEvolución (figura 2.17). 16. La aplicación almacena los datos temporales de examenes y los ingresados en Ventana_consultaEvolución (figura 2.17).
CURSOS ALTERNATIVOS	
Paso 15: El médico realiza diagnóstico de la enfermedad y refiere al paciente a un hospital e ingresa el diagnóstico en la caja de texto K de Ventana_consultaEvolucion (figura 2.17) y selecciona el hospital al que referirá al paciente de la lista desplegable L y selecciona el botón aceptar.	La aplicación almacena los datos de la evolución de la enfermedad, resultado de exámenes y datos del hospital.

CASO DE USO	Obtener listado
ACTORES	Médico
PROPOSITO	Obtener un listado: de citas o de pacientes que atiende.
VISION	Este caso de uso se presenta cuando el médico requiere un listado de las citas para una fecha determinada, o un listado de sus pacientes.
TIPO	Primario y real
REFERENCIAS	Funciones: R-011, R-012

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico indica a la aplicación que desea obtener un listado. 3. El médico selecciona el tipo de listado de la lista desplegable A de Ventana_tipoListado (figura 2.20) y las fechas de inicio y fin y selecciona el botón aceptar.	2. La aplicación muestra la pantalla Ventana_tipoListado (figura 2.20). 4. La aplicación muestra la pantalla Ventana_listado (figura 2.21).
CURSOS ALTERNATIVOS	
---	---

CASO DE USO	Sincronizar datos
ACTORES	Médico, ACE
PROPOSITO	Sincronizar la base de datos de la computadora de escritorio y la base de datos del dispositivo móvil.
VISION	Este caso de uso se da cuando el médico quiere sincronizar los datos entre el dispositivo móvil y la computadora de escritorio.
TIPO	Primario y real
REFERENCIAS	Funciones: R-013

CURSO TIPICO DE LOS EVENTOS	
ACCION DEL ACTOR	**RESPUESTA DE LA APLICACION**
1. El médico inicia el servicio de sincronización en la computadora de escritorio. 2. La aplicación en la computadora de escritorio (ACE) "escucha" las peticiones de sincronización provenientes del dispositivo móvil. 3. El médico inicia la sincronización en el dispositivo móvil. 5. ACE realiza la sincronización.	4. La aplicación hace la petición de sincronización. 6. La aplicación muestra la pantalla Ventana_sincronizacion (figura 2.22).
CURSOS ALTERNATIVOS	

---	---

DETALLE DE LA TECNICA

DIAGRAMAS DE COLABORACION

Un diagrama de colaboración es semejante al diagrama de secuencia, en el sentido de que representa la interacción entre los objetos.

Los objetos se representan por medio de rectángulos y los mensajes enviados de un objeto a otro mediante flechas, mostrando el nombre del mensaje, los parámetros y la secuencia del mensaje.

2.4.2. Diagramas de Colaboración

Lista de diagramas de colaboración

Crear expediente

- crearExpediente
- agregarDatosPaciente

Iniciar Consulta

- crearConsulta
- agregarMotivoConsulta
- agregarReferencia

Realizar examen físico

- agregarConstantes

Diagnosticar enfermedad

- agregarAntecedentePersonal
- agregarAntecedenteFamiliar
- agregarDiagnostico

Prescribir tratamiento

- agregarTratamiento

Asignar cita

- crearCita

Determinar evolución de la enfermedad

- agregarEvolución
- agregarExamen

Obtener listado

- obtenerListado

Sincronizar datos

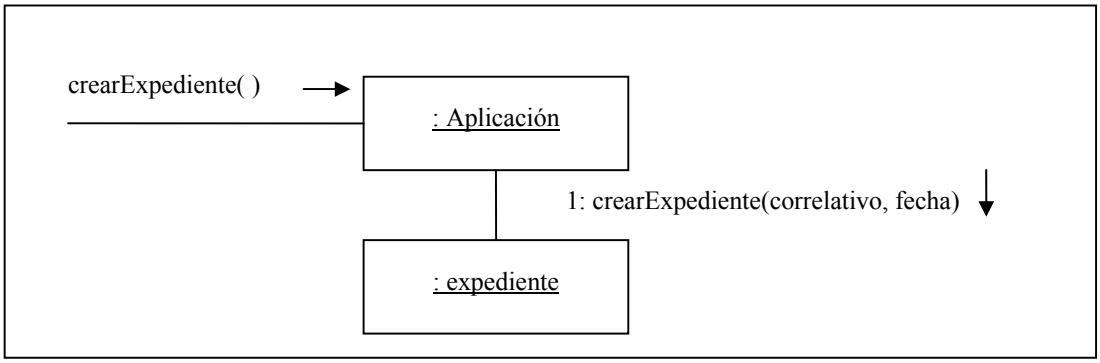
- sincronizar

Editar

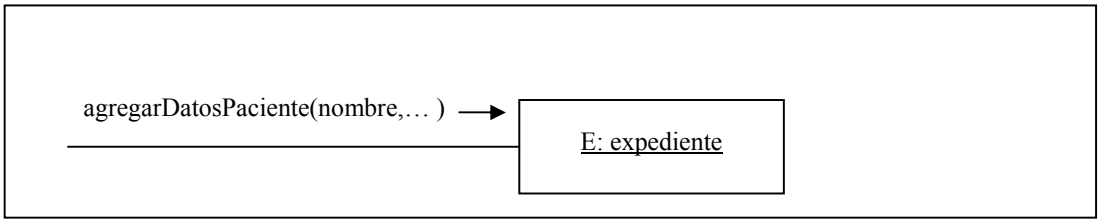
- editarExpediente
- editarConsulta
- editarEvolucion

CASO DE USO: Crear expediente

Operación: crearExpediente

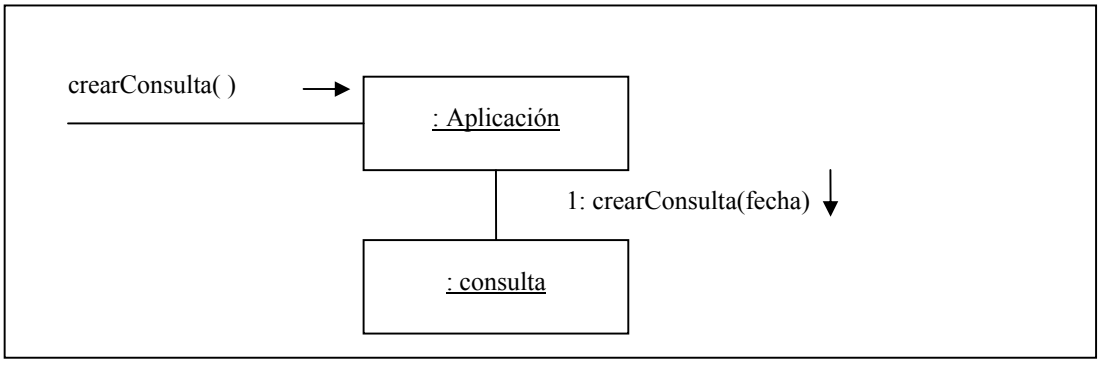


Operación: agregarDatosPaciente

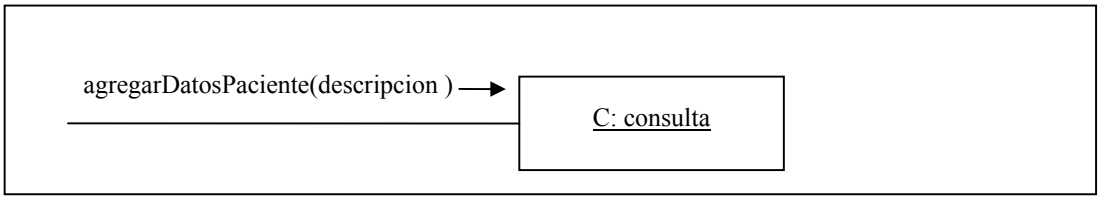


CASO DE USO: Iniciar consulta

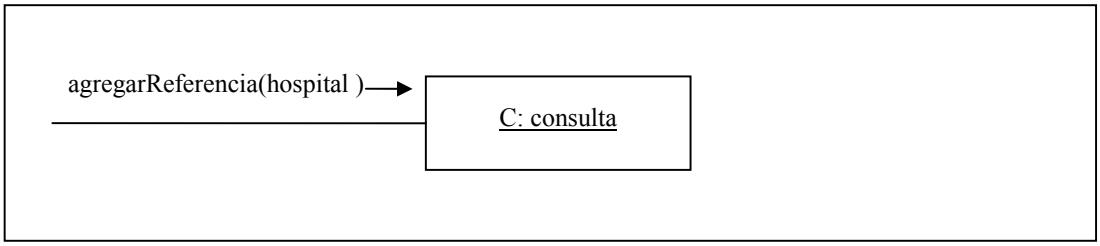
Operación: crearConsulta



Operación: agregarMotivoConsulta

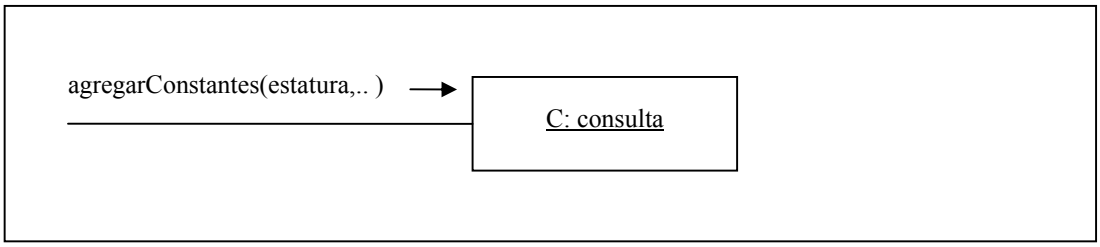


Operación: agregarReferencia



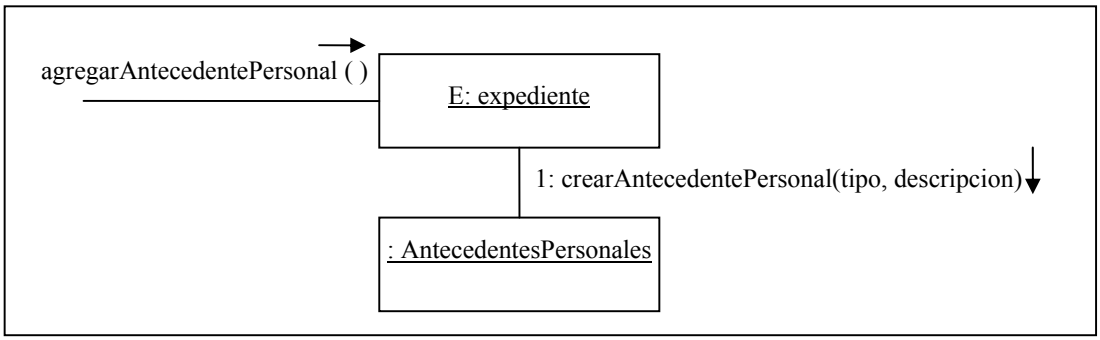
CASO DE USO: Realizar examen físico

Operación: agregarConstantes

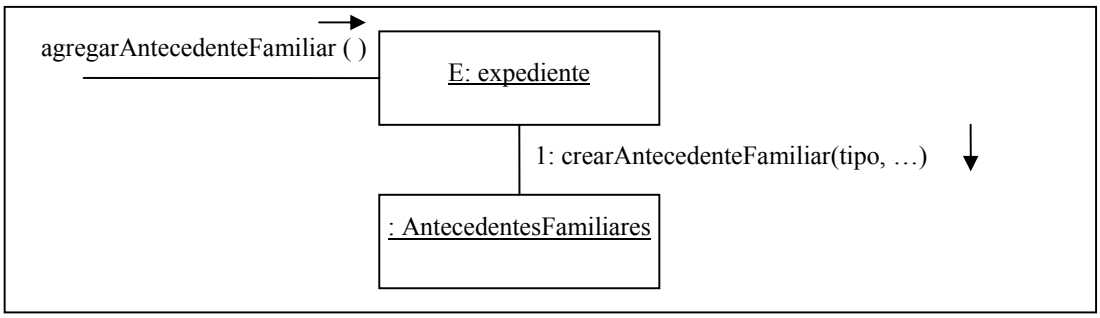


CASO DE USO: Diagnosticar enfermedad

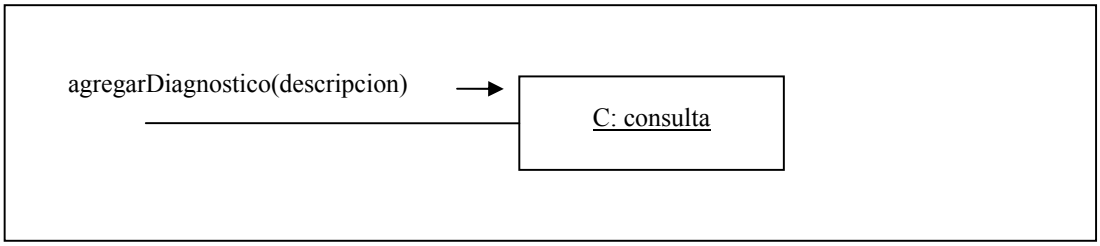
Operación: agregarAntecedentePersonal



Operación: agregarAntecedenteFamiliar

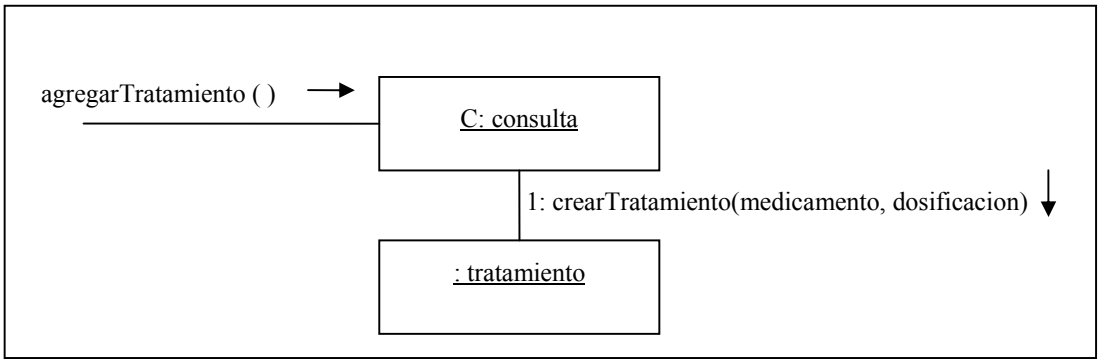


Operación: agregarDiagnostico



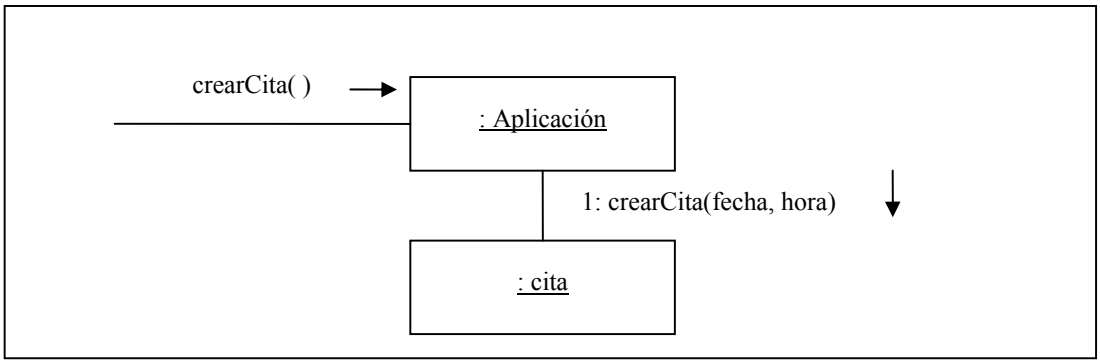
CASO DE USO: Prescribir tratamiento

Operación: agregarTratamiento



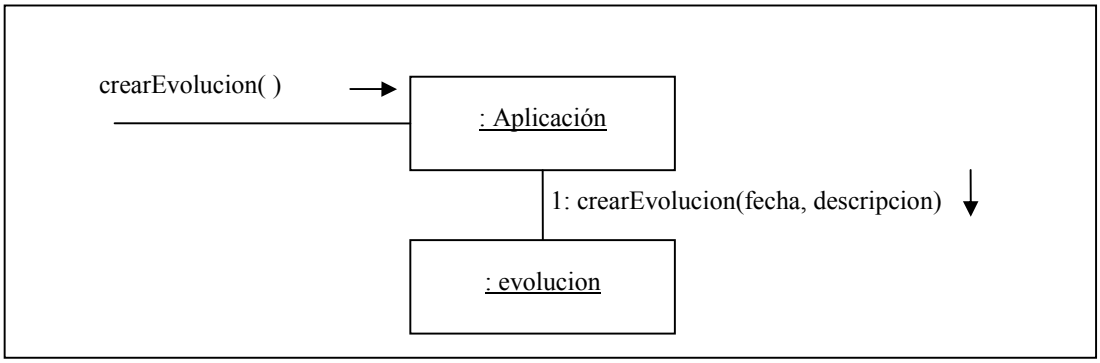
CASO DE USO: Asignar cita

Operación: crearCita

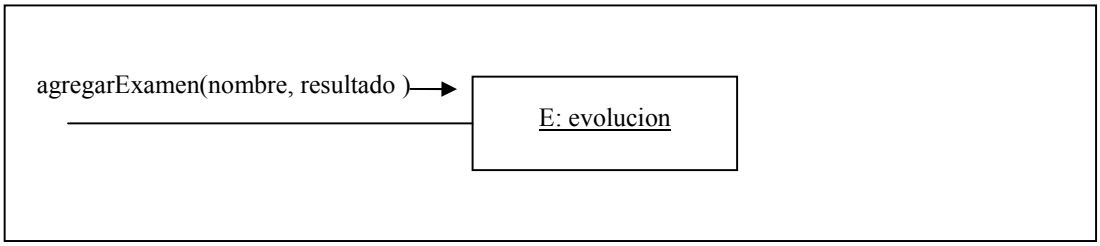


CASO DE USO: Determinar evolución de la enfermedad

Operación: agregarEvolucion

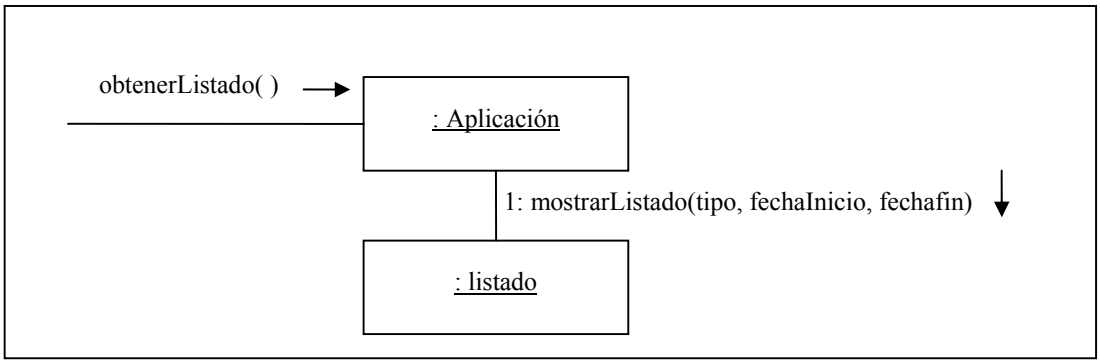


Operación: agregarExamen



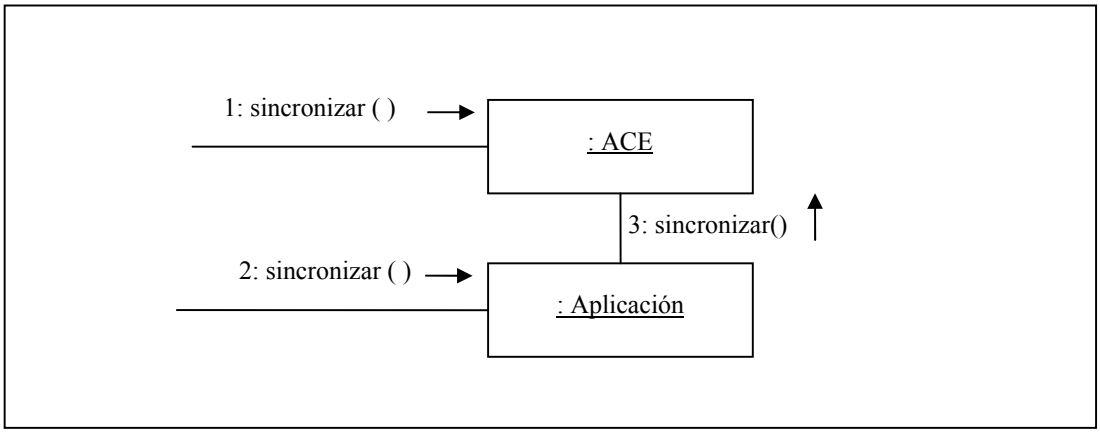
CASO DE USO: Obtener listado

Operación: obtenerListado



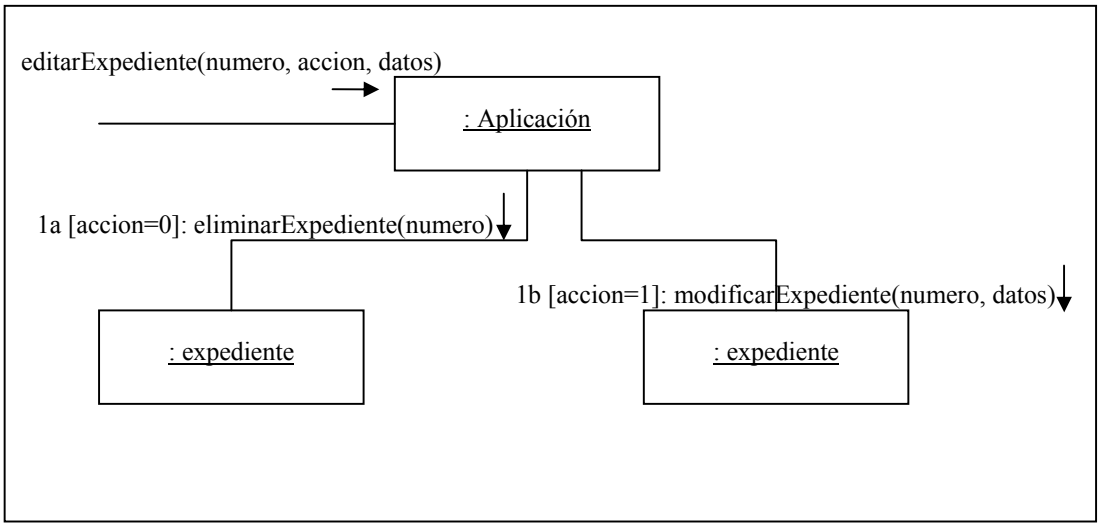
CASO DE USO: Sincronizar datos

Operación: sincronizar

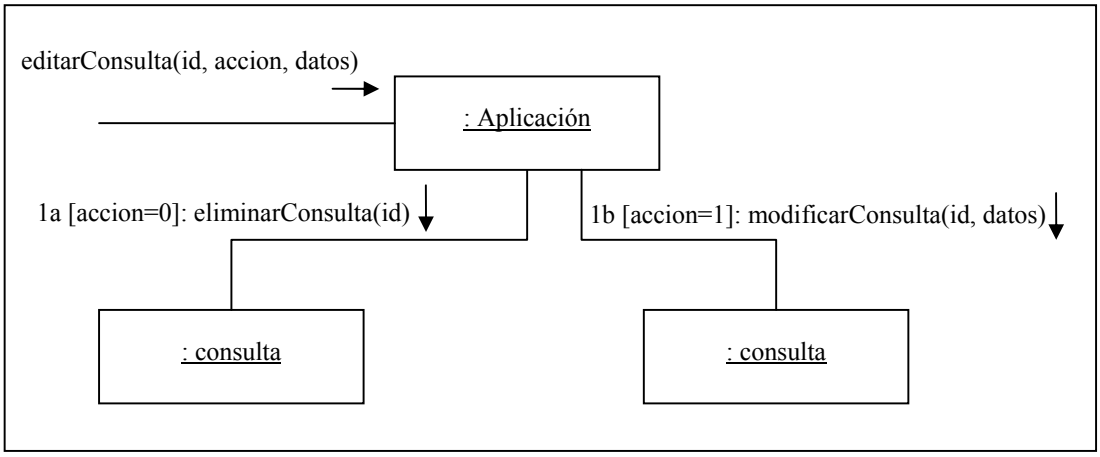


CASO DE USO: Editar

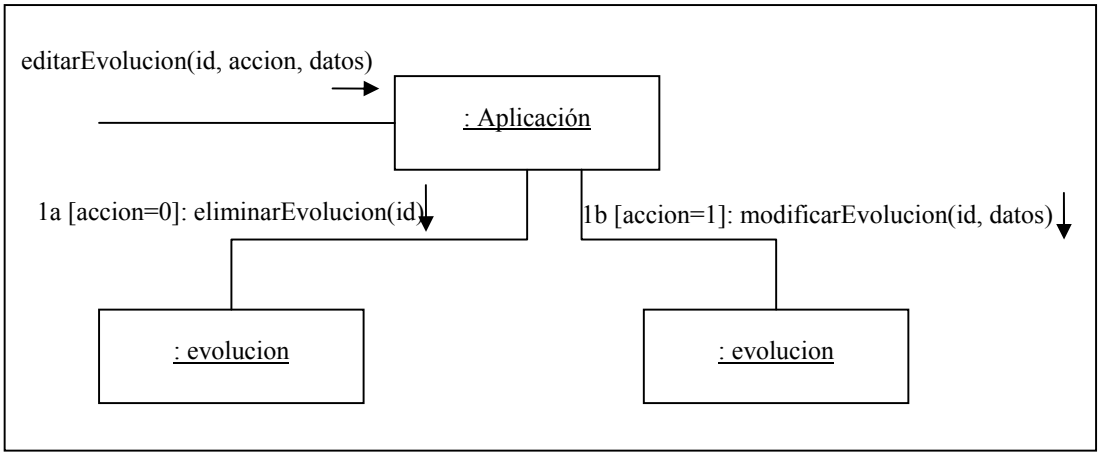
Operación: editarExpediente



Operación: editarConsulta



Operación: editarEvolucion



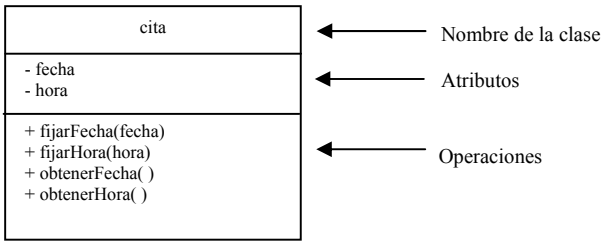
DETALLE DE LA TECNICA

DIAGRAMA DE CLASES

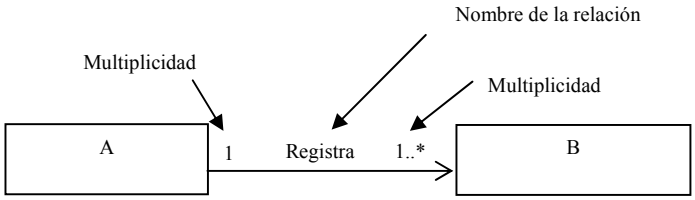
Un diagrama de clases presenta las clases del sistema o aplicación con sus relaciones. La definición de clase incluye atributos y operaciones.

SIMBOLOGIA

Clase



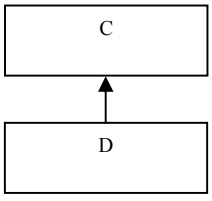
Relaciones estructurales



Un objeto de la clase A registra 1 o más objetos de la clase B

Un objeto de la clase B es registrado por 1 objeto de la clase A

Relaciones de herencia



La clase D hereda los atributos y operaciones de la clase C

2.4.3. Diagrama de Clases

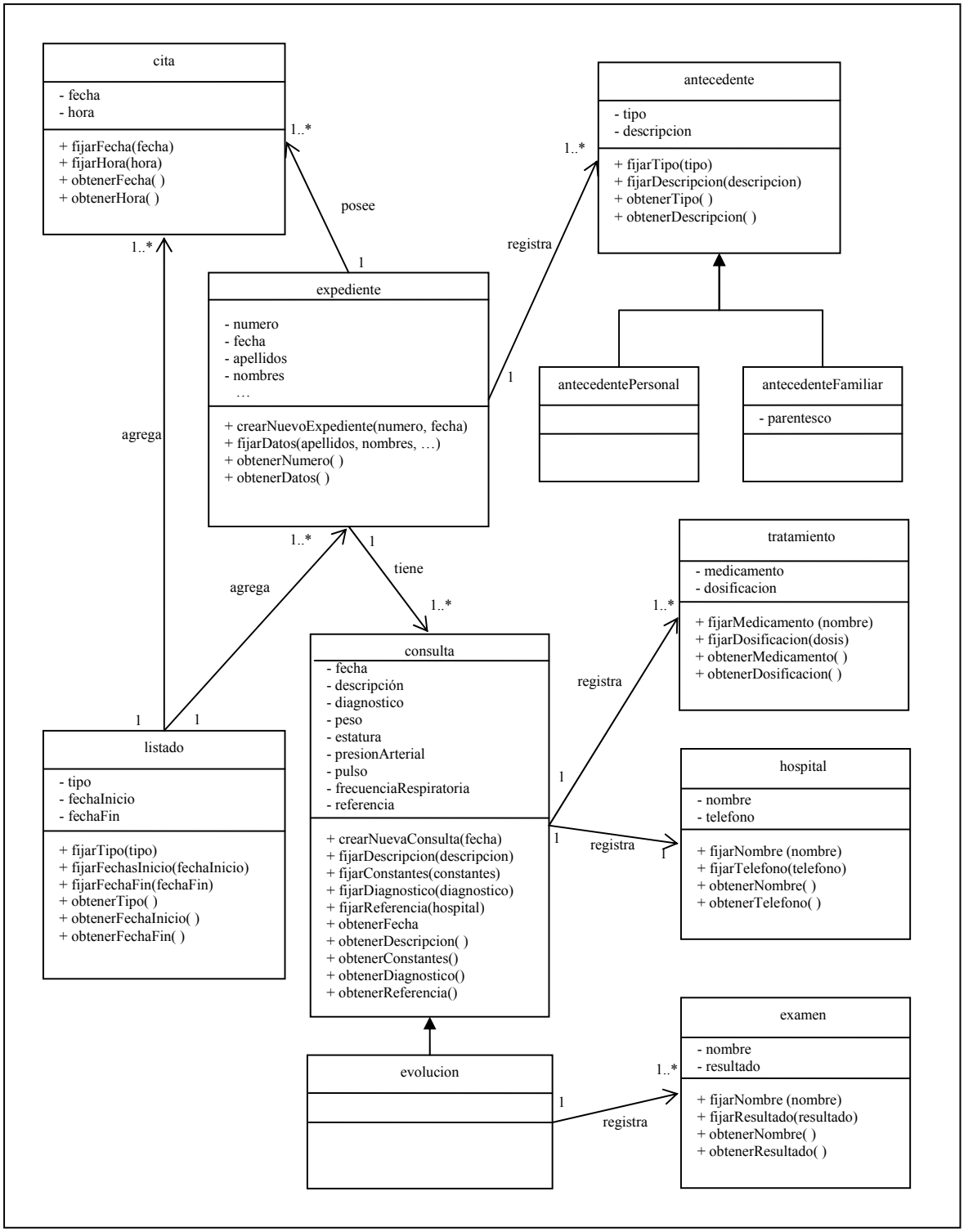


Figura 2.4 Diagrama de clases

2.4.4. Diseño de la Base de Datos

Normalización

La normalización es un proceso basado en reglas que permite convertir una relación en varias sub-relaciones. El proceso consiste en ir aplicando de forma sucesiva las diferentes formas normales, en la Primera Forma Normal (1FN) se eliminan los grupos repetitivos, los atributos han de ser atómicos; en la Segunda Forma Normal (2FN) debe haber dependencia completa, los atributos no principales dependen de forma completa de la clave principal; en la Tercera Forma Normal (3FN), los atributos no principales deben depender de la clave primaria de forma no transitiva, es decir depender directamente, y no a través de otro atributo.

La normalización no solo garantiza una mejor gestión de los datos, sino que provee una estructura que facilita el proceso de diseño físico de la base de datos.

A continuación se muestra el proceso de normalización para la base de datos de la aplicación prototipo.

Relación encontrada:

ENTIDAD (NUMERO, FECHA_EXPEDIENTE, NOMBRE, DIRECCION, TELEFONO, CONSULTA, EVOLUCION, DIAGNOSTICO, ANTECEDENTES, TRATAMIENTO, EXAMEN, CONSTANTES, REFERENCIA, HOSPITAL, CITA)

Aplicando 1FN

El dominio de CONSULTA, EVOLUCION, ANTECEDENTES, TRATAMIENTO, EXAMEN, HOSPITAL y CITA solo deben tener valores atómicos.

ENTIDAD (NUMERO, FECHA_EXPEDIENTE, NOMBRE, DIRECCION, TELEFONO, FECHA_CONSULTA, DESCRIPCION_CONSULTA, FECHA_EVOLUCION, DESCRIPCION_EVOLUCION, DIAGNOSTICO, TIPO_ANTECEDENTES, DESCRIPCION_ANTECEDENTES, MEDICAMENTO, DOSIFICACION,

NOMBRE_EXAMEN, RESULTADO_EXAMEN, CONSTANTES, REFERENCIA, NOMBRE_HOSPITAL, TELEFONO_HOSPITAL, FECHA_CITA, HORA_CITA)

Aplicando 2FN

Se puede ver que los siguientes atributos no dependen funcionalmente de manera total de la clave NUMERO:

FECHA_CONSULTA, DESCRIPCION_CONSULTA, FECHA_EVOLUCION, DESCRIPCION_EVOLUCION, DIAGNOSTICO, TIPO_ANTECEDENTES, DESCRIPCION_ANTECEDENTES, MEDICAMENTO, DOSIFICACION, NOMBRE_EXAMEN, RESULTADO_EXAMEN, CONSTANTES, REFERENCIA, NOMBRE_HOSPITAL, TELEFONO_HOSPITAL, FECHA_CITA, HORA_CITA

ENTIDAD 1 (NUMERO, NOMBRE, DIRECCION, TELEFONO)

ENTIDAD 2 (ID, FECHA_CONSULTA, DESCRIPCION_CONSULTA, DIAGNOSTICO, CONSTANTES, REFERENCIA)

ENTIDAD 3 (ID, FECHA_EVOLUCION, DESCRIPCION_EVOLUCION, CONSTANTES, REFERENCIA)

ENTIDAD 4 (ID, TIPO_ANTECEDENTES, DESCRIPCION_ANTECEDENTES)

ENTIDAD 5 (ID, MEDICAMENTO, DOSIFICACION)

ENTIDAD 6 (ID, NOMBRE_EXAMEN, RESULTADO_EXAMEN)

ENTIDAD 7 (ID, NOMBRE_HOSPITAL, TELEFONO_HOSPITAL)

ENTIDAD 8 (ID, FECHA_CITA, HORA_CITA)

Aplicando 3FN

Se puede apreciar que en las entidades anteriores están en 3FN ya que no existen dependencias transitivas entre sus atributos.

Entidades resultantes después de la normalización:

EXPEDIENTE (número, nombre, dirección, teléfono)

CONSULTA (id, fecha, descripción, diagnóstico, constantes, referencia)

EVOLUCION (id, fecha, descripción, diagnóstico, constantes, referencia)

ANTECEDENTES (id, tipo, descripción)

TRATAMIENTO (id, medicamento, dosificación)

EXAMEN (id, nombre, resultado)

HOSPITAL (id, nombre, teléfono)

CITA (id, fecha, hora)

DETALLE DE LA TECNICA

DIAGRAMA ENTIDAD-RELACION

Los diagramas entidad-relación o diagramas E-R representan los conceptos (entidades) u objetos del sistema o aplicación y sus relaciones. Estos diagramas facilitan el diseño de las bases de datos.

SIMBOLOGIA

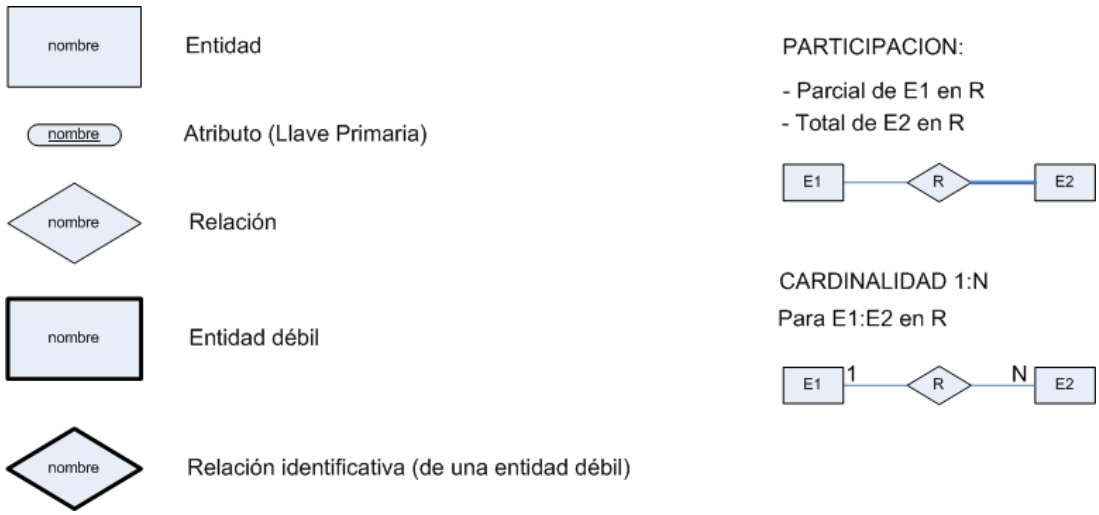


Diagrama Entidad-Relación

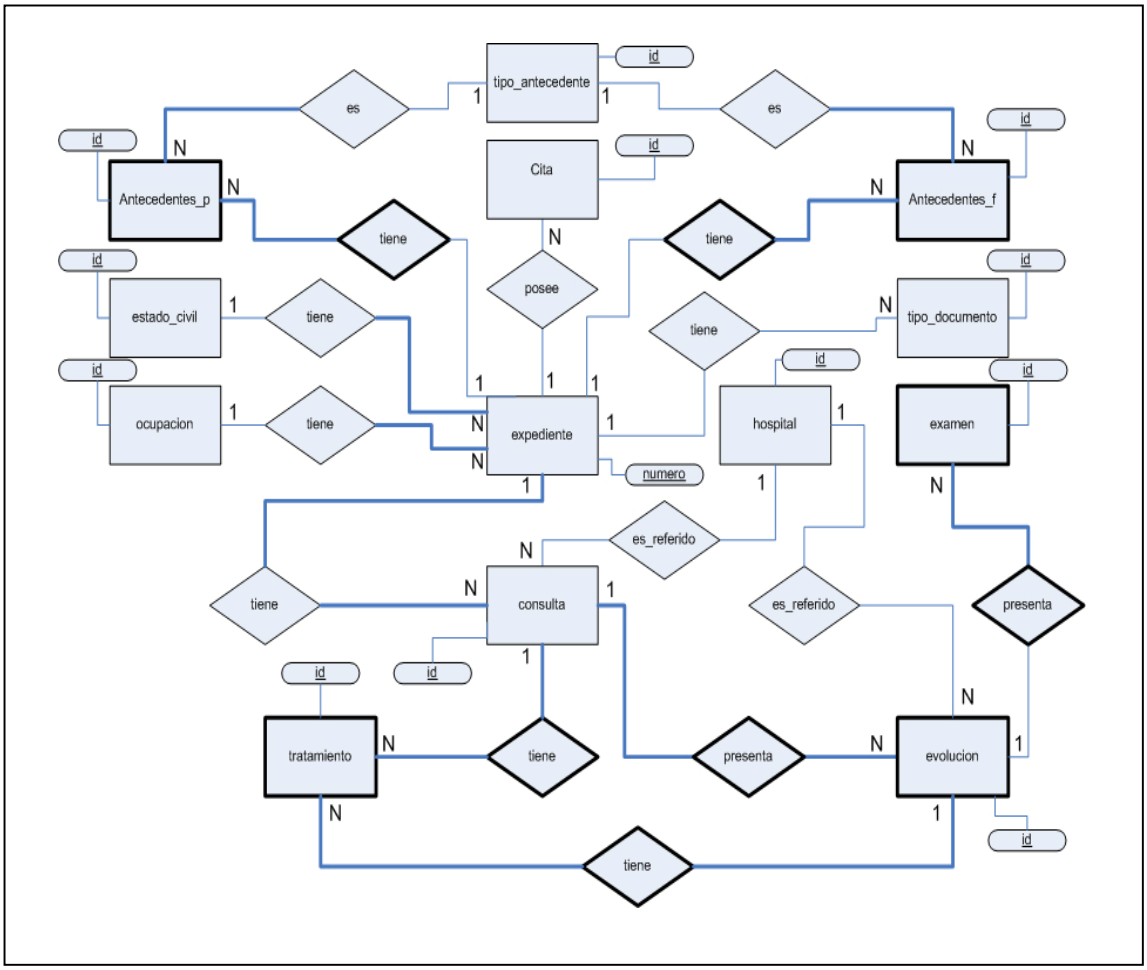


Figura 2.5 Diagrama Entidad-Relación

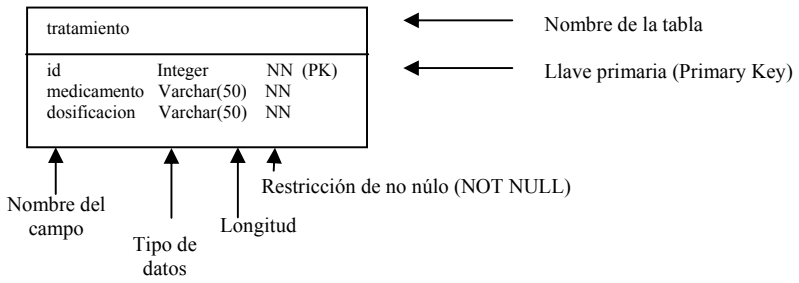
DETALLE DE LA TECNICA

MODELO FISICO DE LA BASE DE DATOS

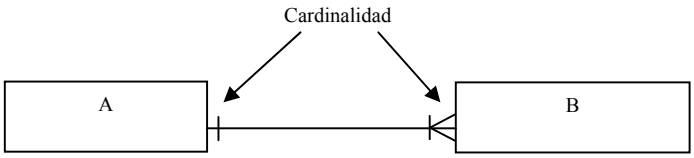
Es un diagrama que representa la base de datos, incluye las tablas, la longitud y tipo de datos de los campos, así como las relaciones entre las tablas.

SIMBOLOGIA

Tabla



Relaciones



Un registro de la tabla A se relaciona con muchos registros de la tabla B

Un registro de la tabla B se relaciona con un registro de la tabla A

Modelo Físico de la base de datos

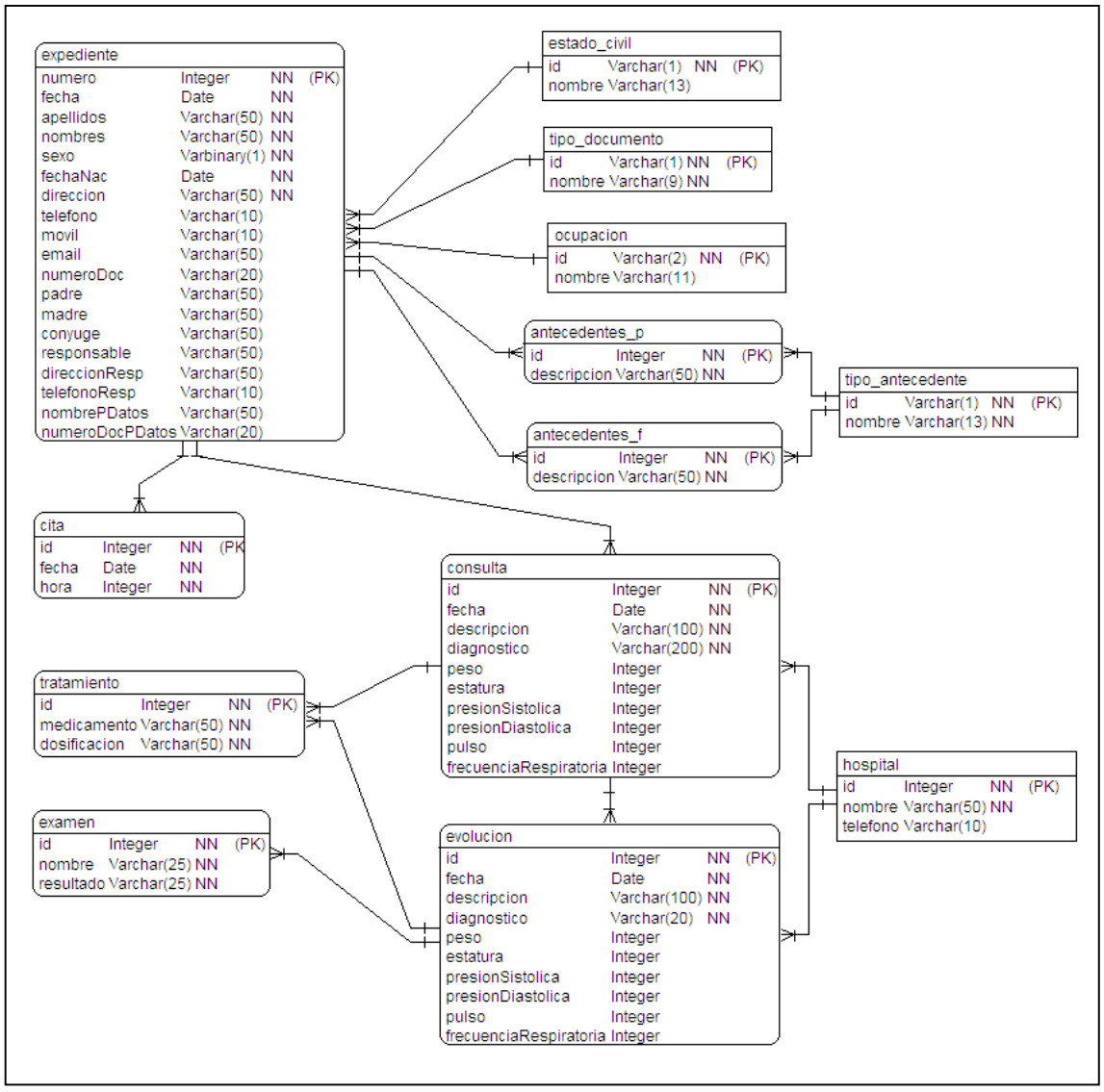


Figura 2.6 Modelo fisico de la base de datos

2.4.5. Arquitectura de la Aplicación

A continuación se presenta un diagrama que, si bien no con notación UML, representa la arquitectura de la aplicación.

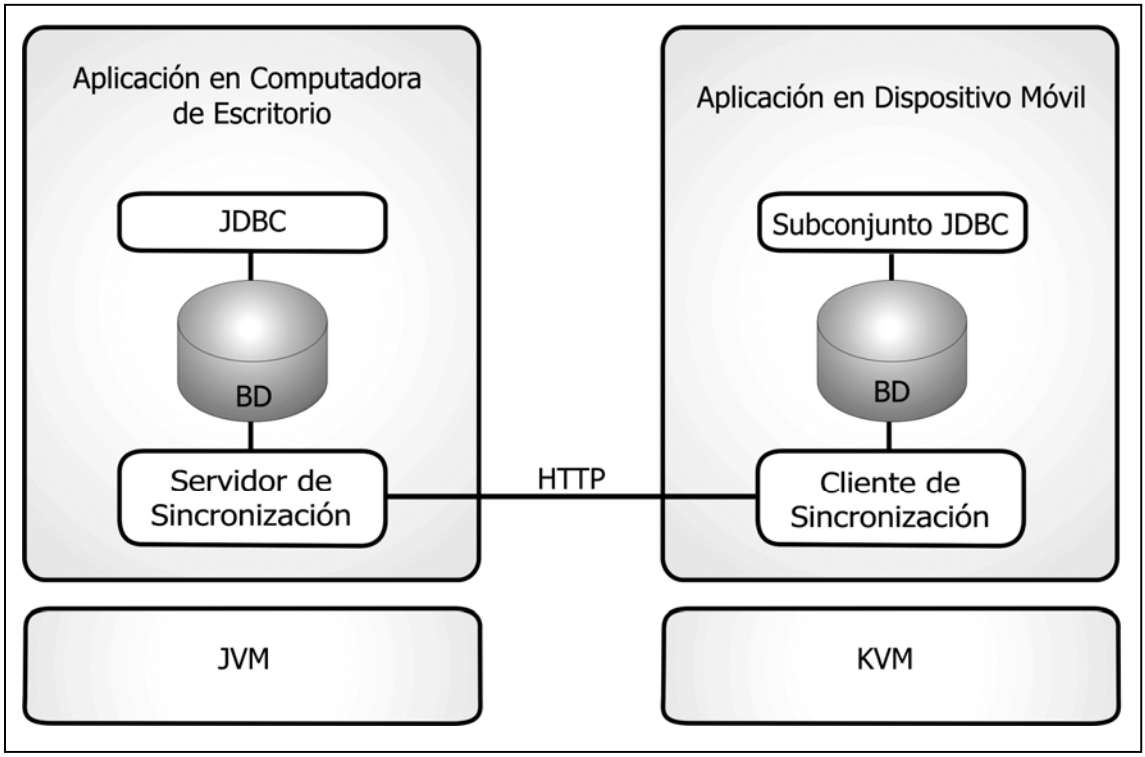


Figura 2.7 Arquitectura de la aplicación de fichas médicas

Descripción de elementos

Aplicación en computadora de escritorio

- JDBC

  Permite la ejecución de operaciones sobre bases de datos.

- BD

  Base de datos residente en la computadora de escritorio.

- Servidor de sincronización

  Escucha las peticiones y realiza la sincronización de datos provenientes de la aplicación en el dispositivo móvil.

- JVM

  Máquina virtual sobre la que se ejecuta la aplicación de escritorio

Aplicación en Dispositivo Móvil

- Subconjunto JDBC

  Es un conjunto reducido de instrucciones, que permite la ejecución de operaciones sobre bases de datos.

- BD

  Base de datos residente en el dispositivo móvil.

- Cliente de sincronización

  Realiza las peticiones de sincronización de datos.

- KVM

  Máquina virtual sobre la que se ejecuta la aplicación en el dispositivo móvil.

HTTP

Es el protocolo que permite la sincronización entre las bases de datos de ambas aplicaciones, empleando el servidor y cliente de sincronización

2.5. Diseño de Datos

A continuación se presenta la especificación de las clases que dan soporte a la aplicación.

Clase DBAccess

Clase que permite acceder a la base de datos.

Atributos	
(package private) static Connection	**c** Para establecer la conexión con la base de datos
(package private) static java.lang.String	**m_dbname** Especifica el nombre de la base de datos
(package private) static Statement	**s** Para las sentencias que se ejecutarán sobre la base de datos

Constructor
DBAccess()

Métodos	
boolean	**borrarDB**() Sirve para borrar la base de datos
void	**conectar**() Realiza la conexión a la base de datos
boolean	**crearTablas**() Sirve para crear las tablas en la base de datos
boolean	**dbExiste**() Verifica si existe la base de datos
void	**desconectar**() Se desconecta de la base de datos

ResultSet	**execute**(java.lang.String sql) Para ejecutar una sentencia sql tipo SELECT sobre la base de datos
int	**executeUpdate**(java.lang.String sql) Para ejecutar una sentencia sql sobre la base de datos
int	**executeUpdateValue**(java.lang.String sql) Para ejecutar una sentencia sql sobre la base de datos
ResultSet	**getTables**() Verifica si la base de datos tiene tablas

Clase Expediente

Clase que provee la funcionalidad para registrar los expedientes de los pacientes.

Atributos	
private java.lang.String	**apellidos** Apellidos del paciente
private java.lang.String	**conyuge** Nombres del cónyuge del paciente
private java.lang.String	**direccion** Dirección del paciente
private java.lang.String	**direccionResp** Dirección del responsable del paciente
private java.lang.String	**email** Dirección de correo electrónico del paciente
private java.util.Date	**fecha** Fecha de creación del expediente
private java.util.Date	**fechaNac** Fecha de nacimiento del paciente
private java.lang.St	**id ESTADO CIVIL**

ring	Estado civil del paciente
private java.lang.String	**id OCUPACION** Ocupación del paciente
private java.lang.String	**id PARENTESCO PDatos** Parentesco de la persona que proporciona los datos del paciente
private java.lang.String	**id TIPO DOCUMENTO paciente** Tipo de documento del paciente
private java.lang.String	**id TIPO DOCUMENTO PDatos** Tipo de documento de la persona que proporciona los datos del paciente
private DBAccess	**m dbaccess** Permite la conexión con la base de datos
private java.lang.String	**madre** Nombre del padre del paciente
private java.lang.String	**movil** Número de teléfono móvil del paciente
private java.lang.String	**nombrePDatos** Nombre de la persona que proporciona los datos del paciente
private java.lang.String	**nombres** Nombres del paciente
private int	**numero** Número de expediente
private java.lang.String	**numeroDoc** Número de documento del paciente
private java.lang.String	**numeroDocPDatos** Número de documento de la persona que proporciona los datos del paciente
private java.lang.St	**padre**

ring	Nombre de la madre del paciente
private java.lang.String	**responsable** Nombres del responsable del paciente
private java.lang.String	**sexo** Sexo del paciente
private java.lang.String	**telefono** Número de teléfono fijo del paciente
private java.lang.String	**telefonoResp** Número telefónico del responsable del paciente

Constructores

Expediente()
Constructor sin parámetros

Expediente(java.util.Date m_fecha, java.lang.String m_apellidos, java.lang.String m nombres, java.lang.String m sexo, java.util.Date m_fechaNac, java.lang.String m_id_ESTADO_CIVIL, java.lang.String m_direccion, java.lang.String m_telefono, java.lang.String m_movil, java.lang.String m_email, java.lang.String m_id_TIPO_DOCUMENTO_paciente, java.lang.String m numeroDoc, java.lang.String m id OCUPACION, java.lang.String m_padre, java.lang.String m_madre, java.lang.String m_conyuge, java.lang.String m_responsable, java.lang.String m_direccionResp, java.lang.String m_telefonoResp, java.lang.String m_nombrePDatos, java.lang.String m_id_PARENTESCO_PDatos, java.lang.String m id TIPO DOCUMENTO PDatos, java.lang.String m_numeroDocPDatos)
Constructor que recibe como parámetros los datos del paciente

Expediente(int m_numero)
Constructor que recibe como parámetro el número de expediente

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla expediente
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial)

	entre las tablas
boolean	**eliminar**() Elimina un registro de la tabla expediente
void	**fijarApellidos**(java.lang.String m_apellidos) Fija los apellidos del paciente
void	**fijarConyuge**(java.lang.String m_conyuge) Fija el nombre del cónyuge del paciente
void	**fijarDireccion**(java.lang.String m_direccion) Fija la dirección del paciente
void	**fijarDireccionResp**(java.lang.String m_direccionResp) Fija la dirección del responsable del paciente
void	**fijarEmail**(java.lang.String m_email) Fija la dirección de correo electrónico del paciente
void	**fijarFecha**(java.util.Date m_fecha) Fija la fecha de creación del expediente
void	**fijarFechaNac**(java.util.Date m_fechaNac) Fija la fecha de nacimiento del paciente
void	**fijarId_ESTADO_CIVIL**(java.lang.String m_id_ESTADO_CIVIL) Fija el estado civil del paciente
void	**fijarId_OCUPACION**(java.lang.String m_id_OCUPACION) Fija la ocupación del paciente
void	**fijarId_PARENTESCO_PDatos**(java.lang.String m_id_PARENTESCO_PDatos) Fija el parentesco de la persona que proporciona los datos del paciente
void	**fijarId_TIPO_DOCUMENTO_paciente**(java.lang.String m_id_TIPO_DOCUMENTO paciente) Fija el tipo de documento del paciente
void	**fijarId_TIPO_DOCUMENTO_PDatos**(java.lang.String m_id_TIPO_DOCUMENTO_PDatos) Fija el tipo de documento de la persona que proporciona los datos

	del paciente
void	**fijarMadre**(java.lang.String m_madre) Fija el nombre de la madre del paciente
void	**fijarMovil**(java.lang.String m_movil) Fija el número de teléfono móvil del paciente
void	**fijarNombrePDatos**(java.lang.String m_nombrePDatos) Fija el nombre de la persona que proporciona los datos del paciente
void	**fijarNombres**(java.lang.String m_nombres) Fija los nombres del paciente
void	**fijarNumero**(int m_numero) Fija el número de expediente
void	**fijarNumeroDoc**(java.lang.String m_numeroDoc) Fija el número de documento del paciente
void	**fijarNumeroDocPDatos**(java.lang.String m_numeroDocPDatos) Fija el número de documento de la persona que proporciona los datos del paciente
void	**fijarPadre**(java.lang.String m_padre) Fija el nombre del padre del paciente
void	**fijarResponsable**(java.lang.String m_responsable) Fija el nombre del responsable del paciente
void	**fijarSexo**(java.lang.String m_sexo) Fija el sexo
void	**fijarTelefono**(java.lang.String m_telefono) Fija el número de teléfono fijo del paciente
void	**fijarTelefonoResp**(java.lang.String m_telefonoResp) Fija el número de teléfono del responsable del paciente
int	**guardar**() Almacena los datos del paciente en la tabla expediente
java.lang.S	**obtenerApellidos**()

tring	Devuelve los apellidos del paciente
java.lang.String	**obtenerConyuge**() Devuelve el nombre del cónyuge del paciente
java.lang.String	**obtenerDireccion**() Devuelve la dirección del paciente
java.lang.String	**obtenerDireccionResp**() Devuelve la dirección del responsable del paciente
java.lang.String	**obtenerEmail**() Devuelve la dirección de correo electrónico del paciente
java.util.Date	**obtenerFecha**() Devuelve la fecha de creación del expediente
java.util.Date	**obtenerFechaNac**() Devuelve la fecha de nacimiento del paciente
java.lang.String	**obtenerId_ESTADO_CIVIL**() Devuelve el estado civil del paciente
java.lang.String	**obtenerId_OCUPACION**() Devuelve la ocupación del paciente
java.lang.String	**obtenerId_PARENTESCO_PDatos**() Devuelve el parentesco de la persona que proporciona los datos del paciente
java.lang.String	**obtenerId_TIPO_DOCUMENTO_paciente**() Devuelve el tipo de documento del paciente
java.lang.String	**obtenerId_TIPO_DOCUMENTO_PDatos**() Devuelve el tipo de documento de la persona que proporciona los datos del paciente
java.lang.String	**obtenerMadre**() Devuelve el nombre de la madre del paciente
java.lang.String	**obtenerMovil**() Devuelve el número de teléfono móvil del paciente

java.lang.String	**obtenerNombrePDatos**() Devuelve el nombre de la persona que proporciona los datos del paciente
java.lang.String	**obtenerNombres**() Devuelve los nombres del paciente
int	**obtenerNumero**() Devuelve el número de expediente
java.lang.String	**obtenerNumeroDoc**() Devuelve el número de documento del paciente
java.lang.String	**obtenerNumeroDocPDatos**() Devuelve el número de documento de la persona que proporciona los datos del paciente
java.lang.String	**obtenerPadre**() Devuelve el nombre del padre del paciente
java.lang.String	**obtenerResponsable**() Devuelve el nombre del responsable del paciente
java.lang.String	**obtenerSexo**() Devuelve el sexo
java.lang.String	**obtenerTelefono**() Devuelve el número de teléfono fijo del paciente
java.lang.String	**obtenerTelefonoResp**() Devuelve el número de teléfono del responsable del paciente
boolean	**recuperar**() Recupera un registro de la tabla expediente

Clase Consulta

Clase que provee la funcionalidad para registrar las consultas de los pacientes.

Atributos

protected java.lang.String	**descripcion** Descripción del motivo de la consulta
protected java.lang.String	**diagnostico** Descripción del diagnóstico realizado
protected int	**estatura** Estatura del paciente
protected java.util.Date	**fecha** Fecha de la consulta
protected int	**frecuenciaRespiratoria** Frecuencia respiratoria del paciente
protected int	**id** Identificador de la consulta
protected int	**id HOSPITAL** Identificador del hospital(al que se pueda referir un paciente)
protected DBAccess	**m dbaccess** Permite la conexión con la base de datos
protected int	**numero EXPEDIENTE** Número de expediente del paciente
protected int	**peso** Peso del paciente
protected int	**presionDiastolica** Presión diastólica del paciente
protected int	**presionSistolica** Presión sistólica del paciente
protected int	**pulso** Pulso del paciente

Constructores

Consulta() Constructor sin parámetros
Consulta(int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el número de expediente
Consulta(int m_numero_EXPEDIENTE, java.util.Date m_fecha, java.lang.String m_descripcion, java.lang.String m_diagnostico, int m_peso, int m_estatura, int m_presionSistolica, int m presionDiastolica, int m pulso, int m frecuenciaRespiratoria, int m_id_HOSPITAL) Constructor que recibe como parámetros los datos de la consulta
Consulta(int m_id, int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el identificador de la consulta y el número de expediente

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
protected boolean	**consistenciaOK**() Necesario para mantener la consistencia (integridad referencial) entre consulta, expediente y hospital
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_numero_EXPEDIENTE) Elimina los registros de la tabla que coincidan con el número de expediente
void	**fijarDescripcion**(java.lang.String m_descripcion) Fija la descripción del motivo de la consulta
void	**fijarDiagnostico**(java.lang.String m_diagnostico) Fija el diagnóstico realizado

void	**fijarEstatura**(int m_estatura) Fija la estatura del paciente
void	**fijarFecha**(java.util.Date m_fecha) Fija la fecha de consulta
void	**fijarFrecuenciaRespiratoria**(int m_frecuenciaRespiratoria) Fija la frecuencia respiratoria del paciente
void	**fijarId_HOSPITAL**(int m_id_HOSPITAL) Fija la el identificador del hospital (al que se refiere un pacienre)
void	**fijarId**(int m_id) Fija el id de la consulta
void	**fijarNumero_EXPEDIENTE**(int m_numero_EXPEDIENTE) Fija el número de expediente
void	**fijarPeso**(int m_peso) Fija el peso del paciente
void	**fijarPresionDiastolica**(int m_presionDiastolica) Fija la presión diastólica del paciente
void	**fijarPresionSistolica**(int m_presionSistolica) Fija la presión sistólica del paciente
void	**fijarPulso**(int m_pulso) Fija el pulso del paciente
boolean	**guardar**() Almacena los datos de la consulta en la tabla correspondiente
java.lang.String	**obtenerDescripcion**() Devuelve la descripción del motivo de la consulta
java.lang.String	**obtenerDiagnostico**() Devuelve el diagnóstico realizado
int	**obtenerEstatura**() Devuelve la estatura del paciente

java.util.Date	**obtenerFecha**() Devuelve la fecha de consulta
int	**obtenerFrecuenciaRespiratoria**() Devuelve la frecuencia respiratoria del paciente
int	**obtenerId HOSPITAL**() Devuelve la el identificador del hospital (al que se refiere un pacienre)
int	**obtenerId**() Devuelve el id de la consulta
int	**obtenerNumero EXPEDIENTE**() Devuelve el número de expediente
int	**obtenerPeso**() Devuelve el peso del paciente
int	**obtenerPresionDiastolica**() Devuelve la presión diastólica del paciente
int	**obtenerPresionSistolica**() Devuelve la presión sistólica del paciente
int	**obtenerPulso**(int m_pulso) Devuelve el pulso del paciente
boolean	**recuperar**() Recupera un registro de la tabla

Clase Evolucion

Clase que provee la funcionalidad para registrar las evoluciones de los pacientes.

Atributos	
private int	**id CONSULTA** Identificador de la consulta

Atributos heredados de la clase Consulta

descripcion, diagnostico, estatura, fecha, frecuenciaRespiratoria, id, id HOSPITAL, m dbaccess, numero EXPEDIENTE, peso, presionDiastolica, presionSistolica, pulso

Constructores

Evolucion()
Constructor sin parámetros

Evolucion(int m_id_CONSULTA)
Constructor que recibe como parámetro el identificador de la consulta

Evolucion(int m_id_CONSULTA, java.util.Date m_fecha, java.lang.String m descripcion, java.lang.String m diagnostico, int m_peso, int m_estatura, int m_presionSistolica, int m_presionDiastolica, int m_pulso, int m_frecuenciaRespiratoria, int m id HOSPITAL)
Constructor que recibe como parámetros los datos de la evolución

Evolucion(int m_id, int m_id_CONSULTA)
Constructor que recibe como parámetro los identificadores de evolución y consulta

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
protected boolean	**consistenciaOK**() Necesario para mantener la consistencia (integridad referencial) entre evolucion, consulta y hospital
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_id_CONSULTA) Elimina los registros de la tabla que coincidan con el

	identificador de consulta
void	**fijarId CONSULTA**(int m_id_CONSULTA) Fija el id de la consulta
boolean	**guardar**() Almacena los datos de la evolución en la tabla correspondiente
int	**obtenerId CONSULTA**() Devuleve el id de la consulta
boolean	**recuperar**() Recupera un registro de la tabla
Métodos heredados de la clase Consulta	
fijarDescripcion, fijarDiagnostico, fijarEstatura, fijarFecha, fijarFrecuenciaRespiratoria, fijarId HOSPITAL, fijarId, fijarNumero EXPEDIENTE, fijarPeso, fijarPresionDiastolica, fijarPresionSistolica, fijarPulso, obtenerDescripcion, obtenerDiagnostico, obtenerEstatura, obtenerFecha, obtenerFrecuenciaRespiratoria, obtenerId HOSPITAL, obtenerId, obtenerNumero EXPEDIENTE, obtenerPeso, obtenerPresionDiastolica, obtenerPresionSistolica, obtenerPulso	

Clase Tratamiento

Clase que provee la funcionalidad para registrar los tratamientos de los pacientes.

Atributos	
private int	**b_PADRE** Bandera para identificar la tabla padre entre consulta y evolución
private java.lang.String	**dosificacion** Descripción de la dosificación
private int	**id** Identificador del tratamiento

private int	**id_PADRE** Identificador de consulta o evolución
protected DBAccess	**m_dbaccess** Permite la conexión con la base de datos
private java.lang.String	**medicamento** Descripción del medicamento

Constructores

Tratamiento()
Constructor sin parámetros

Tratamiento(int m_b_PADRE, int m_id_PADRE)
Constructor que recibe como parámetro el identificador del consulta o evolución

Tratamiento(int m_id, int m_b_PADRE, int m_id_PADRE)
Constructor que recibe como parámetro los identificadores de tratamiento y, consulta o evolución

Tratamiento(int m_b_PADRE, int m_id_PADRE, java.lang.String m_medicamento, java.lang.String m_dosificacion)
Constructor que recibe como parámetros los datos del tratamiento

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
private boolean	**consistenciaOK**() Necesario para mantener la consistencia (integridad referencial) entre tratamiento, consulta y evolucion
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_b_PADRE, int m_id_PADRE)

	Elimina los registros de la tabla que coincidan con el identificador de consulta o evolucion
void	**fijarB_PADRE**(int m_b_PADRE) Fija la bandera para identificar si el padre es consulta o evolución
void	**fijarDosificacion**(java.lang.String m_dosificacion) Fija la descripción de la dosificación del medicamento prescrito
void	**fijarId_PADRE**(int m_id_PADRE) Fija el id de la consulta o evolución
void	**fijarId**(int m_id) Fija el id del tratamiento
void	**fijarMedicamento**(java.lang.String m_medicamento) Fija la descripción del medicamento
boolean	**guardar**() Almacena los datos del tratamiento prescrito al paciente
int	**obtenerB_PADRE**() Devuelve la bandera para identificar si el padre es consulta o evolución
java.lang.String	**obtenerDosificacion**() Devuelve la descripción de la dosificación del medicamento prescrito
int	**obtenerId_PADRE**() Devuelve el identificador de la consulta o evolución
int	**obtenerId**() Devuelve el id del tratamiento
java.lang.String	**obtenerMedicamento**() Devuelve la descripción del medicamento
boolean	**recuperar**() Recupera un registro de la tabla

private boolean	**registro**(java.lang.String query) Comprueba que exista el identificador en la tabla padre consulta o evolución

Clase Examen

Clase que provee la funcionalidad para registrar los exámenes de los pacientes.

Atributos

private int	**id** Identificador del examen
private int	**id_EVOLUCION** Identificador de la evolución
protected DBAccess	**m_dbaccess** Permite la conexión con la base de datos
private java.lang.String	**nombre** Descripción del examen
private java.lang.String	**resultado** Descripción del resultado

Constructores

Examen() Constructor sin parámetros
Examen(int m_id_EVOLUCION) Constructor que recibe como parámetro el identificador de la evolución
Examen(int m_id, int m_id_EVOLUCION) Constructor que recibe como parámetro los identificadores de examen y evolución
Examen(int m_id_EVOLUCION, java.lang.String m_nombre, java.lang.String m resultado) Constructor que recibe como parámetros los datos del examen

Métodos	
boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
private boolean	**consistenciaOK**() Necesario para mantener la consistencia (integridad referencial) entre examen y evolucion
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_id_EVOLUCION) Elimina los registros de la tabla que coincidan con el identificador de evolucion
void	**fijarId_EVOLUCION**(int m_id_EVOLUCION) Fija el id de la evolución
void	**fijarId**(int m_id) Fija el id del examen
void	**fijarNombre**(java.lang.String m_nombre) Fija la descripción del examen
void	**fijarResultado**(java.lang.String m_resultado) Fija la descripción del resultado del examen
boolean	**guardar**() Almacena los datos del examen
int	**obtenerId_EVOLUCION**() Devuelve el identificador de la evolución
int	**obtenerId**() Devuelve el id del examen
java.lang.String	**obtenerNombre**()

	Devuelve la descripción del examen
java.lang.String	**obtenerResultado**() Devuelve la descripción del resultado del examen
boolean	**recuperar**() Recupera un registro de la tabla

Clase Cita

Clase que provee la funcionalidad para registrar las citas de los pacientes.

Atributos

private java.util.Date	**fecha** Fecha de la cita
private int	**hora** Hora de la cita
private int	**id** Identificador de la consulta
protected DBAccess	**m_dbaccess** Permite la conexión con la base de datos
private int	**minutos** Minutos de la cita
private int	**numero_EXPEDIENTE** Número de expediente del paciente

Constructores

Cita() Constructor sin parámetros
Cita(int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el número de expediente

Cita(int m_numero_EXPEDIENTE, java.util.Date m_fecha, int m_hora, int m_minutos) Constructor que recibe como parámetros los datos de la cita
Cita(int m_id, int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el identificador de la cita y el número de expediente

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
protected boolean	**consistenciaOK**() Necesario para mantener la consistencia (integridad referencial) entre cita y expediente
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_numero_EXPEDIENTE) Elimina los registros de la tabla que coincidan con el número de expediente
void	**fijarFecha**(java.util.Date m_fecha) Fija la fecha de cita
void	**fijarHora**(int m_hora, int m_minutos) Fija la hora de la cita
void	**fijarId**(int m_id) Fija el id de la cita
void	**fijarNumero_EXPEDIENTE**(int m_numero_EXPEDIENTE) Fija el número de expediente
boolean	**guardar**() Almacena los datos de la cita en la tabla correspondiente

java.util.Date	**obtenerFecha**() Devuelve la fecha de cita
java.lang.String	**obtenerHora**() Devuelve la hora de la cita
int	**obtenerId**() Devuelve el id de la cita
int	**obtenerNumero_EXPEDIENTE**() Devuelve el número de expediente
boolean	**recuperar**() Recupera un registro de la tabla

Clase Antecedentes_p

Clase que provee la funcionalidad para registrar los antecedentes personales de los pacientes.

Atributos	
protected java.lang.String	**descripcion** Descripción del antecedente
protected int	**id** Identificador del antecedente
protected java.lang.String	**id_TIPO_ANTECEDENTE** Identificador del tipo de antecedente
protected DBAccess	**m_dbaccess** Permite la conexión con la base de datos
protected int	**numero_EXPEDIENTE** Número de expediente del paciente
Constructores	

Antecedentes_p() Constructor sin parámetros
Antecedentes_p(int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el número de expediente
Antecedentes_p(int m_id, int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el identificador del antecedente y el número de expediente
Antecedentes_p(int m_numero_EXPEDIENTE, java.lang.String m_id_TIPO_ANTECEDENTE, java.lang.String m_descripcion) Constructor que recibe como parámetros los datos del antecedente

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
protected boolean	**consistenciaOK**() Necesario para mantener la consistencia (integridad referencial) entre antecedentes y expediente
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_numero_EXPEDIENTE) Elimina los registros de la tabla que coincidan con el número de expediente
void	**fijarDescripcion**(java.lang.String m_descripcion) Fija la descripción del antecedente
void	**fijarId_TIPO_ANTECEDENTE**(java.lang.String m_id_TIPO_ANTECEDENTE) Fija el tipo de antecedente
void	**fijarId**(int m_id) Fija el id del antecedente

void	**fijarNumero_EXPEDIENTE**(int m_numero_EXPEDIENTE) Fija el número de expediente
boolean	**guardar**() Almacena los datos del antecedente en la tabla correspondiente
java.lang.String	**obtenerDescripcion**() Devuelve la descripción del antecedente
java.lang.String	**obtenerId_TIPO_ANTECEDENTE**() Devuelve el tipo de antecedente
int	**obtenerId**() Devuelve el id del antecedente
int	**obtenerNumero_EXPEDIENTE**() Devuelve el número de expediente
boolean	**recuperar**() Recupera un registro de la tabla

Clase Antecedentes_f

Clase que provee la funcionalidad para registrar los antecedentes familiares de los pacientes.

Atributos

private java.lang.String	**id_PARENTESCO** Identificador del parentesco

Atributos heredados de la clase Antecedentes_p

descripcion, id, id_TIPO_ANTECEDENTE, m_dbaccess, numero_EXPEDIENTE

Constructores

Antecedentes_f()
Constructor sin parámetros

Antecedentes f(int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el número de expediente
Antecedentes f(int m_id, int m_numero_EXPEDIENTE) Constructor que recibe como parámetro el identificador del antecedente y el número de expediente
Antecedentes f(int m_numero_EXPEDIENTE, java.lang.String m id TIPO ANTECEDENTE, java.lang.String m descripcion, java.lang.String m_id_PARENTESCO) Constructor que recibe como parámetros los datos del antecedente

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
private boolean	**consistenciaPOK**() Necesario para mantener la consistencia (integridad referencial) entre antecedentes y parentesco
boolean	**eliminar**() Elimina un registro de la tabla
boolean	**eliminarCascada**(int m_numero_EXPEDIENTE) Elimina los registros de la tabla que coincidan con el número de expediente
void	**fijarId**(java.lang.String m_id_PARENTESCO) Fija el id del parentesco
boolean	**guardar**() Almacena los datos del antecedente en la tabla correspondiente
java.lang.String	**obtenerId PARENTESCO**() Devuelve el id del parentesco
boolean	**recuperar**()

	Recupera un registro de la tabla

Métodos heredados de la clase Antecedentes_p

consistenciaOK, fijarDescripcion, fijarId TIPO_ANTECEDENTE, fijarId, fijarNumero_EXPEDIENTE, obtenerDescripcion, obtenerId TIPO_ANTECEDENTE, obtenerId, obtenerNumero_EXPEDIENTE

Clase Hospital

Clase que provee la funcionalidad para registrar los datos de los hospitales.

Atributos

private int	**id** Identificador del hospital
private DBAccess	**m_dbaccess** Permite la conexión con la base de datos
private java.lang.String	**nombre** Nombre del hospital
private java.lang.String	**telefono** Número telefónico del hospital

Constructores

Hospital()
Constructor sin parámetros

Hospital(int m_id)
Constructor que recibe como parámetro el identificador del hospital

Hospital(java.lang.String m_nombre, java.lang.String m_telefono)
Constructor que recibe como parámetros los datos del hospital

Métodos

boolean	**actualizar**() Actualiza un registro de la tabla
private boolean	**consistencia**() Necesario para mantener la consistencia (integridad referencial) entre las tablas
boolean	**eliminar**() Elimina un registro de la tabla
void	**fijarId**(int m_id) Fija el id del hospital
void	**fijarNombre**(java.lang.String m_nombre) Fija el nombre del hospital
void	**fijarTelefono**(java.lang.String m_telefono) Fija el número telefónico del hospital
boolean	**guardar**() Almacena los datos del hospital en la tabla correspondiente
int	**obtenerId**() Devuelve el id del hospital
java.lang.String	**obtenerNombre**() Devuelve el nombre del hospital
java.lang.String	**obtenerTelefono**() Devuelve el número telefónico del hospital
boolean	**recuperar**() Recupera un registro de la tabla

2.6. Diseño de la Interfaz de Usuario

A continuación se presentan las pantallas principales que servirán para la interacción con el usuario.

(a) (b)

Figura 2.8 Ventana_crearExpediente

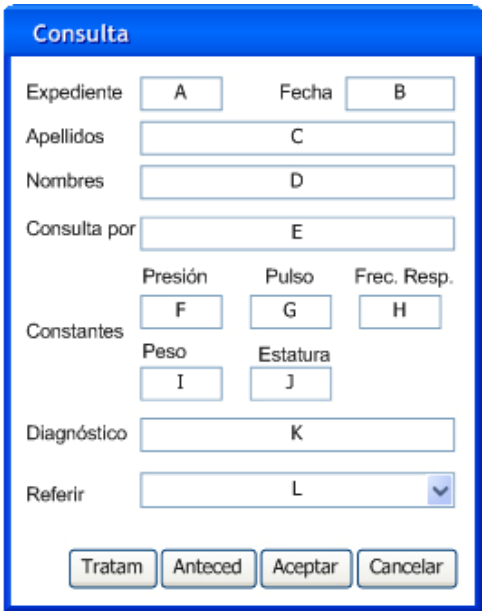


Figura 2.9 Ventana_consulta

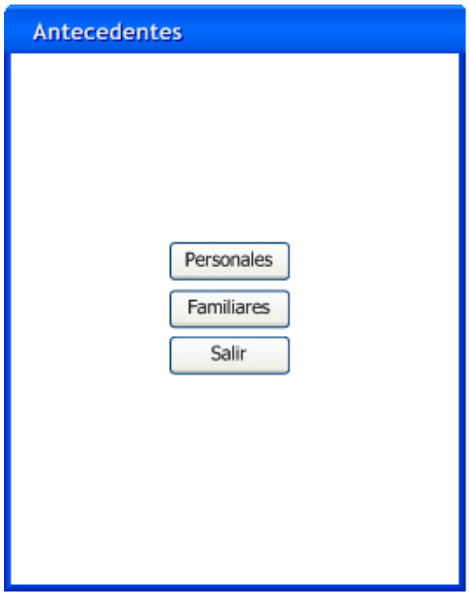


Figura 2.10 Ventana_menuAntecedentes

Figura 2.11 Ventana_antecedentesPersonales

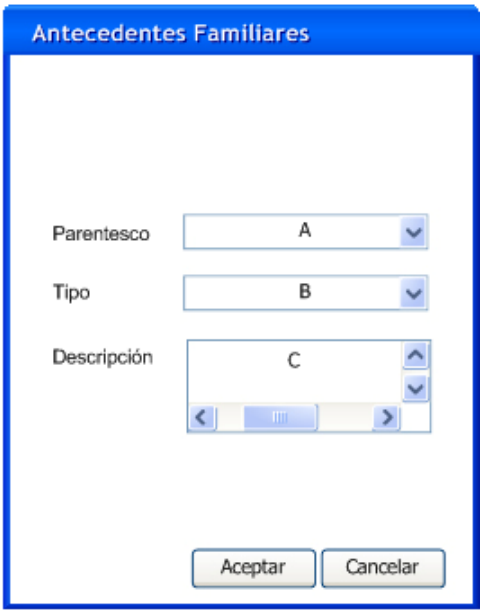


Figura 2.12 Ventana_antecedentesFamiliares

Figura 2.13 Ventana_menuTratamiento

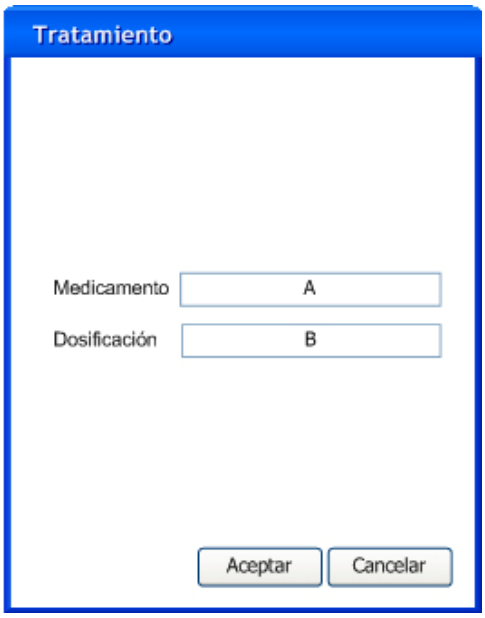


Figura 2.14 Ventana_tratamiento

Figura 2.15 Ventana_citas

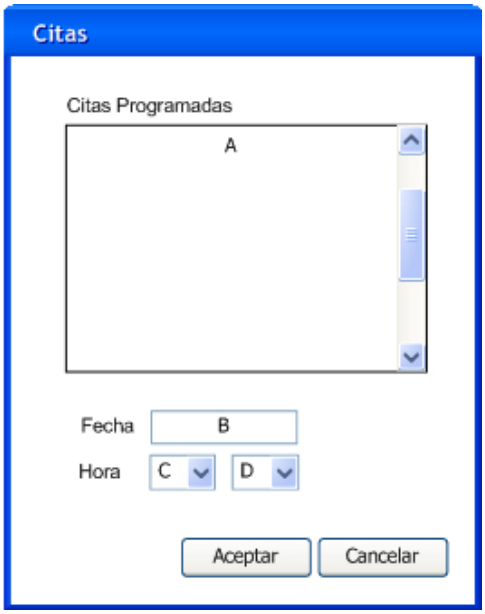


Figura 2.16 Ventana_citasProgramadas

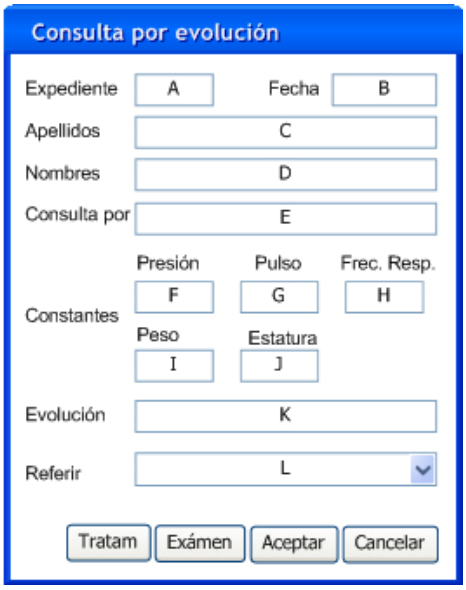


Figura 2.17 Ventana_consultaEvolucion

Figura 2.18 Ventana_menuExamen

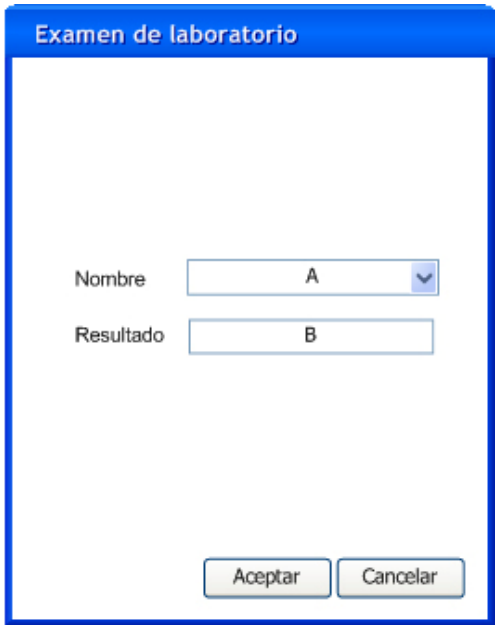


Figura 2.19 Ventana_examen

Figura 2.20 Ventana_tipoListado

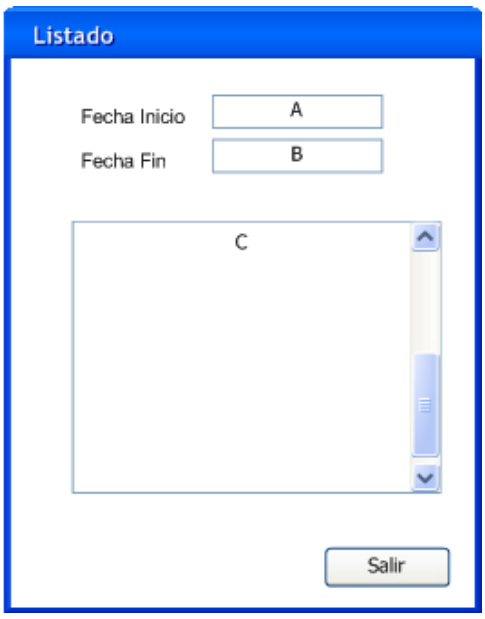


Figura 2.21 Ventana_listado

Figura 2.22 Ventana_sincronizacion

2.7. Codificación y Pruebas

2.7.1. Tecnologías Empleadas en el Desarrollo de la Aplicación

Lenguaje de programación

Para la programación de dispositivos móviles, existen diversas alternativas, tal como se puede apreciar en la sección 1.3.

Los criterios empleados para la selección del lenguaje de programación son los siguientes: posibilidad de realizar desarrollo multiplataforma, producto multiplataforma (que la aplicación desarrollada pueda ejecutarse en diferentes dispositivos), disponibilidad de drivers para la conectividad con servidores de base de datos, IDEs con emulación de dispositivos móviles, bajo costo, sencillez, soporte técnico del fabricante.

Después de aplicar los criterios antes mencionados, se elige Java 2 Micro Edition, ya que permite un desarrollo y producto multiplataforma, admite conexiones a bases de datos a través de JDBC, existen IDEs con emulación de dispositivos móviles y, si bien no es un lenguaje tan sencillo como otros disponibles en el mercado, está soportado de una manera robusta por el fabricante.

Base de datos

En lo concerniente a las bases de datos para dispositivos móviles, en la sección 1.3 se puede apreciar que existe más de una alternativa.

Los criterios empleados para la selección de la base de datos son los siguientes: proveer utilerías para la sincronización de datos, proveer un subconjunto JDBC para la conectividad a través del lenguaje de programación, footprint pequeño, soporte SQL, bajo costo.

La base de datos que mejor se adapta a los criterios antes mencionados es Point Base, ya que provee de un servidor de sincronización de datos, y si bien no tiene un bajo costo, posee un footprint muy reducido y proporciona un subconjunto JDBC para establecer la conectividad.

2.7.2. Herramientas Utilizadas en el Proceso de Desarrollo

J2ME Wireless Toolkit

Es una herramienta provista por Sun Microsystems, que incluye emulador de dispositivos móviles.

JEdit

Dado que el J2ME Wireless Toolkit no provee de un editor de código, se emplea JEdit para tal propósito.

2.7.3. Descripción de Archivos

ARCHIVO	DESCRIPCION
movilMed.jar	Archivo empaquetado que contiene la aplicación completa
movilMed.jad	Archivo que describe la aplicación contenida en movilMed.jar
MANIFEST.MF	Arechivo de manifiesto que describe el contenido del archivo movilMed.jar
movilMed.java	Contiene la pantalla de acceso a la aplicación y el menú principal.
DBAccess.java	Clase para la manipulación de la base de datos.
Expediente.java	Clase para la gestión de los expedientes.
Consulta.java	Clase para la gestión de las consultas.
Evolucion.java	Clase para la gestión de la evolución de los pacientes.
Tratamiento.java	Clase para la gestión del tratamiento de los pacientes.
Antecedentes_p.java	Clase para la gestión de los antecedentes personales de los pacientes.
Antecedentes_f.java	Clase para la gestión de los antecedentes familiares de los pacientes.
Cita.java	Clase para la gestión de las citas de los pacientes.
Examen.java	Clase para la gestión de los exámenes de los pacientes.
Hospital.java	Clase para la gestión de los hospitales a los que se refiere a los pacientes.
FechaDB.java	Clase para el manejo de fechas entre las pantallas y la base de datos.
IniciarDB.java	Clase que permite inicializar la base de datos.
PantallaExpedienteNv.java	Contiene la pantalla para la adición de expedientes.
PantallaExpedienteEd.java	Contiene la pantalla para la modificación y eliminación de expedientes.
PantallaConsultaNv.java	Contiene la pantalla para la adición de consultas.
PantallaConsultaEd.java	Contiene la pantalla para la modificación y eliminación de consultas.
PantallaEvolucionNv.java	Contiene la pantalla para la adición de la evolución de un paciente.
PantallaEvolucionEd.java	Contiene la pantalla para la modificación y eliminación de la evolución de un paciente.
PantallaListado.java	Contiene la pantalla para mostrar listados de pacientes y citas.
PantallaSincronizar.java	Contiene la pantalla que permite sincronizar los datos entre el

	dispositivo móvil y un servidor.
PantallaCambiarClave.java	Contiene la pantalla que permite cambiar la clave de la aplicación.

Tabla 2.5 Descripción de archivos con conforman la aplicación

2.7.4. Pruebas de la Aplicación

Cuando se desarrolla aplicaciones, siempre es necesario realizar pruebas, y para la aplicación prototipo desarrollada, se han realizado los siguientes tipos de pruebas.

Caja negra

Este concepto hace alusión al hecho de que se desconoce (o se hace caso omiso) de la forma en que opera internamente la aplicación. Aquí lo que se busca es probar todas las funcionalidades ofrecidas por la aplicación, ingresando datos, si bien no reales, lo más cercano posibles a los que ingresará verdaderamente el usuario final.

Caja blanca

En este tipo de prueba, se está conciente del funcionamiento interno de la aplicación, realizando un rastreo de la ejecución de líneas de código, en la medida de que se van ingresando los datos.

2.7.5. Código Fuente

A continuación se reproduce parte del código de la aplicación prototipo, se muestran dos clases que interactúan con la base de datos y dos clases que conforman parte de la interfaz de usuario.

Clases para la gestión de la base de datos

Archivo: DBAccess.java

```
//Importamos los paquetes necesarios
import com.pointbase.me.*;
import javax.microedition.rms.*;

/**
 * Clase que permite acceder a la base de datos
 * @see
```

```
 * @version    1.0 Nov 2006
 * @since      1.0 Nov 2006
 * @author     Nelson Hernández
 */
public class DBAccess {
	/**
	 * Para establecer la conexión con la base de datos
	 */
	static Connection c = null;
	/**
	 * Para las sentencias que se ejecutarán sobre la base de datos
	 */
	static Statement s = null;
	/**
	 * Especifica el nombre de la base de datos
	 */
	static String m_dbname = "fichasMedicas"; // Nombre de la base de datos



	/**
	  * Realiza la conexión a la base de datos
	  */
	public void conectar() {
		// Crea una nueva conexión
		try {
			c = DriverManager.getConnection(
			"jdbc:pointbase:micro:"+ m_dbname,"admin", "galeno");
			s = c.createStatement();
		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

	/**
	  * Se desconecta de la base de datos
	  */
	public void desconectar() {
		//Cierra la conexión a la base de datos
		try {
			if(s != null){
				s.close();
				s = null;
```

```
                }
                if(c!= null){
                        c.close();
                        c = null;
                }
        } catch (Exception e) {
                System.out.println(e.getMessage());
        }
}


/**
  * Verifica si existe la base de datos
  * @return true si existe la base de datos
 * @return false si no existe la base de datos
  */
public boolean dbExiste(){
        try{
                ResultSet rs = getTables();
                if (rs == null || rs.next() == false)
                        return false;
        }
        catch( Exception e){
                System.out.println( e.getMessage());
                return false;
        }
        return true;
}

/**
  * Verifica si la base de datos tiene tablas
  * @return true si se han encontrado tablas en la base de datos
 * @return false si no se han encontrado tablas la base de datos
  */
public ResultSet getTables(){
        try{
                return c.getMetaData().getTables(null, null, null, null);
        }catch(SQLException e){
                System.out.println(e.getMessage());
                return null;
        }
}
```

```
/**
 * Sirve para borrar la base de datos
 * @return true si no hay problema al eliminar la base de datos
 * @return false si hay problema al eliminar la base de datos
 */
public boolean borrarDB(){
        //Cerramos la conexión
        desconectar();
        //obtenemos una lista de todos los recod store en el RMS
        String lista[]=RecordStore.listRecordStores();

        //Iteramos a través de la lista de record stores
        for(int i=0;lista!=null && i<lista.length;i++){
                //Elminamos los record stores que coincidan con el nombre
                //de la base de datos
                if(lista[i].startsWith(m_dbname)){
                        try{
                                RecordStore.deleteRecordStore(lista[i]);
                        }
                        catch(Exception e){
                                System.out.println( e.getMessage());
                                return false;
                        }
                }
        }
        return true;
}


/**
 * Sirve para crear las tablas en la base de datos
 * @return true si no hay problema de creación
 * @return false si hay problema en la creación
 */
public boolean crearTablas(){
       //Cadenas para la creación de las tablas
       String crearEstado_civil = "CREATE TABLE estado_civil("
                                + "id VARCHAR(1) NOT NULL,"
                                + "nombre VARCHAR(13) NOT NULL,"
                                + "CONSTRAINT pk_estado_civil "
```

```
                                + "PRIMARY KEY (id))";

String crearTipo_documento = "CREATE TABLE tipo_documento("
                           + "id VARCHAR(1) NOT NULL,"
                           + "nombre VARCHAR(9),"
                           + "CONSTRAINT pk_tipo_documento "
                           + "PRIMARY KEY (id))";

String crearOcupacion =      "CREATE TABLE ocupacion("
                             + "id VARCHAR(2) NOT NULL,"
                           + "nombre VARCHAR(11),"
                           + "CONSTRAINT pk_ocupacion "
                           + "PRIMARY KEY (id))";

String crearParentesco =     "CREATE TABLE parentesco("
                           + "id VARCHAR(1) NOT NULL,"
                           + "nombre VARCHAR(10) NOT NULL,"
                           + "CONSTRAINT pk_parentesco "
                           + "PRIMARY KEY (id))";

String crearExpediente =     "CREATE TABLE expediente("
                           + "numero INT IDENTITY NOT NULL,"
                           + "fecha DATE NOT NULL,"
                           + "apellidos VARCHAR(50) NOT NULL,"
                           + "nombres VARCHAR(50) NOT NULL,"
                           + "sexo VARCHAR(1) NOT NULL,"
                           + "fechaNac DATE NOT NULL,"
                           + "id_ESTADO_CIVIL VARCHAR(1),"
                           + "direccion VARCHAR(50) NOT NULL,"
                           + "telefono VARCHAR(10),"
                           + "movil VARCHAR(10),"
                           + "email VARCHAR(50),"
                           + "id_TIPO_DOCUMENTO_paciente VARCHAR(1),"
                           + "numeroDoc VARCHAR(20),"
                           + "id_OCUPACION VARCHAR(2),"
                           + "padre VARCHAR(50),"
                           + "madre VARCHAR(50),"
                           + "conyuge VARCHAR(50),"
                           + "responsable VARCHAR(50),"
                           + "direccionResp VARCHAR(50),"
                           + "telefonoResp VARCHAR(10),"
                           + "nombrePDatos VARCHAR(50),"
```

```
                                    + "id_PARENTESCO_PDatos VARCHAR(1),"
                                    + "id_TIPO_DOCUMENTO_PDatos VARCHAR(1),"
                                    + "numeroDocPDatos VARCHAR(20),"
                                    + "CONSTRAINT pk_expediente "
                                    + "PRIMARY KEY (numero))";

                String crearCita =          "CREATE TABLE cita("
                                    + "id INT IDENTITY NOT NULL,"
                                    + "id_EXPEDIENTE INT,"
                                    + "fecha DATE NOT NULL,"
                                    + "hora INT NOT NULL,"
                                    + "minutos INT NOT NULL,"
                                    + "CONSTRAINT pk_cita "
                                    + "PRIMARY KEY (id))";

                String crearTipo_antecedente = "CREATE TABLE tipo_antecedente("
                                    + "id VARCHAR(1) NOT NULL,"
                                    + "nombre VARCHAR(13) NOT NULL,"
                                    + "CONSTRAINT pk_tipo_antecedente "
                                    + "PRIMARY KEY (id))";

                String crearAntecedentes_p = "CREATE TABLE antecedentes_p("
                                    + "id INT IDENTITY NOT NULL,"
                                    + "numero_EXPEDIENTE INT NOT NULL,"
                                    + "id_TIPO_ANTECEDENTE VARCHAR(1) NOT
NULL,"
                                    + "descripcion VARCHAR(50) NOT NULL,"
                                    + "CONSTRAINT pk_antecedentes_p "
                                    + "PRIMARY KEY (id, numero_EXPEDIENTE))";

                String crearAntecedentes_f = "CREATE TABLE antecedentes_f("
                                    + "id INT IDENTITY NOT NULL,"
                                    + "numero_EXPEDIENTE INT NOT NULL,"
                                    + "id_TIPO_ANTECEDENTE VARCHAR(1) NOT
NULL,"
                                    + "descripcion VARCHAR(50) NOT NULL,"
                                    + "id_PARENTESCO VARCHAR(1) NOT NULL,"
                                    + "CONSTRAINT pk_antecedentes_f "
                                    + "PRIMARY KEY (id, numero_EXPEDIENTE))";

                String crearHospital =       "CREATE TABLE hospital("
                                    + "id INT IDENTITY NOT NULL,"
                                    + "nombre VARCHAR(50) NOT NULL,"
```

```
                                + "telefono VARCHAR(10),"
                                + "CONSTRAINT pk_hospital "
                                + "PRIMARY KEY (id))";

String crearConsulta =          "CREATE TABLE consulta("
                                + "id INT IDENTITY NOT NULL,"
                                + "fecha DATE NOT NULL,"
                                + "descripcion VARCHAR(100) NOT NULL,"
                                + "diagnostico VARCHAR(200) NOT NULL,"
                                + "peso INT,"
                                + "estatura INT,"
                                + "presionSistolica INT,"
                                + "presionDiastolica INT,"
                                + "pulso INT,"
                                + "frecuenciaRespiratoria INT,"
                                + "id_HOSPITAL INT,"
                                + "CONSTRAINT pk_consulta "
                                + "PRIMARY KEY (id))";

String crearEvolucion =         "CREATE TABLE evolucion("
                                + "id INT IDENTITY NOT NULL,"
                                + "id_CONSULTA INT NOT NULL,"
                                + "fecha DATE NOT NULL,"
                                + "descripcion VARCHAR(100) NOT NULL,"
                                + "diagnostico VARCHAR(200) NOT NULL,"
                                + "peso INT,"
                                + "estatura INT,"
                                + "presionSistolica INT,"
                                + "presionDiastolica INT,"
                                + "pulso INT,"
                                + "frecuenciaRespiratoria INT,"
                                + "id_HOSPITAL INT,"
                                + "CONSTRAINT pk_evolucion "
                                + "PRIMARY KEY (id))";

String crearTratamiento =       "CREATE TABLE tratamiento("
                                + "id INT IDENTITY NOT NULL,"
                                + "b_PADRE INT NOT NULL,"
                                + "id_PADRE INT NOT NULL,"
                                + "medicamento VARCHAR(50) NOT NULL,"
                                + "dosificacion VARCHAR(50) NOT NULL,"
                                + "CONSTRAINT pk_tratamiento "
```

```
                                    + "PRIMARY KEY (id, id_PADRE))";

                String crearExamen =          "CREATE TABLE examen("
                                    + "id INT IDENTITY NOT NULL,"
                                    + "id_EVOLUCION INT NOT NULL,"
                                    + "nombre VARCHAR(25) NOT NULL,"
                                    + "resultado VARCHAR(25) NOT NULL,"
                                    + "CONSTRAINT pk_examen "
                                    + "PRIMARY KEY (id, id_EVOLUCION))";

                String crearUsuarios =        "CREATE TABLE usuario("
                                    + "id VARCHAR(1) NOT NULL,"
                                    + "nombre VARCHAR(10),"
                                    + "clave VARCHAR(6),"
                                    + "CONSTRAINT pk_usuario "
                                    + "PRIMARY KEY (id))";

                //Cadenas para la inserción de registros
                String insertarEstado_civil = "INSERT INTO estado_civil "
                                    + "VALUES(?,?)";
                String insertarTipo_documento = "INSERT INTO tipo_documento "
                                    + "VALUES(?,?)";
                String insertarOcupacion = "INSERT INTO ocupacion "
                                    + "VALUES(?,?)";
                String insertarParentesco = "INSERT INTO parentesco "
                                    + "VALUES(?,?)";
                String insertarTipo_antecedente = "INSERT INTO tipo_antecedente "
                                    + "VALUES(?,?)";
                String insertarUsuario = "INSERT INTO usuario "
                                    + "VALUES(?,?,?)";
                String[] ids={"1","2","3","4","5","6","7","8","9"};

                String[] estados = {"Soltero", "Casado", "Viudo", "Acompañado"};
                String[] tipo_doc = {"Carnet", "DUI", "DNI", "Pasaporte", "Otro"};
                String[] ocupacion = {"Jornalero", "Estudiante", "Comerciante",
"Empleado", "Desempleado", "Otro"};
                String[] parentesco = {"Madre", "Padre", "Abuela", "Abuelo",
"Hermano", "Cónyuge", "Amigo", "Otro"};
                String[] antecedente = {"Alérgico", "Tóxico", "Fisiológico",
"Patológico", "Ginecológico", "Quirúrgico"};

                try{
                        //Creamos las tablas
                        s.execute(crearEstado_civil);
```

```
                        s.execute(crearTipo_documento);
                        s.execute(crearOcupacion);
                        s.execute(crearParentesco);
                        s.execute(crearExpediente);
                        s.execute(crearCita);
                        s.execute(crearTipo_antecedente);
                        s.execute(crearAntecedentes_p);
                        s.execute(crearAntecedentes_f);
                        s.execute(crearHospital);
                        s.execute(crearConsulta);
                        s.execute(crearEvolucion);
                        s.execute(crearTratamiento);
                        s.execute(crearExamen);
                        s.execute(crearUsuarios);



                        //Ingresamos registros a las tablas
                        PreparedStatement p =
c.prepareStatement(insertarEstado_civil);
                        for (int i=0; i<4; i++){
                                p.setString(1, ids[i]);
                                p.setString(2, estados[i]);
                                p.executeUpdate();
                        }
                        p.close();

                        p = c.prepareStatement(insertarTipo_documento);
                        for (int i=0; i<5; i++){
                                p.setString(1, ids[i]);
                                p.setString(2, tipo_doc[i]);
                                p.executeUpdate();
                        }
                        p.close();



                        p = c.prepareStatement(insertarOcupacion);
                        for (int i=0; i<6; i++){
                                p.setString(1, ids[i]);
                                p.setString(2, ocupacion[i]);
                                p.executeUpdate();
                        }
                        p.close();
```

```
                p = c.prepareStatement(insertarParentesco);
                for (int i=0; i<8; i++){
                        p.setString(1, ids[i]);
                        p.setString(2, parentesco[i]);
                        p.executeUpdate();
                }
                p.close();

                p = c.prepareStatement(insertarTipo_antecedente);
                for (int i=0; i<6; i++){
                        p.setString(1, ids[i]);
                        p.setString(2, antecedente[i]);
                        p.executeUpdate();
                }
                p.close();


                p = c.prepareStatement(insertarUsuario);

                p.setString(1, "1");
                p.setString(2, "admin");
                p.setString(3, "0000");
                p.executeUpdate();

                p.close();

                c.commit();

        }
        catch (SQLException e){
                System.out.println(e.getMessage());
                return false;
        }
        return true;
 }

/**
 * Para ejecutar una sentencia sql sobre la base de datos
 * @param sql String sentencia sql que se ejecutará
 * @return -1 si hay problema en la ejecución
 */
 public int executeUpdate(String sql) {
        try {
```

```
                return(s.executeUpdate(sql));
        } catch (Exception e) {
                System.out.println(e.getMessage());
                return -1;
        }
}

/**
 * Para ejecutar una sentencia sql sobre la base de datos
* @param sql String sentencia sql que se ejecutará
* @return Numero que indica el valor autonumérico generado en la inserción
* @return 0 si hay problema en la ejecución
 */
public int executeUpdateValue(String sql) {
      try{
              s.executeUpdate(sql);
              ResultSet rs = s.getGeneratedKeys();
              if(rs==null || !rs.next()) {
                      return 0;
              }
              int ultimo_id = rs.getInt(1);
              rs.close();
              System.out.println(sql);
              return ultimo_id;
      }
      catch (Exception e) {
               System.out.println(e.getMessage());
               return 0;
      }
}

/**
 * Para ejecutar una sentencia sql tipo SELECT sobre la base de datos
* @param sql String senencia sql que se ejecutará
* @return ResulSet con el resultado de la ejecución del SELECT
* @return NULL si hay problemas en la ejecución
 */
public ResultSet execute(String sql) {
        ResultSet rs = null;
        try {
                if (s.execute(sql)) {
                        rs = s.getResultSet();
```

```
                        }
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                        return null;
                }
                finally {
                        return rs;
                }
        }
}
```

Archivo: Expediente.java

```
//Importamos los paquetes necesarios
import java.util.*;
import com.pointbase.me.*;

/**
 * Clase que provee la funcionalidad para registrar los expedientes de los pacientes
 * @see        DBAccess
 * @version    1.0 Nov 2006
 * @since      1.0 Nov 2006
 * @author     Nelson Hernández
 */
public class Expediente{
        /**
         * Número de expediente
         */
        private int numero;
        /**
         * Fecha de creación del expediente
         */
        private String fecha;
        /**
         * Apellidos del paciente
         */
        private String apellidos;
        /**
         * Nombres del paciente
         */
        private String nombres;
        /**
```

```
 * Sexo del paciente
 */
private String sexo;
/**
 * Fecha de nacimiento del paciente
 */
private String fechaNac;
/**
 * Estado civil del paciente
 */
private String id_ESTADO_CIVIL;
/**
 * Dirección del paciente
 */
private String direccion;
/**
 * Número de teléfono fijo del paciente
 */
private String telefono;
/**
 * Número de teléfono móvil del paciente
 */
private String movil;
/**
 * Dirección de correo electrónico del paciente
 */
private String email;
/**
 * Tipo de documento del paciente
 */
private String id_TIPO_DOCUMENTO_paciente;
/**
 * Número de documento del paciente
 */
private String numeroDoc;
/**
 * Ocupación del paciente
 */
private String id_OCUPACION;
/**
 * Nombre de la madre del paciente
 */
```

```
private String padre;
/**
 * Nombre del padre del paciente
 */
private String madre;
/**
 * Nombres del cónyuge del paciente
 */
private String conyuge;
/**
 * Nombres del responsable del paciente
 */
private String responsable;
/**
 * Dirección del responsable del paciente
 */
private String direccionResp;
/**
 * Número telefónico del responsable del paciente
 */
private String telefonoResp;
/**
 * Nombre de la persona que proporciona los datos del paciente
 */
private String nombrePDatos;
/**
 * Parentesco de la persona que proporciona los datos del paciente
 */
private String id_PARENTESCO_PDatos;
/**
 * Tipo de documento de la persona que proporciona los datos del paciente
 */
private String id_TIPO_DOCUMENTO_PDatos;
/**
 * Número de documento de la persona que proporciona los datos del paciente
 */
private String numeroDocPDatos;
/**
 * Permite la conexión con la base de datos
 */
private DBAccess m_dbaccess = null;
```

```
/**
 * Constructor sin parámetros
 */
public Expediente(){
	numero = 0;
}
/**
 * Constructor que recibe como parámetro el número de expediente
 * @param m_numero int numero de expediente
 */
public Expediente(int m_numero){
	numero = m_numero;
}
/**
 * Constructor que recibe como parámetros los datos del paciente
 */
public Expediente(
	//int m_numero,
	String m_fecha,
	String m_apellidos,
	String m_nombres,
	String m_sexo,
	String m_fechaNac,
	String m_id_ESTADO_CIVIL,
	String m_direccion,
	String m_telefono,
	String m_movil,
	String m_email,
	String m_id_TIPO_DOCUMENTO_paciente,
	String m_numeroDoc,
	String m_id_OCUPACION,
	String m_padre,
	String m_madre,
	String m_conyuge,
	String m_responsable,
	String m_direccionResp,
	String m_telefonoResp,
	String m_nombrePDatos,
	String m_id_PARENTESCO_PDatos,
	String m_id_TIPO_DOCUMENTO_PDatos,
	String m_numeroDocPDatos
```

```
){
        numero = 0;
        fecha = m_fecha;
        apellidos = m_apellidos;
        nombres = m_nombres;
        sexo = m_sexo;
        fechaNac = m_fechaNac;
        id_ESTADO_CIVIL = m_id_ESTADO_CIVIL;
        direccion = m_direccion;
        telefono = m_telefono;
        movil = m_movil;
        email = m_email;
        id_TIPO_DOCUMENTO_paciente = m_id_TIPO_DOCUMENTO_paciente;
        numeroDoc = m_numeroDoc;
        id_OCUPACION = m_id_OCUPACION;
        padre = m_padre;
        madre = m_madre;
        conyuge = m_conyuge;
        responsable = m_responsable;
        direccionResp = m_direccionResp;
        telefonoResp = m_telefonoResp;
        nombrePDatos = m_nombrePDatos;
        id_PARENTESCO_PDatos = m_id_PARENTESCO_PDatos;
        id_TIPO_DOCUMENTO_PDatos = m_id_TIPO_DOCUMENTO_PDatos;
        numeroDocPDatos = m_numeroDocPDatos;
}

/**
 * Fija el número de expediente
 * @param m_numero int numero de expediente
 */
public void fijarNumero(int m_numero){
        numero = m_numero;
}

/**
 * Fija la fecha de creación del expediente
 * @param m_fecha Date fecha de creación del expediente
 */
public void fijarFecha(String m_fecha){
        fecha = m_fecha;
}
```

```
/**
 * Fija los apellidos del paciente
 * @param m_apellidos String apellidos del paciente
 */
public void fijarApellidos(String m_apellidos){
	apellidos = m_apellidos;
}
/**
 * Fija los nombres del paciente
 * @param m_nombres String nombres del paciente
 */
public void fijarNombres(String m_nombres){
	nombres = m_nombres;
}
/**
 * Fija el sexo
 * @param m_sexo String sexo del paciente
 */
public void fijarSexo(String m_sexo){
	sexo = m_sexo;
}
/**
 * Fija la fecha de nacimiento del paciente
 * @param m_fechaNac Date fecha de nacimiento del paciente
 */
public void fijarFechaNac(String m_fechaNac){
	fechaNac = m_fechaNac;
}
/**
 * Fija el estado civil del paciente
 * @param m_id_ESTADO_CIVIL String estado civil del paciente
 */
public void fijarId_ESTADO_CIVIL(String m_id_ESTADO_CIVIL){
	id_ESTADO_CIVIL = m_id_ESTADO_CIVIL;
}
/**
 * Fija la dirección del paciente
 * @param m_direccion String dirección del paciente
 */
public void fijarDireccion(String m_direccion){
	direccion = m_direccion;
}
```

```
	/**
	 * Fija el número de teléfono fijo del paciente
	 * @param m_telefono String número de teléfono fijo del paciente
	 */
	public void fijarTelefono(String m_telefono){
		telefono = m_telefono;
	}
	/**
	 * Fija el número de teléfono móvil del paciente
	 * @param m_movil String número de teléfono móvil del paciente
	 */
	public void fijarMovil(String m_movil){
		movil = m_movil;
	}
	/**
	 * Fija la dirección de correo electrónico del paciente
	 * @param m_email String dirección de correo electrónico del paciente
	 */
	public void fijarEmail(String m_email){
		email = m_email;
	}
	/**
	 * Fija el tipo de documento del paciente
	 * @param m_id_TIPO_DOCUMENTO_paciente String tipo de documento del paciente
	 */
	public void fijarId_TIPO_DOCUMENTO_paciente(String
m_id_TIPO_DOCUMENTO_paciente){
		id_TIPO_DOCUMENTO_paciente = m_id_TIPO_DOCUMENTO_paciente;
	}
	/**
	 * Fija el número de documento del paciente
	 * @param m_numeroDoc String número de documento del paciente
	 */
	public void fijarNumeroDoc(String m_numeroDoc){
		numeroDoc = m_numeroDoc;
	}
	/**
	 * Fija la ocupación del paciente
	 * @param m_id_OCUPACION String ocupación del paciente
	 */
	public void fijarId_OCUPACION(String m_id_OCUPACION){
		id_OCUPACION = m_id_OCUPACION;
	}
```

```
	/**
	 * Fija el nombre del padre del paciente
	 * @param m_padre String nombre del padre del paciente
	 */
	public void fijarPadre(String m_padre){
		padre = m_padre;
	}
	/**
	 * Fija el nombre de la madre del paciente
	 * @param m_madre String nombre de la madre del paciente
	 */
	public void fijarMadre(String m_madre){
		madre = m_madre;
	}
	/**
	 * Fija el nombre del cónyuge del paciente
	 * @param m_conyuge String nombre del cónyuge del paciente
	 */
	public void fijarConyuge(String m_conyuge){
		conyuge = m_conyuge;
	}
	/**
	 * Fija el nombre del responsable del paciente
	 * @param m_responsable String nombre del responsable del paciente
	 */
	public void fijarResponsable(String m_responsable){
		responsable = m_responsable;
	}
	/**
	 * Fija la dirección del responsable del paciente
	 * @param m_direccionResp String dirección del responsable del paciente
	 */
	public void fijarDireccionResp(String m_direccionResp){
		direccionResp = m_direccionResp;
	}
	/**
	 * Fija el número de teléfono del responsable del paciente
	 * @param m_telefonoResp String número de teléfono del responsable del
paciente
	 */
	public void fijarTelefonoResp(String m_telefonoResp){
		telefonoResp = m_telefonoResp;
	}
```

```
	/**
	 * Fija el nombre de la persona que proporciona los datos del paciente
	 * @param m_nombrePDatos String nombre de la persona que proporciona los
datos del paciente
	 */
	public void fijarNombrePDatos(String m_nombrePDatos){
		nombrePDatos = m_nombrePDatos;
	}
	/**
	 * Fija el parentesco de la persona que proporciona los datos del paciente
	 * @param m_id_PARENTESCO_PDatos String parentesco de la persona que
proporciona los datos del paciente
	 */
	public void fijarId_PARENTESCO_PDatos(String m_id_PARENTESCO_PDatos){
		id_PARENTESCO_PDatos = m_id_PARENTESCO_PDatos;
	}
	/**
	 * Fija el tipo de documento de la persona que proporciona los datos del
paciente
	 * @param m_id_TIPO_DOCUMENTO_PDatos String tipo de documento de la persona
que proporciona los datos del paciente
	 */
	public void fijarId_TIPO_DOCUMENTO_PDatos(String m_id_TIPO_DOCUMENTO_PDatos){
		id_TIPO_DOCUMENTO_PDatos = m_id_TIPO_DOCUMENTO_PDatos;
	}
	/**
	 * Fija el número de documento de la persona que proporciona los datos del
paciente
	 * @param m_numeroDocPDatos String número de documento de la persona que
proporciona los datos del paciente
	 */
	public void fijarNumeroDocPDatos(String m_numeroDocPDatos){
		numeroDocPDatos = m_numeroDocPDatos;
	}



	/**
	 * Devuelve el número de expediente
	 * @return numero int numero de expediente
	 */
	public int obtenerNumero (){
		return numero;
	}


	/**
```

```
 * Devuelve la fecha de creación del expediente
 * @return fecha Date fecha de creación del expediente
 */
public String obtenerFecha(){
	return fecha;
}

/**
 * Devuelve los apellidos del paciente
 * @return apellidos String apellidos del paciente
 */
public String obtenerApellidos(){
	return apellidos;
}

/**
 * Devuelve los nombres del paciente
 * @return nombres String nombres del paciente
 */
public String obtenerNombres(){
	return nombres;
}

/**
 * Devuelve el sexo
 * @return sexo String sexo del paciente
 */
public String obtenerSexo(){
	return sexo;
}

/**
 * Devuelve la fecha de nacimiento del paciente
 * @return fechaNac String fecha de nacimiento del paciente
 */
public String obtenerFechaNac(){
	return fechaNac;
}

/**
 * Devuelve el estado civil del paciente
 * @return id_ESTADO_CIVIL String estado civil del paciente
```

```
 */
public String obtenerId_ESTADO_CIVIL(){
	return id_ESTADO_CIVIL;
}

/**
 * Devuelve la dirección del paciente
 * @return direccion String dirección del paciente
 */
public String obtenerDireccion(){
	return direccion;
}

/**
 * Devuelve el número de teléfono fijo del paciente
 * @return telefono String número de teléfono fijo del paciente
 */
public String obtenerTelefono(){
	return telefono;
}

/**
 * Devuelve el número de teléfono móvil del paciente
 * @return movil String número de teléfono móvil del paciente
 */
public String obtenerMovil(){
	return movil;
}

/**
 * Devuelve la dirección de correo electrónico del paciente
 * @return email String dirección de correo electrónico del paciente
 */
public String obtenerEmail(){
	return email;
}

/**
 * Devuelve el tipo de documento del paciente
 * @return id_TIPO_DOCUMENTO_paciente String tipo de documento del paciente
 */
public String obtenerId_TIPO_DOCUMENTO_paciente(){
```

```
        return id_TIPO_DOCUMENTO_paciente;
}

/**
 * Devuelve el número de documento del paciente
 * @return numeroDoc String número de documento del paciente
 */
public String obtenerNumeroDoc(){
        return numeroDoc;
}

/**
 * Devuelve la ocupación del paciente
 * @return id_OCUPACION String ocupación del paciente
 */
public String obtenerId_OCUPACION(){
        return id_OCUPACION;
}

/**
 * Devuelve el nombre del padre del paciente
 * @return padre String nombre del padre del paciente
 */
public String obtenerPadre(){
        return padre;
}

/**
 * Devuelve el nombre de la madre del paciente
 * @return madre String nombre de la madre del paciente
 */
public String obtenerMadre(){
        return madre;
}

/**
 * Devuelve el nombre del cónyuge del paciente
 * @return conyuge String nombre del cónyuge del paciente
 */
public String obtenerConyuge(){
        return conyuge;
}
```

```
        /**
         * Devuelve el nombre del responsable del paciente
         * @return responsable String nombre del responsable del paciente
         */
        public String obtenerResponsable(){
                return responsable;
        }

        /**
         * Devuelve la dirección del responsable del paciente
         * @return direccionResp String dirección del responsable del paciente
         */
        public String obtenerDireccionResp(){
                return direccionResp;
        }

        /**
         * Devuelve el número de teléfono del responsable del paciente
         * @return telefonoResp String número de teléfono del responsable del
paciente
         */
        public String obtenerTelefonoResp(){
                return telefonoResp;
        }

        /**
         * Devuelve el nombre de la persona que proporciona los datos del paciente
         * @return nombrePDatos String nombre de la persona que proporciona los datos
del paciente
         */
        public String obtenerNombrePDatos(){
                return nombrePDatos;
        }

        /**
         * Devuelve el parentesco de la persona que proporciona los datos del
paciente
         * @return id_PARENTESCO_PDatos String parentesco de la persona que
proporciona los datos del paciente
         */
        public String obtenerId_PARENTESCO_PDatos(){
                return id_PARENTESCO_PDatos;
        }
```

```
	/**
	 * Devuelve el tipo de documento de la persona que proporciona los datos del paciente
	 * @return id_TIPO_DOCUMENTO_PDatos String tipo de documento de la persona que proporciona los datos del paciente
	 */
	public String obtenerId_TIPO_DOCUMENTO_PDatos(){
		return id_TIPO_DOCUMENTO_PDatos;
	}

	/**
	 * Devuelve el número de documento de la persona que proporciona los datos del paciente
	 * @return numeroDocPDatos String número de documento de la persona que proporciona los datos del paciente
	 */
	public String obtenerNumeroDocPDatos(){
		return numeroDocPDatos;
	}

	/**
	 * Almacena los datos del paciente en la tabla expediente
	 * @return numero si no hay problemas al guardar el registro
	 * @return 0 si hay problemas al guardar el registro
	 */
	 public int guardar(){
		if (numero == 0){ //Si es un nuevo objeto
			String query = "INSERT INTO expediente("
					+ "fecha,"
					+ "apellidos,"
					+ "nombres,"
					+ "sexo,"
					+ "fechaNac,"
					+ "id_ESTADO_CIVIL,"
					+ "direccion,"
					+ "telefono,"
					+ "movil,"
					+ "email,"
					+ "id_TIPO_DOCUMENTO_paciente,"
					+ "numeroDoc,"
					+ "id_OCUPACION,"
					+ "padre,"
					+ "madre,"
```

```
                    + "conyuge,"
                    + "responsable,"
                    + "direccionResp,"
                    + "telefonoResp,"
                    + "nombrePDatos,"
                    + "id_PARENTESCO_PDatos,"
                    + "id_TIPO_DOCUMENTO_PDatos,"
                    + "numeroDocPDatos"
                    + ") VALUES("
                    + "'" + fecha + "',"
                    + "'" + apellidos + "',"
                    + "'" + nombres + "',"
                    + "'" + sexo + "',"
                    + "'" + fechaNac + "',"
                    + "'" + id_ESTADO_CIVIL + "',"
                    + "'" + direccion + "',"
                    + "'" + telefono + "',"
                    + "'" + movil + "',"
                    + "'" + email + "',"
                    + "'" + id_TIPO_DOCUMENTO_paciente + "',"
                    + "'" + numeroDoc + "',"
                    + "'" + id_OCUPACION + "',"
                    + "'" + padre + "',"
                    + "'" + madre + "',"
                    + "'" + conyuge + "',"
                    + "'" + responsable + "',"
                    + "'" + direccionResp + "',"
                    + "'" + telefonoResp + "',"
                    + "'" + nombrePDatos + "',"
                    + "'" + id_PARENTESCO_PDatos + "',"
                    + "'" + id_TIPO_DOCUMENTO_PDatos + "',"
                    + "'" + numeroDocPDatos + "'"
                    +")";
            DBAccess m_dbaccess = new DBAccess();
            //Conectamos a la base de datos
            m_dbaccess.conectar();
            int numeroExpediente;
            numeroExpediente = m_dbaccess.executeUpdateValue(query);
            //Nos desconectamos de la base de datos
            m_dbaccess.desconectar();
            m_dbaccess = null;
            return numeroExpediente;
```

```
                }
                else return 0;
        }

        /**
         * Recupera un registro de la tabla expediente
         * @return true si no hay problemas al recuperar el registro
         * @return false si hay problemas al recuperar el registro
         */
        public boolean recuperar(){
                String query = "SELECT * FROM expediente WHERE numero="
                                + numero;
                DBAccess m_dbaccess = new DBAccess();
                //Conectamos a la base de datos
                m_dbaccess.conectar();
                ResultSet rs = null;
                boolean m_bandera = false;
                if ((rs = m_dbaccess.execute(query)) != null){// null hubo un
problema
                        //Fijamos los atributos con los valores obtenidos
                        try{
                                rs.next();
                                fecha = rs.getString(2);
                                apellidos = rs.getString(3);
                                nombres = rs.getString(4);
                                sexo = rs.getString(5);
                                fechaNac = rs.getString(6);
                                id_ESTADO_CIVIL= rs.getString(7);
                                direccion = rs.getString(8);
                                telefono = rs.getString(9);
                                movil = rs.getString(10);
                                email = rs.getString(11);
                                id_TIPO_DOCUMENTO_paciente = rs.getString(12);
                                numeroDoc = rs.getString(13);
                                id_OCUPACION = rs.getString(14);
                                padre = rs.getString(15);
                                madre = rs.getString(16);
                                conyuge = rs.getString(17);
                                responsable = rs.getString(18);
                                direccionResp = rs.getString(19);
                                telefonoResp = rs.getString(20);
                                nombrePDatos = rs.getString(21);
                                id_PARENTESCO_PDatos = rs.getString(22);
```

```
                        id_TIPO_DOCUMENTO_PDatos = rs.getString(23);
                        numeroDocPDatos = rs.getString(24);
                        rs.close();
                        m_bandera = true;
                }
                catch (Exception e){
                        System.out.println("aqui ta!!");
                        System.out.println(e.getMessage());
                }
        }
        else
                m_bandera = false; // no funcionó
        //Nos desconectamos de la base de datos
        m_dbaccess.desconectar();
        m_dbaccess = null;
        return m_bandera;
}

/**
 * Actualiza un registro de la tabla expediente
 * @return true si no hay problemas al actualizar el registro
 * @return false si hay problemas al actualizar el registro
 */
public boolean actualizar(){
        if (numero != 0){ //Si es un objeto modificado
                String query = "UPDATE expediente SET "
                                + "fecha="
                                + "'" + fecha + "',"
                                + "apellidos="
                                + "'" + apellidos + "',"
                                + "nombres="
                                + "'" + nombres + "',"
                                + "sexo="
                                + "'" + sexo + "',"
                                + "fechaNac="
                                + "'" + fechaNac + "',"
                                + "id_ESTADO_CIVIL="
                                + "'" + id_ESTADO_CIVIL + "',"
                                + "direccion="
                                + "'" + direccion + "',"
                                + "telefono="
                                + "'" + telefono + "',"
```

```
                                + "movil="
                                + "'" + movil + "',"
                                + "email="
                                + "'" + email + "',"
                                + "id_TIPO_DOCUMENTO_paciente="
                                + "'" + id_TIPO_DOCUMENTO_paciente + "',"
                                + "numeroDoc="
                                + "'" + numeroDoc + "',"
                                + "id_OCUPACION="
                                + "'" + id_OCUPACION + "',"
                                + "padre="
                                + "'" + padre + "',"
                                + "madre="
                                + "'" + madre + "',"
                                + "conyuge="
                                + "'" + conyuge + "',"
                                + "responsable="
                                + "'" + responsable + "',"
                                + "direccionResp="
                                + "'" + direccionResp + "',"
                                 + "telefonoResp="
                                + "'" + telefonoResp + "',"
                                + "nombrePDatos="
                                + "'" + nombrePDatos + "',"
                                 + "id_PARENTESCO_PDatos="
                                + "'" + id_PARENTESCO_PDatos + "',"
                                + "id_TIPO_DOCUMENTO_PDatos="
                                + "'" + id_TIPO_DOCUMENTO_PDatos + "',"
                                + "numeroDocPDatos="
                                + "'" + numeroDocPDatos + "' "
                                + "WHERE numero="
                                + numero;
                        DBAccess m_dbaccess = new DBAccess();
                        //Conectamos a la base de datos
                        m_dbaccess.conectar();
                        boolean m_bandera = false;
                        if (m_dbaccess.executeUpdate(query) != -1)// -1 hubo un
problema
                               m_bandera = true; //Actualiza el registro
                        else
                               m_bandera = false; // no funcionó
                        //Nos desconectamos de la base de datos
                        m_dbaccess.desconectar();
```

```
                m_dbaccess = null;
                return m_bandera;
        }
        else return false;
}

/**
 * Elimina un registro de la tabla expediente
 * @return true si no hay problemas al eliminar el registro
 * @return false si hay problemas al eliminar el registro
 */
public boolean eliminar(){
        String query = "DELETE FROM expediente WHERE numero="
                        + numero;
        DBAccess m_dbaccess = new DBAccess();
        //Conectamos a la base de datos
        m_dbaccess.conectar();
        boolean m_bandera = false;
        if (m_dbaccess.executeUpdate(query) != -1){// -1 hubo un problema
                //Controlamos la consistencia (integridad referencial)
                 consistencia();
                 numero = 0; //Reseteamos el identificador
                 m_bandera = true;
        }
        else
                 m_bandera = false; // no funcionó
        //Nos desconectamos de la base de datos
        m_dbaccess.desconectar();
        m_dbaccess = null;
        return m_bandera;
}

/**
 * Necesario para mantener la consistencia (integridad referencial) entre las
tablas
 * @return true si no hay problemas durante la ejecución
 * @return false si hay problemas durante la ejecución
 */
private boolean consistencia(){
        //Borramos los antecedentes personales
        Antecedentes_p m_antecedentes_p = new Antecedentes_p();
        m_antecedentes_p.eliminarCascada(numero);
        m_antecedentes_p = null;
```

```
                //Borramos los antecedentes familiares
                Antecedentes_f m_antecedentes_f = new Antecedentes_f();
                m_antecedentes_f.eliminarCascada(numero);
                m_antecedentes_f = null;

                //Borramos las consultas
                Consulta m_consulta = new Consulta();
                m_consulta.eliminarCascada(numero);
                m_consulta = null;

                //Borramos las citas
                Cita m_cita = new Cita();
                m_cita.eliminarCascada(numero);
                m_cita = null;
                return true;
        }
}
```

Clases para la interfaz de usuario

Archivo: movilMed.java

```
//Importamos los paquetes necesarios
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;
import com.pointbase.me.*;

/**
 * Clase que muestra el menú principal de la aplicación
 * @see
 * @version    1.0 Nov 2006
 * @since      1.0 Nov 2006
 * @author     Nelson Hernández
 */
public class movilMed extends MIDlet
        implements CommandListener{
        //Manejador de la pantalla
        private Display display;

        //Pantalla para pedir el ingreso de la clave
        private Form frmClave;
        private TextField txtClave;
```

```
//Variable para controlar el inicio de la apicación
private boolean iniciado;

//Lista para los menus
private List listaMenuPrincipal;
private List listaMenuExpediente;
private List listaMenuConsulta;
private List listaMenuEvolucion;
private List listaMenuListados;
private List listaMenuAdministrar;

//Arreglo de imágenes
Image[] iconoApp;
Image[] iconosMain;
Image[] imagenOpcion;
Image[] iconosGestion;
Image[] iconosListado;

//Botones de comando
private Command cmdSalir;
private Command cmdCancelar;
private Command cmdCancelarInicio;
private Command cmdAceptarInicio;

//Ventana para mostrar la opcion seleccionada
private Alert alertSplash;

private String menuActual = null;


/**
 * Constructor sin parámetros
 */
public movilMed(){
        iniciado=false;

        //Creamos los elementos de interfaz
        txtClave = new TextField ("Ingrese la clave: ", "", 6,
                TextField.PASSWORD);

        //Inicializamos los comandos de control de la aplicación
```

```
                cmdCancelarInicio = new Command("Cancelar",Command.BACK,1);
                cmdAceptarInicio = new Command("Aceptar",Command.OK,2);

                cmdCancelar = new Command("Cancelar",Command.BACK,0);
                cmdSalir = new Command("Salir",Command.EXIT,0);

                //Cargamos las imagenes necesarias para la aplicacion
                iconoApp = new Image[1];
                iconosMain = new Image[6];
                imagenOpcion = new Image[6];
                iconosGestion = new Image[5];
                iconosListado = new Image[2];
                try {
                        //Icono de la aplicacion
                        iconoApp[0] = Image.createImage("/images/movilMed.png");

                        //Iconos del menu principal
                        iconosMain[0] = Image.createImage("/images/expediente.png");
                        iconosMain[1] = Image.createImage("/images/consulta.png");
                        iconosMain[2] = Image.createImage("/images/evolucion.png");
                        iconosMain[3] = Image.createImage("/images/listado.png");
                        iconosMain[4] = Image.createImage("/images/sincronizar.png");
                        iconosMain[5] = Image.createImage("/images/administrar.png");

                        //Imagenes (se muestran al seleccionar una opcion)
                        imagenOpcion[0] =
Image.createImage("/images/expediente_1.png");
                        imagenOpcion[1] = Image.createImage("/images/consulta_1.png");
                        imagenOpcion[2] =
Image.createImage("/images/evolucion_1.png");
                        imagenOpcion[3] = Image.createImage("/images/listado_1.png");
                        imagenOpcion[4] =
Image.createImage("/images/sincronizar_1.png");
                        imagenOpcion[5] =
Image.createImage("/images/administrar_1.png");

                        //Iconos para los submenus de gestión
                        iconosGestion[0] = Image.createImage("/images/nuevo.png");
                        iconosGestion[1] = Image.createImage("/images/editar.png");
                        iconosGestion[2] = Image.createImage("/images/database.png");
                        iconosGestion[3] = Image.createImage("/images/clave.png");
                        iconosGestion[4] =
Image.createImage("/images/errorClave.png");
```

```
                //Iconos para el menú listado
                iconosListado[0] = Image.createImage("/images/citas.png");
                iconosListado[1] = Image.createImage("/images/pacientes.png");
        }
        catch (Exception e){
                System.out.println("Excepción: " + e.getMessage());
        }
}

/**
 * Inicia el MIDLet
 */
public void startApp(){
        //Obtenemos el objeto Display del midlet
        display = Display.getDisplay(this);
        if(!iniciado){ //si es la primera vez que se corre
                login();
        }
        else mainMenu(); //Mostramos el menu principal en pantalla
}


/**
 * Pide el ingreso de la clave para iniciar la aplicación
 */
public void login(){
        //Creamos la pantalla para el acceso a la aplicación
        frmClave = new Form("Clave");

        //Agregamos los elementos de interfaz al formulario
        frmClave.append(txtClave);
        frmClave.addCommand(cmdCancelarInicio);
        frmClave.addCommand(cmdAceptarInicio);
        //Indicamos el manejador de eventos
        frmClave.setCommandListener(this);
        //Mostramos el formulario para ingresar la clave
        display.setCurrent (frmClave);
}

/**
 * Verifica la clave ingresada
 */
```

```
        public void verificarClave(){
                //Realizamos la conexión a la base de datos
                DBAccess m_dbaccess = new DBAccess();
                m_dbaccess.conectar();
                //Obenemos la clave almacenada
                String clave = obtenerClave(m_dbaccess);
                //Nos desconectamos de la base de datos;
                m_dbaccess.desconectar();
                m_dbaccess = null;

                //Si la clave es correcta
                if (clave.compareTo(txtClave.getString())==0){
                        iniciado = true;
                        mainMenu();
                }
                else {
                        Alert mensaje = new Alert("Error","",
                                iconosGestion[4], AlertType.ERROR);
                        display.setCurrent(mensaje);
                }
        }


        /**
         * obtiene la clave guardada en la base de datos
         * @param m_dbaccess DBAccess manejador de la base de datos
         * @return String clave guardada en la base de datos
         */
         public String obtenerClave(DBAccess m_dbaccess){
                if (m_dbaccess.dbExiste()){ //Si existen las tablas
                        //Cadena para obtener la clave
                        String query = "SELECT clave FROM usuario WHERE
nombre='admin'";
                        ResultSet rs = null;
                        if ((rs = m_dbaccess.execute(query)) != null){// null hubo un
problema
                                //Obtenemos la clave la clave
                                try{
                                        rs.next();
                                        String clave = rs.getString(1);
                                        rs.close();
                                        return clave;
                                }
```

```
                        catch (Exception e){
                                System.out.println(e.getMessage());
                                String clave = "0000"; //Clave por defecto
                                return clave;
                        }
                }
                else{
                        String clave = "0000"; //Clave por defecto
                        return clave;
                }
        }
        else{
                String clave = "0000"; //Clave por defecto
                return clave;
        }
 }


/**
 * Muestra el menú principal de la aplicación
 */
public void mainMenu(){
        menuActual = "Menú Principal";
        if(listaMenuPrincipal == null){ //Si aun no se ha creado el menú
                listaMenuPrincipal = new List("movilMed",
                        List.IMPLICIT);
                //Agregamos los elementos del menú principal
                listaMenuPrincipal.append("Expediente", iconosMain[0]);
                listaMenuPrincipal.append("Consulta", iconosMain[1]);
                listaMenuPrincipal.append("Evolución", iconosMain[2]);
                listaMenuPrincipal.append("Listados", iconosMain[3]);
                listaMenuPrincipal.append("Sincronizar", iconosMain[4]);
                listaMenuPrincipal.append("Administrar", iconosMain[5]);
                listaMenuPrincipal.addCommand(cmdSalir);
                //Indicamos el manejador de eventos
                listaMenuPrincipal.setCommandListener(this);
        }
        //Mostramos el menu principal
        if (alertSplash == null){
                splash("MovilMed", null, iconoApp[0]);
                display.setCurrent (alertSplash, listaMenuPrincipal);
        }
```

```
        else{
                display.setCurrent (listaMenuPrincipal);
        }
}

/**
 * Muestra una pantalla de indicando la opción seleccionada en el menú
 * @param strTitulo String título de la pantalla
 * @param strMensaje String mensaje a mostrar
 * @param imagen Image imagen a desplegar en la pantalla
 */
public void splash(String strTitulo, String strMensaje,
        Image imagen){
        alertSplash = new Alert (strTitulo, strMensaje,
                imagen, AlertType.INFO);
        //La ventana esperará 3 segundos
        alertSplash.setTimeout(3000);
}

/**
 * Muestra el menú correspondiente a expediente
 */
public void menuExpediente(){
        menuActual = "Menú Expediente";
        //Agregamos las opciones a la lista
        if (listaMenuExpediente == null){
                listaMenuExpediente = new List("Expedientes",
                        List.IMPLICIT);
                listaMenuExpediente.append("Nuevo", iconosGestion[0]);
                listaMenuExpediente.append("Editar", iconosGestion[1]);
                listaMenuExpediente.addCommand(cmdCancelar);
                //Indicamos el manejador de eventos
                listaMenuExpediente.setCommandListener(this);

                splash("Expedientes", null, imagenOpcion[0]);
                display.setCurrent(alertSplash, listaMenuExpediente);
        }
        else {
                display.setCurrent(listaMenuExpediente);
        }
}
```

```
/**
 * Muestra el menú correspondiente a consulta
 */
public void menuConsulta(){
	menuActual = "Menú Consulta";
	//Agregamos las opciones a la lista
	if (listaMenuConsulta == null){
		listaMenuConsulta = new List("Consultas",
			List.IMPLICIT);
		listaMenuConsulta.append("Nuevo", iconosGestion[0]);
		listaMenuConsulta.append("Editar", iconosGestion[1]);
		listaMenuConsulta.addCommand(cmdCancelar);
		listaMenuConsulta.setCommandListener(this);
		splash("Consultas", null, imagenOpcion[1]);
		display.setCurrent (alertSplash, listaMenuConsulta);
	}
	else {
		display.setCurrent (listaMenuConsulta);
	}
}

/**
 * Muestra el menú correspondiente a evolución
 */
public void menuEvolucion(){
	menuActual = "Menú Evolución";
	//Agregamos las opciones a la lista
	if (listaMenuEvolucion == null){
		listaMenuEvolucion = new List("Evolución",
			List.IMPLICIT);
		listaMenuEvolucion.append("Nuevo", iconosGestion[0]);
		listaMenuEvolucion.append("Editar", iconosGestion[1]);
		listaMenuEvolucion.addCommand(cmdCancelar);
		listaMenuEvolucion.setCommandListener(this);
		splash("Evolución", null, imagenOpcion[2]);
		display.setCurrent (alertSplash, listaMenuEvolucion);
	}
	else {
		display.setCurrent (listaMenuEvolucion);
	}
}
```

```
        /**
         * Muestra el menú correspondiente a listados
         */
        public void menuListado(){
                menuActual = "Menú Listados";
                //Agregamos las opciones a la lista
                if (listaMenuListados == null){
                        listaMenuListados = new List("Listados",
                                List.IMPLICIT);
                        listaMenuListados.append("Citas", iconosListado[0]);
                        listaMenuListados.append("Pacientes", iconosListado[1]);
                        listaMenuListados.addCommand(cmdCancelar);
                        listaMenuListados.setCommandListener(this);
                        splash("Listados", null, imagenOpcion[3]);
                        display.setCurrent (alertSplash, listaMenuListados);
                }
                else {
                        display.setCurrent (listaMenuListados);
                }
        }

        /**
         * Realiza la sincronización de datos
         */
        public void sincronizar(){
                splash("Sincronizar", null, imagenOpcion[4]);
                display.setCurrent (alertSplash, listaMenuPrincipal);
        }

        /**
         *  Muestra el menú correspondiente a administración
         */
        public void menuAdministrar(){
                menuActual = "Menú Administrar";
                //Agregamos las opciones a la lista
                if (listaMenuAdministrar == null){
                        listaMenuAdministrar = new List("Administrar",
                                List.IMPLICIT);
                        listaMenuAdministrar.append("Inicializar BD",
iconosGestion[2]);
                        listaMenuAdministrar.append("Cambiar Clave",
iconosGestion[3]);
```

```
                        listaMenuAdministrar.addCommand(cmdCancelar);
                        //Indicamos el manejador de eventos
                        listaMenuAdministrar.setCommandListener(this);

                        splash("Administrar", null, imagenOpcion[5]);
                        display.setCurrent(alertSplash, listaMenuAdministrar);
                }
                else {
                        display.setCurrent(listaMenuAdministrar);
                }
        }

        /**
         * Muestra la pantalla para agregar un nuevo expediente
         */
        public void nuevoExpediente(){
                PantallaExpedienteNv expediente = new PantallaExpedienteNv(display,
listaMenuPrincipal);
                try{
                        display.setCurrent(expediente.mostrar());
                }
                catch (Exception e){
                        System.out.println("Excepción: " + e.getMessage());
                }
                //Necesario para que después de finalizada la adición, la aplicación
sepa
                //que está mostrando el menú principal
                menuActual = "Menú Principal";
        }

        /**
         * Muestra la pantalla para buscar y editar expedientes
         */
        public void edicionExpediente(){
                PantallaExpedienteEd expediente = new PantallaExpedienteEd(display,
listaMenuPrincipal);
                try{
                        display.setCurrent(expediente.mostrar());
                }
                catch (Exception e){
                        System.out.println("Excepción: " + e.getMessage());
                }
                //Necesario para que después de finalizada la adición, la aplicación
sepa
```

```
                //que está mostrando el menú principal
                menuActual = "Menú Principal";
        }

        /**
         * Muestra la pantalla para confirma el borrado e inicialización de la base
de datos
         */
        public void inicializarBD(){
                IniciarDB m_iniciarDB = new IniciarDB(display, listaMenuPrincipal);
                Displayable frmBorrado = null;
                //Realizamos la conexión a la base de datos
                DBAccess m_dbaccess = new DBAccess();
                m_dbaccess.conectar();

                try{
                        //frmBorrado = m_iniciarDB.mostrar();
                        display.setCurrent(m_iniciarDB.mostrar(m_dbaccess));
                }
                catch (Exception e){
                        System.out.println("Excepción: " + e.getMessage());
                }
                /*
                if (frmBorrado != null)
                        display.setCurrent(frmBorrado);
                else
                        display.setCurrent(listaMenuPrincipal);
                */

                //Nos desconectamos de la base de datos;
                m_dbaccess.desconectar();
                m_dbaccess = null;

                //Necesario para que después de finalizada la adición, la aplicación
sepa
                //que está mostrando el menú principal
                menuActual = "Menú Principal";

        }

        /**
         * Suspende las acciones en segundo plano y libera recursos
         */
```

```
public void pauseApp(){
}

/**
 * Detiene toda actividad del midlet y libera recursos
 */
public void destroyApp(boolean incondicional){
}

/**
 * Método para controlar los comandos
 */
public void commandAction(Command c, Displayable s){
        //Salir
        if (c == cmdSalir || c == cmdCancelarInicio){
                //Finalizamos la ejecución del midlet
                destroyApp(true);
                //Notificamos al sistema que el midlet finaliza
                notifyDestroyed();
        }
        //Si se elige acepatar inicio, se verifica la clave
        else if (c == cmdAceptarInicio){
                verificarClave();
        }
        //Cancela (vuelve a la pantalla anterior)
        else if (c == cmdCancelar){
                //System.out.println("Cancelando");
                mainMenu();
        }
        else{
                List l = (List) display.getCurrent();
                //Si estamos en el menú principal
                if (menuActual=="Menú Principal"){
                        switch(l.getSelectedIndex()){
                                case 0:
                                        menuExpediente();
                                        break;
                                case 1:
                                        menuConsulta();
                                        break;
                                case 2:
                                        menuEvolucion();
```

```
                                        break;
                                case 3:
                                        menuListado();
                                        break;
                                case 4:
                                        sincronizar();
                                        break;
                                case 5:
                                        menuAdministrar();
                        }
                }
                //Si estamos en el menú expediente
                else if (menuActual=="Menú Expediente"){
                        switch(l.getSelectedIndex()){
                                case 0:
                                        nuevoExpediente();

                                        break;
                                case 1:
                                        edicionExpediente();
                        }
                }
                //Si estamos en el menú consulta
                else if (menuActual=="Menú Consulta"){
                        switch(l.getSelectedIndex()){
                                case 0:
                                        //nuevaConsulta();
                                        System.out.println("Nueva consulta");
                                        break;
                                case 1:
                                        //edicionConsulta();
                                        System.out.println("Edición de
consulta");
                        }
                }
                //Si estamos en el menú evolución
                else if (menuActual=="Menú Evolución"){
                        switch(l.getSelectedIndex()){
                                case 0:
                                        //nuevaEvolucion();
                                        System.out.println("Nueva evolución");
                                        break;
                                case 1:
```

```
                                        //edicionEvolucion();
                                        System.out.println("Edición de
evolución");
                                }
                        }
                        //Si estamos en el menú listado
                        else if (menuActual=="Menú Listados"){
                                switch(l.getSelectedIndex()){
                                        case 0:
                                                //citas();
                                                System.out.println("Citas");
                                                break;
                                        case 1:
                                                //pacientes();
                                                System.out.println("Pacientes");
                                }
                        }
                        //Si estamos en el menú administrar
                        else if (menuActual=="Menú Administrar"){
                                switch(l.getSelectedIndex()){
                                        case 0:
                                                inicializarBD();
                                                //System.out.println("Base de datos");
                                                break;
                                        case 1:
                                                //camibarClave();
                                                System.out.println("Cambio de clave");
                                }
                        }
                }
        }
}
```

Archivo: PantallaExpedienteNv.java

```
//Importamos los paquetes necesarios
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

/**
 * Clase que permite ingresar los datos de un nuevo paciente
```

```
 * @see        Expediente
 * @version    1.0 Nov 2006
 * @since      1.0 Nov 2006
 * @author     Nelson Hernández
 */
public class PantallaExpedienteNv
        implements CommandListener, ItemStateListener{
        //Formularios
        private Form frmExpediente;
        private Form frmExpediente2;

        //Elementos de interfaz que se agregan a frmExpediente
        private StringItem strFecha;
        private TextField txtApellidos;
        private TextField txtNombres;
        private ChoiceGroup grpSexo;
        private DateField txtFechaNac;
        private ChoiceGroup grpId_ESTADO_CIVIL;
        private TextField txtDireccion;
        private TextField txtTelefono;
        private TextField txtMovil;
        private TextField txtEmail;
        private ChoiceGroup grpId_TIPO_DOCUMENTO_paciente;
        private TextField txtNumeroDoc;
        private ChoiceGroup grpId_OCUPACION;
        private Command cmdCancelar;
        private Command cmdAceptar;

        //Elementos de interfaz que se agregan a frmExpediente2
        private TextField txtPadre;
        private TextField txtMadre;
        private TextField txtConyuge;
        private TextField txtResponsable;
        private TextField txtDireccionResp;
        private TextField txtTelefonoResp;
        private TextField txtNombrePDatos;
        private ChoiceGroup grpId_PARENTESCO_PDatos;
        private ChoiceGroup grpId_TIPO_DOCUMENTO_PDatos;
        private TextField txtNumeroDocPDatos;
        private Command cmdCancelar2;
        private Command cmdAceptar2;
```

```
        //Formulario padre
        private List frmParent;
        private Display display;

        //Arreglo de imágenes
        Image[] imagenMensaje;

        //Ventana para mostrar la opcion seleccionada
        private Alert alertSplash;

        /**
         * Constructor que recibe como parámetro el display del dispositivo y una
lista
         * @param pantalla Display display del dispositivo
         * @param lista List Lista del menú principal
         */
        public PantallaExpedienteNv(Display pantalla, List lista){
                frmParent = lista;
                display = pantalla;

                imagenMensaje = new Image[2];

                try{
                        //Imagenes para los mensajes
                        imagenMensaje[0] = Image.createImage("/images/error.png");
                        imagenMensaje[1] = Image.createImage("/images/guardado.png");
                }
                catch (Exception e){
                        System.out.println("Excepción: " + e.getMessage());
                }

                //Creamos la pantalla principal (un formulario)
                frmExpediente = new Form("Expediente");

                //Obtenemos la fecha actual
                Calendar cal = Calendar.getInstance();
                StringBuffer buf = new StringBuffer();
                buf.append("Fecha: ");
                buf.append(cal.get(Calendar.DAY_OF_MONTH)+"/");
                buf.append(cal.get(Calendar.MONTH)+1+"/");
                buf.append(cal.get(Calendar.YEAR));
```

```
                //Creamos los elementos de interfaz
                strFecha = new  StringItem("", buf.toString());
                txtApellidos = new TextField ("Apellidos: ", "", 50,
                       TextField.ANY);
                txtNombres = new TextField ("Nombres: ", "", 50,
                       TextField.ANY);
                String[] opcionesSexo = {"M","F"};
                grpSexo = new ChoiceGroup ("Sexo: ", Choice.EXCLUSIVE,
                       opcionesSexo, null);
                txtFechaNac = new DateField("Fecha Nacimiento:",
                       DateField.DATE);
                txtFechaNac.setDate (new Date());
                String[] opcionesEstado = {"Soltero",
                                           "Casado",
                                           "Viudo",
                                           "Acompañado"};
                grpId_ESTADO_CIVIL = new ChoiceGroup ("Estado: ", Choice.EXCLUSIVE,
                       opcionesEstado, null);
                txtDireccion = new TextField ("Dirección: ", "", 50,
                       TextField.ANY);
                txtTelefono = new TextField ("Telef: ", "", 10,
                       TextField.PHONENUMBER);
                txtMovil = new TextField ("Movil: ", "", 10,
                       TextField.PHONENUMBER);
                txtEmail = new TextField ("Email: ", "", 50,
                       TextField.EMAILADDR);
                String[] opcionesTipoDoc = {"Carnet",
                                           "DUI",
                                           "DNI",
                                           "Pasaporte",
                                           "Otro"};
                grpId_TIPO_DOCUMENTO_paciente = new ChoiceGroup ("Tipo Doc: ",
Choice.EXCLUSIVE,
                       opcionesTipoDoc, null);
                txtNumeroDoc = new TextField ("Numero Doc: ", "", 20,
                       TextField.ANY);
                String[] opcionesOcupacion = {"Jornalero",
                                           "Estudiante",
                                           "Comerciante",
                                           "Empleado",
                                           "Desempleado"
                                            };
                grpId_OCUPACION = new ChoiceGroup ("Ocupación: ", Choice.EXCLUSIVE,
```

```
        opcionesOcupacion, null);
cmdCancelar = new Command("Cancelar",Command.BACK,1);
cmdAceptar = new Command("Aceptar",Command.OK,2);

//Agregamos los elementos de interfaz al formulario
frmExpediente.append(strFecha);
frmExpediente.append(txtApellidos);
frmExpediente.append(txtNombres);
frmExpediente.append(grpSexo);
frmExpediente.append(txtFechaNac);
frmExpediente.append(grpId_ESTADO_CIVIL);
frmExpediente.append(txtDireccion);
frmExpediente.append(txtTelefono);
frmExpediente.append(txtMovil);
frmExpediente.append(txtEmail);
frmExpediente.append(grpId_TIPO_DOCUMENTO_paciente);
frmExpediente.append(txtNumeroDoc);
frmExpediente.append(grpId_OCUPACION);
frmExpediente.addCommand(cmdCancelar);
frmExpediente.addCommand(cmdAceptar);

//Indicamos los manejadores de eventos
frmExpediente.setCommandListener(this);
frmExpediente.setItemStateListener(this);

//Segunda pantalla de ingreso de datos del paciente
//Creamos la segunda pantalla de expediente
frmExpediente2 = new Form("Expediente");
//Creamos los elementos de interfaz
txtPadre = new TextField ("Padre: ", "", 50,
        TextField.ANY);
txtMadre = new TextField ("Madre: ", "", 50,
        TextField.ANY);
txtConyuge = new TextField ("Cónyuge: ", "", 50,
        TextField.ANY);
txtResponsable = new TextField ("Responsable: ", "", 50,
        TextField.ANY);
txtDireccionResp = new TextField ("Dirección Resp: ", "", 50,
        TextField.ANY);
txtTelefonoResp = new TextField ("Telef Resp: ", "", 10,
        TextField.PHONENUMBER);
txtNombrePDatos = new TextField ("Datos por: ", "", 50,
```

```
                        TextField.ANY);
                String[] parentesco = {"Madre",
                                       "Padre",
                                      "Abuela",
                                      "Abuelo",
                                       "Hermano",
                                       "Cónyuge"};
                grpId_PARENTESCO_PDatos = new ChoiceGroup ("Parentesco: ",
Choice.EXCLUSIVE,
                        parentesco, null);
                grpId_TIPO_DOCUMENTO_PDatos = new ChoiceGroup ("Tipo Doc: ",
Choice.EXCLUSIVE,
                        opcionesTipoDoc, null);
                txtNumeroDocPDatos = new TextField ("Numero Doc: ", "", 20,
                        TextField.ANY);

                cmdCancelar2 = new Command("Cancelar",Command.BACK,1);
                cmdAceptar2 = new Command("Aceptar",Command.OK,2);

                //Agregamos los elementos de interfaz al formulario
                frmExpediente2.append(txtPadre);
                frmExpediente2.append(txtMadre);
                frmExpediente2.append(txtConyuge);
                frmExpediente2.append(txtResponsable);
                frmExpediente2.append(txtDireccionResp);
                frmExpediente2.append(txtTelefonoResp);
                frmExpediente2.append(txtNombrePDatos);
                frmExpediente2.append(grpId_PARENTESCO_PDatos);
                frmExpediente2.append(grpId_TIPO_DOCUMENTO_PDatos);
                frmExpediente2.append(txtNumeroDocPDatos);
                frmExpediente2.addCommand(cmdCancelar2);
                frmExpediente2.addCommand(cmdAceptar2);

                //Indicamos los manejadores de eventos
                frmExpediente2.setCommandListener(this);
        }

        /**
         * Muestra el primer formulario para el registro de los datos del paciente
         */
        public Displayable mostrar() throws MIDletStateChangeException{

                //Devolvemos el formulario
```

```
                return frmExpediente;
        }

        //Método para controlar los cambios de fecha
        public void itemStateChanged (Item item){
        }

        /**
         * Realiza la validación de datos ingresados en el formulario frmExpediente
         * @return true si no hay problemas de validación
         * @return false si encuentran problemas en la validación
         */
        private boolean datosFrmExpedienteOK(){
                //Verificamos que no esté vacío el campo de apellidos
                if(((txtApellidos.getString()).trim()).length()==0){
                        ventanaError("Error", "Debe ingresar los apellidos",
                                imagenMensaje[0]);
                        return false;
                }
                //Verificamos que no esté vacío el campo de nombres
                else if(((txtNombres.getString()).trim()).length()==0){
                        ventanaError("Error", "Debe ingresar los nombres",
                                imagenMensaje[0]);
                        return false;
                }
                //Verificamos que la fecha de nacimiento esté dentro de un rango
válido
                else if(!fechaOK()){
                        ventanaError("Error", "Fecha de nacimiento errónea",
                                imagenMensaje[0]);
                        return false;
                }

                //Verificamos que no esté vacío el campo de dirección
                else if(((txtDireccion.getString()).trim()).length()==0){
                        ventanaError("Error", "Debe ingresar la dirección",
                                imagenMensaje[0]);
                        return false;
                }
                //Verificamos que no esté vacío el campo de numero documento
                //Si no se ha seleccionado la opción 4 (otro)
                else if(grpId_TIPO_DOCUMENTO_paciente.getSelectedIndex() !=4 &&
                        ((txtNumeroDoc.getString()).trim()).length()==0){
```

```
                    ventanaError("Error", "Debe ingresar el número de documento",
                            imagenMensaje[0]);
                    return false;
            }
            else return true;
    }

    /**
     * Verifica que la fecha esté en el rango permitido
     * @return true si la fecha está dentro del rango
     * @return false si la fecha no está dentro del rango
     */
    private boolean fechaOK(){
            //Fijamos la fecha que será permitida como inferior
            Calendar cal = Calendar.getInstance();
            cal.set(Calendar.DAY_OF_MONTH,1);
            cal.set(Calendar.MONTH,1);
            cal.set(Calendar.YEAR,1900);
            Date fechaInferior = cal.getTime();
            if((txtFechaNac.getDate()).getTime() < fechaInferior.getTime()){
                    return false;
            }
            else return true;
    }

    /**
     * Almacena los datos del paciente en la base de datos
     * @return true si se el registro se almacenó correctamente
     * @return false si hubo problema al almacenar el registro
     */
    private boolean guardar(){

            //Fecha actual
            Calendar cal = Calendar.getInstance();

            //Creamos un objeto a partir de la clase expediente
            Expediente m_expediente = new Expediente();
            //Asignamos los atributos a expediente
            m_expediente.fijarFecha(FechaDB.fechaToDB(cal.getTime()));
            m_expediente.fijarApellidos(txtApellidos.getString().toUpperCase());
            m_expediente.fijarNombres(txtNombres.getString().toUpperCase());
            m_expediente.fijarSexo(String.valueOf(grpSexo.getSelectedIndex() +
1));
```

```
            m_expediente.fijarFechaNac(FechaDB.fechaToDB(txtFechaNac.getDate()));
            m_expediente.fijarId_ESTADO_CIVIL(String.valueOf(
                    grpId_ESTADO_CIVIL.getSelectedIndex() + 1));
            m_expediente.fijarDireccion(txtDireccion.getString().toUpperCase());
            m_expediente.fijarTelefono(txtTelefono.getString());
            m_expediente.fijarMovil(txtMovil.getString());
            m_expediente.fijarEmail(txtEmail.getString());
            m_expediente.fijarId_TIPO_DOCUMENTO_paciente(String.valueOf(
                    grpId_TIPO_DOCUMENTO_paciente.getSelectedIndex() + 1 ));
            m_expediente.fijarNumeroDoc(txtNumeroDoc.getString());
            m_expediente.fijarId_OCUPACION(String.valueOf(
                    grpId_OCUPACION.getSelectedIndex() + 1 ));
            m_expediente.fijarPadre(txtPadre.getString().toUpperCase());
            m_expediente.fijarMadre(txtMadre.getString().toUpperCase());
            m_expediente.fijarConyuge(txtConyuge.getString().toUpperCase());

        m_expediente.fijarResponsable(txtResponsable.getString().toUpperCase());

        m_expediente.fijarDireccionResp(txtDireccionResp.getString().toUpperCase());
            m_expediente.fijarTelefonoResp(txtTelefonoResp.getString());

        m_expediente.fijarNombrePDatos(txtNombrePDatos.getString().toUpperCase());
            m_expediente.fijarId_PARENTESCO_PDatos(String.valueOf(
                    grpId_PARENTESCO_PDatos.getSelectedIndex() + 1 ));
            m_expediente.fijarId_TIPO_DOCUMENTO_PDatos(String.valueOf(
                    grpId_TIPO_DOCUMENTO_PDatos.getSelectedIndex() + 1 ));
            m_expediente.fijarNumeroDocPDatos(txtNumeroDocPDatos.getString());

            //Guardamos los datos del paciente
            if(m_expediente.guardar() != 0){
                    System.out.println("Expediente guardado");
            }
            else System.out.println("Hubo un error al guardar");
            //System.out.println("guardando");
            m_expediente = null;
            return true;
        }

        private int obtenerNumero(TextField m_campo){
            int m_numero = 0;
                    try{
                            m_numero = Integer.parseInt(m_campo.getString());
                    } catch (Exception e) {
```

```
                        System.out.println(e.getMessage());
                }
        return m_numero;
}

private int obtenerCadena(TextField m_campo){
        int m_numero = 0;
                try{
                        m_numero = Integer.parseInt(m_campo.getString());
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }
        return m_numero;
}

/**
 * Muestra una pantalla de indicando error
 * @param strTitulo String título de la pantalla
 * @param strMensaje String mensaje a mostrar
 * @param imagen Image imagen a desplegar en la pantalla
 */
public void ventanaError(String strTitulo, String strMensaje,
        Image imagen){
        alertSplash = new Alert (strTitulo, strMensaje,
                imagen, AlertType.ERROR);
        display.setCurrent(alertSplash);
}

/**
 * Muestra una pantalla de indicación
 * @param strTitulo String título de la pantalla
 * @param strMensaje String mensaje a mostrar
 * @param imagen Image imagen a desplegar en la pantalla
 */
public void splash(String strTitulo, String strMensaje,
        Image imagen){
        alertSplash = new Alert (strTitulo, strMensaje,
                imagen, AlertType.INFO);
        //La ventana esperará 3 segundos
        alertSplash.setTimeout(3000);
}
```

```
        /**
         * Controla los eventos de comando
         * @param c Command comando seleccionado
         * @param s Displayable display del dispositivo
         */
        public void commandAction(Command c, Displayable s){
                //Salir
                if (c == cmdCancelar || c == cmdCancelar2){
                        //Mostramos en pantalla el formulario padre
                        display.setCurrent(frmParent);
                }
                //Si elegimos aceptar
                else if (c == cmdAceptar){
                        //Mostramos en pantalla el segundo formulario de ingreso
                        if (datosFrmExpedienteOK())
                                display.setCurrent(frmExpediente2);
                }
                //Si elegimos aceptar
                else if (c == cmdAceptar2){
                        guardar();
                        display.setCurrent(frmParent);
                }
        }
}
```

3. Manual de Usuario

MovilMed

Manual de Usuario

Por: Nelson Hernández

3.1. Iniciando MovilMed

Para iniciar la aplicación, hay que buscar el nombre de la misma, el cual es ***MovilMed***. Aparecerá entonces una pantalla semejante a la mostrada en la figura 3.1, pidiendo el ingreso de la clave de la aplicación.

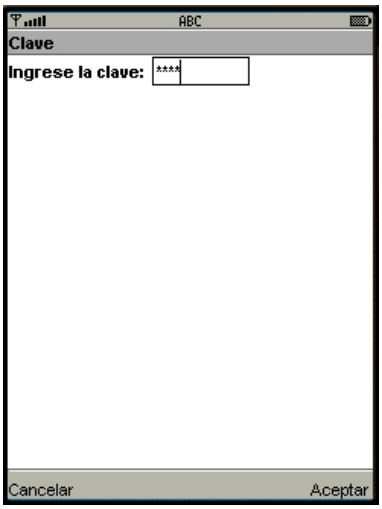
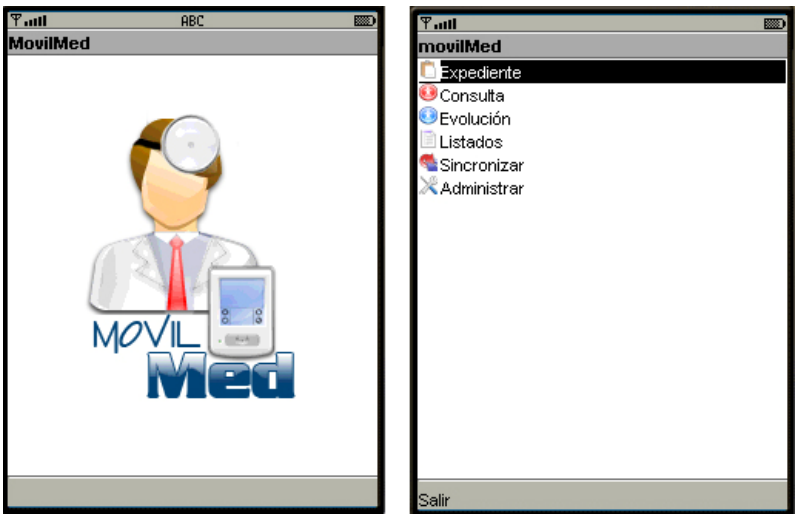


Figura 3.1 Ingreso de clave de la aplicación

Si es la primera vez que se ejecuta la aplicación, se debe ingresar la clave 0000, misma que es posible cambiar, tal como se detallará posteriormente. Luego seleccionar ***Aceptar***.

Si la clave ingresada es correcta, se mostrará una pantalla con el ícono de la aplicación (ver figura 3.2 a) y luego el menú principal de la aplicación (ver figura 3.2 b), caso contrario, se mostrará una pantalla indicando el error (ver figura 3.3).



(a) (b)

Figura 3.2 Inicio de la aplicación. (a) Icono de la aplicación, (b) Menú principal

Figura 3.3 Pantalla de error por clave incorrecta

3.2. Menú Expediente

A través del menú ***Expediente*** es posible adicionar y modificar los expedientes de los pacientes, al seleccionarlo, aparecerá otra pantalla con las opciones ***Nuevo*** y ***Editar*** (ver figura 3.4).

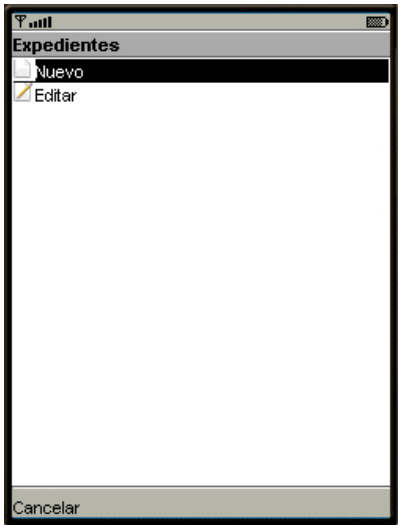


Figura 3.4 Menú de adición y modificación

3.2.1. Nuevo Expediente

Al seleccionar la opción ***Nuevo*** de la pantalla adición y modificación (figura 3.4), aparecerá otra pantalla para agregar un nuevo expediente (ver figura 3.5).

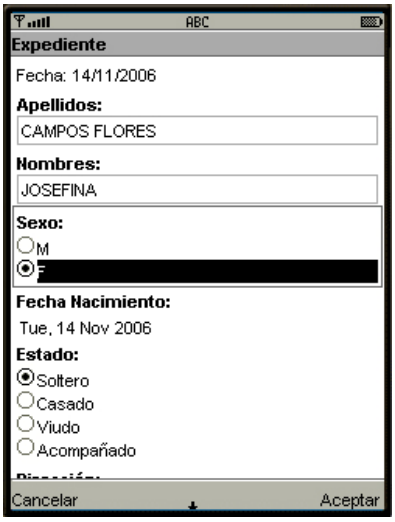


Figura 3.5 Pantalla de adición de expediente

Para ingresar la fecha de nacimiento del paciente, hay que seleccionar el campo respectivo y entonces aparecerá un calendario como el mostrado en la figura 3.6, donde es posible seleccionar el año, mes y día de nacimiento.

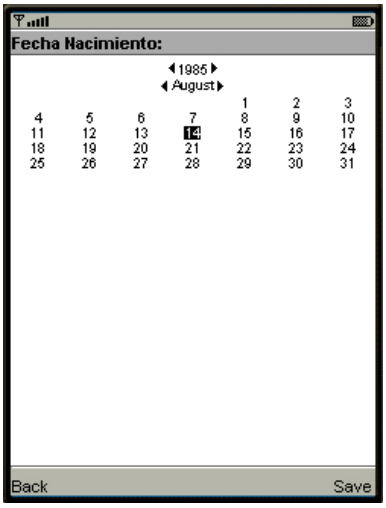


Figura 3.6 Selección de fecha

Después de seleccionada la fecha apropiada, se debe dar clic en ***Guardar***, entonces se desplegará nuevamente la pantalla de adición de expediente (figura 3.5).

Una vez llenos los campos, se deberá seleccionar ***Aceptar***, entonces se desplegará otra pantalla (figura 3.7) donde se debe terminar de llenar los datos relacionados con el paciente. Hay campos que es obligatorio llenarlos, si se trata de dejarlos vacíos, aparecerá una mensaje indicando error (figura 3.8).

Figura 3.7 Pantalla 2 de adición de expediente

Figura 3.8 Mensaje de error en ingreso de datos

Al finalizar se deberá seleccionar ***Aceptar***, entonces aparecerá un mensaje (figura 3.9) indicando que el registro ha sido guardado.

Figura 3.9 Mensaje de registro guardado

3.2.2. Editar Expediente

Al seleccionar la opción ***Editar*** de la pantalla adición y modificación (figura 3.4) aparecerá otra pantalla para seleccionar el expediente (ver figura 3.10), en esta pantalla se puede ingresar el número de expediente y/o los apellidos y/o nombres del paciente.

Figura 3.10 Pantalla de selección de expediente

Si se ingresa el número de expediente y éste se encuentra en la base de datos, se mostrará el expediente respectivo en la pantalla de edición de expedientes (figura 3.11). En esta pantalla es posible modificar o eliminar los datos del paciente, para modificar los datos, una vez hechos los cambios deseados, se deben seleccionar ***Aceptar*** y entonces aparecerá otra pantalla para continuar la modificación (figura 3.12), y finalmente se debe seleccionar ***Aceptar***, entonces aparecerá un mensaje semejante al mostrado en la figura 3.9. Si lo que se desea es borrar el registro, habrá que seleccionar la opción ***Eliminar***, entonces aparecerá una pantalla (figura 3.13).

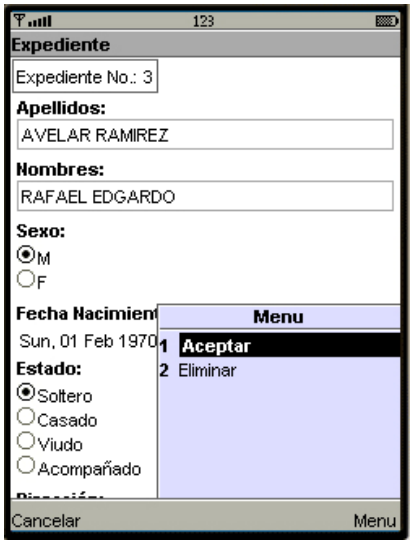


Figura 3.11 Pantalla de edición de expediente

Figura 3.12 Pantalla 2 de edición de expediente

Figura 3.13 Mensaje de eliminación de registro

Al no ingresar número de expediente, si no que los apellidos y/o nombres, y si se encuentran registros que coincidan, aparecerá una lista con dichos registros (figura 3.14), en esta lista es posible seleccionar el expediente que se desea modificar, al seleccionarlo aparecerá una pantalla como la mostrada en la figura 3.11.

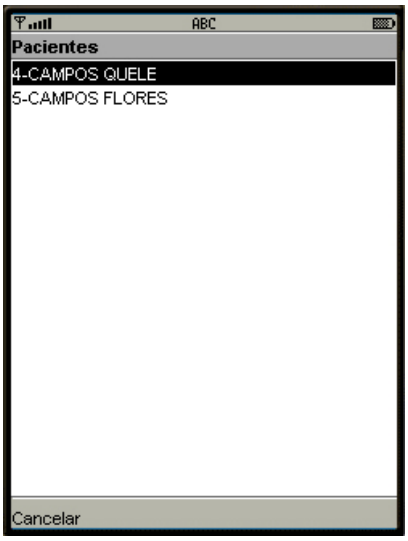


Figura 3.14 Lista de expedientes

3.3. Menú Consulta

A través del menú ***Consulta*** es posible adicionar y modificar las consultas de los pacientes, al seleccionarlo, aparecerá otra pantalla con las opciones ***Nuevo*** y ***Editar*** semejantes a las mostradas en la figura 3.4.

3.3.1. Nueva Consulta

Al seleccionar la opción ***Nuevo*** de la pantalla adición y modificación (figura 3.4), aparecerá otra pantalla de búsqueda del paciente (semejante a la mostrada en la figura 3.10) al que se agregará la consulta. Una vez seleccionado el paciente al que se le asignará la consulta, aparecerá una pantalla semenjante a la mostrada en la figura 3.15. Para guardar el registro, se debe seleccionar ***Menú*** y luego ***Aceptar***. Aparecerá entonces un mensaje similar al de la figura 3.9 indicando que el registro se ha guardado.

Figura 3.15 Pantalla de adición de consulta

3.3.2. Editar Consulta

Al seleccionar la opción ***Editar*** de la pantalla adición y modificación (figura 3.4), aparecerá otra pantalla de búsqueda del paciente similar a la mostrada en la figura 3.10. Una vez seleccionado el paciente, aparecerá entonces una ventana similar a la mostrada en la figura 3.15 donde es posible modificar o eliminar la consulta.

3.4. Menú Evolución

A través del menú ***Evolución*** es posible adicionar y modificar la evolución de la enfermedad de los pacientes, al seleccionarlo, aparecerá otra pantalla con las opciones ***Nuevo*** y ***Editar*** semejantes a las mostradas en la figura 3.4.

3.4.1. Nueva Evolución

Al seleccionar la opción ***Nuevo*** de la pantalla adición y modificación (figura 3.4), aparecerá otra pantalla de búsqueda del paciente (semejante a la mostrada en la figura 3.10) al que se agregará la evolución. Una vez seleccionado el paciente al que se le asignará la evolución, aparecerá una pantalla semenjante a la mostrada en la figura 3.16. Para guardar el registro, se debe seleccionar ***Menú*** y luego ***Aceptar***. Aparecerá entonces un mensaje similar al de la figura 3.9 indicando que el registro se ha guardado.

Figura 3.16 Pantalla de adición de evolución

3.4.2. Editar Evolución

Al seleccionar la opción ***Editar*** de la pantalla adición y modificación (figura 3.4), aparecerá otra pantalla de búsqueda del paciente similar a la mostrada en la figura 3.10. Una vez seleccionado el paciente, aparecerá entonces una ventana similar a la mostrada en la figura 3.16 donde es posible modificar o eliminar la evolución.

3.5. Menú Listados

A través de este menú es posible obtener listados de citas y de pacientes. Cuando se selecciona una opción (ya sea citas o listados), aparece una pantalla (figura 3.17) solicitando el ingreso del rango de fechas para el cual se desea el listado. Una vez hecho esto, aparece una pantalla con el listado respectivo, semejante al mostrado en la figura 3.14.

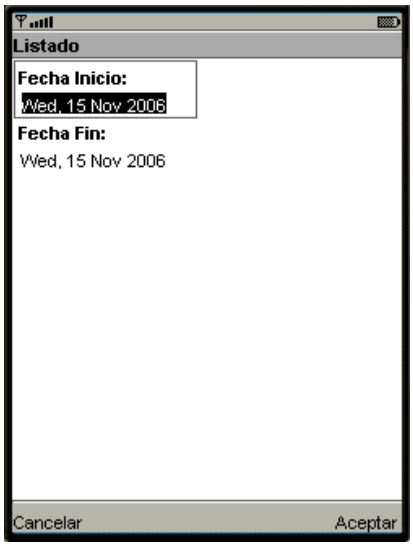


Figura 3.17 Pantalla fechas listado

3.6. Menú Sincronizar

Esta opción permite realizar la sincronización entre el dispositivo móvil y un servidor. Después de realizado el proceso, aparecerá un mensaje semejante al mostrado en la figura 3.18.

Figura 3.18 Pantalla de sincronización

3.7. Menú Administrar

Al seleccionar está opción, aparecerá otra pantalla con las opciones ***Inicializar DB*** y ***Cambiar Clave*** (ver figura 3.19).

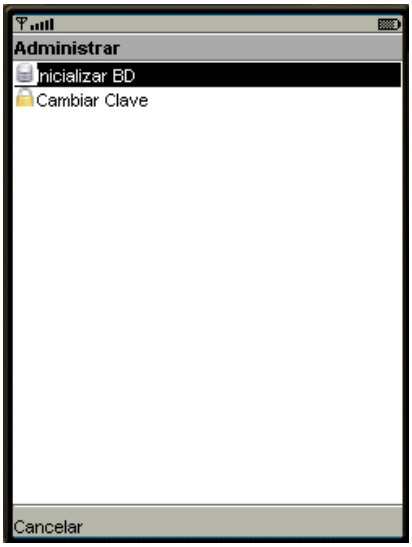


Figura 3.19 Pantalla de administración

3.7.1. Inicializar DB

Esta función permite borrar la base de datos, se recomienda utilizarla solamente cuando se instala por primera vez la aplicación. Si se selecciona esta opción, habiendo ya ingresado registros en la base de datos, se mostrará un mensaje de advertencia, semejante al mostrado en la figura 3.20.

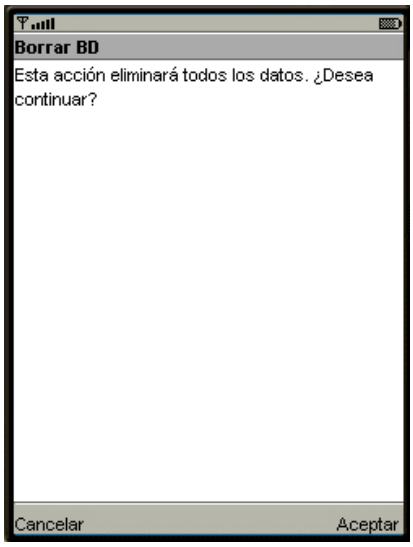


Figura 3.20 Confirmación de inicialización de la base de datos

3.7.2. Cambiar Clave

Por defecto la aplicación se incluye con la clave 0000, sin embargo es posible cambiar dicha clave a través de la opción cambiar clave. Al seleccionar esta opción, se muestra una pantalla semejante a la mostrada en la figura 3.21.

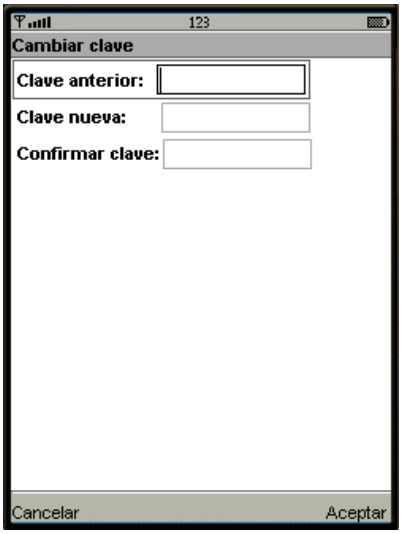


Figura 3.21 Pantalla para cambiar la clave

4. Manual de Instalación

MovilMed

Manual de Instalación

Por: Nelson Hernández

La forma más estandarizada de instalar MIDlets en dispositivos móviles, es a través de sitios de Web, sin embargo también es posible la instalación directa por cable, por infrarrojos o por otras tecnologías semejantes.

4.1. Requerimientos de instalación

- Soporte para le ejecución de Midlets
- CLDC 1.0
- MIDP 2.0
- 500 KB de memoria libres

4.2. Instalación a Través de Internet

Para instalar la aplicación prototipo MovilMed a través de Internet, se necesitan los siguientes archivos.

ARCHIVO	DESCRIPCION
movilMed.htm	Página web que contien un enlace al archivo descriptor del MIDlet.
movilMed.jad	Archivo que describe la aplicación contenida en movilMed.jar
movilMed.jar	Archivo empaquetado que contiene la aplicación completa

Contenido del archivo movilMed.htm

```
<HTML>
<A href="movilMed.jad">Instalar movilMed</A>
</HTML>
```

Tal como puede apreciarse en el código anterior, lo único que se necesita es un enlace al archivo *movilMed.jad*.

Contenido del archivo movilMed.jad

```
MIDlet-1: movilMed, movilMed.png, movilMed
```

```
MIDlet-Jar-Size: 327680
MIDlet-Jar-URL: http://servidor/movilMed.jar
MIDlet-Name: movilMed
MIDlet-Vendor: Nelson Hernández
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

Tal como puede apreciarse en el contenido del archivo movilMed.jad, existe una línea que dice `MIDlet-Jar-URL: http://servidor/movilMed.jar`, esta línea le indica al dispositivo la ubicación del archivo *movilMed.jar*, para que pueda descargarlo. En este caso la aplicación estaría en un sitio llamado *servidor*.

Pasos para la instalación

1. Navegar en la página web que incorpora el enlace al archivo *movilMed.jad*
2. Seleccionar en enlace para descargar el archivo
3. Descargar el arhivo *movilMed.jad*
4. Confirmar instalación del MIDlet *movilMed.jar*

4.3. Instalación Directa

Esta forma de instalación es más dependiente del dispositivo, es decir, éste debe proporcionar una forma de sincronización con una computadora de escritorio. En ese sentido, basta con transferir los archivos *movilMed.jad* y *movilMed.jar* al dispositivo.

En el caso de instalación en PDAs con el sistema operativo Palm, primero es necesario convertir la aplicación, empleando una utilieria llamada *midp4palm* y luego copiar, utilizando la herramienta *Hot Sync* que se incluye con los dispositivos palm, tanto el archivo *movilMed.prc* generado como el archivo *MIDP.prc* que se incluye con el *midp4palm*.

4.4. Desinstalación

La desinstalación de las aplicaciones es muy sencilla, cada dispositivo provee sus propias utilidades, por lo general seleccionando la aplicación y activando sus opciones, entre las cuales se presenta la de eliminar, esta opción permite eliminar tanto la aplicación como su área de datos.

Conclusiones y Trabajo Futuro

Conclusiones

- Las tecnologías para la programación de aplicaciones para dispositivos móviles, en su grado de desarrollo actual, permiten emplear metodologías semejantes a las aplicadas en el desarrollo de aplicaciones de computadoras de escritorio, es decir, con todas sus etapas desde los requerimientos hasta el diseño, pasando por análisis y pruebas; no haciéndose necesario el aprendizaje de nuevas metodologías.
- J2ME es un lenguaje muy adecuado para programar aplicaciones para dispositivos móviles, sin embargo, algunas especificaciones de los perfiles no son restrictivas, lo que puede derivar en problemas de portabilidad.
- Las bases de datos basadas en Java permiten mayor flexibilidad cuando se emplea J2ME, como es el caso de PointBase.

Trabajo Futuro

- Desarrollar una aplicación de escritorio que extienda la funcionalidad de la aplicación móvil y que emplee la base de datos con la cual se realiza la sincronización. De forma tal que un médico pueda emplear la aplicación móvil para recabar la información cuando realice las visitas a los pacientes y emplear la aplicación de escritorio cuando atienda a los pacientes en su consultorio.
- Desarrollar aplicaciones soportadas por tecnologías similares a J2ME, como HawHaw y SuperWaba.

Apéndice A

Tecnologías de programación de aplicaciones con J2ME

Java es una plataforma de software desarrollada por Sun Microsystems. Su principal característica es que está pensada de tal manera que los programas creados en ella puedan ejecutarse sin cambio alguno en diferentes tipos de arquitecturas y dispositivos computacionales.

Actualmente Java tiene en el mercado tres paquetes con los que cubre una amplia gama de dispositivos. Cada uno tiene muy claramente marcado su campo de actuación. De esta manera Sun puede crear ediciones mejores y más preparadas, dotando a cada edición de las herramientas específicas que necesita y librándola de las que no necesita para su función. Las ediciones disponibles son:

- J2EE (Java 2 Enterprise Edition)
- J2SE (Java 2 Standard Edition)
- J2ME (Java 2 Micro Edition)

J2EE

Es una edición enfocada a dar cobertura a los servidores de empresa. Se necesitaba un producto software distribuido, que pudiera compartir fácilmente la información contenida en diferentes ubicaciones físicas, una edición con un mayor número de opciones para las redes, se necesita un lenguaje que pudiera crear los que se conocen como “enterprise applications” (Programas de empresa). Para ello Sun decidió formar la J2EE.

J2SE

Es la edición más conocida de Java. Es un conjunto de herramientas y APIs que permiten desarrollar applets y otro tipo de aplicaciones. Está especialmente enfocada a cubrir las necesidades de las aplicaciones de Internet que corren en las computadoras personales y las estaciones de trabajo.

J2ME

Esta es la edición enfocada en la programación de dispositivos móviles. Se trata de una apuesta de Sun por abarcar y solucionar los problemas de dispositivos con pocos recursos: poca memoria, con limitaciones computacionales, display pequeños. Para ello creo J2ME que resume la funcionalidad de J2SE adaptándola a los requisitos mínimos necesarios para dispositivos móviles, inalámbricos, PDAs.

En la figura A.1 se representan las tres ediciones de Java 2 según su tamaño y opciones en común.

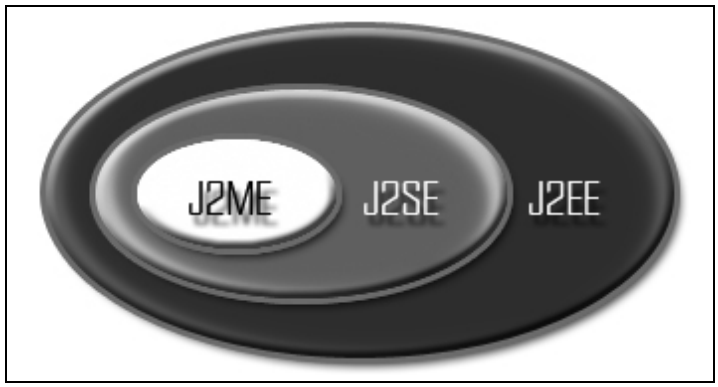


Figura A. 1 Ediciones de Java 2

Para una mejor comprensión, la figura A.2 muestra la estructura de la tecnología Java, de abajo hacia arriba, se puede observar el hardware y el sistema operativo de cada uno de las máquinas o dispositivos, que es totalmente dependiente del fabricante y completamente transparente a los programadores Java. Por encima se sitúa la capa correspondiente a la Máquina Virtual de Java (JVM), que se encarga de ejecutar los bytecodes de las aplicaciones Java. Sobre la capa de la JVM se encuentra el API base de Java, consistente en todas las clases y librerías de clases comunes a todas las ediciones. También se puede apreciar que J2ME se divide en dos configuraciones: la Configuración para Dispositivos Conectados (CDC) y la Configuración para Dispositivos con Conexión Limitada (CLDC), así mismo existen perfiles, los cuales se apoyan en las configuraciones, actualmente el perfil más importante para dispositivos móviles es el Perfil para Dispositivos de Información Móvil (MIDP).

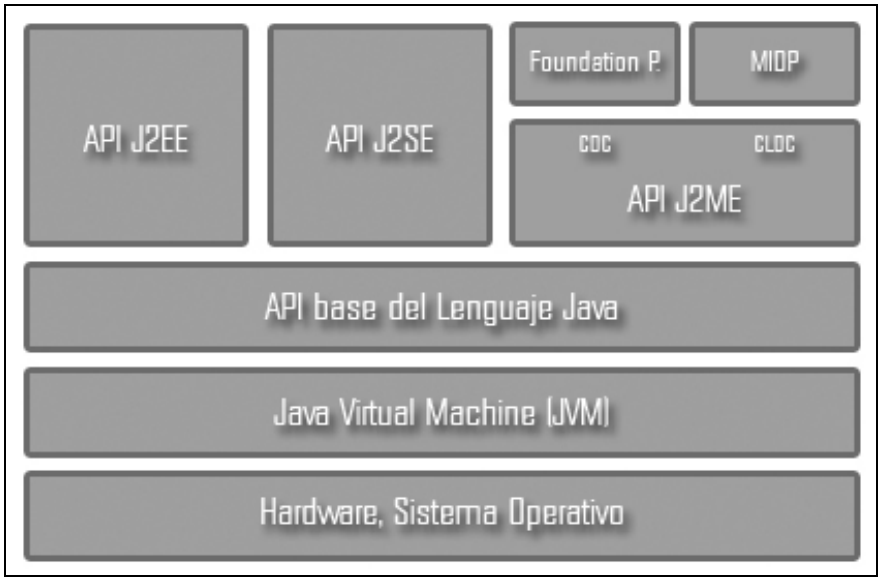


Figura A. 2 Tecnología Java

A continuación se verán las diferencias básicas entre J2ME y J2SE:

Tipos de datos soportados

- J2ME soporta un subconjunto de los tipos de datos de J2SE.
- Los tipos float y double no son soportados por que la mayoría de dispositivos CLDC no tienen unidad de coma flotante y porque es una operación muy costosa.

Preverificación (dentro y fuera del dispositivo on-line y off-line)

- En J2ME parte de la verificación se realiza fuera del dispositivo (off-line) de esta manera se reduce la carga de la máquina, el resto de la verificación sí se realiza en el dispositivo (on-line).

Archivos especiales

- En J2ME es necesaria la inclusión de unos archivos especiales que contienen información de las aplicaciones que incluyen. Estos archivos se denominan manifiesto (.mf) y descriptor (.jad).

Nueva librería gráfica (javax.microedition.lcdui)

- Mediante el paquete lcdui J2ME define un nuevo conjunto de clases para la creación de interfaces gráficas. Estas clases están adaptadas a dispositivos con memorias muy limitadas y pantallas de tamaño reducido.

Ausencia del main

- En J2ME no existe una función main como punto de entrada para la ejecución del programa. En su caso existe el método startApp().

Desaparición del Garbage Collector

- En J2ME no existe el Garbage Collector (recolector de basura). De esta manera se reduce la utilización de memoria. Por ello es necesario eliminar de forma explícita todos aquellos elementos que no vayan a ser utilizados más.

Arquitectura de J2ME

J2ME al igual que un ambiente completo de Java (Java Runtime Environment) abarca una Máquina Virtual, Configuraciones, Perfiles y Paquetes adicionales opcionales; la figura A.3 muestra la arquitectura de J2ME para dispositivos móviles.

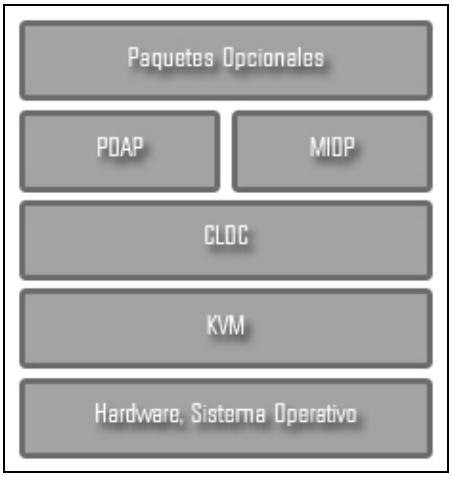


Figura A. 3 Arquitectura J2ME

KVM

La máquina virtual de Java (JVM), es un programa encargado de interpretar código intermedio (bytecode) de los programas Java precompilados a código máquina (que pueda ser ejecutado por la plataforma), realizar las llamadas pertinentes al sistema operativo y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo.

Para J2ME existen dos configuraciones CLDC y CDC, cada una con unas características propias. CLDC es la utilizada para PDAs, organizadores personales, buscapersonas, teléfonos móviles. Y CDC es la utilizada para decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos y sistemas de navegación en automóviles. Como consecuencia, cada una requiere su propia máquina virtual. La VM (Máquina Virtual o Virtual Machine) de la configuración CLDC se denomina KVM y la de la configuración CDC se denomina CVM.

Características de KVM (Kilobyte Virtual Machine)

El objetivo de la tecnología KVM es el de crear la máquina virtual de java más pequeña y completa que mantenga los aspectos más importantes del lenguaje de programación Java y que funcione en dispositivos de recursos limitados y con conexión a red.

Sus principales características son:

- Altamente portable.
- Compacta.
- Con una carga de memoria entre los 40Kb y los 80Kb.
- Para procesadores de 16 o 32 bits y capacidades de unos pocos cientos de Kilo Bytes (originalmente 128 KB de memoria).

Limitaciones de KVM con respecto a JVM

- No hay soporte para tipos en coma flotante (no existen por tanto los tipos double ni float).

- No existen cargadores de clases (class loaders) definidos por el usuario. Sólo existen los predefinidos.
- No se permiten los grupos de hilos o hilos daemon (Cuándo se quiera utilizar grupos de hilos se deben utilizar los objetos Colección para almacenar cada hilo en el ámbito de la aplicación).
- No existe soporte para JNI (Java Native Interface) debido a los recursos limitados de memoria.
- No existe la finalización de instancias de clases. No existe el método Object.finalize().
- No hay referencias débiles.
- Limitada capacidad para el manejo de excepciones debido a que el manejo de éstas depende en gran parte de las APIs de cada dispositivo por lo que son éstos los que controlan la mayoría de las excepciones.

Librerías CLDC

CLDC es un tipo de configuración. Una configuración define el ambiente de ejecución básico, es decir un conjunto de clases principales y una máquina virtual especifica que están orientadas a conformar el corazón de las implementaciones para dispositivos con características específicas.

La CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, procesamiento y memoria. La mayoría de las restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño. Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

- Disponer entre 160Kb y 512Kb de memoria total disponible. Mínimo 128Kb de memoria no volátil para la KVM y librerías CLDC, y 32Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.
- Procesador de 16 o 32 bits con al menos 25Mhz de velocidad.
- Tener conexión a algún tipo de red.

Así mismo las librerías CLDC aportan una serie de funcionalidades a los dispositivos:

- Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
- Un subconjunto de las librerías Java del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad.

Perfiles (PADP y MIDP)

El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Se debe tener en cuenta que un perfil siempre se construye sobre una configuración determinada, de este modo, se puede pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad específica.

Las aplicaciones desarrolladas sobre un determinado perfil van a ser portables a cualquier dispositivo que soporte ese perfil; cabe destacar que un dispositivo puede soportar varios perfiles y que sobre una configuración pueden existir diversos perfiles.

En J2ME hasta el momento se han definido los siguientes perfiles que se pueden clasificar en dos grupos:

1. Construidos sobre la CDC
2. Construidos sobre la CLDC

Perfiles de J2ME construidos sobre la CDC

Foundation Profile: Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital.

Personal Profile: El Personal Profile está construido sobre la CDC y es un subconjunto de la plataforma J2SE v1.3, que proporciona un entorno con un completo soporte gráfico

AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades Web y soporte de applets Java. Este perfil requiere una implementación del Foundation Profile.

RMI Profile: Este perfil también está construido sobre la CDC y requiere que una implementación del Foundation Profile se construya debajo de él.

Perfiles de J2ME construidos sobre la CLDC

PDA Profile: El PDA Profile está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x 100 pixels) con un factor 2:1. Actualmente este perfil se encuentra en fase de definición.

MID Profile: Está construido sobre la CLDC y es el perfil más usado actualmente para el desarrollo de aplicaciones para dispositivos móviles.

MIDP 1.0 y 2.0

MIDP es el acrónimo de Perfil para dispositivos de información móvil (Mobile Information Device Profile) se apoya en CLDC. MIDP fue el primer perfil desarrollado, en su desarrollo participan empresas tales como Motorola, Ericcson, Samsung, Sony, Nokia, NTT DoCoMo, Palm, Siemens, Sun Microsystems, Mitsubishi, Symbian, AOL. Hasta el momento se han definido dos perfiles, el MIDP 1.0 y el MIDP 2.0, cuyas especificaciones finales salieron en septiembre de 2000 y noviembre de 2002 respectivamente.

MIDP 1.0

A continuación se describen las características de este perfil:

Hardware

Diseñado para dispositivos con las siguientes características mínimas:

- Poca capacidad computacional y de memoria
- Conectividad limitada (en torno a 9600bps).
- Capacidad gráfica muy reducida (mínimo un display de 96x54 píxeles monocromo).
- Entrada de datos alfa numérica reducida.

- 128Kb de memoria no volátil para componentes MIDP.
- 8Kb de memoria no volátil para datos persistentes de aplicaciones.
- 32Kb de memoria volátil en tiempo de ejecución para la pila Java.

APIs

El MIDP 1.0 incluyó solo las APIs más básicas, aquellas que eran consideradas como requerimiento absoluto para alcanzar una amplia portabilidad.

Estas APIs están relacionadas con:

- Aplicación (Definición de la semántica y control de la aplicación MIDP).
- Interfaz de usuario (manejo de entradas y despliegues).
- Almacenamiento persistente.
- Comunicaciones en red.
- Temporizadores (Timers).

MIDP 2.0

Gracias al avance tecnológico hoy en día se cuenta con dispositivos móviles mucho más completos y funcionales. Ahora existen terminales con más memoria, pantallas a color mejoradas, soporte para multimedia, cámara fotográfica e incluso cámara de video, acceso a Internet. Por todo ello era necesaria una nueva versión de MIDP para sacarle provecho a estos avances.

A continuación se describen las características de este perfil:

Hardware

Características mínimas:

PANTALLA

- Tamaño: 96x54
- Colores: 1-bit (blanco y negro)
- Forma del Pixel (proporción de aspecto): Aprox. 1:1

ENTRADA

- Teclado de una mano o teclado de dos manos o pantalla táctil.

MEMORIA

- 256 KB de memoria no volátil para la implementación de MIDP, además de la necesaria para CLDC.
- 8 KB para memoria no volátil para datos persistentes para que las aplicaciones puedan guardar datos.
- 128 KB de memoria volátil, para la ejecución de la aplicación.

SONIDO

- Debe poder reproducir tonos, vía hardware o con un algoritmo software.

REDES

- Dos vías, acceso inalámbrico posiblemente intermitente con ancho de banda limitado.

APIs

Al igual que MIDP 1.0, la versión 2.0 de este perfil solo incluyó las APIs consideradas como requerimiento indispensable para asegurar la portabilidad de las aplicaciones, pero añadió las necesarias para estar acorde con los nuevos avances existentes:

- Manejo del ciclo de vida de las Aplicaciones (Definición de la semántica de una aplicación MIDP y cómo ésta es controlada).
- Firmado de aplicaciones y seguridad para dominios privilegiados.
- Transacciones seguras entre usuarios finales (https).
- Registro de aplicaciones push (modelo de servidor push).
- Interconexión a redes.
- Almacenamiento persistente.
- Sonido.

- Temporizadores.
- Interfaces de usuario mejoradas (que incluye despliegue, entradas y soporte para juegos).

En la tabla A.1 se presenta un resumen de los tipos de paquetes que define los perfiles MIDP en sus dos versiones.

TIPO	PAQUETE MID 2.0	PAQUETE MID 1.0
Interfaz de usuario	javax.microedition.lcdui	javax.microedition.lcdui
Juegos	javax.microedition.lcdui.game	-
Ciclo de vida	javax.microedition.midlet	javax.microedition.midlet
Interconexión a redes	javax.microedition.io	javax.microedition.io
Seguridad	javax.microedition.pki	-
Sonido	javax.microedition.media	-
	javax.microedition.media.control	-
Persistencia de datos	javax.microedition.rms	javax.microedition.rms
Básicos o del núcleo	javax.microedition.io	javax.microedition.io

Tabla A. 1 Paquetes de los perfiles MID

Actualmente existen en el mercado diversas herramientas de software que permiten desarrollar aplicaciones para dispositivos móviles empleando el lenguaje J2ME. Para implementar una aplicación MIDP (Midlet) se necesita de una serie de herramientas que se pueden agrupar en las tres categorías siguientes:

- SDK para Java
- Un emulador para aplicaciones MIDP
- Un IDE (opcional)

SDK

Un SDK es un Kit de desarrollo de software. Se utilizará el provisto por Sun, el J2SE SDK. Éste paquete sirve para desarrollar programas Java y proporciona el entorno de ejecución necesario para las aplicaciones. Es necesaria la instalación de éste producto en primer lugar, ya que el resto de herramientas se apoyan en él. Está disponible para las plataformas Windows, Linux y Solaris.

Emulador MIDP

Antes de subir las aplicaciones al móvil para probarlas es necesario tener una herramienta de preverificación en la computadora que compruebe si funciona correctamente. Ésa es, principalmente, la razón de existir de los emuladores MIDP. En el mercado es posible encontrar un gran número de ellos.

Se empleará el J2ME Wireless Toolkit que es un potente emulador provisto por Sun, que además trae un completo paquete de aplicaciones entre las que se puede encontrar un sencillo pero eficaz IDE que facilita la tarea de crear y compilar las aplicaciones MIDP.

IDE

Un IDE, relacionado con programación es un Entorno Integrado de Desarrollo (Integrated development environment). No es una herramienta indispensable, pero puede ser de gran ayuda al escribir Midlets.

En otras palabras, al tener instalado un IDE lo que se obtiene es un conjunto de herramientas que facilitan el desarrollo. Alguna de estas herramientas son: un editor de texto, un buscador de clases o browser de clases, compilación a través de menús.

Se empleará JEdit, éste es un editor de texto orientado a la programación. Además incluye herramientas de edición, gestión de archivos e incorpora su propio lenguaje de macros. Está hecho en Java y es gratuito. Para trabajar en JEdit se necesita tener instalado el Java Runtime Environment 1.3 (o posterior).

J2SDK

Se trabajará con Java 2 Edición estándar SDK (J2SE SDK), se detallará la instalación bajo el sistema operativo Windows XP.

Descarga

1. Hay que descargar el instalador de:

   http://java.sun.com/j2se/1.4.2/download.html y elegir la versión apropiada, para el caso J2SE v 1.4.2_12 SDK

2. Aceptar la licencia

3. Descargar la versión Windows Offline Installation, la cual provee el J2SDK completo

Instalación

1. Iniciar la instalación dando doble clic en el icono de la aplicación.
2. Aparecerá una ventana mostrando el contrato de licencia, seleccionar I accept the terms in the license agreement y luego dar clic en Next.
3. En la siguiente ventana, es posible seleccionar las características de la instalación, incluyendo el directorio, se recomienda no cambiar nada y dar clic en Next.
4. En la siguiente ventana sobre el registro de Plug-In en los navegadores web, se recomienda no cambiar nada y dar clic en Install.
5. Esperar a que termine de copiar todos los archivos. Este proceso tomará varios minutos.
6. Cuando termine, aparecerá un mensaje indicando que la instalación de Java 2 SDK fue exitosa, dar clic en finish, reiniciar la computadora, de ser necesario.

Configuración

Verificación de la configuración

1. Abrir el MSDOS o Símbolo del Sistema. Normalmente se encuentra en: Menú Inicio / Programas / Accesorios. También se puede abrir desde: Menú Inicio / Ejecutar y escribir cmd o command.
2. Una vez hecho esto, se abrirá la ventana de símbolo del sistema. En él escribir: javac.

El JDK estará correctamente configurado si al dar Enter, se obtiene un mensaje similar al mostrado en la figura A.4

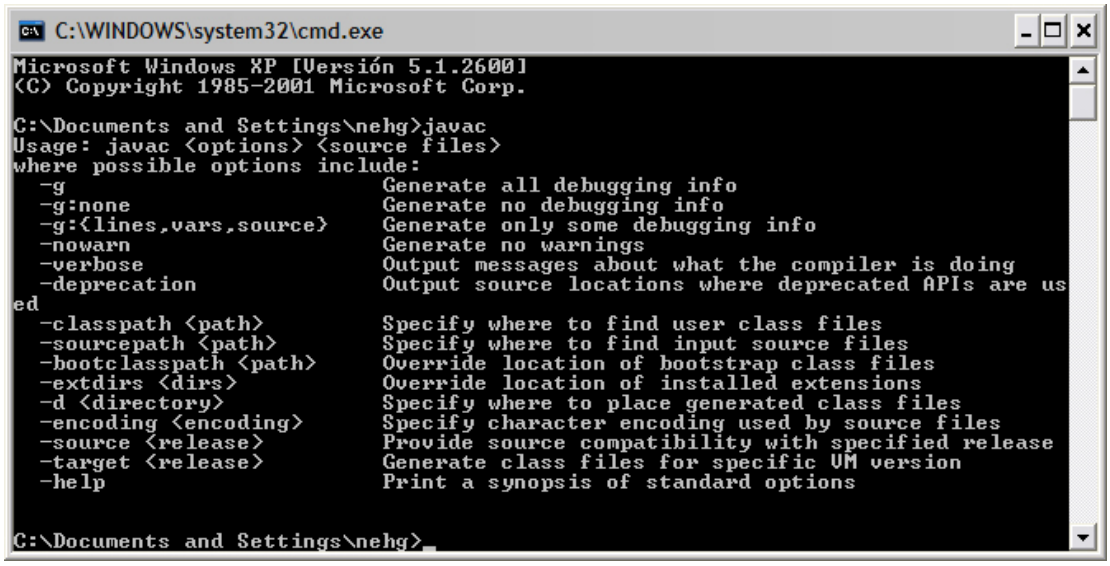


Figura A. 4 Variables de entorno configuradas

Sin embargo, se deberá configurar manualmente si se obtiene un mensaje como el mostrado en la figura A.5.

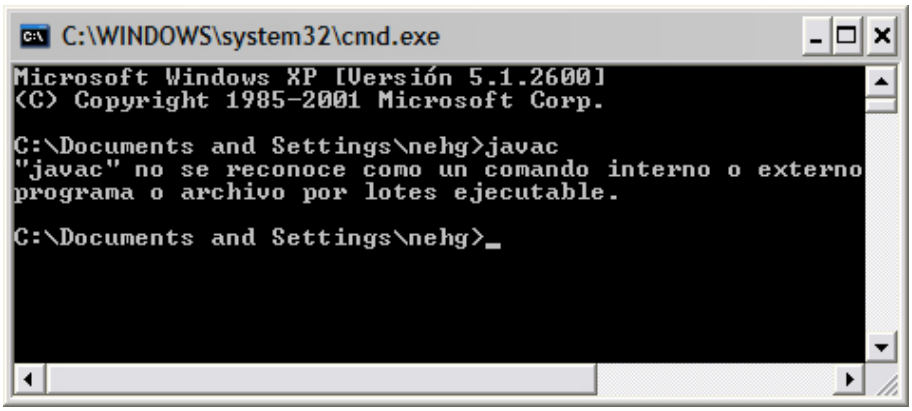


Figura A. 5 Variables de entorno no configuradas

Configuración Manual

1. Dar clic derecho sobre Mi PC y seleccionar Propiedades.
2. Seleccionar la pestaña Opciones avanzadas y dar clic en el botón Variables de entorno.

3. Tanto en el cuadro superior como el inferior, buscar una variable que se llama Path, si aparece en los dos cuadros, los puntos 4, 5 y 6 deben hacerse para cada una de ellas.
4. Dar clic sobre ella y luego sobre el botón Modificar.
5. En el cuadro de valor de variable, AÑADIR al final (NO sustituir ni BORRAR nada) la ruta del directorio bin de java. Por ejemplo si se instaló la "j2sdk1.4.2_12" y no se cambió el directorio de instalación por defecto, se tendría que añadir: ;c:\j2sdk1.4.2_12\bin.
6. Dar clic en Aceptar.

En la figura A.6 se puede ver un ejemplo de la configuración de la variable path.

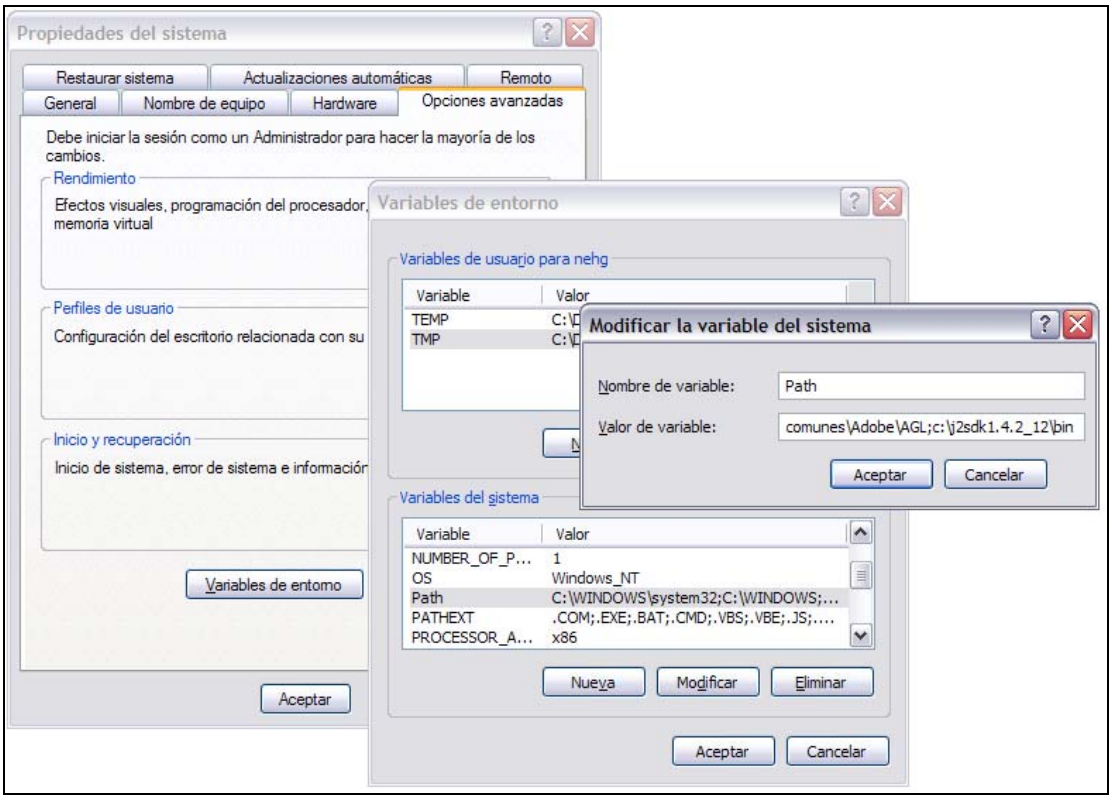


Figura A. 6 Configuración de variables de entorno

7. Buscar nuevamente en la lista de variables si existe CLASSPATH. Si no existe, pasar al punto 11, no obstante, si aparece en los dos cuadros, el punto 8, 9 y 10 debe hacerse para cada una de ellas.

8. Dar clic sobre ella y luego sobre el botón Modificar.
9. En el cuadro de valor de variable, AÑADIR al final (NO sustituir ni BORRAR nada) lo siguiente ";."(un punto y coma y un punto, sin las comillas).
10. Dar clic en Aceptar.
11. Dar clic en el botón Aceptar para que se guarden los cambios y que se cierre la ventana de variables de entorno.
12. Cerrar la ventana de Propiedades del sistema.

Ahora al repetir los pasos de verificación de la configuración, deberá salir el mensaje apropiado (figura A.4), en caso contrario, revisar que los puntos anteriores se hayan realizado correctamente.

J2ME Wireless Toolkit

La herramienta J2MEWTK (Java 2 Micro Edition Wireless Toolkit) soporta el desarrollo de aplicaciones Java que corren en dispositivos móviles. Para el funcionamiento de J2MEWTK, es necesario tener instalado el Java 2 SDK, Standard Edition (J2SE SDK), versión 1.4.2 o superior.

Descarga

1. Entrar en: `http://java.sun.com/products/sjwtoolkit/`
2. Elegir la versión de J2MEWTK, para el caso: J2ME Wireless Toolkit 2.2
3. Hay que realizar el proceso de registro para que permita la descarga
4. Aceptar la licencia
5. Descargar tanto el toolkit como el parche

Instalación

1. Iniciar la instalación dando doble clic en el icono de la aplicación.
2. Aparecerá una ventana de bienvenida del instalador, dar clic en ***Next***.

3. En la siguiente ventana se mostrará la licencia, para aceptarla dar clic en ***Yes***.
4. Por defecto el instalador detectará la carpeta de instalación del J2SDK y lo mostrará en una ventana, dar clic en ***Next***.
5. Aparecerá una ventana donde es posible cambiar el directorio de instalación del toolkit, se recomienda dejar el por defecto, dar clic en ***Next***.
6. En la siguiente ventana aparecerá la carpeta de programas del menú inicio, dar clic en ***Next***.
7. El instalador mostrará un resumen de la configuración elegida, dar clic en ***Next***.
8. Esperar a que termine la instalación, cuando esto suceda, aparecerá una ventana indicando que la instalación fue exitosa, dar clic en ***Finish***.

Actualización

Para actualizar el toolkit, se debe descomprimir el archivo (el parche que se descargó juntamente con el toolkit) en la carpeta de instalación del J2ME Wireless Toolkit, esto hará que se sobrescriban los archivos pertinentes.

1. Dar clic derecho sobre el archivo comprimido que contiene el parche.
2. Elegir la opción ***extraer todo…*** .
3. Se iniciará el asistente para la extracción de carpetas comprimidas (en zip), dar clic en ***Siguiente***.
4. En la siguiente ventana se debe seleccionar el destino donde se extraerán los archivos, acá se debe elegir la carpeta donde se instaló el tookit, si no se cambió el directorio por defecto, será: c:\WTK22 (ver figura A.7), dar clic en ***Siguiente***.
5. Aparecerá una ventana indicando que un archivo ya existe (ver figura A.8), y preguntará si se desea reemplazar, dar clic en ***Si a todo***.
6. Cuando el asistente haya terminado, aparecerá una ventana indicando que la extracción ha finalizado, se debe quitar el cheque de Mostrar archivos extraídos y dar clic en ***Finalizar***.

Figura A. 7 Selección de directorio para extracción de archivos

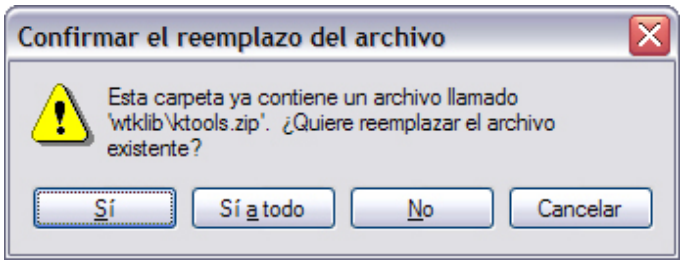


Figura A. 8 Confirmación de reemplazo de archivos

JEDIT

JEdit se distribuye bajo los términos de la Licencia Pública General de GNU. Como ésta es una aplicación escrita en Java, corre sobre Windows, Linux, Mac OS X, y otras plataformas.

Descarga

1. Entrar en: `http://www.jedit.org/`
2. Elegir ***Download***
3. En el apartado ***Stable version*** seleccionar ***Windows installer***

4. Elegir un ***mirror*** (servidor) dando clic en ***Download*** para iniciar la descarga

Instalación

1. Iniciar la instalación dando doble clic en el icono de la aplicación.
2. Aparecerá una ventana de bienvenida del asistente, dar clic en ***Next***.
3. En la siguiente ventana, seleccionar ***I accept the agreement*** y dar clic en ***Next***.
4. Aparecerá una ventana para seleccionar el directorio de instalación, se recomienda dejar el directorio por defecto y dar clic en ***Next***.
5. En la siguiente ventana es posible elegir los componentes a ser instalados, por defecto aparece instalación completa, dar clic en ***Next***.
6. Se mostrará una ventana don aparece el nombre de la carpeta de programas del menú inicio, permite cambiar el nombre o incluso omitirla dando clic en ***Don't create a start menu folder***, dar clic en ***Next***.
7. En la siguiente ventana dar clic en ***Next***.
8. Aparecerá una ventana con un resumen de la configuración de la instalación, dar clic en ***Install***.
9. Finalmente aparecerá una ventana indicando que la instalación ha terminado, quitar el cheque de ***Launch JEdit*** y dar clic en ***Finish***.

Midlets

Los Midlets son aplicaciones basadas en el perfil MIDP capaces de correr en dispositivos móviles. Se mostrará a continuación la creación de un Midlet.

los midlet tienen una sección de importación de paquetes, una clase principal, un constructor, no cuentan con un método main, sin embargo debe implementar métodos abstractos como el startApp(), en la figura A.9 se muestran las secciones principales de un midlet.

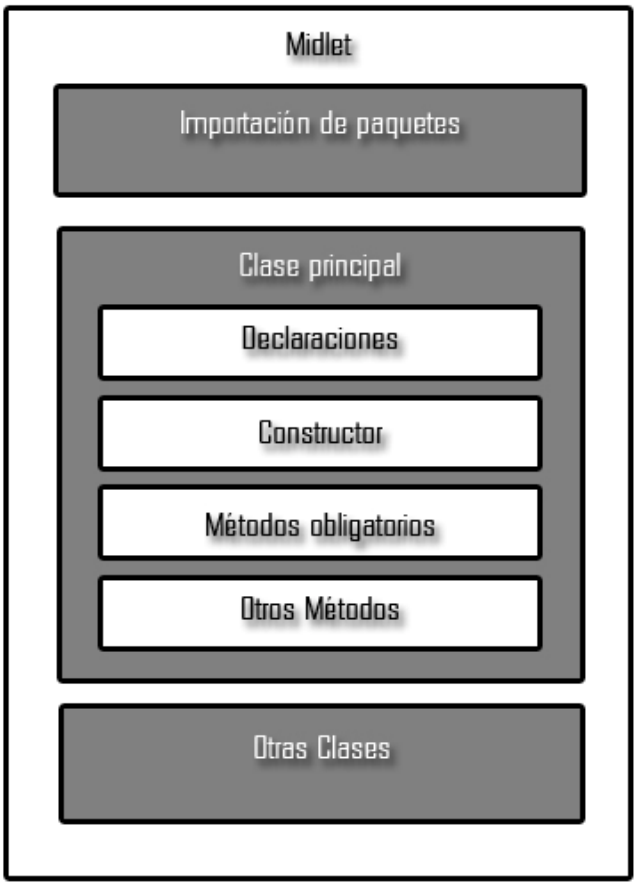


Figura A. 9 Cuerpo de un midlet

Consideraciones en el desarrollo de midlets

Para el desarrollo de midlets se deben tomar en cuenta lo siguiente:

1. Todo midlet debe importar los paquetes javax.microedition.midlet.* y javax.microedition.lcdui.*, de la siguiente manera:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cada midlet debe extender la clase MIDlet

```
public class HolaMundo extends MIDlet {
```

3. Un midlet no debe tener ningún método public static void main(), como ocurre en las aplicaciones Java de escritorio.

4. Un midlet ha de heredar de la clase javax.microedition.midlet, concretamente de javax.microedition.midlet.midlet, y obligatoriamente tiene que heredar los siguientes métodos: startApp(), pauseApp() y destroyApp().
5. Un midlet desde que se crea hasta que se destruye, puede encontrarse en diferentes estados.

Ciclo de vida de un midlet

El ciclo de vida de un midlet lo constituyen los diversos estados por los que atraviesa (ver figura A.10). Los estados posibles de un midlet son:

DETENIDO. Estado en el que se encuentra un midlet que ha sido creado pero todavía no ha ejecutado por primera vez el método ***startApp()***. Éste también puede ser provocado llamando a los métodos ***pauseApp()*** o ***notifyPaused()***. En este estado, el midlet mantiene los recursos mínimos posibles.

ACTIVO. Estado de ejecución del midlet en el entra posterior a la ejecución inicial del método ***startApp()*** o por la recuperación después de la llamada al método ***resumeRequest()*** tras una pausa.

DESTRUIDO. Provocado por la llamada a los métodos de ***destroyApp()*** o ***notifyDestroyed()***. Si el midlet entra en el estado destruido, toda su actividad habrá terminado, no pudiendo pasar a otro estado.

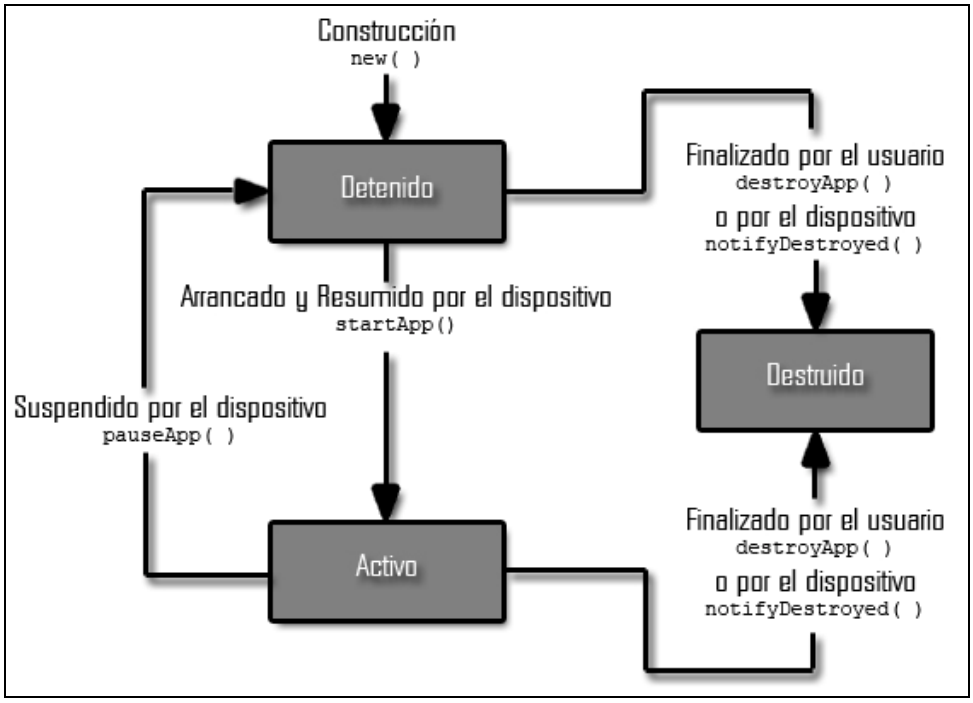


Figura A. 10 Ciclo de vida de un midlet

Creación del Midlet HolaMundo

El Midlet que se desarrollará, siguiendo la tradición, es uno que muestre el mensaje: Hola Mundo.

Pasos para la creación del Midlet HolaMundo

1. Ingresar al entorno ***KToolbar***, por defecto estará en: ***Menú inicio*** / ***Programas*** / ***J2ME Wireless Toolkit 2.2*** / ***KToolbar***.
2. Aparecerá una ventana como la de la figura A.11, dar clic en ***New Project***.

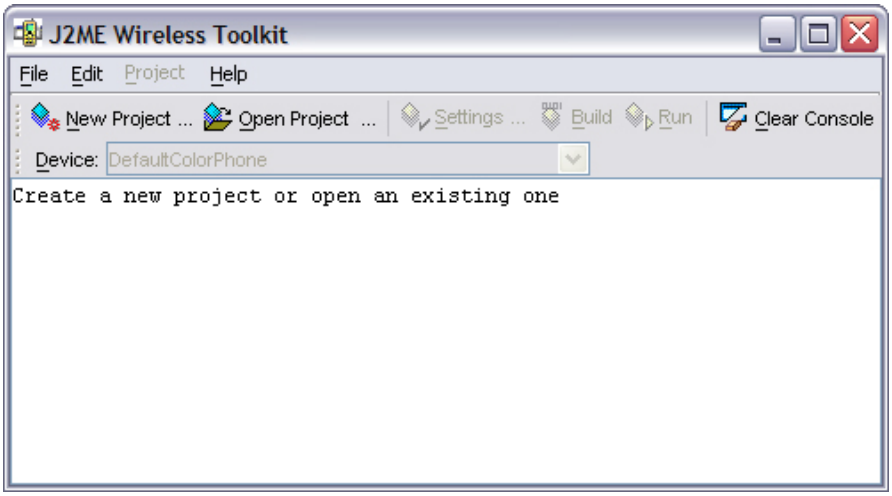


Figura A. 11 Entorno KToolbar

3. Aparecerá una ventana solicitando el nombre del proyecto y la clase, escribir HolaMundo en ambos (ver figura A.12) y dar clic en ***Create Project***.

4. Se mostrará una ventana de configuración (ver figura A.13), en la pestaña ***API Selection*** en ***Target Platform*** seleccionar ***Custom***.

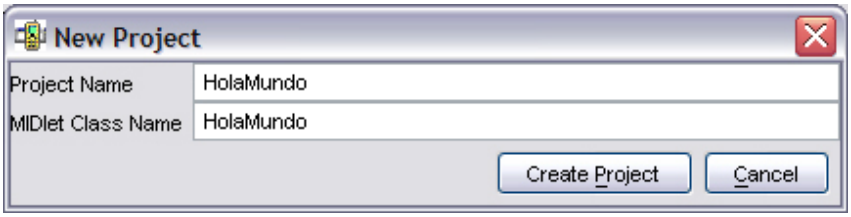


Figura A. 12 Nombres del proyecto y clase

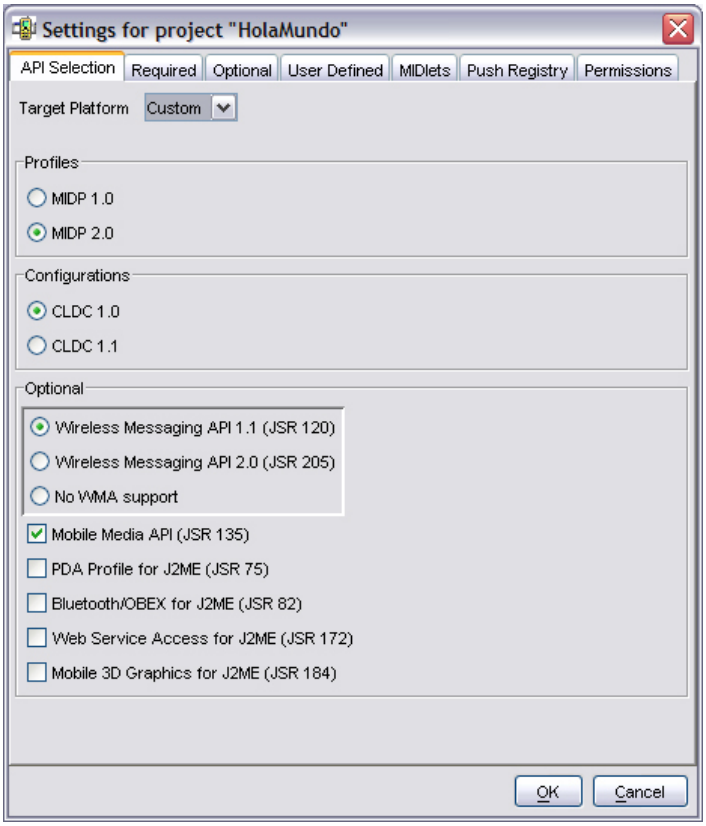


Figura A. 13 Configuración del proyecto

5. Ahora se debe elegir las opciones de acuerdo a las características soportadas por el dispositivo móvil, por ejemplo MIDP 2.0 y CLDC 1.0, luego dar clic en ***OK***.

6. Aparecerá entonces en la ventana de KToolbar (ver figura A.14) una serie de mensajes indicando donde ubicar los archivos del proyecto.

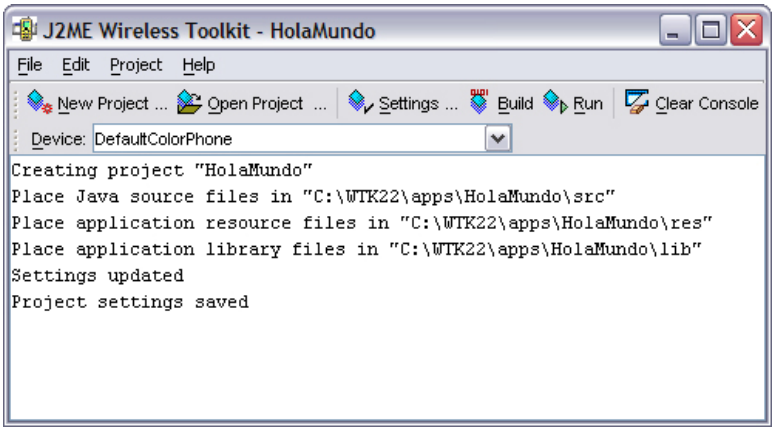


Figura A. 14 Mensajes de creación del proyecto

Ahora KToolbar ha creado la estructura de directorios donde se deben albergar los diferentes archivos de la aplicación (ver figura A.15), por defecto estará en: ***C:\WTK22\apps\HolaMundo***. Cada una de las carpetas tiene un propósito, ***src*** es para almacenar los archivos fuente y ***res*** es para los recursos como sonidos e imágenes.

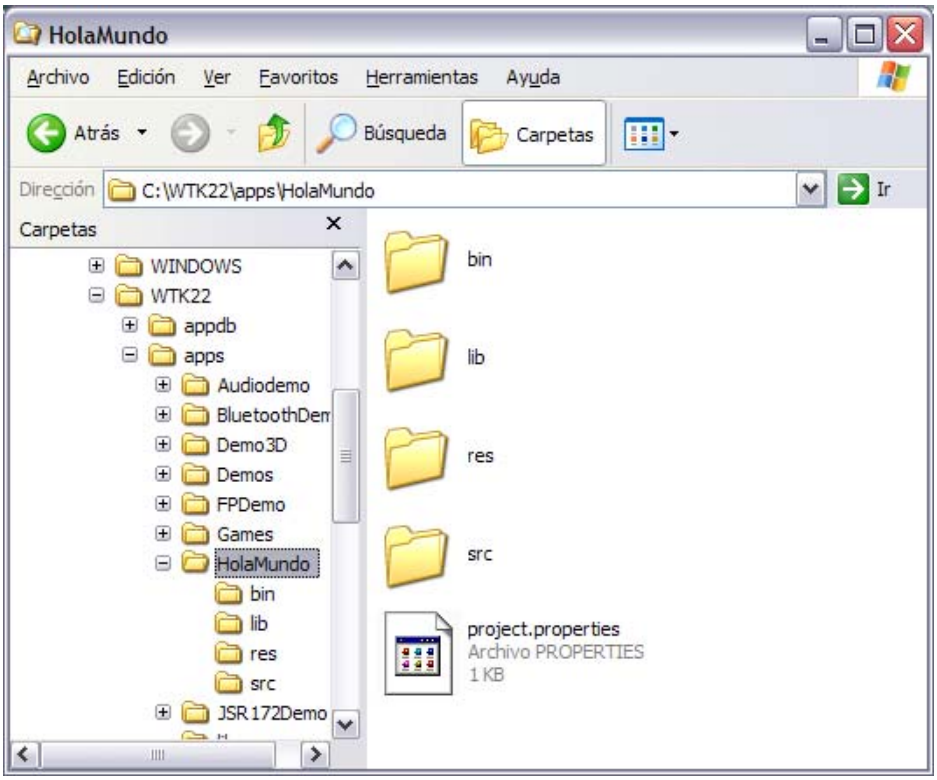


Figura A. 15 Estructura de directorios del proyecto

En vista de que KToolbar no cuenta con un editor integrado, se hace necesario el uso de otra herramienta, bien puede emplearse un editor de texto normal como ***notepad***, sin embargo existen otros editores que, por ser orientados a la programación, hacen más cómoda la edición, este es el caso de JEdit.

7. Abrir el JEdit, por defecto estará en ***Menú inicio*** / ***Programas*** / ***jEdit 4.2*** / ***jEdit***.
8. Lo primero que se debe hacer es guardar el archivo, esto es para que el editor reconozca que es una aplicación java y vaya coloreando el texto, lo cual mejora la

presentación y comprensión del código. El archivo deberá guardarse en la carpeta ***src*** del proyecto HolaMundo, es decir, en ***C:\WTK22\apps\HolaMundo\src***, con el nombre ***HolaMundo.java***, es importante recalcar que se debe escribir tal y como se ha señalado, ya que el nombre de la clase principal tiene que ser idéntico al nombre del archivo que lo contiene, diferenciando mayúsculas y minúsculas.

9. Escribir el código siguiente:

```
//Importamos los paquetes javax.microediton.MIDLET y javax.microedition.lcdui
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

//Clase principal
public class HolaMundo extends MIDlet implements CommandListener{
	//Declaraciones
	private Command comandosalir;
	private Display mostrar;
	private Form pantalla;

	//Constructor
	public HolaMundo(){
		//Obtenemos el objeto Display del midlet
		mostrar = Display.getDisplay(this);
		//Creamos el comando salir
		comandosalir = new Command("Salir",Command.EXIT,2);
		//Creamos la pantalla principal (un formulario)
		pantalla = new Form("HolaMundo");
		//Creamos y añadimos la cadena de texto a la pantalla
		StringItem saludo = new StringItem("","Hola Mundo!!!");
		pantalla.append(saludo);
		//Añadimos el comando salir e indicamos que clase lo manejará
		pantalla.addCommand(comandosalir);
		pantalla.setCommandListener(this);
	}

	public void startApp() throws MIDletStateChangeException{
		//Seleccionamos la pantalla a mostrar
		mostrar.setCurrent(pantalla);
	}
```

```
        public void pauseApp(){
        }

        public void destroyApp(boolean incondicional){
        }

        public void commandAction(Command c, Displayable s){
                //Salir
                if (c == comandosalir){
                        destroyApp(true);
                        notifyDestroyed();
                }
        }
}
```

10. Guardar los cambios, con esto ya se puede cerrar el jEdit.
11. Volver al entorno KToolbar y dar clic en el icono ***Build*** de la barra de herramientas o en el menú ***Project / Build***, si todo marcha bien, aparecerá el mensaje ***Build complete***. El compilado con KToolbar incluye la preverificación, entonces el programa ya está compilado y preverificado.
12. Una vez compilado y preverificado el proyecto, se debe dar clic en el icono ***Run*** de la barra de herramientas o en el menú ***Project / Run***. Es posible probar la aplicación en diferentes dispositivos, seleccionándolos de la lista desplegable ***Device*** y dando clic nuevamente en ***Run***. Con el dispositivo ***DefaultColorPhone*** se obtendría un resultado similar al mostrado en la figura A.16 a.
13. Para correr la aplicación, dar clic en el botón situado bajo la palabra ***Launch***, está acción mostrará la frase ***Hola Mundo !!!*** (ver figura A.16 b).

(a) (b)

Figura A. 16 (a) Menú para ejecutar la aplicación, (b) Aplicación en ejecución

Una vez verificado el funcionamiento de la aplicación a través del emulador, ya se puede empaquetar el programa, dejándolo listo para ser descargado y ejecutado en un dispositivo real.

Para ejecutar el proyecto en un dispositivo móvil se necesita del archivo .jar y .jad; el primero es un archivo comprimido que contiene las clases (.class) que ha generado la compilación de la aplicación, también puede incluir otros recursos como sonidos e imágenes y además contiene un archivo con extensión .mf conocido como archivo de manifiesto, éste contiene información sobre las clases contenidas en el archivo .jar; el segundo es necesario para la distribución, contiene información necesaria para la

instalación de la aplicación. A continuación se describen los pasos para empaquetar el midlet HolaMundo.

1. Volver a KToolbar
2. Ingresar al menú Project / Package / Create Package
3. Aparecerá el mensaje ***Building "HolaMundo"*** y posteriormente indicará que los archivos .jar y .jar han sido creados en el directorio ***bin*** de la aplicación, es decir, en ***C:\WTK22\apps\HolaMundo\bin***.

Es posible editar los atributos contenidos en el archivo .jad, dando clic en el icono ***Settings*** de la barra de herramientas de KToolbar, esto abrirá una ventana de configuración, luego se debe dar clic en la pestaña ***Required*** o en la ***Optional***, dependiendo de los atributos que se deseen cambiar. El significado de estos atributos se describe en la tabla A.2.

ATRIBUTO	CARACTER	DESCRIPCION
MIDlet-Name	Obligatorio	Nombre del conjunto, o suite, de midlets que lo identifica. Es el nombre que se presenta al usuario.
MIDlet-Version	Obligatorio	Versión de la suite de midlets, que puede utilizarse para instalación, actualización o información al usuario. El valor por defecto es 0.0.0
MIDlet-Vendor	Obligatorio	Distribuidor o creador del midlet. Texto libre que se presenta al usuario.
MIDlet-Icon	Opcional	Nombre del archivo de imagen en formato .png que identifica gráficamente al conjunto de midlets.
MIDlet-Description	Opcional	Descripción del conjunto de midlets. Texto libre que se puede incorpora de forma opcional en el archivo de descripción.
MIDlet-Info-URL	Opcional	Dirección URL en donde se encuentra la descripción adicional de la distribución de la suite de midlets.
MIDlet-<n>	Obligatorio	El nombre, icono y clase de cada n midlet en el archivo .jar separados por coma. El valor de n más pequeño debe ser 1 y los siguientes han de ser números consecutivos. Nombre, identifica al midlet; icono, es el nombre del archivo dentro de la distribución .jar con la imagen .png que identifica al midlet n; clase, es el nombre de la clase MIDlet que implementa el midlet n.
MIDlet-Jar-URL	Opcional	Dirección URL, desde la que se puede descargar el archivo de distribución .jar.
MIDlet-Jar-size	Opcional	Tamaño del archivo de distribución de la suite .jar en bytes.

MIDlet-Jar-Data-Size	Opcional	Número mínimo de bytes de la zona de almacenamiento de datos que necesita el midlet. Por defecto es 0.
MicroEdition-Profile	Obligatorio	Perfil J2ME requerido, p.e., MIDP-1.0. Cuando aparece más de una versión, deben estar separadas por espacios.
MicroEdition-Configuration	Obligatorio	Configuración J2ME requerida, p.e., CLDC-1.0.

Tabla A. 2 Descripción de atributos del archivo .jar

Las líneas siguientes muestran el contenido del archivo .jad correspondiente al midlet HolaMundo.

```
MIDlet-1: HolaMundo, HolaMundo.png, HolaMundo
MIDlet-Jar-Size: 1288
MIDlet-Jar-URL: http://localhost/j2me/HolaMundo.jar
MIDlet-Name: HolaMundo
MIDlet-Vendor: Nelson Hernandez
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

Finalmente se deben transferir los archivos .jar y .jad al dispositivo móvil, y para esto hay varias formas, puede ser por puerto infrarrojo, bluetooth, cable de datos, o a través de Internet.

Posterior a la transferencia, sólo queda entrar en el menú del dispositivo móvil, acceder a la carpeta donde fueron copiados los archivos y ejecutar el .jar, *et voilà*.

Explicación del Midlet HolaMundo

Se explicará a continuación el código del midlet HolaMundo.

Importación de paquetes

El código del midlet comienza con las líneas siguientes:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

Con esto se está importando dos paquetes:

1. El paquete ***javax.microedition.midlet***. Es necesario para que el proyecto se pueda ejecutar en un dispositivo móvil.

2. El paquete ***javax.microedition.lcdui***. Provee de los elementos que se utilizarán en la interfaz de usuario.

Clase principal

Para iniciar la clase principal, se hace con la siguiente línea de código:

```
public class HolaMundo extends MIDlet implements CommandListener{
```

Con la línea anterior se realizan varias acciones:

1. Se inicia la clase principal que debe llamarse igual que el archivo, en este caso ***HolaMundo*** (diferenciando mayúsculas de minúsculas).
2. Se hereda (o extiende, ***extends***) la clase MIDLET, que es fundamental para crear clases ejecutables.
3. Se implementa ***(implements***) la interfaz ***CommandListener*** que define una estructura de clase en la cual se deben declarar unos objetos (llamados comandos), los cuales responderán a alguna acción del usuario. En la aplicación HolaMundo se implementa el método ***commandAction()***.

Declaraciones

Una vez iniciada la clase principal, se hacen las declaraciones, con el siguiente código.

```
private Command comandosalir;
private Display mostrar;
private Form pantalla;
```

Con esto se han declarado los siguientes objetos:

1. Un comando (objeto) de la clase ***Command***, que representará la acción salir: ***comandosalir***.
2. Un objeto de la clase ***Display***, que se encarga de controlar lo que se muestra en pantalla: ***mostrar***.
3. Un objeto de la clase ***Form*** que definirá una pantalla donde se agregarán objetos de entrada y salida de datos por teclado y pantalla, respectivamente: ***pantalla***.

Constructor

Con la siguiente línea se inicia el constructor.

```
public HolaMundo(){
```

El constructor debe llevar el mismo nombre que la clase, no hay que especificar tipo de retorno (ni si quiera poner void) y es de acceso público.

En el constructor de clase se establecen los valores que han de tomar las variables, se crean las instancias de las clases.

Cuando se ejecuta un midlet, se crea un objeto display. Este objeto será el encargado de mostrar información en la pantalla. Para poder utilizarlo, se tiene que obtener una referencia a dicho objeto. Esto es lo que hace precisamente la siguiente línea mediante el método ***getDisplay()*** del objeto estático ***Display***.

```
mostrar = Display.getDisplay(this);
```

Un comando es un elemento que permite interactuar con el usuario. Para crear un comando se debe crear una instancia (con ***new***) de la clase ***Command()***. Como parámetro se le pasa el texto del comando, el tipo de comando y la prioridad. En un comando, la prioridad indica la importancia de ese comando con respecto a otros en la pantalla, dicha prioridad puede ir de 1 a 10, siendo 1 el valor de prioridad más alto.

```
comandosalir = new Command("Salir",Command.EXIT,2);
```

En la siguiente línea se instancia (con ***new***) un objeto ***Form*** con título ***HolaMundo*** y de nombre ***pantalla***. Ésta será la pantalla principal del proyecto.

```
pantalla = new Form("HolaMundo");
```

En la siguiente línea se declara y crea una cadena de texto inicializándola con la frase

Hola Mundo!!!.

```
StringItem saludo = new StringItem("","Hola Mundo!!!");
```

Luego se agrega (mediante el método ***append***) a la pantalla principal (***pantalla***) dicha cadena de texto.

```
pantalla.append(saludo);
```

Ahora se añade el comando (mediante el método ***addCommand()***) a la lista de comandos de la pantalla principal (***pantalla***). Luego se establece la clase que quedará a la "escucha" de esos comandos, para ello se utiliza la clase ***setCommandListener()***. En este caso, el método encargado de procesar los comandos está dentro de la propia clase HolaMundo, por lo que se emplea el operador ***this***.

El método que se encarga de procesar los comandos es ***commandAction()***, éste se implementa más posteriormente.

```
pantalla.addCommand(comandosalir);
pantalla.setCommandListener(this);
```

Métodos obligatorios

Hay tres métodos estáticos que es obligatorio declararlos (aunque no contengan código), éstos son: pauseApp, destroyApp y startApp. En la aplicación HolaMundo sólo se codifica el último de los citados anteriormente, así pues, el resto quedan vacíos.

El método startApp es el que se ejecuta justo después del constructor, podría compararse con el procedimiento main de los programas escritos en la mayoría de los lenguajes.

En este caso, el método ***startApp***, sólo lleva una instrucción que se encarga de seleccionar (con el método ***setCurrent()***), la pantalla que se mostrará, en este ejemplo ***pantalla*** es la pantalla principal.

```
public void startApp() throws MIDletStateChangeException{
        mostrar.setCurrent(pantalla);
}
```

El siguiente método se encarga de controlar las acciones que se deben realizar cuando la aplicación queda pausada por el usuario, por la misma aplicación o por otra.

```
public void pauseApp(){
        }
```

El método destroyApp, se encarga de controlar las acciones que deben llevarse a cabo antes de finalizar la aplicación.

```
public void destroyApp(boolean incondicional){
}
```

Otros métodos

Las siguientes líneas de código sirven para implementar el método que se encarga de procesar los comandos:

```
public void commandAction(Command c, Displayable s){
        //Salir
        if (c == comandosalir){
                destroyApp(false);
                notifyDestroyed();
```

```
            }
        }
```

Cuando el usuario genera un comando, se llama al método ***commandAction()***. Este método recibirá dos parámetros:

1. El comando que se generó (en este caso será ***c***).
2. Y un objeto de la clase Displayable, que contiene la pantalla del comando (en este caso será ***s***).

Se comprueba con ***c*** si éste es el comando que se declaró antes y que representa la acción de salir. Sí, es así, se cierra la aplicación con el método ***destroyApp(true)*** y se notifica el fin de la aplicación con ***notifyDestroyed()***.

Apéndice B

PointBase Micro

Actualmente existen en el mercado diversos gestores de bases de datos orientados a los dispositivos móviles, y en todos ellos prevalece la característica del small footprint, es decir, consumo mínimo de memoria. El acceso a las bases de datos, desde aplicaciones en dispositivos móviles, generalmente se hace a través de ODBC o JDBC. Además, la mayoría de estos gestores proporciona utilerías para sincronización de datos entre el dispositivo móvil y un servidor de base de datos corporativo.

PointBase Micro y Transformation Server es la solución presentada por DataMirror Mobile Solutions Inc. para el almacenamiento y sincronización de datos entre dispositivos móviles y servidores de bases de datos corporativas.

Descarga

DataMirror proporciona versiones de evaluación de sus productos, por lo tanto es posible descargar PointBase Micro, de la siguiente manera:

1. Entrar en: `http://www.pointbase.com`
2. Dar clic en ***Download***
3. Completar el proceso de registro
4. Elegir el producto a evaluar
5. Aceptar la licencia
6. Descargar el instalador y documentación apropiada al sistema operativo

Instalación

Se explicará brevemente la instalación de PointBase Micro en el sistema operativo de escritorio Windows.

1. Descomprimir el archivo descargado.

2. Ejecutar el archivo descomprimido.
3. Aparecerá una ventana de bienvenida al asistente de instalación, dar clic en ***Next*** para iniciar la instalación.
4. La siguiente ventana mostrará un mensaje de bienvenida del asistente, dar clic en ***Siguiente***.
5. El asistente mostrará una ventana con el acuerdo de licencia del software, seleccionar ***I accept the terms of the License Agreement*** y dar clic en ***Next***.
6. Aparecerá una ventana donde debe seleccionarse el tipo de instalación entre ***Complete*** y ***Custom***, elegir ***Complete*** y dar clic en ***Next***.
7. En la siguiente ventana es posible cambiar el directorio de instalación de PointBase, se recomienda dejar el por defecto y dar clic en ***Next***.
8. Se visualizará una ventana donde se puede seleccionar la versión de la Máquina Virtual de Java, elegir la apropiada (1.4 o superior) y dar clic en ***Install***.
9. Al finalizar el proceso de instalación aparecerá una ventana indicándolo, dar clic en ***Done***.

Acceso a la Base de Datos con J2ME

Después de instalar PointBase, se genera una estructura de directorios como la mostrada en la figura B.1.

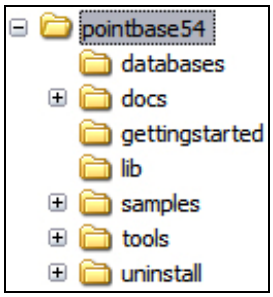


Figura B. 1 Directorios de PointBase

Dentro de la carpeta ***lib*** se encuentran los archivos necesarios para acceder a la base de datos. Para el perfil MIDP se debe utilizar la librería ***pbmicroMIDP54ev.jar*** (Donde: 54 es la versión, es decir 5.4, y ev significa que es una versión de evaluación).

Se mostrará a continuación un ejemplo de acceso a base de datos desde de una aplicación J2ME empleando la librería ***pbmicroMIDP54ev.jar***. Se presupone que ***Java*** está instalado en el directorio ***C:\j2sdk1.4.2_12*** y que el ***J2ME Wireless Toolkit*** está instalado en ***C:\WTK22***.

1. Empleando ***KToolbar***, crear un proyecto con el nombre ***eisi***, lo cual generará una estructura de directorios como la mostrada en la figura B.2

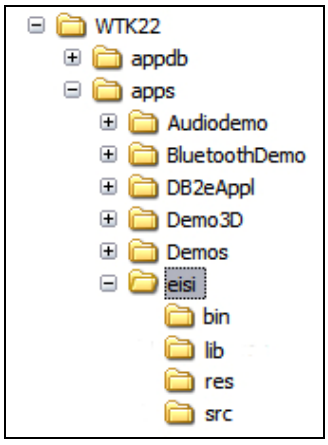


Figura B. 2 Proyecto eisi

2. Crear dos carpetas dentro de la carpeta ***eisi*** una con el nombre ***classes*** y la otra con el nombre ***preverified***.

3. Haciendo uso de una utilería de compresión, como ***winrar***, extraer el contenido del archivo ***pbmicroMIDP54ev.jar***, el cual tiene dos carpetas: ***com*** y ***META-INF***. Copiar la carpeta ***com*** dentro de la carpeta ***classes*** creada en el paso anterior. Con lo cual quedará una estructura como la mostrada en la figura B.3.

4. En un editor de texto plano (como el ***notepad***), escribir el código mostrado a continuación y guardarlo con el nombre ***eisi.java*** dentro de la carpeta ***src*** del proyecto ***eisi***.

```
//Importamos los paquetes javax.microediton.MIDLET y javax.microedition.lcdui
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
//Importamos el paquete para el manejo de la base de datos
import com.pointbase.me.*;

//Clase principal
public class eisi extends MIDlet implements CommandListener{
	//Declaraciones
	private Command comandosalir;
	private Display mostrar;
	private Form pantalla;

	static Connection con = null; // Para la conexión
	//Constructor
	public eisi(){
		//Obtenemos el objeto Display del midlet
		mostrar = Display.getDisplay(this);
		//Creamos el comando salir
		comandosalir = new Command("Salir",Command.EXIT,2);
		//Creamos la pantalla principal (un formulario)
		pantalla = new Form("EISI");
		//Creamos y añadimos la cadena de texto a la pantalla
		StringItem saludo = new StringItem("","Base de datos");
		pantalla.append(saludo);
		//Añadimos el comando salir e indicamos que clase lo manejará
		pantalla.addCommand(comandosalir);
		pantalla.setCommandListener(this);
		//Nos conectamos a la base de datos eisi
		try{
			con = DriverManager.getConnection(
				"jdbc:pointbase:micro:eisi", "admin", "admin");
			System.out.println("Conexión establecida");
		} catch (Exception e){
			System.out.println(e.getMessage());
		}
	}
	public void startApp() throws MIDletStateChangeException{
		//Seleccionamos la pantalla a mostrar
		mostrar.setCurrent(pantalla);
	}
	public void pauseApp(){
```

```
        }
        public void destroyApp(boolean incondicional){
                // Cerramos la conexión a la base de datos
                try {
                        if(con!= null){
                                con.close();
                                con = null;
                        }
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }
        }
        public void commandAction(Command c, Displayable s){
                //Salir
                if (c == comandosalir){
                        destroyApp(true);
                        notifyDestroyed();
                }
        }
}
```

Tal como puede apreciarse en el código anterior, para establecer la conexión a la base de datos, es necesario importar el paquete ***com.pointbase.me.**** y luego con la siguiente línea se conecta a la base de datos ***eisi***, con el usuario ***admin*** cuya clave es ***admin***.

```
con = DriverManager.getConnection(
        "jdbc:pointbase:micro:eisi", "admin", "admin")
```

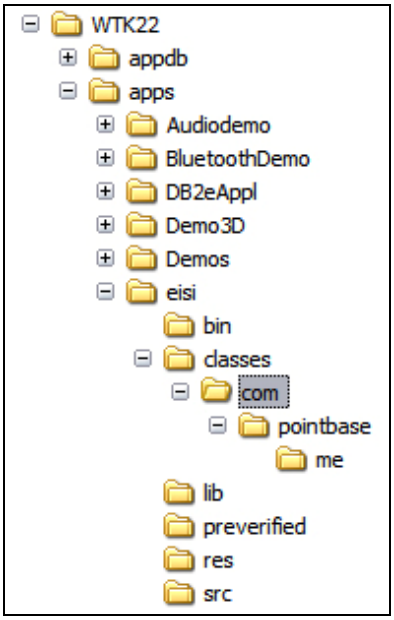


Figura B. 3 Clases del proyecto eisi

La ejecución del código mostrado en los siguientes pasos presupone la configuración de la variable ***path*** apuntando hacia los directorios de instalación de ***Java*** y del ***J2ME Wireless Toolkit***. Puede configurarse la variable ***path*** empleando la siguiente instrucción desde la línea de comandos del DOS.

```
set path=%path%;C:\j2sdk1.4.2_12\bin;C:\WTK22\bin
```

5. Compilar la aplicación ejecutando la siguiente instrucción desde la línea de comandos del DOS.

```
javac -bootclasspath
c:\wtk22\lib\cldcapi10.jar;c:\wtk22\lib\midpapi20.jar;c:\wtk22\apps\eisi\classes -d
c:\wtk22\apps\eisi\classes c:\wtk22\apps\eisi\src\*.java
```

6. Realizar el proceso de preverificación mediante la siguiente instrucción desde la línea de comandos del DOS.

```
preverify -classpath c:\wtk22\lib\cldcapi10.jar;c:\wtk22\lib\midpapi20.jar -d
c:\wtk22\apps\eisi\preverified c:\wtk22\apps\eisi\classes
```

7. Empaquetar la aplicación con la siguiente instrucción (no olvidar el punto al final de la instrucción) desde la línea de comandos del DOS.

```
jar cfm c:\wtk22\apps\eisi\bin\eisi.jar c:\wtk22\apps\eisi\bin\MANIFEST.MF -C
c:\wtk22\apps\eisi\preverified .
```

8. Actualizar en el archivo ***eisi.jad*** (ubicado en ***C:\WTK22\apps\eisi\bin***) la especificación del tamaño del archivo ***eisi.jar***. De la siguiente manera:

    a. Verificar el tamaño del archivo ***eisi.jar*** (ubicado en ***C:\WTK22\apps\eisi\bin***), dando clic derecho sobre él y luego clic en ***propiedades*** (ver figura B.4), luego tomar nota del ***tamaño*** (ver figura B.5).

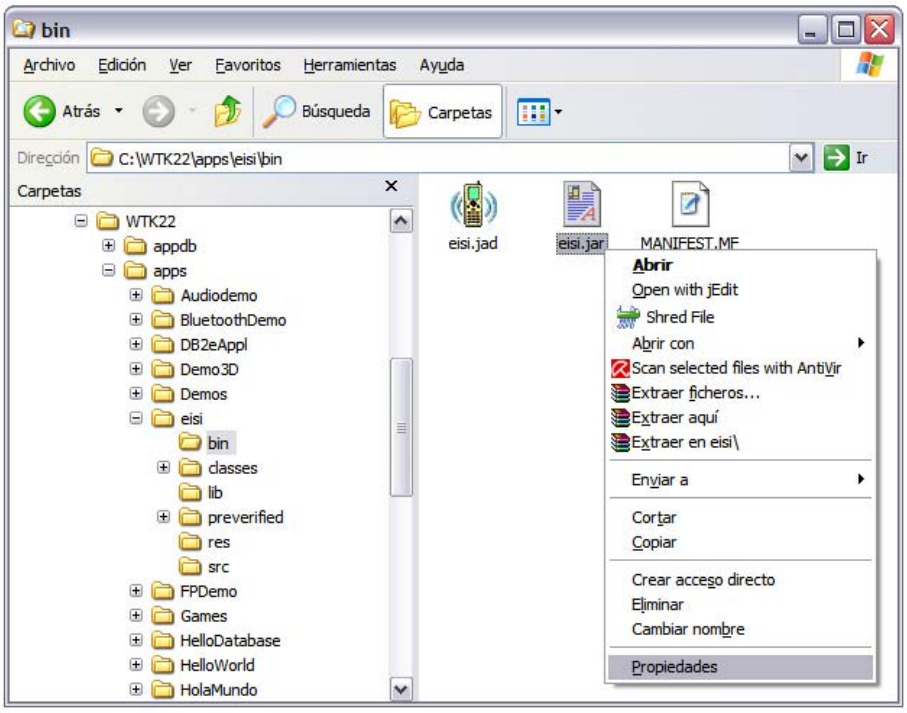


Figura B. 4 Propiedades del archivo esi.jar

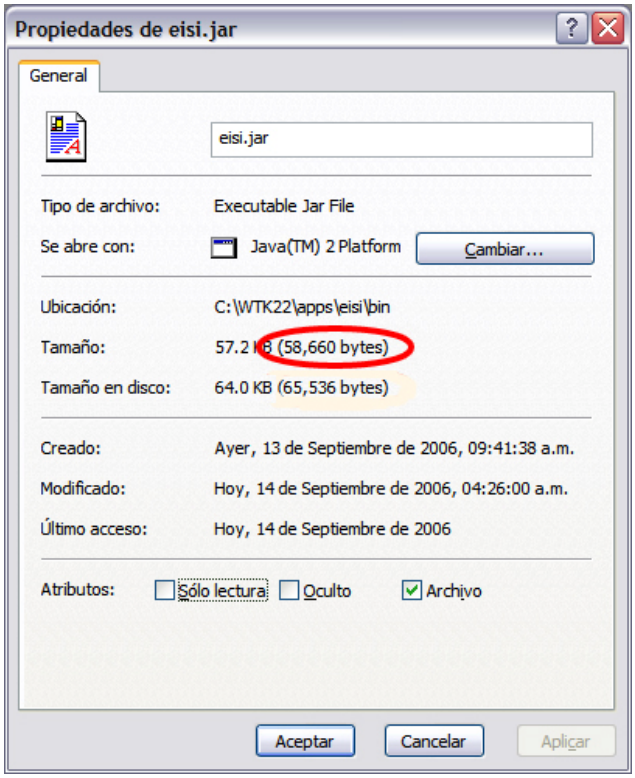


Figura B. 5 Tamaño del archivo eisi.jar

b. Abrir el archivo ***eisi.jad*** (ubicado en ***C:\WTK22\apps\eisi\bin***) con un editor de texto plano (como el ***notepad***) y buscar la línea ***MIDlet-Jar-Size:***, cambiar el tamaño especificado, por el actual (el obtenido en el paso anterior), quedando el contenido del archivo tal como se muestra a continuación:

```
MIDlet-1: eisi, eisi.png, eisi
MIDlet-Jar-Size: 58660
MIDlet-Jar-URL: eisi.jar
MIDlet-Name: eisi
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

Guardar los cambios y cerrar el archivo.

9. Ejecutar la aplicación con la siguiente instrucción desde la línea de comandos del DOS.

```
c:\wtk22\bin\emulator -cp c:\wtk22\apps\eisi\bin\eisi.jar -Xdescriptor:
c:\wtk22\apps\eisi\bin\eisi.jad
```

Glosario

ADO	*ActiveX Data Object*, es uno de los mecanismos que usan los programas de computadoras para comunicarse con las bases de datos, darles órdenes y obtener resultados de ellas.
API	*Application programming interface* o Interface de programación de aplicaciones. Es el conjunto de rutinas del sistema que se pueden usar en un programa para la gestión de entrada/salida, gestión de ficheros, etc. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.
Atributo	Es una característica, propiedad o elemento de un objeto. El conjunto de los atributos definen el estado del objeto.
Atributo Atómico	Es un atributo no divisible, por ejemplo: El atributo DUI, no tiene sentido dividirlo, no tendrá significado para la entidad, ya que éste es un número indivisible.
C	Lenguaje de programación.
C++	Lenguaje de programación orientado a objetos, basado en el lenguaje C.
Caso de uso escencial	Caso de uso libre de aspectos tecnológicos y detalles de implementación (diseño, interfaz de usuario, etc.).
Caso de uso primario	Representa un proceso común e importante. un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización software.

Certificado	Un certificado es una especie de "pasaporte" electrónico el cual establece las credenciales de un sitio por medio del cual se realicen pagos, compras u otras transacciones a través de redes como Internet.
Clase	Estructura que define como son los objetos, indicando sus atributos y sus acciones.
CLDC	*Connected Limited Device Configuration*. Es una especificación para configuraciones J2ME.
Constructor	Es un método especial de una clase que sirve para construir objetos de dicha clase. Su nombre es el mismo que el de la clase.
Descriptor (.jad)	Fichero cuyo objetivo es el de proporcionar la información requerida para comprobar que el MIDLET suite es adecuado para el dispositivo en el que queremos instalarlo. Su extensión es .JAD y al contrario que el manifiesto, éste no se incluye en el paquete.
Driver	Pequeño programa cuya función es controlar el funcionamiento de un dispositivo del ordenador bajo un determinado sistema operativo. También hacer referencia a programas que permiten el acceso a una base de datos desde un programa.
Embedded	En español empotrado o incluido, hace referencia a un sistema computacional que es parte integral de un sistema más grande. Subsistema que ejecuta o controla una función.
Emulador	Programa que permite reproducir el comportamiento de un computador u arquitectura hardware en otro.
Encriptación	Encriptación es el proceso mediante el cual la información es codificada de tal manera que no pueda ser interpretada fácilmente. Es una medida de seguridad utilizada para que al momento de transmitir la información ésta no pueda ser interceptada por intrusos. Opcionalmente puede existir además un proceso de desencriptación a través del cuál la información puede ser interpretada una vez que llega a su lugar de origen.

Entorno de Ejecución de Java	El entorno de ejecución incluye una Máquina Virtual de Java, librerías de clases y otros ficheros que soportan la ejecución de programas escritos en el lenguaje Java.
Footprint	El *footprint* es la porción de memoria que requiere una aplicación para funcionar.
Framework	*Framework* es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
GCC	GCC (*GNU Compiler Collection*, antes GNU C Compiler) - una colección de compiladores, diseñados dentro del Proyecto GNU.
GNU	El proyecto GNU fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completo libre. Actualmente brinda la posibilidad de resolver, casi, la totalidad de los problemas de tratamiento informático con software libre. Esto incluye desde un juego hasta el núcleo del sistema operativo, pasando por aplicaciones ofimáticas. El núcleo más conocido del proyecto GNU es Linux.
HTML	El HTML, acrónimo inglés de *Hyper Text Markup Language* (lenguaje de marcación de hipertexto), es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.
HTTP	*Hyper Text Transfer Protocol*, es decir, protocolo de transferencia de hipertexto. Es un protocolo de solicitud/respuesta entre clientes y servidores.
IDE	Un entorno (o ambiente) integrado de desarrollo o en inglés *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas para un programador.
Intellisense	Es una característica que introdujo Microsoft en sus entornos de desarrollo, la cual permite autocompletado de nombres de variables, funciones y métodos.

J2ME	*Java 2 Micro Edition*. Es la respuesta de Sun Microsystems para una plataforma de dispositivos móviles.
Java	Java es una plataforma de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales
JDBC	JDBC es el acrónimo de *Java Database Connectivity*, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando instrucciones SQL del modelo de base de datos que se utilice.
JVM	La máquina virtual de Java (en inglés *Java Virtual Machine*, JVM) es un programa ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.
Manifiesto (.mf)	Fichero cuya finalidad es describir el contenido del fichero .JAR con información tal como el nombre, versión, vendedor, etc. también se incluye en este fichero una entrada por cada MIDLET que lo compone.
Método	Es la implementación de un algoritmo que representa una operación o función que un objeto realiza. El conjunto de los métodos de un objeto determinan el comportamiento del objeto.
Midlet	Un MIDlet es una aplicación MIDP. Tiene que satisfacer los requisitos siguientes para funcionar en un dispositivo móvil: 1) La clase principal necesita ser una subclase de javax.microedition.midlet.MIDlet, 2) El MIDlet necesita ser empaquetado dentro de un archivo .jar, 3) El archivo del jar necesita ser pre-verificado.

Midlet Suite	Todos los ficheros necesarios para un MIDLET es lo que se denomina MIDLET Suite y en concreto se compone de: a) Ficheros java encapsulados en un fichero JAR, b) Fichero manifiesto describiendo el contenido del fichero jar (manifest.mf), c) Recursos como imágenes también incluidas en el fichero jar, d) Fichero descriptor de la aplicación java (Java Application Descriptor File -JAD).
MIDP	*Mobile Information Device Profile*. Conjunto de APIs Java que se implementan sobre los dispositivos CLDC.
Objeto	Un objeto es una instancia de una clase, que encapsula estado y procesos. Pueden representar cosas reales o conceptuales.
ODBC	*Open Database Connectivity*, interfaz normalizada, o intermedia, para acceder a una base de datos desde un programa.
PHP	PHP (acrónimo recursivo de "*PHP: Hypertext Preprocessor*", originado inicialmente del nombre PHP Tools, o *Personal Home Page Tools*) es un lenguaje de programación interpretado. Se utiliza entre otras cosas para la programación de páginas web activas, y se destaca por su capacidad de mezclarse con el código HTML.
RSA	Algoritmo criptográfico de clave publica desarrollado por Rivest, Shamir y Adelman.
SDK	*Software Development Kit* o Kit de desarrollo de software, es un conjunto de aplicaciones para desarrollar programas en un determinado lenguaje o para un determinado entorno.
SQL	El Lenguaje de Consulta Estructurado (*Structured Query Language*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. En 1992 se lanzó el estándar ampliado y revisado del SQL llamado SQL-92 o SQL2.
UML	Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modelling Language*) es un lenguaje de modelado de sistemas de software.

Variable Path	Variable del entorno, cuyo valor contiene los directorios donde el sistema buscara cuando intente encontrar un comando o aplicación.
WAP	*Wireless Application Protocol* (protocolo de aplicaciones inalámbricas) es un estándar abierto internacional para aplicaciones que utilizan las comunicaciones inalámbricas, p.e. acceso a Internet desde un teléfono móvil.
Web-to-Go	Es un *framework* que permite a los desarrolladores crear aplicaciones basadas en web, que se pueden usar online (conectado a la red de la compañía o a Internet) u offline.
Windows CE O/S	Sistema operativo embebido de tiempo real. Es un sistema de 32-bit escalable, viene con una plataforma de construcción, que constituye una herramienta para modificar y depurar el sistema operativo. Windows CE es empleado en una amplia gama de dispositivos de comunicaciones, entretenimiento e informática móvi.
XML	Es el acrónimo del inglés *eXtensible Markup Language* (lenguaje de marcado ampliable o extensible) desarrollado por el World Wide Web Consortium (W3C).

Referencias Bibliográficas

Libros de texto

- [FRO04] Froufe Quintas, Agustín; Jorge Cárdenes, Patricia, **Java 2 Micro Edition (J2ME). Manual de usuario y tutorial**, Ra-Ma Editorial, España, 2004.
- [BOO99] Booch, Grady; Rumbaugh, James; Jacobson, Ivar, **El lenguaje Unificado de Modelado**, Addison Wesley Iberoamericana, España, 1999.
- [CEB05] Ceballos Sierra, Francisco Javier, **Java 2. Curso de programación**, 3ª Edición, Ra-Ma Editorial, España, 2005.

Proyectos fin de carrera

- [ABU05] Abujas Pereira, Jerónimo, **Programación de Juegos para telefonía móvil y PDA**, Universidad de Cádiz, España, 2005.
- [NAV05] Navarrete González, Javier, **Tutorial de programación de dispositivos móviles**, Universidad de Cádiz, España, 2005.

Sitios web

- [SYM06] Symbian Ltd, **Symbian OS,** (`http://www.symbian.com/symbianos/`) (1 agosto 2006).

  (Información acerca del sistema operativo Symbian)
- [WIK06-1] Wikipedia, **Windows CE**, (`http://es.wikipedia.org/wiki/Windows_CE`) (1 agosto 2006).

  (Información acerca del sistema operativo Windows CE)
- [WIK06-2] Wikipedia, **Windows Mobile**, (http://es.wikipedia.org/wiki/Windows_Mobile) (1 agosto 2006).

  (Información acerca del sistema operativo Windows Mobile)
- [MIK06] Mikehall's Embedded WEblog, **Windows Mobile vs. Windows CE**, (`http://blogs.msdn.com/mikehall/rss.xml`) (1 agosto 2006).

(Información acerca del sistema operativo Windows CE y Windows Mobile)

- [MSC06-1]Microsoft Corporation, **Preguntas y respuestas de Windows CE**, (http://support.microsoft.com/default.aspx?scid=kb%3Bes%3B158182) (1 agosto 2006).

  (Información acerca del sistema operativo Windows CE)

- [PMX06] PocketMéxico, **Bill Gates presenta el Windows Mobile 5.0**, (http://www.pocketmexico.com/displayarticle703.html) (1 agosto 2006).

  (Información acerca del sistema operativo Windows Mobile)

- [WIK06-3] Wikipedia, **Palm OS**, (http://es.wikipedia.org/wiki/Palm_OS) (2 agosto 2006).

  (Información acerca del sistema operativo Palm)

- [PAL06-1] PalmSource Inc, **Palm OS Cobalt**, (http://www.palmsource.com/es/palmos/cobalt.html) (2 agosto 2006).

  (Información acerca del sistema operativo Palm Cobalt)

- [PAL06-2] PalmSource Inc, **Palm OS Garnet**, (http://www.palmsource.com/es/palmos/garnet.html) (2 agosto 2006).

  (Información acerca del sistema operativo Palm Garnet)

- [PAL06-3] PalmSource Inc, **Palm OS Prestaciones y ventajas**, (http://www.palmsource.com/es/palmos/features.html) (2 agosto 2006).

  (Información acerca de las ventajas del sistema operativo Palm)

- [MSC06-2] Microsoft Corporation, **Desarrollar soluciones móviles,** (http://www.microsoft.com/spanish/msdn/comunidad/comunidades/aplicaciones_moviles/art02/default.asp) (31/julio/2006).

  (Información acerca del desarrollo de aplicaciones para dispositivos móviles)

- [PCW06] PC World digital, **Desarrollos para Palm**, (http://www.idg.es/pcworld/articulo.asp?idart=119469) (31 julio 2006).

  (Información acerca del desarrollo de aplicaciones para dispositivos móviles que corran el sistema operativo Palm)

- [LNX06]Linux Para Todos, **Dispositivos moviles**, (http://www.linuxparatodos.net/geeklog/article.php?story=20050414044103357) (31/julio/2006).

  (Información acerca del desarrollo de aplicaciones para dispositivos móviles)

- [RIS06] Revista: Informática y Salud, **La programación de Dispositivos de Cómputo Móviles**, (http://www.seis.es/seis/is/is40/programacion.htm) (31 julio 2006).

  (Información acerca de sistemas operativos que corren en dispositivos móviles y de lenguajes y entornos de programación de aplicaciones para dispositivos móviles)

- [MSC06-3] Microsoft Corporation, **Windows Mobile Emulator**, (http://msdn.microsoft.com/mobility/downloads/Emulator/default.aspx) (1 agosto 2006).

  (Información acerca del emulador para el sistema operativo Windows Mobile)

- [SNS06] SoftNews NET SRL, **hsqldb Database Engine 1.8.0 RC9**, (http://linux.softpedia.com/get/Database/Database-Engines/hsqldb-Database-Engine-660.shtml) (7 agosto 2006).

  (Información acerca del gestor de bases de datos HSQLDB)

- [OST06] Open Source Technology Group, **hsqldb Database Engine - Default branch**, (http://freshmeat.net/projects/hsqldb/freshmeat.net_%20Project%20details%20for%20hsqldb%20Database%20Engine.htm) (7 agosto 2006).

  (Información acerca del gestor de bases de datos HSQLDB)

- [IBM06-1] IBM, **DB2 Everyplace Express Edition**, (http://www-306.ibm.com/software/info/ecatalog/es_ES/products/M609940P78104G50.html) (7 agosto 2006).

  (Información acerca del gestor de bases de datos DB2 Everyplace)

- [CWD06] ComputerWorld, **IBM adapta DB2 a los dispositivos de mano Lanza una nueva versión de DB2 Everyplace adecuada para .Net**, (http://www.idg.es/computerworld/impart.asp?id=153260) (7 agosto 2006).

  (Información acerca del gestor de bases de datos DB2 Everyplace)

- [IBM06-2] IBM, **DB2 Everyplace**, (http://www-306.ibm.com/software/data/db2/everyplace/) (7 agosto 2006).

  (Información acerca del gestor de bases de datos DB2 Everyplace)

- [PED06-1] Pearson Education, **Mobile Database Review: Oracle 9i Lite**, (http://www.samspublishing.com/articles/article.asp?p=25484&rl=1) (7 agosto 2006).

  (Información acerca del gestor de bases de datos Oracle 9i Lite)

- [ORC06-1] Oracle, **Oracle Database Lite 10g Build, Deploy, and Manage Mobile Applications**, (http://www.oracle.com/database/Lite_Edition.html) (7 agosto 2006).

  (Información acerca del gestor de bases de datos Oracle Database Lite)

- [ORC06-2] Oracle, **Oracle9i Lite A Technical White Paper**, (http://otn.oracle.com/tech/wireless/papers/q4-02/Oracle_Lite_wp11-02.pdf) (7 agosto 2006).

  (Información acerca del gestor de bases de datos Oracle 9i Lite)

- [PED06-2] Pearson Education, **PointBase 4.0 Micro Edition**, (http://www.informit.com/articles/article.asp?p=22285&seqNum=6&rl=1) (8 agosto 2006).

  (Información acerca del gestor de bases de datos PointBase Micro)

- [MIC06] The mobile internet community - Mobic, **PointBase Is First to Market with a JAVA Technology-based SQL Relational Database Running on the JAVA 2, Micro Edition (J2ME) and JAVA 2, Standard Edition (J2SE)**, (http://www.mobic.com/oldnews/2001/06/pointbase_is_first_to_market_wit.htm) (8 agosto 2006).

  (Información acerca del gestor de bases de datos PointBase)

- [WIK06-4] Wikipedia, **Adaptive Server Anywhere**, (http://es.wikipedia.org/wiki/Adaptive_Server_Anywhere) (7 agosto 2006).

  (Información acerca del gestor de bases de datos Adaptive Server Anywhere)

- [SYB06] Sybase, **SQL Anywhere Studio**, (http://www.sybase.com/detail?id=1025129) (7 agosto 2006).

  (Información acerca del gestor de bases de datos Sybase SQL Anywhere)

- [SWH06] Super Warehouse, **Sybase SQL Anywhere Studio 9.X Base**, (http://www.superwarehouse.com/Sybase_SQL_Anywhere_Studio_9.X_10_Users/17890-09/pf/328191) (7 agosto 2006).

  (Información acerca del gestor de bases de datos Sybase SQL Anywhere)

- [IBM06-3] IBM, **Guía de instalación y del usuario de DB2 Everyplace**, (http://www.ibm.com/software/data/db2/everyplace/library.html)(6 septiembre 2006).

  (Información acerca del gestor de bases de datos DB2 Everyplace)

- [HAW06] HawHaw, **Make your website mobile!**, (http://www.hawhaw.de/) (14 noviembre 2006).

  (Información acerca de la tecnología HawHaw para el desarrollo de aplicaciones para dispositivos móviles)

- [WIK06-5] Wikipedia, **SuperWaba**, (http://es.wikipedia.org/wiki/SuperWaba) (14 noviembre 2006).

  (Información acerca de la tecnología SuperWaba para el desarrollo de aplicaciones para dispositivos móviles)

Índice Alfabético

Base de Datos, 95

cita, 11, 12, 17, 21, 42, 43, 46, 72, 79, 80, 85, 89, 122, 123, 124, 145, 170

Clases, 94, 140, 170, 253

consulta, i, 9, 11, 12, 16, 19, 27, 38, 39, 40, 41, 42, 43, 44, 50, 51, 67, 68, 76, 77, 78, 79, 87, 112, 113, 114, 115, 116, 117, 118, 119, 120, 122, 131, 146, 170, 172, 177, 182, 203

Dispositivos Móviles, 1, 3, 1, 3

Emuladores, 3

evolución, i, 10, 11, 12, 17, 22, 44, 45, 68, 80, 81, 85, 90, 116, 117, 118, 119, 120, 121, 139, 177, 182, 183, 204

expediente, 9, 11, 12, 16, 18, 37, 38, 39, 40, 41, 42, 43, 44, 60, 75, 85, 87, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 122, 123, 124, 125, 126, 127, 144, 145, 151, 154, 155, 159, 160, 164, 166, 167, 169, 172, 176, 179, 182, 187, 190, 191, 197, 198, 199, 200, 201, 202

fichas médicas, iii, 9, 15, 102

IDEs, 3, 138

J2ME, i, 5, 7, 139, 229, 230, 235, 243, 249, 250, 253, 258, 260, 263, 267

J2MEWTK, 229

Java, i, 3, 5, 6, 7, 138, 229, 231, 249, 250, 253, 259, 260, 261, 263

Java 2 Micro Edition, i, 5, 138, 229, 260, 263

Manual de Instalación, 209

Manual de Usuario, 194

Midlet, 232, 235, 260, 261

Modelo Conceptual, 26, 59

móviles, i, ii, iii, 1, 2, 3, 4, 5, 6, 7, 8, 9, 138, 139, 210, 229, 232, 248, 260, 263, 265

MovilMed, 175, 195, 210

PDA, i, 1, 2, 6, 263

PointBase, 4, 8, 248, 249, 267

Prototipo, 9

RMS, 4, 7, 143

SmartPhones, i, 1