



Desarrollo de Aplicaciones Móviles I

ÍNDICE

Página

Presentación	7
Red de contenidos	9
Unidad de aprendizaje 1: Java Micro Edition	
1.1 Tema 1 : Introducción a las aplicaciones móviles	11
1.1.1. : Introducción a JME	13
1.1.2. : Arquitectura JME	15
1.1.3. : Midlets: definición, estados y estructura básica	20
Unidad de aprendizaje 2: Interfaces	
2.1 Tema 2 : Desarrollo de Interfaces	35
2.1.1. : Interfaces de alto y bajo nivel	35
2.1.2. : Clases contenedoras: TextBox, Alert, List, Form	39
2.1.3. : Clases adicionales a las contenedoras: Image, Ticker	49
2.1.4. : Clases Item: TextField, StringItem, ChoiceGroup, ImageItem, DateField	51
2.2 Tema 3 : Manejo de eventos	55
2.2.1. : Configuración de escuchadores: ItemStateListener, ItemCommandListener	55
2.3 Tema 4 : Graficador	57
2.3.1. : Introducción a la interface gráfica	57
2.3.2. : Clase Canvas	57
2.3.3. : La API de juegos	58
Unidad de aprendizaje 3: Gestión de Almacenamiento de Registros JME	
3.1 Tema 5 : Gestión de Registros	81
3.1.1. : Introducción al sistema de almacenamiento de registros (RMS)	81
3.1.2. : Clase RecordStore: métodos openRecordStore, closeRecordStore	82
3.1.3. : Manipulación de registros: métodos addRecord,	83

setRecord, deleteRecord y getRecord

Unidad de aprendizaje 4: Conectividad JME

4.1 Tema 6 : Aplicaciones conectadas	93
4.1.1. : Conectividad: Clase Connector	93
4.1.2. : Uso del protocolo HTTP: Clases Connection / URLConnection	93
4.1.3. : Intercambio de datos: Clases InputStream / OutputStream	95

Unidad de aprendizaje 5: Android – Fundamentos y principales componentes

5.1 Tema 7 : Fundamentos de Android	103
5.1.1. : Arquitectura de una aplicación móvil Android. Configuración, principales librerías y herramientas de desarrollo.	103
5.1.2. : SDK (Standard Development Kit). Virtual Machine. Aplicación “Hola Mundo” para la plataforma Android.	105
5.2 Tema 8 : Fundamentos de aplicaciones Android	121
5.2.1. : El ciclo de vida de una aplicación Android.	121
5.2.2. : Prioridades y estados de una aplicación Android.	121
5.2.3. : Principales componentes: Introducción al uso de Actividades y la gestión de Recursos.	123
5.3 Tema 9 : Interfaces de Usuario	139
5.3.1. : Fundamentos de diseño y creación de componentes básicos Android.	139
5.3.2. : Creación de vistas personalizadas, controles y gestión de eventos.	142
5.4 Tema 10 : Componente Intent	150
5.4.1. : Componente Intent: definición y características. Uso de Intents para gestionar Actividades.	150

Unidad de aprendizaje 6: Persistencia de datos en Android

6.1 Tema 11 : Persistencia de Datos: Android SQLite.	163
--	-----

6.1.1. : Introducción a SQLite: cursores y contenedores de valores. Gestión de bases de datos con SQLite. Uso del componente SQLiteOpenHelper (RMS)	163
---	-----

Unidad de aprendizaje 7: Gestión Audio, Video y Uso de dispositivos Visuales

7.1 Tema 12 : Audio, Video y Uso de Dispositivos visuales	167
7.1.1. : Componente Media Player. Definición, características y principales aplicaciones.	167
7.1.2. : Visualización de videos usando la vista Video.	169
7.1.3. : Grabación de audio y video.	170

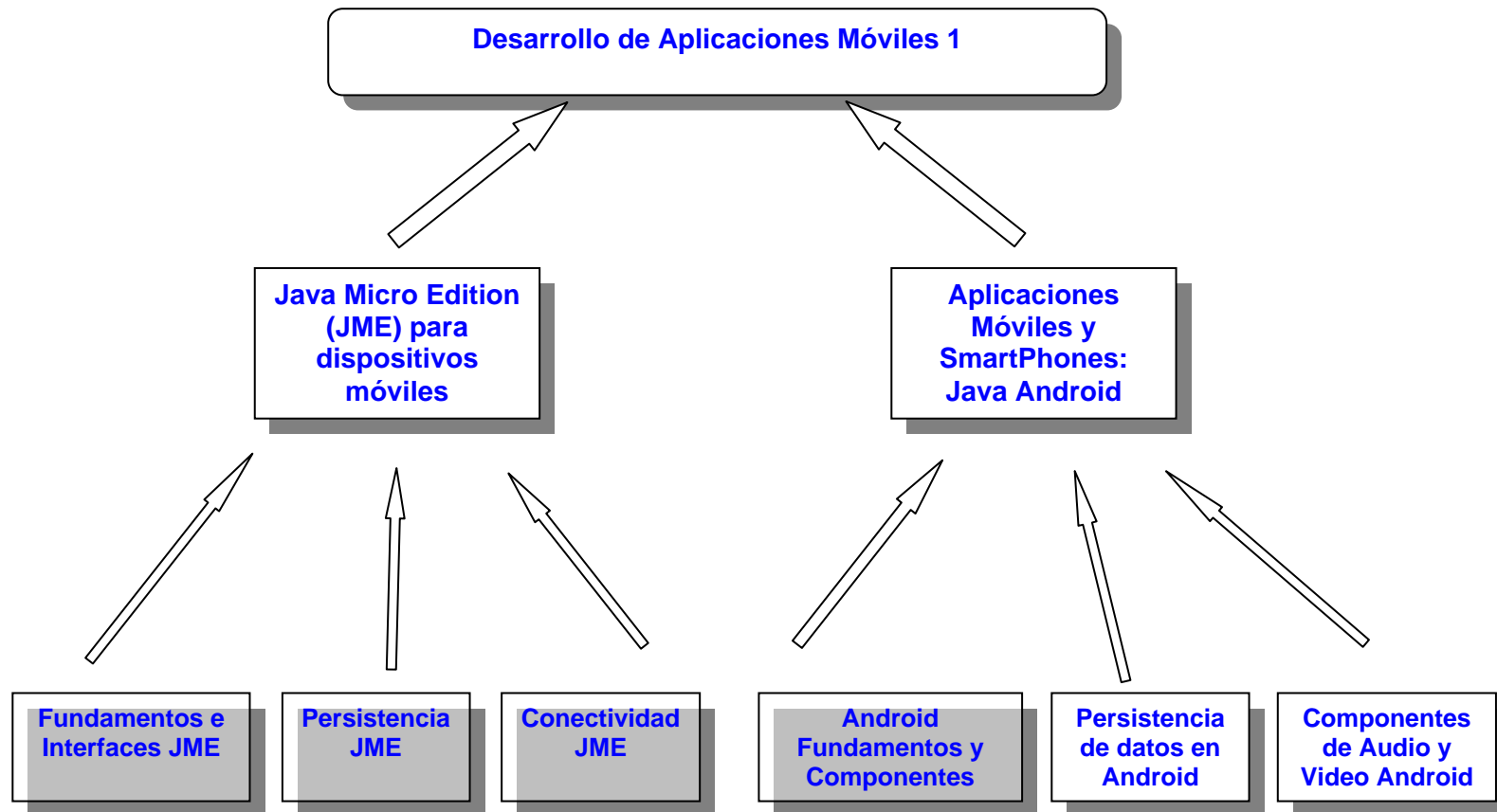
PRESENTACIÓN

El Taller de **Desarrollo de Aplicaciones Móviles I** es un curso que pertenece a la línea de programación y desarrollo de aplicaciones siendo un curso de especialidad sólo en la carrera de Computación e Informática. Permite al alumno iniciarse en la programación de dispositivos móviles bajo el entorno propio de un dispositivo móvil.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros, que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos, que debe desarrollar, es decir, los subtemas. Por último, encontrará las actividades que deberá desarrollar en cada sesión, que le permitirán reforzar lo aprendido en la clase.

El curso es práctico y consiste en un taller de programación. En la primera parte del curso, se revisan las características de Java Micro Edition (JME) y las interfaz de alto nivel y sus clases. Luego, se desarrollan aplicaciones que permitan el manejo del dispositivo móvil para ingresar, procesar y presentar información haciendo uso de las diversas clases de tipo ítem. Se concluye con el almacenamiento de información en memoria persistente para la posterior comunicación con servidores de red. En la segunda parte, se revisan las principales características de la plataforma Android y su integración con el lenguaje Java. Se utilizan sus principales componentes, tales como Activities, Intents, View y Layouts. Se crean aplicaciones con los principales componentes visuales de la plataforma, utilizando como repositorio de persistencia de datos SQLite.

RED DE CONTENIDOS



UNIDAD DE
APRENDIZAJE**1**

JME: JAVA MICRO EDITION

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno señala los equipos móviles que se pueden usar para un tipo de aplicación determinada y, asimismo, elabora la estrategia general para su desarrollo e instalación, usando la información sobre la arquitectura JME y la documentación técnica de cada equipo en particular.

TEMARIO

1.1 Tema 1 : Introducción a las aplicaciones móviles

1.1.1. : Introducción a JME

1.1.2. : Arquitectura de JME

1.1.3. : MIDlets: Definición, estados y estructura básica. Gestor de aplicaciones

ACTIVIDADES PROPUESTAS

- Los alumnos crean la aplicación Hola Mundo para la plataforma Java JME
- Los alumnos implementan una aplicación móvil utilizando los componentes básicos de un MIDLET.

1.1 Introducción a las aplicaciones móviles

Durante los últimos años, la telefonía celular ha alcanzado niveles altos en desarrollo de equipos y su evolución continúa siendo extremadamente acelerada. En efecto, la evolución de la tecnología móvil ha permitido llevar al mercado soluciones que brindan rentabilidad y ofrecen una mejor calidad de vida, ya que ponen al servicio del cliente la integración de las comunicaciones con la información. Hoy en día, se puede observar cómo el desarrollo de aplicaciones para dispositivos móviles va creciendo en el mercado comercial y se puede apreciar cómo aplicaciones que antes solo se podían manejar en un computador, funcionan en forma similar en un dispositivo móvil.

Las características concretas de este tipo de dispositivos han obligado a los desarrolladores de Java a construir un subconjunto del lenguaje y a reconfigurar sus principales bibliotecas para permitir su adaptación a un entorno limitado, propio de los dispositivos móviles. Todo esto hace que necesitemos una nueva plataforma de desarrollo y ejecución sobre la que centraremos nuestro estudio: Java Micro Edition, o de forma abreviada: JME.

1.1.1. Introducción a JME

1.1.1.1. Aplicaciones

La tecnología Java se creó como una herramienta de programación para ser usada en un proyecto de Sun Microsystems en el año 1991. En un principio fue diseñado para generar aplicaciones que controlaran electrodomésticos.

Con la llegada de Internet Java2 obtiene gran difusión. Esto se produce debido a su gran robustez e independencia de la plataforma donde se ejecutase el código, lo cual lo hace ideal para la creación de componentes interactivos integrados en páginas Web y programación de aplicaciones independientes. Estos componentes se denominaron applets. Con los años, Java ha progresado enormemente en varios ámbitos, e hizo necesaria la creación de diferentes ediciones, cada una de ellas orientadas a algún tipo de aplicación en particular.

La edición Java Micro Edition fue presentada por primera vez en 1999 por Sun Microsystems, orientada al desarrollo de aplicaciones para dispositivos pequeños. Su primera versión estaba orientada al desarrollo sobre Palms. Es actualmente la tecnología mas difundida para el desarrollo de aplicaciones para móviles.

1.1.1.2. Características

Algunas de las características de la tecnología Java, y que han sido en buena parte la razón de su gran difusión, se indican a continuación:

- **Orientado a objetos.** Al programar en Java se pueden usar las clases existentes, derivar otras clases a partir de las primeras y crear nuevas clases.
- **Portable.** Las aplicaciones java2 se ejecutarán sobre cualquier sistema que tenga instalado una máquina virtual java. No dependen de un hardware sistema operativo en particular.
- **Interpretado.** La ejecución de una aplicación Java2 es realizada por la máquina virtual, instrucción por instrucción.

- **Robusto.** La forma en que se diseñó la arquitectura de Java2 hace que los errores graves sean menos frecuentes.

1.1.1.3. Ediciones

Existen varias ediciones de Java, cada una de ellas diseñada para cierto ambiente en particular, tal como se indica en la figura 1.1.

- **Java Standard Edition (JSE)**

Es la edición que se emplea para desarrollar aplicaciones de escritorio, así como applets. Es también la versión que base para las demás ediciones.

- **Java Enterprise Edition (JEE)**

Es la edición que se emplea para desarrollar aplicaciones distribuidas, tales como aplicaciones Web o servlets. Incluye a la edición estándar.

- **Java Micro Edition (JME)**

Es la edición que se emplea para desarrollar aplicaciones para dispositivos móviles, denominados MIDlets. Es un subconjunto de la edición estándar con ciertas clases adicionales orientadas a las necesidades de este tipo de aplicaciones.



Figura 1.1: Ediciones de Java

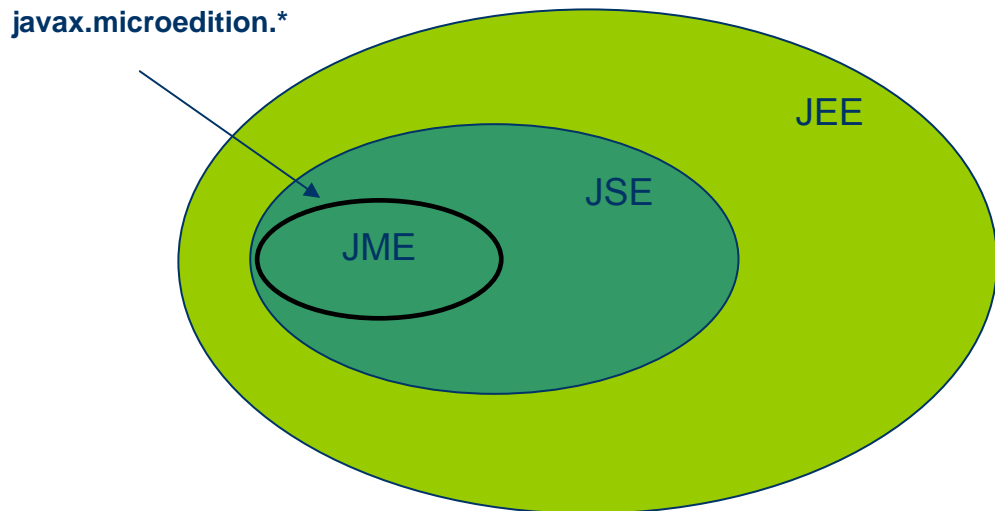


Figura 1.2: Las clases de JME son un subconjunto de JSE

- Java Card

Es la versión de Java enfocada a aplicaciones que se ejecutan en “tarjetas inteligentes” (smart cards), encontradas en equipos móviles, tarjetas de crédito, etc. Es una versión muy limitada de Java.

1.1.2. Arquitectura JME

La edición JME es una plataforma, una colección de tecnologías y especificaciones diseñadas para soportar la gran diversidad de dispositivos en el mercado. Por esta razón, su arquitectura es modular y escalable para conservar su funcionalidad y portabilidad. Los elementos que la conforman están graficados en la figura 1.3 y se especifican a continuación.

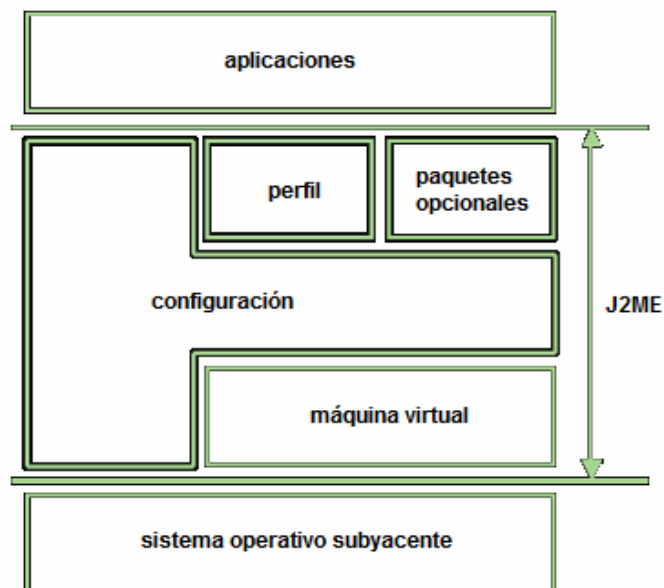


Figura 1.3. Arquitectura de JME

1.1.2.1. Sistema Operativo

Es desarrollado por el proveedor del dispositivo y sobre él se ejecutará la máquina virtual, también desarrollado por el proveedor. En buena parte de los dispositivos móviles, consiste de un sistema propietario limitado. Sin embargo, existen en el mercado algunos sistemas operativos para dispositivos móviles avanzados, tales como Symbian, Windows Mobile, Android, Palm OS, Mac IOS, etc.



Figura 1.4. Sistemas operativos para móviles

1.1.2.2. Máquina Virtual

Una máquina virtual de Java (JVM: Java Virtual Machine) es un programa encargado de lo siguiente:

- Interpretar el código intermedio (bytecode) de los programas Java a código máquina ejecutable por la plataforma;
- Efectuar las llamadas pertinentes al sistema operativo subyacente;
- 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 subyacente. Las implementaciones tradicionales de JVM son, en general, muy pesadas en cuanto a memoria ocupada y requerimientos computacionales. JME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos que, en algunos casos, suprimen algunas características con el fin de obtener una implementación menos exigente.

Kilo Virtual Machine (KVM)

Se corresponde con la máquina virtual más pequeña desarrollada por Sun. Su nombre KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de máquina virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria. La KVM está escrita en lenguaje C, aproximadamente unas 24000 líneas de código, y fue diseñada para ser:

- Pequeña, con una carga de memoria entre los 40Kb y los 80Kb, dependiendo de la plataforma y las opciones de compilación
- Alta portabilidad
- Modulable
- Lo más completa y rápida posible y sin sacrificar características para las que fue diseñada

Compact Virtual Machine (CVM)

La CVM soporta las mismas características que la máquina virtual de JSE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2Mb o más de memoria RAM.

1.1.2.3. Configuración

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. Éstas APIs describen las características básicas, comunes a todos los dispositivos:

- Características soportadas del lenguaje de programación Java
- Características soportadas por la Máquina Virtual Java
- Bibliotecas básicas de Java y APIs soportadas

Existen dos configuraciones en JME: CLDC, orientada a dispositivos con limitaciones computacionales y de memoria y CDC, orientada a dispositivos con no tantas limitaciones.

Configuración de dispositivos con conexión - CDC (Connected Device Configuration)

La CDC está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos y sistemas de navegación en automóviles. CDC usa una máquina virtual Java2 similar en sus características a una de JSE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo. Ésta máquina virtual es la que hemos visto como CVM (Compact Virtual Machine). La CDC está enfocada a dispositivos con las siguientes capacidades:

- Procesador de 32 bits;
- Disponer de 2 Mb o más de memoria total, incluyendo memoria RAM y ROM;
- Poseer la funcionalidad completa de la máquina virtual Java2;
- Conectividad a algún tipo de red.

Configuración de dispositivos limitados con conexión - CDC (Connected Limited Device Configuration)

La CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Un ejemplo de estos dispositivos son teléfonos móviles, buscapersonas, PDAs, organizadores personales, etc.

Algunas de estas limitaciones 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 160 Kb y 512 Kb de memoria total disponible. Como mínimo se debe disponer de 128 Kb de memoria no volátil para la máquina virtual Java2 y las bibliotecas CLDC, y 32 Kb de memoria volátil para la máquina virtual en tiempo de ejecución.
- Procesador de 16 a 32 bits con al menos 25 MHZ de velocidad;
- Ofrecer bajo consumo, debido a que estos dispositivos trabajan con suministro de energía limitado, normalmente baterías;
- Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps). La CLDC aporta las siguientes funcionalidades a los dispositivos:
- Un subconjunto del lenguaje Java2 y todas las restricciones de su máquina virtual (KVM);

- Un subconjunto de las bibliotecas Java del núcleo;
- Soporte para entrada y salida básica;
- Soporte para acceso a redes;
- Seguridad.

1.1.2.4. Perfiles

El perfil es el que define las APIs que controlan el ciclo de vida de la aplicación, interfaz de usuario, etc. Más concretamente, un perfil es un conjunto de APIs orientado a un ámbito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan, por ejemplo, electrodomésticos, teléfonos móviles, etc. y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil. Aquí podemos encontrar grandes diferencias entre interfaces, desde el menú textual de los teléfonos móviles hasta los táctiles de los PDAs.

Perfiles de 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. Este perfil incluye gran parte de los paquetes de la JSE, pero excluye, totalmente, los paquetes “java.awt” Abstract Windows Toolkit (AWT) y “java.swing” que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriera una GUI, entonces sería necesario un perfil adicional

- Personal Profile:

El Personal Profile es un subconjunto de la plataforma JSE v1.3, y 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 requiere una implementación del Foundation Profile, se construye encima de él. El perfil RMI soporta un subconjunto de las APIs JSE v1.3 RMI. Algunas características de estas APIs se han eliminado del perfil RMI debido a las limitaciones de cómputo y memoria de los dispositivos.

Perfiles 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 200x100 pixels.

- Mobile Information Device Profile (MIDP):

Este perfil está construido sobre la configuración CLDC. Al igual que CLDC, fue la primera configuración definida para JME. MIDP fue el primer perfil definido para esta plataforma y diseñado para dispositivos como teléfonos celulares y otros equipos de limitada capacidad.

1.1.3. MIDlets: Definición, estados y estructura básica. Gestor de aplicaciones

1.1.3.1. Definición

Un MIDlet es una aplicación Java2 para dispositivos móviles. Se caracteriza por usar las librerías del perfil MIDP sobre la configuración CLDC. Los MIDlets han sido diseñados pensando en que su principal medio de instalación será la descarga a través de Internet.

Al medio que garantiza la correcta descarga de la aplicación se denomina OTA (over the air).

1.1.3.2. Elementos

MIDlet jar

Es el archivo de extensión jar, que contiene las clases y otros recursos asociados, por ejemplo, las imágenes. Adicionalmente, incluye un archivo manifiesto con información acerca del contenido, este archivo tiene el nombre manifest.mf.

Descriptor

Es el archivo de texto plano con extensión jad. Contiene diversos datos que son leídos por el gestor de aplicaciones al momento de descargar el MIDlet. Revisar el capítulo siguiente para más detalles acerca del gestor de aplicaciones. Los datos guardados pueden ser obligatorios y opcionales, y adicionalmente se pueden agregar datos personalizados de la aplicación

- Datos obligatorios
 - ✓ MIDlet-Name: Nombre del MIDlet
 - ✓ MIDlet-Version: Versión del MIDlet
 - ✓ MIDlet-Vendor: Nombre del vendedor
 - ✓ MIDlet-Jar-URL: URL del sitio de descarga el MIDlet.
 - ✓ MIDlet-Jar-Size: Tamaño del archivo jar
 - ✓ MicroEdition-Profile: Nombre y versión del perfil
 - ✓ MicroEdition-Configuration: Nombre y versión de la configuración

Importante:

Se utiliza el término MIDlet suite para referirse a los dos archivos, MIDlet jar y descriptor, especialmente cuando contienen varias clases de tipo MIDlet.

1.1.3.3. Gestor de Aplicaciones

Para que todo aplicativo pueda ser instalado y funcione en un dispositivo móvil, éste debe contar con ciertas características tanto en hardware como en software.

En cuanto al hardware, el equipo debe contar con los siguientes elementos:

- ✓ Un procesador, que es el que se va encargar de administrar los diferentes accesorios del dispositivo celular;
- ✓ Una cantidad de memoria para el manejo de los procesos del sistema operativo y aplicaciones. Cabe resaltar que esta memoria se divide en una memoria tipo RAM, que es donde se ejecutan las aplicaciones, y una memoria tipo ROM, en la cual se almacena parte de las aplicaciones que contiene el dispositivo móvil. Además de poder tener una gran capacidad de batería.
- ✓ En cuanto los requerimientos de software, el dispositivo debe contar con una aplicación dentro del sistema operativo que sea el encargado de manejar las aplicaciones. A dicha aplicación se le denomina gestor de aplicaciones.

El gestor de aplicaciones o AMS (Application Management System) es el encargado de administrar y controlar todas las aplicaciones existentes en el dispositivo móvil. Se encarga de monitorear el funcionamiento de las aplicaciones y de administrar la ejecución del mismo, ya que todo equipo móvil tiene una función primordial que es la de comunicar voz, el AMS debe ser capaz de detener la aplicación en curso cuando ingrese una llamada o un mensaje de texto al dispositivo móvil.

Al detectar el ingreso de una llamada telefónica, el AMS invocará al método pausa del aplicativo, el cual colocará en espera a la aplicación para que el usuario móvil tenga la opción de contestar la llamada recibida. Una vez finalizada la conversación, el AMS preguntará si desea retomar la ejecución de la aplicación,

1.1.3.4. Ciclo de vida de un Midlet

El ciclo de vida representa la existencia del MIDlet desde el momento que el aplicativo se encuentra empaquetado. Las fases por las que pasa se indican en la siguiente figura:

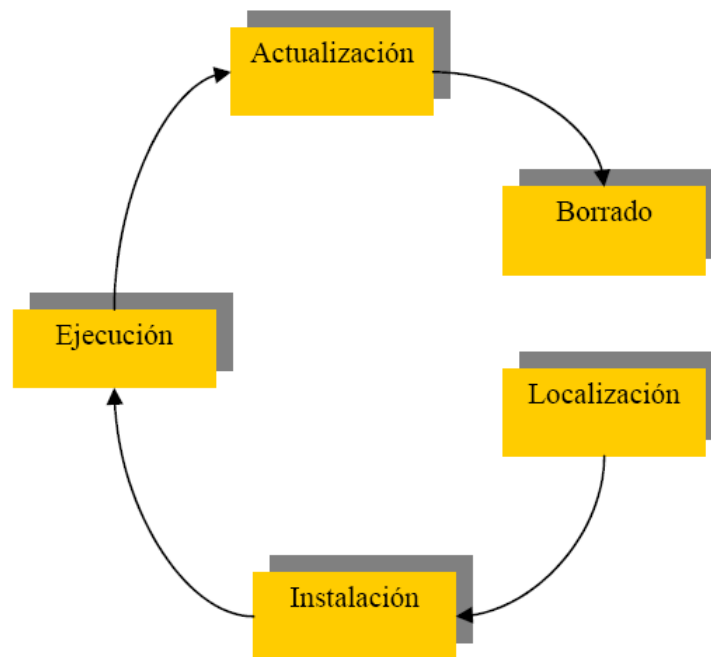


Figura 1.5. Fases del ciclo de vida de una aplicación

Localización:

Es la fase en la que el aplicativo es localizado para ser instalado en el dispositivo celular. Dicha localización se realiza por cualquiera de los medios de instalación usados, como cable de datos, infrarrojo, Bluetooth y OTA.

Instalación:

Es la fase en el cual el MIDlet es instalado en el equipo.

Ejecución:

Es la fase en la cual el MIDlet es activado para su ejecución. El AMS se encarga de cargar la aplicación y los elementos que implica para la ejecución, como la máquina virtual.

Actualización:

Es la fase en el cual el MIDlet es actualizado en el equipo.

Borrado:

Es la fase en el cual el MIDlet es eliminado del equipo.

1.1.3.4. Estados de Un Midlet

El AMS no solo se encarga de gestionar el ciclo de vida del MIDlet sino de manejar los estados de éste durante su ejecución, los cuales se indican en la siguiente figura:

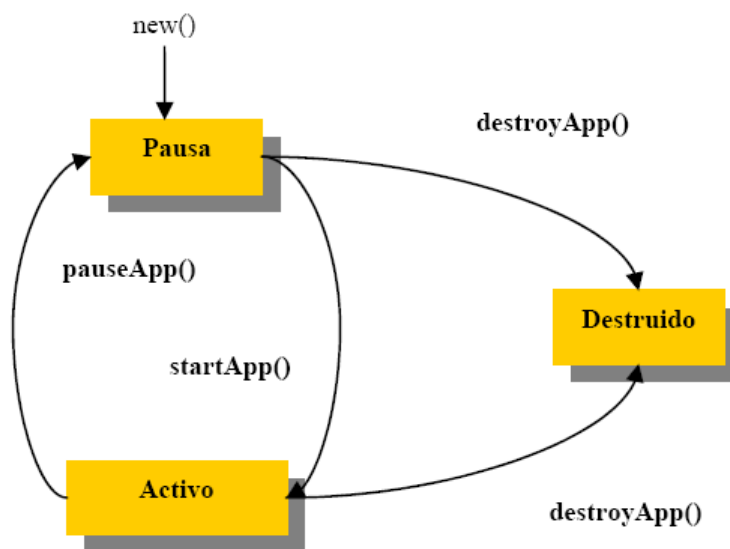


Figura 1.6. Estados de una aplicación en ejecución

- En primer lugar, el AMS crea una nueva instancia del MIDlet y realiza la llamada a su constructor, pasando éste al estado de “Pausa” durante un corto período de tiempo que es utilizado por el AMS para cargar en memoria el MIDlet y algunos recursos asociados.
- Cuando el dispositivo está preparado para ejecutar el MIDlet, el AMS invoca al método `MIDlet.startApp()` y pasa al estado de “Activo”. El MIDlet, entonces, ocupa todos los recursos que necesita para su ejecución.
- Durante el estado “Activo”, el MIDlet puede pasar al estado de “Pausa” por una acción del usuario, o bien, por el AMS (por ejemplo, por el ingreso de una llamada telefónica) que reduciría, en todo lo posible, el uso de los recursos del dispositivo por parte del MIDlet.

- Tanto en el estado “Activo” como en el de “Pausa”, el MIDlet puede pasar al estado “Destruído” realizando una llamada al método `MIDlet.destroyApp()`. Esto puede ocurrir porque el MIDlet haya finalizado su ejecución o porque una aplicación prioritaria necesite ser ejecutada en memoria en lugar del MIDlet. Una vez destruido el MIDlet, éste libera todos los recursos ocupados.

1.1.3.4. Estructura básica

Un MIDlet es una aplicación realizada usando el perfil MIDP; por lo tanto, para crear una aplicación con estas características se debe realizar lo siguiente:

- ✓ Crear una clase que herede de la clase `javax.microedition.midlet.MIDlet`, para que el AMS pueda gestionar sus estados y tener acceso a sus propiedades;
- ✓ Sobrescribir los métodos que el MIDlet posee, que son: `startApp`, `pauseApp` y `destroyApp`.

Asimismo, tomar en cuenta las siguientes observaciones:

- ✓ Los MIDlets, al igual que los applets carecen de la función `main()`. Aunque existiese, el gestor de aplicaciones la ignoraría por completo.
- ✓ Un MIDlet no puede realizar una llamada a `System.exit()`. Una llamada a este método lanzaría la excepción `SecurityException`.

La estructura básica de un MIDlet quedaría como sigue:

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

public class MiMidlet extends MIDlet{

    public MiMidlet(){
        /* Constructor de clase. Aquí se inicializan las variables. */
    }

    public startApp(){
        /* Incluir el código que el MIDlet ejecutará cuándo se active. */
    }

    public pauseApp(){
        /* Incluir el código que el MIDlet ejecutará cuándo se pause (Opcional) */
    }

    public destroyApp(){
        /* Incluir el código que el MIDlet ejecutará cuándo sea destruido.
         * Normalmente se liberaran los recursos ocupados por el MIDlet tales
         * como memoria, procesador, etc (Opcional) */
    }
}
```


1.1.3.4. La clase Midlet

Es la clase base de todas las aplicaciones JME basadas en el perfil MIDP. A continuación se describen sus métodos más importantes.

Constructor

Método que posee el mismo nombre de la clase sin argumentos. Aquí se pone el código que se ejecutará al inicio de la aplicación. Si la llamada a este constructor falla, se lanzaría la excepción `SecurityException`.

pauseApp()

- Definición

```
protected abstract void pauseApp( )
```

- Descripción

Método que será llamado por el AMS cuando el MIDlet pasa a un estado de "Pausa". Para esto, el MIDlet debe estar en estado "Activo". Cuando se crea un MIDlet, este método debe ser sobrescrito para indicar lo que se realizará cuando se produzca dicho cambio de estado, por ejemplo, grabar algún dato en la memoria persistente. Si ocurre una excepción `RuntimeException` durante la llamada a `MIDlet.pauseApp()`, el MIDlet será destruido inmediatamente. Se llamará a su método `MIDlet.destroyApp()` para liberar los recursos ocupados.

startApp()

- Definición

```
protected abstract void startApp( ) throws  
MIDletstateChangeException
```

- Descripción

Método que será llamado por el AMS cuando el MIDlet pasa a un estado "Activo". Para esto el MIDlet debe estar en estado de "Pausa". Cuando se crea un MIDlet, este método debe ser sobrescrito para indicar lo que se realizará cuando se produzca dicho cambio de estado, por ejemplo, mostrar la ventana inicial. En el caso de que el MIDlet no pueda pasar al estado "Activo" en este momento pero si pueda hacerlo en un momento posterior, se lanzaría la excepción `MIDletstateChangeException`.

destroyApp()

- Definición

```
protected abstract void ( boolean incondicional )  
throws MIDletstateChangeException
```

- Descripción

Método que será llamado por el AMS cuando el MIDlet pasa a un estado “Destruído”. Para esto el MIDlet debe estar en estado de “Pausa” o “Activo”. Cuando se crea un MIDlet, este método debe ser sobrescrito para indicar lo que se realizará cuando se produzca dicho cambio de estado, por ejemplo, liberar todos los recursos y/o salvar en el almacenamiento persistente algún dato que que deba ser guardado.

Si el parámetro ‘incondicional’ es false, el MIDlet puede lanzar la excepción MIDletStateException para indicar que no puede ser destruido en este momento. Si es true, el MIDlet asume su estado de destruido independientemente de como finalice el método.

notifyDestroyed()

-Definición

```
public final void notifyDestroyed()
```

-Descripción

Este método es usado para indicar al AMS que el MIDlet ha entrado en el estado de “Destruído”. Normalmente, se llama este método después de haber llamado a destroyApp(). El AMS considerará que todos los recursos que ocupaba el MIDlet están libres para su uso.

notifyPaused()

-Definición

```
public final void notifyPaused()
```

-Descripción

Este método es usado para indicar al AMS que el MIDlet ha entrado en el estado de “Pausa”. Este método sólo debe ser invocado cuándo el MIDlet esté en el estado “Activo”. Para volver al estado “Activo” se deberá llamar al método resumeRequest().

resumeRequest()

-Definición

```
public final void resumeRequest()
```

-Descripción

Este método proporciona un mecanismo a los MIDlets para indicar al AMS su cambio de estado a “Activo”. El AMS, en consecuencia, es el encargado de determinar qué aplicaciones han de pasar a este estado llamando al método startApp().

Importante.- Los métodos startApp(), pauseApp() y destroyApp() los utiliza el AMS para comunicarse con el MIDlet, mientras que los métodos resumeRequest(), notifyPaused()

y `notifyDestroyed()` los utiliza el MIDlet para comunicarse con el AMS.

checkPermission()

-Definición

```
public final int checkPermission(String permiso)
```

-Descripción

Devuelve el estado de un permiso para la aplicación. El permiso se especifica en el parámetro del mismo nombre. A continuación algunos ejemplos de permisos:

- `javax.wireless.messaging.sms.send`: Permiso para enviar mensajes de texto de tipo SMS
- `javax.microedition.pim.ContactList.read`: Permiso para leer la agenda de contactos
- `javax.microedition.io.Connector.bluetooth.client`: Permiso para conectarse como cliente a través de Bluetooth

Los valores devueltos por el método se corresponden con la siguiente descripción:

- 0: si el permiso ha sido denegado
- 1: si el permiso ha sido habilitado
- -1: si el estado es desconocido

Usando este método podemos verificar con anterioridad si el usuario ha autorizado a la aplicación a realizar ciertos servicios. De este modo, se podrá realizar algún tipo de notificación al usuario.

getAppProperty()

-Definición

```
public final String getAppProperty(String key)
```

-Descripción

Este método devuelve el valor de una propiedad guardada en el descriptor. El nombre de la propiedad a recuperar debe ir indicado en el parámetro `key`. El método nos devuelve un `String` con el valor de la propiedad o `null` si no existe ningún valor asociado al parámetro `key`. Si `key` es `null`, se lanzará la excepción `NullPointerException`.

platformRequest()

-Definición

```
public void boolean platformRequest(String url)
```

-Descripción

Este método permite llamar a algún servicio proporcionado por el dispositivo. Dicho servicio se especifica en el parámetro `url`, y varían de acuerdo al modelo del mismo.

1.1.3.5. Ejercicio 1: Aplicación Hola Mundo

Construir una aplicación que realice lo siguiente:

- Mostrar una ventana con el mensaje “Bienvenido a J2ME”
- Tendrá un botón para salir de la aplicación.



Se muestra a continuación el código necesario para implementar la funcionalidad solicitada:

```

// Aquí están el MIDlety clases relacionadas
import javax.microedition.midlet.*;
// Aquí están las clases de la interfaz de usuario
import javax.microedition.lcdui.*;

public class HolaMundo extends MIDlet implements CommandListener{

    // Variables a utilizar
    private Command botonSalir;
    private Display pantalla;
    private Form formulario;

    public HolaMundo(){
        // Inicializacion de variables
        pantalla = Display.getDisplay(this);
        botonSalir = new Command("Salir", Command.EXIT, 2);
        formulario = new Form("Hola");
        StringItem texto = new StringItem(" ", "Bienvenido a J2ME");

        // Adicion del StringItem
        formulario.append(texto);

        // Inicio de la captura de eventos de entrada
        formulario.addCommand(botonSalir);
        formulario.setCommandListener(this);
    }

    public void startApp() throws MIDletStateChangeException{
        // Inicio de la aplicación.
        // Se establece a formulario como la pantalla inicial
        pantalla.setCurrent(formulario);
    }

    public void pauseApp(){ }

    public void destroyApp(boolean unconditional){ }

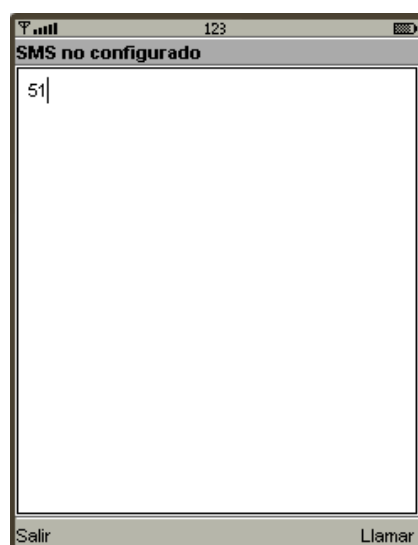
    public void commandAction(Command c, Displayable s){
        // Se ha pulsado un botón
        if (c == botonSalir){
            // Si el boton salir, salir de la aplicación se Notifica al Gestor
            // que finalice la aplicacion
            destroyApp(false);
            notifyDestroyed();
        }
    }
}

```

1.1.3.5. Ejercicio 2: Aplicación Hola Mundo

Construir una aplicación que realice lo siguiente:

- Permita ingresar un número telefónico en formato internacional. Para esto se usará un control TextBox, cuyo valor inicial será la propiedad “prefijo” (que debe ser creada en el descriptor).
- El título de la ventana indicará si la aplicación tiene o no el permiso de envío de mensajes.
- Tendrá dos botones: uno para salir y otro para realizar la llamada usando el método platformRequest().



Se muestra a continuación el código necesario para implementar la funcionalidad solicitada:

```

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

/* Esta aplicación utiliza algunas funciones de la clase MIDlet, como:
- getAppProperty: Para obtener el valor de una propiedad
- checkPermission: Para verificar el estado de un permiso
- platformRequest: Para efectuar una llamada a una función del equipo,
en este caso, para realizar una llamada telefónica
- notifyDestroyed: Para salir de la aplicación */

// La clase tiene que heredar de MIDlet
public class Llamada extends MIDlet implements CommandListener {

    // Se realiza lo siguiente:
    // 1. Obtener una referencia a la pantalla
    // 2. Crear una ventana de texto
    // 3. Crear dos comandos

    private Display dsp = Display.getDisplay(this);
    private TextBox txtTelefono=new TextBox("Telefono", "", 13, TextField.PHONENUMBER);
    private Command cmdLlamar = new Command("Llamar", Command.OK, 2);
    private Command cmdSalir = new Command("Salir", Command.EXIT, 2);

    public Llamada(){
        // Se lee la variable "prefijo" del descriptor
        if(getAppProperty("prefijo") != null)
            txtTelefono.setString(getAppProperty("prefijo"));

        // Se verifica el permiso para enviar mensajes
        int permiso = checkPermission("javax.wireless.messaging.sms.send");

        // Dependiendo del valor devuelto, se actualiza el título
        if (permiso == 0)
            txtTelefono.setTitle("SMS no permitidos");
        else if (permiso == 1)
            txtTelefono.setTitle("SMS permitidos");
        else
            txtTelefono.setTitle("SMS no configurado");

        // Se agregan los botones
        txtTelefono.addCommand(cmdLlamar);
        txtTelefono.addCommand(cmdSalir);
        txtTelefono.setCommandListener(this);
    }

    protected void destroyApp(boolean arg0){ }
    protected void pauseApp(){ }











    protected void startApp() throws MIDletStateChangeException {
        // Cuando se inicia la aplicación se muestra la ventana
        dsp.setCurrent(txtTelefono);
    }

    public void commandAction(Command c, Displayable d){
        // Se verifica el botón pulsado
        if(c == cmdLlamar){
            // Si es el botón Llamar, se realiza la llamada
            // usando la función platformRequest

            try{
                platformRequest("tel:" + txtTelefono.getString());
            } catch(Exception ex) {}
        } else if (c == cmdSalir) {
            // Si es el botón Salir, se realiza la llamada usando las funciones
            // destroyApp (que de momento está vacía) y notifyDestroyed()
            destroyApp(false);
            notifyDestroyed();
        }
    }
}

```

Resumen

-  Un *MIDlet* es una aplicación construida usando el perfil MIDP sobre la configuración CLDC. Ha sido diseñada para ser descargada a través de Internet vía OTA. Está conformada por dos archivos: el archivo jar (que contiene clases, recursos y un manifiesto) y el descriptor o archivo jad. Tanto el manifiesto como el descriptor contienen información sobre el contenido del *MIDlet*.
-  El Gestor de aplicaciones es una aplicación instalada en el dispositivo móvil y es quien controla todo el ciclo de vida del *MIDlet*, así como los estados por los que pasa durante su ejecución.
-  Para construir un *MIDlet*, el paso inicial es crear una clase que herede de la clase *MIDlet* y sobrescribir los métodos *startApp()*, *pauseApp()* y *destroyApp()*.
-  La clase *MIDlet* tiene adicionalmente métodos para cambio de estado (*notifyDestroyed()*, *notifyPaused()*, *resumeRequest()*), para obtener el estado de los permisos (*checkPermission()*) y para leer propiedades del descriptor (*getAppProperty()*)
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
 -  <http://www.mailxmail.com/curso/informatica/j2me>
 -  http://programacion.com/java/tutorial/ags_j2me/
 -  <http://leo.ugr.es/J2ME/MIDP/enlaces.htm>
-  Referencia oficial de Java (en inglés)
 -  <http://www.oracle.com/technetwork/java/javame/index.html>

UNIDAD DE
APRENDIZAJE

2

JME: INTERFACES

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno elabora aplicaciones que permitan la visualización de gráficos en 2D y el ingreso y visualización de datos, aplicando nociones de buenas prácticas de programación, con el empleo de las clases de la interfaz de alto nivel de la plataforma Java Micro Edition.

TEMARIO

2.1 Tema 2 : Desarrollo de interfaces

2.1.1. : Interfaces de alto y bajo nivel

2.1.2. : Clases contenedoras: TextBox, Alert, List

2.1.3. : Clases adicionales a los contenedores: Command, Ticker, Image

2.1.4. : Clases Ítem: TextField, StringItem, ChoiceGroup, ImageItem, DateField

2.2 Tema 3 : Manejo de eventos

2.2.1. : Configuración de escuchadores: ItemStateListener, ItemCommandListener

2.3 Tema 4 : Graficador

2.3.1. : Introducción a la interface gráfica

2.3.2. : Clases Canvas y Graphics

ACTIVIDADES PROPUESTAS

- Los alumnos crean aplicaciones utilizando contenedores de Alto y Bajo nivel JME

2.1 Desarrollo de interfaces

Las interfaces de usuarios se refieren a las clases que van a gestionar el manejo de la pantalla de los dispositivos móviles. Para ello, JME define 2 tipos de interfaces, con las cuales el desarrollador puede generar las pantallas que interactuarán con el usuario final.

2.1.1. Interfaces de alto y bajo nivel

2.1.1.1. Alto nivel

Está diseñada para manejar componentes portables como cajas de texto, botones, imágenes, formularios, etc. En esta interfaz, el usuario depende mucho de la pantalla del dispositivo móvil, ya que éste define la forma en que serán visualizados.

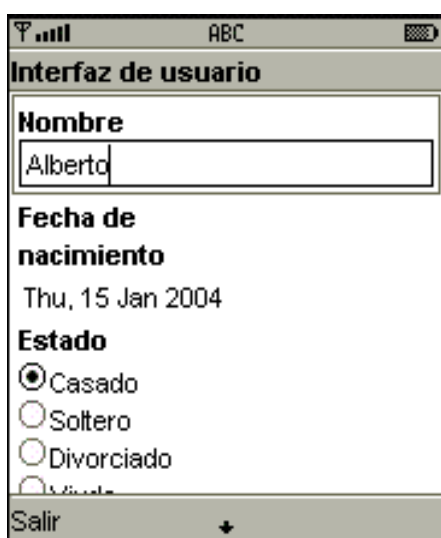


Figura 2.1. Ejemplo de interfaz de alto nivel

2.1.1.2. Bajo nivel

Aquí se tiene control total de toda la pantalla del dispositivo móvil ya que se dará un control completo sobre los recursos del dispositivo y podremos controlar eventos de bajo nivel como, por ejemplo, el rastreo de pulsaciones de teclas. Como contraparte, las aplicaciones construidas con esta interfaz son menos portables. Generalmente, esta interfaz se utiliza para la creación de juegos donde el control sobre lo que aparece por pantalla y las acciones del usuario juegan un papel fundamental.

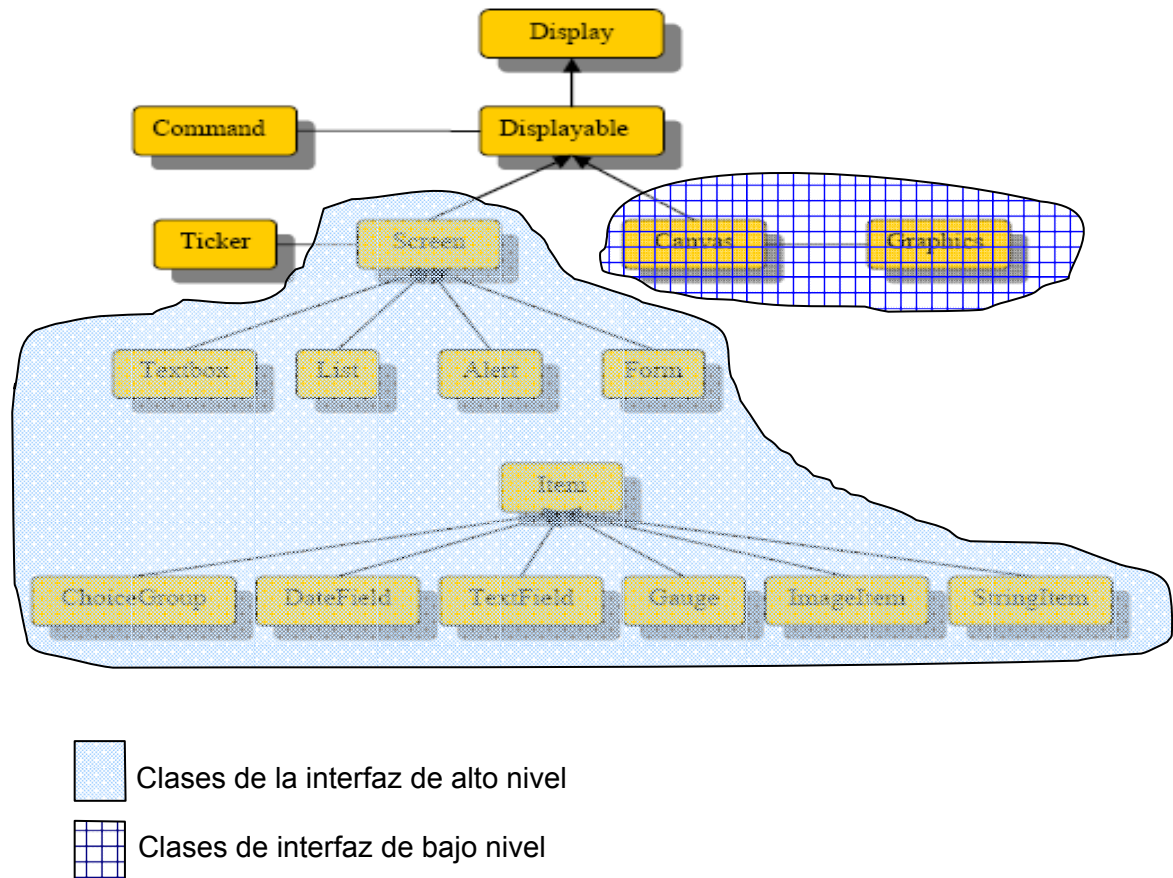


Figura 2.2. Diagrama de clases de las interfaces de alto y bajo nivel

2.1.1.1. Ejercicio: Aplicación Datos Display

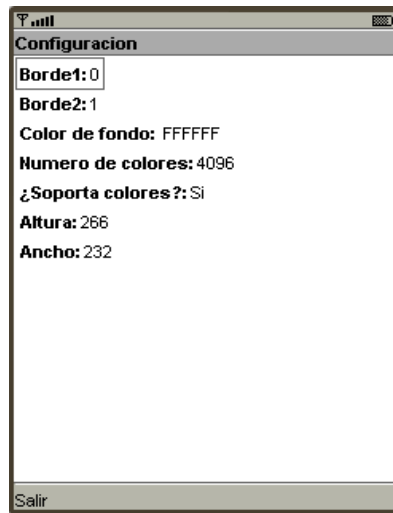
Construir una aplicación que realice lo siguiente:

Mostrar una ventana con el título “Configuración” y con las siguientes propiedades de la pantalla:

- ✓ Estilo del borde cuando los controles están resaltados
- ✓ Estilo del borde cuando los controles no están resaltados
- ✓ Color de fondo
- ✓ Numero de colores
- ✓ Un valor que indique si soporta colores o no.
- ✓ Alto de la pantalla
- ✓ Ancho de la pantalla

Tendrá un botón para salir de la aplicación

Se muestra a continuación un ejemplo de la pantalla que debe implementar:



El código necesario para implementar la funcionalidad solicitada será el siguiente:

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

public class DatosDisplay extends MIDlet implements CommandListener{
    // Variables a utilizar
    private Command botonSalir; // Para manejar los eventos de entrada
    private Display pantalla;    // Para manipular la pantalla
    private Form formulario;     // Clase que hereda de Displayable

    public DatosDisplay(){
        botonSalir = new Command("Salir", Command.EXIT, 2);
        formulario = new Form("Configuracion");
        // Obtiene una referencia de la pantalla
        pantalla = Display.getDisplay(this);

        // Variables de tipo StringItem con datos diversos
        // Estilo del borde de los controles cuando estan resaltados
        StringItem borde = new StringItem("Borde1:",
            String.valueOf(pantalla.getBorderStyle(true)));

        // Estilo del borde de los controles cuando no estan resaltados
        StringItem borde2 = new StringItem("Borde2:",
            String.valueOf(pantalla.getBorderStyle(false)));

        // Color usado para el fondo de la pantalla
        int colorFondo = pantalla.getColor(Display.COLOR_BACKGROUND);
        StringItem color = new StringItem("Color de fondo: ",
            Integer.toHexString(colorFondo).toUpperCase());

        /*NOTA: La funcion getColor puede recibir otros parametros constantes
        * Cuando la pantalla no está iluminada
        * Display.COLOR_BACKGROUND: Color de fondo
        * Display.COLOR_BORDER: Color del borde
        * Display.COLOR_FOREGROUND: Color de la letra

        * Cuando la pantalla está iluminada
        * Display.COLOR_HIGHLIGHTED_BACKGROUND: Color de fondo
        * Display.COLOR_HIGHLIGHTED_BORDER: Color del borde
        * Display.COLOR_HIGHLIGHTED_FOREGROUND: Color de la letra
        */
        // Numero de colores en total que soporta la pantalla
        StringItem numColores = new StringItem("Numero de colores:",
            String.valueOf(pantalla.numColors()));

        // Valor que nos indica si la pantalla soporta colores o no
        StringItem esColor = new StringItem("¿Soporta colores?:",
            String.valueOf(((pantalla.isColor())? "Si": "No")));
    }
}
```

```

// Altura de la pantalla
StringItem altura = new StringItem("Altura:",
    String.valueOf(formulario.getHeight()));

// Ancho de la pantalla
StringItem ancho = new StringItem("Ancho:",
    String.valueOf(formulario.getWidth()));

// Adicion de los objetos de tipo StringItem con los datos
formulario.append(borde);
formulario.append(borde2);
formulario.append(color);
formulario.append(numColores);
formulario.append(esColor);
formulario.append(altura);
formulario.append(ancho);

// Adicion del comando e inicio de la captura de eventos
formulario.addCommand(botonSalir);
formulario.setCommandListener(this);
}

public void startApp() throws MIDletStateChangeException {
    // Inicio de la aplicacion
    // Se establece a formulario como la pantalla inicial
    pantalla.setCurrent(formulario);
}

public void pauseApp(){ }

public void destroyApp(boolean unconditional) { }

public void commandAction(Command c, Displayable d){
    // Se ha pulsado un boton
    if (c == botonSalir) {
        // Si es el boton Salir, se realiza la llamada
        // usando las funciones destroyApp
        // (que de momento esta vacia)
        // y notifyDestroyed()
        destroyApp(false);
        notifyDestroyed();
    }
}
}

```

2.1.2. Clases contenedoras: TextBox, Alert, List, Form

2.1.2.1 TextBox

Esta clase se encarga de representar a una caja de texto para el ingreso de datos de diferente tipo. Este control ocupará toda la pantalla del dispositivo móvil. Se describen a continuación sus principales métodos:

Método	Sintaxis	Definición
Constructor	<pre>public TextBox (String titulo, String textoInicial, int longitudMaxima, int filtros)</pre>	<p>Permite construir un control de tipo TextBox.</p> <p>Sus parámetros son:</p> <ul style="list-style-type: none"> título: Título a mostrar en la parte superior de la pantalla. textoInicial: Texto mostrado al inicio. longitudMaxima: Longitud máxima de caracteres. <p><u>Ejemplo:</u> El siguiente fragmento crea una ventana de texto que permite el ingreso de una contraseña numérica de máximo 10 dígitos e indica que es un dato sensible.</p> <pre>private TextBox txt = new TextBox("Clave", "", 10, TextField.NUMERIC TextField.PASSWORD TextField.SENSITIVE);</pre>
delete()	<pre>public void delete(int desplazamiento, int longitud)</pre>	<p>Borra caracteres del TextBox.</p> <p>Sus parámetros son::</p> <ul style="list-style-type: none"> desplazamiento: Posición inicial a borrar longitud: Cantidad de caracteres a eliminar
getCaretPosition()	<pre>public int getCaretPosition()</pre>	Devuelve la posición del cursor en pantalla.
getChars()	<pre>public int getChars(char [] datos)</pre>	Copia el contenido del TextBox en el arreglo "datos" y devuelve el número de caracteres copiados.
getMaxSize()	<pre>public int getMaxSize()</pre>	Devuelve el tamaño máximo del TextBox.
getString()	<pre>public String getString()</pre>	Devuelve el contenido del TextBox.

insert()	<pre>public void insert (char[] datos, int des, int long, int pos) public void insert (char[] datos, int pos)</pre>	Inserta el contenido del arreglo datos en el TextBox. <ul style="list-style-type: none"> • datos: arreglo a copiar • des: posición del arreglo desde donde se copiará • long: cantidad de caracteres que se copiarán • pos: posición en el TextBox donde se colocarán los caracteres.
setChars()	<pre>public void setChars (char[] datos, int des, int long)</pre>	Reemplaza el contenido del TextBox por el arreglo "datos". <ul style="list-style-type: none"> • datos: arreglo a copiar • des: posición del arreglo desde donde se copiará • long: cantidad de caracteres que se copiarán
setMaxSize()	<pre>public int setMaxSize(int capacidad)</pre>	Establece el tamaño máximo del TextBox. Devuelve el tamaño que realmente se le ha asignado.
setString()	<pre>public void setString(String texto)</pre>	Establece el contenido del TextBox

2.1.2.2 Alert

Esta clase se utiliza para mostrar mensajes en toda la pantalla del dispositivo móvil. Tiene un tiempo de visualización (timeout), luego de lo cual desaparece (por defecto es 2000 milisegundos), así como un tipo que define su comportamiento. Se describen a continuación sus principales métodos:

Método	Sintaxis	Definición
Constructor	<pre>public Alert (String titulo, String mensaje, Image imagen, AlertType tipoAlerta) public Alert (String titulo) Donde tipoAlerta puede tomar uno de los siguientes valores: • AlertType.INFO • AlertType.WARNING • AlertType.ERROR • AlertType.ALARM • AlertType.CONFIRMATION</pre>	Permite construir un control de tipo Alert. Sus parámetros son: <ul style="list-style-type: none"> • título: Título a mostrar en la parte superior de la pantalla • mensaje: Mensaje que se mostrará • imagen: Imagen que se mostrará. Si no se desea mostrar ninguna imagen el valor debe ser null. • tipoAlerta: Es un entero que define el comportamiento que tendrá el Alert. Este comportamiento depende del dispositivo y puede consistir en mostrar un sonido, realizar una vibración, etc.

		<p>Ejemplo El siguiente fragmento crea un Alert de tipo INFO, con el título “Éxito” y el mensaje “Proceso culminó con éxito”.</p> <pre>private Alert alt = new Alert("Exito", "Proceso culmino con exito", null, AlertType.INFO);</pre> <p>El siguiente fragmento crea un Alert con el título “Éxito” y el resto de parámetros por defecto, esto indica:</p> <pre>private Alert alt = new Alert("Exito");</pre>
getImage()	public Image getImage()	Devuelve la imagen configurada para la alerta.
getString()	public String getString()	Devuelve el mensaje configurado para la alerta.
getTimeout()	public int getTimeout()	Devuelve el tiempo que se mostrará la alerta (en milisegundos).
getType()	public AlertType getType()	Devuelve el tipo de alerta definido para un objeto Alert.
setTimeout()	public void setTimeout (int timeout)	Define el tiempo que se mostrará un objeto Alert (en milisegundos). Se puede usar la constante <code>Alert.FOREVER</code> para indicar que el Alert no va a desaparecer automáticamente.
setType()	public void setType (AlertType tipo)	Define el tipo de alerta que se mostrará.

2.1.2.3 List

Esta clase nos permite utilizar diferente tipos de lista de opciones en la pantalla del dispositivo móvil. Puede ser utilizado como una lista de selección, ya sea única o múltiple, o para la elaboración de un menú con el que interactuará el usuario de la aplicación. Sus elementos se identifican por un índice o posición que comienza en cero (0). Se describen a continuación sus principales métodos:

Método	Sintaxis	Definición
Constructor	<pre>public List (String titulo, int tipo) public List (String titulo, int tipo, String[] elementos, Image[] imagenes)</pre>	<p>Permite construir un control de tipo List.</p> <p>Sus parámetros son:</p> <ul style="list-style-type: none"> • Título: Título a mostrar en la parte superior de la pantalla • tipo: Dato entero que indica el tipo de lista. • elementos: arreglo conteniendo la lista inicial de elementos • imágenes: arreglo del mismo tamaño de “elementos”, conteniendo sus imágenes, o null si no existieran imágenes. <p><u>Ejemplo:</u> El siguiente fragmento crea una lista exclusiva con 3 elementos.</p> <pre>private String astrTipoDoc[] = {"Partida", "DNI", "Pasaporte"}; private List lst = new List("Tipo Documento", List.EXCLUSIVE, astrTipoDoc, null);</pre> <p>El siguiente fragmento crea una lista implícita con 3 elementos:</p> <pre>private String astrOpciones[] = {"Ingreso", "Reportes", "Salir"}; private List lst = new List("Opciones", List.IMPLICIT, astrOpciones, null);</pre> <p>El siguiente fragmento crea una lista multiple sin elementos:</p> <pre>private List lst = new List("Pasatiempos", List.MULTIPLE);</pre>
append()	<pre>public int append (String texto, Image img)</pre>	<p>Añade un elemento al final de la lista y devuelve su índice en la lista.</p> <ul style="list-style-type: none"> • texto: Elemento a agregar • img: Imagen del elemento a agregar o null si no existiera.
delete()	<pre>public void delete(int pos)</pre>	<p>Elimina el elemento en la posición que se especifica.</p>
deleteAll()	<pre>public void deleteAll()</pre>	<p>Elimina todos los elementos de la lista.</p>
getFont()	<pre>public Font getFont(int pos)</pre>	<p>Devuelve la fuente del elemento en la posición indicada por “pos”.</p>
getImage()	<pre>public Image getImage(int pos)</pre>	<p>Devuelve la imagen del elemento en la posición indicada por “pos”.</p>

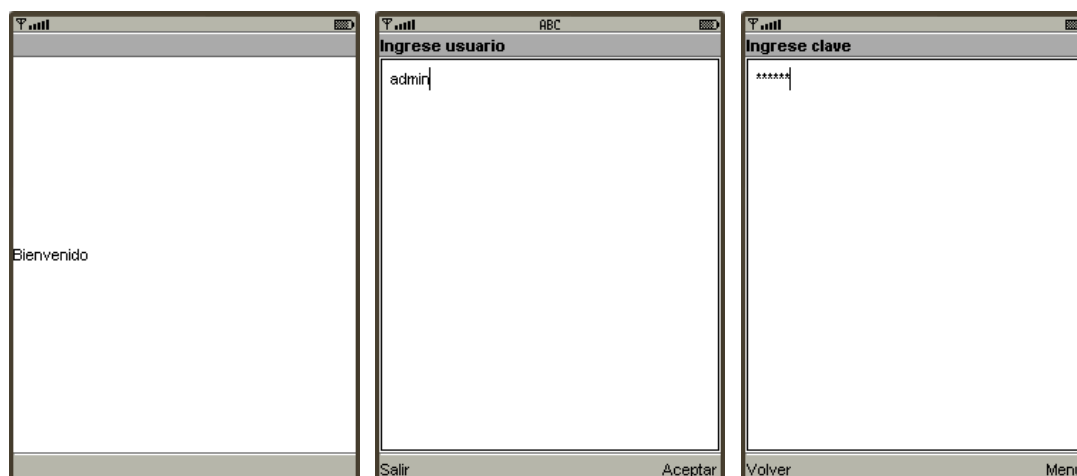
getSelectedFlags()	public int getSelectedFlags (boolean[] estado)	Llena el arreglo “estado” con el estado de los elementos (true si fueron seleccionados, false si no) y devuelve el número de elementos seleccionados. Este método se usa para las listas múltiples.
getSelectedIndex()	public int getSelectedIndex()	Obtiene el índice del elemento seleccionado. Este método se usa para las listas exclusiva e implícita.
isSelected()	public boolean isSelected(int pos)	Devuelve un valor que indica si el elemento señalado por el parámetro “pos” está seleccionado.

2.1.2.4 Ejercicio: Aplicación InicioSesion

Construir una aplicación que realice lo siguiente:

- Mostrar una ventana inicial de bienvenida por 2 segundos.
- Luego mostrar una ventana que permita el ingreso de un usuario. Esta ventana tendrá botones para salir y aceptar.
- Si se pulsa aceptar, mostrar una ventana que permita el ingreso de una contraseña. Esta ventana tendrá botones para salir, volver y aceptar.
- Si se pulsa aceptar, verificar que el usuario y contraseña sean iguales a las propiedades “usuario” y “clave” del descriptor. Se deben crear previamente dichas propiedades. Mostrar una ventana de éxito o error según sea el caso.

Se muestra a continuación las pantallas a implementar en el presente ejercicio:



Debe implementar el siguiente código:

```
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
```

```

public class InicioSesion extends MIDlet implements CommandListener {

    // Obtener referencia de la pantalla
    private Display dsp = Display.getDisplay(this);

    // Creacion del Alert de bienvenida
    private Alert altBienvenida = new Alert("", "Bienvenido", null, AlertType.INFO);
    // Creacion del TextBox para el ingreso del usuario
    private TextBox txtUsuario = new TextBox("Ingrese usuario", "", 10, TextField.ANY);
    // Creacion del TextBox para el ingreso de clave
    private TextBox txtClave = new TextBox("Ingrese clave", "", 10, TextField.ANY |
                                           TextField.PASSWORD);

    // Creacion del Alert para los mensajes de error
    private Alert altError = new Alert("Error", "", null, AlertType.ERROR);

    // Creacion de botones
    private Command cmdSalir = new Command("Salir", Command.EXIT, 2);
    private Command cmdVolver = new Command("Volver", Command.BACK, 2);
    private Command cmdAceptar = new Command("Aceptar", Command.OK, 2);

    public InicioSesion() {
        // Agregar botones e indicar la clase que procesara los eventos
        txtUsuario.addCommand(cmdSalir);
        txtUsuario.addCommand(cmdAceptar);
        txtUsuario.setCommandListener(this);

        // Agregar botones e indicar la clase que procesara los eventos
        txtClave.addCommand(cmdSalir);
        txtClave.addCommand(cmdAceptar);
        txtClave.addCommand(cmdVolver);
        txtClave.setCommandListener(this);
    }

    protected void destroyApp(boolean arg0) { }
    protected void pauseApp() { }

    protected void startApp() throws MIDletStateChangeException {
        // Mostrar el Alert de bienvenida y luego el TextBox de usuario
        dsp.setCurrent(altBienvenida, txtUsuario);
    }

    public void commandAction(Command c, Displayable d) {
        // Se pulso un boton
        if(c == cmdSalir) {
            // Si el boton es salir, salir de la aplicacion
            destroyApp(false);
            notifyDestroyed();
        } else if(c == cmdAceptar) {
            // Si el boton es Aceptar, verificar cual es la ventana
            if(d == txtUsuario) {
                // Si es la ventana de Usuario, validar el texto ingresado.
                // Si el texto es correcto, ir a la ventana de Clave
                if(!txtUsuario.getString().equals("") && txtUsuario.getString().equals(
                    getAppProperty("Usuario"))){
                    dsp.setCurrent(txtClave);
                } else {
                    // Si el texto es incorrecto, mostrar mensaje
                    altError.setString("Usuario no valido");
                    dsp.setCurrent(altError);
                }
            } else if (d == txtClave) {
                // Si es la ventana de Clave, validar el texto ingresado.
                // Si el es correcto, mostrar los datos en un Alert
                if(!txtClave.getString().equals("") && txtClave.getString().equals(
                    getAppProperty("Clave"))){
                    // Crear una cadena con el usuario y la clave
                    String strMensaje = "Ingreso correcto" +
                        "\n\rUsuario: " + txtUsuario.getString() +
                        "\n\rClave: " + txtClave.getString();
                    Alert altIngresoCorrecto = new Alert(
                        "Exit", strMensaje, null, AlertType.INFO);
                }
            }
        }
    }
}

```

```

        // Cambiarle el timeout para que no desaparezca
        altIngresoCorrecto.setTimeout(Alert.FOREVER);
        // Mostrar Alert
        dsp.setCurrent(altIngresoCorrecto);
    } else {
        // Si no es correcto, mostrar un mensaje de error
        altError.setString("Usuario no valido");
        dsp.setCurrent(altError);
    }
}
} else if(c == cmdVolver) {
    // Si el boton es Volver, ir a la ventana de Usuario
    dsp.setCurrent(txtUsuario);
}
}
}

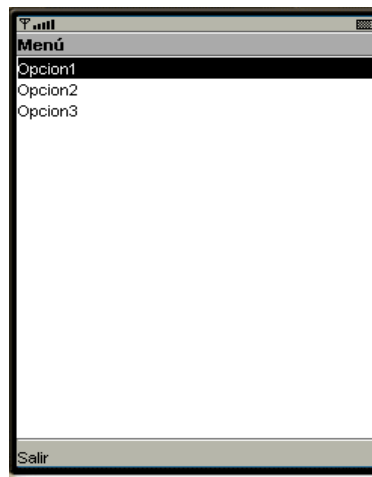
```

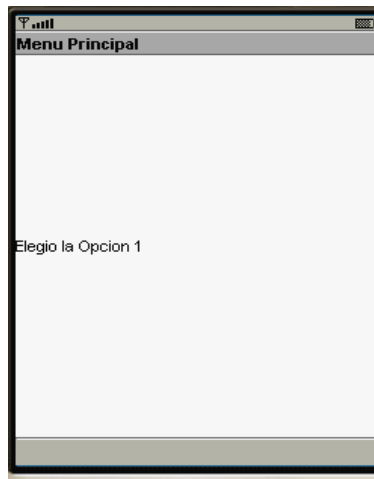
2.1.2.5 Ejercicio: Aplicación MenuInicial

Construir una aplicación que realice lo siguiente:

- Mostrar un menú que, al seleccionar una opción, muestre, en un Alert, un mensaje
- Adicionalmente, debe tener un botón “salir”.

Se muestra a continuación las pantallas a implementar en el presente ejercicio:





Debe implementar el siguiente código:

```

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

public class MenuInicial extends MIDlet implements CommandListener{
    private Command salir; //Declaracion de Variables
    private Display pantalla;
    private List menu;
    private Alert alerta;

    public MenuInicial(){
        pantalla = Display.getDisplay(this); //Creacion de pantallas

        menu = new List("Menú",List.IMPLICIT);
        menu.insert(0,"Opcion3",null);
        menu.insert(0,"Opcion2",null);
        menu.insert(0,"Opcion1",null);

        alerta=new Alert("Menu Principal");
        alerta.setTimeout(3000);

        salir = new Command("Salir",Command.EXIT,1);

        menu.addCommand(salir);
        menu.setCommandListener(this);
    }

    public void startApp(){
        pantalla.setCurrent(menu); // Coloca el menu en pantalla
    }

    public void pauseApp(){ }
    public void destroyApp(boolean unconditional) { }

    public void commandAction(Command c, Displayable d) {
        if (c == List.SELECT_COMMAND && d == menu){
            // Si se seleccionó la opción del menu
            switch(menu.getSelectedIndex()){
                case 0:
                    alerta.setString("Eligio la Opcion 1");
                    pantalla.setCurrent(alerta);
                    break;
                case 1:
                    alerta.setString("Eligio la Opcion 2");
                    pantalla.setCurrent(alerta);
                    break;
                case 2:
                    alerta.setString("Eligio la Opcion 3");
                    pantalla.setCurrent(alerta);
                    break;
            }
        } else if (c == salir) {
            // Salir de la aplicacion
            destroyApp(false);
            notifyDestroyed();
        }
    }
}

```

2.1.2.6 Form

Un formulario es un contenedor de controles donde cada control nos va a permitir el ingreso y/o visualización de un dato. Los controles se almacenan en un arreglo dentro del formulario donde el primer elemento tiene el índice cero (0). Todos los controles son objetos de tipo Item, aunque, generalmente, se usarán clases que heredan de Item, los cuales se grafican en la siguiente figura:

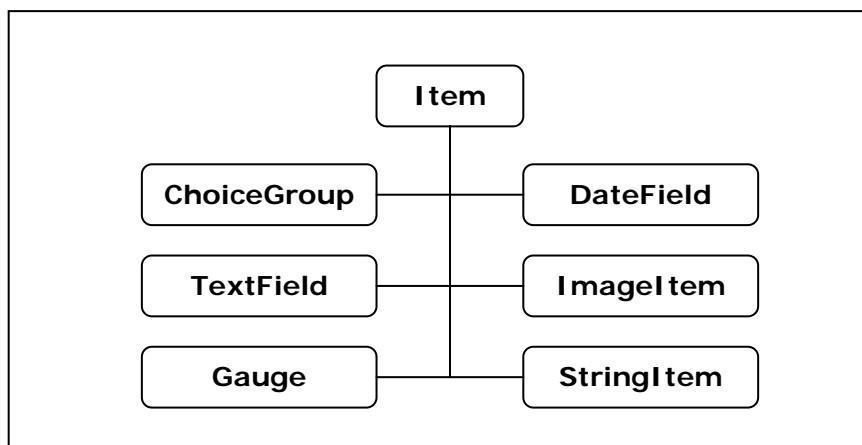


Figura 2.3 - Jerarquía de clases tipo Item

2.1.3. Clases adicionales a los contenedores: Image, Ticker

2.1.3.1 Image

Esta clase nos permite visualizar una imagen. El formato estándar para los archivos de imagen de los dispositivos móviles es PNG, aunque dependiendo del equipo se puede trabajar con otros formatos conocidos, como GIF, JPG, etc. No existe un constructor para una imagen, en lugar de eso se usa el método estático `createImage`. Sólo algunas clases, como `Alert`, `List` o `Form`, pueden mostrar una imagen.

2.1.3.2 Ticker

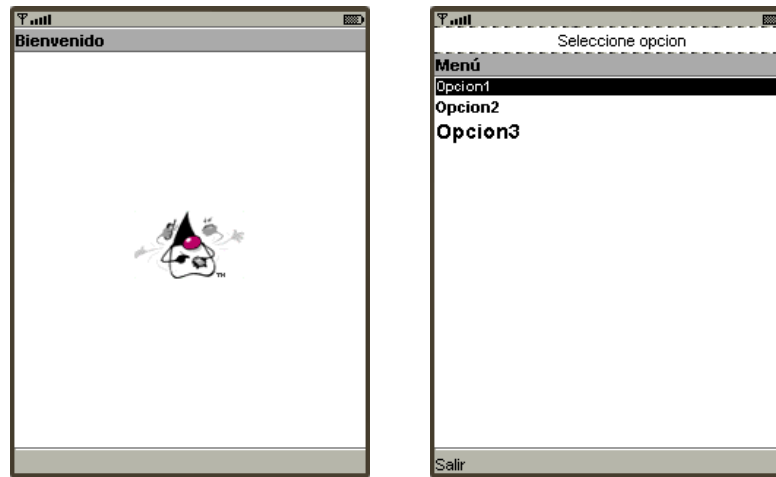
Un objeto `Ticker` define una especie de marquesina que aparecerá en la parte superior de la pantalla. Todos los controles que heredan de `Displayable` pueden mostrar un `Ticker`. Para esto, se debe usar el método `setTicker`.

2.1.3.3 Ejemplo: Aplicación MenuInicial

Modificar el ejercicio anterior para que realice lo siguiente:

- Mostrar una ventana inicial con una imagen
- Agregar un `Ticker` al menú con el texto “Seleccione opción”
- Modificar los tipos de letra de las opciones del menú

La siguiente figura indica como deberá verse la pantalla.



Debe implementar el siguiente código:

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

public class MenuInicial extends MIDlet implements CommandListener{
    private Command salir; //Declaracion de Variables
    private Display pantalla;
    private List menu;
    private Alert alerta;

    private Alert bienvenida; // Ventana de bienvenida
    private Image imgDuke; // Imagen

    public MenuInicial(){
        pantalla = Display.getDisplay(this); //Creacion de pantallas

        menu = new List("Menú",List.IMPLICIT);
        menu.insert(0,"Opcion3",null);
        menu.insert(0,"Opcion2",null);
        menu.insert(0,"Opcion1",null);

        // Creación de fuentes
        Font fnt0=Font.getFont(Font.FACE_PROPORTIONAL,Font.STYLE_BOLD,Font.SIZE_SMALL);
        Font fnt1=Font.getFont(Font.FACE_PROPORTIONAL,Font.STYLE_BOLD,Font.SIZE_MEDIUM);
        Font fnt2=Font.getFont(Font.FACE_PROPORTIONAL,Font.STYLE_BOLD,Font.SIZE_LARGE);

        // Cambiar las fuentes del menú
        menu.setFont(0, fnt0);
        menu.setFont(1, fnt1);
        menu.setFont(2, fnt2);

        Ticker ticker = new Ticker("Seleccione opcion");//Crear el Ticker
        menu.setTicker(ticker); // Mostrar el Ticker

        alerta=new Alert("Menu Principal");
        alerta.setTimeout(3000);

        // Crear la imagen
        try{
            imgDuke = Image.createImage("/duke.png");
        } catch(Exception ex) {
            imgDuke = null;
        }

        // Crear el alert y agregarle la imagen
        bienvenida = new Alert("Bienvenido");
        bienvenida.setTimeout(3000);
        bienvenida.setImage(imgDuke);

        salir = new Command("Salir",Command.EXIT,1);
        menu.addCommand(salir);
        menu.setCommandListener(this);
    }
}
```

```

public void startApp(){
    // Mostrar el Alert y luego el menú
    pantalla.setCurrent(bienvenida, menu);
}

public void pauseApp() { }
public void destroyApp(boolean unconditional) { }

public void commandAction(Command c, Displayable d) {
    if (c == List.SELECT_COMMAND && d == menu) {
        // Si selecciono opcion del menu
        switch(menu.getSelectedIndex()){
            case 0:
                alerta.setString("Elegio la Opcion 1");
                pantalla.setCurrent(alerta);
                break;
            case 1:
                alerta.setString("Elegio la Opcion 2");
                pantalla.setCurrent(alerta);
                break;
            case 2:
                alerta.setString("Elegio la Opcion 3");
                pantalla.setCurrent(alerta);
                break;
        }
    } else if (c == salir) {
        // Seleccione salir de la aplicacion
        destroyApp(false);
        notifyDestroyed();
    }
}
}

```

2.1.4 Clases Ítem: TextField, StringItem, ChoiceGroup, ImageItem, DateField

2.1.4.1 TextField

Esta clase nos permite crear objetos de tipo caja de texto dentro de un formulario. Es similar al objeto TextBox, con la diferencia que pueden existir varios objetos de tipo TextField al mismo tiempo en un solo formulario

2.1.4.2 StringItem

La clase StringItem es una clase similar a una etiqueta, la cual podrá ser insertada en un formulario. Esta clase también permite simular un botón o hyperlink.

2.1.4.3 ChoiceGroup

Esta clase nos permite crear listas de selección que podrán ser incrustados en un formulario, tal como lo hacía el objeto List.

2.1.4.4 ImageItem

La clase ImageItem permite la manipulación de imágenes que podrán ser incrustados en un formulario. Esta clase al igual que la clase StringItem, permite simular un objeto de tipo boton o hyperlink.

2.1.4.5 DateField

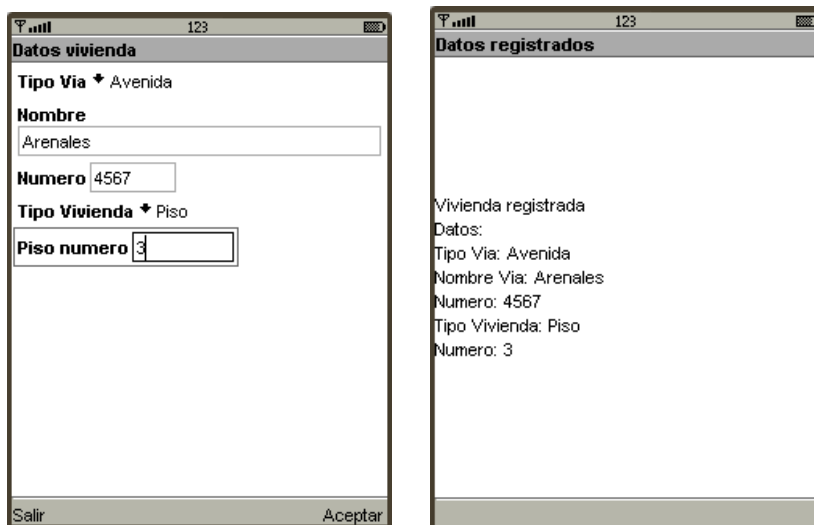
Esta clase nos permite crear objetos de tipo caja de texto especializadas en el ingreso de valores de tipo fecha y hora.

2.1.4.6 Ejercicio:

Construir una aplicación que realice lo siguiente:

- Mostrar una ventana con el título “Datos vivienda”, que permita ingresar los siguientes datos:
- Tipo vía: Avenida, Calle o Pasaje
- Nombre vía: Dato alfanumérico, 100 caracteres como máximo
- Número: Dato numérico, 5 dígitos como máximo
- Tipo vivienda: Casa, piso, departamento, habitación
- Sub Número: Dato numérico, 5 dígitos como máximo. Solo se muestra si el tipo de vivienda no es Casa
- La aplicación tendrá dos botones: uno para salir y otro para mostrar una ventana que muestre los datos que se han ingresado.

La siguiente figura indica cómo deberá verse la pantalla.



Debe implementar el siguiente código:

```
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.ItemStateListener;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
```

```

public class DatosVivienda extends MIDlet implements CommandListener,
                                   ItemStateListener {

    // Referencia a la pantalla
    private Display dsp = Display.getDisplay(this);

    // Creación del formulario y sus controles
    private Form frmVivienda = new Form("Datos vivienda");
    private String astrTiposVia[] = {"Avenida", "Calle", "Pasaje"};
    private ChoiceGroup chgTipoVia = new ChoiceGroup("Tipo Via",
                                                    ChoiceGroup.POPUP,
                                                    astrTiposVia,
                                                    null);

    private TextField txfNombreVia = new TextField("Nombre", "", 100, TextField.ANY);
    private TextField txfNumero = new TextField("Numero", "", 5, TextField.NUMERIC);

    private String astrTiposVivienda[] = {"Casa", "Piso", "Departamento", "Habitacion"};
    private ChoiceGroup chgTipoVivienda = new ChoiceGroup("Tipo Vivienda",
                                                         ChoiceGroup.POPUP,
                                                         astrTiposVivienda,
                                                         null);

    private TextField txfSubNumero = new TextField("", "", 5, TextField.NUMERIC);

    // Creación de los botones
    private Command cmdAceptar = new Command("Aceptar", Command.OK, 2);
    private Command cmdSalir = new Command("Salir", Command.EXIT, 2);

    public DatosVivienda(){
        // Agregar controles
        // El control de txfSubNumero no se agrega en el constructor.
        // Este control se agregara solo si se selecciona un elemento
        // diferente de "Casa"
        frmVivienda.append(chgTipoVia);
        frmVivienda.append(txfNombreVia);
        frmVivienda.append(txfNumero);
        frmVivienda.append(chgTipoVivienda);

        // Agregar botones
        frmVivienda.addCommand(cmdAceptar);
        frmVivienda.addCommand(cmdSalir);

        // Indicar la clase que procesara los eventos
        frmVivienda.setCommandListener(this);
        frmVivienda.setItemStateListener(this);
    }

    protected void destroyApp(boolean arg0) { }

    protected void pauseApp() { }

    protected void startApp() throws MIDletStateChangeException {
        // Establecer la ventana inicial
        dsp.setCurrent(frmVivienda);
    }

    public void commandAction(Command c, Displayable d) {
        // Se pulso un botón
        if(c == cmdAceptar) {
            // Crear el mensaje que se mostrará
            // Se concatenan todos los datos
            String strMensaje = "Vivienda registrada\n\rDatos:"
                + "\n\rTipo Via: " + chgTipoVia.getString(chgTipoVia.getSelectedIndex())
                + "\n\rNombre Via: " + txfNombreVia.getString()
                + "\n\rNumero: " + txfNumero.getString()
                + "\n\rTipo Vivienda: " + chgTipoVivienda.getString(
                    chgTipoVivienda.getSelectedIndex()) +
                "\n\rNumero: " + txfSubNumero.getString();

            // Crear el Alert correspondiente
            Alert altMensaje = new Alert("Datos ingresados",
                                         strMensaje,
                                         null,
                                         AlertType.CONFIRMATION);
        }
    }
}

```

```

        // Mostrar el Alert
        dsp.setCurrent(altMensaje);
    } else if (c == cmdSalir) {
        // Salir de la aplicacion
        destroyApp(false);
        notifyDestroyed();
    }
}

public void itemStateChanged(Item i) {
    // Se modifiko el estado de algun elemento. Se verifica cual fue
    if(i == chgTipoVivienda) {
        // Si el elemento fue chgTipoVivienda
        // Verificar si el indice es diferente de 0
        // Esto indicaria que se ha seleccionado un elemento diferente de "Casa"
        if (chgTipoVivienda.getSelectedIndex() != 0) {
            // Se obtiene el elemento seleccionado para mostrarlo en la etiqueta
            String tipoVivienda = chgTipoVivienda.getString(
                chgTipoVivienda.getSelectedIndex());

            // Se cambia el valor de la etiqueta
            txfSubNumero.setLabel(tipoVivienda + " numero");

            // Se verifica el número de elementos del formulario.
            // Si son 4 o menos se agrega txfSubNumero, en caso contrario
            // no se agrega, pues indicaria que ya se ha agregado
            if(frmVivienda.size() <= 4){
                frmVivienda.append(txfSubNumero);
            }
        } else {
            // El indice es igual a 0, esto indica
            // que se ha seleccionado "Casa"
            // Se verifica si los elementos son mas que 4
            // Si asi fuera, se elimina el control txfSubNumero
            if(frmVivienda.size() > 4){
                frmVivienda.delete(4);
            }
        }
    }
}
}
}
}
}

```

2.2 Manejo de eventos

2.2.1. Configuración de escuchadores: `ItemCommandListener`, `ItemStateListener`,

Eventos del teclado

Adicionalmente a los eventos del teclado producidos por objetos `Command` agregados a ventanas, también se pueden agregar objetos `Command` a objetos de tipo `Item`. Adicionalmente, si los controles son de tipo `StringItem` o `ImageItem`, se producirá un efecto similar a tener botones o hyperlinks dentro del formulario.

La forma de implementar el manejo de los eventos de cambio de estado es el siguiente:

- ✓ Crear una clase que implemente la interfaz `ItemCommandListener` y sobrescribir el método `commandAction`. Este método es diferente del método del mismo nombre correspondiente a la interfaz `CommandListener`.

```
public class MiClase extends MIDlet implements
ItemCommandListener
```

- ✓ Crear un formulario y controles tipo `Item`. Si los controles son `StringItem` o `ImageItem`, asignar el valor de la constante `Item.BUTTON` o `Item.HYPERLINK` a su propiedad "Apariencia". Asimismo, la etiqueta debe ser la cadena vacía.

```
private Form frm = new Form("Prueba");
private TextField txf = new TextField("Prueba", "",
                                     10, TextField.ANY);
private StringItem sti = new StringItem("", "Aceptar",
                                     Item.BUTTON);
```

- ✓ Crear tantos objetos `Command` como botones se deseen utilizar. Estos objetos deben ser de tipo `Command.ITEM`.

```
private Command cmdAceptar = new Command("Aceptar",
                                          Command.ITEM, 2);
private Command cmdNuevo = new Command("Nuevo",
                                         Command.ITEM, 2);
```

- ✓ Agregarle al objeto `Item` los objetos `Command` correspondientes usando el método `addCommand()`

```
txf.addCommand(cmdNuevo);
sti.addCommand(cmdAceptar);
```

- ✓ Opcionalmente, indicar si el botón agregado se activará con el botón `SELECT`, usando el método `setDefaultCommand()`

```
sti.setDefaultCommand(cmdAceptar);
```

- ✓ Indicarle al objeto `Item` la referencia al objeto que implementa la interfaz `ItemCommandListener` usando el método `setItemCommandListener`

```
txf.setItemCommandListener(this);
sti.setItemCommandListener(this);
```

- ✓ Agregarle los controles `Item` al formulario:

```
frm.append(txf);
frm.append(sti);
```

- ✓ Poner la lógica correspondiente en el método `commandAction`:

```
public void commandAction(Command c, Item i) {
    if (c == cmdAceptar && i == sti) {
        /* Poner la logica que se desea realizar*/
    } else if (c == cmdNuevo && i == txf) {
        /* Poner la logica que se desea realizar*/
    }
}
```

Eventos de cambio de estado

Los eventos de cambio de estado se producen cuando un control de tipo `Item` modifica su valor. Por ejemplo, cuando un usuario ingresa un dato en un control `TextField`, se generará el evento cada vez que se escriba un caracter o, cuando se modifica el valor seleccionado en un control `ChoiceGroup`, se generará el evento con cada cambio en la selección.

La forma de implementar el manejo de los eventos de cambio de estado es el siguiente:

- ✓ Crear una clase que implemente la interfaz `ItemStateListener` y sobrescribir el método `itemStateChanged`

```
public class MiClase extends MIDlet implements
ItemStateListener
```

- ✓ Crear un formulario y controles tipo `Item`

```
private Form frm = new Form("Prueba");
private TextField txf = new TextField("Prueba", "",
                                     10, TextField.ANY);
```

- ✓ Agregarle los controles `Item` que cambiarán de estado:

```
frm.append(txf);
```

- ✓ Indicarle al objeto `Form` el objeto que procesará los eventos usando el método `setItemStateListener`

```
frm.setItemStateListener(this);
```


- ✓ Poner la lógica correspondiente en el método `itemStateChanged`

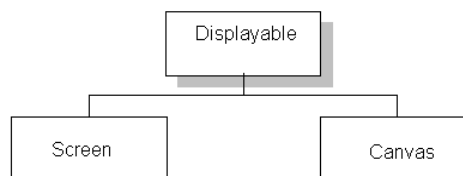
```
public void itemStateChanged(Item i) {  
    if (i == txf) {  
        /* Poner la logica que se desea realizar*/  
    }  
}
```

2.3 Graficador

2.3.1. Introducción a la interface gráfica

Cuando se diseñó JME, se tuvo en cuenta que una de las claves para que esta tecnología tuviera éxito era que tenía que ser capaz de hacer funcionar juegos, y hacerlo de forma adecuada. Por ello, los MIDlets pueden controlar la pantalla al más bajo nivel posible.

Hasta ahora, hemos trabajado con diversas clases que nos permiten definir la interface de usuario de manera estándar. Todas estas clases derivaban de la clase `Screen`, que a su vez derivaba de `Displayable`. En el diagrama de clases mostrado a continuación podemos observar que de `Displayable` también se deriva la clase `Canvas`. Esta clase es capaz de mostrar información gráfica a nivel de píxel. Es por ellos que la llamamos interfaz de bajo nivel.



2.3.2. La clase Canvas

Básicamente podemos realizar tres operaciones con una clase de tipo `Canvas`:

- ✓ Dibujar gráficos
- ✓ Escribir texto
- ✓ Dibujar imágenes

Cuando trabajamos con gráficos en JME, en vez de utilizar un objeto de la clase `Form` como pantalla principal de la aplicación, utilizamos uno derivado de la clase `Canvas`. Esto es posible ya que al igual que `Form`, `Canvas` es una clase hija de `Displayable`. Asimismo, la clase `Canvas` implementa el método `paint()`. Éste es el método que se invoca cada vez que la pantalla necesita ser redibujada. Por lo tanto, todo el código encargado de pintar en la pantalla lo ubicaremos normalmente aquí.

2.3.3. La API de juegos

La API de MIDP contempla las siguientes clases:

- ✓ GameCanvas
- ✓ Layer
- ✓ LayerManager
- ✓ TiledLayer
- ✓ Sprite

2.3.3.1 GameCanvas.- Clase abstracta que hereda de Canvas con un buffer asíncrono que permite recoger el estado de las teclas del dispositivo móvil.

Método	Sintaxis	Definición
flushGraphics()	<pre>public void flushGraphics() public void flushGraphics(int x, int y, int width, int height)</pre>	Coloca el contenido del buffer a la pantalla sin modificar el contenido de éste. Si los parámetros no han sido definidos, el tamaño del área volcada es el mismo que el del GameCanvas. En caso contrario se vuelca solo el espacio comprendido en la región especificada.
getGraphics()	protected Graphic getGraphics()	Devuelve el objeto Graphics que representa la imagen almacenada en el off-screen buffer y sobre el cual se puede utilizar todos los métodos que están disponibles en dicha clase aún en segundo plano para luego ser expuesto. Si se tratase de un objeto Graphics perteneciente a un objeto Canvas las acciones que se realizaran se mostrarían inmediatamente en la pantalla del dispositivo.
getKeyStates()	public int getKeyStates()	<p>Devuelve un entero en el que se codifica qué teclas están pulsadas y cuáles no. Cada bit del entero devuelto representa una tecla y se codificará de la siguiente manera:</p> <p>1 - Si la tecla está siendo pulsada o lo ha sido alguna vez desde que se hizo la última llamada a este método.</p> <p>0 - Si la tecla no está siendo pulsada y no se ha pulsado ninguna vez desde la última llamada a este método.</p> <p>Este comportamiento de "cerrojo" asegura que cualquier pulsación, por rápida que sea, será capturada independientemente del intervalo de tiempo que utilicemos para verificar periódicamente si ha habido alguna pulsación.</p>
paint()	public void paint(Graphics g)	Pinta el objeto Graphics del GameCanvas con el contenido del buffer. Para que sea visible se debe invocar al método flushGraphics().

2.3.3.2 Layer.- Clase abstracta que representa un elemento visual del juego con propiedades como posición, tamaño y visibilidad.

Método	Sintaxis	Definición
getHeight()	<code>public final int getHeight()</code>	Devuelve la altura del objeto Layer en pixels.
getWidth()	<code>public final int getWidth()</code>	Devuelve la anchura del objeto Layer en pixels.
getX()	<code>public final int getX()</code>	Devuelve la coordenada horizontal de la esquina superior izquierda del objeto.
getY()	<code>public final int getY()</code>	Devuelve la coordenada vertical de la esquina superior izquierda del objeto.
isVisible()	<code>public final boolean isVisible()</code>	Indica si el objeto es visible (true) o no (false).
move (int x, int y)	<code>public void move (int x, int y)</code>	Mueve el objeto Layer horizontalmente una distancia x verticalmente una distancia y. Las coordenadas del objeto están sujetas a ajuste si las distancias especificadas hacen que se sobrepasen los valores <code>Integer.MIN_VALUE</code> o <code>Integer.MAX_VALUE</code>
setPosition (int x, int y)	<code>public void setPosition(int x, int y)</code>	Mueve el Layer de tal manera que su esquina superior izquierda se sitúe en el punto (x,y). Si no invocamos este método por defecto estará situado en (0,0).
paint()	<code>public void paint(Graphics g)</code>	Pinta el objeto si está en modo visible, no haciendo nada en caso contrario. Todas las subclases de Layer deben implementar este método. Dichas implementaciones son las responsables de comprobar si el objeto Layer está o no visible. Los atributos del objeto Graphics no se ven alterados como resultado de invocar este método.
setVisible(boolean visible)	<code>public void setVisible(boolean visible)</code>	Indica si el objeto Layer es visible o no. Es importante principalmente a la hora de invocar el método <code>paint(Graphics g)</code> ya que como se acaba de ver este método sólo se ejecutará cuando el Layer sea visible.

2.3.3.3 LayerManager.- La clase LayerManager permite controlar y gestionar una serie de objetos Layer que formen parte de la misma aplicación. Para ello mantiene una lista ordenada en la que los Layers pueden ser insertados, accedidos o eliminados. LayerManager es una superestructura que almacena una serie de objetos Layer y tiene acceso a todos los datos de cada uno de ellos: posición, profundidad (es decir, si se superpone a otro Layer o al revés), estado, etc.

VENTANA VISIBLE (View Window)

Permite controlar el tamaño y la posición de la región relativa al sistema de coordenadas del objeto LayerManager. Es muy útil cuando una capa contiene una imagen mayor que el tamaño de la pantalla del dispositivo, como por ejemplo un fondo. De esta manera se genera los efectos de barrido (scrolling) y panorámica (panning).

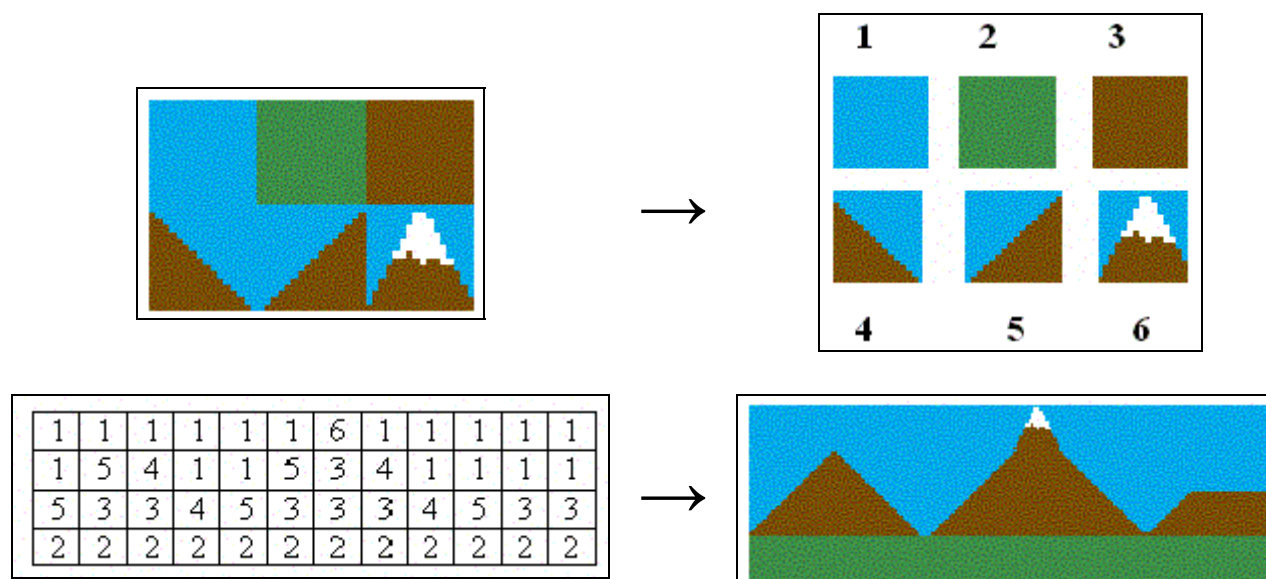
Método	Sintaxis	Definición
append(Layer l)	public void append(Layer l)	Añade el Layer especificado a un LayerManager. Le asigna el mayor índice, el que corresponde a las capas más profundas. Si el LayerManager ya contenía el mismo Layer éste será borrado antes de almacenarlo.
getLayerAt(int index)	public Layer getLayerAt(int index)	Devuelve el objeto Layer almacenado en la posición especificada.
getSize()	public int getSize()	Devuelve el número de Layers almacenados en un LayerManager.
insert(Layer l, int index)	public void insert(Layer l, int index)	Inserta el Layer especificado en un LayerManager en la posición indicada por index. Si ya había sido añadido con anterioridad será borrado antes de insertarlo.
paint(Graphics g, int x, int y)	public void paint (Graphics g, int x, int y)	Dibuja en la pantalla la View Windows visible del LayerManager en la posición indicada con las coordenadas x e y. Estas coordenadas son relativas a la posición del objeto Graphics, por lo que al desplazarlo supone que la ventana visible se desplace también (pero no su contenido). Las imágenes que estén totalmente fuera de la ventana visible o que pertenezcan a un Layer en estado invisible no serán pintadas.
remove(Layer l)	public void remove(Layer l)	Elimina del LayerManager el Layer especificado. Si dicho Layer no se encuentra este método no hace nada.
setViewWindow(int x, int y, int width, int height)	public void setViewWindow(int x, int y, int width, int height)	Indica la View Windows.

2.3.3.4 **TiledLayer**.- Clase que hereda de Layer y que representa un elemento visual compuesto por un conjunto de celdas, cada una de las cuales tiene asociada una imagen que denominaremos baldosa (tile). Tener un objeto TiledLayer equivale a tener una serie de piezas de un rompecabezas reordenable para obtener una imagen con el aspecto deseado. Esto simplifica el uso de imágenes de gran tamaño pues se podrá definirla con una serie de celdas combinadas y repetidas convenientemente.

Celdas y Baldosas¹

El método constructor de la clase TiledLayer es `TiledLayer(int m, int n, Image i, int tileWidth, int tileHeight)`. Al invocarlo se divide la imagen indicada en baldosas de un tamaño indicado por `tileWidth` y `tileHeight`. Al mismo tiempo se genera un conjunto de celdas que conforman una matriz con `m` filas y `n` columnas. El siguiente código muestra cómo sería el proceso:

```
Image image = Image.createImage("/board.png");
TiledLayer tiledLayer = new TiledLayer(10, 10, image, 16, 16);
```



¹ Fuente.- http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/midp2_0/PracticaGame/

- Baldosas estáticas: son aquellas que tienen asociada una imagen fija que no cambiará durante toda la ejecución.
- Baldosas dinámicas o animadas: son aquellas que están asociadas a una imagen que puede variar durante el juego. Es muy útil para variar el aspecto de una porción importante de la pantalla sin necesidad de ir cambiando las celdas una por una. Tan solo hay que declarar varias baldosas dinámicas y variar cuando convenga la imagen a la que hacen referencia.

IMPORTANTE: los identificadores de baldosas dinámicas son siempre enteros NEGATIVOS.

Método	Sintaxis	Definición
<code>createAnimatedTile(int staticTileIndex)</code>	<code>public void createAnimatedTile (int staticTileIndex)</code>	Crea una baldosa dinámica nueva a partir de una estática. El método devuelve el índice que identifica a la nueva baldosa.
<code>fillCells(int col, int row, int numCols, int numRows, int tileIndex)</code>	<code>public void fillCells (int col, int row, int numCols, int numRows, int tileIndex)</code>	Rellena una región de la matriz de celdas con la baldosa especificada. Puede ser una baldosa animada, estática o transparente (identificador 0).
<code>setAnimatedTile(int animatedTileIndex, int staticTileIndex)</code>	<code>public void setAnimatedTile (int animatedTileIndex, int staticTileIndex)</code>	Asocia a una baldosa animada una determinada baldosa estática la cual puede ser cambiada a lo largo de la ejecución del programa.
<code>getAnimatedTile(int animatedTileIndex)</code>	<code>public int getAnimatedTile (int animatedTileIndex)</code>	Devuelve la baldosa a la que hace referencia una determinada baldosa dinámica.
<code>setCell(int col, int row, int tileIndex)</code>	<code>public void setCell(int col, int row, int tileIndex)</code>	Fija el contenido de una celda asignándole una baldosa que puede ser de cualquier tipo.
<code>getCell(int col, int row)</code>	<code>public int getCell(int col, int row)</code>	Devuelve el índice de la baldosa (sea del tipo que sea) que se encuentra en la celda especificada.
<code>getCellHeight()</code>	<code>public int getCellHeight()</code>	Devuelve la altura de las celdas, en general.
<code>getCellWidth()</code>	<code>public int getCellWidth()</code>	Devuelve la anchura de las celdas, en general.
<code>getColumns()</code>	<code>public int getColumns()</code>	Devuelve el número de columnas de la matriz de celdas del objeto TiledLayer.
<code>getRows()</code>	<code>public int getRows()</code>	Devuelve el número de filas de la matriz de celdas del objeto TiledLayer.
<code>setStaticTileSet(Image i, int tileWidth, int tileHeight)</code>	<code>public void setStaticTileSet (Image I, int tileWidth, int tileHeight)</code>	Cambia la imagen fuente sobre la que se creó el objeto TiledLayer. Si el número de baldosas que se generan con la nueva imagen es el mismo que ya existía, los identificadores serán los mismos por lo que el contenido de la matriz

		de celdas se conservará. Si no es así, todas las celdas perderán su contenido que será puesto a cero tal como si se creara un nuevo objeto TiledLayer.
<code>paint(Graphics g)</code>	<code>public void paint(Graphics g)</code>	Dibuja en pantalla el contenido del objeto TiledLayer. Es equivalente al método <code>paint(...)</code> de la clase Layer puesto que TiledLayer es una subclase.

2.3.3.5 Sprite.- Sprite es otra subclase de Layer y se asume como equivalente conceptual de TiledLayer. Sin embargo, la diferencia radica en que en el TiledLayer se generan baldosas a partir una única imagen, los sprites generan una única imagen animada a partir de varias imágenes (frames).

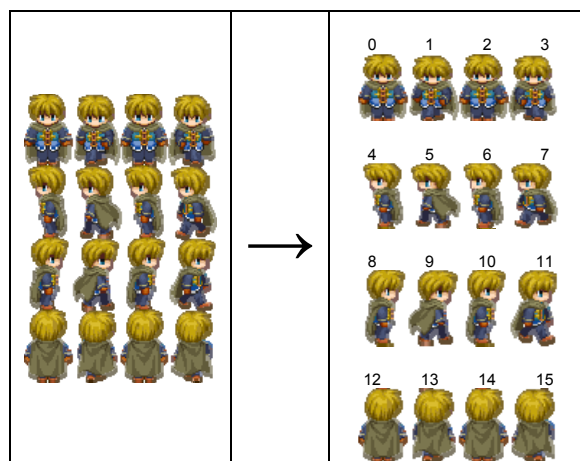
Frames

Para generar el objeto Sprite que proporcione la animación deseada utilizaremos la siguiente expresión:

`Sprite sprite = new Sprite(imagen, frameWidth, frameHeight);` donde:

`imagen`: objeto Image creado a partir de un archivo fuente que contine los frames para crear la animación.

`frameWidth, frameHeight`: dimensiones de cada frame.



A partir de estos frames es posibles crear secuencias de movimientos como `derecha({8,9,10,11})` o `izquierda({4,5,6,7})`.

Método	Sintaxis	Definición
<code>collidesWith(Image image, int x, int y, boolean pixelLevel)</code>	<code>public final boolean collidesWith (Image image, int x, int y, boolean pixelLevel)</code>	Comprueba si existe colisión entre el objeto Sprite y la imagen especificada cuya posición es indicada mediante los enteros x e y. El argumento <code>pixelLevel</code> indica si se debe utilizar o no detección por niveles de píxel.
<code>collidesWith(Sprite sprite, boolean pixelLevel)</code>	<code>public final boolean collidesWith(Sprite sprite, boolean pixelLevel)</code>	Método similar al anterior solo que en este caso la colisión se detecta entre dos objetos Sprite. El funcionamiento de este método es idéntico al anterior.
<code>collidesWith(TiledLayer tiledLayer, boolean pixelLevel)</code>	<code>public final boolean collidesWith(TiledLayer tiledLayer, boolean pixelLevel)</code>	Método idéntico a los dos anteriores, en este caso aplicado a la colisión con un objeto <code>TiledLayer</code> .
<code>defineCollisionRectangle(int x, int y, int width, int height)</code>	<code>public void defineCollisionRectangle(int x, int y, int width, int height)</code>	Método que permite definir el tamaño y la posición del área de colisión (o rectángulo de colisión) del Sprite. A la hora de comprobar si existe o no colisión con algún otro elemento visual sólo se realizará la comprobación sobre el área definida con este método. Por defecto, está definido respecto a la posición (0,0) (esquina superior izquierda) y con un tamaño igual al del objeto.
<code>defineReferencePixel(int x, int y)</code>	<code>public void defineReferencePixel(int x, int y)</code>	Método que permite definir cuál será el píxel de referencia del objeto <code>Sprite</code> . Para hacer referencia a un píxel concreto lo haremos a partir de su posición relativa respecto a la esquina superior izquierda. Por defecto el píxel de referencia será el (0,0). El hecho de cambiarlo no hará que el Sprite cambie de posición en la pantalla.
<code>getFrame()</code>	<code>public final int getFrame()</code>	Devuelve el índice del frame mostrado en ese instante.
<code>getFrameSequenceLength()</code>	<code>public int getFrameSequenceLength()</code>	Devuelve el número de frames que forman la secuencia de imágenes.
<code>nextFrame()</code>	<code>public void nextFrame()</code>	Selecciona el frame siguiente en la secuencia circular del objeto <code>Sprite</code> .
<code>prevFrame()</code>	<code>public void prevFrame()</code>	Selecciona el frame anterior en la secuencia circular del objeto <code>Sprite</code> .
<code>setFrame()</code>	<code>public void setFrame(int sequenceIndex)</code>	Selecciona un frame determinado que será indicado por el índice que le fue asignado al ser creado. Esta acción no será visible hasta que no se invoque el método <code>paint(Graphics g)</code> .

<code>setFrameSequence(int[] sequence)</code>	<code>public void setFrameSequence(int[] sequence)</code>	Indica cuál es la secuencia de frames a reproducir. Todos los objetos Sprite son creados con una secuencia que por defecto reproduce los frames en orden creciente. Si al invocar este método lo hacemos pasando un argumento null se volverá a esta secuencia por defecto.
<code>setImage(Image image, int frameWidth, int frameHeight)</code>	<code>public void setImage(Image image, int frameWidth, int frameHeight)</code>	Cambia la imagen fuente asociada al objeto Sprite. Se especifica el tamaño de los nuevos frames que se deben crear con lo que el número de éstos que se generen puede variar respecto al que ya existía.

2.3.2.1. Ejercicio 1: Juego básico en Java

Se muestra a continuación la interface y el código necesario para implementar un juego básico de “Naves estelares” en Java.



```
public class Juego extends GameCanvas implements Runnable {
    final int WIDTH_CANVAS = getWidth();
    final int HEIGHT_CANVAS = getHeight();
    final int WIDTH_GAME = 12*32;
    final int HEIGHT_GAME = HEIGHT_CANVAS;
    final int WIDTH_VIEW = WIDTH_GAME/2;
    final int HEIGHT_VIEW = HEIGHT_GAME;

    int xPosEscenario = (WIDTH_GAME/2)-(WIDTH_CANVAS/2);
    int yPosEscenario = 0;
    int xPosNaveNodriz = (WIDTH_GAME/2) - (132/2);
    int yPosNaveNodriz = 0;
    int xPosNave = (WIDTH_GAME/2) - (35/2);
    int yPosNave = HEIGHT_CANVAS - 30;

    Image imgEscenario = null;
    Image imgNave = null;
    Image imgFoton = null;

    LayerManager lmgJuego;
    TiledLayer tlyEscenario;
    Sprite sprNave;
    Sprite sprFoton;
    Vector foton = new Vector();
    Vector piezasNaveNodriz = new Vector();

    Graphics g;
```

```

public Juego(boolean b) {

    super(b);
    g = getGraphics();
    lmgJuego = new LayerManager();

    try {
        imgEscenario = Image.createImage("/infinito.png");
    } catch (java.io.IOException e) {}

    tlyEscenario = new TiledLayer(12,20,imgEscenario,32,18);
    int[] mapEscenario = {
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,2,3,4,3,4,3,4,3,4,3,4,
        1,2,3,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,4,3,4,3,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,2,3,4,3,4,3,4,
        1,2,1,2,1,2,1,4,1,2,1,2,
        3,4,3,4,3,4,3,4,1,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,4,3,4,3,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,4,
        3,4,3,4,3,4,3,4,3,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,4,3,4,3,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,4,3,4,3,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,4,3,4,3,4,3,4,
        1,2,1,2,1,2,1,2,1,2,1,2,
        3,4,3,4,3,4,3,4,3,4,3,4
    };

    for (int i = 0; i < mapEscenario.length; i++) {
        int column = i % 12;
        int row = (i - column) / 12;
        tlyEscenario.setCell(column, row, mapEscenario[i]);
    }

    tlyEscenario.setVisible(true);

    for (int i = 0; i <= 76; i++) {
        int column = i % 11;
        int row = (i - column) / 11;
        Image imgPieza = null;

        try {
            String ruta = "/nodriza/platillo_" + (i+1) + ".png";
            imgPieza = Image.createImage(ruta);
        } catch (IOException e) { e.printStackTrace(); }

        Sprite sprParte = new Sprite(imgPieza);
        sprParte.setPosition(column*12 + xPosNaveNodriza, row * 6 + yPosNaveNodriza);
        piezasNaveNodriza.addElement(new PiezaNaveNodriza(sprParte));
        sprParte.setVisible(true);
        lmgJuego.insert(sprParte,0);
    }

    try {
        imgNave = Image.createImage("/nave.png");
    } catch (java.io.IOException e) {}

    sprNave = new Sprite(imgNave,35,30);
    int[] sequence = {0,1};
    sprNave.setFrameSequence(sequence);
    sprNave.setPosition(xPosNave,yPosNave);
    sprNave.setVisible(true);

    try {
        imgFoton = Image.createImage("/foton.png");
    } catch (java.io.IOException e) {}
}

```

```

        sprFoton = new Sprite(imgFoton,12,12);
        sprFoton.setFrameSequence(sequence);
        sprFoton.setVisible(false);
        lmgJuego.append(sprNave);
        lmgJuego.append(tlyEscenario);
        lmgJuego.setViewWindow(xPosEscenario, yPosEscenario, WIDTH_VIEW, HEIGHT_VIEW);
        lmgJuego.paint(g,0,0);
        flushGraphics();
    }

    public void run() {
        while (true) {
            try {
                for(int iFoton = 0; iFoton < fotones.size(); iFoton++) {
                    Sprite foton = (Sprite)fotones.elementAt(iFoton);
                    if(foton.isVisible()){
                        foton.setPosition(foton.getX(),foton.getY()-4);
                        foton.nextFrame();

                        if(foton.getY() < -12 ){
                            lmgJuego.remove(foton);
                            fotones.removeElement(foton);
                        }

                        for (int iPieza = 0; iPieza < piezasNaveNodrizas.size(); iPieza++) {
                            PiezaNaveNodrizas pieza = (PiezaNaveNodrizas)
                                piezasNaveNodrizas.elementAt(iPieza);
                            Sprite sprParte = (Sprite)pieza.getPieza();

                            if(pieza.isColisionada()==false &&
                                foton.collidesWith(sprParte, true)){
                                System.out.println("Colisión entre foton: " + iFoton +
                                    " con pieza:"+iPieza);
                                pieza.setColisionada(true);
                                lmgJuego.remove(foton);
                                lmgJuego.remove(sprParte);
                                fotones.removeElement(foton);
                                break;
                            }
                        }
                    }
                }

                int keyState = this.getKeyStates();
                if (keyState!=0) {
                    actualizaPosicion(keyState);
                    lmgJuego.setViewWindow( xPosEscenario, yPosEscenario,
                                            WIDTH_VIEW, HEIGHT_VIEW);
                    sprNave.setRefPixelPosition(xPosNave,yPosNave);
                }

                lmgJuego.paint(g,0,0);
                flushGraphics();
                Thread.sleep(50);
            } catch (Exception e){ System.out.println(e); }
        }
    }

    public void actualizaPosicion(int key){
        if ((key & LEFT_PRESSED) != 0){
            if(xPosEscenario > 0){
                xPosEscenario -= 4;
                sprNave.prevFrame();
                xPosNave -= 6;
            }
        }

        if ((key & RIGHT_PRESSED) != 0){
            if(xPosEscenario < WIDTH_GAME - WIDTH_VIEW ){
                xPosEscenario += 4;
                sprNave.nextFrame();
                xPosNave += 6;
            }
        }
    }

```

```

        if (((key & 256) != 0) || ((key & UP_PRESSED) != 0)){
            Sprite nuevo = new Sprite(sprFoton);
            nuevo.setPosition(xPosNave + sprNave.getWidth()/2 - sprFoton.getWidth()/2,
                            yPosNave);
            fotones.addElement(nuevo);
            nuevo.setVisible(true);
            lmgJuego.insert(nuevo,0);
        }
    }
}

```

```

class PiezaNaveNodriz{
    private Sprite pieza;
    private boolean colisionada;

    public Sprite getPieza() {
        return pieza;
    }
    public void setPieza(Sprite pieza) {
        this.pieza = pieza;
    }
    public boolean isColisionada() {
        return colisionada;
    }
    public void setColisionada(boolean colisionada) {
        this.colisionada = colisionada;
    }
    public PiezaNaveNodriz(Sprite pieza) {
        this.pieza = pieza;
    }
}

```

```

public class Principal extends MIDlet{
    Display display;
    Juego juego;

    public void startApp() {
        display = Display.getDisplay(this);
        juego = new Juego(true);
        display.setCurrent(juego);
        Thread t = new Thread(juego);
        t.start();
    }

    // Implementa el metodo pauseApp()
    public void pauseApp() { }

    // Implementa el metodo commandApp()
    public void destroyApp(boolean unconditional) { }
}

```

2.3.2.2. Ejercicio 2: Juego avanzando en Java

Se muestra a continuación la interface y el código necesario para implementar un juego avanzado con diversos “escenarios” en Java.




```

private void cargarSprites(){
    objetosColision = new Vector();
    objetoColision = null;

    try {
        objetoColision = Image.createImage("/imagenes/colisiones/cc1.PNG");
    } catch (IOException e1) {
        e1.printStackTrace();
    }

    Sprite sprObjetoColision = new Sprite(objetoColision);
    for(int i = 0; i < mapEsc.length; i++){
        int columna = i % 24;
        int fila = (i - columna) / 24;
        tlyEscenario.setCell(columna, fila, mapEsc[i]);
        if(mapEsc[i]==6){
            objetosColision.addElement(new ObjetoColision(columna,
                                                            fila,
                                                            columna*24 + xPosEscenario,
                                                            fila*24 + yPosEscenario,
                                                            new Sprite(sprObjetoColision),objetoColision));
        }
    }
}

public void start(){
    enJuego = true;
    Thread t = new Thread(this);
    t.start();
}

public void run() {
    Graphics g = getGraphics();
    while (enJuego){
        leeTeclado();
        dibujaPantalla(g);
        try {
            Thread.sleep(retraso);
        } catch (Exception e) { }
    }
}

protected void dibujaPantalla(Graphics g) {
    g.setColor(200, 200, 200);
    g.fillRect(0, 0, getWidth(), getHeight());

    tlyEscenario.setVisible(true);
    tlyEscenario.setPosition(xPosEscenario, yPosEscenario);
    sprPersonaje.setPosition(xPosPersonaje, yPosPersonaje);
    sprPersonaje.setFrame(framesPersonaje);
    tlyEscenario.paint(g);

    for (int i = 0; i < objetosColision.size(); i++) {
        ObjetoColision obj = (ObjetoColision)
            objetosColision.elementAt(i);
        g.drawImage(obj.getObjImage(), obj.getxPosObjeto(),
            obj.getyPosObjeto(), Graphics.TOP|Graphics.LEFT);
    }

    //Pinta el sprite
    sprPersonaje.paint(g);
    //Pinta la pantalla
    flushGraphics();
}

```

```

private boolean colision(Sprite personaje){

    for (int i = 0; i < objetosColision.size(); i++) {
        ObjetoColision obj = (ObjetoColision) objetosColision.elementAt(i);
        if(personaje.collidesWith(obj.getObjImage(),
                                obj.getyPosObjeto(),
                                obj.getxPosObjeto(),
                                true)){
            System.out.println("Colision X=" + obj.getxPosObjeto() +
                               "; Y=" + obj.getyPosObjeto());
            return true;
        }
    }

    return false;
}

private void leeTeclado() {
    int keyStates = getKeyStates();

    if ((keyStates & RIGHT_PRESSED) != 0){
        if (frameActualPersonaje<9 || frameActualPersonaje>11){
            frameActualPersonaje = 9;
        }

        if(enMovimiento){
            //Nada que hacer
        } else {
            if(xPosEscenario == -335){
                if ( colision(sprPersonaje) ){
                    System.out.println("Colision!!!!");
                } else {
                    xPosPersonaje = Math.min(anchoCanvas-50, xPosPersonaje + 5);
                }
            } else {
                xPosEscenario -= 5;
                actualizarObjetosColision(-5,0);
            }
        }

        framesPersonaje = frameActualPersonaje;
        frameActualPersonaje++;
    }

    if ((keyStates & LEFT_PRESSED) != 0){
        if (frameActualPersonaje<5 || frameActualPersonaje>7){
            frameActualPersonaje = 5;
        }

        if (enMovimiento){
            //Nada que hacer
        } else {
            if (xPosEscenario == 0){
                if ( colision(sprPersonaje) ){
                    System.out.println("Colision!!!!");
                } else {
                    xPosPersonaje=Math.max(20,xPosPersonaje-5);
                }
            } else {
                xPosEscenario += 5;
                actualizarObjetosColision(5,0);
            }
        }

        framesPersonaje = frameActualPersonaje;
        frameActualPersonaje++;
    }
}

```

```

        if ((keyStates & UP_PRESSED) != 0){
            if (frameActualPersonaje<13 || frameActualPersonaje>15){
                frameActualPersonaje = 13;
            }

            if (enMovimiento){
                //Nada que hacer
            } else {
                if (yPosEscenario == 0){
                    if ( colision(sprPersonaje) ){
                        System.out.println("Colision!!!!");
                    } else {
                        yPosPersonaje = Math.max(10,yPosPersonaje-5);
                    }
                } else {
                    yPosEscenario += 5;
                    actualizarObjetosColision(0,5);
                }
            }

            framesPersonaje = frameActualPersonaje;
            frameActualPersonaje++;
        }

        if ((keyStates & DOWN_PRESSED) != 0){
            if (frameActualPersonaje < 1 || frameActualPersonaje >3){
                frameActualPersonaje = 1;
            }

            if (enMovimiento){
                //Nada que hacer
            } else {
                if (yPosEscenario == -225){
                    if ( colision(sprPersonaje) ){
                        System.out.println("Colision!!!!");
                    } else {
                        yPosPersonaje = Math.min(altoCanvas-80, yPosPersonaje + 5);
                    }
                } else{
                    yPosEscenario -= 5;
                    actualizarObjetosColision(0,-5);
                }
            }

            framesPersonaje = frameActualPersonaje;
            frameActualPersonaje++;
        }
    }

    void actualizarObjetosColision(int x, int y){
        for (int i = 0; i < objetosColision.size(); i++) {
            ObjetoColision obj = (ObjetoColision) objetosColision.elementAt(i);
            obj.setxPosObjeto(obj.getxPosObjeto() + x);
            obj.setyPosObjeto(obj.getyPosObjeto() + y);
        }
    }
}

```

Clase Objeto Colisión

```

class ObjetoColision{
    private int xPosObjeto;
    private int yPosObjeto;
    private Image objImage;
    private Sprite objSprite;
    private int xColumnaObjeto;
    private int yFilaObjeto;

    public int getXColumnaObjeto() { return xColumnaObjeto; }

    public void setXColumnaObjeto(int xColumnaObjeto) {
        this.xColumnaObjeto = xColumnaObjeto;
    }

    public Sprite getObjSprite() {
        return objSprite;
    }

    public int getYFilaObjeto() {
        return yFilaObjeto;
    }

    public void setYFilaObjeto(int yFilaObjeto) {
        this.yFilaObjeto = yFilaObjeto;
    }

    public void setObjSprite(Sprite objSprite) {
        this.objSprite = objSprite;
    }

    public ObjetoColision(int columna, int fila, int x, int y,
        Sprite objeto, Image figura) {
        xColumnaObjeto = columna;
        yFilaObjeto = fila;
        xPosObjeto = x;
        yPosObjeto = y;
        objSprite = objeto;
        objImage = figura;
    }

    public ObjetoColision(int x, int y, Image i) {
        xPosObjeto = x;
        yPosObjeto = y;
        objImage = i;
    }

    public int getXPosObjeto() {
        return xPosObjeto;
    }

    public void setXPosObjeto(int xPosObjeto) {
        this.xPosObjeto = xPosObjeto;
    }

    public int getYPosObjeto() {
        return yPosObjeto;
    }

    public void setYPosObjeto(int yPosObjeto) {
        this.yPosObjeto = yPosObjeto;
    }

    public Image getObjImage() {
        return objImage;
    }

    public void setObjImage(Image objImage) {
        this.objImage = objImage;
    }
}

```

```
public class Principal extends MIDlet {
    Juego objJuego;
    Display pantalla;

    public void startApp() {
        try {
            pantalla = Display.getDisplay(this);
            objJuego = new Juego();

            pantalla.setCurrent(objJuego);
            objJuego.start();
            objJuego.run();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void pauseApp() {
    }

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

Resumen

- 📖 La clase *Alert* define una ventana que muestra un mensaje, y adicionalmente puede tener una imagen y un tiempo de visualización (*timeout*).
- 📖 La clase *TextBox* define una ventana que pide el ingreso de un dato, el cual puede ser de diferentes tipos.
- 📖 La clase *List* se usa para mostrar una lista de selección de datos. Dicha selección puede ser única o múltiple. También, se utiliza para la creación de menús.
- 📖 La clase *Image* se usa para mostrar una imagen. El formato estándar de los archivos imagen para móviles es PNG. Estos archivos deben ser copiados en la carpeta *res* del proyecto para poder ser usados.
- 📖 La clase *Ticker* se usa para mostrar un texto de una manera similar a una marquesina. Un ticker debe ser agregado a un objeto que herede de *Displayable* (*TextBox*, *Alert*, *List*, *Form*, *Canvas*) para poder ser visto.
- 📖 La clase *Form* se usa como contenedor de objetos de tipo *Item*. De este modo, permite la creación de ventanas para el ingreso de múltiples datos.
- 📖 La clase *StringItem* es una sub clase de *Item*. Permite crear controles similares a las etiquetas dentro de un formulario. También, pueden ser usadas para simular un botón o *hyperlink* dentro de un formulario.
- 📖 Las clases *TextField* y *ChoiceGroup* son subclases de *Item*. Son muy similares a *TextBox* y *List*, con la diferencia que están dentro de un formulario.
- 📖 Desde un formulario, se puede controlar los cambios de estado de los controles tipo *Item* utilizando la interfaz *ItemStateListener*.
- 📖 Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

🔗 <http://www.mailxmail.com/curso/informatica/j2me>

🔗 http://programacion.com/java/tutorial/ags_j2me/

UNIDAD DE
APRENDIZAJE**3**

JME: GESTIÓN DE ALMACENAMIENTO DE REGISTROS JME

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno elabora aplicaciones que permitan el ingreso, almacenamiento y visualización de datos, así como la manipulación de registros, aplicando nociones de buenas prácticas de programación, con el empleo de la clase `RecordStore` y sus clases relacionadas pertenecientes a la plataforma Java Micro Edition.

TEMARIO

3.1 Tema 5 : Gestión de Registros

- 3.1.1. : Introducción al sistema de almacenamiento de registros (RMS)
- 3.1.2. : Clase `RecordStore`: métodos `openRecordStore`,
`closeRecordStore`
- 3.1.3. : Manipulación de registros: métodos `addRecord`, `setRecord`,
`deleteRecord` y `getRecord`

ACTIVIDADES PROPUESTAS

- Los alumnos crean aplicaciones JME con persistencia utilizando `RecordStores`.

3.1 Gestión de Registros

3.1.1. Introducción al sistema de almacenamiento de registros (RMS)

JME proporciona un mecanismo a los MIDlets que les permite almacenar datos de forma persistente para su futura recuperación. Este mecanismo está implementado sobre una pequeña base de datos basada en registros que llamaremos Record Management System o RMS (Sistema de gestión de registros).

La información será guardada en el dispositivo en una zona de memoria dedicada para este propósito. La cantidad de memoria, la zona asignada y los mecanismos de integridad dependerán de cada dispositivo móvil.

El concepto que tiene Java Micro Edition para manejar los datos se muestra en la siguiente figura:

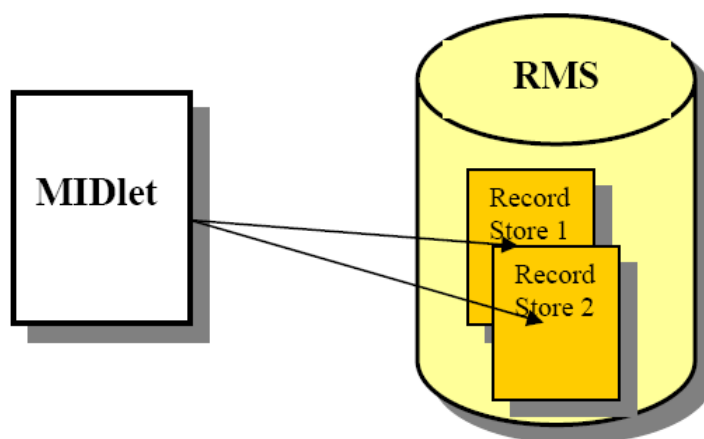


Figura 3.1 – Relación del Midlet con RMS (Parte A)

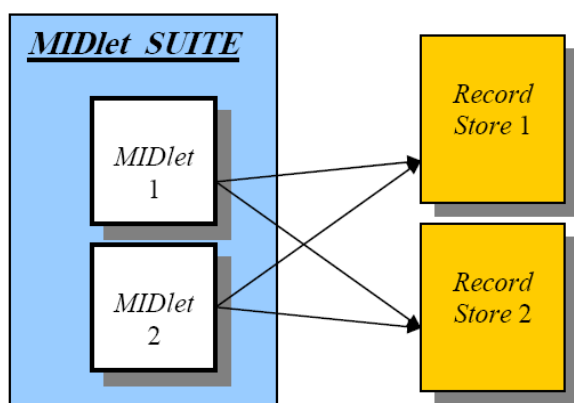


Figura 3.1 – Relación del Midlet con RMS (Parte B)

3.1.2. Clase RecordStore: métodos openRecordStore, closeRecordStore

La clase RecordStore permite administrar la creación, acceso y eliminación de los diferentes RecordStore. Para ello, solo necesitamos crear una sola variable, la cual podrá acceder a todos los registros almacenados en el RMS. Esta clase no cuenta con un método constructor, por lo que la creación de cada RecordStore recae sobre el método openRecordStore(). Los registros se almacenan en forma de arreglo de bytes

Dentro de sus principales métodos destacan:

openRecordStore()

-Definición

```
public static RecordStore openRecordStore(String nombre,  
boolean crear)  
public static RecordStore openRecordStore(String nombre,  
boolean crear, int modo, boolean editar)  
public static RecordStore openRecordStore(String nombre,  
String vendedor, String midletSuite)
```

-Descripción

Permite crear un RecordStore. Sus parámetros son:

- ✓ nombre: Nombre del RecordStore
- ✓ crear: Dato lógico que indica si el RecordStore será creado en caso no existiera
- ✓ editar: Dato lógico que indica si el RecordStore podrá ser modificado
- ✓ vendedor: Dato que indica el vendedor del MIDletSuite, al cual pertenece el RecordStore
- ✓ vendedor: Dato que indica el nombre del MIDletSuite, al cual pertenece el RecordStore

closeRecordStore()

-Definición

```
public void closeRecordStore()
```

Descripción

Cierra el RecordStore.

3.1.3. Manipulación de registros: métodos addRecord, setRecord, deleteRecord y getRecord

Para manipular datos los métodos más usados son los siguientes:

deleteRecordStore()

-Definición

```
public static void deleteRecordStore(String nombre)
```

-Descripción

Elimina el RecordStore especificado.

listRecordStores()

-Definición

```
public static String [] listRecordStores()
```

-Descripción

Devuelve un arreglo con los nombre de los RecordStores disponibles.

addRecord()

-Definición

```
public int addRecord(byte [] datos, int inicio, int longitud)
```

-Descripción

Permite agregar un nuevo registro al RecordStore abierto. Para ingresar un registro, este debe estar en forma de arreglo de bytes y se pasara la posición inicial desde donde se insertará el dato y la cantidad de bytes que ingresarán en el registro. El método devolverá el identificador del registro ingresado.

setRecord()

-Definición

```
public void setRecord(int id, byte [] datos, int inicio,  
int longitud)
```

-Descripción

Permite modificar el registro identificado con el parámetro "id" en el RecordStore abierto. Para esto, el registro debe ingresar en forma de arreglo de bytes y se pasará la posición inicial desde donde se insertará el dato y la cantidad de bytes que ingresarán en el registro.

deleteRecord()

-Definición

```
public void deleteRecord(int id)
```

-Descripción

Elimina el registro identificado con el valor parámetro "id".

getRecord()

-Definición

public byte [] getRecord(int id)

-Descripción

Devuelve el registro identificado con el valor parámetro "id".

getNumRecords()

-Definición

public int getNumRecords()

-Descripción

Devuelve la cantidad de registros que se encuentra almacenados en el RecordStore.

getSize()

-Definición

public int getSize()

-Descripción

Retorna el tamaño de memoria utilizado por el RecordStore.

3.1.3.1. Ejercicio:Aplicación Agenda

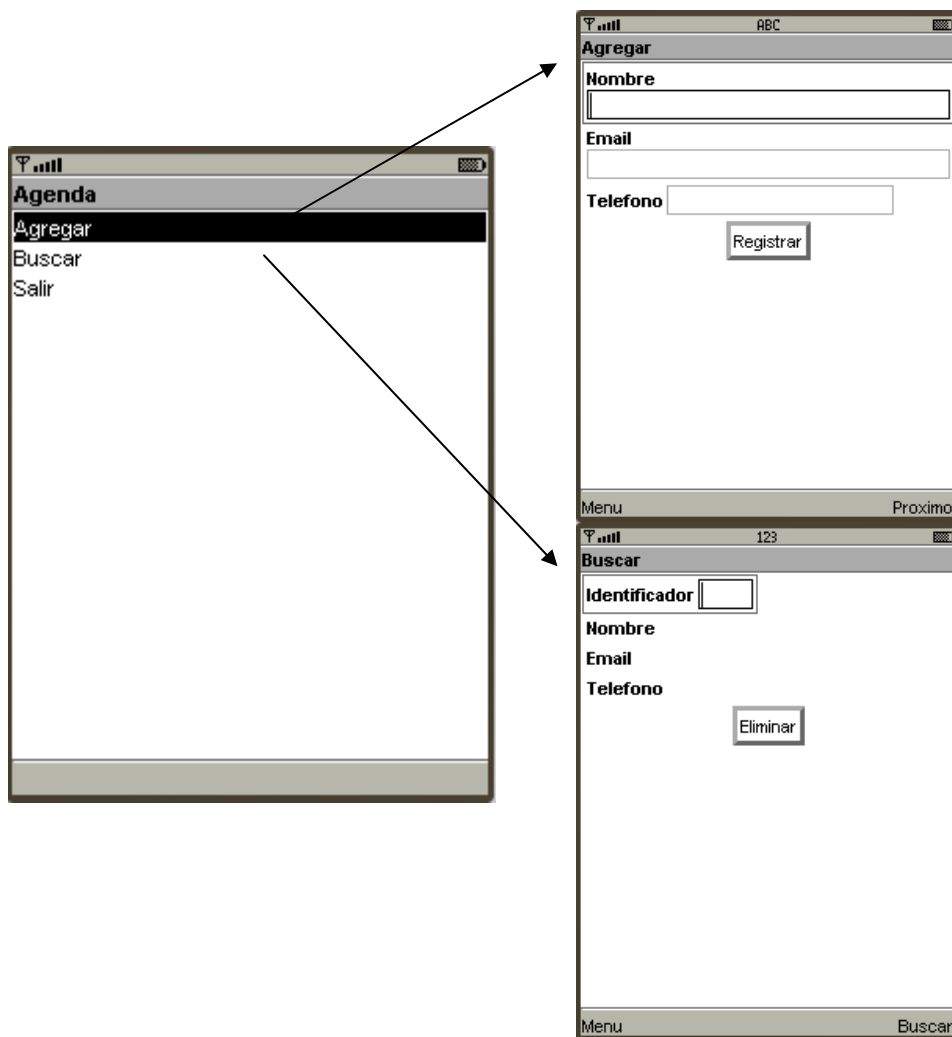
Construir una aplicación que realice lo siguiente:

Mostrar un menú con el título "Agenda" y las siguientes opciones:

- ✓ Agregar, que carga una ventana para agregar nuevos registros
- ✓ Buscar, que carga una ventana para buscar y eliminar registros por su identificador
 - La ventana "Agregar" debe permitir el ingreso de 3 datos:
 - Nombre (Alfanumérico, 50 caracteres)
 - Email (correo electrónico, 50 caracteres)
 - Teléfono (teléfono, 14 dígitos)
- ✓ Además debe tener 3 botones:
 - Registrar: Que grabará el registro en un RecordStore, mostrando en una ventana el resultado de la operación y el identificador del registro
 - Menú: Que regresa al menú inicial
 - Próximo: Que limpia el valor de los controles
 - La ventana "Buscar" debe permitir el ingreso de 1 dato:
- ✓ Identificador (Numérico, 3 dígitos)
- ✓ Además debe tener 3 botones:

- Eliminar: Que eliminará el registro en el RecordStore usando el identificador ingresado, mostrando en una ventana el resultado de la operación
- Menú: Que regresa al menú inicial
- Buscar: Que busca el registro en el RecordStore usando el identificador ingresado, mostrando el valor del registro en cada control, o mostrando un mensaje indicando que el registro no existe

La siguiente figura indica como deberá verse la pantalla



Debe implementar el siguiente código:

```

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.ItemCommandListener;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreNotOpenException;

public class Agenda extends MIDlet implements CommandListener, ItemCommandListener {
    // Referencia a la pantalla
    private Display dsp = Display.getDisplay(this);

    // Creacion de menu
    private List lstMenu = new List("Agenda", List.IMPLICIT,
        new String [] {"Agregar", "Buscar", "Salir"}, null);

    // Creacion de formulario Agregar y sus controles
    private Form frmAltas = new Form("Agregar");
    private TextField txfNombre = new TextField("Nombre", "", 50, TextField.ANY);
    private TextField txfEmail = new TextField("Email", "", 50, TextField.EMAILADDR);
    private TextField txfTelefono = new TextField("Telefono", "", 14,
        TextField.PHONENUMBER);
    private StringItem stiRegistrar = new StringItem("", "Registrar", Item.BUTTON);

    // Creacion de formulario Buscar y sus controles
    private Form frmBajas = new Form("Buscar");
    private TextField txfId = new TextField("Identificador", "", 3, TextField.NUMERIC);
    private StringItem stiNombre = new StringItem("Nombre", "", Item.PLAIN);
    private StringItem stiEmail = new StringItem("Email", "", Item.PLAIN);
    private StringItem stiTelefono = new StringItem("Telefono", "", Item.PLAIN);
    private StringItem stiEliminar = new StringItem("", "Eliminar", Item.BUTTON);

    // Creacion de botones
    private Command cmdMenu = new Command("Menu", Command.BACK, 2);
    private Command cmdRegistrar = new Command("Registrar", Command.OK, 2);
    private Command cmdProximo = new Command("Proximo", Command.OK, 2);
    private Command cmdEliminar = new Command("Eliminar", Command.CANCEL, 2);
    private Command cmdBuscar = new Command("Buscar", Command.CANCEL, 2);

    // Delimitador usado para separar los campos de cada registro
    private final String DELIMITADOR = "%*%";

    public Agenda() {
        // Para procesamiento de eventos del men
        lstMenu.setCommandListener(this);

        // Agregar los controles al Agregar
        stiRegistrar.addCommand(cmdRegistrar);
        stiRegistrar.setDefaultCommand(cmdRegistrar);
        stiRegistrar.setItemCommandListener(this);
        stiRegistrar.setLayout(Item.LAYOUT_CENTER);
        frmAltas.append(txfNombre);
        frmAltas.append(txfEmail);
        frmAltas.append(txfTelefono);
        frmAltas.append(stiRegistrar);
        frmAltas.addCommand(cmdMenu);
        frmAltas.addCommand(cmdProximo);
        frmAltas.setCommandListener(this);
    }

```

```

// Agregar los controles al Buscar
stiEliminar.addCommand(cmdEliminar);
stiEliminar.setDefaultCommand(cmdEliminar);
stiEliminar.setItemCommandListener(this);
stiEliminar.setLayout(Item.LAYOUT_CENTER);
frmBajas.append(txfId);
frmBajas.append(stiNombre);
frmBajas.append(stiEmail);
frmBajas.append(stiTelefono);
frmBajas.append(stiEliminar);
frmBajas.addCommand(cmdMenu);
frmBajas.addCommand(cmdBuscar);
frmBajas.setCommandListener(this);
}

protected void destroyApp(boolean arg0) { }

protected void pauseApp() { }

protected void startApp() throws MIDletStateChangeException {
    // Establecer ventana inicial
    dsp.setCurrent(lstMenu);
}

public void commandAction(Command c, Displayable d) {
    // Procesamiento de eventos
    if(c == List.SELECT_COMMAND) {
        // Procesamiento de eventos del menu
        switch(lstMenu.getSelectedIndex()){
            case 0:
                // Opcion agregar
                limpiarControlesAltas();
                dsp.setCurrent(frmAltas);
                break;
            case 1:
                // Opcion buscar y eliminar
                limpiarControlesBajas();
                dsp.setCurrent(frmBajas);
                break;
            case 2:
                // Opcion Salir
                destroyApp(false);
                notifyDestroyed();
                break;
        }
    } else if(c == cmdMenu) {
        // Regresar al menu inicial
        dsp.setCurrent(lstMenu);
    } else if(c == cmdProximo) {
        // Limpiar controles del formulario frmAltas
        limpiarControlesAltas();
    } else if(c == cmdBuscar) {
        // Realizar la busqueda
        if(!llenarControlesBajas(Integer.parseInt(txfId.getString())) {
            // Si la busqueda no tuvo exito, mostrar mensaje
            Alert altMensaje = new Alert("Error", "Registro no existe", null,
                                         AlertType.ERROR);
            dsp.setCurrent(altMensaje);
        }
    }
}

private boolean llenarControlesBajas(int idRegistro) {
    // Metodo que busca un registro y muestra sus datos
    RecordStore rs = null;
    try
    {
        // Crear el RecordStore y lee el registro indicado en el parametro
        rs = RecordStore.openRecordStore("Agenda", true);
        byte abyteRegistro[] = rs.getRecord(idRegistro);
    }
}

```

```

        // Convierte el registro a una cadena para dividirlo en sus campos
        String strRegistro = new String(abytRegistro);
        int intIndiceEmail = strRegistro.indexOf(DELIMITADOR) + DELIMITADOR.length();
        int intIndiceTelefono = strRegistro.indexOf(DELIMITADOR, intIndiceEmail) +
                                DELIMITADOR.length();
        String strNombre=strRegistro.substring(0,intIndiceEmail- DELIMITADOR.length());
        String strEmail = strRegistro.substring(intIndiceEmail,
                                                intIndiceTelefono - DELIMITADOR.length());
        String strTelefono = strRegistro.substring(intIndiceTelefono);

        // Muestra los datos obtenidos
        stiNombre.setText(strNombre);
        stiEmail.setText(strEmail);
        stiTelefono.setText(strTelefono);

        // Devuelve verdadero si la busqueda fue conforme
        return true;
    } catch(Exception ex) {
        // Imprime el error en la consola para depuracion
        ex.printStackTrace();
    } finally {
        // Cierra el RecordStore
        cerrarRecordStore(rs);
    }

    // Devuelve falso si la busqueda no fue exitosa o hubo un error
    return false;
}

private void limpiarControlesAltas() {
    // Inicializa los controles del formulario frmAltas
    txfNombre.setString("");
    txfEmail.setString("");
    txfTelefono.setString("");
}

private void limpiarControlesBajas() {
    // Inicializa los controles del formulario frmBajas
    stiEmail.setText("");
    stiNombre.setText("");
    stiTelefono.setText("");
}

private void cerrarRecordStore(RecordStore rs) {
    // Cierra el RecordStore pasado como parametro
    if(rs != null) {
        try {
            rs.closeRecordStore();
        } catch (Exception ex) {
            // Imprime el error en la consola para depuracion
            ex.printStackTrace();
        }
    }
}

public void commandAction(Command c, Item i) {
    // Procesamiento de eventos
    RecordStore rs = null;
    if(c == cmdRegistrar) {
        // Grabacion de un registro
        try {
            // Se crea una cadena con los valores de todos los campos
            String strRegistro = txfNombre.getString() + DELIMITADOR
                                + txfEmail.getString() + DELIMITADOR
                                + txfTelefono.getString();

            // Creacion del RecordStore y adiccion del registro
            rs = RecordStore.openRecordStore("Agenda", true);
            int numRegistro = rs.addRecord(strRegistro.getBytes(), 0,
                                           strRegistro.getBytes().length);

```



```
        // Mostrar mensaje con el resultado
        Alert altMensaje = new Alert("Informacion", "Registro ingresado "
                                     + numRegistro, null, AlertType.INFO);
        dsp.setCurrent(altMensaje);
    } catch(Exception ex) {
        // Imprime el error en la consola para depuracion
        ex.printStackTrace();
    } finally {
        // Cerrar RecordStore
        cerrarRecordStore(rs);
    }
} else if(c == cmdEliminar) {
    // Eliminacion de un registro
    // Esta variable nos indicara si se elimino correctamente o no
    boolean blnEliminado = false;

    try {
        // Se obtiene el identificador ingresado
        int idRegistro = Integer.parseInt(txfId.getString());

        // Se abre el RecordStore y se elimina el registro
        rs = RecordStore.openRecordStore("Agenda", true);
        rs.deleteRecord(idRegistro);

        // Se limpian los controles del formulario frmBajas
        limpiarControlesBajas();

        // Se actualize el valor de la variable
        blnEliminado = true;
    } catch(Exception ex) {
        // Imprime el error en la consola para depuracion
        ex.printStackTrace();
    } finally {
        // Cerrar RecordStore
        cerrarRecordStore(rs);
    }

    // Se muestra un mensaje de exito o error segun el valor de la
    // variable blnEliminado
    Alert altMensaje = new Alert("Resultado");
    if(blnEliminado) {
        altMensaje.setString("Registro eliminado correctamente");
        altMensaje.setType(AlertType.INFO);
    } else {
        altMensaje.setString("Se produjo un error ");
        altMensaje.setType(AlertType.ERROR);
    }

    dsp.setCurrent(altMensaje);
}
}
```

Resumen

- 📖 Para almacenamiento de datos, la plataforma JME proporciona una base de datos basada en registros conocida como RMS. El RMS está conformado por conjuntos de registros o *RecordStores*, los cuales almacenan los datos en registros conteniendo arreglos de *bytes*.
- 📖 Existe un esquema de seguridad que consiste en que sólo puede acceder al *RecordStore* el *MIDlet* que lo creó y los *MIDlets* pertenecientes al mismo *MIDlet suite*. Para que otros *MIDlets* puedan acceder, el *MIDlet* creador debe especificar dicho permiso al momento en que el *RecordStore* sea creado.
- 📖 La forma correcta de usar un *RecordStore* es abrirlo, realizar alguna de las operaciones conocidas (agregar, modificar, eliminar y leer) y cerrar.
- 📖 Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:
 - 🔗 <http://www.mailxmail.com/curso/informatica/j2me>
 - 🔗 http://programacion.com/java/tutorial/ags_j2me/
 - 🔗 <http://leo.ugr.es/J2ME/MIDP/enlaces.htm>

UNIDAD DE
APRENDIZAJE

4

CONECTIVIDAD JME

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno elabora aplicaciones que se conecten con un sitio web usando el protocolo http e intercambie datos, aplicando nociones de buenas prácticas de programación, con el empleo de la clase `HTTPConnection` y sus clases relacionadas pertenecientes a la plataforma Java Micro Edition y la interfaz de desarrollo Eclipse.

TEMARIO

4.1 Tema 6 : Aplicaciones conectadas

4.1.1. : Conectividad: Clase `Connector`

4.1.2. : Uso del protocolo HTTP: Clases `Connection` / `HTTPConnection`

4.1.3. : Intercambio de datos: Clases `InputStream` / `OutputStream`

ACTIVIDADES PROPUESTAS

- Los alumnos crean aplicaciones JME que interactúan con otras usando el protocolo http.

4.1 Aplicaciones conectadas

4.1.1. Conectividad: Clase Connector

JME proporciona un conjunto de clases que pueden ser usados por los MIDlets para poder comunicarse con cualquier punto de una red a la cual el dispositivo móvil esté conectado. Generalmente, la red a la que nos conectamos es Internet, y los puntos en la red son sitios web con quienes podemos comunicarnos usando los protocolos http/HTTPS.

A continuación, se dan algunos ejemplos del uso de las comunicaciones en aplicaciones JME:

- ✓ Validación de usuario y clave antes de ingresar a la aplicación
- ✓ Lectura de datos guardados en una base de datos central
- ✓ Envío de los datos ingresados en un RecordStore para ser grabados en una base de datos central

Un aspecto importante, cuando se construye una aplicación de este tipo, es que la parte correspondiente a la comunicación va siempre en un segundo hilo o proceso. Esto es porque este tipo de conexión es generalmente lento, y podrían bloquear el equipo si se usa un solo hilo o proceso. Asimismo, tomar en cuenta que las conexiones deben cerrarse al finalizar la aplicación.

Clase Connection. Clase usada para crear los objetos de tipo Connection, que son las que se usarán para conectarse. Cuenta con los siguientes métodos:

open()

-Definición

```
public static Connection open(String url)
public static Connection open(String url, int modo)
public static Connection open(String url, int modo,
                             boolean timeout)
```

-Descripción

Permite crear un objeto de tipo Connection. Sus parámetros son:

- ✓ url: Url del sitio a donde se desea conectarse
- ✓ modo: Modo de conexión.
- ✓ timeout: Dato lógico que indica si se usan los valores de timeout del equipo

4.1.2. Uso del protocolo HTTP: Clases Connection / HTTPConnection

Clase Connection

Clase que manipula la conexión con el sitio de red. De esta interfaz, heredan varias sub interfaces según el protocolo. De este modo, para conectarse con un sitio usando el protocolo http, se usara la interfaz HTTPConnection.

Clase **HTTPConnection**

Es una conexión de tipo HTTP. Permite conectarse a un sitio Web y realizar operaciones de lectura y escritura. Debido a que la conexión a un sitio Web puede ser un proceso lento, es recomendable poner todo el procesamiento dentro de un hilo diferente del proceso principal.

A continuación se describen sus principales métodos:

close()

-Definición

```
public void close()
```

-Descripción

Cierra la conexión.

getLength()

-Definición

```
public long getLength()
```

-Descripción

Obtiene la longitud de la respuesta del sitio web.

getRequestMethod()

-Definición

```
public String getRequestMethod()
```

-Descripción

Obtiene el método de comunicación con el sitio web. Los métodos están dados por las constantes `HTTPConnection.GET` y `HTTPConnection.POST`.

getRequestProperty()

-Definición

```
public String getRequestMethod(String propiedad)
```

-Descripción

Obtiene el valor de una propiedad de la consulta HTTP.

getResponseCode()

-Definición

```
public int getResponseCode()
```

-Descripción

Obtiene el código de respuesta del sitio web. La lista de valores es grande. A continuación, algunos de ellos:

- ✓ HTTPConection.HTTP_OK: Conexión correcta
- ✓ HTTPConection.HTTP_NOT_FOUND: Sitio no encontrado
- ✓ HTTPConection.HTTP_FORBIDDEN: Sitio prohibido
- ✓ HTTPConection.HTTP_INTERNAL_ERROR: Error interno

openInputStream()

-Definición

```
public InputStream openInputStream()
```

-Descripción

Obtiene un objeto de tipo InputStream, necesario para leer los datos del sitio web.

openOutputStream()

-Definición

```
public OutputStream openOutputStream()
```

-Descripción

Obtiene un objeto de tipo OutputStream, necesario para escribir los datos hacia el sitio web.

4.1.3. Intercambio de datos: Clases InputStream / OutputStream

Clase InputStream

Clase usada para leer datos desde una conexión. Sus principales métodos son:

close()

-Definición

```
public void close()
```

Descripción

Cierra el flujo de entrada.

read()

-Definición

```
public int read()  
public int read(byte [] datos)  
public int read(byte [] datos, int inicio, int longitud)
```

-Descripción

Lee los datos de respuesta del sitio web. En el primer caso, devuelve un entero leído del sitio web. En los otros casos, los bytes leídos se guardan en el arreglo "datos" y se devuelve la cantidad de bytes. Sus parámetros son:

- ✓ **datos:** Arreglo que contendrá los datos leídos
- ✓ **inicio:** Índice en el arreglo donde se comenzarán a grabar los datos
- ✓ **longitud:** Cantidad de bytes que se guardarán en el arreglo

Clase **OutputStream**

Clase usada para escribir datos hacia una conexión. Sus principales métodos son:

close()

-Definición

```
public void close()
```

-Descripción

Cierra el flujo de salida.

read()

-Definición

```
public void write(int dato)
public void write(byte [] datos)
public void write(byte [] datos, int inicio, int
longitud)
```

-Descripción

Escribe los datos al sitio web. En el primer caso, escribe el dato entero especificado. En los otros casos, se escribirán los bytes del arreglo "datos". Sus parámetros son:

- ✓ **datos:** Arreglo que contendrá los datos que van a ser escritos
- ✓ **inicio:** Índice en el arreglo donde se comienzan los datos que van a ser escritos
- ✓ **longitud:** Cantidad de bytes en el arreglo que van a ser escritos

4.1.3.1. Ejercicio: Aplicación Inicio sesión

Construir una aplicación que realice lo siguiente:

Mostrar un formulario con el título "Login" y que permita ingresar los siguientes datos:

- ✓ Usuario (alfanumérico, 10 caracteres)
- ✓ Clave (alfanumérico y contraseña, 10 caracteres)
- ✓ Adicionalmente, los siguientes botones
- ✓ Un botón "Aceptar", que se conectará con un sitio web pasándole como parámetros el usuario y la clave
- ✓ Un botón "Salir" para salir de la aplicación

La aplicación debe mostrar el resultado devuelto por el sitio web en un StringItem llamado "Resultado"

The image shows a mobile application window with a title bar containing a signal icon, the text 'ABC', and a battery icon. The main content area is titled 'Login' and contains the following elements:

- A label 'Usuario' followed by a text input field.
- A label 'Clave' followed by a text input field.
- A label 'Resultado' followed by a large, empty rectangular area for displaying the result.

At the bottom of the window, there is a bar with two buttons: 'Salir' on the left and 'Aceptar' on the right.

Debe implementar el siguiente código:

```

import java.io.InputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class InicioSesionHTTP extends MIDlet implements CommandListener, Runnable {

    private Display dsp = Display.getDisplay(this);
    private Form frm = new Form("Login");
    private TextField txfUsuario = new TextField("Usuario", "", 10, TextField.ANY);
    private TextField txfClave = new TextField("Clave", "", 10, TextField.ANY |
                                                TextField.PASSWORD);
    private StringItem stiResultado = new StringItem("Resultado", "", Item.PLAIN);

    private Command cmdAceptar = new Command("Aceptar", Command.OK, 2);
    private Command cmdSalir = new Command("Salir", Command.EXIT, 2);

    public InicioSesionHTTP() {
        frm.append(txfUsuario);
        frm.append(txfClave);
        frm.append(stiResultado);
        frm.addCommand(cmdAceptar);
        frm.addCommand(cmdSalir);
        frm.setCommandListener(this);
    }

    protected void destroyApp(boolean arg0) { }
    protected void pauseApp() { }

    protected void startApp() throws MIDletStateChangeException {
        dsp.setCurrent(frm);
    }

    public void commandAction(Command c, Displayable d) {
        // Procesamiento de los botones
        if(c == cmdAceptar) {
            // Iniciar un hilo en segundo plano
            Thread t = new Thread(this);
            t.start();
        } else if(c == cmdSalir) {
            destroyApp(false);
            notifyDestroyed();
        }
    }

    public void run() {
        // Conexion al sitio web
        HttpConnection http = null;
        try {
            // Establecer la direccion del sitio web
            String strURL = "http://127.0.0.1/ServidorMIDP/inicio.asp?usuario="
                + txfUsuario.getString() + "&clave=" + txfClave.getString();
            System.out.println(strURL);

            // Conexion al sitio web
            http = (HttpConnection) Connector.open(strURL);
            if(http.getResponseCode() == HttpConnection.HTTP_OK) {
                // Lectura de los datos devueltos
                int lngBytes = (int) http.getLength();
                byte abytdatos[] = new byte[lngBytes];
                InputStream is = http.openInputStream();
                is.read(abytdatos);
                is.close();
            }
        } catch (Exception e) {
            // Manejo de excepciones
        }
    }
}

```

```
        // Mostrar los datos devueltos
        String strDatos = new String(abytDatos);
        System.out.println(strDatos);
        stiResultado.setText(strDatos);
    }

    System.out.println("fin");
} catch(Exception ex) {
    // Mostrar los datos devueltos
    stiResultado.setText("Error: " + ex.toString());
    System.out.println("Error: " + ex.toString());
} finally {
    // Cerrar la conexion
    if (http != null) {
        try {
            http.close();
        } catch(Exception ex) {}
    }
}
}
```

Resumen

📖 La plataforma JME proporciona un conjunto de clases para permitir la conexión vía HTTP con un sitio Web.

📖 Debido a que la comunicación con un sitio Web es un proceso lento, se recomienda realizarlo en un hilo diferente del proceso principal.

📖 Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

🔗 <http://www.mailxmail.com/curso/informatica/j2me>

🔗 http://programacion.com/java/tutorial/ags_j2me/

🔗 <http://leo.ugr.es/J2ME/MIDP/enlaces.htm>

UNIDAD DE
APRENDIZAJE**5**

ANDROID: FUNDAMENTOS Y PRINCIPALES COMPONENTES

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno elabora aplicaciones que integran Activities, componentes visuales, Intents y recursos externos pertenecientes a la plataforma Android.

TEMARIO

5.1 Tema 7 : Fundamentos de Android

- 5.1.1. : Arquitectura de una aplicación móvil Android. Configuración, principales librerías y herramientas de desarrollo
- 5.1.2. : SDK (Standard Development Kit). Virtual Machine. Aplicación “Hola Mundo” para la plataforma Android

5.2 Tema 8 : Fundamentos de aplicaciones Android

- 5.2.1. : Android Software Stack y el ciclo de vida de una aplicación Android.
- 5.2.2. : Prioridades y estados de una aplicación Android.
- 5.2.3. : Principales componentes: Introducción al uso de Actividades y la gestión de Recursos.

5.3 Tema 9 : Interfaces de Usuario

5.3.1. : Fundamentos de diseño y creación de componentes básicos Android.

5.3.2. : Creación de vistas personalizadas, controles y gestión de eventos.

5.4 Tema 10 : Componente Intent

5.4.1. : Definición y características: Uso de Intents para gestionar Activities.

ACTIVIDADES PROPUESTAS

- Los alumnos instalan y configuran el entorno de desarrollo Android.
- Los alumnos implementan una aplicación móvil básica utilizando las principales características de la plataforma de desarrollo Android

5.1 Fundamentos de Android

Android es un “conjunto” de componentes software (software “stack”) especialmente diseñado para dispositivos móviles; incluye básicamente: un sistema operativo, un middleware y aplicaciones base. Para trabajar con Android podemos descargar el Standard Development Kit (Android SDK), el cual nos proporciona las herramientas y librerías necesarias para empezar a desarrollar aplicaciones sobre la plataforma Android utilizando el lenguaje de programación Java.

5.1.1. Arquitectura Android

Se muestra a continuación un diagrama con los principales componentes desplegados sobre el sistema operativo Android:

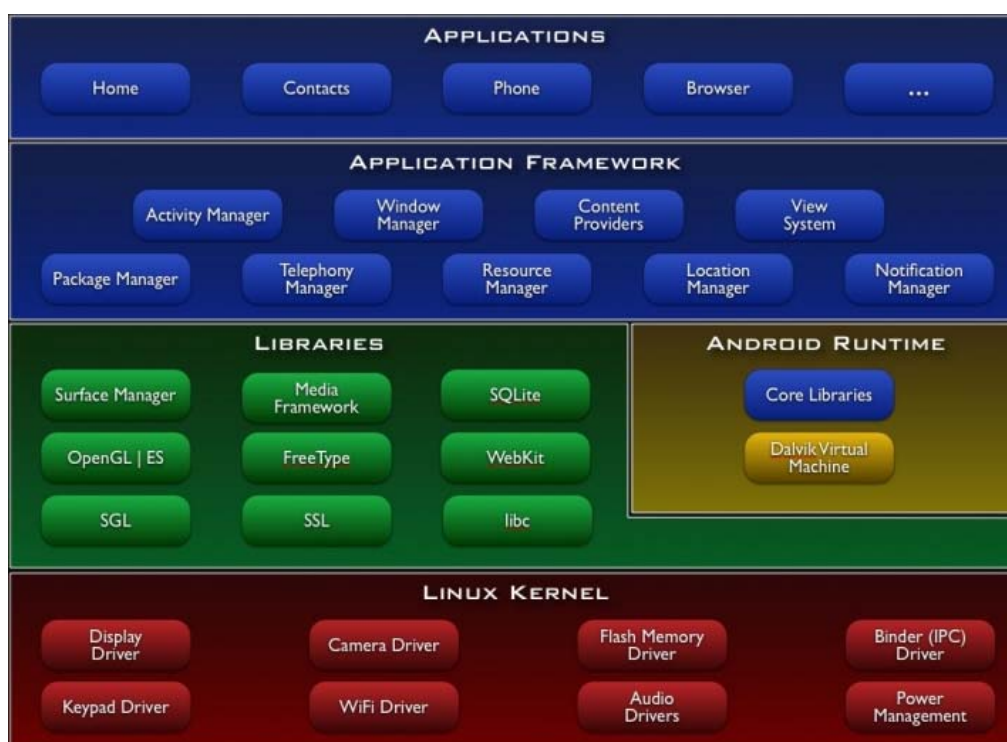


Figura 5.1

5.1.1.1. Aplicaciones

Android cuenta con un conjunto de aplicaciones “core” que incluyen entre otras un cliente email, un programa para enviar mensajes SMS, calendario, mapas, un browser, contactos personales. Todas las aplicaciones han sido escritas utilizando el lenguaje de programación Java.

5.1.1.2. Framework de aplicaciones

Android proporciona una plataforma de desarrollo abierta, esta característica facilita que los desarrolladores puedan construir de manera colaborativa, aplicaciones extremadamente ricas y innovadoras: los desarrolladores tienen total libertad para aprovechar las características del dispositivo de hardware con el que estén trabajando, obtener información de localización, ejecutar servicios en background, configurar alarmas, agregar notificaciones a la barra de estado, entre otras características.

Los desarrolladores tienen también acceso a las mismas librerías que son usadas por las aplicaciones core de Android. La arquitectura de las aplicaciones Android está diseñada para reutilizar componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (tomando en cuenta las restricciones de seguridad establecidas por el framework). Este mismo mecanismo permite que los componentes puedan ser reemplazados por el usuario.

Detrás de todas las aplicaciones Android existen un conjunto de servicios y sistemas los cuales se describen a continuación:

- ✓ Views. Un conjunto de vistas que pueden ser usadas para construir una aplicación, incluyendo listas, grids, cajas de texto, botones e incluso un navegador web que puede ser incrustado dentro de otros componentes.
- ✓ Content Providers. Que posibilitan a las aplicaciones acceder a datos de otras aplicaciones (tales como la relación de Contactos), o compartir sus propios datos.
- ✓ Un Resource Manager. Proporcionan acceso a recursos externos no basados en código (que no se encuentran dentro de una clase java) tales como cadenas localizadas, gráficos y archivos xml de layouts.
- ✓ Un Notification Manager. Posibilita a todas las aplicaciones mostrar alertas customizadas en la barra de estado.
- ✓ Un Activity Manager. Gestiona el ciclo de vida de las aplicaciones Android.

5.1.1.3. Librerías

Android incluye un conjunto de librerías C/C++ usadas por varios componentes del sistema Android. Estas capacidades son expuestas a los desarrolladores a través del framework de aplicación Android. Algunas de estas librerías core se listan a continuación:

- System C library. Una implementación Berkeley Software Distribution (BSD) derivada de las librerías C estándar (libc), especialmente diseñada para ser incrustada en dispositivos basados en Linux.

- ✓ **Media Libraries.** Basadas en PacketVideo's OpenCORE; las librerías soportan la reproducción y grabación de muchos formatos populares de audio y video así como el uso de archivos de imágenes tales como MPEG4, H.264, MP3, AAC, AMR, JPG, y PNG.
- ✓ **Surface Manager.** Gestiona el acceso y uso de capas para componer gráficos 2D y 3D y utilizarlos en múltiples aplicaciones.
- ✓ **LibWebCore.** Un motor de navegador web bastante moderno que es la base del navegador Android así como del component web View.
- ✓ **SGL.** Motor para generar gráficos 2D.
- ✓ **Librerías 3D.** Una implementación basada en OpenGL ES 1.0 APIs; esta librería usa aceleración de gráficos 3D por hardware (cuando está disponible) o aceleración de gráficos 3d vía software (componentes que sí están incluidos).
- ✓ **SQLite.** Motor de base de datos relacional bastante ligero, disponible para todas las aplicaciones Android.

5.1.1.4. Entorno de ejecución Android (Android Runtime)

Android incluye un conjunto de librerías core que proporcionan la mayoría de las funcionalidades disponibles en las librerías core del lenguaje de programación Java.

5.1.1.5. Linux Kernel

Android se basa en la versión 2.6 de Linux para servicios core del sistema tales como seguridad, gestión de memoria, gestión de procesos y conectividad. El kernel también actúa como una capa de abstracción entre el hardware y el resto del software “stack”.

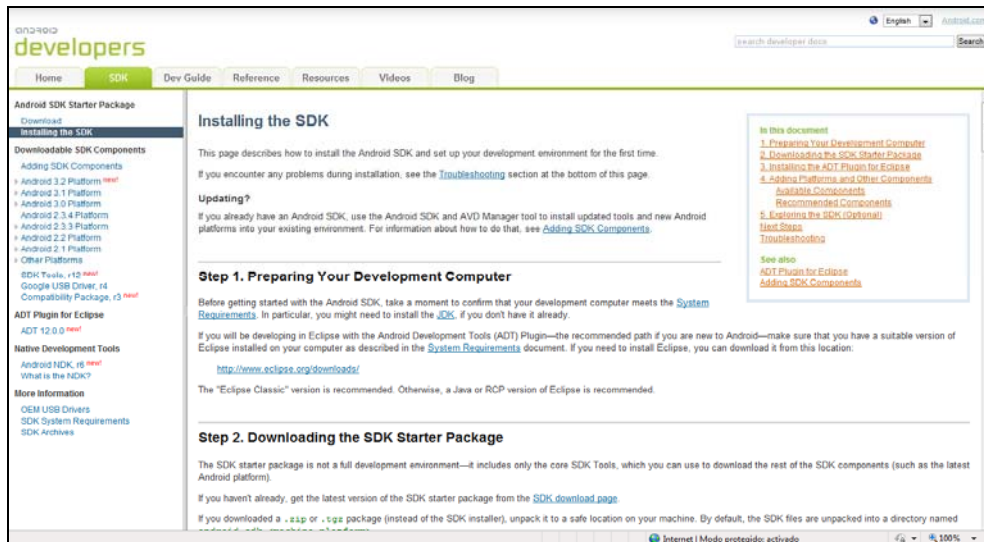
5.1.2. SDK (Standard Development Kit). Virtual Machine. Aplicación “Hola Mundo” para la plataforma Android.

Se describen a continuación los pasos a seguir para la instalación y configuración del entorno de desarrollo Android, así como la creación de la aplicación “Hola Mundo”

5.1.2.1. Instalación y configuración de Android

Antes de instalar los componentes se debe verificar si nuestra computadora cumple con los requisitos mínimos para poder desarrollar aplicaciones Android. Estos pueden ser revisados en detalle en el siguiente enlace:

<http://developer.android.com/sdk/installing.html>



Los requisitos más importantes son los siguientes:

- ✓ Tener instalado el JDK (versiones 5 o 6) y
- ✓ Contar con la IDE Eclipse (versión 3.5 o superior)

Una vez verificados los requisitos, seguir los siguientes pasos:

a) Paso 1: Descargar el Standard Development Kit de Android (SDK)

Para ello, ingresar a la siguiente dirección:

<http://developer.android.com/sdk/index.html>

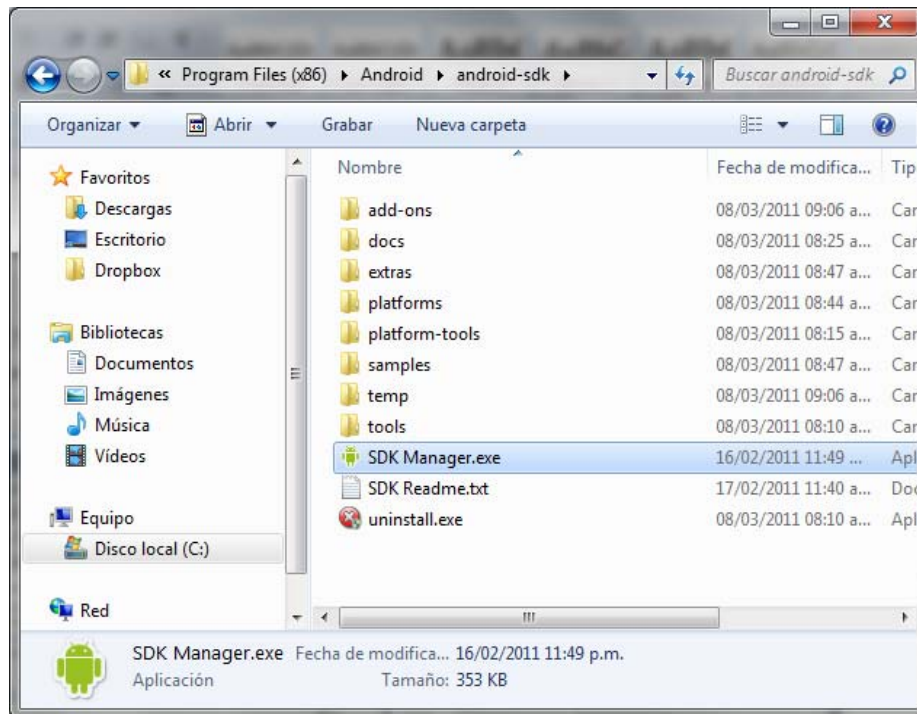
Se visualizarán diversas opciones de descarga, tal como se muestran a continuación:

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r12-windows.zip	36486190 bytes	8d6c104a34cd2577c5506c55d981aebf
	installer_r12-windows.exe (Recommended)	36531492 bytes	367f0ed4ecd70aefc290d1f7dcb578ab
Mac OS X (intel)	android-sdk_r12-mac_x86.zip	30231118 bytes	341544e4572b4b1afab123ab817086e7
Linux (i386)	android-sdk_r12-linux_x86.tgz	30034243 bytes	f8485275a8dee3d1929936ed538ee99a

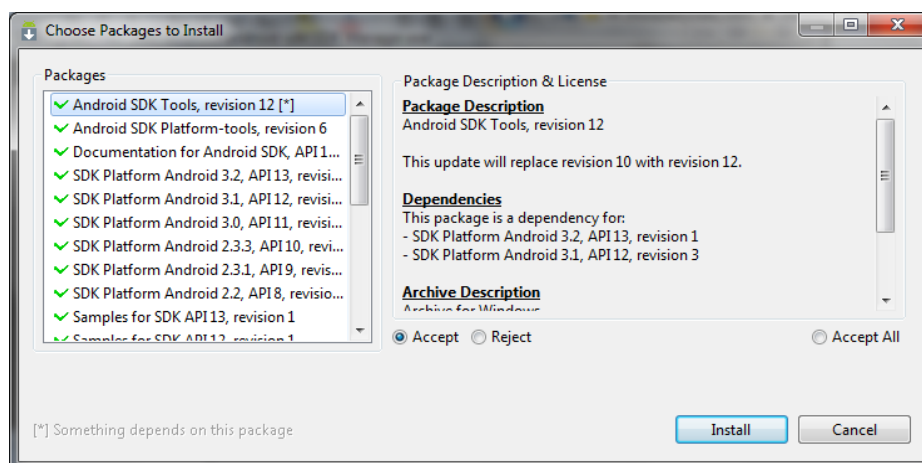
Nota.- Dado que el entorno de trabajo para este manual es windows seleccionar el paquete para dicha plataforma. Una vez descargado el archivo, instalar el SDK.

b) Paso 2: Instalar los **paquetes de desarrollo Android** que se utilizarán para la implementación de las aplicaciones.

Una vez instalado el SDK, ubicarse en la carpeta de instalación tal como se muestra en la siguiente pantalla:

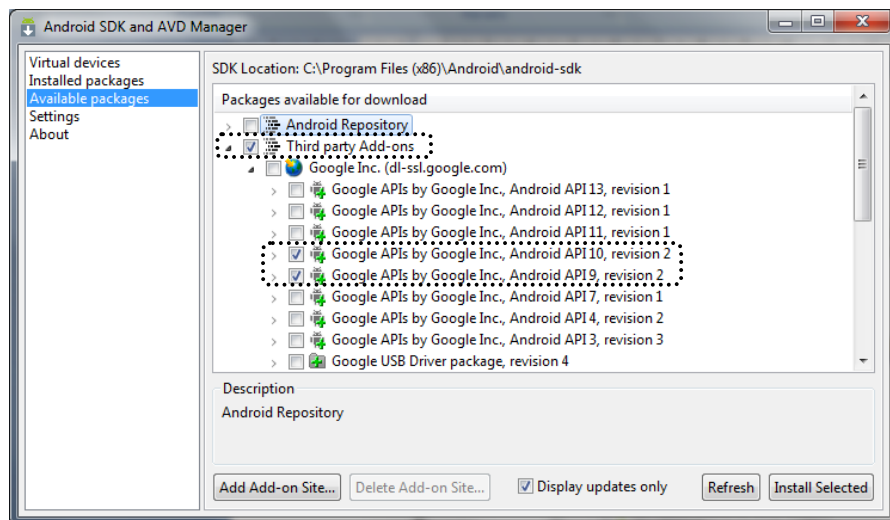
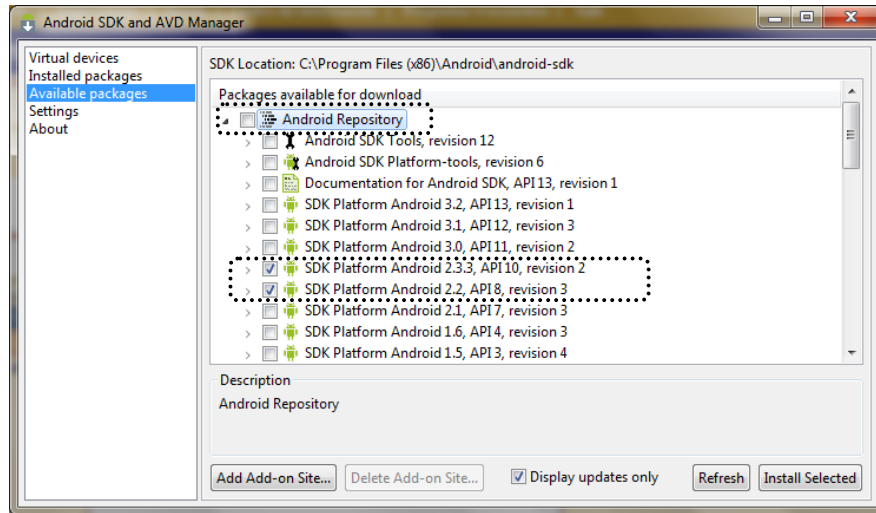


Seleccionar el archivo SDK manager.exe para acceder a las opciones de configuración de la herramienta. Por defecto se visualiza la siguiente pantalla:



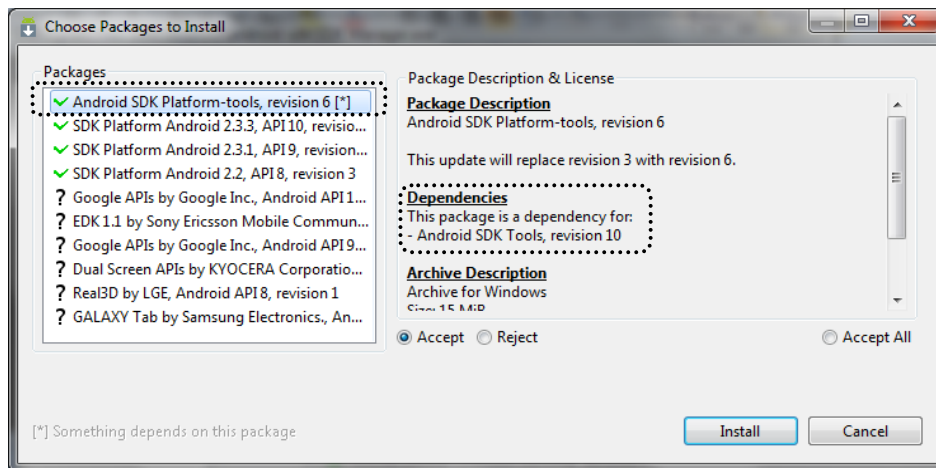
En esta pantalla se observan todos los paquetes disponibles seleccionados por defecto para su instalación. Dado que muchos de ellos son para dispositivos móviles tipo “Table” y no SmartPhones, seleccionar el botón “Cancel”.

Al seleccionar la opción Cancelar en la pantalla previa, se visualiza la siguiente ventana. En ella se selecciona la entrada “Paquetes disponibles” y se escogen solo aquellos paquetes que se desea instalar.



Posteriormente, pulsar el botón “Install Selected”.

Se visualizará la pantalla inicial solo con los paquetes seleccionados. Finalmente seleccionar el botón “Instalar”.



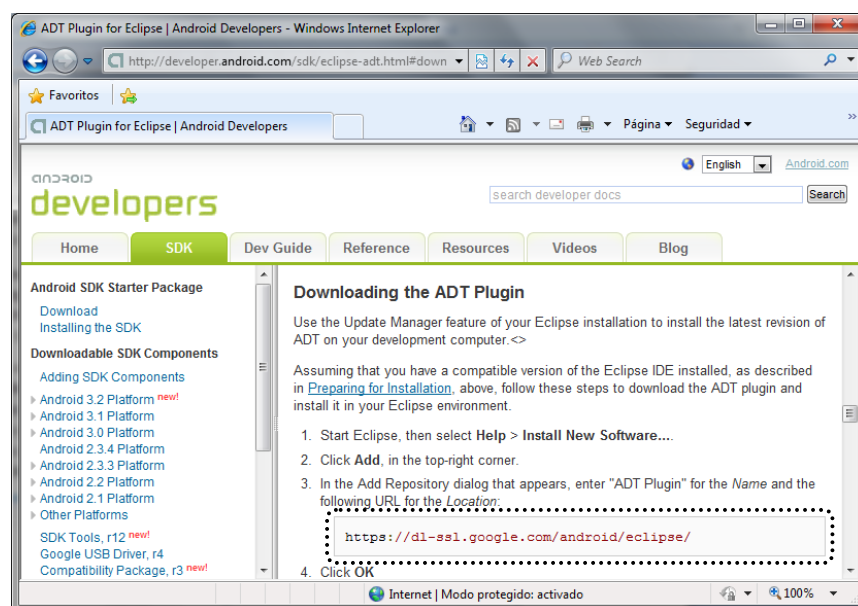
Notas.-

- 1) Nótese que al instalar un paquete en particular, este puede **tener dependencias** que también se deberán instalar para su correcto funcionamiento. En el ejemplo, el paquete seleccionado, es una dependencia de otros dos.
- 2) Para el presente manual, se decidió instalar las versiones 2.3.3 y 2.2 del repositorio Android y las versiones nivel 9 y 10 de la API de Google puesto que cubren la mayoría de versiones instaladas en los dispositivos móviles con soporte para Android en la actualidad.

c) Paso 3: Instalar el PlugIn de Android (ADT) para Eclipse.

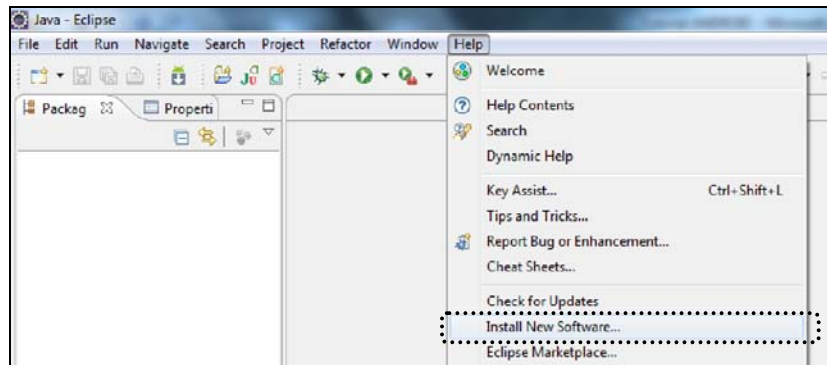
Se cuenta con información detallada del Android Development ToolKit (ADT) en el siguiente enlace:

<http://developer.android.com/sdk/eclipse-adt.html>

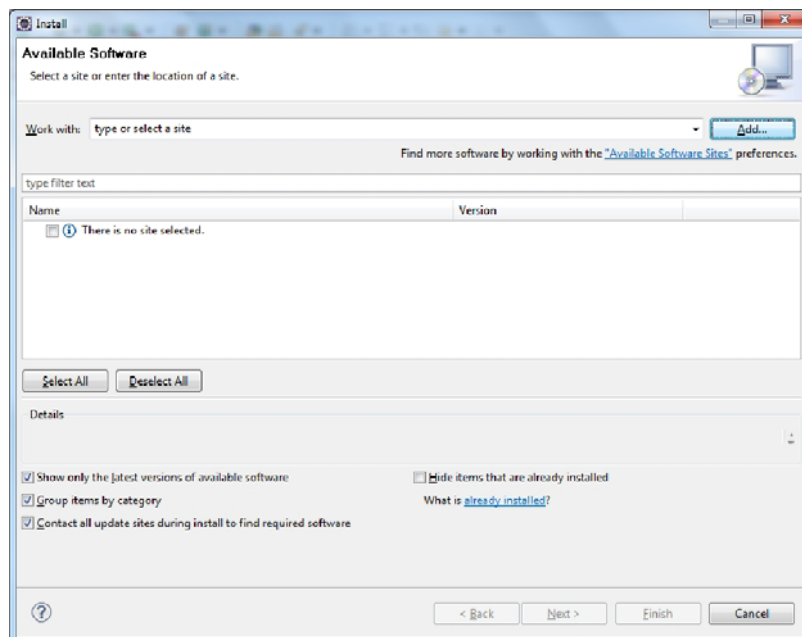


Nota.- <https://dl-ssl.google.com/android/eclipse/> es la url de descarga que debe colocarse en la IDE Eclipse para instalar el ADT.

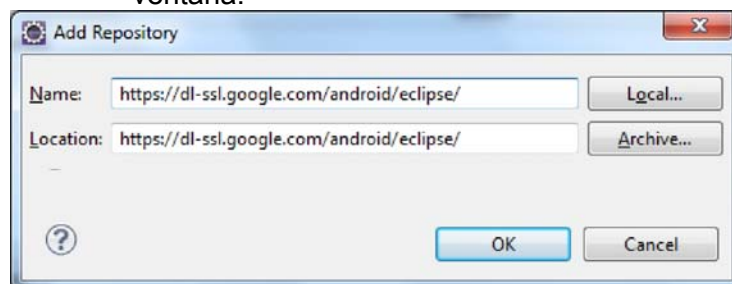
Para el presente manual se asume la pre-existencia de la herramienta de desarrollo Eclipse instalada. Para continuar con la instalación, seleccionar la opción “Install New Software” del menú Help.



Se visualizará la siguiente pantalla:

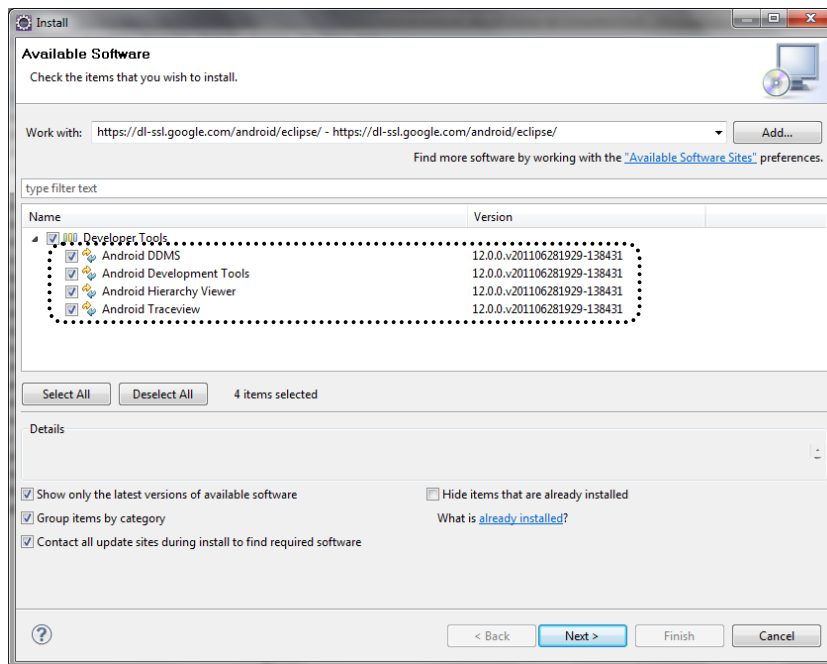


Seleccionar el botón “Add”. Se visualizará la siguiente ventana:

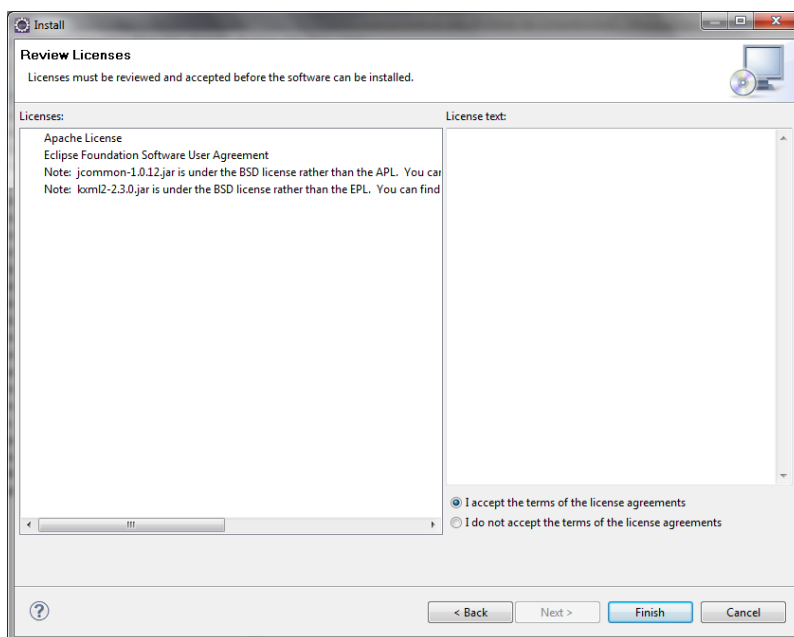


En el campo **Name** se coloca un texto descriptivo que identifique el componente a instalar. En el campo **Location** se debe ingresar el url de descarga del plugin de Android para Eclipse. En el ejemplo se colocó en ambos campos el url de descarga.

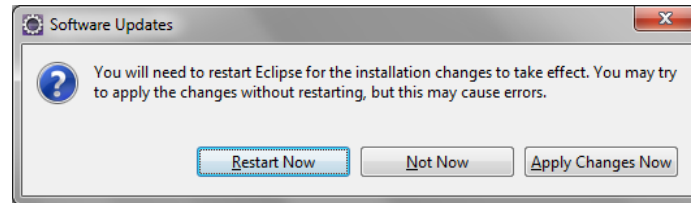
Al seleccionar el botón “Ok” se visualizará una ventana con las herramientas a instalar en Eclipse:



Pulsar el botón “Next” y, en la siguiente pantalla se aceptan los términos y condiciones de la licencia de uso del Plugin.



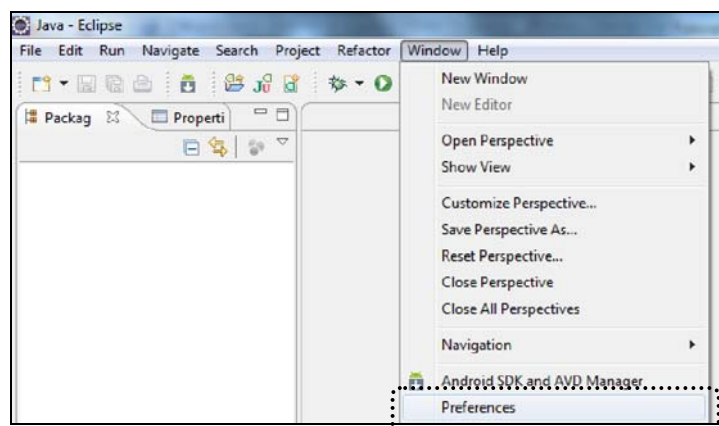
La instalación del Plugin iniciará al pulsar el botón Finish. Luego, la aplicación será reiniciada.



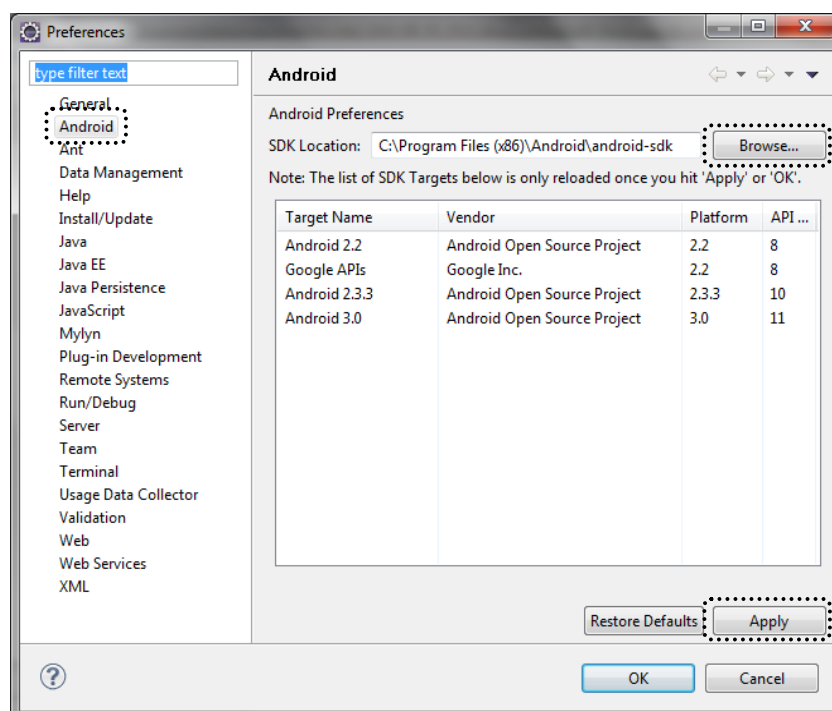
Seleccionar el botón “Restart Now” para terminar el proceso de instalación.

d) Paso 4: Referenciar las librerías de Android desde Eclipse.

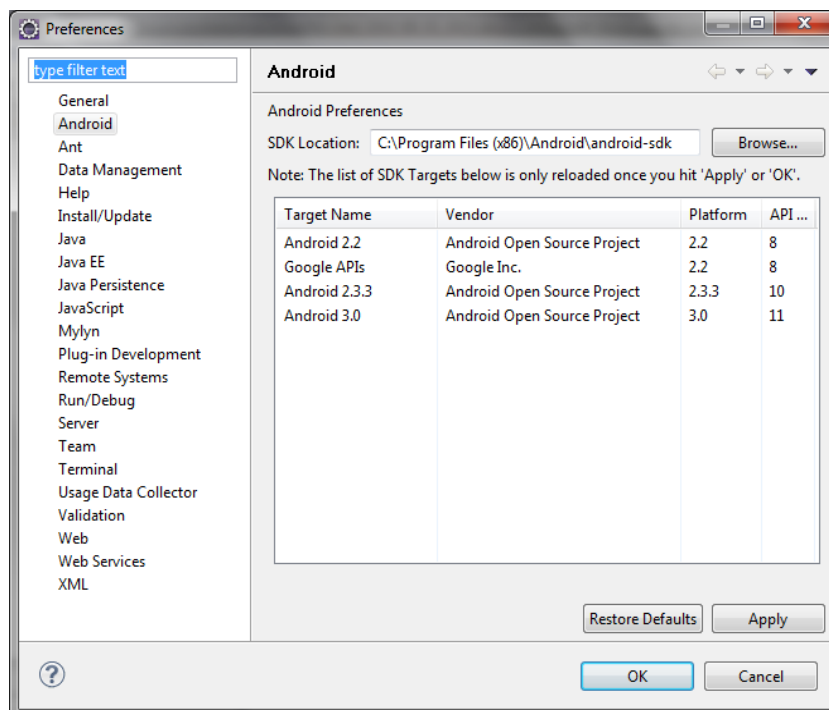
Una vez reiniciado el Eclipse, seleccionar la opción “Preferences” del menú Windows.



Se visualizará la siguiente ventana. Seleccionar la entrada Android y luego con el botón ubicar el directorio de instalación del Android SDK.



Al seleccionar el botón “Apply”, se visualizarán los paquetes de Android descargados previamente.



Finalmente, seleccionar el botón “Ok” para terminar la configuración de la referencia a los paquetes Android instalados con el SDK.

e) Paso 5: ¡Excelente!, se ha culminado de configurar exitosamente el entorno de desarrollo Android.



5.1.2.2. Aplicación Hola Mundo Android

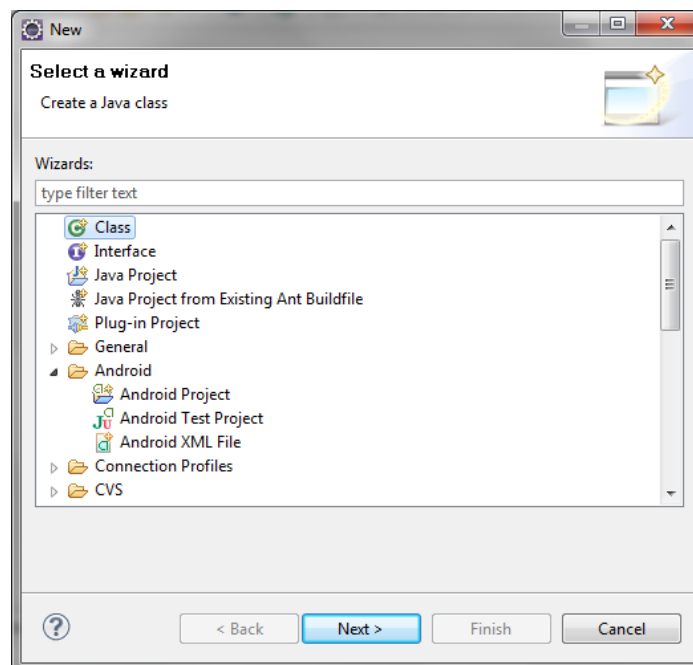
Una vez instalado el PlugIn de Android, Eclipse contará con un nuevo tipo de proyecto: “Android Project”.

a) Paso 1: Seleccionar del menú File, la opción “Android Project”.



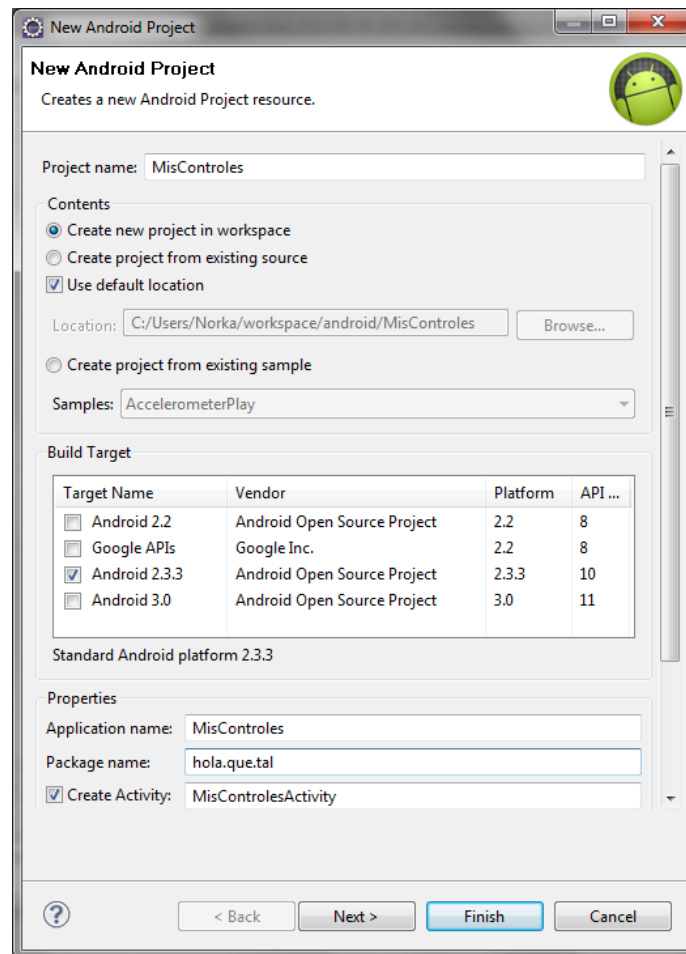
b) Paso 2: De no encontrar la opción, seleccionar del menú File, la opción New / Other.

c) Paso 3: Seleccionar del Wizard la entrada Android y luego Android Project.



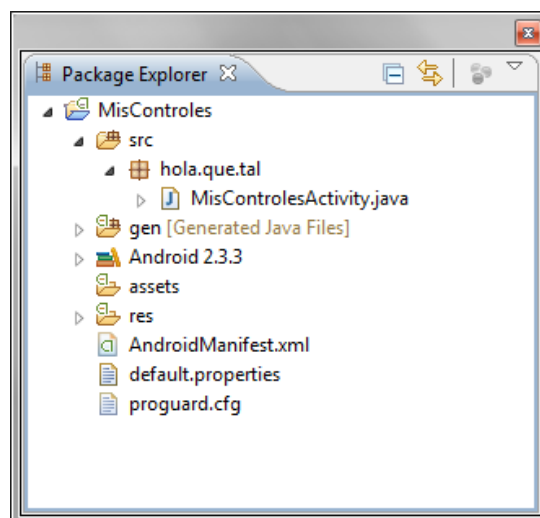
d) Paso 4: Seleccionar el botón “Next”. Se visualizará la siguiente pantalla en la que deberá ingresar los siguientes datos:

- 1) **Project name.**- El nombre del proyecto
- 2) **Build Target.**- La versión de Android con la cual la aplicación será compatible
- 3) **Application name.**- El nombre de la aplicación. Por defecto, es el mismo nombre del proyecto.
- 4) **Package name.**- El nombre del paquete que debe tener, al menos, dos identificadores separados por el operador punto “.”.
- 5) **Create Activity.**- El nombre del componente Activity. Por convención, es el nombre de la aplicación seguido de la palabra Activity.



e) Paso 5: Al seleccionar el botón “Finish” se creará el proyecto **MisControles** dentro de la IDE Eclipse.

De manera automática se crea la clase MisControlesActivity. Este componente es de tipo **Activity** y pertenece a la capa de presentación de Android.



Un Activity debe estar registrado en el archivo de configuración de Android: **AndroidManifest.xml**, tal como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>

<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="hola.que.tal"
    android:versionCode="1"
    android:versionName="1.0">

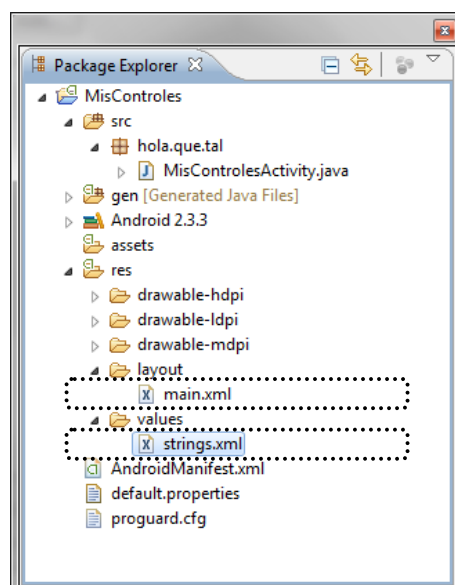
    <uses-sdk android:minSdkVersion="10"/>

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name=".MisControlesActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

En un proyecto Android también se crean de manera automática los siguientes archivos de recursos:

- 1) **main.xml**, es el componente visual que por defecto se crea en la aplicación y contiene componentes tipo **View** tales como Layouts, TextView, Buttons, etc. En el ejemplo, este archivo es referenciado por el Activity MisControlesActivity.
- 2) **string.xml**, es un archivo de recursos en el cual se pueden definir **constantes** que contienen valores de diferentes tipos de datos tales como cadenas, colores, arreglos de cadenas, etc.



Se observa que el archivo `main.xml` posee como componentes un `LinearLayout` y un `TextView`. El campo `TextView` hace referencia a la constante “hello” definida en el archivo `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>

</LinearLayout>
```

La constante “hello” ha sido definida en el archivo `string.xml` con el siguiente valor: “Hello World, MisControlesActivity!”.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <string name="hello">Hello World, MisControlesActivity!</string>
    <string name="app_name">MisControles</string>
</resources>
```

La clase `MisControlesActivity` referencia al archivo `main.xml` a través del método `setContentView`.

```
package hola.que.tal;

import android.app.Activity;
import android.os.Bundle;

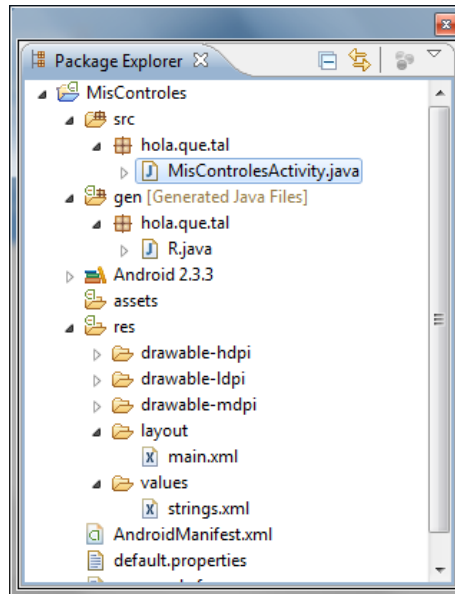
public class MisControlesActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

}
```

Nota.- El método `setContentView` referencia al archivo `main.xml` a través de la clase `R` y su componente `layout.main`.

La clase `R` representa a todos los recursos que se hayan creado dentro del proyecto y se genera automáticamente. Es una clase **NO editable** pues se genera y actualiza de manera automática.



```

/* AUTO-GENERATED FILE. DO NOT MODIFY.

package hola.que.tal;

public final class R {
    public static final class attr {
    }

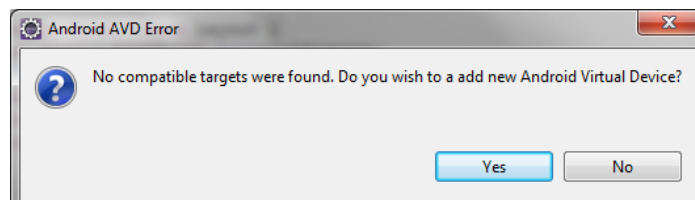
    public static final class drawable {
        public static final int icon=0x7f020000;
    }

    public static final class layout {
        public static final int main=0x7f030000;
    }

    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}

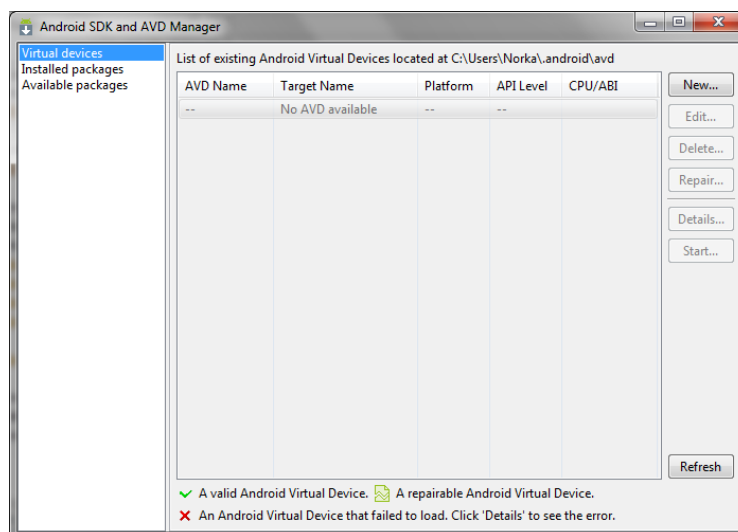
```

- f) Paso 6: Para ejecutar la aplicación se selecciona el proyecto y, desde el menú contextual, se selecciona Run As / **Android Application**. Es posible que la primera vez que se ejecute la aplicación se presente la siguiente pantalla:



Esto significa que no se ha configurado el emulador. En ese caso seleccionar el botón NO.

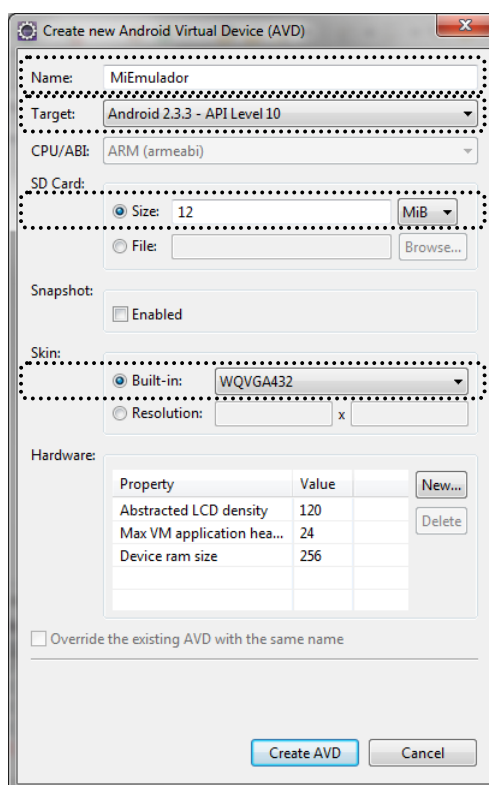
g) Paso 7: Configurar el emulador seleccionando del menú Window de Eclipse, la opción **Android SDK and AVD Manager**. Se visualizó la siguiente ventana:



h) Paso 8: Pulsar el botón “**New**” para crear un nuevo Emulador.

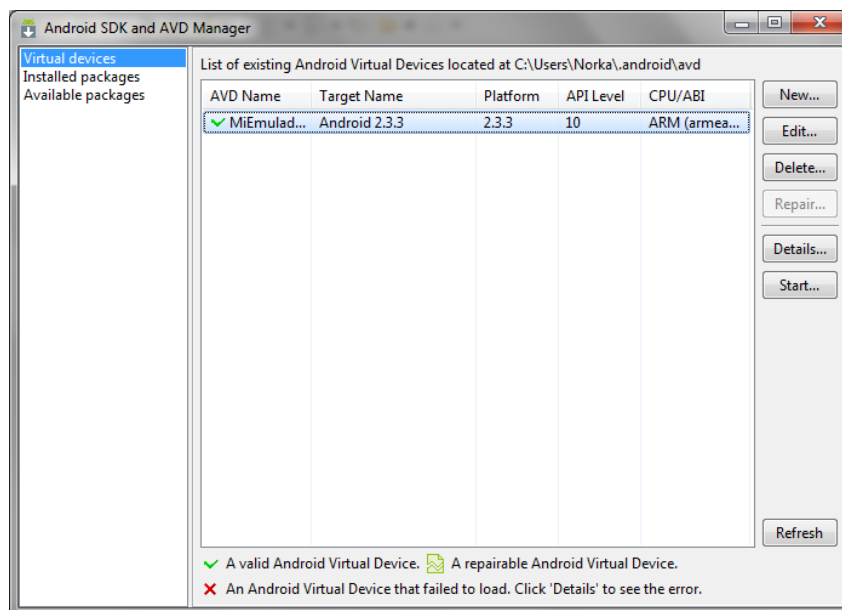
Se visualizará la siguiente pantalla. Ingresar los siguientes valores:

- 1) El nombre de nuestro Emulador
- 2) La versión de Android que representará el Emulador
- 3) El tamaño del SD Card.
- 4) La resolución del equipo a utilizar



i) Paso 9: Seleccionar el botón “Create AVD”.

Se visualizará la siguiente ventana con un “check” en color verde que indica que el Emulador creado es válido.



j) Paso 10: Cerrar la ventana y ejecutar nuevamente la aplicación. Se visualizará el **emulador** con la **aplicación** tal como se muestra en la siguiente ventana:



k) Paso 11: ¡Muy bien!, se ha culminado la implementación de la primera aplicación Android: ¡Hola Mundo!



5.2 Fundamentos de aplicaciones Android

5.2.1. El ciclo de vida de una aplicación Android

A diferencia de las aplicaciones tradicionales, las aplicaciones Android tienen un control limitado sobre su propio ciclo de vida. Los componentes de la aplicación deben escuchar si se genera algún cambio en el estado de la aplicación y reaccionar adecuadamente, teniendo un especial cuidado en caso se deba terminar de manera inesperada la ejecución de la aplicación.

Por defecto, cada aplicación Android se ejecuta en su propio proceso, cada uno de los cuales está ejecutándose en una instancia independiente de Dalvik. La administración de la memoria y el proceso es gestionada por el entorno de ejecución.

Android gestiona sus recursos de manera “agresiva”, haciendo lo necesario para que el dispositivo móvil permanezca activo. Esto significa que los procesos (y las aplicaciones que contengan) serán eliminados sin un aviso previo en algunos casos, para liberar recursos para aplicaciones de más alta prioridad, que generalmente están interactuando directamente con el usuario en un momento determinado. Esto es parte del proceso de priorización.

5.2.2. Prioridades y estados de una aplicación Android

El orden en el cual los procesos son eliminados para reclamar recursos, es determinado por la prioridad de las aplicaciones que alojen estos procesos.

Si dos aplicaciones tienen la misma prioridad, el proceso que haya tenido una prioridad más bajo durante un tiempo mayor será eliminado primero. La prioridad de un proceso es también afectada por las dependencias entre procesos; si una aplicación tiene una dependencia de un componente “Service” o “ContentProvider” proporcionados por una segunda aplicación, la aplicación secundaria tendrá una prioridad al menos tan alta como la prioridad de la aplicación que soporta.

Es importante estructurar nuestra aplicación correctamente para asegurar que su prioridad es la adecuada para el trabajo que está haciendo, de no hacerlo, nuestra aplicación podría ser eliminada mientras está en medio de una tarea importante.

Se muestra a continuación el cuadro de prioridades para determinar el orden de terminación de una aplicación.

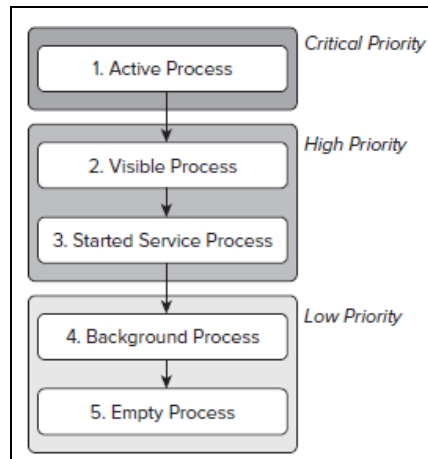


Figura 5.2

Active Process. Los procesos activos (en foreground) tienen componentes de aplicación interactuando con el usuario. Estos son los procesos que Android siempre tratará de mantener listos para reclamar cualquier recurso. Generalmente tendremos muy pocos de estos procesos y ellos serán eliminados solo como la última opción.

Como ejemplos de Active Process tenemos:

- ✓ **Activities** en un estado activo, es decir, aquellos que están en foreground respondiendo a los eventos del usuario.
- ✓ **Services** ejecutando los métodos onStart, onCreate o onDestroy
- ✓ **Broadcast Receivers** ejecutando el método onReceive.

Visible Process. Los procesos visibles pero inactivos son aquellos que están alojando Activities “visibles”. Los Activities visibles son aquellos que en efecto son visibles pero no están en el foreground o atendiendo los eventos del usuario. Esto sucede cuando un Activity está solo parcialmente oscurecida (por un Activity no full Screen o por un Activity transparente). Generalmente hay muy pocos procesos visibles, y éstos serán eliminados solo bajo circunstancias extremas para permitir a los procesos activos continuar su ejecución.

Started Service Process. Procesos que están alojando servicios que han sido iniciados. Los servicios soportan procesamiento que debería continuar sin una interface visible. Debido a que los servicios en background no interactúan directamente con el usuario, éstos reciben una prioridad ligeramente más baja que las actividades visibles. Estos servicios son considerados aún procesos foreground y no serán eliminados, a menos que requieran recursos los procesos activos o visibles.

Background Process. Son procesos que alojan actividades que no son visibles y que no tienen ningún servicio en ejecución. Generalmente tendremos un alto número de procesos en background que Android eliminará usando el patrón “last seen – first killed” para obtener recursos para procesos en foreground.

Empty Process. Para mejorar la performance general del sistema, Android generalmente retendrá en memoria una aplicación, después de que ésta ha alcanzado el final de su ciclo de vida. Android mantiene este “caché” para mejorar el tiempo de carga de las aplicaciones que son relanzadas.

5.2.3. Principales componentes: Introducción al uso de actividades y la gestión de recursos

5.2.3.1. Creando recursos

Los recursos en una aplicación Android son almacenados debajo de la carpeta `res/` dentro de la jerarquía de nuestro proyecto. En esta carpeta cada uno de los diferentes tipos de recursos disponibles es almacenado en una subcarpeta conteniendo estos recursos.

Creando recursos de valores simples. Los valores simples que soporta Android incluyen tipos tales como strings, color, dimension y arreglos de enteros y strings. Todos los valores simples son almacenados dentro de archivos xml en la carpeta `res/values`. En el ejemplo mostrado a continuación se muestran todos los tipos simples que podríamos usar dentro de Android.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">To Do List</string>
  <color name="app_background">#FF0000FF</color>
  <dimen name="default_border">5px</dimen>
  <array name="string_array">
    <item>Item 1</item>
    <item>Item 2</item>
    <item>Item 3</item>
  </array>
  <array name="integer_array">
    <item>3</item>
    <item>2</item>
    <item>1</item>
  </array>
</resources>
```

Figura 5.3

Layouts. Los recursos de tipo layout nos permiten desacoplar la capa de presentación de nuestra aplicación Android diseñando la distribución de nuestras interfaces de usuario en un archivo XML en vez de hacerlo directamente en el código.

El uso más común de un layout es para definir la interfaz de usuario de un Activity. Una vez definido en el archivo XML, el layout es “desplegado” dentro del activity usando el método `setContentView`.

5.2.3.2. Activities

Representan la capa de presentación de nuestra aplicación Android. Cada pantalla en la aplicación será una extensión de la clase Activity. Los Activities usan Views para crear interfaces de usuario que muestren información y respondan a las acciones del usuario. Haciendo una analogía con las aplicaciones de escritorio, un Activity sería el equivalente a un formulario.

5.2.3.3. Ciclo de vida de un Activity

Entender el ciclo de vida de un Activity es importante para asegurar que nuestra aplicación proporcione una adecuada experiencia al usuario así como también una correcta gestión de sus recursos.

Activity Stack. El estado de cada Activity es determinado por su posición en la pila de Activities, un collection de tipo “last in – first out”, de todos los Activities que se estén ejecutando actualmente. Cuando un nuevo Activity es iniciado, la ventana en primer plano (foreground) que tiene asociada es movida automáticamente a la parte más alta de la pila (top). Si por ejemplo el usuario retrocede usando el botón “Back”, o el Activity en primer plano es cerrado, el siguiente Activity en la pila se moverá hacia arriba convirtiéndose en el Activity activo. Se muestra el detalle de este comportamiento en la siguiente figura:

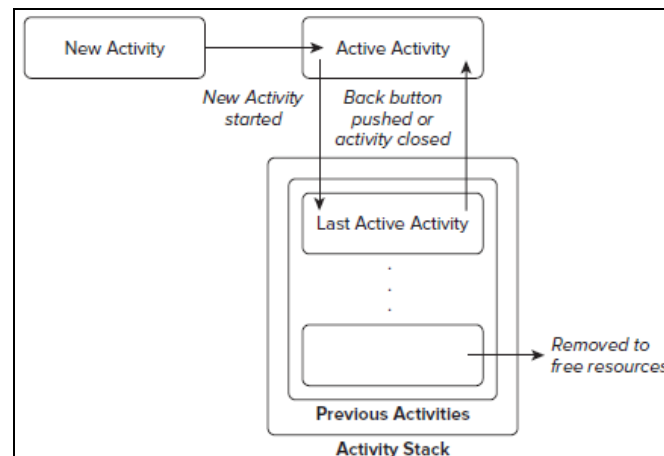


Figura 5.4

5.2.3.4. Estados de un Activity

Conforme los Activities son creados o destruidos, estos ingresan y salen de la pila de Activities tal como se muestra en la figura previa. Un Activity puede tener los siguientes estados:

Active. Cuando un Activity está en el top de la pila, este es visible, centrado, es un Activity en primer plano que está recibiendo el input del usuario. Android intentará mantenerlo vivo a todo costo, eliminando Activities que se encuentren más abajo en la pila de

ser necesario, para asegurar que el Activity activo tengo todos los recursos que necesite. Cuando otro Activity llegue a estar activo, este será automáticamente pausado.

Paused. En algunos casos, nuestro Activity será visible pero no tendrá foco, de ser este el caso, se encontrará en el estado pausado. Este estado es alcanzado si un Activity transparente o no “full screen” es activo en frente de nuestro Activity. Cuando un Activity ha sido pausado, es tratado todavía como si estuviera activo; no recibe eventos del usuario. En situaciones extremas, Android eliminará un Activity pausado, para recuperar recursos para el Activity activo. Cuando un Activity llega a estar totalmente obscurecido, este es detenido.

Stopped. Cuando un Activity no es visible, este es detenido. El Activity permanecerá en memoria, reteniendo toda su información de estado; sin embargo, es ahora un candidato para ser terminado cuando el sistema requiera memoria por algún motivo. Cuando un Activity está detenido, es importante guardar datos y el estado de la interface de usuario. Una vez que un Activity se cierra o se sale de él (closed or exited) este pasa al estado inactivo.

Inactive. Después que un Activity ha sido eliminado y antes de que sea puesto en marcha nuevamente, este se encontrará en estado inactivo. Los Activities inactivos han sido removidos de la pila de Activities y necesitan ser reiniciados antes de que puedan ser mostrados nuevamente y usados.

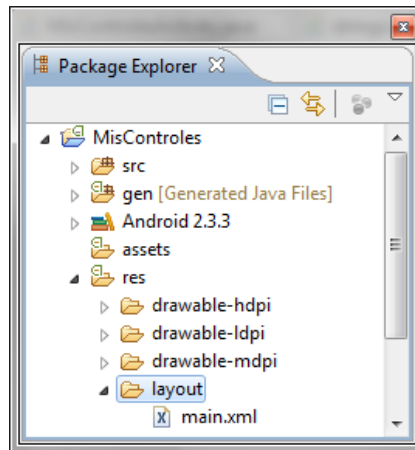
Las transiciones entre estados son no determinísticas y son gestionadas totalmente por el gestor de memoria Android. Android empezará cerrando aplicaciones que contengan Activities inactivos, seguidas por aquellas que tengan Activities en estado stopped. En casos extremos eliminará aquellas que estén pausadas.

Tip: es importante grabar todo el estado de la interface de usuario y hacer persistentes todos los datos cuando un Activity es pausado o detenido (paused or stopped).

5.2.3.5. Aplicación “Controles” Android

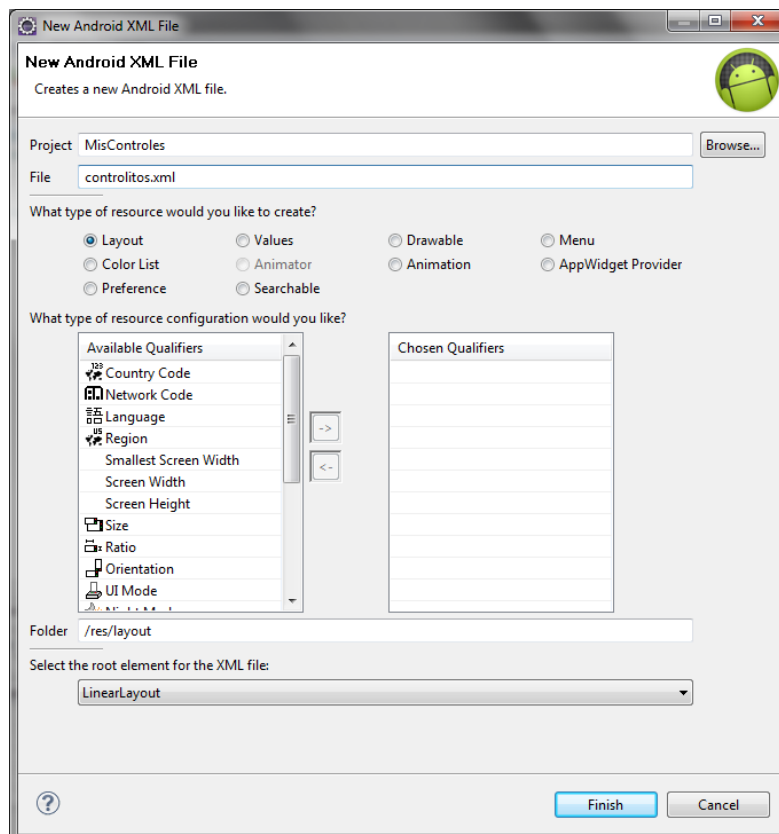
La funcionalidad a implementar toma como base la aplicación MisControles del ejercicio previo.

a) Paso 1: Seleccione la carpeta layout, haga clic derecho / new / **Android XML File**



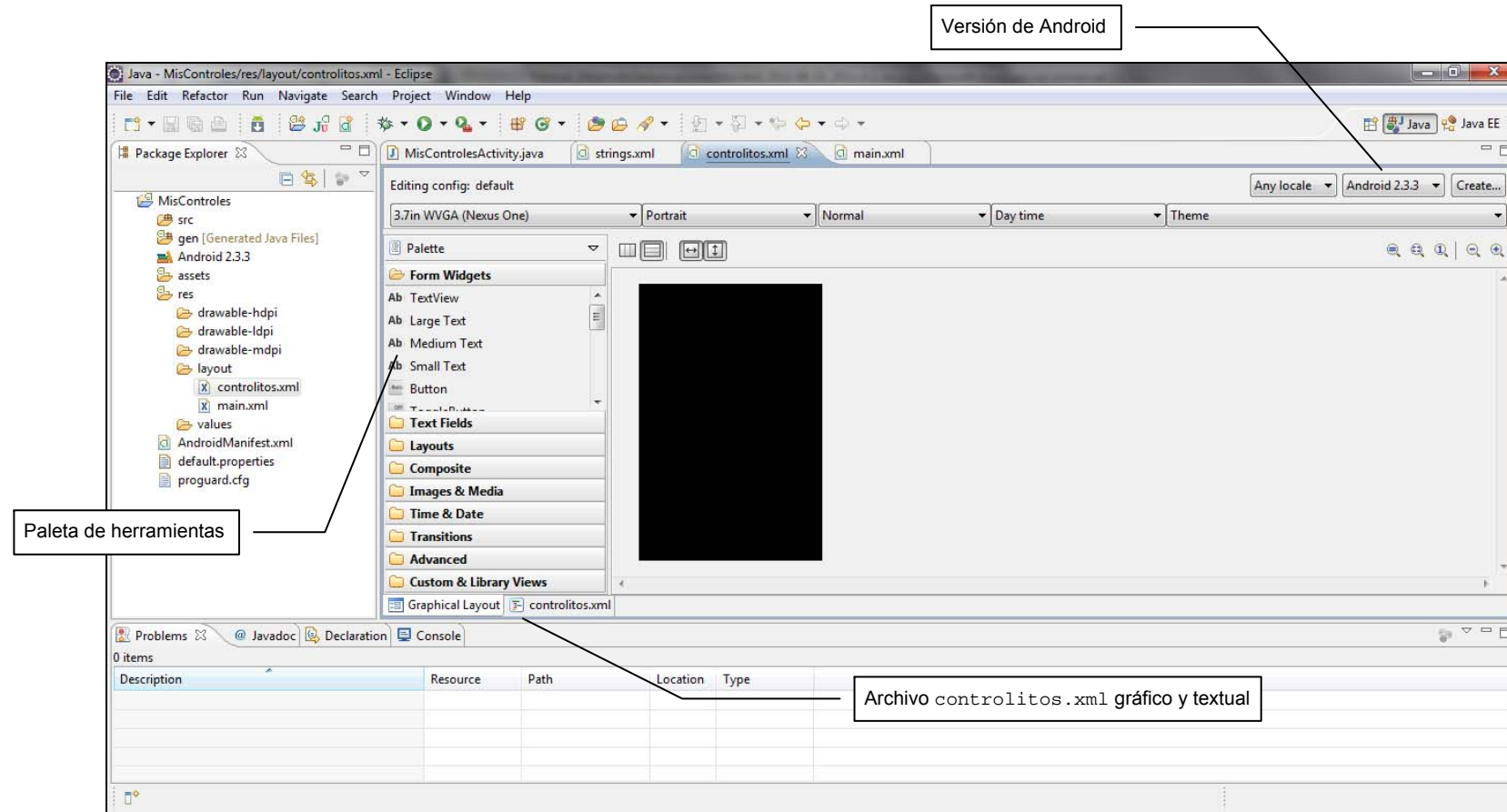
Importante.- Recuerde que dentro de la carpeta layout podremos crear diversas distribuciones (layouts) definidas a nivel de archivo xml que luego podrán ser referenciadas en cualquier clase de nuestra aplicación a través de **R.layout**. (La clase R de Android se actualiza automáticamente cada vez que creamos un nuevo recurso).

b) Paso 2: Visualizará la siguiente ventana. Ingrese en el campo File el nombre **controlitos.xml**



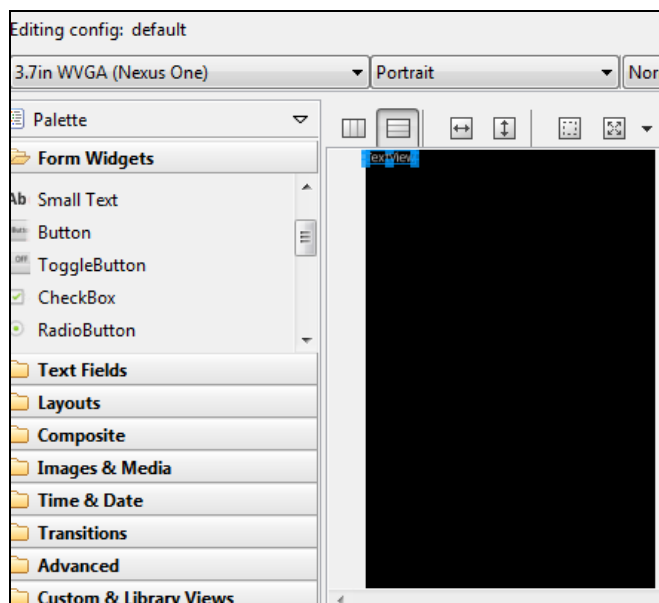
Debe notar que el tipo de recurso a crear es un **Layout** y que el elemento raíz a ser creado en el archivo XML será de tipo **LinearLayout**.

c) Paso 3: Seleccione el botón “Finish”. Visualizará una ventana similar a la siguiente. En ella podremos, entre otras características:



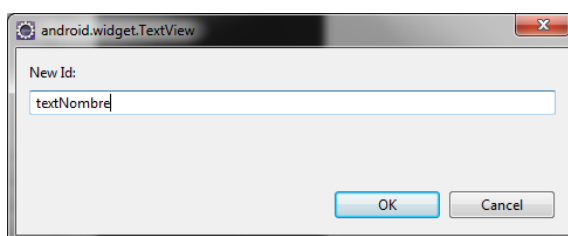
- d) Paso 4: Estando en modo gráfico, arrastremos a la zona de diseño un componente de tipo TextView.

El componente de tipo TextView se encuentra en el submenú **Form Widgets**. Visualizará una pantalla similar a la siguiente:



Ha creado su primer componente de tipo TextView.

- e) Paso 5: Ubíquese sobre el TextView haga clic derecho y seleccione la opción **Edit Id**:



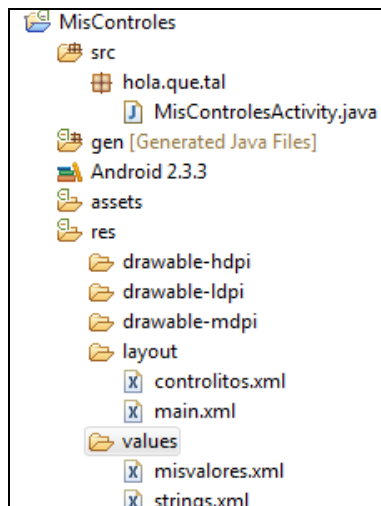
En la pantalla mostrada, puede modificar el nombre por defecto del control, ingrese **textNombre** y seleccione el botón **OK**.

- f) Paso 6: También puede modificar el contenido del texto a mostrar. Para ello, podríamos cambiar el texto por defecto, directamente en el archivo **controlitos.xml**.

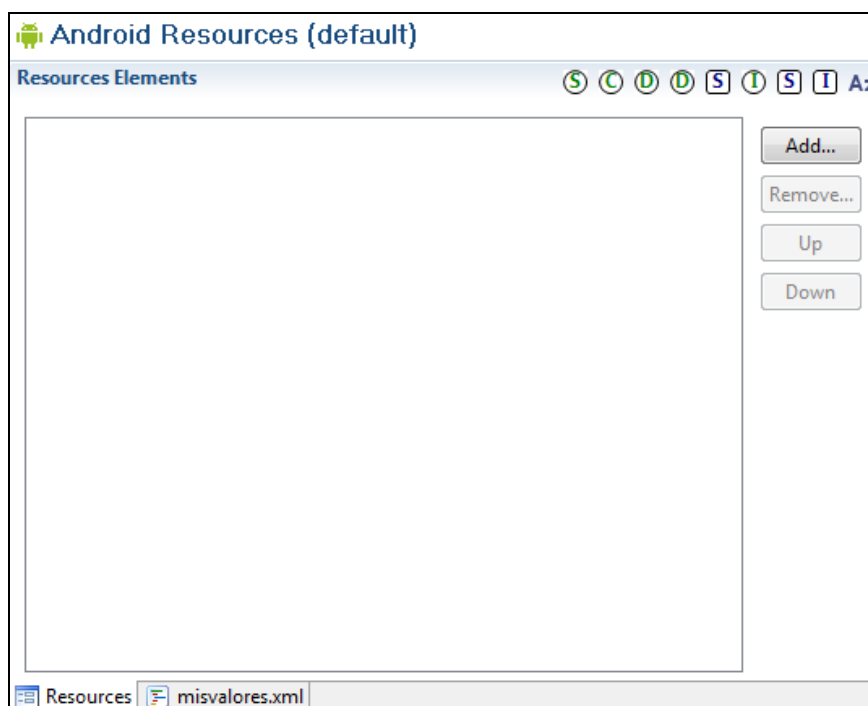


Sin embargo, y siguiendo las buenas prácticas de programación, haremos lo más configurable nuestra aplicación.

g) Paso 7: ubíquese en la carpeta **values** del proyecto y de manera similar al paso 1, cree el archivo **misvalores.xml**

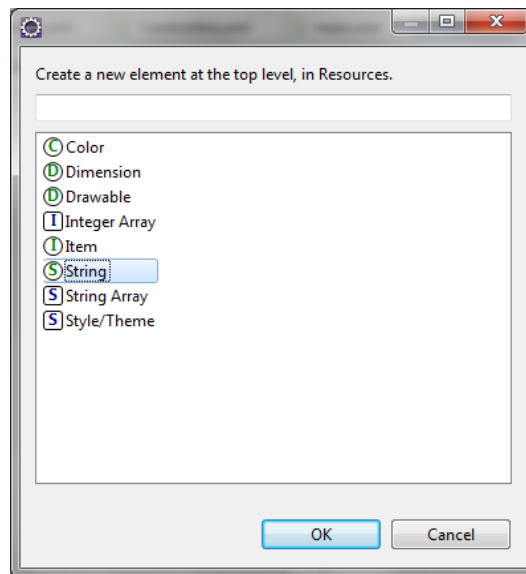


Una vez creado el archivo, visualizará una ventana similar a la siguiente:

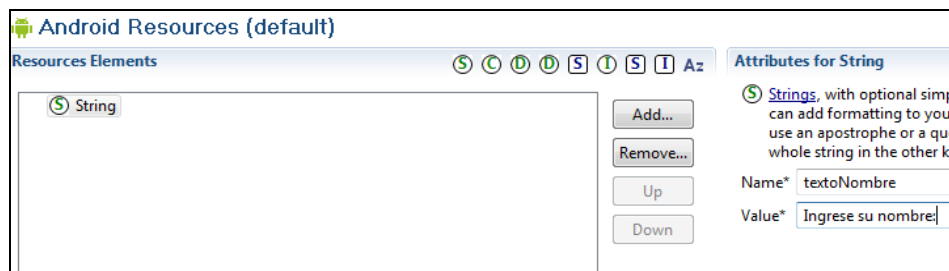


h) Paso 8: Tenemos acceso a la vista de diseño (denominada Resources) o la vista xml del archivo. Seleccionamos a continuación el botón “Add” para agregar un nuevo recurso.

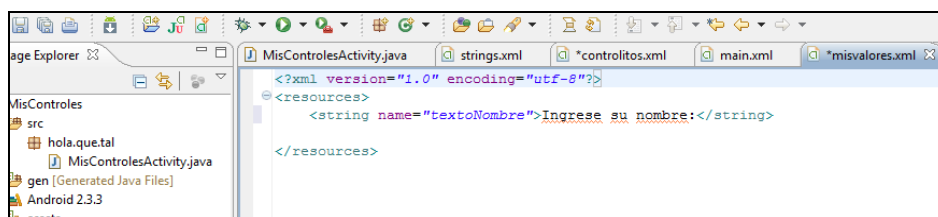
- i) Paso 9: Visualizamos la siguiente pantalla. En ella, seleccionamos el tipo de elemento a crear, para este ejemplo, seleccionaremos el tipo **String**. A continuación hacemos clic sobre el botón “Ok”.



En la pantalla mostrada a continuación, ingresamos el nombre del recurso y el valor asociado.

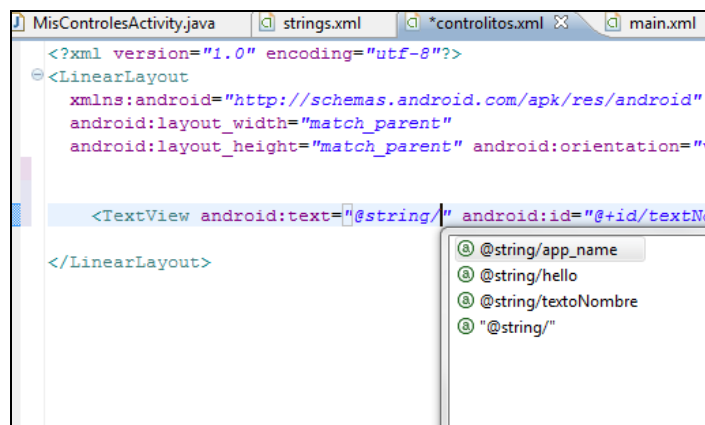


Si seleccionamos la vista xml visualizaremos el resultado de esta acción a nivel de código.

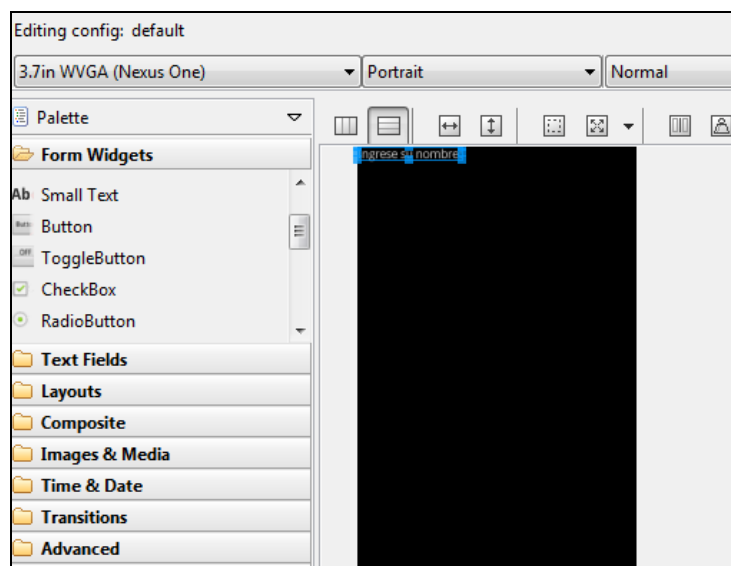


Confirmamos los cambios haciendo clic sobre el botón Guardar.

- j) Paso 10: En el archivo controlitos.xml, nos ubicamos en el atributo **android:text** del TextView y haciendo uso de la combinación de teclas CTRL – <barra espaciadora> obtenemos el menú contextual del cual seleccionaremos el recurso creado: **textoNombre**.



El resultado obtenido, lo podemos observar en la vista de diseño del archivo controlitos.xml



k) Paso 11: A continuación y haciendo uso de los siguientes controles:

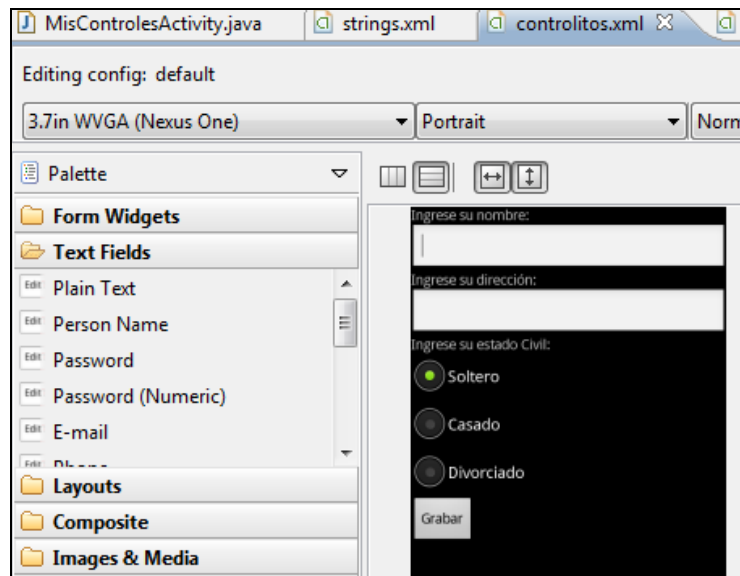
Submenú Form Widgets:

- ✓ TextView
- ✓ Button
- ✓ RadioGroup

Submenú TextFields

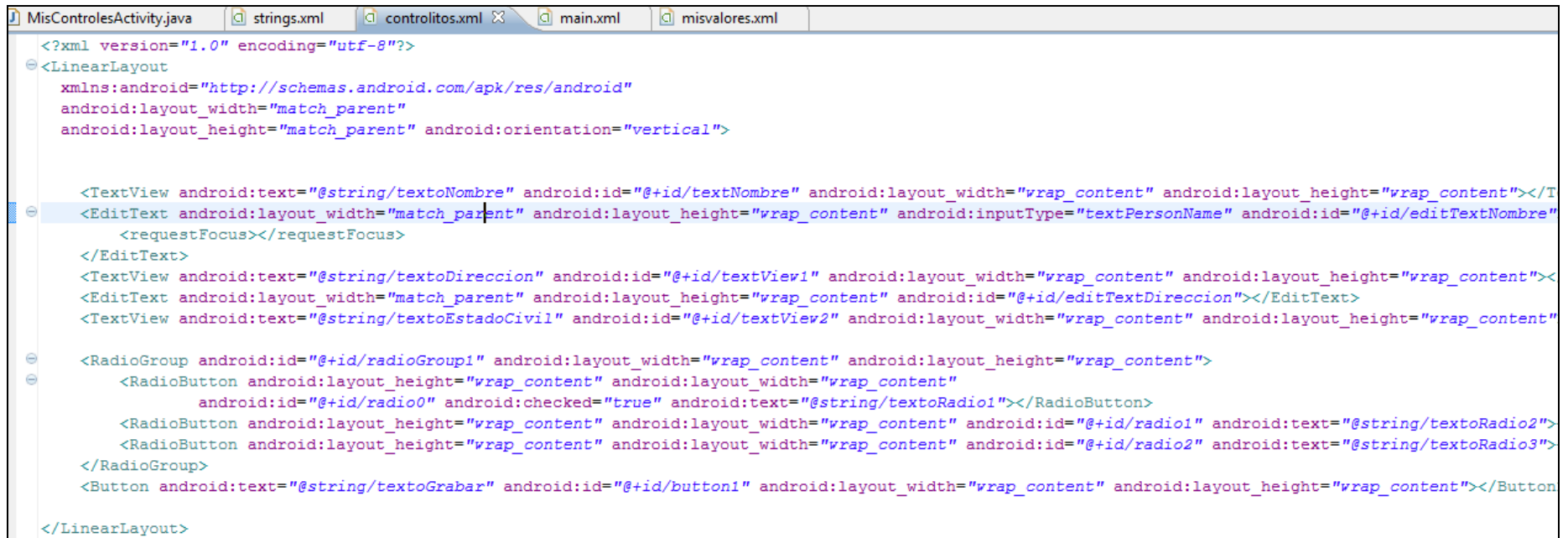
- ✓ Plain Text (EditText)
- ✓ Person Name (EditText)

Dibuje una interface similar a la mostrada en la siguiente pantalla:



Recuerde que debe crear los **recursos** respectivos en el archivo `misvalores.xml`. También es importante modifique los identificadores (`id`) por defecto de cada uno de los controles que cree en la vista de diseño.

Se muestra a continuación el archivo controlitos.xml en su vista xml para que pueda verificar cuáles son los cambios requeridos para implementar la interface seleccionada. Por ejemplo, los textos de cada radio button son recursos definidos en el archivo misvalores.xml



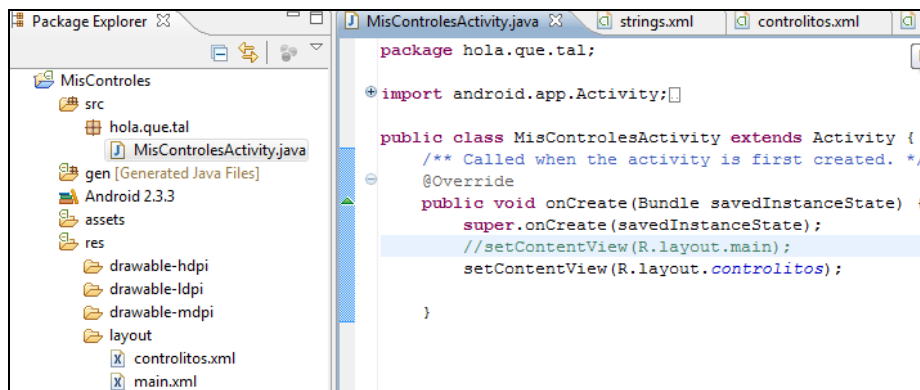
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" android:orientation="vertical">

    <TextView android:text="@string/textoNombre" android:id="@+id/textNombre" android:layout_width="wrap_content" android:layout_height="wrap_content"></T
    <EditText android:layout_width="match_parent" android:layout_height="wrap_content" android:inputType="textPersonName" android:id="@+id/editTextNombre"
        <requestFocus></requestFocus>
    </EditText>
    <TextView android:text="@string/textoDireccion" android:id="@+id/textView1" android:layout_width="wrap_content" android:layout_height="wrap_content"><
    <EditText android:layout_width="match_parent" android:layout_height="wrap_content" android:id="@+id/editTextDireccion"></EditText>
    <TextView android:text="@string/textoEstadoCivil" android:id="@+id/textView2" android:layout_width="wrap_content" android:layout_height="wrap_content"

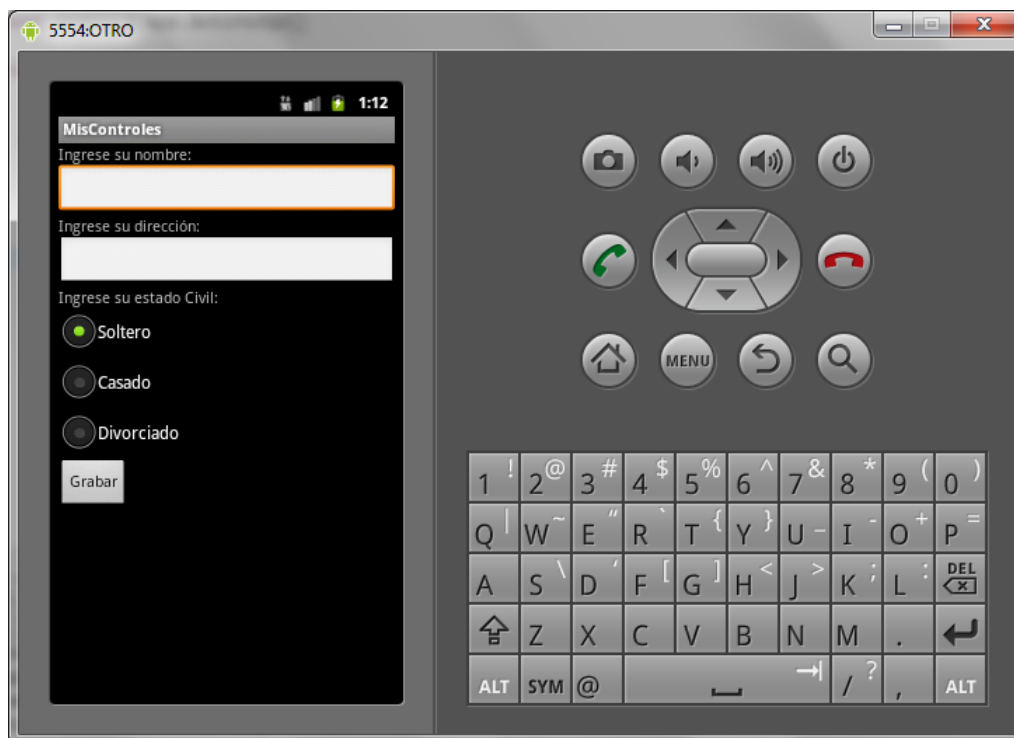
    <RadioGroup android:id="@+id/radioGroup1" android:layout_width="wrap_content" android:layout_height="wrap_content">
        <RadioButton android:layout_height="wrap_content" android:layout_width="wrap_content"
            android:id="@+id/radio0" android:checked="true" android:text="@string/textoRadio1"></RadioButton>
        <RadioButton android:layout_height="wrap_content" android:layout_width="wrap_content" android:id="@+id/radio1" android:text="@string/textoRadio2">
        <RadioButton android:layout_height="wrap_content" android:layout_width="wrap_content" android:id="@+id/radio2" android:text="@string/textoRadio3">
    </RadioGroup>
    <Button android:text="@string/textoGrabar" android:id="@+id/button1" android:layout_width="wrap_content" android:layout_height="wrap_content"></Button

</LinearLayout>
```

- l) Paso 12: Para poder ejecutar la aplicación, debe modificar primero el Activity **MisControlesActivity.java** y referenciar en el método **setContentView** al archivo **controlitos.xml**



Al ejecutar la aplicación visualizará una pantalla similar a la siguiente:



- m) Paso 13: ¡Excelente!, ha culminado de usar de manera básica la herramienta de diseño de interfaces visuales para Android



5.2.3.6. Ejercicio propuesto 1: modificando el “background de un TextView”

Debe modificar el color de fondo de los TextView: Ingrese su nombre” e “Información Adicional”:

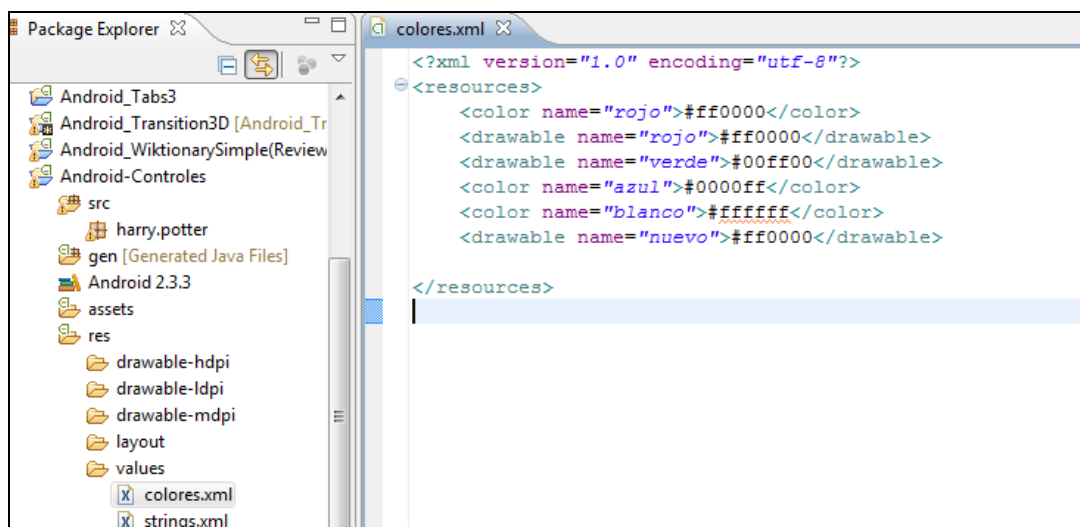


Para ello:

- a) Paso 1: Se recomienda usar objetos de tipo Drawable y no Color. Revise la documentación y determine por qué es más conveniente usar objetos de tipo Drawable:

<http://developer.android.com/guide/topics/resources/drawable-resource.html>

- b) Paso 2: Se recomienda cree un archivo de recursos, dentro del cual puede practicar con objetos de tipo color y drawable tal como se muestra a continuación:



- c) Paso 3: Finalmente, recuerde que los TextView deben referenciar al nuevo color de fondo que ha creado:

```
<TextView
    android:layout_height="wrap_content"
    android:text="@string/texto1"
    android:layout_width="match_parent"
    android:id="@+id/textViewMensaje1"
    android:textColor="@color/blanco"
    android:background="@drawable/rojo" >

</TextView>
```

- c) Paso 3: ¡Excelente!, debe haber culminado exitosamente su primer ejercicio propuesto.

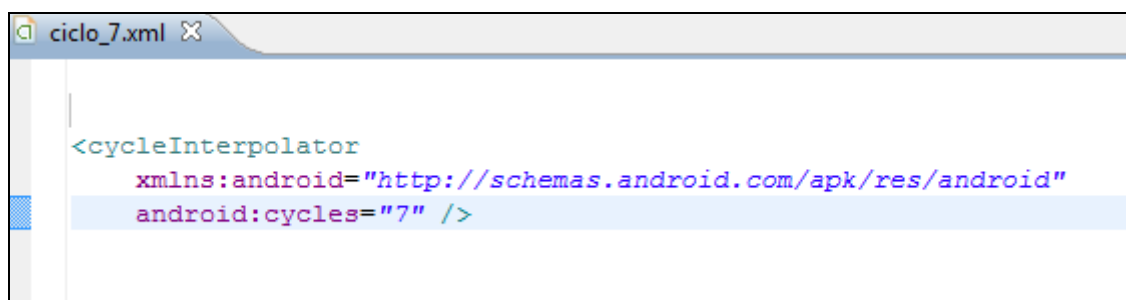
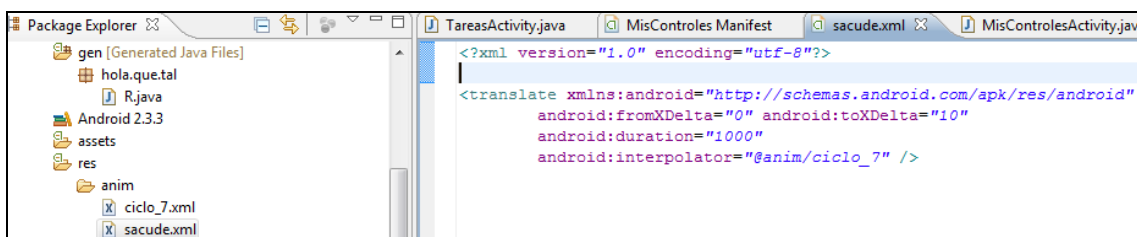


5.2.3.6. Ejercicio propuesto 2: animando la cajita de texto: “Ingrese su nombre”

Al hacer clic sobre el botón Grabar, la caja de Texto para ingresar el nombre “celebrará ☺” el ingreso, a través de un movimiento (shaking) horizontalmente 7 veces.

Para implementar la funcionalidad solicitada deberá:

- a) Paso 1: Crear objetos de tipo animación dentro de la carpeta res/anim de su proyecto. Se sugiere cree los siguientes archivos ciclo_7.xml y sacude.xml :



Se recomienda revisar el siguiente link <http://developer.android.com/reference/android/view/animation/package-summary.html> para consultar los principales conceptos de Animación y el uso de las etiquetas <translate> y <cycleInterpolator> de Android.

- b) Paso 2: Debe modificar la clase MisControlesActivity (la cual debe configurar como principal) para:
 - 1) Referenciar al botón “Grabar” de la interface y asociarle un listener
 - 2) Crear un objeto de tipo animación en base al recurso del mismo tipo creado en el archivo xml.
 - 3) Iniciar la animación
 - 4) Visualizar en consola el texto ingresado en la cajita de texto.

```

ciclo_7.xml  MisControlesActivity.java X
package hola.que.tal;
import android.app.Activity;

public class MisControlesActivity extends Activity implements OnClickListener {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // setContentView(R.layout.main);
        setContentView(R.layout.controlitos);

        // referenciamos al botón Grabar
        View botonGrabar = findViewById(R.id.buttonGrabar);
        botonGrabar.setOnClickListener( this );
    }

    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        Animation animacion = AnimationUtils.loadAnimation(this,R.anim.sacude);
        EditText cajita=(EditText)findViewById(R.id.editTextNombre);
        cajita.startAnimation(animacion);

        // visualizamos el texto digitado en la cajita
        System.out.println("El texto ingresado es: "+cajita.getText());
    }
}

```

LogCat				
Log				
Time	pid	tag	Message	
08-07 22:50...	I 43	ActivityM...	Start proc hola.que.tal for activity hola.que.tal/.MisControlesAct	
08-07 22:50...	I 781	jdwp	Ignoring second debugger -- accepting and dropping	
08-07 22:50...	I 43	ActivityM...	Displayed hola.que.tal/.MisControlesActivity: +3s745ms	
08-07 22:50...	D 266	dalvikvm	GC_EXPLICIT freed 10K, 55% free 2595K/5703K, external 716K/1038K,	
08-07 22:50...	W 781	KeyCharac...	No keyboard for id 0	
08-07 22:50...	W 781	KeyCharac...	Using default keymap: /system/usr/keychars/qwerty.kcm.bin	
08-07 22:50...	D 122	dalvikvm	GC_EXPLICIT freed 102K, 50% free 2985K/5895K, external 1942K/2440K	
08-07 22:50...	I 781	System.out	El texto ingresado es: arriba siempre arriba	

c) Paso 3: ¡Excelente!, debe haber culminado exitosamente su segundo ejercicio propuesto.



5.3 Interfaces de usuario

5.3.1. Fundamentos de diseño y creación de componentes básicos Android

En lo que respecta al diseño de interfaces de usuario, Android introduce nuevos componentes y terminología, la cual se detalla a continuación:

Views. Los Views son las clases base para todos los elementos de interfase visuales (comúnmente conocidos como controles o widgets). Todos los controles de interface de usuario, incluyendo las clases layout, son derivadas de la clase View.

View Groups. Son extensiones de la clase View que pueden contener múltiples Views hijas. Heredamos de la clase ViewGroup para crear controles compuestos integrados por Views hijas interconectadas. La clase ViewGroup es también heredada para ayudar a los layout managers a diseñar controles dentro de un Activity.

Activites. Representan la ventana o pantalla que se está visualizando. Los Activities son el equivalente Android de los Forms de una aplicación java Desktop. Para visualizar una interface de usuario asignamos un View (usualmente un layout) a un Activity.

5.3.1.1. Creando creando interfaces de usuario de tipo Activity con View.

Un nuevo Activity es incializado siempre con una pantalla en blanco sobre la cual alojamos una interface de usuario en particular. Para asignar la interface de usuario invocamos al método setContentView, el cual acepta como parámetro el identificador de un recurso de tipo Layout o una instancia de la clase View. Esto nos permite definir la interface de usuario a partir de código o usando la mejor práctica correspondiente a referenciar un recurso externo de tipo Layout.

El uso de recursos externos de tipo Layout, nos permite desacoplar la capa de presentación de la lógica de nuestra aplicación, proporcionándonos la flexibilidad para cambiar la presentación sin cambiar la lógica de la aplicación. Esto nos permite especificar diferentes Layouts optimizados para diferentes configuraciones de hardware.

Se muestra a continuación un ejemplo en el cual se asocia a un Activity una interface de usuario, utilizando un recurso externo de tipo Layout:

```
public class MisControlesActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);
        setContentView(R.layout.controlitos);
    }

}
```

También es posible crear y asociar una interface de usuario a un Activity a través de código:

```
public class MisControlesActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {

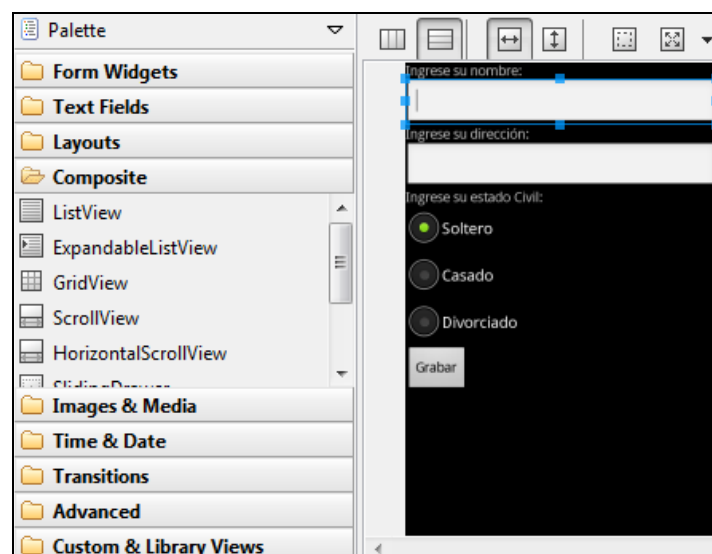
        super.onCreate(savedInstanceState);

        TextView miTextView = new TextView(this);
        setContentView(miTextView);
        miTextView.setText("Hola mundo, que fácil es ANDROID");
    }

}
```

5.3.1.2. La caja de herramientas “Android Widget”

Android nos proporciona una caja de herramientas con Views estándar para ayudarnos a crear interfaces básicas. El **Plugin de Android para Eclipse** nos permite utilizarlas fácilmente cuando nos encontramos en el modo de edición de recurso externo tipo Layout.



A continuación se describen algunas de las Views más utilizadas:

- ✓ **TextView.** Es una etiqueta de texto estándar de solo lectura. Soporta la visualización de líneas múltiples, formateo de strings, entre otras características.
- ✓ **EditText.** Es una caja de texto editable. Acepta múltiples líneas y la característica de texto autocompletado.
- ✓ **ListView.** Es un View Group que crea y gestiona una lista vertical de Views, mostrándolos como filas dentro de la lista. El ListView más simple muestra una lista con valores de tipo string.
- ✓ **spinner.** Es un control compuesto que muestra un TextView y un ListView asociado que nos permite seleccionar un ítem de una lista para ser mostrado en una caja de texto.

5.3.1.3. Layouts

Los Layout managers (conocidos normalmente como Layouts) son extensiones de la clase ViewGroup usadas para posicionar controles hijos de una interface de usuario. Los Layouts pueden estar anidados, lo cual nos permite crear interfaces complejas usando una combinación de Layouts.

A continuación se describen los Layouts más importantes dentro de la plataforma Android:

FrameLayout.- Es el más simple de los LayoutManagers. Este Layout ubica cada View hijo que agreguemos, en la esquina superior izquierda. De esta manera se crea una pila de múltiples Views (conforme sigamos agregando estos componentes), obscureciendo las que estén debajo de la que esté en la cima de la pila.

LinearLayout.- Este Layout alinea cada View hijo que se agregue en una línea vertical u horizontal. Un Layout Vertical tiene una columna de Views, mientras que un Layout horizontal tiene una fila de Views.

RelativeLayout.- Es el más flexible de todos los Layouts nativos. Este Layout permite definir la posición de cada View hijo que se agregue de manera relativa a los otros Views y a los límites de la pantalla.

TableLayout.- Este Layout permite distribuir Views usando una grilla de filas y columnas.

Gallery.- Un Layout de este tipo muestra una fila simple de ítems en una lista de scroll horizontal.

5.3.2. Creación de vistas personalizadas, controles y gestión de eventos

Android nos permite heredar de vistas existentes, ensamblar controles a partir de otros y crear vistas totalmente nuevas. Estas características posibilitan crear interfaces muy atractivas y optimizadas para nuestros flujos de trabajo.

Cuando diseñamos interfaces de usuario es importante balancear los aspectos estéticos y la usabilidad de nuestros componentes. Dado que es posible crear nuestros propios controles, podríamos estar tentados de crearlos desde cero, sin embargo, debemos tomar en cuenta que las Views estándar serán familiares para los usuarios de otras aplicaciones Android y éstas además son actualizadas automáticamente con las mejoras de la versión de la plataforma. Por otro lado, en pantallas pequeñas, en las que el usuario presta poca atención a los detalles, la familiaridad puede brindar una mayor usabilidad que el tener un control único y demasiado elaborado.

5.3.2.1. Modificando vistas existentes

La caja de herramientas de Android nos proporciona Views que satisfacen muchos de los requerimientos de interfaces de usuario existentes, sin embargo, estos controles han sido creados necesariamente genéricos. Al personalizar estas vistas básicas, eliminamos la necesidad de recodificar el comportamiento existente de las mismas, al mismo tiempo que podemos adaptar la interfaz de usuario y la funcionalidad de acuerdo a nuestras necesidades existentes.

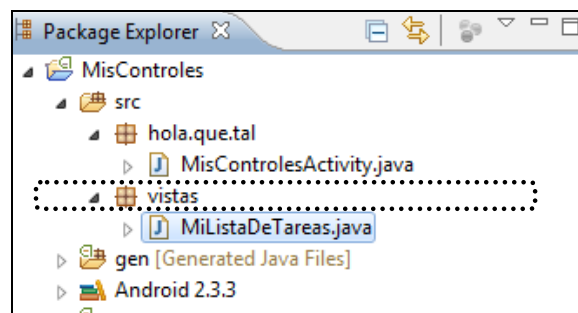
Para crear un nuevo View basados en un control existente, debemos necesariamente heredar de éste. A continuación implementaremos un ejemplo de aplicación.

5.3.2.2. Implementando el componente: “Mi lista de tareas”

Por defecto, este componente usa controles de tipo `TextView` para representar cada fila en un `ListView`. Personalizaremos a continuación la apariencia de la lista heredando de `TextView` y sobre escribiendo el método `onDraw`.

- a) Paso 1: creamos la clase `MiListaDeTareas` que hereda de `TextView`. Sobreescribiremos el método `onDraw` e implementaremos constructores que invoquen a un nuevo método `init`.

A continuación se muestra el paquete dentro del cual debe crear la clase `MiListaDeTareas` y el detalle del código de la clase a implementar:



Nota.- Nótese cómo se invoca a la funcionalidad padre del método `onDraw` con la palabra clave `super`.

```
package vistas;

import android.content.Context;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.widget.TextView;

public class MiListaDeTareas extends TextView {

    public MiListaDeTareas(Context context, AttributeSet ats, int ds) {
        super(context, ats, ds);
        init();
    }

    public MiListaDeTareas(Context context) {
        super(context);
        init();
    }

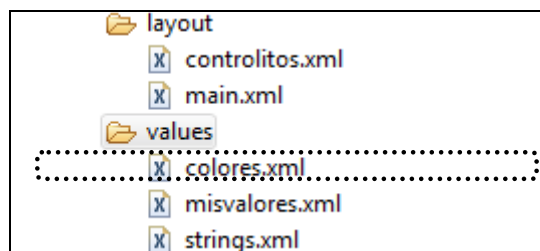
    public MiListaDeTareas(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    private void init() {

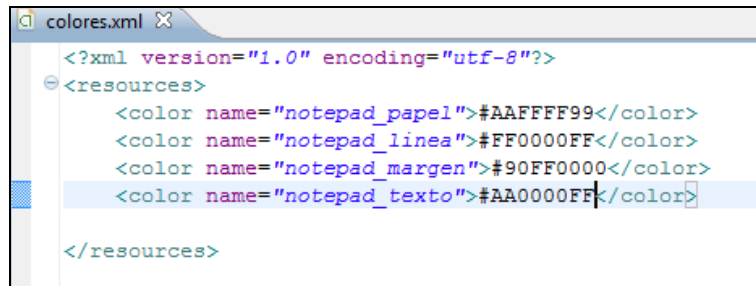
    }

    @Override
    public void onDraw(Canvas canvas) {
        // Usamos el TextView base para dibujar el texto
        super.onDraw(canvas);
    }
}
```

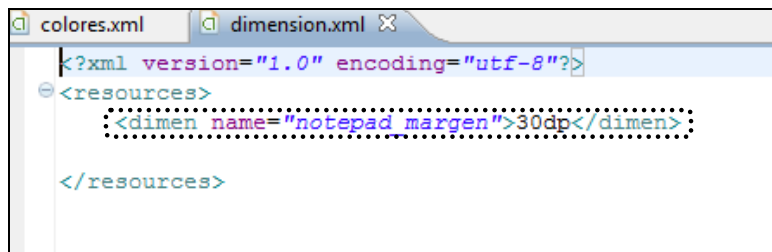
- b) Paso 2: creamos un nuevo archivo de recursos: `colores.xml`, dentro de la carpeta `res/values`. Crearemos colores para el “papel”, “margen”, “línea” y “texto”



Se muestra a continuación el detalle de colores creados en la vista xml del archivo de recursos:



- c) Paso 3: creamos un nuevo archivo de recursos: dimension.xml, y agregamos un valor para el ancho del margen del papel.



- d) Paso 4: Dado que ya hemos definido los recursos necesarios, estamos en condiciones de personalizar la apariencia del control MiListaDeTareas. Creamos a continuación variables de instancia privadas para almacenar objetos de tipo Paint que usaremos para dibujar el fondo (*background*) del papel y el ancho del margen.

```
private void init() {
    // Referencia a los recursos disponibles para la vista
    Resources myResources = getResources();

    // Crear los pinceles que se usaran en el metodo onDraw
    marginPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    marginPaint.setColor(myResources.getColor(R.color.notepad_margen));

    linePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    linePaint.setColor(myResources.getColor(R.color.notepad_linea));

    // Obtener el background del papel y el ancho del margen
    paperColor = myResources.getColor(R.color.notepad_papel);
    margin = myResources.getDimension(R.dimen.notepad_margen);
}
```

- e) Paso 5: para dibujar el papel sobreescribimos el método onDraw y dibujamos la imagen usando los objetos de tipo Paint que hemos creado. Una vez que hemos dibujado el papel, invocamos al método onDraw de la clase padre y dibujamos el texto tal como normalmente se hace.


```

@Override
public void onDraw(Canvas canvas) {
    // Asignar el color del papel
    canvas.drawColor(paperColor);

    // Dibujar líneas rectas
    canvas.drawLine(0, 0, getMeasuredHeight(), 0, linePaint);
    canvas.drawLine(0, getMeasuredHeight(),
        getMeasuredWidth(), getMeasuredHeight(), linePaint);

    // Dibujar el margen
    canvas.drawLine(margin, 0, margin, getMeasuredHeight(), marginPaint);

    // Mover el texto a través del margen
    canvas.save();
    canvas.translate(margin, 0);

    // Usar la funcionalidad estándar del TextView para dibujar el texto
    super.onDraw(canvas);

    canvas.restore();
}

```

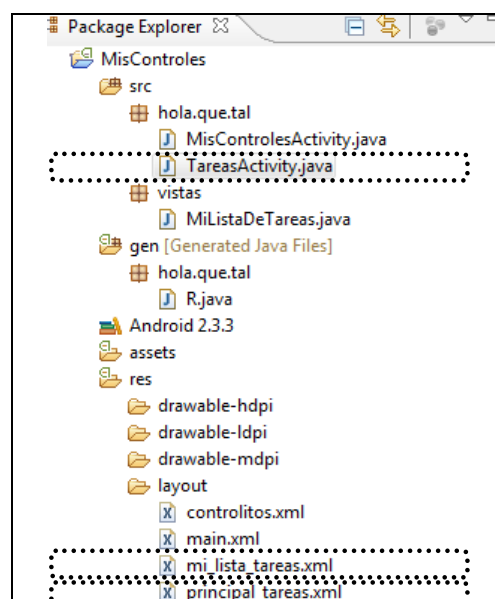
f) Paso 6: Para usar el control **MiListaDeTareas** debemos incluirlo en un archivo de recursos de tipo Layout. Creamos el archivo `mi_lista_tareas.xml` en la carpeta `res/layout`.

```

<?xml version="1.0" encoding="utf-8"?>
<vistas.MiListaDeTareas
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:scrollbars="vertical"
    android:textColor="@color/notepad_texto"
    android:fadingEdge="vertical"
/>

```

g) Paso 7: A continuación creamos un nuevo Activity: **TareasActivity**, el cual referenciará a `principal_tareas.xml` y utilizará el `ListView` personalizado `mi_lista_tareas.xml`



Se muestra a continuación la vista xml del archivo principal_tareas.xml, en él se observan los componentes visuales básicos necesarios para este ejercicio: EditText y ListView:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editTextTarea">
    </EditText>

    <ListView
        android:id="@+id/miListView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    </ListView>

</LinearLayout>
```

Dado que se iniciará la ejecución de la aplicación a partir del Activity TareasActivity, debe registrarlo en el archivo de configuración principal de Android como el Activity principal. Esto se logra con la etiqueta:

```
<action android:name="android.intent.action.MAIN" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="hola.que.ta1"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="10" />

    <application android:icon = "@drawable/icon" android:label="@string/app_name">
        <activity
            android:name = ".MisControlesActivity"
            android:label = "@string/app_name">
        </activity>

        <activity
            android:name = "TareasActivity">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

    </application>
</manifest>
```

h) Paso 8: Debe implementar el siguiente código dentro de la clase PrincipalActivity:

```

package hola.que.tal;

import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnKeyListener;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;

public class TareasActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.principal_tareas);

        // Referencia al listview creado en principal_tareas
        ListView miListView = (ListView) findViewById(R.id.miListView);
        // Referencia a la caja de texto
        final EditText editTextTarea = (EditText) findViewById(R.id.editTextTarea);

        // Crear un arraylist para almacenar los items de listview
        final ArrayList<String> todoItems = new ArrayList<String>();

        // Referencia al listview personalizado
        int resID = R.layout.mi_lista_tareas;

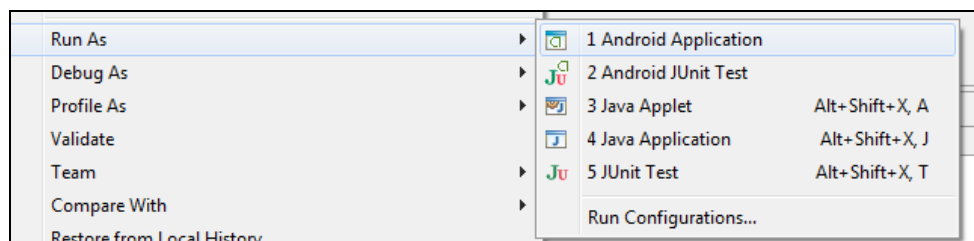
        // Asociar el listview personalizado con el arraylist
        final ArrayAdapter<String> aa = new ArrayAdapter<String>(this, resID,
            todoItems);



        // La asociacion se refleja en el listview de principal_tareas
        miListView.setAdapter(aa);

        editTextTarea.setOnKeyListener(new OnKeyListener() {
            public boolean onKey(View v, int keyCode, KeyEvent event) {
                if (event.getAction() == KeyEvent.ACTION_DOWN) {
                    if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
                        todoItems.add(0, editTextTarea.getText().toString());
                        editTextTarea.setText("");
                        aa.notifyDataSetChanged();
                        return true;
                    }
                }
                return false;
            }
        });
    }
}

```

i) Paso 9: Finalmente ejecutamos la aplicación como una **Android Application**.



- j) Paso 10: Visualizaremos una pantalla similar a la siguiente. Ingresamos un texto en EditText  y seleccionamos el botón central DPAD  para ingresar el texto a la lista:



- k) Paso 11: El resultado final es el que se muestra en la siguiente pantalla:



- l) Paso 12: ¡Misión cumplida!, se ha creado exitosamente un componente personalizado en Android: un **ListView** personalizado.



5.4 Componente Intent

Los Intents son utilizados como un mecanismo para pasar mensajes, tanto al interior de una aplicación como entre aplicaciones. Los Intents pueden ser usados para:

- ✓ Declarar la “intención” de que un Service o Activity será inicializado para ejecutar una acción, usualmente con o sobre una pieza de dato en particular.
- ✓ Transmitir que ha ocurrido un evento
- ✓ Iniciar de manera explícita un Service o Activity

Uno de los usos más comunes para los Intents es iniciar nuevas Actividades, sea explícitamente (especificando la clase que se debe cargar) o implícitamente (solicitando que una acción sea ejecutada sobre un pieza de datos en particular). En el último caso, la acción necesitada no será ejecutada por un Activity dentro de la aplicación.

5.4.1. Definición y características: Uso de Intents para gestionar Activites

Uno de los usos más comunes de un Intent es relacionar componentes dentro de una aplicación. Los Intents son usados para iniciar y generar transiciones entre Activities.

Para abrir un Activity, invocamos al método `startActivity` pasándole un Intent, tal como se muestra a continuación:

```
startActivity(miIntent);
```

El Intent podría explícitamente especificar la clase de tipo Activity a abrir, o incluir una acción que un Activity deberá ejecutar. En este último caso, el “run time” escogerá un Activity dinámicamente usando un proceso conocido como “Intent Resolution”.

El método `startActivity` encuentra e inicia el Activity que mejor coincida con su Intent.

5.4.1.1. Iniciando nuevos Activities de manera explícita

Para iniciar una clase Activity de manera explícita, debemos crear un nuevo Intent, especificar el contexto de la aplicación actual, y la clase Activity a iniciar. Se muestra a continuación un ejemplo de lo indicado:

```
Intent elIntent = new Intent(MiActivity.class,  
                             ElOtroActivity.class);  
  
startActivity(elIntent);
```

Después de que el método `startActivity` es invocado, el nuevo Activity (en el ejemplo, **ElOtroActivity**), será creado y llegará a ser visible y activo, moviéndose a la parte superior de la pila de Activities.

Al invocar al método `finish` del nuevo Activity, o presionar el botón “back” del dispositivo móvil, se cerrará el Activity y será removido de la pila. De Manera alternativa, programáticamente podemos navegar hacia el Activity anterior o cualquier otro Activity, invocando al método `startActivity`.

5.4.1.2. Iniciando un Activity de manera implícita.

Un Intent implícito es un mecanismo que permite a componentes anónimos de la aplicación “atender” solicitudes de acciones específicas.

Esto significa que podríamos preguntarle al sistema para lanzar un Activity que pueda ejecutar una acción específica sin conocer explícitamente cuál es la aplicación, o Activity que la hará.

Cuando construimos un Intent implícito a ser usado con el método `startActivity`, debemos identificar una acción a ser ejecutada y opcionalmente proporcionar un URI de referencia a los datos necesarios para ejecutar la acción. Podemos también enviar información adicional agregando información a través de componentes “extras” al Intent.

Por ejemplo, para permitirle al usuario hacer una llamada desde nuestra aplicación, podríamos implementar un nuevo objeto “Dialer” o usar un Intent implícito que solicite la acción “marcar” (dialing) un número telefónico (input que representaríamos con un URI) tal como se muestra a continuación:

```
Intent intento = new Intent(Intent.ACTION_DIAL,
                           Uri.parse("tel:511-11111111"));

startActivity(intento);
```

En los casos en que múltiples Activities puedan ejecutar la acción solicitada, se le brindará al usuario la posibilidad de seleccionar el componente de su elección.

5.4.1.3. Acciones Android nativas

A continuación se muestran algunas acciones nativas de Android, las cuales son representadas a través de constantes estáticas de tipo String en la clase Intent. Cuando creamos Intents implícitos, podemos usar estas acciones llamadas “Activity Intent” para para iniciar Activities dentro de nuestra aplicación.

ACTION_DELETE. Inicia un Activity que nos permite eliminar la data especificada en el URI del Intent.

ACTION_EDIT. Solicita un Activity que pueda editar la data especificada en el URI del Intent.

ACTION_INSERT. Invoca a un Activity capaz de insertar nuevos Items en el cursor especificado en el URI del Intent.

ACTION_VIEW. Es la acción genérica más común de todas. Se solicita visualizar la data proporcionada en el URI del Intent de la mejor manera posible. Diferentes aplicaciones gestionarán esta solicitud, dependiendo del esquema de datos proporcionado en el URI del Intent. Por ejemplo, direcciones “http” serán visualizadas en un browser, números telefónicos serán marcados utilizando un componente “dialer”, direcciones de geo localización, serán mostradas utilizando probablemente una aplicación de google maps.

5.4.1.4. Usando filtros para atender Intents implícitos

Si un Intent está solicitando realizar una acción sobre unos datos en particular, ¿cómo sabe Android que componente invocar para atender esta solicitud? Llegan al rescate “los filtros Intent”.

Los filtros Intent nos permiten registrar actividades para ejecutar alguna acción sobre datos específicos.

El uso de filtros Intent, permiten a cualquier componente “anunciar” que son capaces de responder a una acción solicitada desde cualquier aplicación instalada en el dispositivo móvil.

Para registrar un componente como un potencial gestor de Intents, debemos agregar la etiqueta `<intent-filter>` al registro del componente en el archivo de configuración de Android. Se deben utilizar los siguientes tags (y sus atributos asociados) dentro del nodo `<intent-filter>`.

action. Utilizamos el atributo `android:name` para especificar el nombre de la acción a ser atendida. Cada Intent Filter puede tener sólo una etiqueta `action`. Las acciones deben ser strings **no repetidos** (únicos) y autodescriptivos. La mejor práctica es un usar un sistema de nombres basado en la convención de nombres de **paquetes** java.

category. Usamos el atributo `android:name` para indicar bajo que circunstancias la acción será atendida. Cada etiqueta Intent Filter puede incluir múltiples etiquetas `category`. Podemos usar nuestras propias categorías o usar valores estándar proporcionados por Android tal como se muestran a continuación:

ALTERNATIVE. Esta categoría especifica que la acción estará disponible como una alternativa a la acción por defecto ejecutada sobre un ítem de este tipo.

DEFAULT. Permite que un componente sea la acción por defecto para el tipo de dato especificado en el Intent Filter. Esto es necesario para Activities que son ejecutados a través de Intents implícitos.

LAUNCHER. El uso de esta categoría hace que un Activity aparezca en el “application launcher” de la aplicación.

Existen otros valores estándar que podrá consultar en la página oficial de Android:

<http://developer.android.com/index.html>

data. Esta etiqueta permite especificar qué tipos de datos podrá utilizar nuestro componente. Es posible incluyamos varias etiquetas de tipo data en atención a nuestras necesidades. Podemos usar cualquier combinación de los siguientes atributos para especificar los datos que nuestro componente soportará: android:host, android:mimeType, android:path, android:port, android:scheme.

A continuación se muestra un ejemplo del uso de la etiqueta Intent Filter.

```
<activity
    android:name="NoMeRindoActivity">

    <intent-filter>

        <action android:name="hola.que.tal.NoMeRindoActivity" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="hola.que.tal.datos/*" />

    </intent-filter>

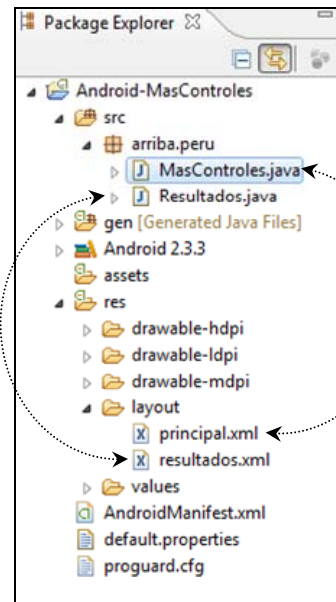
</activity>
```

5.4.1.5. Ejercicio de aplicación: Invocando Activities con Intents

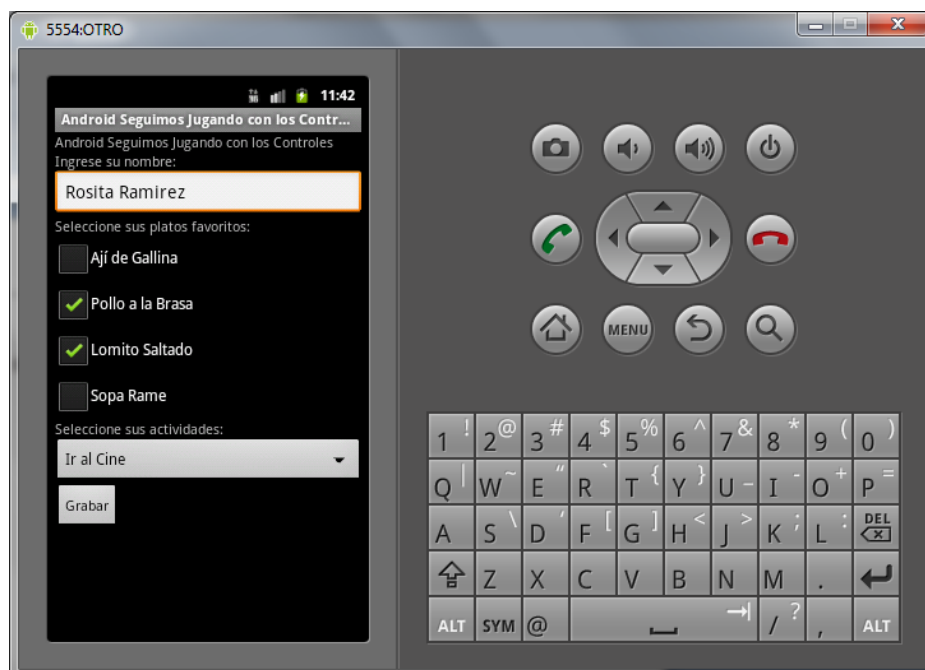
A continuación se describen los pasos que debe seguir para implementar el ejercicio propuesto:

a) Paso 1: Cree un nuevo proyecto con la siguiente estructura:

Dos Activities (1) con sus respectivos archivos de recursos de tipo layout (2).



b) Paso 2: Cree una interface tal como se muestra en la siguiente pantalla:



El archivo de recursos que contiene la interface es `principal.xml`. Se muestra a continuación su vista xml:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/app_name">
    </TextView>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textViewNombre"
        android:text="@string/textoCajita">
    </TextView>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:id="@+id/editTextNombre">
        <requestFocus></requestFocus>
    </EditText>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textViewCheckBox"
        android:text="@string/textoCheckBox">
    </TextView>

    <CheckBox
        android:text="Ají de Gallina"
        android:id="@+id/checkBox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </CheckBox>

    <CheckBox
        android:text="Pollo a la Brasa"
        android:id="@+id/checkBox2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </CheckBox>

    <CheckBox
        android:text="Lomito Saltado"
        android:id="@+id/checkBox3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </CheckBox>

    <CheckBox
        android:text="Sopa Rame"
        android:id="@+id/checkBox3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </CheckBox>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView1"
        android:text="@string/textoCombrito">
    </TextView>

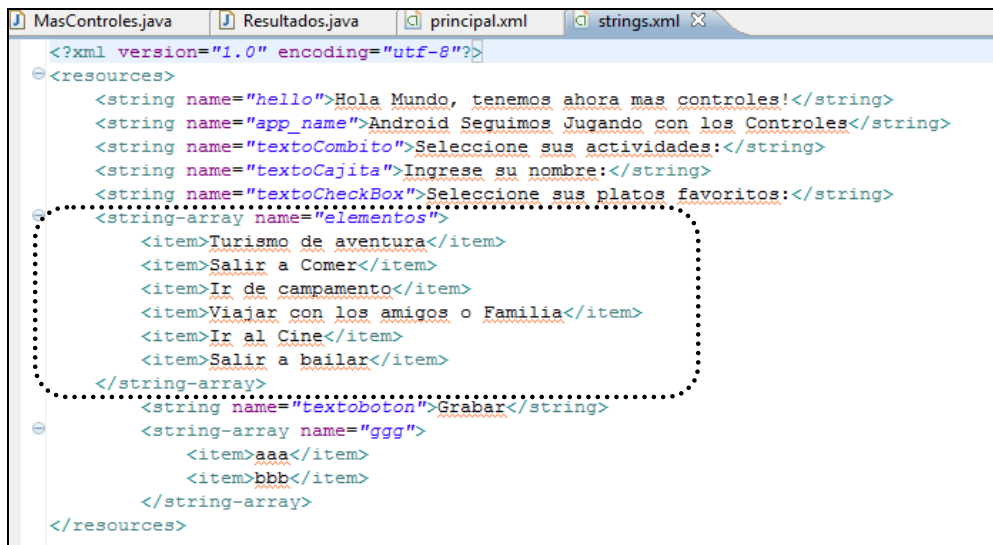
    <Button
        android:id="@+id/buttonGrabar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textoboton">
    </Button>

```

```
<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/spinnerElementos"
    android:entries="@array/elementos">
</Spinner>

</LinearLayout>
```

Nota.- El control Spinner, que es semejante a un cuadro combinado, tiene como elementos un arreglo de cadenas denominado elementos. Esta especificación se hace mediante la etiqueta **android:entries**. La lista ha sido definida en el archivo de recursos strings.xml el cual se muestra a continuación:



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hola Mundo, tenemos ahora mas controles!</string>
    <string name="app_name">Android Seguimos Jugando con los Controles</string>
    <string name="textoComboto">Seleccione sus actividades:</string>
    <string name="textoCajita">Ingrese su nombre:</string>
    <string name="textoCheckBox">Seleccione sus platos favoritos:</string>
    <string-array name="elementos">
        <item>Turismo de aventura</item>
        <item>Salir a Comer</item>
        <item>Ir de campamento</item>
        <item>Viajar con los amigos o Familia</item>
        <item>Ir al Cine</item>
        <item>Salir a bailar</item>
    </string-array>
    <string name="textoboton">Grabar</string>
    <string-array name="ggg">
        <item>aaa</item>
        <item>bbb</item>
    </string-array>
</resources>
```

- c) Paso 3: Debe implementar la lógica necesaria en la clase MasControles de modo que pueda:
- Recuperar los datos ingresados por el usuario
 - Visualizarlos en la consola de Android
 - Enviar datos hacia otro Activity a través de un Intent

```
public class MasControles extends Activity {

    EditText cajitaNombre=null;
    CheckBox check1 = null;
    CheckBox check2 = null;
    CheckBox check3 = null;
    Spinner preferencias = null;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.principal);

        // Referencia a los componentes de input
        cajitaNombre = (EditText)findViewById(R.id.editTextNombre);
        check1 = (CheckBox)findViewById(R.id.checkBox1);
        check2 = (CheckBox)findViewById(R.id.checkBox2);
        check3 = (CheckBox)findViewById(R.id.checkBox3);
        preferencias = (Spinner)findViewById(R.id.spinnerElementos);

        // Referencia al boton del Activity
        Button botonGrabar = (Button)findViewById(R.id.buttonGrabar);
        botonGrabar.setOnClickListener(new Apoyo());
    }

    // Crear una inner class
    private class Apoyo implements OnClickListener{

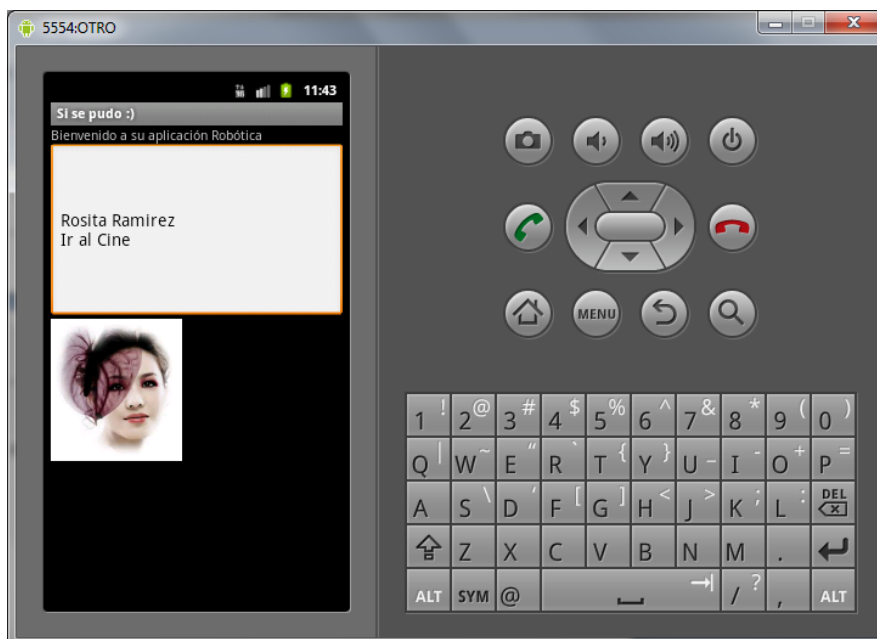
        @Override
        public void onClick(View v) {
            // Visualizar la informacion ingresada/seleccionada por el usuario
            System.out.println("Nombre: " + cajitaNombre.getText());
            System.out.println("Actividad seleccionada:" +
                preferencias.getSelectedItem());
            System.out.println("platos favoritos:");

            if(check1.isChecked()){ System.out.println(check1.getText()); }
            if(check2.isChecked()){ System.out.println(check2.getText()); }
            if(check3.isChecked()){ System.out.println(check3.getText()); }

            // Invocar a otro activity usando el método startActivity
            Bundle datos = new Bundle();
            datos.putString("nombre", cajitaNombre.getText()+"");
            datos.putString("actividad", preferencias.getSelectedItem()+"");

            // Crear el intent que invocará al activity Resultado
            Intent irAResultado = new Intent(MasControles.this, Resultados.class);
            // Cargar el intent con información adicional
            irAResultado.putExtras(datos);
            startActivity(irAResultado);
        }
    }
}
```

- d) Paso 4: Seleccionar el botón “Grabar” y se visualizará la siguiente pantalla:



Esta pantalla y los datos visualizados son dibujados por el **Activity** Resultados, el cual referencia al archivo de recursos resultados.xml mostrado a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Bienvenido a su aplicación Robótica"
            android:id="@+id/textViewTitulo">
        </TextView>

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textMultiLine"
            android:lines="8" android:id="@+id/editTextTodo">
            <requestFocus></requestFocus>
        </EditText>

        <ImageView
            android:src="@drawable/rostrom1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imageViewFoto">
        </ImageView>

    </LinearLayout>
</ScrollView>
```

El código que debe implementar en la clase Resultados es:

```
public class Resultados extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);

        // Recuperar los datos que ingreso el usuario en principal y
        // cargarlos en los campos del layout resultado sabiendo
        // que esta llegando un bundle

        Intent elIntent= getIntent();
        Bundle losExtras=elIntent.getExtras();
        String vnombre =(String)losExtras.get("nombre");
        String vactividad =(String)losExtras.get("actividad");

        setContentView(R.layout.resultados);

        EditText todo=(EditText)findViewById(R.id.editTextTodo);
        todo.append(vnombre+"\n");
        todo.append(vactividad);
    }
}
```

e) Paso 5: ¡Misión cumplida!, se ha invocado exitosamente a un **Activity** a través de un **Intent**



UNIDAD DE
APRENDIZAJE

6

ANDROID: PERSISTENCIA DE DATOS EN ANDROID

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno elabora aplicaciones que obtienen y almacenan diversos tipos de recursos utilizando Android SQLite.

TEMARIO

6.1 Tema 10 : Persistencia de datos: Android SQLite

6.1.1. : Introducción a SQLite: cursores y contenedores de valores.

Gestión de bases de datos con SQLite. Uso del componente SQLiteOpenHelper.

ACTIVIDADES PROPUESTAS

- Los alumnos implementan ejercicios que involucran los principales componentes de persistencia SQLite de Android.

6.1 Persistencia de datos: Android SQLite

La persistencia de datos estructurados en Android es proporcionada a través de los siguientes mecanismos:

Bases de datos SQLite. Esta es la mejor opción para datos estructurados dentro de nuestra aplicación. Cada aplicación puede crear su propia base de datos sobre la cual tendrá el control absoluto.

Content Providers. Este es un mecanismo genérico, que a través de una interface bien definida nos permite usar y compartir datos.

Normalmente, se utiliza SQLite para almacenar y gestionar datos estructurados y complejos de nuestra aplicación. Las bases de datos de Android son almacenadas físicamente en la carpeta `/data/data/<package_name>/databases` de nuestro dispositivo móvil o emulador. Por defecto, todas las bases de datos son privadas, solo accesibles a la aplicación que las creó.

6.1.1. Introducción a SQLite: cursores y contenedores de valores.

Gestión de bases de datos con SQLite. Uso del componente SQLiteOpenHelper

6.1.1.1. SQLite

SQLite es un gestor de base de datos relacional que cuenta con las siguientes características básicas:

- ✓ Open Source
- ✓ Ligero
- ✓ Cumple con los estándares
- ✓ Single tier

Ha sido implementado como una librería en lenguaje C bastante compacta que es incluida como parte del software stack de Android. Al haber sido implementada como una librería, cada base de datos de SQLite, es una parte integral de la aplicación que la crea, de esta manera, se eliminan las dependencias externas, se minimiza la latencia, simplificándose también los procesos de sincronización y bloqueo de transacciones.

Dada la alta confiabilidad y reputación ganada por este gestor de base de datos, ha sido seleccionada para ser el gestor de datos de muchos tipos de dispositivos móviles tales como MP3s players, iphone, Ipod Touch, etc.

6.1.1.2. Cursores y contenedores de valores

Los “ContentValues” son usados para insertar nuevas filas en una tabla. Cada objeto de tipo ContentValues representa una fila de una tabla. Las consultas en Android son retornadas como objetos de tipo Cursor.

Los cursores incluyen diversos métodos para navegación tales como:

- ✓ moveToFirst. Mueve el cursor a la primera fila del resultado obtenido de la consulta.
- ✓ moveToNext. Mueve el cursor al siguiente registro.
- ✓ getColumnIndex. Retorna el índice de la columna del índice especificado.
- ✓ getCount. Retorna el número de filas en el result set.

El método startManagingCursor integra el ciclo de vida del cursor en la invocación de un Activity. Cuando terminamos de usar un cursor, debemos invocar al método stopManagingCursor.

6.1.1.3. Ejercicio: creando una clase adaptadora genérica para base de datos

Es una buena práctica usar una clase de tipo Helper para simplificar nuestra interacción con la base de datos. A continuación debe implementar el siguiente código, el cual permitirá abrir, crear y actualizar una base de datos.

```

import android.content.ContentValues;
import android.content.Context;
import android.database.*;
import android.database.sqlite.*;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.util.Log;

public class MyDBAdapter {
    private static final String DATABASE_NAME = "myDatabase.db";
    private static final String DATABASE_TABLE = "mainTable";
    private static final int DATABASE_VERSION = 1;

    // The index (key) column name for use in where clauses.
    public static final String KEY_ID = "_id";

    // The name and column index of each column in your database.
    public static final String KEY_NAME = "name";
    public static final int NAME_COLUMN = 1;
    // TODO: Create public field for each column in your table.
    // SQL Statement to create a new database.
    private static final String DATABASE_CREATE = "create table " + DATABASE_TABLE +
        " (" + KEY_ID +
        " integer primary key autoincrement, " +
        KEY_NAME + " text not null);";

    // Variable to hold the database instance
    private SQLiteDatabase db;
    // Context of the application using the database.
    private final Context context;
    // Database open/upgrade helper
    private myDbHelper dbHelper;

    public MyDBAdapter(Context _context) {
        context = _context;
        dbHelper = new myDbHelper(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public MyDBAdapter open() throws SQLException {
        db = dbHelper.getWritableDatabase();
        return this;
    }

    public void close() {
        db.close();
    }

    public long insertEntry(MyObject _myObject) {
        ContentValues contentValues = new ContentValues();
        // TODO fill in ContentValues to represent the new row
        return db.insert(DATABASE_TABLE, null, contentValues);
    }

    public boolean removeEntry(long _rowIndex) {
        return db.delete(DATABASE_TABLE, KEY_ID + "=" + _rowIndex, null) > 0;
    }

    public Cursor getAllEntries () {
        return db.query(DATABASE_TABLE, new String[] {KEY_ID, KEY_NAME}, null, null,
            null, null, null);
    }

    public MyObject getEntry(long _rowIndex) {
        MyObject objectInstance = new MyObject();
        // TODO Return a cursor to a row from the database and
        // use the values to populate an instance of MyObject
        return objectInstance;
    }

    public int updateEntry(long _rowIndex, MyObject _myObject) {
        String where = KEY_ID + " = " + _rowIndex;
        ContentValues contentValues = new ContentValues();
        // TODO Fill in the ContentValues based on the new object
        return db.update(DATABASE_TABLE, contentValues, where, null);
    }
}

```

```

private static class myDbHelper extends SQLiteOpenHelper {

    public myDbHelper(Context context, String name, CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    // Se invoca cuando la base de datos no existe y se necesita crear una nueva
    @Override
    public void onCreate(SQLiteDatabase _db) {
        _db.execSQL(DATABASE_CREATE);
    }

    // Se invoca cuando la versión de la base no coincide y debe ser actualizada
    @Override
    public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
        // Log the version upgrade.
        Log.w("TaskDBAdapter", "Upgrading from version " +
            _oldVersion + " to " +
            _newVersion + ", which will destroy all old data");

        // Upgrade the existing database to conform to the new version. Multiple
        // previous versions can be handled by comparing _oldVersion and _newVersion
        // values.

        // Este ejemplo elimina la base antigua y crea una nueva
        _db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
        // Crear la nueva
        onCreate(_db);
    }
}

/** Dummy object to allow class to compile */
static class MyObject {
}

```

6.1.1.4. Clase SQLiteOpenHelper

SQLiteOpenHelper es una clase abstracta usada para implementar las mejores prácticas de para crear, abrir y actualizar una base de datos. Al usar objetos de este tipo, podemos ocultar la lógica usada para decidir si una base de datos necesita ser creada, abierta o actualizada. La mejor práctica para crear una base de datos SQLite es creando una subclase de SQLiteOpenHelper y sobre escribiendo el método onCreate, dentro del cual colocaremos la lógica necesaria para crear las tablas en base de datos.

```

public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
        KEY_WORD + " TEXT, " +
        KEY_DEFINITION + " TEXT)";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}

```

UNIDAD DE
APRENDIZAJE

7

ANDROID: AUDIO, VIDEO Y USO DE DISPOSITIVOS AUDIOVISUALES

LOGRO DE LA UNIDAD DE APRENDIZAJE

- 3 Al término de la unidad, el alumno elabora aplicaciones visuales que gestionan recursos de Audio y Video dentro de la plataforma Android..

TEMARIO

7.1 Tema 12 : Audio, Video y Uso de Dispositivos visuales

- 7.1.1. : Componente Media Player. Definición, características y principales aplicaciones.
- 7.1.2. : Visualización de videos usando la vista Video.
- 7.1.3. : Grabación de audio y video: Uso de Intents para grabar video.

ACTIVIDADES PROPUESTAS

- Los alumnos implementan ejercicios que involucran el uso de los componentes MediaPlayer y VideoView

7.1 Audio, video y uso de dispositivos visuales

7.1.1. Componente Media Player: definición, características y aplicaciones

La reproducción de Multimedia en Android es gestionada por la clase MediaPlayer. Podemos ejecutar diversos tipos de medios multimedia almacenados en archivos locales, o como recursos de la aplicación, así como flujos de un URL de red como de otros medios. En cada caso el formato del archivo y el tipo de multimedia a ser ejecutado es abstraído del desarrollador.

La gestión de audio y video con la clase MediaPlayer es como gestionar una máquina que puede asumir difentes estados, tales como los que se muestran a continuación:

- ✓ Inicializar el Media Player con “multimedia” a ser ejecutada
- ✓ Preparar el Media Player para la reproducción
- ✓ Iniciar la reproducción
- ✓ Pausar o detener la reproducción antes de que esta termine
- ✓ Reproducción completada exitosamente

Se muestra a continuación un ejemplo para probar Audio con el componente Media Player:

```
import android.app.Activity;
import android.widget.LinearLayout;
import android.os.Bundle;
import android.os.Environment;
import android.view.ViewGroup;
import android.widget.Button;
import android.view.View;
import android.content.Context;
import android.util.Log;
import android.media.MediaRecorder;
import android.media.MediaPlayer;
import java.io.IOException;

public class AudioRecordTest extends Activity{
    private static final String LOG_TAG = "AudioRecordTest";
    private static String mFileName = null;

    private RecordButton mRecordButton = null;
    private MediaRecorder mRecorder = null;
    private PlayButton mPlayButton = null;
    private MediaPlayer mPlayer = null;

    private void onRecord(boolean start) {
        if (start) {
            startRecording();
        } else {
            stopRecording();
        }
    }

    private void onPlay(boolean start) {
        if (start) {
            startPlaying();
        } else {
            stopPlaying();
        }
    }
}
```

```

private void startPlaying() {
    mPlayer = new MediaPlayer();
    try {
        mPlayer.setDataSource(mFileName);
        mPlayer.prepare();
        mPlayer.start();
    } catch (IOException e) {
        Log.e(LOG_TAG, "prepare() failed");
    }
}

private void stopPlaying() {
    mPlayer.release();
    mPlayer = null;
}

private void startRecording() {
    mRecorder = new MediaRecorder();
    mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mRecorder.setOutputFile(mFileName);
    mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

    try {
        mRecorder.prepare();
    } catch (IOException e) {
        Log.e(LOG_TAG, "prepare() failed");
    }

    mRecorder.start();
}

private void stopRecording() {
    mRecorder.stop();
    mRecorder.release();
    mRecorder = null;
}

class RecordButton extends Button {
    boolean mStartRecording = true;

    OnClickListener clicker = new OnClickListener() {
        public void onClick(View v) {
            onRecord(mStartRecording);
            setText(((mStartRecording)? "Stop": "Start") + " recording");
            mStartRecording = !mStartRecording;
        }
    };

    public RecordButton(Context ctx) {
        super(ctx);
        setText("Start recording");
        setOnClickListener(clicker);
    }
}

class PlayButton extends Button {
    boolean mStartPlaying = true;

    OnClickListener clicker = new OnClickListener() {
        public void onClick(View v) {
            onPlay(mStartPlaying);
            setText(((mStartPlaying)? "Stop": "Start") + " playing");
            mStartPlaying = !mStartPlaying;
        }
    };

    public PlayButton(Context ctx) {
        super(ctx);
        setText("Start playing");
        setOnClickListener(clicker);
    }
}

```



```

public AudioRecordTest() {
    mFileName = Environment.getExternalStorageDirectory().getAbsolutePath();
    mFileName += "/audiorecordtest.3gp";
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    LinearLayout ll = new LinearLayout(this);
    mRecordButton = new RecordButton(this);
    ll.addView(mRecordButton,
        new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT,
            0));
    mPlayButton = new PlayButton(this);
    ll.addView(mPlayButton,
        new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT,
            0));
    setContentView(ll);
}

@Override
public void onPause() {
    super.onPause();
    if (mRecorder != null) {
        mRecorder.release();
        mRecorder = null;
    }

    if (mPlayer != null) {
        mPlayer.release();
        mPlayer = null;
    }
}
}

```

7.1.2 Visualización de videos usando la vista Video

La manera más simple de reproducir video es usando el control `VideoView`. Este control incluye una superficie sobre la cual el video es visualizado. El control encapsula un objeto `Media Player` que gestiona la reproducción del video.

Para asignar un video a reproducir, solo es necesario que invoquemos al método `setVideoPath` o `setVideoUri` para especificar el path a un archivo local o el URI de un `Content Provider` o un flujo de video remoto:

- ✓ `streamingVideoView.setVideoUri("http://www.mysite.com/videos/myvideo.3gp");`
- ✓ `localVideoView.setVideoPath("/sdcard/test2.3gp");`

A continuación se muestra un ejemplo de aplicación del componente `ViewVideo`:

```

import android.app.Activity;
import android.os.Bundle;
import android.widget.MediaController;
import android.widget.Toast;
import android.widget.VideoView;

public class VideoViewDemo extends Activity {

    /**
     * TODO: Set the path variable to a streaming video URL or a local media
     * file path.
     */
    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.videoview);
        mVideoView = (VideoView) findViewById(R.id.surface_view);

        if (path == "") {
            // Tell the user to provide a media file URL/path.
            Toast.makeText(
                VideoViewDemo.this,
                "Please edit VideoViewDemo Activity, and set path"
                + " variable to your media file URL/path",
                Toast.LENGTH_LONG).show();
        } else {
            /*
             * Alternatively, for streaming media you can use
             * mVideoView.setVideoURI(Uri.parse(URLstring));
             */
            mVideoView.setVideoPath(path);
            mVideoView.setMediaController(new MediaController(this));
            mVideoView.requestFocus();
        }
    }
}

```

7.1.3 Grabación de Audio y video

La manera más sencilla de iniciar la grabación de un video es usando la constante estática `ACTION_VIDEO_CAPTURE` en un Intent pasado como parámetro al método `startActivityForResult`:

```

startActivityForResult(new Intent(
    MediaStore.ACTION_VIDEO_CAPTURE), RECORD_VIDEO);

```

A continuación se muestra un ejemplo detallado:

```

private static int RECORD_VIDEO = 1;
private static int HIGH_VIDEO_QUALITY = 1;
private static int MMS_VIDEO_QUALITY = 0;

private void recordVideo(Uri outputpath) {
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    if (outputpath != null)
        intent.putExtra(MediaStore.EXTRA_OUTPUT, output);
    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, HIGH_VIDEO_QUALITY);
    startActivityForResult(intent, RECORD_VIDEO);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    if (requestCode == RECORD_VIDEO) {
        Uri recordedVideo = data.getData();
        // TODO Do something with the recorded video
    }
}

```