# Ciclo Formativo DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Módulo 8

# Programación multimedia y dispositivos móviles

Unidad Formativa 1

## Desarrollo de aplicaciones para dispositivos móviles

INICIATIVA Y COORDINACIÓN
**IFP** Innovación en Formación Profesional

*Supervisión editorial y metodológica:*
Departamento de Producto de Planeta Formación

*Supervisión técnica y pedagógica:*
Departamento de Enseñanza de **IFP** Innovación en Formación Profesional

Módulo: Programación multimedia y dispositivos móviles
UF 1 / Desarrollo de Aplicaciones Multiplataforma

MÓDULO 8

Unidad Formativa 1

# Programación multimedia y dispositivos móviles

## Desarrollo de aplicaciones para dispositivos móviles

## Esquema de contenido

# 1. DESARROLLO DE CÓDIGO

A lo largo del capítulo nos centraremos en el proceso de creación de aplicaciones Android, para lo cual estudiaremos las herramientas y fases de construcción, así como también, el proceso de compilación, preverificación, empaquetado y ejecución.

## 1.1. HERRAMIENTAS Y FASES DE CONSTRUCCIÓN

En la Sección Entorno para Android, en la página 38, se describió el entorno de desarrollo para Android (las herramientas involucradas) y también el proceso de desarrollo de una aplicación. Pero, antes de crear nuestra primera aplicación en Android, es necesario verificar que el entorno de desarrollo ha sido configurado correctamente. Para ello, es recomendable como primera medida, añadir al entorno Eclipse un ejemplo de aplicación de los proporcionados por el SDK y ejecutarlo en un emulador Android. Después, estaremos listos para escribir nuestra primera aplicación e interactuar con ella a través del emulador y finalmente a través del dispositivo físico. A continuación describiremos las fases de construcción de una aplicación Android.

### 1.1.1. Creación del proyecto

El proyecto Android puede ser un proyecto Eclipse. Las herramientas de construcción de Android, si tienen integrado Eclipse, convertirán los contenidos del proyecto en un archivo APK (Android Package, Paquete Android), que es la aplicación Android propiamente dicha. De esta forma se podrá probar la aplicación en el emulador o sobre el dispositivo.

Para crear un proyecto desde la línea de comando, debemos elegir una herramienta que nos permita ejecutar el comando create project. Este comando nos permite indicarle al paquete Java la ruta en la que se encuentra el código de la aplicación, el nivel de la API y todos los archivos necesarios para construir la aplicación Android.

Vamos a crear nuestro primer proyecto Android, para ello, desde el entorno de trabajo Eclipse, seleccionamos Archivo > Nuevo > Proyecto Android. Utilizaremos como nombre del proyecto Mi Primer App Android. A continuación, seleccionamos el nivel de la API, en este caso, escogeremos el nivel 3 correspondiente a Android 1.5, de forma que se fije automáticamente la mínima versión SDK para la API del nivel 3.

### 1.1.2. Creación del AndroidManifest.xml

Un elemento fundamental es el manifiesto, el cual recoge todos los componentes de nuestra aplicación, como, por ejemplo, los permisos. Android utiliza el manifest en tiempo de ejecución para ligar la aplicación con el sistema operativo. El manifiesto también es utilizado por el Android Market para presentar las aplicaciones a los usuarios, de acuerdo a la versión de Android que posean.

Existen otros archivos importantes, los cuales se crean por defecto con una nueva aplicación Android, estos se recogen en la Tabla 2.1.

| Android file | Descripción general |
|---|---|
| default.properties | Generada automáticamente con la creación de un proyecto. Define el destino de la aplicación y otras requeridas por sistema. |
| Src Folder | Carpeta en la que reside el código fuente de la aplicación. |
| src/com.androidbook.mirimerappendroid.java | Archivo de código fuente que define el punto de entrada de la aplicación Android. |
| res Folder | Carpeta donde se manejan todos los recursos de la aplicación. |
| res/drawable/icon.png | Icono de la aplicación mostrado en la pantalla. |
| res/layout/main.xml | Archivo de la apariencia de la pantalla. |
| res/values/strings.xml | Recursos de la aplicación. |

### 1.1.3. Creación del AVD

Como se ha explicado en la Sección Emuladores para Android, para probar una aplicación en el emulador de Androides preciso crear un AVD (Android Virtual Device, Dispositivo Virtual Android), es más, podremos crear varios AVD, cada uno para emular un dispositivo Android con características hardware particulares (tamaño de pantalla, versiones de Android, etc.).

### 1.1.4. Definición del nivel de la API

Al crear el proyecto debemos indicar a Android los niveles máximos y mínimos de la API que soportará la aplicación. De igual manera, al crear el AVD debemos indicarle a Android qué nivel de la API deberá emular el AVD y así podremos ver cómo se ejecuta la aplicación en diferentes dispositivos, implementando diferentes versiones de Android.

### 1.1.5. Creación de la configuración de lanzamiento para el proyecto

El siguiente paso es configurar las circunstancias bajo las cuales se lanzará la aplicación MiPrimerAppAndroid en el entorno Eclipse. En esta fase es donde se configuran las opciones del emulador y el punto de entrada de la aplicación. Podemos crear configuraciones de ejecución y de depuración de forma separada. Estas opciones las definimos en Eclipse desde el menú Ejecutar. Para crear una configuración de ejecución básica: seleccionamos Ejecutar > Ejecutar Configuraciones, hacemos doble clic sobre la aplicación Android, elegimos un nombre para nuestra configuración de ejecución, por ejemplo, MiPrimerAppAndroidRunConfig, a continuación buscamos MiPrimerAppAndroid en el área de trabajo y seleccionamos la pestaña llamada Target, en ella fijamos el modo de selección del dispositivo a Manual. Hay que tener en cuenta que si se fija la selección del dispositivo a Automática, cuando seleccionemos Ejecutar o Depurar en Eclipse, nuestra aplicación se ejecutará automáticamente y se instalará en el dispositivo que esté conectado en ese momento. En otro caso, este iniciará el emulador con un AVD específico. En el modo manual siempre se nos preguntará si queremos que nuestra aplicación se lance en un emulador específico, en una nueva instancia del emulador (permitiéndonos especificar un AVD), o si deseamos que nuestra aplicación sea lanzada en otro dispositivo. Si el emulador ya se está ejecutando, se conectará el dispositivo y el modo se fijará en Automático. A continuación seleccionamos Aplicar y luego Cerrar. De esta manera tendremos una configuración de ejecución para nuestra aplicación.

## 1.2. COMPILACIÓN, PREVERIFICACIÓN, EMPAQUETADO Y EJECUCIÓN

Una vez creado el proyecto, ya estamos listos para ejecutar la aplicación Mi Primer AppAndroid en el emulador. Para ello:

1. Seleccionar el icono Ejecutar como desde la barra de herramientas.
2. Seleccionar MiPrimerAppAndroidRunConfig en el menú desplegable. Si no aparece la aplicación en la lista, seleccionamos el elemento Ejecutar Configuraciones y fijamos la configuración apropiada y así la veremos en la lista la próxima vez que ejecutemos la configuración.
3. Sł la instancia del emulador. Si anteriormente seleccionamos el modo manual, entonces se nos preguntará por la instancia del emulador. En este paso podremos seleccionar una nueva instancia, si así lo queremos.
4. Seleccionar el AVD, en nuestro caso seleccionaremos el AVD correspondiente a mapSd1.5. Ver la ejecución de la aplicación en el emulador (el emulador aparecerá pasados unos minutos).
5. Hacer clic sobre el botón Menú para desbloquear el emulador y así dar paso a la ejecución de la aplicación.
6. el botón Home en el emulador, para terminar la ejecución de la aplicación.

Terminado el proceso de simulación de la aplicación en el emulador, podremos depurar la aplicación. El entorno Eclipse nos permite incluir puntos de interrupción, ver la información del logCat y depurar.

El DDMS (Dalvik Debug Monitor Service, Servicio Dalvik de Monitoreo de Depuración) permite seguir y manipular el emulador y el estado del dispositivo. Para depurar la aplicación, basta con seleccionar la opción Depurar y lanzar la configuración creada, en este caso MiPrimerAppAndroid, esperaremos a que el emulador se inicie y seguiremos las instrucciones de las ventanas de diálogo que vayan apareciendo.

Podemos mejorar las funcionalidades de nuestra aplicación, añadiendo por ejemplo un soporte multimedia, o servicios de geolocalización.

# 2. INTERFACES DE USUARIO

El SDK de Android contiene componentes y widgets que facilitan el diseño de la interfaz de usuario, la comunicación e interacción con las Activities creadas para la aplicación. Primero, es necesario aclarar algunos conceptos relacionados con la interfaz de usuario:

- **View:** Dentro del SDK de Android existe un paquete Java denominado android.view que contiene tanto interfaces como clases utilizadas para dibujar en la pantalla. Sin embargo, cuando nos referimos al objeto View, hablamos únicamente a la clase android.view.View. Además, View es la clase base para los widgets y layouts.
- **Widget:** El SDK de Android también contiene un paquete Java denominado android. Widget que se deriva de la clase View. Los widgets representan clases dentro de ese paquete y contienen casi cualquier cosa que queramos dibujar, incluido: imageView, FrameLayout, EditText y los objetos Button.
- **Layout:** Es un widget particular que se encuentra en el paquete android.widget. Es un objeto View, no dibuja nada en la pantalla, pero actúa como un contenedor (padre) que soporta otros widgets (hijos). Así, su función es definir el lugar de la pantalla en el que se dibujarán los widgets hijos, de acuerdo a determinadas reglas.

Es posible diseñar interfaces de usuario sencillas o más elaboradas, y orientadas a una o más pantallas. Además, los layouts y los widgets se pueden definir mediante dos técnicas principales: utilizando un recurso xml de la aplicación a través de programación en tiempo de ejecución.

## 2.1. CREACIÓN DE LAYOUTS MEDIANTE RECURSOS XML

Es posible crear archivos layout en XML como recursos proporcionados en el directorio del proyecto res/layout. Utilizaremos este método si: queremos definir elementos estáticos en la pantalla, las propiedades de los widgets son conocidas de antemano y fijamos atributos, por defecto, que pueden ser modificados en tiempo de ejecución mediante programación.

Veamos un ejemplo de un layout en XML llamado /res/layout/main.xml:

**Ejemplo**

```
<?xml version="1. O' encoding = "utf-8"?>
<Linearlayout xmls: android=
    "http://schemas.android.com/apk/res/android"
     android:Orientation="vertical"
     android:layout_width="fill parent"
     android:layout_heigth="fill parent"
>
<TextView
    android:id="@+id/TextView1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
```

```
    android:text="Hola!"
/>
<TextView
    android:id="@+id/TextView2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="60px"
    android:text="Primero en aparecer"
/>
</LinearLayout>
```

Este código es fácil de leer y de mantener. Si queremos lanzar de nuevo este layout, solo tenemos que escribir una línea:

```
setContentView(R.layout.main);
```

## 2.2. CREACIÓN DE LAYOUTS MEDIANTE PROGRAMACIÓN

Es posible crear los componentes de la interfaz de usuario en tiempo de ejecución programando. Sin embargo este es un procedimiento engorroso y difícil de mantener, por lo que es recomendable seguir el método de creación de interfaces de usuario mediante recursos XML, ya que es una técnica más visual y que proporciona una estructura más organizada, como se aprecia en el ejemplo del apartado 1.2.1

Veamos un ejemplo, en el que se muestra una Activity que instancia una vista LinearLayout y pone dos objetos TextView dentro de ella. Las acciones se realizan en tiempo de ejecución.

**Ejemplo**

```
public void onCreate(Bundle savedInstanceState) {
    Super.onCreate(savedInstance State);
    TextView text1 = new TextView (this);
    Text1.setText("Hola!");
    TextView text2 = new TextView(this);
    Text2.setText("Primero en aparecer");
    Text2.setTextSize((float)60);
    LinearLayout 11 = new LinearLayout(this);
    11.setOrientation(Linearlayout.VERTICAL);
    11.addView(text1);
    11.addView(text2);
    setContentView(11);
}
```

El significado de cada línea del código anterior es el siguiente:

En la primera línea llamamos al método onCreate() cuando se crea la Activity y se llama también al constructor para la clase base. A continuación, instanciamos los widgets TextView1 y TextView2 y fijamos las propiedades de cada uno de ellos, utilizando el método setText(). Luego fijamos todos los atributos de estos widgets, mediante llamadas al método sobre el objeto TextView. Ahora bien, para que los widgets TextView1 y TextView2 se muestren correctamente, los encapsularemos utilizando un layout, en este caso, utilizamos LinearLayout con orientación VERTICAL, de este modo el TextView2 comienza por debajo del TextView1, ambos alineados a la izquierda de la pantalla. Los widgets se añaden al layout en el orden en el que queremos que se muestren. Por último, llamamos al método setcontentView() para dibujar el LinearLayout y su contenido en la pantalla. Así pues, si añadimos widgets necesitaremos más atributos y por consiguiente, el código irá creciendo.

Al comparar las dos formas de creación de layouts, observamos que el uso de recursos XML facilita la vista del código y el mantenimiento del mismo. Sin embargo, independientemente del método utilizado, el resultado final es el mismo.

Como se explicó en el apartado 1.2, View es la clase base para la creación de interfaces en Android. De ella se derivan todos los widgets como EditText o Button.

Existe una clase especial derivada de View llamada ViewGroup, la cual permite mostrar los objetos View en la pantalla de una forma organizada. La clase ViewGroup puede contener otros objetos View (hijos) o subclases, las cuales se añaden mediante programación de código utilizando el método addView(). Si utilizamos recursos XML, estos se añaden definiendo el objeto hijo como un nodo en XML.

Las subclases directas derivadas de la clase ViewGroup son aquellas que terminan en la palabra layout, como por ejemplo LinearLayout, RelativeLayout o FrameLayout. Todas ellas se utilizan para posicionar grupos de objetos View (widgets) en diferentes formas en la pantalla.

Las subclases indirectas de la clase ViewGroup actúan como contenedores y proporcionan al usuario funcionalidades que le permiten interactuar con ellas como widgets normales. Estas clases se nombran de acuerdo a la funcionalidad que proporcionan, por ejemplo: Gallery (lista que se muestra de forma horizontal), Grid View, Image:Switcher, Scroll View, TabHost (es el contenedor más complejo donde cada Tab puede contener un View) y ListView.

Los usuarios no interactúan directamente con los objetos Layout, sino con los objects View que contienen.

Una herramienta muy útiles la HierarchyViewer, mediante la cual podemos revisar los objetos View y las relaciones entre padres e hijos, lanzando tanto la aplicación como el HierarchyViewer en el emulador. Además, es una herramienta necesaria en el proceso de depuración, ya que permite ver por qué algo no se dibuja en la pantalla o si un View está disponible. También es particularmente útil para ver cómo las aplicaciones manejan sus layouts y despliegues. De esta manera, si queremos crear una layout similar en otra aplicación nos será más fácil).

Hay otros tipos de layouts, cada uno tiene un método diferente y orden en el cual muestra los widgets View hijos en la pantalla. Los layouts se derivan de android.view.ViewGroup. El SDK de Android incluye los siguientes tipos de layouts:

- **Absolute Layout:** Permite especificar coordenadas x,y para una ubicación exacta. No se utiliza a partir de Android 1.5 R1.

- **Framework Layout:** Permite mostrar múltiples imágenes en la misma región. El tamaño del layout corresponderá al View hijo más grande en la pila de elementos View.
- **LinearLayout:** organiza sus objetos View hijos en una sola fila o columna, dependiendo de si el atributo orientación se fija como horizontal o vertical. Útil para crear formularios.
- **RelativeLayout:** permite especificar dónde están los widgets View hijos en relación con los otros. Es decir, se pueden posicionar above, below, to the left of o bien to the right of, otro View referido por su único identificador.
- **TableLayout:** organiza los hijos dentro de columnas. Los objetos View se añaden dentro de cada fila de la tabla mediante el layout TableRow (que básicamente es un LinearLayout con orientación horizontal) para cada fila de la tabla.

También es posible crear layouts complejos, ya que los layouts contienen objetos View los cuales son ellos mismos un View que puede contener otros layouts. Todos ellos tienen atributos, los cuales se aplican a cualquier View hijo dentro de ese layout y se pueden fijar mediante programación en tiempo real pero, por lo general, se fijan en archivos XML utilizando la sintaxis: android:Layout attribute_name="value".

En la creación de interfaces de usuario contamos con los elementos básicos como el widget TextView, el cual se utiliza para mostrar cadenas fijas de texto o etiquetas en la pantalla. Lo encontramos en el paquete android.widget, porque es un View, al cual se le aplican todos los atributos estándar (ancho, alto, visibilidad, etc.).

El SDK de Android proporciona varios controles para recuperar los datos del usuario, los más utilizados son EditText (es un TextView editable) y Spinner, los cuales se escriben de la siguiente forma dentro de un archivo layout XML:

**Ejemplo**

```
<EditText
    android:id='@+id/EditTextO1"
    android:layout_height="wrap_content"
    android:hint="type here"
    android:lines="4"
    android:layout_width="fill_parent"
/>
<Spinner
    android:id="@+id/Spinner01"
    android:layout_width="fill_parent"
    android:layout_height="wrap content"
    android:entries="@array/colors"
    android:prompt="@string/spin_prompt"
/>
```

En la definición de EditText encontramos dos elementos interesantes: hint con el que ponemos un texto en una caja de texto que desaparecerá en cuanto el usuario comience a escribir y el otro atributo es lines, que define el número de líneas que tendrá la caja de texto.

En ocasiones nos interesará limitar el número de selecciones disponibles para el usuario. Así, cuando el usuario realiza su selección una ventana emergente muestra el texto seguido por una lista de posibles selecciones, de las cuales solo se puede escoger una y después de lo cual la ventana emergente desaparece. En la definición del Spinner, el atributo entries se fija para los valores que muestra asignándoles a un array, en este caso @array/colors.

Otros elementos típicos en el diseño de una interfaz son los botones. El SDK de Android proporciona diferentes tipos como:

- **Button:** se utiliza para realizar acciones como enviar un formulario o confirmar una selección. Pueden contener texto o imagen.
- **CheckBox:** este botón tiene dos estados, seleccionado y no seleccionado, se utiliza para activar o desactivar múltiples elementos de una lista
- **Toggle Button:** es similar al CheckBox, pero permite mostrar un estado de forma visual, el comportamiento por defecto de este botón es como el de un botón de encendido/apagado.
- **Radio Button:** permite la selección de un elemento. La agrupación de varios botones de este tipo dentro de un contenedor llamado RadioGroup permite que se seleccione un RadioButton a la vez.

La clase android.widget.Button proporciona la implementación para un botón básico en el SDK de Android y dentro del layout XML los botones se implementan utilizando el elemento Button. El atributo principal para un botón básico es el campo de texto, con el cual identificamos al botón (por ejemplo, Aceptar, Cancelar, Enviar, etc.)

**Ejemplo**

```
<Button
    android:id="@+id/basic_button"
    android:layout_width="wrap content"
    android:layout_height="wrap content"
    android:text="Basic_Button"
/>
```

Otros elementos frecuentemente utilizados en el diseño de interfaces son los indicadores visuales de información, como barras de progreso (pequeñas, mediana y larga), relojes, entre otros.

La definición del recurso layout XML para una barra de progreso es como sigue:

```
<ProgressBar
    android:id="@+id/progress_bar"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Basic Button"
/>
```

La definición del estilo de la barra de progreso horizontal se hace del siguiente modo:

```
<ProgressBar
        android:id="@+id/progress_bar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="fill_parent"
        android:text="Basic_Button"
/>
```

Otros widgets que se pueden incorporar a las interfaces de usuario son: Seekbar para ajustar el volumen, rating para mostrar u obtener una clasificación de un usuario o chronometer para indicar el paso del tiempo.

También podemos configurar menús de opciones mediante el control OptionsMenu el cual puede contener iconos, submenús y teclas rápidas. Para ello es necesario sobrescribir la implementación del método onCreateOptionsMenu en nuestra Activity.

Por otra parte, para proporcionar al usuario acciones adicionales al seleccionar determinados elementos, utilizaremos ContextMenu, un subtipo de Menu. En este caso es necesario implementar el método oncreate ContextMenu() en nuestra Activity para que el menú contextual se muestre en pantalla.

Hasta aquí hemos visto algunos de los eventos básicos. Sin embargo, existen otros eventos generados por otras acciones del usuario, por ejemplo cuando toca la pantalla, cuando se presiona la pantalla hasta que se realiza una acción, cuando se realizan gestos o cuando se cambia de orientación (horizontal o vertical), para todos estos eventos Android proporciona las clases correspondientes.

De otra parte, podemos trabajar con estilos, que son valores comunes para el atributo View. Los estilos se pueden aplicar a widgets View individuales y pueden incluir configuraciones para las fuentes o el color del texto, es decir, los atributos dependen del View dibujado. Esto significa que cada atributo del estilo puede cambiar la apariencia de un objeto particular.

Los estilos se definen dentro del archivo recurso res/values/styles.xml. Este archivo XML contiene una etiqueta resources con un número de etiquetas de estilo, las cuales contienen un elemento etiqueta para cada atributo y su valor que será aplicado con el estilo.

Si tenemos más de un estilo entonces hablamos de temas, los cuales permiten aplicar el estilo a todos los objetos View dentro de una Activity específica. Esta es una forma de hacer la interfaz de usuario más consistente y una forma de definir esquemas de color y configuraciones comunes.

El SDK de Android proporciona muchos componentes para la interfaz de usuario. Hemos visto que es posible combinar diferentes widgets para construir interfaces más ricas, sencillas e intuitivas, así como también la importancia de detectar las acciones del usuario y cómo manejar dichos eventos. Finalmente hemos visto que se pueden aplicar temas para proporcionar un estilo consistente a la aplicación.

## 3. BASES DE DATOS Y ALMACENAMIENTO

Android utiliza bases de datos para almacenar la información que es necesario que permanezca aun después de que el usuario cierre la aplicación o apague el dispositivo. Por ello, los desarrolladores de aplicaciones para Android disponen de la base de datos SQLite, la cual combina la interfaz de SQL con una pequeña memoria y una velocidad aceptable.

SQLite es soportada en tiempo de ejecución, es estable y popular en muchos dispositivos pequeños y fáciles de usar. Algunas de las razones por las cuales es una buena elección para desarrollar aplicaciones son: es una base de datos que no requiere configuración, no tiene un servidor sino un conjunto de librerías que proporcionan funcionalidad a la base de datos, es una base de datos de un solo archivo, lo que la convierte en una base de datos segura y es de código abierto.

SQLite no es la única alternativa para construir una base de datos en Android, también es posible utilizar bases como JavaDB o MongoDB, pero será necesario recopilar las librerías necesarias.

Para trabajar con SQLite es necesario crear una subclase de SQLiteOpen Helper, la cual contiene la lógica para crear y actualizar una base de datos. Esta subclase necesita tres métodos:

- **Constructor:** toma el contexto (por ejemplo, una Activity), el nombre de la base de datos, un cursor opcional (generalmente null) y un entero que representa la versión del esquema de la base de datos que estamos utilizando.
- **OnCreate():** el cual pasa el objeto SQLiteDatabase que necesitamos rellenar con las tablas y datos iniciales apropiados.
- **OnUpgrade():** el cual permite pasar de una versión de un esquema a otra. Dentro de este método se realiza la edición de los datos de las tablas hasta encajar con la estructura de la nueva versión.

De otra parte, si queremos leer el contenido de la base de datos, la subclase SQLiteOpenHelper proporciona el método getReadableDatabase() el cual devuelve un objeto de tipo SQLiteDatabase sobre el cual podemos realizar las operaciones de consulta de datos. Si lo que queremos es escribir en la base de datos, tendremos que llamar al método getWritableDatabase().

Los resultados de las consultas los obtenemos mediante el objeto Cursor, el cual se puede gestionar de forma manual, cerrándolo al terminar la consulta o utilizando el método startManagingCursor() asociado a una Activity particular, la cual se encargará de llamar al método de activate() cuando la Activity se pare, o llamar al método requery() cuando la Activity esté disponible de nuevo y cerrándolo cuando la Activity se destruya.

Para liberar el objeto SQLiteDatabase es necesario cerrar la conexión llamando al método close().

## 4. CONTEXTO GRÁFICO

Android permite dibujar en la pantalla imágenes PNG y JPG, texto o formas básicas, todos ellos con varios colores, estilos y gradientes. También podemos modificarlos mediante transformaciones estándar e incluso animarlos. Para trabajar con la pantalla es necesario utilizar Canvas válidos, que se pueden crear con una clase que extienda View e implementando el método con Draw(). Los gráficos 2D se pueden generar de varias formas

- **Canvas:** permite crear el espacio rectangular para el dibujo. Hay diversos métodos para dibujar imágenes, texto, formas y soporte para el recorte de regiones. Las dimensiones del Canvas están definidas por la vista del contenedor. Podemos obtener las dimensiones del Canvas mediante los métodos getHeight() y getWidth().
- **View:** es la forma más sencilla de generar un gráfico, pero es poco flexible.
- **SurfaceView:** es la mejor opción si la información dibujada requiere muchas actualizaciones. Se utilizan dos threads, uno para controlar la superficie a pintar y otro para ir dibujando independientemente de los eventos que se estén sucediendo en el thread principal.

Otra clase importante es Paint, la cual encapsula el estilo (Style), el color (con el método setColor() y los colores predefinidos en Android dentro de la clase android. graphics.Color y la información de la representación que se aplica a gráficos, formas, textos en una Typeface. También se utilizan otras subclases importantes como gradient, que pueden ser lineales, radiales o de tipo barrido. De otra parte, Paint incluye utilidades para presentar el texto en la pantalla con diferentes tipos de letras (se pueden definir propiedades como antialisado, subrayado y tachado mediante el método setFlags() del objeto Paint y estilos. También se pueden incluir fuentes personalizadas, pero hay que asegurarse de que el estilo a utilizar es soportado por el dispositivo.

Android también permite trabajar con mapas de bits dentro de un Canvas válido y dentro del método onDraw() de View, utilizando uno de los métodos drawBitmap().

Existe una clase que permite transformar un mapa de bits, esa clase es Matrix, con ella podemos crear reflejos, rotar imágenes, entre otras operaciones. El siguiente código muestra un ejemplo de uso de la clase Matrix:

**Ejemplo**

```
import android.graphics.Bitmap;
import android.graphics.Matrix:
Matrix mirrorMatrix = new Matrix();
mirrorMatrix.preScale (-1, 1);
Bitmap mirrorImage = Bitmap.CreateBitmap(image, 0, 0
Image.getWidth(), image.getHeight(), mirrorMatrix, false);
```

Otros efectos y utilidades las encontramos en el paquete android. graphics como parte de la SKD de Android.

Para definir y dibujar formas básicas se utiliza la clase ShapeDrawable con las clases especializadas Shape. Para ellas se utiliza el paquete android. graphics, drawable. shapes, el cual incluye rectángulos, rectángulos con bordes redondeados, óvalos, arcos, entre otros. Si lo que queremos es definir otro tipo de formas,

las podemos hacer definiendo trayectorias con la clase android. graphics. Path que encapsula líneas y curvas. El siguiente código define una estrella de cinco puntas:

```
import android.graphics.Path;
Path p = new Path();
        p.move(50, 0);
        p.lineTo(25, 100);
        p.lineTo(100, 50);
        p.lineTo(0, 50);
        p.lineTo(75, 100);
        p.lineTo(50, 0);
```

Podemos encapsular esta trayectoria en una PathShape, crear un ShapeDrawable y pintarla de azul:

```
import android.graphics.draWable.Shapedrawable;
import android.graphics.drawable.shapes.PathShape;
ShapeDrawable star =
        New ShapeDraWable(new PathShape (p, 100, 100));
        star.setIntrinsicHeight(100);
        star.setIntrinsicWidth(100);
        star.getPaint(100).setColor(Color.BLUE);
```

Android soporta tres tipos de animaciones gráficas:

- **GIF animados:** almacenan los frames de animación dentro de la imagen, en este caso solo tenemos que incluir los GIF como cualquier otro recurso gráfico dibujable.
- **Animación frame a frame:** el usuario debe proporcionar todos los frames gráficos de la animación. Se utiliza para transformaciones gráficas complejas que no se implementan fácilmente mediante programación, por ejemplo crear la ilusión óptica en una imagen, simplemente proporcionando un número de frames y velocidad adecuada para intercambiarlos.
- **Animación interpolada:** solo se necesita un único gráfico, a partir del cual se aplican transformaciones mediante programación.

## 4.1. EVENTOS DE TECLADO

Cada presión de una tecla se interpreta como una secuencia de eventos de teclado. La presión comienza con ACTION DOWN, en el caso de una presión continuada, entonces el evento inicial con ACTION DOWN es seguido de un valor no-cero para getRepeatCount() y el último evento de la tecla, cuando esta vuelve a su posición inicial (es decir, no presionada) es ACTION UP. Si se cancela la presión en la tecla, el evento de tecla arriba tendrá el evento FLAG CANCELED.

Los eventos de teclado van acompañados por un código de tecla getKeyCode(), un código de escaneo getScanCode() y un metaestado getMetaState(). Las constantes del código de tecla se definen en esta

clase. Las constantes del código Scan son códigos específicos obtenidos del sistema operativo y por lo general no son significativos para las aplicaciones a menos que se interpreten utilizando KeyCharacterMap. Los metaestados, por su parte, describen el estado actual de los modificadores del teclado como META SHIFT ON o META ALT ON.

Las múltiples combinaciones de teclas realizan diversas funciones en diferentes dispositivos de entrada, por lo tanto debemos ser cautos a la hora de interpretarlos. Es recomendable utilizar Key Character Map asociado al dispositivo de entrada cuando se asignen las teclas a los caracteres. Por lo tanto, no debemos olvidar que puede haber múltiples dispositivos activos al mismo tiempo y cada uno debe tener su propio mapa de caracteres.

Para navegar entre opciones y pantallas, Android trabaja con cuatro botones físicos, con los cuales se puede realizar acciones como bajar/subir el volumen, por ejemplo. Los eventos se pueden personalizar mediante la clase KeyEvent, la cual mantiene un registro de los botones que existen en un dispositivo Android, por ejemplo: KEYCODE_POWER (botón de encendido), KEYCODE_BACK (botón de retorno), KEYCODE_MENU (botón de menú), KEYCODE_HOME (botón de hogar), KEYCODE_SEARCH (botón de búsqueda), KEYCODE_CAMERA (botón de cámara), KEYCODE_VOLUMEN, UP (botón de subir Volumen), KEYCODE_VOLUMEN_DOWN (botón de bajar volumen), DPAD, KEYCODE_DPAD_CENTER, KEYCODE_DPAD_UP y KEYCODE_DPAD_DOWN, entre otros. El sistema envía el evento del botón presionado al método que pueda devolver la llamada. Los métodos que pueden atender estas llamadas son:

- Para los botones físicos: onKeyUp(), onKeyDown (), onKeyLongPress ().
- Para responder al trackball (si lo hay): onTrackBallEvent ().
- Para responder a un toque en la pantalla: onTouchEvent().
- Para responder a la pérdida de enfoque: onFocusChanged().

Obviamente si queremos personalizar nuestra aplicación, podremos sobrescribir estos métodos.

## 4.2. IMÁGENES

Android permite mostrar imágenes, por ejemplo de mapas de bits, en la pantalla de nuestro dispositivo, utilizando el widget ImageView. También podemos capturar imágenes utilizando el objeto Camera del SDK de Android (android. hardware.Camera), el cual controla la cámara de los teléfonos que tiene el soporte de cámara activado. La función de vista previa de la cámara se basa en la asignación de un SurfaceHolder de un tipo apropiado. De esta manera, las aplicaciones pueden controlar el lugar y el tamaño del área de previsualización utilizada por la cámara.

Las imágenes se pueden almacenar en el directorio local de la aplicación, sin embargo en algunas aplicaciones resulta útil tener las imágenes en la biblioteca de imágenes compartida del dispositivo. Para tal fin se utiliza el objeto ContentStore junto con MediaStore.

También podemos asignar las imágenes como fondo de pantalla utilizando algunos métodos a los cuales se accede desde el objeto Context. Los fondos de pantalla son una excelente herramienta de personalización. El fondo de pantalla actual se debe recuperar con una llamada a los métodos getWallpaper() o peekWallpaper(). Para definir el tamaño del fondo de pantalla se utilizan los métodos getWallpaper(), DesiredMinimunHeight() y getWallpaperDesiredMinimuniWidth(). Finalmente, se asigna el fondo de pantalla con una llamada al método setWallpaper(). La aplicación necesita el permiso android.permission. SET_WALLPAPER dentro del archivo AndroidManifest.xml. Para borrar un fondo de pantalla se llama al método clearWallpaper() de Context.

# 5. CONTEXTO GRÁFICO

La conectividad depende de la localización del usuario y de las características de su dispositivo. Los usuarios demandan aplicaciones estables, lo cual implica que los desarrolladores deben tener mucho cuidado cuando diseñan aplicaciones que requieren el uso de la Red. Android proporciona muchas herramientas y clases para tal fin.

## 5.1. MODELO DE HILOS

La forma más flexible para lograr un thread (hilo/subproceso) Android amigable en segundo plano es crear una instancia de una subclase Handler. Solo se necesita un objeto Handler por actividad y no es necesario registrarlo manualmente. Simplemente creando la instancia es suficiente para registrarla en el subsistema de hilos de Android.

El hilo se puede comunicar con la clase Handler, la cual hará todo el trabajo sobre el hilo de la interfaz de usuario de la actividad. Es importante, por lo tanto, que si la interfaz de usuario cambia, por ejemplo, por la actualización de los widgets, esto debe ocurrir sólo en el hilo de la interfaz de usuario de la actividad.

Hay dos opciones para comunicarse con la clase Handler:

- **Mensajes:** para enviar un mensaje a Handler, es necesario invocar obtainMessage() y así obtener el objeto Message. Cuanto más complicado sea el procesamiento del Handler, más probabilidades hay de que sea necesario poner los datos en Message para ayudar a Handler a distinguir los diferentes eventos. Lo siguiente será enviar el Message a Handler a través de su cola de mensajes, utilizando alguno de los siguientes métodos:
  1. *sendMessage():* pone el mensaje en la cola inmediatamente.
  2. *sendMessage AtFrontOfQueue():* pone el mensaje en la cola inmediatamente, colocándolo al frente de la cola de mensajes de modo que el mensaje tendrá prioridad sobre los otros de la cola.
  3. *sendMessageAtTime():* pone el mensaje en la cola a la hora indicada, expresada en forma de milisegundos
  4. *sendMessageDelayed():* pone el mensaje en la cola, después de un mensaje de retraso expresado en milisegundos.

  Para procesar esos mensajes, Handler necesita implementar handle Message (), el cual será llamado con cada mensaje que aparezca en la cola de mensajes. Esto hace posible la actualización de la interfaz de usuario.

- **Objetos Runnable:** Handler ejecuta los objetos Runnable en el hilo de la actividad del la interfaz de usuario. Dado que Handler soporta los métodos post() y postDelayed() para añadir los objetos Runnable a la cola, también es posible utilizar esos mismos métodos en Vieu. De esta forma se reduce significativamente el código, pero se pierde un poco de flexibilidad.

Algunas veces no sabemos si estamos ejecutando sobre el hilo de la interfaz de usuario de nuestra aplicación; para resolver esto Activity ofrece el método runOnUiThread(). Este trabaja como los métodos post() en Handler y View haciendo que el objeto Runnable se ejecute en el hilo de la interfaz de usuario si no estamos sobre el hilo de la interfaz de usuario en ese momento. Pero si ya estamos sobre ese hilo entonces invoca inmediatamente Runnable. De esta forma: no tendremos retardo si estamos en el hilo de la IU, y nos aseguraremos en caso de no estar aún.

## 6. MANEJO DE CONEXIONES HTTP

En el desarrollo de aplicaciones para dispositivos móviles un factor importante a tener en cuenta es la conectividad. Por ello, los desarrolladores deben proporcionar alternativas para las conexiones inestables y tener en cuenta que el correcto funcionamiento de una aplicación dependerá no solo de las características de la misma sino también de las características de la red disponible y de las del dispositivo móvil. En este sentido, el SDK de Android proporciona las herramientas y clases necesarias para asegurar un buen funcionamiento de la aplicación.

Para transferir datos hacia y desde la Red, se utiliza el protocolo HTTP De esta forma se puede encapsular cualquier tipo de datos y asegurarlos mediante SSL (Secure Socket Layer, Capa de Conexión Segura).

Para leer datos de un sitio web, podemos aprovechar la clase URL que pertenece al paquete java.net, para leer una cantidad fija de texto desde un archivo en un servidor web.

```
import java.io.InputStream;

import java.net.URL;

try {
        URL text = new URL (
                "http://www.urjc.es");
                InputStream isText = text.OpenStream();
                Byte [] bText = new byte(250);
                int readSize = isText.read(bText);
                Log.i("Net", "readSize = " + readSize);
                Log.i("Net", "bText = " + new String(bText));
                isText.close();
        } catch (Exception e) {
                log.e("Net", "Error en la llamada a la red", e);
        }
}
```

Lo primero que se hace es crear un objeto URL con la URL del sitio que nos interesa leer. Accedemos al recurso indicado en la URL, leemos los datos y cerramos el Inputstream.

Para que las conexiones funcionen en cualquier aplicación Android, en la aplicación debe aparecer la siguiente frase en el archivo del manifiesto XML:

```
<uses-permission android:name="android.permission.INTERNET'/>
```

El SDK de Android proporciona dos formas para manejar la sobrecarga de procesamiento del thread principal de la IU:  la clase AsyncTask, que encapsula el procesamiento en segundo plano y facilita la comuni-

cación con el thread de la IU, mientras maneja el ciclo de vida de la tarea en segundo plano dentro del contexto del ciclo de vida de la actividad. Esta clase crea una interfaz más sencilla para las operaciones asíncronas que la creada manualmente con Java Thread. Y la clase estándar de Java Thread.

Conocer el estado de la red es una operación importante. Para ello, el SDK de Android a través de diversos métodos de la clase Connectivity Manager permite determinar si la conexión a la red está disponible, antes de intentar utilizar un recurso. Así por ejemplo, para saber si la red móvil está disponible y conectada utilizaríamos un código como el siguiente:

```
import android.net.ConnectivityManager;

import android.net.NetworkInfo;

ConnectivityManager cm = (ConnectivityManager)

        getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo ni = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);

boolean isWifiAvail = ni.isAvailable();

boolean isWifiConn = ni.isConnected();

ni = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);

boolean isMobileAvail = ni.isAvailable();

boolean isMobileConn = ni.isConnected();

status.setText(

        "WiFi\nAvail = " + isWifiAvail +

        "\nConn" = " + isWifiConn +

        "\nMobile\nAvial = " + isMobileAvail +

        "\nConn" = " + isMobileConn);
```

Mediante el método getSystemService() se recupera una instancia del objeto ConnectivityManager, después esta instancia recupera los objetos NetworkInfo tanto para TYPE_WIFI como para TYPE_MOBILE. Sin embargo, el que una red esté disponible no implica que el recurso de la red esté disponible, pero con una llamada al método requestRouterToHost() se puede conocer el estado. De esta forma, el usuario puede obtener una mejor información ante problemas en la Red.

No debemos olvidar que aunque los emuladores permiten simular varios tipos de redes móviles y controlar la latencia de la red, no pueden simular el comportamiento real de una red en condiciones normales, por ejemplo, cuando el usuario está sin cobertura o va por un túnel, ya que solo las pruebas en circunstancias reales arrojarán resultados reales.

## 6.1. COMPLEMENTOS DE LOS NAVEGADORES

El estándar Android Browser está basado en Webkit y desarrollado por el motor V8 Javascript de Google, lo que lo convierte en un navegador muy robusto, con buen rendimiento e interpretación de las páginas

web. Sin embargo, existen navegadores alternativos capaces de ofrecer una experiencia de usuario rica y atractiva. A continuación analizamos las características de algunos de los más relevantes.

- **Opera Mini:** Ofrece una experiencia de navegación agradable y sólida. Soporta pestañas ilimitadas, a través de una ventana de previsualización de navegación, así como la gama habitual de las opciones de accesibilidad como los bloqueadores de ventanas emergentes (popup blockers) y tamaños de texto. Construido en gestión de RSS Feed y Bookmark Sync (a través de Opera Link), también hacen que Opera se destaque sobre el resto.

  Además, Opera ofrece una compresión server-side de imágenes, lo que le permite escoger entre diferentes calidades de imagen para ayudar en los tiempos de carga/ancho de banda. Las imágenes son más comprimidas a través de los servidores de Opera y posteriormente cargadas en el teléfono, con una mejora importante en los tiempos de carga.

  De otra parte, Opera no ofrece soporte para Flash y si se abren muchas pestañas, éstas se amontonan demasiado haciendo difícil la navegación entre ellas. Entre las novedades encontramos que el Opera Mini Next (versión previa del próximo navegador Opera Mini), ofrece varias funciones orientadas a las redes sociales. La nueva versión de la aplicación llegará a los dispositivos con Smart Page, una característica que debería incluirse en todos los navegadores Opera Mini para teléfonos estándar. A través de ella, los usuarios se beneficiarán de acceso rápido a las redes sociales e integración con Facebook y Twitter directamente en la pantalla principal. También ofrece acceso a las últimas noticias que aparecen en la Web.

| Característica | Android | Opera Mini | Skyfire | Dolphin Browser HD | Ascope |
|---|---|---|---|---|---|
| Licencia | Libre | Libre | Libre | Libre/pago | Libre/pago |
| Múltiples pestañas | < 8 | Ilimitadas | < 8 | < 8 | < 8 |
| Multitouch | X | X | X | X | X |
| Zoom ajustable | X | X | X | X | X |
| Bloqueo de ventanas emergentes | X | X | X | X | X |
| Soporte flash | X | | X | X | X |
| Control volumen | | | | X | X |
| Acceso rápido marcación | | X | | X | X |
| Guardar página | | | | X | X |
| Soporte piel/tema | | | | X | X |
| Gestión de clave | X | X | X | X | X |
| Gestión de RSS | | X | | | |

- **Skyfire:** Ofrece la posibilidad de ver vídeos en flash en una amplia gama de sitios que detecta de forma automática, en un reproductor dedicado, eliminando la necesidad de ampliar el vídeo de forma manual. También incorpora una cómoda barra de herramientas desde la que se puede ac-

cedera Twitter, Facebook, etc. En general, es fácil de usar, cómodo y presenta buenas alternativas para organizar las páginas preferidas o para compartir páginas vía correo, SMS o redes sociales. Entre las ventajas hay que destacar: carga rápida de páginas, soporte para Flash, Java y vídeo en streaming, comodidad de la página de inicio, incluye un práctico tutorial, detecta la presencia de vídeos en la web. Entre los inconvenientes sobresalen algunos problemas de estabilidad.

- **Dolphin Browser HD:** Es uno de los mejores navegadores para Android. Posee una interfaz de usuario con un diseño que facilita el acceso rápido a las funcionalidades que ofrece. Entre ellas hay que destacar su zoom, con soporte multitouch, la navegación inteligente por pestañas, el control por voz Sonar y la navegación por gestos. Está muy orientado a las redes sociales y facilita compartir datos a través de Facebook, Twitter, etc. Entre las ventajas sobresalen: navegación por pestañas y modo de navegación privada, muy rápido, navegación por gestos, permite añadir extensiones como Evernote o Gmail y la sincronización entre varios dispositivos. Entre los inconvenientes se encuentran: algunos problemas al descargar extensiones y la aparición de molestas ventanas emergentes.

- **Xscope:** Ofrece un conjunto de herramientas ideales para diseñadores gráficos. La nueva versión incorpora más de setenta novedades. Con sus herramientas se puede saber las dimensiones de cualquier objeto que tengamos en pantalla, distancia entre objetos, reglas para ajustar la posición de objetos con una exactitud del 100%, plantillas personalizadas para crear diseños en varias plataformas (iPhone, Android, entre otros). En definitiva, las opciones son abundantes para los navegadores Android, pero con el navegador estándar que ofrece una experiencia de navegación robusta, la diferencia depende de las funciones de cada navegador. Para los que buscan un navegador cargado de plugins, Delfín Browser HD es la mejor opción; para los que quieren la mejor experiencia de navegación Web, Opera Mini es el adecuado, y para aquellos que quieren hacer malabares con varias pestañas fácilmente o navegar por sitios pesados en Javascript, Xscope es una buena alternativa. La Tabla 2.2 resume las principales características de los navegadores presentados en este apartado.

## 6.2. ENVÍO Y RECEPCIÓN DE MENSAJERÍA

La integración de los servicios de mensajería en una aplicación proporciona un marco para el desarrollo de relaciones sociales, que hoy por hoy son demandadas por los usuarios en todos los dispositivos de comunicación. En Android, esta funcionalidad está disponible en el paquete android. telephony.gsm y para hacer uso de ella se requiere de dos permisos diferentes, dependiendo de si la aplicación envía o recibe mensajes. De esta forma, el archivo AndroidManifest. xml debe contener las líneas que indiquen dichas acciones:

```
<uses-permission android: name="android. permission. SEND_SMS" />

<uses-permission android: name="android. permission. RECEIVE_SMS" />
```

Para que una aplicación pueda enviar un SMS es necesario obtener primero una instancia del Sms Manager y luego, simplemente hay que hacer una llamada. Sin embargo, hay que tener en cuenta que la aplicación debe tener información acerca de si el mensaje ha sido enviado o no. Para ello se debe utilizar PendingIntent que permitirá escuchar el estado del SMS.

De otra parte, si nos interesa que la aplicación pueda recibir mensajes, es necesario que registre un BroadcastReceiver para escuchar los intentos asociados con la recepción de un mensaje. El SDK de Android

permite descodificar el paquete de datos recibido mediante una llamada al método SMS. Message.createFromPdu(), ya que el mensaje contiene varios arrays de datos formados por datos PDU (formato utilizado por los protocolos inalámbricos de mensajería). Finalmente, para recuperar el cuerpo del SMS se llama al método getDisplayMessageBody().

Es recomendable estar atento a los cambios que pueden ser introducidos en las nuevas versiones del SDK de Android, respecto a esta y todas las funcionalidades ofrecidas para el desarrollo de aplicaciones.

**Actividad**

Seleccionar una aplicación de la lista de aplicaciones disponibles en: http://developer.android.com/resources/browser.html?tag=sample y realizar un análisis sobre el desarrollo de la aplicación seleccionada:

1. ¿Qué tipo de metodología software sería la más apropiada para desarrollar dicha aplicación?
2. Entender cómo los terminales destino determinan la funcionalidad final de la aplicación.
3. Realizar el análisis de factibilidad de la aplicación.
4. Realizar un análisis de riesgos.
5. Hacer un seguimiento de la funcionalidad del teléfono a través de la gestión de configuración.
6. Hacer el análisis de la aplicación desde el punto de vista de su estabilidad en un sistema con limitaciones de memoria.
7. Definir las características de la interfaz de usuario, teniendo en cuenta varios dispositivos móviles.
8. Probar la aplicación en un emulador y en un dispositivo real.
9. Determinar qué factores pueden afectar la venta de la aplicación.
10. Definir el proceso de mantenimiento de la aplicación.

**Caso práctico**

Se propone la creación de una aplicación que permita obtener de una base de datos, los nombres de los empleados de una empresa. La aplicación debe permitir además ver la información asociada al empleado, como su cargo, número de teléfono, dirección de correo electrónico. También debe permitir llamar y enviar correos a cada empleado desde la propia aplicación. Finalmente, debe ser posible navegar a través del organigrama de la empresa.

## Índice