Introducción a Android

Índice

1. Android

Android es un sistema operativo de código abierto para dispositivos móviles, se programa principalmente en Java, y su núcleo está basado en Linux.

1.1. Historia

Antiguamente los dispositivos empotrados sólo se podían programar a bajo nivel y los programadores necesitaban entender completamente el hardware para el que estaban programando.

En la actualidad los sistemas operativos abstraen al programador del hardware. Un ejemplo clásico es Symbian. Pero este tipo de plataformas todavía requieren que el programador escriba código C/C++ complicado, haciendo uso de librerías propietarias. Especiales complicaciones pueden surgir cuando se trabaja con hardware específico, como GPS, trackballs, pantallas tácticles, etc.

Java ME abstrae completamente al programador del hardware, pero las limitaciones impuestas por la máquina virtual le restringen mucho su libertad a la hora de acceder al hardware del dispositivo.

Esta situación motivó la aparición de Android, cuya primera version oficial (la 1.1) se publicó en febrero de 2009. Esto coincidió con la proliferación de smartphones con pantallas táctiles.

Desde entonces han ido apareciendo versiones nuevas del sistema operativo, desde la 1.5 llamada Cupcake y que se basaba en el núcleo de Linux 2.6.27, hasta la versión que usaremos en este módulo, la versión 2.3, conocida como Gingerbread y basada en el núcleo 2.6.35.x. Cada versión del sistema operativo tiene un nombre inspirado en la repostería, siguiendo un orden alfabético con respecto al resto de versiones de Android (Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, etc).

1.2. Open Source

Android - tanto el sitema operativo, como la plataforma de desarrollo - están liberados bajo la licencia de Apache. Esta licencia permite a los fabricantes añadir sus propias extensiones propietarias, sin tener que ponerlas en manos de la comunidad de software libre.

Al ser de open source, Android hace posible:

- la existencia de una gran comunidad de desarrollo, gracias a sus completas APIs y documentación ofrecida
- desarrollar desde cualquier plataforma (Linux, Mac, Windows, etc)
- su uso en cualquier tipo de dispositivo móvil

- que cualquier fabricante pueda diseñar un dispositivo que trabaje con Android, incluso adaptando o extendiendo el sistema para satisfacer las necesidades de su dispositivo concreto
- un gran valor añadido para los fabricantes de dispositivos: las empresas se ahorran el coste de desarrollar un sistema operativo completo desde cero
- un gran valor añadido para los desarrolladores: éstos se ahorran tener que programar APIs, entornos gráficos, aprender acceso a dispositivos hardware particulares, etc.

Android está formado por los siguientes componentes:

- núcleo basado en el de Linux para el manejo de memoria, procesos y hardware (se trata de una rama independiente de la rama principal, de manera que las mejoras introducidas no se incorporan en el desarrollo del núcleo de GNU/Linux)
- bibliotecas open source para el desarrollo de aplicaciones, incluyendo SQLite, WebKit, OpenGL y manejador de medios
- entorno de ejecución para las aplicaciones Android. La máquina virtual Dalvik y las bibliotecas específicas dan a las aplicaciones funcionalidades específicas de Android
- un framework de desarrollo que pone a disposición de las aplicaciones los servicios del sistema como el manejador de ventanas, de localización, proveedores de contenidos, sensores y telefonía
- SDK (kit de desarrollo de software) que incluye herramientas, plug-in para Eclipse, emulador, ejemplos y documentación
- interfaz de usuario útil para pantallas táctiles y otros tipos de dispositivos de entrada, como por ejemplo, teclado y trackball
- aplicaciones preinstaladas que hacen que el sistema operativo sea útil para el usuario desde el primer momento. Cabe destacar que cuenta con las últimas versiones de Flash Player
- muy importante es la existencia del Android Market, y más todavía la presencia de una comunidad de desarrolladores que publican allí sus aplicaciones, tanto de pago como gratuitas. De cara al usuario, el verdadero valor del sistema operativo está en las aplicaciones que se puede instalar

El principal responsable del desarrollo de Android es la Open Handset Alliance, un consorcio de varias compañías que tratan de definir y establecer una serie de estándares abiertos para dispositivos móviles. El consorcio cuenta con decenas de miembros que se pueden clasificar en varios tipos de empresas:

- operadores de telefonía móvil
- fabricantes de dispositivos
- fabricantes de procesadores y microelectrónica
- compañías de software
- compañías de comercialización

Por lo tanto, Android no es "de Google" como se suele decir, aunque Google es una de las empresas con mayor participación en el proyecto.

Por último concluimos esta sección considerando algunas cuestiones éticas. Uno de los

aspectos más positivos de Android es su carácter de código abierto. Gracias a él, tanto fabricantes como usuarios se ven beneficiados y tanto el proceso de programación de dispositivos móviles como su fabricación se acelera. Todos salen ganando.

Otra consecuencia de que sea de código abierto es la mantenibilidad. Los fabricantes que venden dispositivos con Android tienen el compromiso de que sus aparatos funcionen. Si apareciera algún problema debido al sistema operativo (no nos referimos a que el usuario lo estropee, por supuesto) el fabricante, en última instancia, podría abrir el código fuente, descubrir el problema y solucionarlo. Esto es una garantía de éxito muy importante.

Por otro la seguridad informática también se ve beneficiada por el código abierto, como ha demostrado la experiencia con otros sistemas operativos abiertos frente a los propietarios.

Hoy en día los dispositivos móviles cuentan con hardware que recoge información de nuestro entorno: cámara, GPS, brújula y acelerómetros. Además cuentan con constante conexión a Internet, a través de la cuál diariamente circulan nuestros datos más personales. El carácter abierto del sistema operativo nos ofrece una transparencia con respecto al uso que se hace de esa información. Por ejemplo, si hubiera la más mínima sospecha de que el sistema operativo captura fotos sin preguntarnos y las envía, a los pocos días ya sería noticia.

Esto no concierne las aplicaciones que nos instalamos. Éstas requieren una serie de permisos antes de su instalación. Si los aceptamos, nos hacemos responsables de lo que la aplicación haga. Es necesario aclarar que esto no es un problema de seguridad,ya que los problemas de seguridad realmente surgen si se hace algo sin el consentimiento ni conocimiento del usuario.

No todo son aspectos éticos positivos, también abundan los preocupantes.

Los teléfonos con Android conectan nuestro teléfono con nuesto ID de google. Hay una transmisión periódica de datos entre Google y nuestro terminal: correo electrónico, calendario, el tiempo, actualizaciones del Android Market, etc. En este sentido, el usuario depende de Google (ya no dependemos sólo de nuestro proveedor de telefonía móvil). Además, la inmensa mayoría de los servicios que Google nos ofrece no son de código abierto. Son gratuitas, pero el usuario desconoce la suerte de sus datos personales dentro de dichas aplicaciones.

El usuario puede deshabilitar la localización geográfica en su dispositivo, y también puede indicar que no desea que ésta se envíe a Google. Aún así, seguimos conectados con Google por http, dándoles información de nuestra IP en cada momento. De manera indirecta, a través del uso de la red, ofrecemos información de nuestra actividad diaria. Si bien el usuario acepta estas condiciones de uso, la mayoría de los usuarios ni se paran a pensar en ello porque desconocen el funcionamiento del sistema operativo, de los servicios de Google, y de Internet en general.

Lógicamente, nos fiamos de que Google no hará nada malo con nuestros datos, ya que no

ha habido ningún precedente.

2. Aplicaciones Android

Las aplicaciones Android están compuestas por un conjunto heterogéneo de componentes enlazados mediante un archivo llamado `AndroidManifest.xml` que los describe e indica cómo interactúan. Este archivo también contiene metainformación acerca de la aplicación, como por ejemplo los requerimientos que debe cumplir la plataforma sobre la que se ejecuta.

Una aplicación Android estará compuesta por los siguientes componentes (no necesariamente todos ellos):

- **Actividades**. Las actividades son la capa de presentación de la aplicación. Cada pantalla a mostrar en la aplicación será una subclase de la clase `Activity`. Las actividades hacen uso de componentes de tipo `View` para mostrar elementos de la interfaz gráfica que permitan mostrar datos y reaccionar ante la entrada del usuario.
- **Servicios**. Los servicios son componentes que se ejecutan en el background de la aplicación, ya sea actualizando fuentes de información, atendiendo a diversos eventos, o activando la visualización de notificaciones en una actividad. Se utilizan para llevar a cabo procesamiento que debe ser realizado de manera regular, incluso en el caso en el que nuestras actividades no sean visibles o ni siquiera estén activas.
- **Proveedores de contenidos**. Permiten almacenar y compartir datos entre aplicaciones. Los dispositivos Android incluyen de serie un conjunto de proveedores de contenidos nativos que permiten acceder a datos del terminal, como por ejemplo los contactos o el contenido multimedia.
- **Intents**. Los intents constituyen una plataforma para el paso de mensajes entre aplicaciones (y también dentro de una misma aplicación). Emitiendo un intent al sistema declara la intención de tu aplicación de que se lleve a cabo una determinada acción. El sistema será el encargado de decidir quién lleva a cabo las acciones solicitadas.
- **Receptores**. Permiten a tu aplicación hacerse cargo de determinadas acciones solicitadas mediante intents. Los receptores iniciarán automáticamente la aplicación para responder a un intent que se haya recibido, haciendo que sean ideales para la creación de aplicaciones guiadas por eventos.
- **Widgets**. Se trata de componentes visuales que pueden ser añadidos a la ventana principal de Android.
- **Notificaciones.** Las notificaciones permiten comunicarse con el usuario sin necesidad de robar el foco de la aplicación activa actualmente o de interrumpir a la actividad actual. Pror ejemplo, cuando un dispositivo recibe un mensaje de texto, avisa al usuario mediante luces, sonidos o mostrando algún icono.

2.1. El archivo Manifest

Cada proyecto Android debe contener un archivo llamado `AndroidManifest.xml` en la carpeta raíz de su jerarquía de carpetas. Este archivo permite establecer la estructura de la aplicación y su metainformación, así como sus componentes o sus requisitos.

Incluye un nodo por cada uno de los componentes de una aplicación (actividades, servicios, proveedores de contenidos, etc.) y mediante el uso de Intents y permisos determina cómo interactuan unos con otros o con otras aplicaciones. También incluye atributos para especificar la metainformación asociada a la aplicación, como su icono, por ejemplo.

Veamos como ejemplo el archivo `AndroidManifest` de un proyecto Android recién creado:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="es.ua.jtech.android"
   android:versionCode="1"
   android:versionName="1.0">
   <uses-sdk android:minSdkVersion="9" />

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

El elemento raíz del fichero es `<manifest>`, cuyo atributo `package` tendrá como valor el nombre del paquete del proyecto. El atributo `versionCode` permite al desarrollador indicar el número de versión actual de la aplicación. Este valor lo usará el desarrollador para comparar entre versiones. Por otra parte, el atributo `versionName` contendrá la cadena que describirá la versión y que sí que se mostrará a los usuarios.

El elemento `<manifest>` contendrá otros elementos que definirán diferentes componentes de la aplicación, opciones de seguridad y requisitos. Uno de estos nodos es `<uses-sdk>`. Este elemento permite especificar, entre otras cosas, cuál debe ser la versión mínima del sistema Android que debe encontrarse instalada en el dispositivo donde se vaya a ejecutar la aplicación mediante el parámetro `minSdkVersion`. Es importante que le asignemos un valor a este parámetro. En caso contrario se asignará un valor por defecto y nuestra aplicación puede sufrir un error en tiempo de ejecución si se intenta acceder a un elemento de la API de Android no soportada. En este ejemplo el valor del atributo es 9, lo cual se corresponde con la versión de Android 2.3.1. Obsérvese que la versión del SDK no se corresponde con la versión de la plataforma y que no se puede derivar una de la otra. Para saber qué versión del SDK se corresponde con cada versión de Android podemos consultar `http://developer.android.com/guide/appendix/api-levels.html`.

El archivo Manifest sólo puede contener un elemento `<application>`. Se utiliza para establecer la metainformación de la aplicación (el nombre de la aplicación, su icono asociado, etc.). En este ejemplo tanto el valor del atributo `icon` como el del atributo `label`, que hacen referencia respectivamente al icono y al nombre de la aplicación, hacen referencia a sendos recursos de la aplicación. Más adelante en esta sesión hablaremos de los recursos.

El elemento `<application>` deberá contener un elemento de tipo `<activity>` por cada actividad presente en nuestra aplicación. Su atributo `name` contendrá el nombre de la clase de la actividad. Esto es importante, porque intentar iniciar una actividad que no esté listada en este fichero hará que se produzca un error en tiempo de ejecución. Cada elemento `<activity>` podrá contener a su vez un elemento de tipo `<intent-filter>` que permita especificar a qué Intents puede responder la actividad. En este ejemplo utilizamos este campo para indicar que nuestra única actividad es además la actividad principal, y por lo tanto la que se deberá mostrar al iniciar la aplicación.

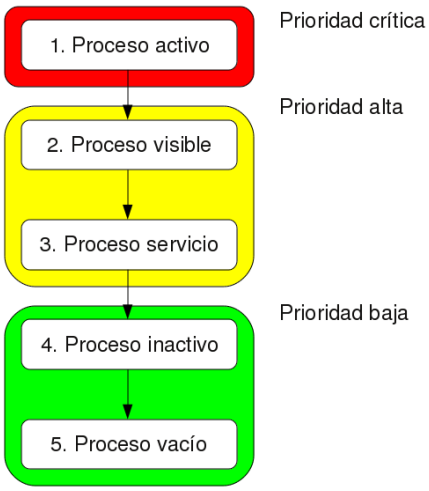
Conforme avancemos en éste y otros módulos iremos añadiendo nuevos elementos al archivo `AndroidManifest.xml`.

2.2. El ciclo de ejecución de una aplicación Android

Al contrario que en otros entornos, las aplicaciones Android tienen muy poco control sobre su propio ciclo de ejecución. Los componentes de una aplicación Android deberían estar atentos a los cambios producidos en el estado de la misma y reaccionar a los mismos como corresponda, estando especialmente preparados para el caso de una finalización repentina de la ejecución de la aplicación.

Por defecto cada aplicación Android se ejecutará en su propio proceso, cada uno con su propia instancia asociada de Dalvik (la máquina virtual de Android). Android administra sus recursos de manera agresiva, haciendo todo lo posible para que el dispositivo siempre responda a la interacción del usuario, lo cual puede llevar a que muchas aplicaciones dejen de ejecutarse de manera repentina, muchas veces incluso sin un aviso previo, con el objetivo de liberar recursos para aplicaciones de mayor prioridad. Estas aplicaciones de mayor prioridad suelen ser normalmente aquellas que están interactuando con el usuario en ese preciso instante.

El orden en el que los procesos de las aplicaciones son detenidos viene determinado por la prioridad de las mismas, la cual a su vez es equivalente a la prioridad de su componente de mayor prioridad. Cada uno de los estados en los que se puede encontrar una aplicación se resume en la siguiente figura y se detalla a continuación:



Ciclo de ejecución de los procesos en Android

- **Procesos activos:** son aquellos procesos que contienen aplicaciones que se encuentran interactuando con el usuario en ese preciso instante. Android liberará recursos para intentar que estos procesos activos siempre respondan sin latencia. Los procesos activos sólo serán detenidos como último recurso.
- **Procesos visibles**: procesos visibles pero inactivos, ya sea porque sus correspondientes aplicaciones se están mostrando detrás de otras o porque no están respondiendo a ninguna entrada del usuario. Esto sucede cuando una Actividad se encuentra parcialmente oculta por otra actividad, ya sea porque ésta última es transparente o no ocupa toda la pantalla. Estos procesos son detenidos tan solo bajo condiciones extremas.
- **Procesos asociados a servicios en ejecución**: los servicios permiten que exista procesamiento sin necesidad de que exista una interfaz de usuario visible. Debido a que estos servicios no interactúan directamente con el usuario, reciben una prioridad ligeramente inferior a la de los procesos visibles. Sin embargo se siguen considerando procesos activos y no serán detenidos a menos que sea estrictamente necesario.
- **Procesos inactivos**: se trata de procesos que albergan actividades que ni son visibles ni se encuentran realizando un procesamiento en este momento, y que además no están ejecutando ningún servicio. El orden en el que se detendrán estos procesos vendrá determinado por el tiempo que éstos llevan inactivos desde la última vez que fueron visibles, de mayor a menor.
- **Procesos vacíos**: son el resultado del intento de Android de retener aplicaciones en memoria una vez que éstas han terminado a modo de caché. Con esto se consigue que al lanzar de nuevo la aplicación se requiera menos tiempo.

3. Recursos

Se suele considerar una buena práctica de programación mantener todos los recursos de la

aplicación que no sean código fuente separados del propio código, como imágenes, cadenas de texto, etc. Android permite externalizar recursos de diversos tipos, no sólo los comentados anteriormente, sino que recursos más complejos como los `layouts`, o lo que es lo mismo, la especificación de la interfaz gráfica de las diferentes actividades.

Una ventaja adicional de externalizar los recursos es que se trata de un mecanismo simple para proporcionar valores diferentes a estos recursos dependiendo del hardware o del idioma del usuario. Si lo hacemos todo de manera correcta, será Android el encargado, al iniciar una actividad, de seleccionar los recursos adecuados.

3.1. Creación de recursos

Todos los recursos de la aplicación se almacenan bajo la carpeta `res/` del proyecto. Dentro de esta carpeta encontrarmos diferentes subcarpetas para distintos tipos de recursos.

Existen nueve tipos principales de recursos que tendrán su propia subcarpeta: valores simples, `Drawables`, `layouts`, animaciones, etilos, menús, `searchables`, XML y recursos `raw`. Al compilar nuestra aplicación estos recursos serán incluidos en el paquete *apk* que puede ser instalado en el dispositivo.

Durante el proceso de compilación se generará también una clase `R` que contendrá referencias a cada uno de los recursos. Esto nos permitirá referenciar a los recursos desde nuestro código fuente. En los ejercicios veremos ejemplos de esto. Por otra parte, a lo largo del curso iremos haciendo uso de algunos de estos recursos que se han comentado anteriormente. Iremos aprendiendo su uso sobre la marcha.

3.2. Unas pocas palabras sobre la creación de recursos para diferentes idiomas y configuraciones de hardware

En Android es posible preparar nuestra aplicación para poder ser ejecutada haciendo uso de diferentes idiomas o de configuraciones de hardware. Para ello definiremos archivos de recursos específicos. Android escogerá en tiempo de ejecución el archivo o archivos de recursos adecuados. Para conseguirlo definimos una estructura paralela de directorios dentro de la carpeta `res`, haciendo uso del guión `-` para indicar las diferentes alternativas de recursos que se están proporcionando. En el siguiente ejemplo se hace uso de una estructura de carpetas que permita tener valores por defecto para las cadenas, así como cadenas para el idioma francés y el francés de Canadá:

```
Project/
   res/
      values/
         strings.xml
      values-fr/
         strings.xml
      values-fr-rCA/
         strings.xml
```

Aparte de poder definir recursos para diferentes lenguajes utilizando especificadores como los que acabamos de ver (en, en-rUS, es, etc.), podría ser interesante conocer algunos otros referidos a hardware. En el siguiente listado se proporcionan algunos ejemplos:

- **Tamaño de pantalla:** `small` (para resoluciones menores que HVGA), `medium` (resoluciones al menos hasta HVGA pero menores que VGA) y `large` (para VGA o resoluciones mayores).
- **Orientación de la pantalla**: los valores pueden ser `port` (formato vertical), `land` (formato horizontal) o `square`.
- **Anchura/longitud de la pantalla**: usaremos `long` o `notlong` para recursos pensados especialmente para pantalla ancha (por ejemplo, para WVGA usaríamos `long` y para QVGA `notlong`).
- **Densidad de la pantalla**: medida en puntos por pulgada (*dots per inch* o *dpi*). Lo mejor es usar los valores `ldpi` para baja densidad (120dpi), `mdpi` para media densidad (160dpi) y `hdpi` para alta densidad (240dpi).

Aviso:

Si no se encuentra un directorio de recursos que se corresponda con la configuración del dispositivo en la que se está ejecutando la aplicación, se lanzará una excepción al intentar acceder al recurso no encontrado. Para evitar esto se debería incluir una carpeta por defecto para cada tipo de recurso, sin ninguna especificación de idioma, configuración de la pantalla, etc.

4. Actividades

Para crear las diferentes ventanas de interfaz de nuestra aplicación deberemos crear subclases de `Activity`. Cada actividad contendrá objetos de la clase `View` que permitirán mostrar los diferentes elementos gráficos de la interfaz gráfica así como añadir interactividad. En este sentido, se podría interpretar que cada actividad es como si fuera un formulario.

Deberemos añadir una nueva actividad por cada pantalla que queramos que pueda mostrar nuestra aplicación. Esto incluye la ventana principal de nuestra aplicación, la primera que se mostrará al iniciar nuestro programa, y desde la cual podremos acceder a todas las demás. Para movernos entre pantallas comenzaremos una nueva actividad (o volveremos a una anterior desde otra previamente ejecutada). La mayoría de las actividades están diseñadas para ocupar toda la pantalla, pero es posible crear actividades "flotantes" o semitransparentes.

4.1. Creando actividades

Para crear una nueva actividad añadimos a nuestra aplicación una subclase de `Activity`. Dentro de esta clase se deberá definir la interfaz gráfica de la actividad e implementar su

funcionalidad. El esqueleto básico de una actividad se muestra a continuación:

```
package es.ua.jtech.android;

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

public class MiActividad extends Activity {
   /** Método invocado al crearse la actividad **/
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   }
}
```

Para añadir la actividad a la aplicación no basta con crear la clase correspondiente, sino que además deberemos registrarla en el Manifest. Para ello añadiremos un nuevo nodo `<activity>` dentro del elemento `<application>`. Esto es importante, porque una actividad que no haya sido incluida en el Manifest de esta forma no podrá ser mostrada por nuestra aplicación. Los atributos de `<activity>` permiten incluir información sobre su icono, los permisos que necesita, los temas que utiliza, etc. A continuación tenemos un ejemplo de este tipo de elemento:

```
<activity android:label="@string/app_name"
             android_name=".MiActividad">
</activity>
```

Como parte del contenido del elemento `<activity>` incluiremos los nodos `<intent-filter>` necesarios para indicar los `Intent` a los que escuchará nuestra aplicación y por lo tanto a los que reaccionará. Los `Intent` serán tratados más adelante, pero es necesario destacar que para que una actividad sea marcada como actividad principal (y por lo tanto, como la primera actividad que se ejecutará al inicarse nuestra aplicación) debe incluir el elemento `<intent-filter>` tal cual se muestra en el siguiente ejemplo:

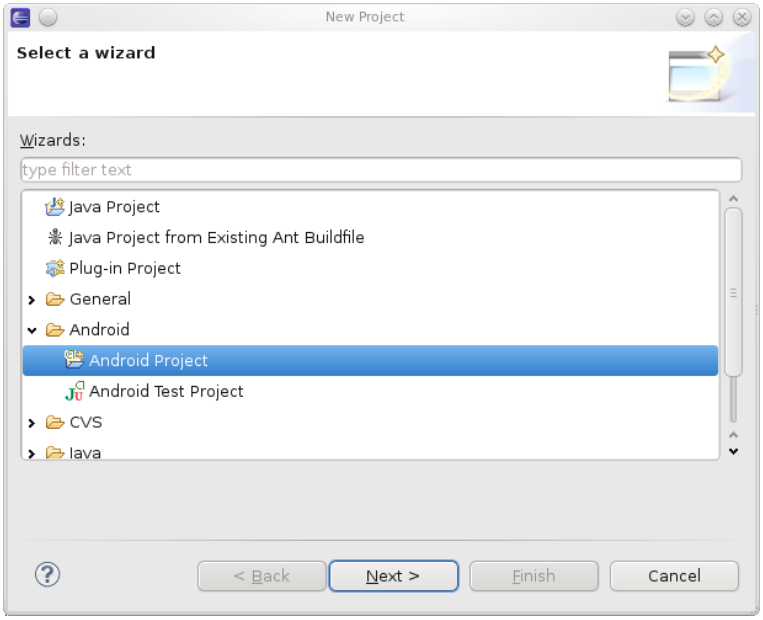
```
<activity android:label="@string/app_name"
             android_name=".MiActividad">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

5. Nuestra primera aplicación

En esta sección veremos cómo unir todo lo visto anteriormente para crear nuestra primera aplicación Android. La aplicación constará de una única actividad, que mostrará un botón cuya etiqueta será un número. Cada vez que pulsemos el botón se incrementará el valor del número.

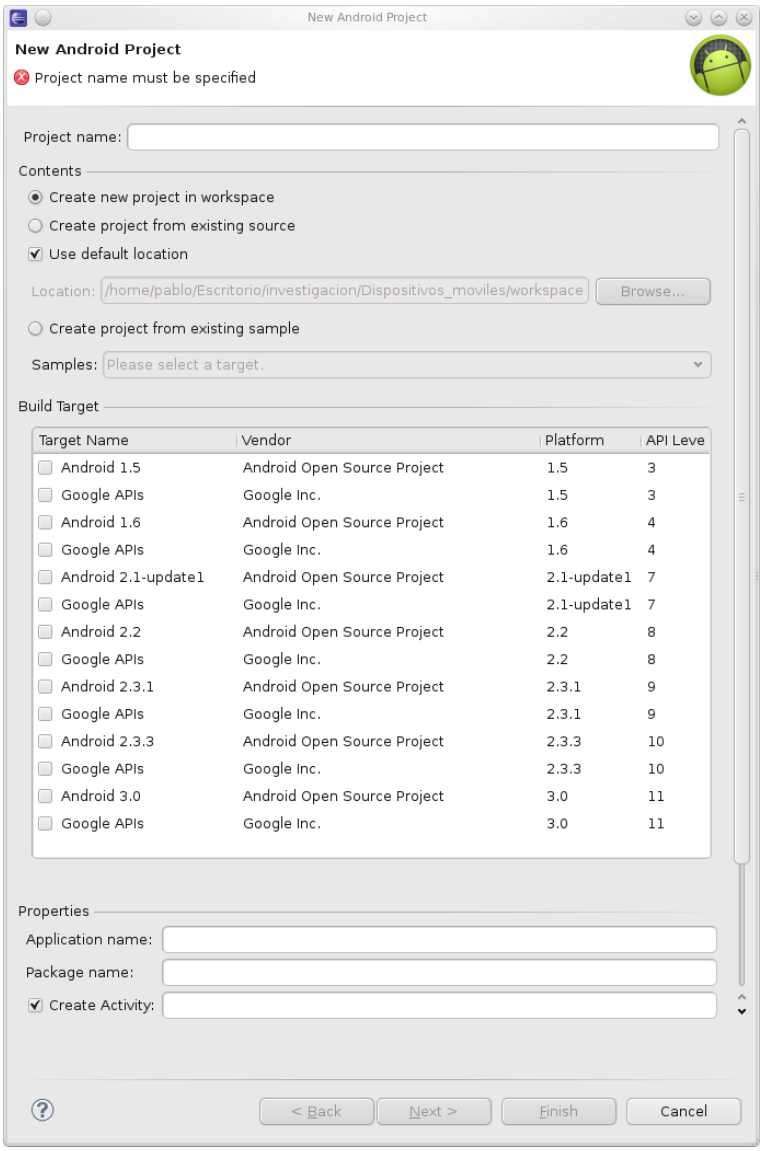
5.1. Creando el proyecto

En primer lugar vamos a crear el proyecto. Para ello, en Eclipse, seleccionamos la opción `Project` dentro del submenú `New...` que podemos encontrar bajo el menú `File`. Entre todos los tipos de proyecto a crear, seleccinamos `Android Project` dentro de la categoría `Android`.



Seleccionando el tipo de proyecto a crear (proyecto Android)

A continuación se mostrará una ventana en la que deberemos cuáles son las opciones del proyecto, como por ejemplo la versión del SDK de Android que utilizaremos, su nombre, su actividad principal, etc.



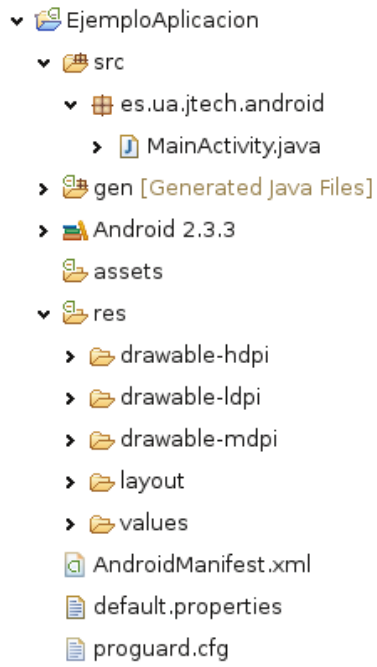
Dándole valor a los parámetros del proyecto

En esta ventana de opciones rellenaremos los siguientes campos (el resto de campos se dejarán con su valor por defecto):

- **Project name:** EjemploAplicacion
- **Build target:** Android 2.3.3
- **Application name:** Ejemplo de aplicación
- **Package name:** es.ua.jtech.android
- **Create activity:** MainActivity

Tras pulsar `Finish` se creará el proyecto `EjemploAplicacion` y podremos consultar su estructura de ficheros y directorios en el explorador de paquetes que aparece en la parte

izquierda de la interfaz de Eclipe. En el directorio raíz del proyecto tendremos el archivo `AndroidManifest.xml` con toda la información acerca de la aplicación. También veremos qeu se ha creado un paquete llamado `es.ua.jtech.android` dentro de la carpeta `src`. Dicho paquete contiene tan solo la definición de una clase, la de la actividad `MainActivity`. Obsérvese como también se ha generado la carpeta `res` que servirá para almacenar los recursos de la aplicación, y la carpeta `gen`, que contiene código autogenerado, y que por lo tanto nosotros no debemos tocar.



El contenido de nuestro nuevo proyecto

El archivo `AndroidManifest.xml` contiene el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
             package="es.ua.jtech.android"
             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=".MainActivity"
                  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>
```

Obsérvese el uso del elemento `<uses-sdk>`, con cuyo atributo `android:minSdkVersion` especifica la versión mínima del sistema Android que se le va a exigir al dispositivo móvil de destino para poder ejecutar la aplicación. El elemento `<application>` tiene dos atributos que permiten indicar el icono que se va a usar en la aplicación (atributo `android:icon`) y el nombre de la misma, el cual se mostrará en la pantalla del dispositivo (atributo `android:label`). El valor de ambos atributos tiene una sintaxis bastante peculiar. Esto es así porque lo que se está indicando es que el valor de dichos recursos se obtendrá a partir de los recursos de la aplicación, lo cual es algo que explicaremos un poco más adelante. Por último, el archivo Manifest indica que la aplicación constará de una única actividad, de nombre `MainActivity`, que además será la actividad principal (tal como se especifica por medio del uso del elemento `<intent-filger>`, tal como se ha explicado anteriormente).

5.2. Definiendo los recursos de la aplicación

Vamos ahora a echarle un vistazo al contenido de algunos de los recursos creados con la aplicación. Recuerda que todos los recursos se guardan bajo la carpeta `res`. Dentro de esta carpeta existe a su vez otra llamada `values`, y en su interior se almacena el archivo `strings.xml`. Este fichero sirve para almacenar las cadenas de caracteres (y otros elementos similares) que utilizaremos en nuestra aplicación, como por ejemplo las que se van a mostrar en la interfaz de las distintas actividades. El objetivo de disponer de un fichero `strings.xml` es poder crear aplicaciones independientes del idioma, de tal forma que podríamos disponer de diferentes carpetas *values* con sus correspondientes ficheros *strings.xml*, permitiendo al sistema operativo del dispositivo móvil escoger entre una u otra dependiendo del idioma en el que se encuentre configurado. El contenido del archivo `strings.xml` una vez creado el proyecto es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World, MainActivity!</string>
   <string name="app_name">Ejemplo de aplicación</string>
</resources>>
```

El archivo inicialmente creado contiene dos elementos de tipo `<string>`. El contenido de cada uno de esos elementos es una cadena de caracteres, y a cada una de ellas se le asigna un identificador mediante el atributo `name`. Fíjate en que el identificador de la segunda cadena, que se corresponde con el nombre que le dimos a la aplicación en las opciones del proyecto, tiene como identificador `app_name`. Y este es precisamente el identificador que se usaba en el archivo Manifest para indicar cuál era la etiqueta de nuestra actividad y nuestra aplicación. Así que, por ejemplo, cuando en el archivo Manifest estamos dándole al atributo `android:lagel` del elemento `<activity>` el valor `@string/app_name`, en realidad estamos indicando que queremos que el valor de dicho atributo sea la cadena de caracteres almacenada en el archivo `strings.xml` cuya etiqueta sea `app_name`.

La otra cadena se utiliza para indicar qué texto aparecerá en el cuadro de texto que se muestra por defecto en la actividad inicial creada cuando iniciamos un nuevo proyecto. Se

referencia en el archivo `main.xml` de la carpeta `layouts` en los recursos. De momento la ignoramos.

En la carpeta de recursos de la aplicación disponemos también de tres carpetas para almacenar elementos gráficos (elementos `drawable`) a tres diferentes resoluciones: alta (`drawable-hdpi`), media (`drawable-mdpi`) y baja (`drawable-ldpi`). Al crear el proyecto todas estas carpetas contienen un archivo con el mismo nombre: `icon.png`. Dicho archivo contendrá el icono de nuestra aplicación. En el archivo Manifest dábamos al atributo `android:icon` del elemento `<application>` el valor `@drawable/icon`. Esto significa que queremos que se utilice como icono de la aplicación la imagen almacenada con el nombre `icon` (obsérvese como se omite la extensión del fichero) en las carpetas `drawable`. Deberemos pues proporcionar a la aplicación tres copias de la misma imagen `icon.png` pero a tres resoluciones diferentes, que deberemos guardar en la carpeta correspondiente (`drawable-hdpi`, `drawable-mdpi` o `drawable-ldpi`). El sistema Android será el encargado de mostrar una imagen u otra según la resolución de la pantalla del dispositivo.

Un últimpo tipo de recursos que nos puede ser útil, sobre todo al principio del curso, son los `layouts`. Se trata de ficheros XML almacenados en la carpeta `layout` de los recursos de la aplicación, y que nos permitirán definir la disposición de las vistas en las interfaces gráficas de cada una de las actividades de la aplicación. Aprenderemos más sobre este tema en posteriores sesiones.

5.3. La actividad principal

Finalmente vamos a examinar el código de nuestra actividad principal, a la que hemos llamado `MainActivity`, y vamos a modificar dicho código para añadir la funcionalidad deseada a nuestra aplicación. El código de la actividad `MainActivity` es el siguiente:

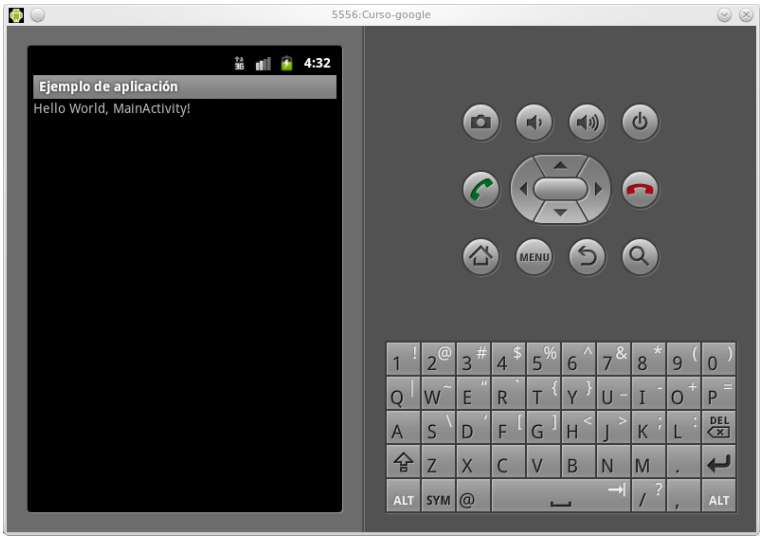
```
package es.ua.jtech.android;

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

public class MainActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
}
```

La clase `MainActivity` hereda de la clase `Activity`, que entre otras cosas establece cómo debe ser el comportamiento de una actividad durante el ciclo de ejecución de la aplicación. El único método implementado inicialmente es una sobrecarga del método `onCreate()`, que se podría interpretar como un constructor: contiene el código que se ejecutará al crearse la actividad. De momento lo único que contiene es una llamada al método `onCreate()` de la clase padre, y una llamada al método `setContentView()` que

utiliza el archivo `main.xml` de la carpeta `layout` de los recursos para definir qué vistas tendrá la interfaz gráfica de la aplicación.

El parámetro de `setContentView()` es un **identificador de recurso**, y es un ejemplo de la forma que tendremos de referenciar a los diferentes recursos desde nuestro código Android. El objeto R se genera automáticamente por el SDK a partir de los recursos de la aplicación. Así pues, al utilizar el valor `R.layout.main` estamos accediendo al identificador asociado al archivo `main.xml` dentro de la carpeta `layout` de los recursos.

Si ejecutamos ahora nuestra aplicación, el aspecto de la misma será el que se muestra a continuación. Lo único que contiene la interfaz de la actividad `MainActivity` es un campo de texto (una vista de la clase `TextView`) que muestra la cadena del archivo `strings.xml` cuyo identificador es `hello`.



Aspecto de nuestra aplicación antes de realizar ninguna modificación

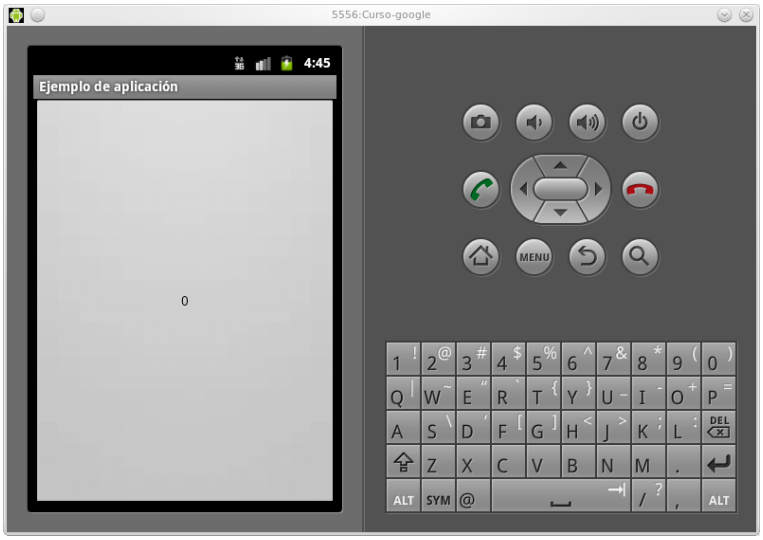
Veamos ahora cómo añadir un botón. Más adelante veremos una forma más correcta de hacerlo, pero de momento añadiremos el botón por medio de código. Nuestro método `onCreate()` debería quedar de la siguiente forma:

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   boton = new Button(this);
   boton.setText(R.string.texto_boton);
   setContentView(boton);
}
```

donde boton es un objeto de la clase `Button` definida como un atributo de la clase y donde `R.string.texto_boton` hace referencia a una cadena con identificador `text_boton` que habremos añadido al final del archivo `strings.xml` de los recursos de la aplicación:

```
<string name="texto_boton">0</string>
```

El aspecto que tendrá ahora nuestra aplicación será el siguiente:



Nuestra aplicación tras añadir un botón

Sólo queda añadirle funcionalidad al botón, de tal forma que cada vez que se pulse, se sume 1 al valor que contiene. Esto lo haremos añadiendo al botón un manejador del evento `onClick`:

```
boton.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
      int numero = Integer.parseInt(boton.getText().toString());
      numero++;
      boton.setText(new Integer(numero).toString());
   }
});
```

El código completo de la actividad `MainActivity` sería el siguiente:

```
package es.ua.jtech.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
    Button boton;

        /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        boton = new Button(this);
        boton.setText(R.string.texto_boton);
        setContentView(boton);

        boton.setOnClickListener(new OnClickListener() {
                        public void onClick(View v) {
                                int numero =
```

```
Integer.parseInt(boton.getText().toString());
                                numero++;
                                boton.setText(new
Integer(numero).toString());
                            }
        });
    }
}
```

6. El ciclo de ejecución de una actividad

Terminamos esta primera sesión incidiendo nuevamente en la administración que lleva a cabo Android de sus diferentes elementos ejecutables. Hemos hablado anteriormente en esta sesión de la ejecución de aplicaciones; veamos ahora cómo se administra la ejecución de las diferentes actividades dentro de una aplicación.

Conforme se produce la ejecución de una determinada aplicación irá modificandose el estado de sus correspondientes actividades. El estado de una actividad servirá para determinar su prioridad en el contexto de su aplicación padre. Y esto es importante, porque hemos de recordar que la prioridad de una aplicación, y por lo tanto, la probabilidad de que dicha aplicación sea detenida en el caso en el que sea necesario liberar recursos del sistema, dependerá de cuál sea la de su actividad de mayor prioridad.

6.1. Pilas de actividades

El estado de cada actividad viene determinado por su posición en la pila de actividades, una colección de tipo `last-in-first-out` que contiene todas las actividades de la aplicación actualmente en ejecución. Cuando comienza una nueva actividad, aquella que se encontrara mostrándose en ese momento se mueve al tope de la pila. Si el usuario pulsa el botón que permite volver a la actividad anterior o se cierra la actividad que se estuviera mostrando en ese determinado momento, la actividad que se encontrara en el tope de la pila sale de ella y pasa a ser la actividad activa.

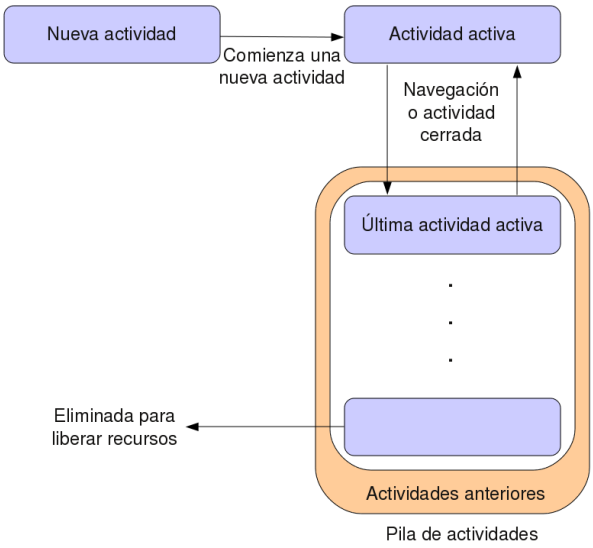


Diagrama de la pila de actividades

La pila de actividades será también usada por el sistema en el caso en el que se quiera determinar la prioridad de una actividad a partir de la prioridad de sus diferentes actividades.

Conforme las actividades se crean o destruyen van entrando o saliendo de la pila. Al hacerlo pueden ir transitando entre cuatro diferentes estados:

- **Activa**: se trata de la actividad que se está ejectuando en ese momento: es visible, tiene el foco de la aplicación y es capaz de recibir datos por parte del usuario. Android tratará por todos los medios de mantener esta actividad en ejecución, deteniendo cualquier otra actividad en la pila siempre que sea necesario.
- **En pausa**: se trata del estado en el que se encuentra una actividad cuando ésta está activa pero no dispone del foco. Este estado se puede alcanzar por ejemplo cuando se encuentra situada por debajo de otra transparente o que no ocupe toda la pantalla. Una actividad en pausa recibe el mismo tratamiento que una actividad activa, con la única diferencia de que no recibe eventos relacionados con la entrada de datos.
- **Detenida**: este es el estado en el que se encuentra una actividad que no es visible en ese momento. La actividad permanece en memoria, manteniendo toda su información asociada. Sin embargo, ahora la actividad podría ser escogida para ser eliminada de la memoria en el caso en el que se requieran recursos en otra parte del sistema. Por eso es importante almacenar los datos de una actividad y el estado de su interfaz de usuario cuando ésta pasa a estar detenida.
- **Inactiva:** una actividad estará inactiva si se ha terminado su ejecución o si todavía no se ha iniciado durante la ejecución de la aplicación. Las actividades inactivas han sido extraídas de la pila de actividades y deben ser reiniciadas para poder ser mostradas y utilizadas.

Todo este proceso debe ser transparente al usuario. No debería haber ninguna diferencia entre actividades pasando a un estado activo desde cualquiera de los otros estados. Por lo tanto puede ser interesante hacer uso de los diferentes manejadores de eventos relativos al cambio de estado de una actividad para almacenar los datos de la misma cuando pasa a estar detenida o inactiva y para volver a leerlos cuando ésta pasa a estar activa. Para manejar estos diferentes eventos podemos sobrecargar las siguientes funciones:

```
// Equivalente a un constructor
// Recibe un objeto conteniendo el estado de la interfaz de usuario
// guardada en la anterior llamada a onSaveInstanceState
public void onCreate(Bundle savedInstaceState)

// Se puede utilizar en lugar de la anterior durante el proceso
// de restaurar el estado de la interfaz de usuario
public void onRestoreInstanceState(Bundle savedInstanceState)

// Llamada cuando la actividad pasa a estar visible
public void onStart()

// Llamada antes de cualquier llamada a onStart, excepto la primera vez
public void onRestart()

// Cuando una actividad pasa a estar activa
public void onResume()

// Cuando una actividad deja de estar activa
public void onPause()

// Inmediatamente antes de llamar a onPause
public void onSaveInstanceState(Bundle savedInstanceState)

// Llamada cuando la actividad deja de estar visible
public void onStop()

// Equivalente a un destructor
public void onDestroy()
```