Introducción a Android

Índice

1. Android

Android es un sitema 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 bibliotecas (libraries) propietarias. Especiales complicaciones pueden surgir cuando se trabaja con hardware específico, como GPS, trackballs o touchscreens, etc.

Java ME abstrae completamente al programador del hardware, pero su limitación de máquina virtual le recorta mucho la libertad para acceder al hardware del dispositivo.

Esta situación motivó la aparición de Android, cuya primera version oficial (la 1.1) se publició 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 que se basaba en el núcleo de Linux 2.6.27 hasta la versión 4.0.x que está orientada tablets y a teléfonos móviles. Cada versión del sistema operativo tiene un nombre inspirado en la repostería, que cumple un orden alfabético con respecto al resto de versiones de Android (Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, 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:

- Una comunidad de desarrollo, gracias a sus completas APIs y documentación ofrecida.
- Desarrollo desde cualquier plataforma (Linux, Mac, Windows, etc).
- Un sistema operativo para cualquier tipo de dispositivo móvil, al no estar diseñado para un sólo tipo de móvil.

- Posibilidad para cualquier fabricante de diseñar un dispositivo que trabaje con Android, y la posibilidad de abrir el sistema operativo y adaptarlo o extenderlo para su dispositivo.
- Valor añadido para los fabricantes de dispositivos: las empresas se ahorran el coste de desarrollar un sistema operativo completo para sus dispositivos.
- Valor añadido para los desarrolladores: los desarrolladores se ahorran tener que programar APIs, entornos gráficos, aprender acceso a dispositivos hardware particulares, etc.

¿De qué está hecho Android?

- Núcleo basado en el de Linux para el manejo de memoria, procesos y hardware. (Se trata de un branch, 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 versions de Flash Player.
- Muy importante es la exitencia del Android Market, y más todavía la presencia de una comunidad de desarrolladores que suben ahí aplicaciones, tanto de pago como gratuitas. De cara al usuario, el verdadero valor del sistema operativo está en las aplicaciones que se puede instalar.

¿Quién desarrola Android?

La Open Handset Alliance. 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

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

1.2.1. 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 el progreso en la programación de dispositivos móviles y en su fabricación, se abarata y 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. Esto no es un problema de seguridad, los problemas de seguridad ocurren 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.

1.3. Dispositivos

Los dispositivos con Android son numerosos. Aunque los hay de tipos muy diferentes, los principales son los tablets y los teléfonos móviles. La configuración típica es la de pantalla táctil y algunos botones para el menú, salir, home, apagar, volumen. Muchos dispositivos también cuentan con un teclado físico, y algunos incluso cuentan con varias pantallas.

Teléfono de desarrollador de Android

Motorla Charm

Tablet Dawa

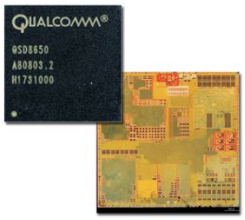
HTC Evo 4G

Sony Ericsson Xperia Mini

La mayoría de los dispositivos diseñados para Android utilizan procesadores con arquitectura ARM. ARM significa Advanced RISC Machine y, como su nombre indica, se trata de arquitecturas RISC (Reduced Instruction Set Computer) y son procesadores de

32 bits. Aproximadamente el 98% de los teléfonos móviles usan al menos un procesador basado en arquitectura ARM.

Qualcomm es el fabricante del procesador Snapdragon, procesador basado en ARM que se está utilizando en los últimos dispositivos móviles del mercado. Realmente es una plataforma que en algunos de sus modelos incluye dos CPUs de 1.5 GHz, HSPA+, GPS, Bluetooth, grabación y reproducción de video full definition, Wi-Fi y tecnologías de televisión móvil. Lo están utilizando no sólo los fabricantes de teléfonos, sino también los de tablets y netbooks.



Procesador Snapdragon

2. Desarrollo de aplicaciones

Aunque el desarrollo de librerías de bajo nivel es posible con el Android Native Development Toolkit, vamos a centrarnos en el desarrollo de aplicaciones con el Android Software Development Toolkit.

2.1. Android SDK

El SKD de android incluye numerosas y completas API's para facilitar el desarrollo. Algunas de las características más relevantes son:

- Licencias, distribución y desarrollo gratuitos, tampoco hay procesos de aprobación del software.
- Acceso al hardware de WiFi, GPS, Bluetooth y telefonía, permitiendo realizar y recibir llamadas y SMS.
- Control completo de multimedia, incluyendo la cámara y el micrófono.
- APIs para los sensores: acelerómetros y brújula.
- Mensajes entre procesos (IPC).
- Almacenes de datos compartidos, SQLite, acceso a SD Card.
- Aplicaciones y procesos en segundo plano.
- Widgets para la pantalla de inicio (escritorio).
- Integración de los resultados de búsqueda de la aplicación con los del sistema.
- Uso de mapas y sus controles desde las aplicaciones.
- Aceleración gráfica por hardware, incluyendo OpenGL ES 2.0 para los 3D.

Muchas de estas características ya están, de una manera o de otra, para los SDK de otras plataformas de desarrollo móvil. Las que diferencian a Android del resto son:

- Controles de Google Maps en nuestras aplicaciones
- Procesos y servicios en segundo plano
- Proveedores de contenidos compartidos y comunicación entre procesos
- No diferencia entre aplicaciones nativas y de terceros, todas se crean igual, con el mismo aspecto, y con las mismas posibilidades de usar el harware y las APIs.
- Widgets de escritorio

2.2. Capas

El núcleo de Linux es la capa encargada de los controladores (drivers) del hardware, los procesos, la memoria, seguridad, red, y gestión de energía. Es la capa que abstrae el resto de las capas del hardware.

El Android run time es lo que hace a Android diferente de una distribución de Linux embebido. Está compuesto por las librerías "core" (núcleo) y por Dalvik, la máquina virtual de Java, basada en registros que cuenta con el núcleo de Linux para la gestión de hilos y para el manejo de memoria a bajo nivel.

El framework de aplicaciones está compuesto por las clases que se utilizan para crear aplicaciones Android: actividades, servicios, views, proveedores de contenidos, etc.

Finalmente la capa de aplicaciones está compuesta por las aplicaciones nativas y por las de terceros, así como las de desarrollo.

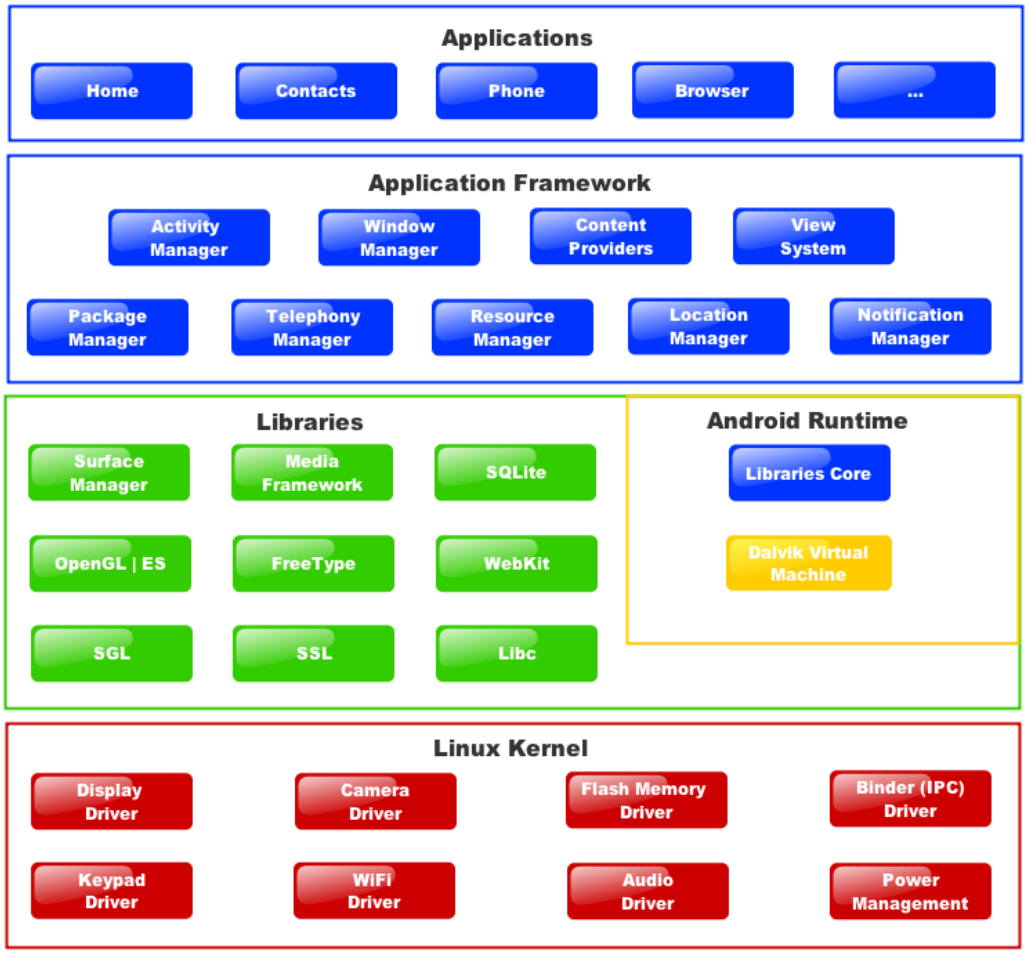


Diagrama de Android

Las clases más importantes para el desarrollo de aplicaciones en Android son las siguientes:

- `ActivityManager`: Controla el ciclo de vida de las actividades.
- `View`: Se usan para construir interfaces en las actividades.
- `NotificationManager`: Mecanismo no intrusivo para mostrar avisos al usuario.
- `ContentProvider`: Permiten intercambiar datos de una manera estandarizada.
- Resource Manager: permite usar en la aplicación recursos que no forman parte del código, como XML, strings, recursos gráficos, audio, vídeo, etc.

2.3. Tipos de aplicaciones

Hay tres tipos de aplicaciones para Android: las de primer plano (foreground), las de segundo plano (background) y los widget (o AppWidget).

Las aplicaciones de primer plano constan de actividades que muestran una interfaz de usuario. No tienen mucho control sobre su ciclo de vida, ya que en cualquier momento se les puede robar el foco, o pueden pasar a segundo plano. Si faltan recursos, tanto de memoria, como de procesador, pueden ser incluso terminadas. El objetivo del sistema operativo es que la aplicación que en este momento esté en primer plano ofrezca respuestas fluidas. Si para ello hay que terminar otras aplicaciones en segundo plano, Android lo hará.

Las aplicaciones de segundo plano se denominan servicios. Las hay de dos tipos: servicios puros, o servicios combinados con actividades que permiten configurarlos. El ejemplo típico es el de un reproductor multimedia. En cuanto hemos seleccionado qué cancion reproducir, cerramos la actividad y un servicio mantiene el audio en reproducción.

Otro tipo de aplicaciones son los widget. Se trata de apliaciones de pequeña interfaz gráfica que se colocan sobre el escritorio (home) de Android y que se refrescan cada cierto intervalo de tiempo. Normalmente detrás de ellas corre un servicio de actualización. Un ejemplo es el del reloj, o el calendario.

2.4. Consideraciones para el desarrollo

El desarrollo para dispositivos móviles y embebidos requiere que el programador tenga especial cuidado en determinados aspectos. El sistema operativo no puede encargarse de controlar todo porque limitaría demasiado al programador, así que determinados aspectos como ahorrar CPU y memoria son responsabilidad del programador en la mayoría de los casos. La limitaciones de hardware que el dispositivo impone suelen ser:

- Pequeña capacidad de procesamiento
- Memoria RAM limitada
- Memoria permanente de poca capacidad
- Pantallas pequeñas de poca resolución
- Transferencias de datos costosa (en términos de energía y económicos) y lenta
- Inestabilidad de las conexiones de datos
- Batería muy limitada
- Necesidad de terminar la aplicación en cualquier momento

A partir de estas limitaciones, las consideraciones que el desarrollador debe tener son: ser eficiente en cuanto a bucles y cálculos; asumir poca memoria y optimizar el almacenamiento al máximo, eliminando siempre los datos que ya no van a ser necesarios; diseñar para pantallas de distintos tamaños pero sobretodo tener en cuenta las más pequeñas; ahorrar comunicaciones y asumir que serán lentas y que fallarán, lo cuál no debe limitar la capacidad de respuesta de la aplicación.

También es muy importante respetar al usuario. No hay que robar el foco a otras aplicaciones, hay que usar avisos lo mínimo posible y de la manera menos intrusiva, hay que mantener un estilo de interfaz consistente con el sistema y que responda bien.

También hay que intentar que la aplicación se recupere rápidamente al rearrancarla y que se mantenga en el último estado en el que estuvo, para que el usuario no note la diferencia cuando la aplicación ha sido terminada o cuando ha continuado su ejecución tras estar en segundo plano. También, no hace falta ni decir, que el usuario no quiere gastar más batería de lo necesario y eso conciene el procesamiento y sobretodo la comunicación por red.

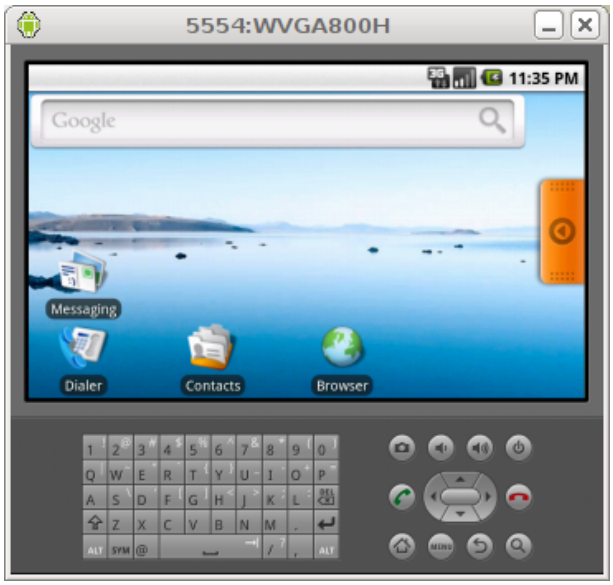
Si desarrollamos nuestras aplicaciones con responsabilidad y respetamos los consejos de desarrollo de Android (Guía de desarrollo de Android http://developer.android.com/guide/index.html, sección Best Practices) contribuiremos a que el sistema operativo Android ofrezca mejores experiencias a sus usuarios. Al fin y al cabo los usuarios juzgan a un sistema operativo por sus aplicaciones, aunque éstas sean de terceros. A los desarrolladores nos interesa que la plataforma sobre la que programamos sea utilizada por el máximo número de usuarios posible para que nuestras aplicaciones lleguen a más gente.

3. Emulador

Android SDK viene con un emulador en el que podemos probar la mayoría de nuestras aplicaciones. Desde Eclipse podemos ejecutar nuestras aplicaciones directamente en un emulador arrancado, que corre sobre un puerto. También podemos tener varios emuladores arrancados para que se comuniquen entre ellos si la aplicación lo requiere. Dentro del emulador contamos con una distribución de Android instalada con sus aplicaciones nativas y la mayoría de las funcionalidades.

Podemos simular GPS, llamadas entrantes, salientes, SMS, entradas por la pantalla y el teclado, reproducción de audio y vídeo, y comunicaciones por red. Todos los tipos de aplicaciones están soportadas: widgets, servicios y actividades. Se pueden instalar y desinstalar como si de un móvil real se tratara.

Todavía no se pueden simular conexiones por USB, captura por la cámara, auriculares o manos libres conectados al móvil, conexión o desconexión de la corriente eléctrica ni el estado de la batería, bluetooth, tampoco la inserción y extracción de la tarjeta de memoria SD, sensores, ni tampoco el multitouch.



Emulador de Android

El emulador incluye una consola a la que se puede conectar por telnet:

```
telnet localhost <puerto>
```

El puerto suele ser el 5554, como en el ejemplo de la imagen. Se indica en la barra de la ventana del emulador. Este puerto también se utiliza como número de teléfono para simular llamadas y SMS.

El emulador de Android necesita información de configuración del dispositivo virtual, denominado AVD, Android Virtual Device. Debemos crear un dispositivo AVD para poder arrancar el emulador con determinada configuración. En esta configuración debemos especificar el nivel de API que tiene el dispositivo (por ejemplo, para el caso de Android 2.2 el nivel de API es el 8), el tipo de pantalla y el resto del hardware: tarjeta SD, presencia de teclado físico, etc. Esta configuración se puede crear con el AVD Manager del plugin de Eclipse.

Toda la información sobre el emulador de Android se puede encontrar en el sitio oficial http://developer.android.com/guide/developing/tools/emulator.html. Ahí están enumerados todos los comandos de control y de configuración del emulador.

4. AndroidManifest.xml

Todos los proyectos de Android contienen un fichero `AndroidManifest.xml`. Su finalidad es declarar una serie de metadatos de la aplicación que el dispositivo debe conocer antes de instalarla. En él se indican: el nombre del paquete, el nombre de la aplicación, las actividades, servicios, receptores broadcast, proveedores de contenidos,

cuál es la actividad principal. En él también se indica el nivel mínimo de API que la aplicación requiere, y los permisos que necesita para trabajar. Los permisos serán concedidos de manera explícita por el usuario si da su consentimiento para instalar la aplicación. También se puede indicar aspectos más avanzados, como los permisos que otras aplicaciones necesitarían para poder interactuar con la nuestra. Las librerías utilizadas también se declaran en este archivo.

La estructura del `AndroidManifest.xml` es la siguiente:

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

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

Nosotros abriremos este archivo básicamente cuando creemos una nueva actividad,

cuando veamos que necesitamos permisos (para acceder a Internet, para acceder a los contactos, etc), cuando declaremos un servicio o un widget, cuando creemos un proveedor de contenidos, o cuando utilicemos una librería, como por ejemplo es la de Google Maps. No nos hará falta tocar nada para la actividad principal, ya que ésta nos la crea el asistente de Eclipse y nos añade la información necesaria en el manifest.

5. Externalizar recursos

La externalización de recursos hace la aplicación más mantenible y fácilmente personalizable y adaptable a otros idiomas. Android ofrece facilidades para la externalización de recursos. Todos los recursos de una aplicación se almacenan en la carpeta `res` del proyecto, de manera jerárquica. El asistente de Eclipse nos genera una estructura de recursos que contiene las carpetas `values`, `drawable-ldpi`, `drawable-mdpi`, `drawable-hdpi` y `layout`. En esta última se guardan los layouts para la interfaz gráfica. En la primera, `values`, se guardan strings, colores y otros recursos que contienen pares de identificador y valor. Si añadimos algún menú o preferencias, los añade en una nueva carpeta, `xml`.

Todos los recursos se guardan en archivos con el siguiente formato:

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

Cada tipo de recurso se especifica con determinado tag de XML. Algunos ejemplos de recursos son los siguientes.

- Strings. Se guardan en `res/values/strings.xml` en formato

```
<string name="saludo">¡Hola!</string>
```

- Colores. Se guardan en `res/values/colors.xml` en formato hexadecimal #RGB, #RRGGBB, #ARGB ó #AARRGGBB, como en el ejemplo:

```
<color name="verde_transparente">#7700FF00</color>
```

- Dimensiones. Se guardan en `res/values/dimensions.xml` en formato `px` (píxeles de pantalla), `in` (pulgadas), `pt` (puntos físicos), `mm` (milímetros físicos), `dp` (píxeles relativos a pantalla de 160-dpi, independientes de la densidad), `sp` (píxeles independientes a la escala), como en el siguiente ejemplo:

```
<dimen name="altura_mifuente">12sp</dimen>
```

- Arrays

```
<array name="ciudades">
        <item>Alicante</item>
        <item>Elche</item>
        <item>San Vicente</item>
</array>
```

- Estilos y temas. Permiten modificar el aspecto de los `View` pasándoles como parámetro el nombre del estilo especificado en los recursos. Por ejemplo, para especificar tamaño y color de la fuente para el estilo "EstiloTexto1":

```
<style name="EstiloTexto1">
        <item name="android:textSize">18sp</item>
        <item name="android:textColor">#00F</item>
</style>
```

Para usar recursos desde el código se puede acceder a sus identificadores con `R.id.TextView01` para los views; con `R.layout.main` para los layouts, donde `main` se correspondería con el layout `res/layout/main.xml`; con `R.string.saludo`, etc. Por ejemplo:

```
        TextView tv = (TextView)findViewById(R.id.TextView01);
        tv.setText(R.string.saludo);
```

Para usar los String desde los layouts (en XML) hay que referenciarlos como

```
        @string/nombrestring
```

De la misma forma que los layout quedan indexados entre los recursos y se pueden referenciar, también se pueden utilizar recursos de imagen, audio, vídeo. Hay que usar sólo minúsculas en los nombres.

6. Plug-in para Eclipse

6.1. Instalación

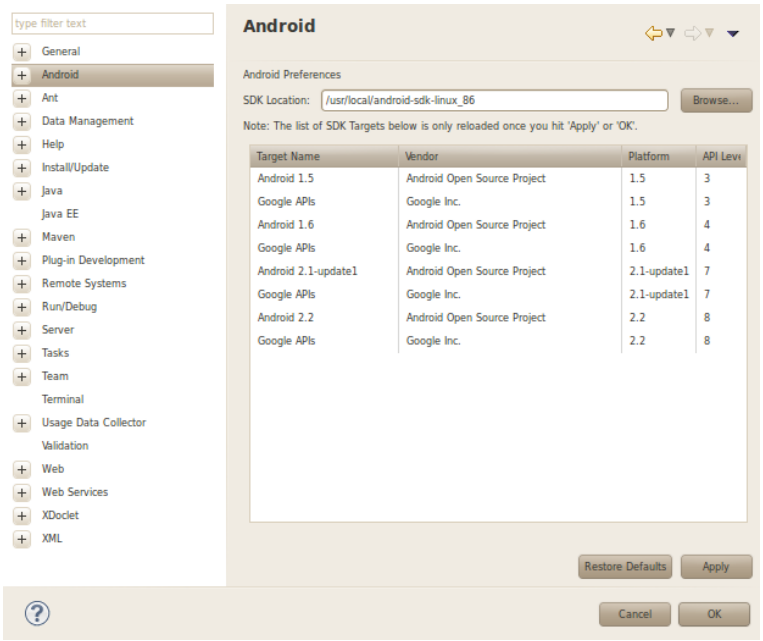
Una vez descargado y descomprimido el Android SDK, hay que instalar el plug-in para Eclipse que incluye una serie de interfaces, herramientas y asistentes, así como editores visuales de los archivos XML.

La información detallada sobre la instalación del Android ADT Plugin para Eclipse está en http://developer.android.com/sdk/eclipse-adt.html. Para instalarlo hay que ir a los menús de Eclipse: Help > Install New Software, y en Available Software pulsar Add, y añadir el sitio:

```
https://dl-ssl.google.com/android/eclipse/
```

Pulsar Ok, seleccionar el nuevo software a instalar, Next, y Finish. Hará falta reiniciar Eclipse.

Antes de empezar a usar el plugin hay que configurarlo. En Windos > Preferences, seleccionamos Android, y para la SDK Location pulsamos Browse para indicar ahí la ruta donde tenemos descomprimido el Android SDK.



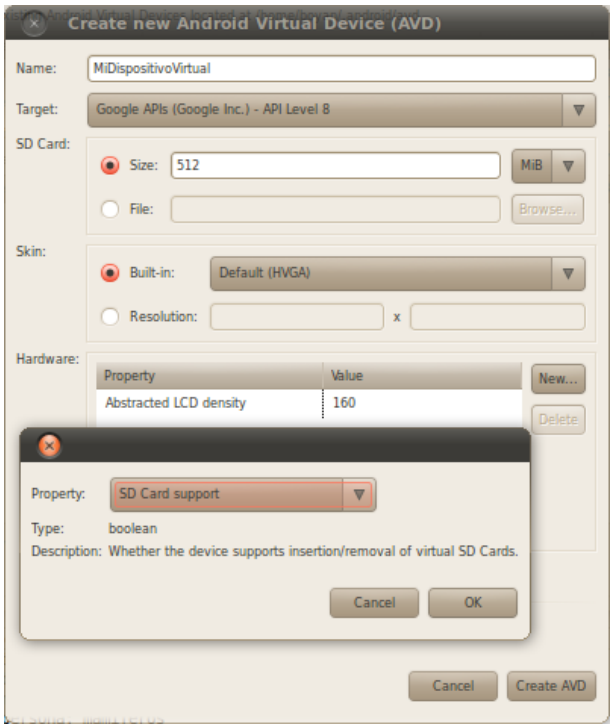
Preferencias Android en Eclipse y SDK location.

Ya está listo para usar. Sólo falta crear un dispositivo AVD para que el Emulador lo pueda arrancar cuando queramos ejecutar nuestras aplicaciones Android.

6.2. Herramientas

Vamos a enumerar algunas de las herramientas que el plug-in proporciona.

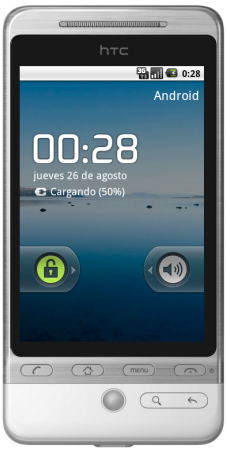
AVD Manager. Accesible desde el menú Window de Eclipse o bien desde la barra de herramientas de Eclipse. Permite crear nuevos dispositivos virtuales (AVDs) y especificar su hardware:



AVD Manager: asistente para crear un nuevo AVD; añadiendo nuevo hardware

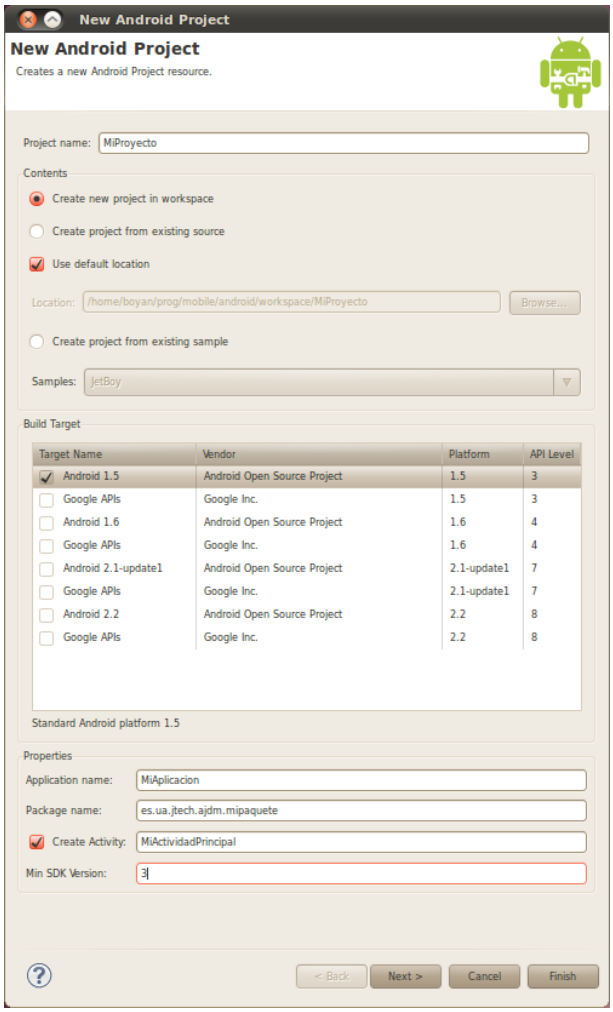
Emulador. Emula los AVDs que hemos creado. Según el AVD y según el "skin" puede tener un aspecto diferentes. Por defecto viene un único skin:

Emulador con el skin por defecto y teclado físico



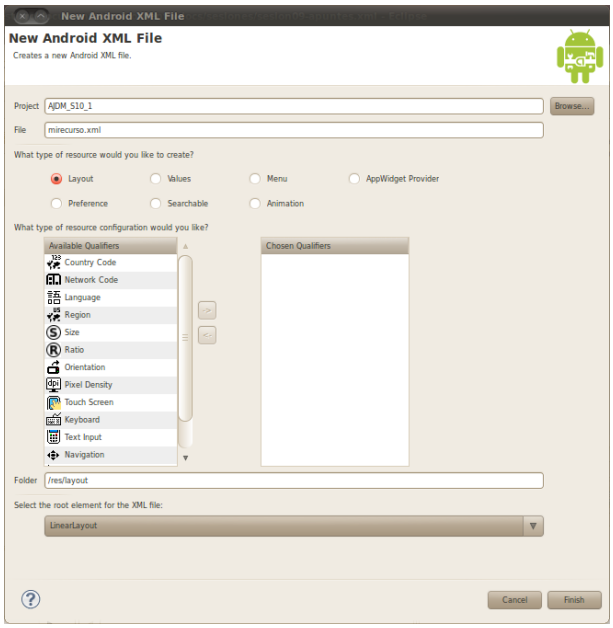
Emulador con el skin de HTC Hero

Asistente para la creación de proyectos. Nos genera la estructura de directorios básica y el AndroidManifest.xml, con una actividad principal cuyo nombre indicamos:



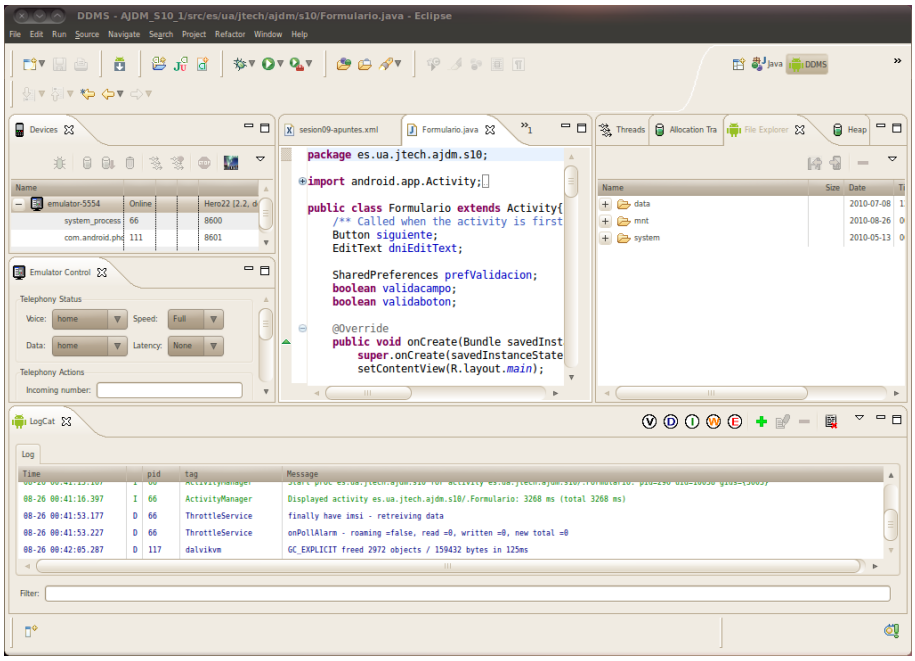
Asistente para la creación de proyectos Android.

Asistente para la creación de recursos XML. Nos ayuda a crear layouts, valores, menús, AppWidgets, preferencias y otros:



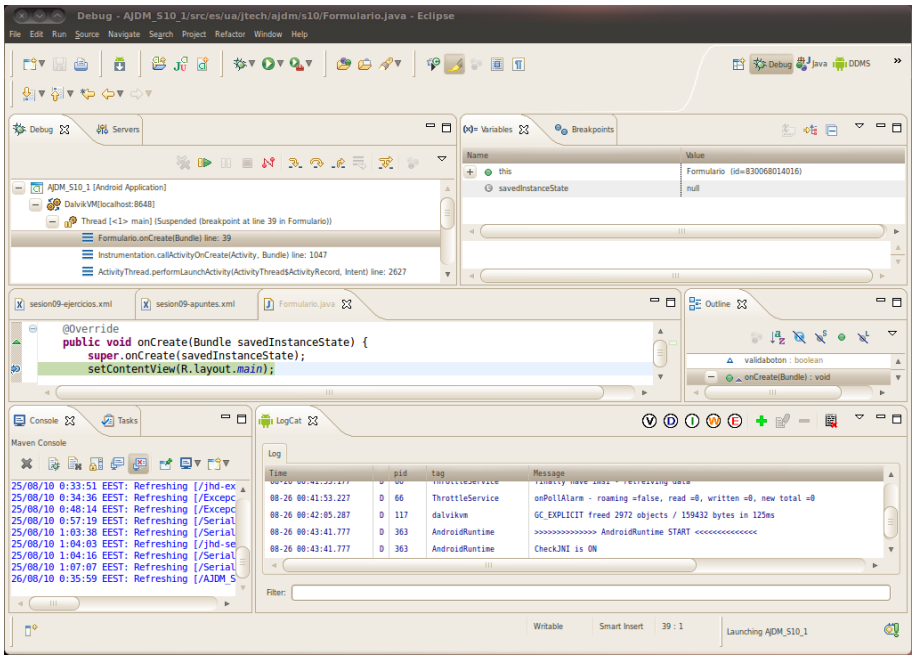
Asistente para la creación de recursos XML de Android.

Vista DDMS. Nos permite monitorizar y controlar el emulador durante la ejecución de los programas. Podemos ver el LogCat, los procesos del dispositivo emulado, los hilos, la reserva de memoria, su sistema de ficheros, y mandar algunas configuraciones al emulador, así como simular llamadas o SMS. Se accede a través de los botones de vistas de Eclipse, en la esquina superior derecha:



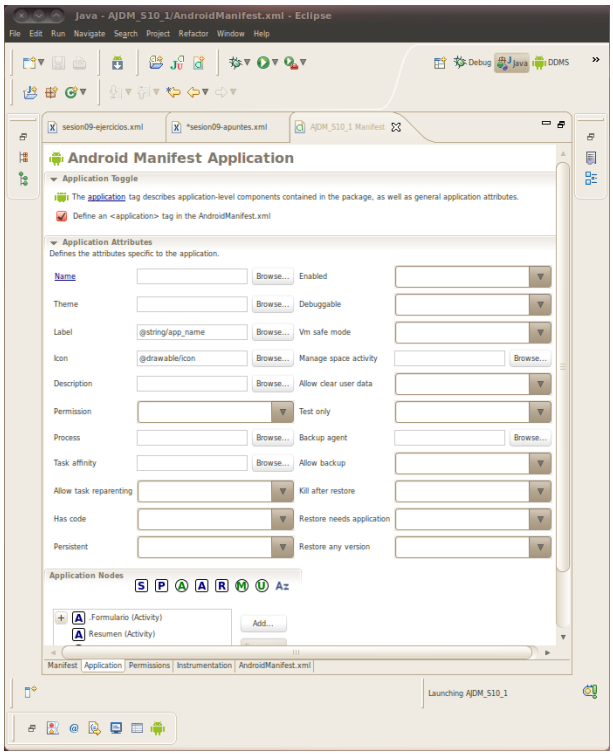
Vista DDMS

Debugging. Es otra vista de Eclipse y nos permite depurar programas emulados exactamente igual que en los programas Java tradicionales. Para que un programa sea publicable, el debugging se debe deshabilitar declarándolo en el AndroidManifest.xml, lo cuál no permitiría depurarlo desde Eclipse.



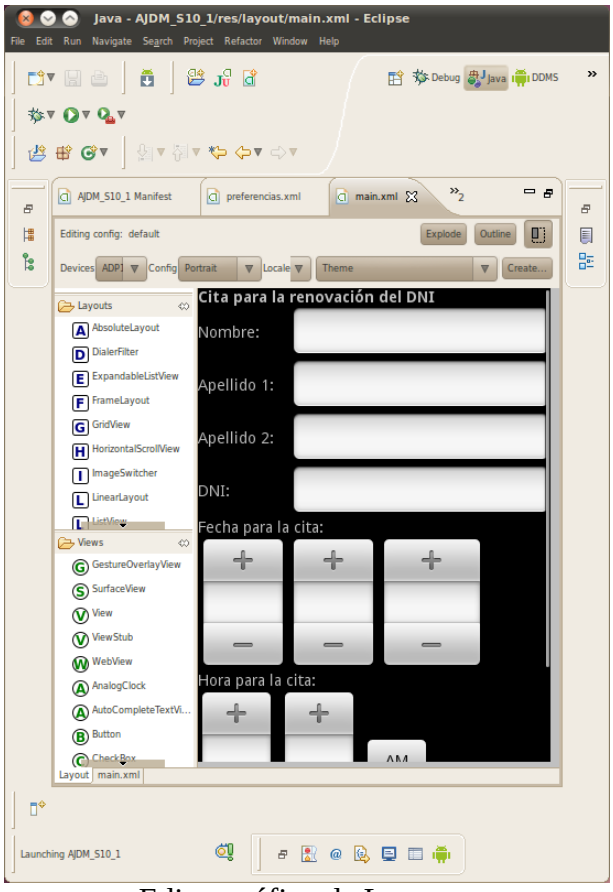
Vista Debug

Editor gráfico del AndroidManifest.xml. Cuando editamos este archivo, en la parte inferior izquierda tenemos una serie de pestañas. La última de ellas nos permite verlo en formato XML, el resto son interfaces gráficos para su edición:



Editor del AndroidManifest.xml

Editor visual de layouts. Nos permite añadir view y layouts y los previsualiza. Para pasar a modo XML, seleccionamos la pestaña correspondiente, en la parte inferior izquierda del editor:

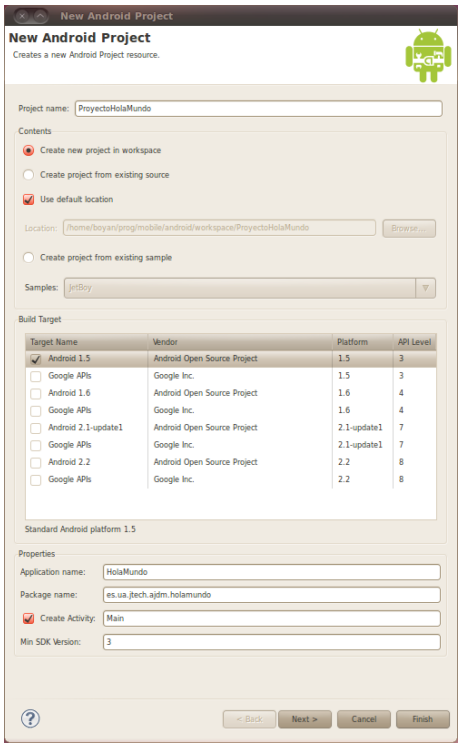


Editor gráfico de Layouts.

7. Hola, mundo

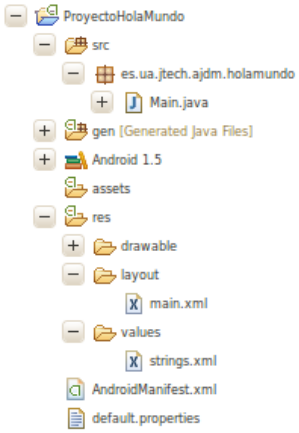
7.1. Proyecto nuevo y recursos

Creamos un New > Android project y lo titulamos `ProyectoHolaMundo`. A través del asistente damos un nombre a la aplicación, a la actividad principal y al paquete, como se ilustra en la imagen. También tenemos que seleccionar la version mínima de API necesaria, si vamos a usar Android 1.5, ésta puede ser la 3.



Asistente de creación del proyecto.

Pulsamos finalizar y se genera la siguiente estructura de directorios:



Estructura de ficheros generada.

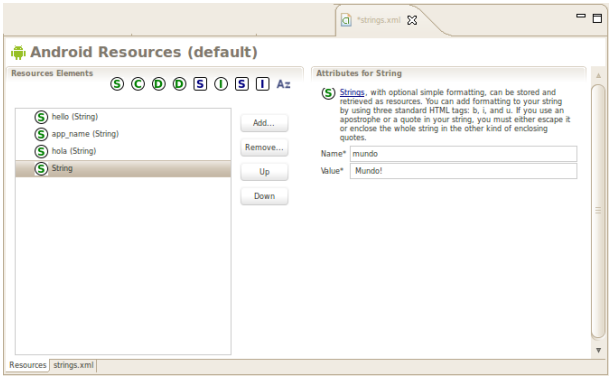
Abrimos el `AndroidManifest.xml` y observamos el código:

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

```
              android:versionName="1.0">
            <application android:icon="@drawable/icon"
android:label="@string/app_name">
                <activity android:name=".Main"
                          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>
            <uses-sdk android:minSdkVersion="3" />

        </manifest>
```

El manifest nos indica, en primer lugar, el nombre del paquete y la versión. Después indica que tenemos una aplicación con un nombre (cuyo valor está en los recursos) y un icono. En esta aplicación tenemos una actividad principal Main. Finalmente indica que se requiere como mínimo la versión 3 de la API. Si un móvil tuviera menor versión, esta aplicación no se instalaría en él.

Vamos a abrir el archivo `res/values/strings.xml` y a comprobar que el nombre de la aplicación está ahí.



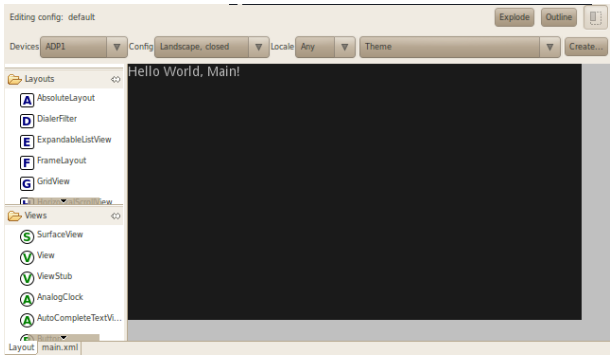
Editor de recursos de tipo string.

También hay un string "hello", que usa el layout principal, como veremos a continuación. De momento vamos a añadir los strings que vamos a utilizar en nuestra aplicación:`"Hola, "`, `" Mundo!"` y `"Hola ¿qué?"`, que se llamarán `"hola"`, `"mundo"` y `"que"` , respectivamente. El XML del archivo debe quedar así:

```
        <?xml version="1.0" encoding="utf-8"?>
        <resources>
            <string name="hello">Hello World, Main!</string>
            <string name="app_name">HolaMundo</string>
                <string name="hola">Hola, </string>
                <string name="mundo"> Mundo!</string>
                <string name="que">Hola ¿qué?</string>
        </resources>
```

7.2. Layout

Ahora podemos editar el layout `main.xml`. En vista de diseño aparece así:



Layout main.xml original.

Eliminamos la etiqueta que ya viene y en su lugar dejamos caer una nueva etiqueta `TextViewy` un botón `Button`. Si ahora miramos el layout en modo XML, veremos algo así:

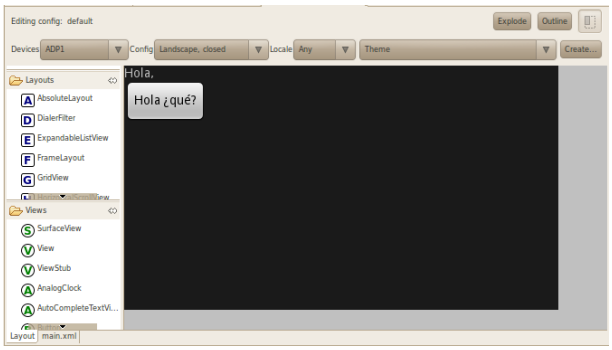
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

<TextView android:text="@+id/TextView01" android:id="@+id/TextView01"
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

<Button android:text="@+id/Button01" android:id="@+id/Button01"
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

</LinearLayout>
```

Vamos a cambiar los atributos `android:text` del `TextView` y del `Button`. En el primero pondremos el string "hola" que ya hemos definido en los strings, y en el segundo pondremos "que", también definido. El código quedará así:

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

<TextView android:text="@string/hola" android:id="@+id/TextView01"
```

```
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

<Button android:text="@string/que" android:id="@+id/Button01"
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

</LinearLayout>
```

Y en vista de diseño se verá así:



Layout main.xml tras realizar los cambios.

Ahora lo podemos ejecutar y ver el resultado en el emulador. Hay que ejecutar, o bien el proyecto, o bien el `Main.java` como Android application. En el emulador tendremos un aspecto similar a la previsualización de la imagen anterior, y el botón será pulsable, sin realizar ninguna acción.

Nota:

Es posible que el emulador se inicie con la pantalla bloqueada. La desbloquearemos como si de un móvil real se tratara. Además el emulador es lento arrancando, a veces hay que tener paciencia.

7.3. Actividad y eventos

Vamos a añadir acción al botón. Para ello vamos a editar el `Main.java` que se encuentra en la carpeta `src`, dentro del paquete que hemos especificado. Se trata de la actividad que hemos llamado `Main` (podía haber tenido otro nombre cualquiera) y hereda de `Activity`, por eso es una actividad. Tiene sobrecargado el método `onCreate(Bundle)` y en él, aparte de ejecutar el mismo método pero de la clase padre, lo que hace en la siguiente instrucción:

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

es poner en pantalla el layout especificado en el recurso `res/layout/main.xml`. En este layout tenemos un botón y una etiqueta. Para poder acceder a ellos desde la actividad

vamos a declararnos dos variables de sus respectivos tipos como campos de la actividad,

```
TextView textView;
Button   button;
```

y vamos a asignarles valores en el método `onCreate`, tras la instrucción `setContentView`:

```
textView = (TextView)findViewById(R.id.TextView01);
button = (Button)findViewById(R.id.Button01);
```

Con esto ya podemos acceder desde el código a ambos View. Empezamos asignando un `onClickListener` al botón, para que escuche los eventos de click sobre este botón en particular:

```
button.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

                }

        });
```

Nota:

En Eclipse, cuando tecleemos "`button.setOnClickListener(new `", la autocompleción (que se activa con Ctrl+spacebar) nos sugerirá el `OnClickListener` y si pulsamos Enter, nos completará la clase con el método `onClick` sobrecargado, puesto que es obligatorio. Es posible que nos marque el `OnClickListener` como error por no encontrarlo, porque no ha añadido automáticamente el import necesario. Podemos añadir el import haciendo caso a la sugerencia para solucionar el error, o bien con "Fix imports", que se consigue con Mayús+Ctrl+o. Finalmente, sólo nos faltará poner el punto y coma ";" al final de todas las llaves y paréntesis.

Dentro del método `OnClickListener.onClick(View)` ejecutamos una acción, en este caso la acción será añadir el string "mundo" al `textView` (que hasta el momento sólo contiene "hola". Así, cuando se pulse el botón, aparte de "hola" aparecerá "mundo":

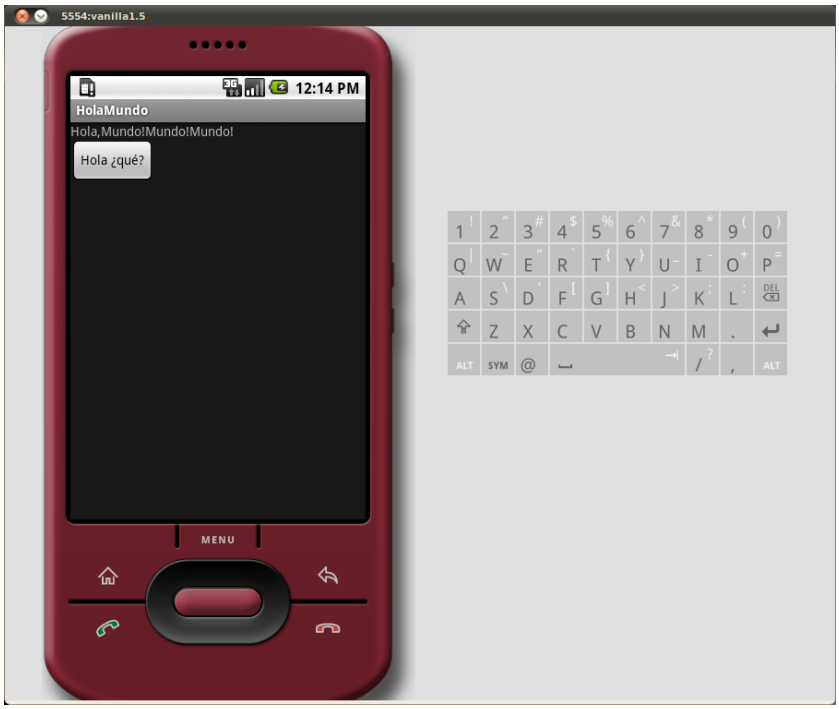
```
textView.append(getString(R.string.mundo));
```

El código completo quedaría así:

```
package es.ua.jtech.ajdm.holamundo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
```

```
public class Main extends Activity {
    /** Called when the activity is first created. */
        TextView textView;
        Button   button;

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

        textView = (TextView)findViewById(R.id.TextView01);
        button = (Button)findViewById(R.id.Button01);

        button.setOnClickListener(new OnClickListener() {

                        @Override
                        public void onClick(View v) {
                                textView.append(getString(R.string.mundo));
                        }

                });
    }

}
```

Ahora podemos volver a ejecutar y comprobar que cada vez que pulsamos el botón se añade el texto correspondiente. En la imagen se ve el resultado tras pulsar el botón tres veces:



Hola Mundo tras pulsar el botón tres veces.