Universidad de Murcia
Área de Tecnologías de la Información
y Comunicaciones Aplicadas

ATICA

Cursos de Promoción Educativa

Desarrollo de Aplicaciones para Dispositivos Móviles *Android*

android um
CURSO DE PROMOCIÓN EDUCATIVA

Autores

Antonio Bernárdez Álvarez
antonio.bernardez@um.es

Félix Gómez Mármol
felixgm@um.es

http://ants.dif.um.es/~felixgm/docencia/android

Índice general

Índice General **7**

Índice de Figuras **12**

Índice de Tablas **13**

Introducción **15**
1. Presentación . . . 15
2. Motivación . . . 16
3. Contenidos . . . 16
4. Objetivos . . . 18
5. Metodología y Evaluación . . . 19
6. Concurso organizado por Neosistec y Accesium . . . 20

Parte I. Desarrollo de Aplicaciones para Dispositivos Móviles *Android* **23**

1. Introducción a Android. Conceptos básicos **23**
1.1. Breve descripción histórica de Android . . . 23
1.2. ¿Qué es Android? . . . 26
1.3. Características del SDK de Android . . . 26
1.4. Arquitectura de Android . . . 27

2. Entorno de desarrollo Eclipse **29**
2.1. Instalación de Eclipse . . . 29
2.1.1. Requisitos del sistema . . . 29
2.1.2. Instalación de la JDK . . . 29
2.1.3. Instalación de Eclipse . . . 31
2.2. Instalación del SDK de Android . . . 33
2.3. Instalación del plugin ADT para Eclipse . . . 36
2.4. Herramientas disponibles para el desarrollo en Eclipse . . . 39

3. Nuestro primer programa **41**
3.1. Creación de la primera aplicación bajo Eclipse . . . 41
3.1.1. Configuración del emulador de Android . . . 43
3.1.2. Ejecutando nuestra aplicación . . . 45
3.2. Explorando nuestra primera aplicación . . . 48
3.2.1. Fichero `.java` . . . 48
3.2.2. Fichero `.xml` . . . 48
3.3. Tipos de aplicaciones a desarrollar en Android . . . 49
3.4. Diferencias entre el desarrollo en Android SDK y en terminales físicos . . . 50

4. ¿Qué es una Actividad?. Aplicaciones **53**
4.1. Los componentes de una aplicación . . . 53
4.2. El manifiesto de aplicación (I) . . . 54
4.2.1. Nodo <uses-sdk> . . . 56
4.2.2. Nodo <uses-configuration> . . . 57
4.2.3. Nodo <uses-feature> . . . 57
4.2.4. Nodo <supports-screens> . . . 57
4.2.5. Nodo <application> . . . 58
4.2.6. Nodo <activity> . . . 58
4.2.7. Nodo <service> . . . 58
4.2.8. Nodo <provider> . . . 58
4.2.9. Nodo <receiver> . . . 59
4.2.10. Nodo <uses-permission> . . . 59
4.2.11. Nodo <permission> . . . 59
4.2.12. Nodo <instrumentation> . . . 59
4.3. El manifiesto de aplicación (II) . . . 60
4.3.1. Pestaña Manifest . . . 60
4.3.2. Pestaña Application . . . 61
4.3.3. Pestaña Permissions . . . 61
4.3.4. Pestaña Instrumentation . . . 62
4.4. Externalizando recursos . . . 62
4.4.1. Valores simples . . . 64
4.4.2. Estilos y temas . . . 66
4.4.3. Imágenes . . . 66
4.4.4. Layouts . . . 66
4.4.5. Animaciones . . . 66
4.4.6. Menús . . . 66

5. Interfaces de Usuario I. Fundamentos **67**
5.1. Introducción . . . 67
5.1.1. *Widgets* . . . 68
5.1.2. Capturando eventos . . . 69
5.1.3. Método findViewById() . . . 69
5.1.4. Pestaña GraphicalLayout . . . 70
5.2. Vistas . . . 71
5.2.1. TextView . . . 71
5.2.2. EditText . . . 73
5.2.3. Button e ImageButton . . . 73
5.2.4. CheckBox y RadioButton . . . 76
5.2.5. ImageView . . . 78
5.2.6. ProgressBar . . . 80
5.2.7. Spinner . . . 82
5.3. Layouts . . . 84
5.3.1. FrameLayout . . . 84
5.3.2. LinearLayout . . . 84
5.3.3. ListView . . . 85
5.3.4. TableLayout . . . 86

6. Interfaces de Usuario II. Recursos, pantallas y menús **87**
6.1. Recursos . . . 87
6.1.1. Creando alias para los recursos . . . 91
6.2. Transiciones entre pantallas . . . 92
6.2.1. Un *layout* por cada pantalla . . . 92
6.2.2. Una `Activity` por cada pantalla . . . 94
6.2.3. *Intents* en el `Manifest.xml` . . . 95
6.2.4. Código de transición entre pantallas . . . 97
6.3. Menús . . . 98

7. Actividad final **101**
7.1. Descripción . . . 101

Parte II. Desarrollo Avanzado de Aplicaciones para Dispositivos Móviles *Android* **105**

1. Ficheros y Bases de Datos **105**
1.1. Manejo de ficheros . . . 105
1.1.1. Ficheros en la memoria interna del dispositivo . . . 105
1.1.2. Aplicación `TestFicheros` . . . 106
1.1.3. Ficheros en la memoria externa del dispositivo . . . 107
1.1.4. Sistemas de ficheros en el emulador . . . 108
1.2. Manejo de Bases de Datos . . . 110
1.2.1. Bases de datos `SQLite` . . . 110

2. SMS, MMS y Telefonía **115**
2.1. Telefonía . . . 115
2.1.1. Realización de una llamada . . . 115
2.1.2. Lectura de las propiedades del dispositivo . . . 115
2.1.3. Aplicación `TestTelefonia` . . . 117
2.2. Mensajería SMS . . . 119
2.2.1. Mensajes SMS a través de *intents* . . . 119
2.2.2. Mensajes SMS a través de la clase `SmsManager` . . . 119
2.2.3. Aplicación `TestSMS` . . . 121
2.3. Mensajería MMS . . . 122
2.3.1. Mensajes MMS a través de *intents* . . . 122

3. Conexiones HTTP y el WebBrowser **123**
3.1. Recursos de Internet . . . 123
3.2. Conexiones HTTP . . . 124
3.2.1. Permiso para la conexión a Internet . . . 124
3.2.2. Aplicación `HTTPImageDownloader` . . . 124
3.3. Aplicación `WebBrowser` . . . 126

4. Audio, Vídeo y Cámara **129**
4.1. Introducción . . . 129
4.2. Audio . . . 129
4.2.1. Formatos de audio soportados en Android . . . 129
4.2.2. Máquina de estados del Media Player . . . 130
4.2.3. Clase `MediaPlayer` . . . 131
4.2.4. Aplicación `TestAudio` . . . 131
4.3. Vídeo . . . 132

4.3.1. Clase `VideoView` . . . 132
4.3.2. Usando el Media Player . . . 133
4.4. Cámara . . . 135

5. Bluetooth **137**
5.1. Introducción . . . 137
5.2. Búsqueda de dispositivos . . . 138
5.2.1. Clase `BluetoothAdapter` . . . 138
5.2.2. Activación de Bluetooth en el dispositivo . . . 138
5.2.3. Descubrimiento de dispositivos remotos . . . 140
5.3. Establecimiento de la conexión . . . 141
5.3.1. Servidor . . . 141
5.3.2. Cliente . . . 142
5.4. Transferencia de datos . . . 144
5.5. Aplicación `TestBluetooth` . . . 144

6. Acelerómetro y Brújula **147**
6.1. Acelerómetro . . . 147
6.1.1. Introducción . . . 147
6.1.2. `SensorManager`, `SensorEventListener` y `SensorEvent` . . . 148
6.1.3. Aplicación `TestAcelerometro` . . . 149
6.2. Brújula . . . 150
6.2.1. Aplicación `TestBrujula` . . . 152

7. GPS y Google Maps **153**
7.1. Configuración de Eclipse para el uso de mapas . . . 153
7.1.1. Generación de la clave para la `Android Maps API` . . . 155
7.2. Aplicación `TestMapas` . . . 158
7.2.1. Clase `MapView` . . . 158
7.2.2. Clase `MapController` . . . 158
7.2.3. Clase `MapActivity` . . . 159
7.2.4. Manifiesto de la aplicación `TestMapas` . . . 159
7.3. Posicionamiento mediante GPS . . . 161
7.3.1. Permisos . . . 161
7.3.2. Interfaz `LocationListener` . . . 161
7.3.3. Clase `Overlay` . . . 162
7.3.4. Clase `LocationManager` . . . 163
7.3.5. Aplicación `TestGPS` . . . 163

8. Actividad final **165**
8.1. Descripción . . . 165

Apéndices **167**

A. Protocolo Bluetooth **167**
A.1. Arquitectura Bluetooth . . . 167
A.2. Perfiles de Bluetooth . . . 168
A.3. La pila de protocolos de Bluetooth . . . 169
A.3.1. La capa de radio . . . 169
A.3.2. La capa de banda base . . . 169
A.3.3. *Link Manager* . . . 170
A.3.4. *Host Controller Interface* . . . 170

A.3.5. Protocolo de adaptación y control de enlaces lógicos L2CAP 170
A.3.6. RFCOMM 170
A.3.7. TCS y SDP 170
A.4. La capa de radio de Bluetooth 171
A.5. La capa de banda base de Bluetooth 171
A.6. La capa L2CAP de Bluetooth 172
A.7. Estructura de la trama de Bluetooth 172

B. Acrónimos **173**

Bibliografía **175**

Índice de Figuras

Desarrollo de Aplicaciones para Dispositivos Móviles *Android* . . . 23

1.1. Algunos dispositivos móviles que soportan Android . . . 23
1.2. Evolución de Android hasta la versión 2.2 . . . 24
1.3. Galaxy Nexus . . . 25
1.4. Arquitectura de Android . . . 27

2.1. Logo del entorno de desarrollo Eclipse . . . 29
2.2. Obteniendo la JDK 7.0 de Java (I) . . . 30
2.3. Obteniendo la JDK 7.0 de Java (II) . . . 30
2.4. Obteniendo Eclipse (I) . . . 31
2.5. Obteniendo Eclipse (II) . . . 31
2.6. Espacio de trabajo en Eclipse . . . 32
2.7. Pantalla de bienvenida a Eclipse . . . 32
2.8. Entorno de trabajo habitual de Eclipse . . . 33
2.9. Obteniendo *Android SDK* . . . 33
2.10. Iniciando la instalación de *Android SDK* . . . 34
2.11. Especificando la ruta de instalación deseada para *Android SDK* . . . 34
2.12. Opciones disponibles a instalar (I) . . . 35
2.13. Opciones disponibles a instalar (II) . . . 35
2.14. Instalando el plugin ADT en Eclipse (I) . . . 36
2.15. Instalando el plugin ADT en Eclipse (II) . . . 36
2.16. Instalando el plugin ADT en Eclipse (III) . . . 37
2.17. Instalando el plugin ADT en Eclipse (IV) . . . 37
2.18. Instalando el plugin ADT en Eclipse (V) . . . 37
2.19. Configurando Eclipse y el plugin ADT (I) . . . 38
2.20. Configurando Eclipse y el plugin ADT (II) . . . 38
2.21. Configurando Eclipse y el plugin ADT (III) . . . 39
2.22. Acceso directo a Android Manager desde Eclipse . . . 39

3.1. Creación de un nuevo proyecto de Android (I) . . . 41
3.2. Creación de un nuevo proyecto de Android (II) . . . 41
3.3. Creación de un nuevo proyecto de Android (III) . . . 42
3.4. Creación de un nuevo proyecto de Android (IV) . . . 43
3.5. Integración Eclipse-Android SDK-Emulador . . . 43
3.6. Configuración del emulador de Android (I) . . . 44
3.7. Configuración del emulador de Android (II) . . . 44
3.8. Configuración del emulador de Android (III) . . . 45
3.9. Configuración del entorno de ejecución de Android (I) . . . 45
3.10. Configuración del entorno de ejecución de Android (II) . . . 46
3.11. Configuración del entorno de ejecución de Android (III) . . . 46

3.12. Ejecución de la aplicación HolaAndroid (I) . . . 47
3.13. Ejecución de la aplicación HolaAndroid (II) . . . 47

4.1. Manifiesto de la aplicación (I) . . . 54
4.2. Manifiesto de la aplicación (II) . . . 55
4.3. Estructura del fichero `Manifest.xml` . . . 56
4.4. Manifiesto: Pestaña Manifest . . . 60
4.5. Manifiesto: Pestaña Application . . . 61
4.6. Manifiesto: Pestaña Permissions . . . 61
4.7. Manifiesto: Pestaña Instrumentation . . . 62
4.8. Package Explorer. Recursos . . . 62
4.9. Clase R . . . 63
4.10. Vista del fichero `strings.xml` (I) . . . 64
4.11. Vista del fichero `strings.xml` (II) . . . 64
4.12. Creando el fichero `colors.xml` . . . 64

5.1. Jerarquía de la clase `View`: *Widgets* y *Layouts* . . . 67
5.2. Grupo de vistas como contenedor de otras vistas . . . 68
5.3. Pestaña GraphicalLayout . . . 70
5.4. Definiendo un nuevo recurso de tipo `String` . . . 71
5.5. Añadiendo un nuevo `TextView` . . . 72
5.6. Ejemplo de uso de un `TextView` . . . 72
5.7. Conversor de divisas (GraphicalLayout) . . . 73
5.8. Añadiendo nuevos `Button`s . . . 73
5.9. Añadiendo un nuevo `ImageButton` (I) . . . 75
5.10. Añadiendo un nuevo `ImageButton` (II) . . . 75
5.11. Nueva aplicación para `CheckBox`, `RadioButton` y `RadioGroup` (I) . . . 77
5.12. Nueva aplicación para `CheckBox`, `RadioButton` y `RadioGroup` (II) . . . 78
5.13. Especificando la ubicación de la imagen para un `ImageView` . . . 79
5.14. Nueva aplicación para `ImageView` llamada `TestImagenes` . . . 80
5.15. Nueva aplicación para `ProgressBar` . . . 80
5.16. Barra de progreso estilizada . . . 82
5.17. Nueva aplicación para probar el `Spinner` . . . 82
5.18. Creando el array con el interfaz gráfico de Eclipse . . . 83
5.19. Ejemplo de uso de un `LinearLayout` . . . 84
5.20. Ejemplo de uso de una `ListView` . . . 85
5.21. Ejemplo de uso de un `TableLayout` . . . 86

6.1. Diferencia entre el uso de los recursos por defecto y el uso de recursos alternativos 87
6.2. Ejemplo de carpetas específicas para distintas configuraciones . . . 89
6.3. Aspecto del *layout* de la pantalla número 1 . . . 92
6.4. Creando el *layout* de la pantalla 2 . . . 93
6.5. Aspecto del *layout* de la pantalla número 2 . . . 93
6.6. Creando la nueva actividad de la pantalla 2 . . . 94
6.7. Trabajando con Intents desde modo gráfico . . . 96
6.8. Modificando los parámetros del nuevo nodo . . . 96
6.9. Creando la etiqueta Intent-filter . . . 97
6.10. Adaptando las etiquetas en el fichero Manifest . . . 97
6.11. Creando un nuevo menú . . . 98
6.12. Añadiendo nuevos ítems al menú . . . 99

Desarrollo Avanzado de Aplicaciones para Dispositivos Móviles *Android* 105

1.1. Pantallas de la aplicación `TestFicheros` 107
1.2. Configuración del emulador para el uso de ficheros externos 109
1.3. Perspectiva DDMS de Eclipse: Tarjeta SD 109
1.4. Perspectiva DDMS de Eclipse: Bases de datos 111

2.1. *Layout* `main` de la aplicación `TestTelefonia` 118
2.2. Aplicación `TestTelefonia` 118
2.3. Mensajes SMS a través de *intents* 119
2.4. *Layout* `main` de la aplicación `TestSMS` 121
2.5. Aplicación `TestSMS` 122

3.1. Permiso `INTERNET` en el manifiesto de `HTTPImageDownloader` 124
3.2. *Layout* `main` de la aplicación `HTTPImageDownloader` 125
3.3. Aplicación `HTTPImageDownloader` 126
3.4. *Layout* `main` de la aplicación `WebBrowser` 127
3.5. Aplicación `WebBrowser` 128

4.1. Máquina de estados del Media Player de Android 130
4.2. *Layout* `main` de la aplicación `TestAudio` 131
4.3. Aplicación `TestVideo` 133
4.4. *Layout* `main` de la aplicación `TestVideoSurface` 134
4.5. *Layout* `main` de la aplicación `TestCamara` 135

5.1. Permisos `BLUETOOTH` y `BLUETOOTH_ADMIN` en el manifiesto de `TestBluetooth` 144
5.2. *Layout* `main` de la aplicación `TestBluetooth` 145

6.1. Ejes de coordenadas para el acelerómetro 147
6.2. *Layout* `main` de la aplicación `TestAcelerometro` 150
6.3. Sistema de orientación de Android 151
6.4. *Yaw* (*Azimuth*), *Pitch* y *Roll* 151
6.5. *Layout* `main` de la aplicación `TestBrujula` 152

7.1. Android SDK and AVD Manager: `Google API` 153
7.2. Creando un nuevo ADV usando la *Google API* 154
7.3. Creando el proyecto `TestMapas` 155
7.4. Obtención de la clave para la `Android Maps API` (I) 156
7.5. Obtención de la clave para la `Android Maps API` (II) 156
7.6. Obtención de la clave para la `Android Maps API` (III) 157
7.7. Obtención de la clave para la `Android Maps API` (IV) 157
7.8. Permisos `INTERNET` y `ACCESS_FINE_LOCATION` en el manifiesto de `TestMapas` . 159
7.9. Librería `com.android.google.maps` en el manifiesto de `TestMapas` 160
7.10. Aplicación `TestMapas` 160
7.11. Permisos `INTERNET` y `ACCESS_FINE_LOCATION` en el manifiesto de `TestGPS` . . 161

Apéndice A. Protocolo Bluetooth 167

A.1. *Scatternet* formada por 12 *piconets* 167
A.2. Pila de protocolos de Bluetooth 169
A.3. *Host Controller Interface* 170
A.4. Coexistencia de Bluetooth y 802.11 en la banda de los 2'4 GHz 171

A.5. Estructura de la trama de Bluetooth 172

Índice de Tablas

Desarrollo de Aplicaciones para Dispositivos Móviles *Android* . . . 23

3.1. Posibles valores para ciertas propiedades gráficas de una aplicación Android . 49

4.1. Valores posibles para el nodo <`uses-sdk`> 56
4.2. Resoluciones y densidades de pantalla habituales 57

6.1. Directorios de recursos en Android 88
6.3. Configuraciones para la definición de recursos en Android 91
6.4. `Strings` de la aplicación `ProbandoPantallas` 92

Desarrollo Avanzado de Aplicaciones para Dispositivos Móviles *Android* 105

1.1. Modos de creación de ficheros para el método `openFileOutput()` 106
1.2. Métodos de la clase `Context` para el manejo de ficheros 106

4.1. Formatos de audio soportados en Android 129

5.1. Constantes de estado de la clase `BluetoothAdapter` 138
5.2. Constantes de descubrimiento de la clase `BluetoothAdapter` 140

6.1. Propiedades de la clase `SensorEvent` 149

Apéndice A. Protocolo Bluetooth 167

A.1. Perfiles de Bluetooth 168
A.2. Clases de dispositivos Bluetooth 169

Introducción

1. Presentación

Este curso de promoción educativa pretende dar una visión general y completa acerca del lenguaje de programación para dispositivos móviles **Android**.

El curso, titulado "Desarrollo de Aplicaciones para Dispositivos Móviles Android", se enmarca dentro de los cursos de promoción educativa ofrecidos por el Área de Tecnologías de la Información y las Comunicaciones Aplicadas (ÁTICA) de la Universidad de Murcia.

Está dirigido principalmente a todas aquellas personas que cuenten con unos conocimientos básicos de programación en el lenguaje Java y que deseen ampliar sus conocimientos y habilidades para ser capaces de desarrollar aplicaciones escritas en Android orientadas a dispositivos móviles tales como teléfonos móviles, PDAs o Smartphones.

Las principales características del curso son:

- 6 créditos ECTS
- Curso virtual (on-line) a través de la plataforma educativa SAKAI[1]
- Posibilidad de solicitud de beca
- Posibilidad de pago con tarjeta de crédito
- Nº alumnos: mínimo 20 / máximo 40
- Web: `http://ants.dif.um.es/~felixgm/docencia/android`

[1] `https://aulavirtual.um.es`

2. Motivación

Mientras que la enseñanza del lenguaje de programación Java ha recibido una gran atención desde muchos órganos de dirección docente y ha sido incluido en la mayoría de planes docentes de carreras como Informática, Telecomunicaciones, Matemáticas, etc., se ha prestado poca atención al desarrollo de aplicaciones para dispositivos móviles.

En este curso se mostrará la facilidad con la que, contando con unos conocimientos básicos de programación en Java, es posible desarrollar rápidamente una aplicación para un dispositivo móvil Android.

Además, muchas empresas del sector de las TIC (Tecnologías de la Información y las Comunicaciones) están apostando por las comunicaciones y los productos orientados a dispositivos móviles. Sin embargo, existe un gran vacío de profesionales y expertos en este sector que hacen de esta tecnología un campo de interés muy importante.

Pensemos que hoy en día todos llevamos encima, a todas horas, un dispositivo móvil (léase teléfono móvil). Se trata pues, de un mercado con grandes posibilidades al cual aún no se le ha sacado todo el partido posible.

3. Contenidos

El curso titulado "**Desarrollo de Aplicaciones para Dispositivos Móviles Android**", trata de dar una visión básica del lenguaje Android y está orientado principalmente a aquellas personas que no hayan tenido ningún contacto previo con Android, o con conocimientos mínimos. El curso se divide en los siguientes 7 temas:

1. **Introducción.**

   En este tema se dará una breve reseña histórica sobre Android y se verán los conceptos básicos que lo engloban; la arquitectura del sistema operativo organizado en capas, su máquina virtual Dalvik, sus librerías y el SDK disponible

2. **Entorno de desarrollo Eclipse**

   Este tema explicará cómo instalar Eclipse junto con el plugin ADT, su correcta configuración y cómo crear un nuevo proyecto de desarrollo de una aplicación Android, así como buena parte de toda la funcionalidad que ofrece: Depuración y visualización de la máquina virtual.

3. **Nuestro primer programa**

   En este tema se pondrá en funcionamiento la primera aplicación bajo Android, y se explicarán los tipos de aplicaciones posibles a desarrollar en Android. Se dará una explicación sobre las diferencias existentes entre la ejecución haciendo uso del simulador incorporado en el plugin de Eclipse y la ejecución en un terminal físico.

4. **¿Qué es una actividad?**

   Este tema está dedicado al concepto de Actividad para Android, se explica la clase `Application`, se introduce el concepto de Manifiesto y se introduce la externalización de recursos.

5. **Interfaces de usuario I. Fundamentos**

   Este tema explicará los fundamentos básicos que rigen las interfaces de usuario en Android, se explica el concepto de vista y cómo crear nuevas vistas. Se exponen también los controles compuestos y los layouts.

6. **Interfaces de usuario II. Recursos, pantallas y menús**

   En este tema se verán los recursos que existen para el dibujo de las interfaces, las resoluciones y densidades que se pueden aplicar a los dibujos, veremos cómo testear interfaces, y crearemos y editaremos menús. Finalmente veremos cómo el lenguaje XML nos puede ayudar a la hora de desarrollar nuestra interfaz de usuario.

7. **Actividad final**

   En este último tema se propone al alumno que, con todos los conocimientos adquiridos a lo largo del curso, desarrolle libremente una aplicación donde se vean reflejados dichos conocimientos. No obstante, se sugerirán algunos ejercicios.

Por su parte, en el curso avanzado, formalmente titulado "**Desarrollo Avanzado de Aplicaciones para Dispositivos Móviles Android**" se explicarán algunas librerías opcionales de Android orientadas a desarrollar aplicaciones Android más complejas y completas. Este curso está principalmente enfocado a los alumnos que hayan realizado el curso básico de Android. El curso se divide en los siguientes 8 temas:

1. **Ficheros y bases de datos.**

   En este tema se mostrará cómo acceder al sistema de ficheros de un dispositivo Android, así como consultar bases de datos

2. **SMS, MMS y telefonía.**

   Este tema explicará de qué manera es posible enviar y recibir mensajes SMS y MMS mediante una aplicación Android. También se verá el manejo de algunas funciones de telefonía a través de Android

3. **Conexiones HTTP y el WebBrowser.**

   En este tema se dará al alumno los conocimientos necesarios para establecer conexiones HTTP mediante el lenguaje Android, al tiempo que se explicará cómo visualizar páginas web en un navegador en este tipo de dispositivos, usando Android

4. **Audio, vídeo y cámara.**

   Este tema está dedicado a conocer cómo manejar ficheros de audio y vídeo mediante terminales Android, así como a utilizar la cámara con la que cuente el dispositivo móvil

5. **Bluetooth.**

   Este tema explicará los conceptos básicos para establecer una comunicación Bluetooth entre terminales Android, haciendo uso de dicho lenguaje de programación

6. **Acelerómetro y brújula.**

   En este tema se verán los conocimientos necesarios para utilizar el acelerómetro y la brújula en dispositivos móviles Android

7. **GPS y Google Maps.**

   En este tema se mostrará cómo utilizar el GPS integrado en los dispositivos móviles Android, al tiempo que se verá cómo integrar Google Maps en nuestras aplicaciones Android

8. **Actividad final.**

   En este último tema se propone al alumno que, con todos los conocimientos adquiridos a lo largo del curso, desarrolle libremente una aplicación donde se vean reflejados dichos conocimientos. Los profesores, no obstante, sugerirán algunos ejercicio

4. Objetivos

Los objetivos del curso de promoción educativa titulado "**Desarrollo de Aplicaciones para Dispositivos Móviles Android**" son:

- Introducir al alumno al lenguaje Android y hacerle ver el gran potencial que dicho lenguaje tiene, así como sus limitaciones
- Mostrar la facilidad de desarrollo con el entorno Enclipse, junto con el plugin ADT
- Dar al alumno los conocimientos necesarios para desarrollar y desplegar aplicaciones Android sencillas
- Enseñar el manejo de layouts, vistas, controles, menús, etc.

Los objetivos del curso de promoción educativa titulado "**Desarrollo Avanzado de Aplicaciones para Dispositivos Móviles Android**" son:

- Mostrar al alumno algunas de las capacidades avanzadas de desarrollo que ofrece el lenguaje Android, haciéndole ver el gran potencial que dicho lenguaje tiene, así como sus limitaciones.
- Enseñar el manejo de ficheros y bases de datos con el lenguaje Android.
- Manejar el envío y recepción de SMS y MMS, así como el acceso a la agenda de contactos o la realización de una llamada telefónica mediante una aplicación Android.
- Conocer cómo se establece una conexión HTTP y cómo abrir un navegador web mediante Android.
- Aprender a utilizar ficheros de audio y vídeo, además de la cámara del dispositivo móvil con el lenguaje Android.
- Mostrar al alumno las capacidades de comunicación, envío y recepción de datos a través del protocolo Bluetooth, haciendo uso del lenguaje Android.
- Manejar el acelerómetro y la brújula de un dispositivo móvil mediante una aplicación de Android.
- Hacer uso del GPS y las funcionalidades ofrecidas por Google Maps mediante el lenguaje Android.

5. Metodología y Evaluación

¡¡IMPORTANTE!!

A lo largo de cada uno de los temas se propondrán al alumno varias actividades optativas que podrá realizar libremente (o no). Algunas de dichas actividades estarán incluso resueltas en la propia web del curso:

`http://ants.dif.um.es/~felixgm/docencia/android`

El alumno podrá enviar a través de SAKAI los resultados de las actividades propuestas que desee que los profesores corrijan (se sugiere que previamente compruebe que el ejercicio es similar a la solución propuesta en la web del curso y se envíe sólo en el caso de que surjan dudas o problemas en la realización de los ejercicios).

El curso cuenta con un último tema en el que se pide al alumno que, con todos los conocimientos adquiridos a lo largo del mismo, desarrolle una aplicación en Android en la que se apliquen varios de los conceptos vistos. Los profesores propondrán algunas actividades, aunque el alumno tiene la posibilidad de enviar su propia aplicación, previa puesta en conocimiento y consentimiento por parte de los profesores (`antonio.bernardez@um.es` y `felixgm@um.es`).

La entrega de la aplicación final se hará de la siguiente manera. El alumno deberá subir a SAKAI un fichero comprimido en **.ZIP** que contenga:

- La carpeta completa del proyecto de Eclipse de dicha aplicación
- Un fichero **contacto.txt** que contenga la información de contacto (e-mail y teléfono móvil) del alumno (sólo para aquellos alumnos que deseen participar en el concurso organizado por Neosistec y Accesium Technology)

El nombre del fichero tendrá el siguiente aspecto:

`apellidos.nombre-edicion-{basico|avanzado}-android-{SI|NO}.zip`

Por ejemplo:

`gomez-marmol.felix-I-basico-android-NO.zip` o
`bernardez-alvarez.antonio-VI-avanzado-android-SI.zip`

Donde `{SI|NO}` indicará la disposición del alumno a participar en el concurso organizado por Neosistec y Accesium Technology.

6. Concurso organizado por Neosistec y Accesium

*"**Neosistec**, empresa especializada en el desarrollo de aplicaciones para dispositivos móviles Android, y **Accesium Technology**, empresa del sector de las nuevas tecnologías participada por DIGIO y la Universidad de Murcia, ambas de ámbito nacional y con sede en Murcia, han tenido conocimiento del curso on-line de promoción educativa de "**Desarrollo de Aplicaciones para Dispositivos Móviles Android**" impartido por la Universidad de Murcia.*

Dada la calidad que hemos podido comprobar en dichos cursos y por encontrarnos en expansión en el área de aplicaciones en movilidad, dentro de nuestro departamento de Desarrollo e Innovación, estamos interesados en promover e incentivar la participación de nuevos alumnos en los mismos.

Es por esto que Neosistec y Accesium Technology desean ofrecer a los tres alumnos más destacados que hayan realizado dicho curso la posibilidad de llevar a cabo unas prácticas en empresa por 2 meses, prorrogables.

El profesor del curso, en cada edición del mismo, seleccionará de entre aquellos alumnos que deseen participar en esta iniciativa, los tres candidatos que presenten los proyectos más completos, innovadores, originales y prácticos. Una vez finalizado el curso, el profesor notificará a ambas empresas, en el plazo de una semana, los nombres e información de contacto de los candidatos seleccionados. Aproximadamente dos semanas después de dicha notificación, tendrán lugar sendas sesiones de exposición y defensa de los proyectos presentados, tanto en las oficinas de Neosistec, como en las de Accesium Technology, siendo los alumnos convocados por éstas últimas. Un tribunal compuesto por personal de cada empresa seleccionará entonces al candidato para realizar prácticas en la misma."

Javier Pita
Director de Neosistec
`http://www.neosistec.com`

Juan Egea
Consejero Delegado de Accesium Technology
`http://www.accesium.es`

Parte I

Desarrollo de Aplicaciones para Dispositivos Móviles *Android*

Tema 1

Introducción a Android. Conceptos básicos

1.1. Breve descripción histórica de Android

Allá por el mes de julio de 2005, Google adquirió una pequeña empresa Californiana llamada Android Inc al Ingeniero de Software Andy Rubin con la intención de desarrollar un sistema operativo móvil que pudiera responder a las expectativas que el mundo de los dispositivos móviles comenzaban a despertar. Bajo los rumores de que Google pudiera llegar incluso a regalar los terminales físicos aprovechando los consecuentes ingresos que obtendría a través de la publicidad de su buscador, Android se hizo famoso desde el primer momento.

De esta forma comenzaron los trabajos para que, en noviembre de 2007 se anunciara oficialmente la creación de la Open Handset Alliance, (una alianza mercantil formada por múltiples empresas que desarrollan estándares abiertos para dispositivos móviles, entre ellas Android) que vendría acompañada del lanzamiento oficial del **Android Software Development Kit**. Esta primera versión supuso un punto de partida y de igual forma una forma de evaluar la aceptación de la idea del mercado, por lo que hasta mediados de agosto de 2008 no apareció la segunda versión conocida como Android 0.9 SDK. Tras las comprobaciones oportunas, a finales del mes de septiembre lanzaron oficialmente la versión Android 1.0 SDK (Release 1).

Figura 1.1: Algunos dispositivos móviles que soportan Android

Seis meses después, en marzo de 2009, Google presentaba la versión 1.1 de Android y lanzaba al mercado **Android Market**, para poder adquirir aplicaciones basadas en Android.

Así, y en vista de los buenos resultados, Google no paró de trabajar, y las nuevas versiones continuaron apareciendo rápidamente, pues en abril de 2009, Google lanzó la versión 1.5 de Android (llamada coloquialmente *Cupcake*) con su respectivo SDK que incluía nuevas características como grabación de video, soporte para stereo Bluetooth, sistema de teclado personalizable en pantalla, reconocimiento de voz y el AppWidget que permitió que los desarrolladores pudieran crear sus propios widgets para la pantalla principal. Todas estas nuevas características unidas al hecho de que los dispositivos T-Mobile G1 y HTC Dream contaran con él, hizo que Android 1.5 fuese la versión que más personas usaron para iniciarse en Android.

La siguiente versión en aparecer fue Android 1.6 *Donut* en septiembre de 2009 con nuevas mejoras (aumento de la capacidad de las búsquedas, indicadores adicionales, etc) que hicieron que muchos usuarios optaran por migrar sus dispositivos de Android 1.5 a 1.6.

La versión 2.0 *Eclair* llegó de la mano del terminal físico Motorola Milestone que ya poseía características mucho más avanzadas y traía por defecto un pequeño número de aplicaciones instaladas.

En enero de 2010, Google volvía nuevamente a sorprender con el lanzamiento de su primer terminal físico, el Google Nexus One al que le añadió una nueva versión de su sistema Android, en este caso la 2.1 con nuevas capacidades 3D que representaron el punto de inflexión entre las versiones 1.x y 2.x.

En mayo de 2010 se lanzó la versión 2.2 de Android (*Froyo*), que incluía soporte entre otros para WiFi IEEE 802.11n, Radio FM, Adobe Flash 10.1, etc. Y en diciembre de ese mismo año apareció la versión 2.3 (*Gingerbread*), que incluye soporte nativo para telefonía VoIP SIP, o comunicaciones NFC, entre otras características reseñables.

Figura 1.2: Evolución de Android hasta la versión 2.2

Fue a finales de 2010, concretamente el 6 de diciembre cuando (*Gingerbread*) vio la luz en forma de la version 2.3 incluyendo importantes mejoras en la interfaz de usuario, mejor soporte para pantallas más grandes, soporte nativo para telefonía VoIP, y muchas otras más. Otro de los aspectos que más fueron valorados de esta versión fue la considerable mejora en el administrador de energía, lo que permite un menor desgaste la batería.

La versión 3.0, 3.1 y 3.2 denominada HoneyComb se diseñó fundamentalmente pensando en los Tablets PC. Se volvió a rediseñar iconos y widgets incorporando vistas 3D y dando a esta versión del sistema una imagen muy actual y tecnológica. Además se añadieron importantes mejoras en términos de conectividad en periféricos y accesorios con conexión USB y soporte para redes Wi-Fi.

Finalmente, y a medias entre 2011 y 2012 sale al mercado la version 4.0 o Ice Cream Sandwich. Esta versión es sin duda la más versátil que existe pues trata de unificar el uso de cualquier dispositivo, tanto teléfonos, tables, televisores, netbooks, etc. Como principal mejora incorpora la aceleración por hardware a través de la GPU lo que permite gráficos de mejor calidad y una experiencia muy mejorada de cara al usuario. Además, esta versión se presenta con una enorme cantidad de posibilidades de personalización por parte del usuario, permitiéndole al mismo crear un teléfono a su gusto.

Google presentó formalmente la versión 4.1 o Jelly Bean en la presentación del evento Google I/O el 27 de Junio de 2012 en Florida (EEUU). Los principales objetivos de esta versión son:

- La mejora de las animaciones del sistema y de la estabilidad a través de lo que se ha dado a conocer como Proyecto Butter.
- El incremento de la calidad estética de las aplicaciones en los nuevos dispositivos que cuentan con pantallas más grandes
- La incorporación de la búsqueda en Google a través de la voz.
- Notificaciones mejoradas.
- Ajuste automático de widgets en el escritorio.
- Google Chrome se convierte en el navegador del sistema por defecto.

Por último comentar que esta versión ha venido acompañada del teléfono también de marca Google: Galaxy Nexus, que cuenta con una pantalla de 4,65 pulgadas que ofrece una resolución de 1280x720 píxeles con tecnología SuperAMOLED, y un procesador de 1,2 Ghz.

Figura 1.3: Galaxy Nexus

Actualmente, las nuevas versiones de Android (4.0 y 4.1) van ganando terreno a las versions 2.2 y 2.3 que llegaron a copar más del 85 % del Mercado. La lucha entre Android y Apple es constante y ambos gigantes luchan por ganar la batalla que les posicione como líder en el sector.

A día de hoy Google ofrece más de 675.000 aplicaciones en su tienda virtual Google Play.

1.2. ¿Qué es Android?

Andy Rubin, que hoy por hoy ostenta el cargo de Vicepresidente de Ingeniería de Google, define Android como:

> *"The first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation."*

Que traducido viene a decir:

> *"La primera plataforma verdaderamente abierta y global para dispositivos móviles, con todo el software para hacer funcionar un teléfono móvil pero sin los obstáculos de los propietarios que han entorpecido la innovación en el mundo de los dispositivos móviles."*

De una forma más cercana al mundo informático, Android es un software de código abierto *open source* que incluye un sistema operativo, un *middleware* o capa intermedia, las principales aplicaciones móviles necesarias para cualquier dispositivo móvil y un conjunto de librerías que permiten escribir nuevas aplicaciones que se podrán instalar en dispositivos móviles que tengan Android instalado.

Además, Android se ha diseñado de forma que tanto las aplicaciones nativas del software como las aplicaciones desarrolladas por terceras personas estén escritas bajo las mismas APIs y funcionen bajo el mismo entorno de ejecución. Esto permite que el dispositivo móvil se convierta en un gran puzzle, en donde se puede sustituir cualquier pieza del mismo por otra que pueda parecernos mejor.

Por último, hoy por hoy Android ofrece en su página web oficial (`www.android.com`), una documentación excelente, una muy amplia comunidad de desarrolladores y no por ello menos importante, coste cero en desarrollo y distribución.

1.3. Características del SDK de Android

Las características que ofrece el *Software Development Kit* de Android son las que se pueden ver a continuación:

- Coste cero de licencia, distribución y desarrollo
- Acceso hardware a WiFi
- Redes GSM, EDGE, y 3G para telefonía y transferencia de datos
- API para servicios basados en localización GPS
- Control total del hardware multimedia, incluyendo reproducción y grabación mediante cámara y micrófono
- API para los sensores hardware incluyendo el acelerómetro y el compás
- Librerías para el uso de Bluetooth en transferencias de datos P2P
- Paso de mensajes IPC
- Almacenes de datos compartidos
- Aplicaciones y procesos en segundo plano

- Fondos de escritorio, carpetas y utilidades de escritorio
- Navegador web Open Source HTML 5
- Soporte completo para aplicaciones que integran control de mapas como parte de su interfaz
- Soporte a través de librerías 2D y 3D OpenGL de gráficos
- Librerías para multimedia audio/vídeo o formato de imágenes

1.4. Arquitectura de Android

La figura 1.4 presenta la arquitectura de Android, así como los niveles que la componen.

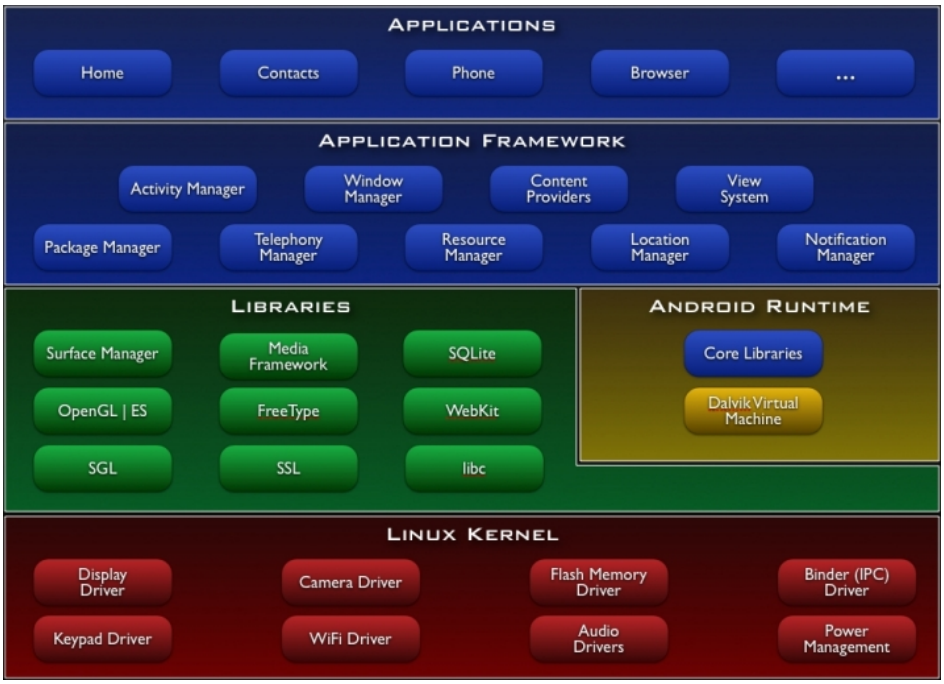

Figura 1.4: Arquitectura de Android

A continuación describiremos brevemente cada uno de los niveles de la arquitectura de Android, a saber: *Applications*, *Application Framework*, *Libraries*, *Android Runtime* y *Linux Kernel*.

Nivel *Applications*

Android cuenta con una serie de aplicaciones de base que incluyen el cliente de correo electrónico, el programa para los mensajes SMS, el calendario, mapas, navegador, lista de contactos, etc.

Nivel *Application Framework*

Mediante este nivel, los desarrolladores tienen acceso a los dispositivos hardware del terminal, pueden ejecutar servicios en segundo plano, activar alarmas, y todo lo necesario para implementar aplicaciones de gran potencia. Para ello, se les da un acceso total al mismo API que se utiliza para el desarrollo de las aplicaciones básicas del nivel *Applications*. De esta

forma se favorece, tal y como veíamos anteriormente, la reutilización de los componentes, y se posibilita el hecho de poder cambiar componentes a gusto del usuario.

Nivel *Libraries*

Android incluye un conjunto de librerías optimizadas escritas en C/C++ que son utilizadas por diversos componentes del sistema. Las capacidades de estas librerías no están disponibles de forma directa y son accesibles a los desarrolladores a través de las APIs del nivel superior. Dicho en otras palabras, no se puede coger una librería escrita en C y usarla directamente; si se quiere utilizar, se debe hacer uso de ella, a través de la API de nivel superior.

Las librerías de las que hablamos son las que se pueden ver a continuación:

- Librería `System C`. `System C` es una implementación derivada de la librería estándar C (conocida como `libc`) adaptada a dispositivos embebidos basados en `Linux`.
- Librerías multimedia. Soportan la reproducción y grabación de audio y vídeo de los formatos más habituales como puedan ser MPEG4, H.264, MP3, JPG, etc.
- `LibWebCore`. Se trata de un motor de navegación web moderno que potencia al navegador Android y al navegador embebido para web.
- `Surface Manager`. Controla el acceso al subsistema de pantalla y al de componentes 2D y 3D para múltiples aplicaciones.
- `SQLite`. Base de datos relacional, ligera y potente.
- Librerías 3D. Una implementación basada en OpenGL 1.0 que incluye aceleración 3D por hardware.
- `FreeType`. Renderizado de vectores y mapas de bits.

Cabe destacar que si se desea desarrollar aplicaciones en donde el rendimiento y la eficiencia sean prioritarios en su máxima expresión, Android proporciona un entorno de desarrollo conocido como NDK que permite la creación de librerías C++ utilizando las librerías `libc` y `libm`, junto con un acceso a bajo nivel a OpenGL. No obstante, no es el objetivo de este curso tratar este nivel.

Nivel *Android Runtime*

Android incluye un conjunto de librerías básicas que proporcionan la mayor parte de la funcionalidad disponible en las librerías de Java.

Además, Android dispone de una máquina virtual similar a la que dispone el lenguaje Java, que ha sido rediseñada de forma que cuando un proceso es lanzado a ejecución disponga de su propia instancia de máquina virtual.

Así, un dispositivo móvil puede tener en un momento dado múltiples instancias de máquinas virtuales ejecutándose en paralelo, gestionándolas todas de forma eficiente y equilibrada en términos de consumo de memoria. Dicha máquina virtual se conoce bajo el nombre de **Dalvik**.

Nivel *Linux Kernel*

A partir de la versión 4.0 Android descansa sobre el núcleo 3.0 de Linux en términos de servicios y seguridad, gestión de memoria y procesos, conexiones de red y montaje de drivers. De igual forma, el núcleo también actúa como nivel de abstracción entre el hardware y el resto de software.

Tema 2

Entorno de desarrollo Eclipse

En este capítulo se desarrollará la instalación y puesta a punto del entorno de trabajo Eclipse para desarrollar aplicaciones en el lenguaje Android. Veremos los requisitos básicos, el software necesario y finalizaremos explicando las herramientas que ofrece el entorno.

> **Nota.-** El alumno es libre de utilizar otro entorno de desarrollo para aplicaciones Android, si así lo prefiere. No obstante, todos los ejemplos y contenidos de este curso están basados y explicados haciendo uso del entorno Eclipse para Windows.

2.1. Instalación de Eclipse

Figura 2.1: Logo del entorno de desarrollo Eclipse

2.1.1. Requisitos del sistema

En primer lugar comenzaremos estudiando los requisitos del sistema necesarios para instalar el entorno de desarrollo Eclipse para desarrollar aplicaciones para Android. Así, los sistemas operativos sobre los que podemos instalar el SDK de Android son los siguientes:

- Windows. Versiones XP, Vista, y 7
- Linux. Sólo en distribuciones que se ejecuten sobre arquitecturas de 32 bits
- Mac OS X 10.5.8 o superior en arquitecturas de 32 bits

2.1.2. Instalación de la JDK

Para poder ejecutar Eclipse, es preciso tener instalado previamente en el ordenador la JDK 7.0 de Java, o superior, que se puede obtener de la siguiente página web, tal y como se observa en la Figura 2.2:
`http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Figura 2.2: Obteniendo la JDK 7.0 de Java (I)

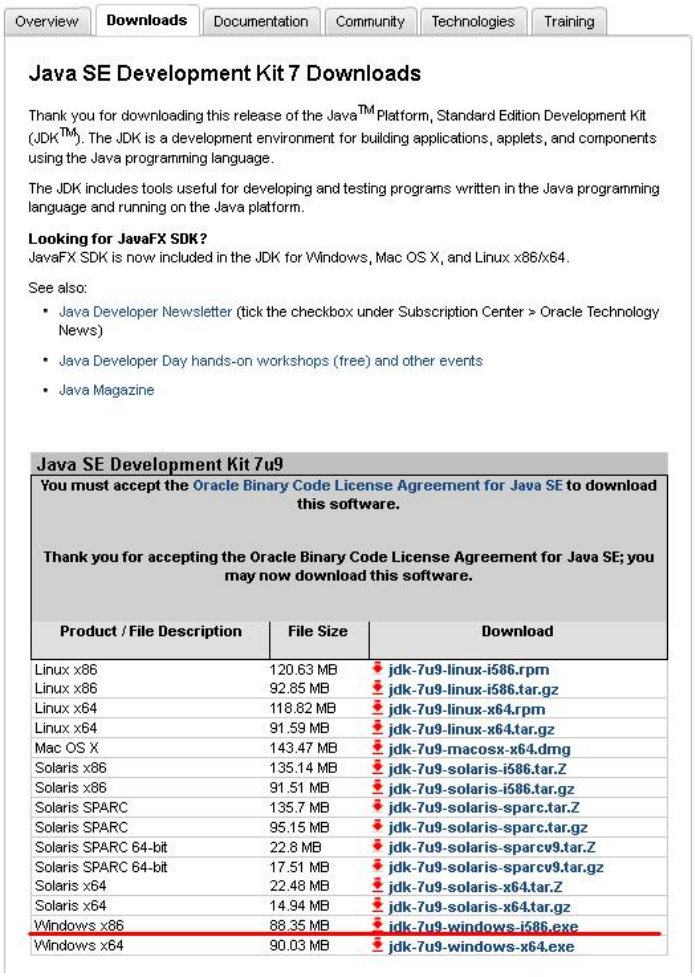

Figura 2.3: Obteniendo la JDK 7.0 de Java (II)

Pulsamos sobre el botón remarcado en rojo y se nos redireccionará a la pantalla mostrada en la figura 2.3.

> **Nota.-** Los usuarios del sistema operativo Linux, también se pueden instalar la JDK desde los distintos repositorios destinados a tal fin, y para el caso de aquellos que utilicen Ubuntu, también es posible instalarla mediante el Gestor de Paquetes Synaptic.

El proceso de instalación de la JDK es francamente sencillo. Únicamente se debe seguir todo el proceso sin modificar ninguna opción. Al finalizar la instalación de la JDK, se abrirá automáticamente una página web de registro. Este paso es totalmente opcional.

2.1.3. Instalación de Eclipse

El siguiente paso consistirá en obtener Eclipse. Eclipse es un entorno de programación basado en Java que cuenta con la ventaja de que no necesita instalación, tan sólo debemos descomprimirlo en la carpeta que prefiramos y a partir de ahí debemos comenzar a trabajar.

En primer lugar deberemos descargar la versión apropiada del entorno de desarrollo Eclipse. Para ello, accedemos a la siguiente página web, tal y como se observa en la figura 2.4:

`http://www.eclipse.org/downloads`

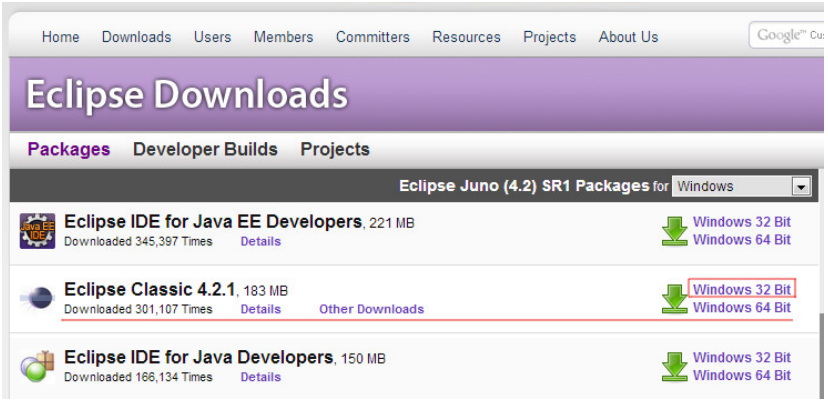

Figura 2.4: Obteniendo Eclipse (I)

Tras seleccionar la versión Eclipse Classic 4.2.1, nos aparecerá una página como la que se muestra en la Figura 2.5, desde la cual nos descargaremos finalmente el entorno de desarrollo.

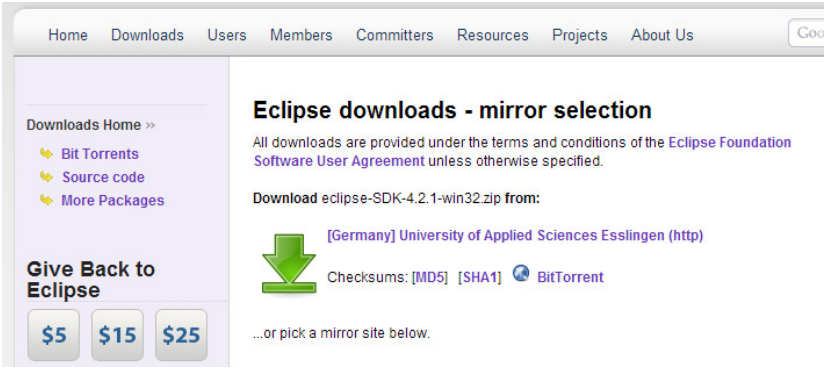

Figura 2.5: Obteniendo Eclipse (II)

Una vez haya finalizado la descarga del fichero de Eclipse, descomprimiremos el fichero en la carpeta que consideremos oportuna. Nosotros hemos decidido instalarlo en la carpeta `C:\eclipse` del sistema.

Como ya se ha comentado anteriormente, Eclipse no requiere de ningún tipo de instalación, por lo que el siguiente paso es directamente ejecutar Eclipse. Para ello, simplemente ejecutaremos el fichero `eclipse.exe` contenido en la carpeta donde hayamos descrompimido el entorno de desarrollo.

Nada más ejecutar Eclipse, se nos solicitará que indiquemos el espacio de trabajo donde se almacenarán los proyectos que se realizarán (véase figura 2.6). Por comodidad, nosotros hemos definido la siguiente carpeta `C:\eclipse\workspace`.

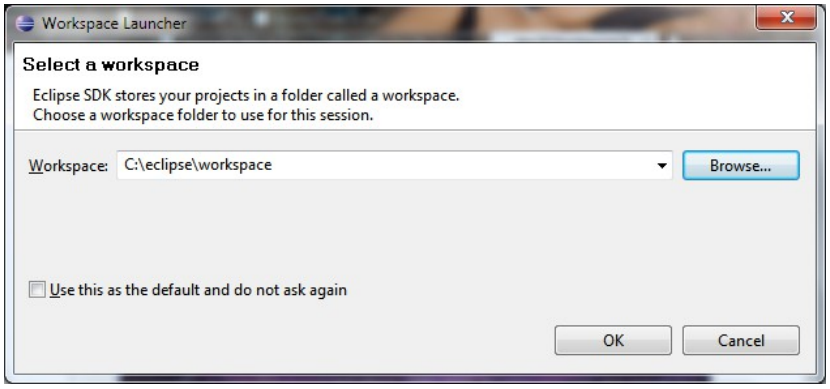

Figura 2.6: Espacio de trabajo en Eclipse

Finalmente, Eclipse se inicia y nos muestra su pantalla de bienvenida, como se observa en la figura 2.7. En ella podemos encontrar información y pequeños tutoriales donde iniciarnos en el mundo de Eclipse dentro del marco de la programación de aplicaciones Java. Eclipse utiliza un sistema de pestañas similar a muchos navegadores, podemos cerrar la pantalla de bienvenida de igual forma a como lo haríamos con una pestaña del navegador.

Figura 2.7: Pantalla de bienvenida a Eclipse

De esta forma ya tendremos nuestro entorno de desarrollo Eclipse preparado para desarrollar aplicaciones Java. La ventana habitual de trabajo de Eclipse es la que se muestra en la figura 2.8.

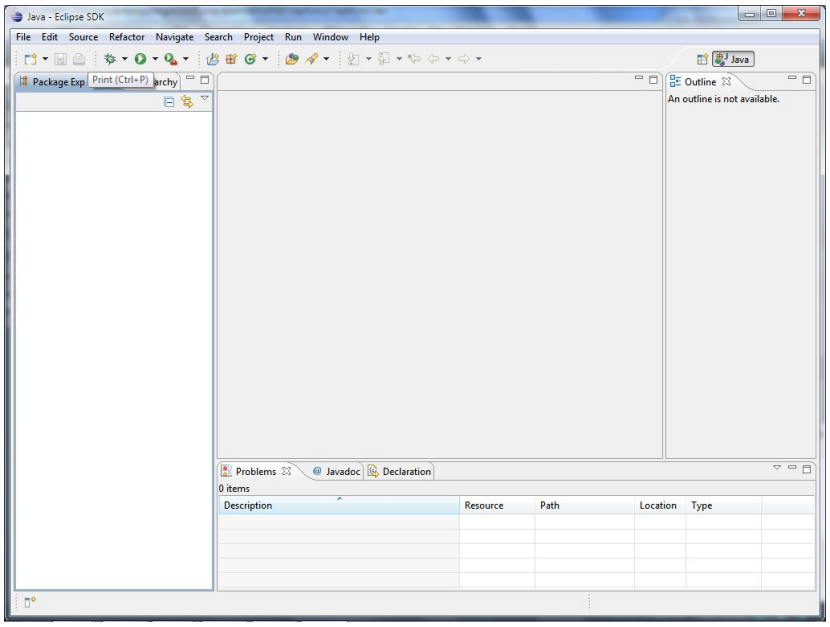

Figura 2.8: Entorno de trabajo habitual de Eclipse

2.2. Instalación del SDK de Android

Una vez que hemos instalado el entorno de desarrollo Eclipse, nos encontramos en disposición de instalar la SDK de Android, así como de configurar Eclipse para que dicha SDK funcione correctamente.

Lo primero que debemos hacer es obtener el *Android SDK*. Este programa instalará el SDK en su totalidad en nuestro sistema operativo. Podemos descargarlo desde la siguiente página web:

`http://developer.android.com/sdk/index.html`

En esa web nos aparecerá una página como se observa en la Figura 2.9.

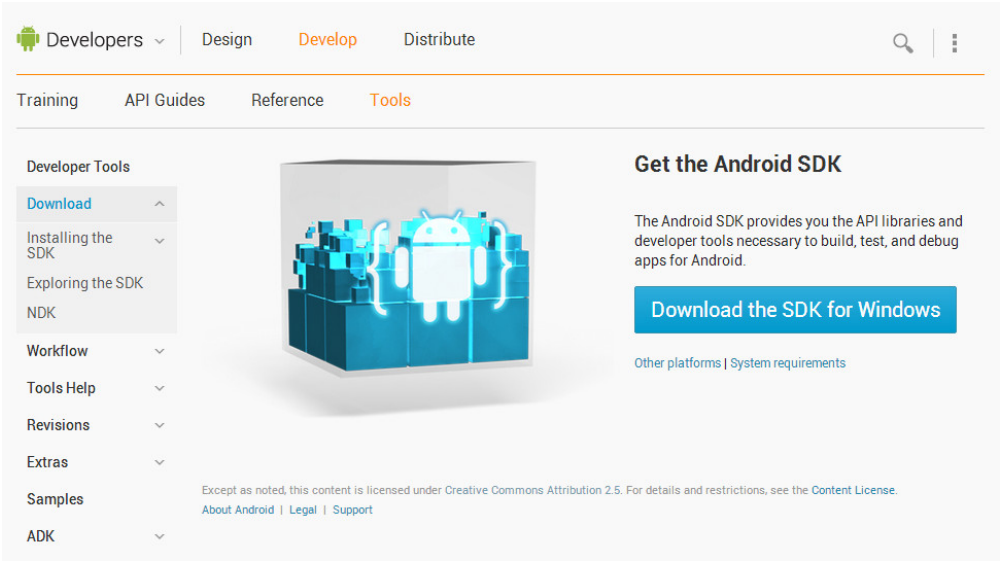

Figura 2.9: Obteniendo *Android SDK*

Una vez descargado el fichero, lo ejecutamos y seguimos los pasos de instalación, como podemos comprobar en la imagen de la figura 2.10.

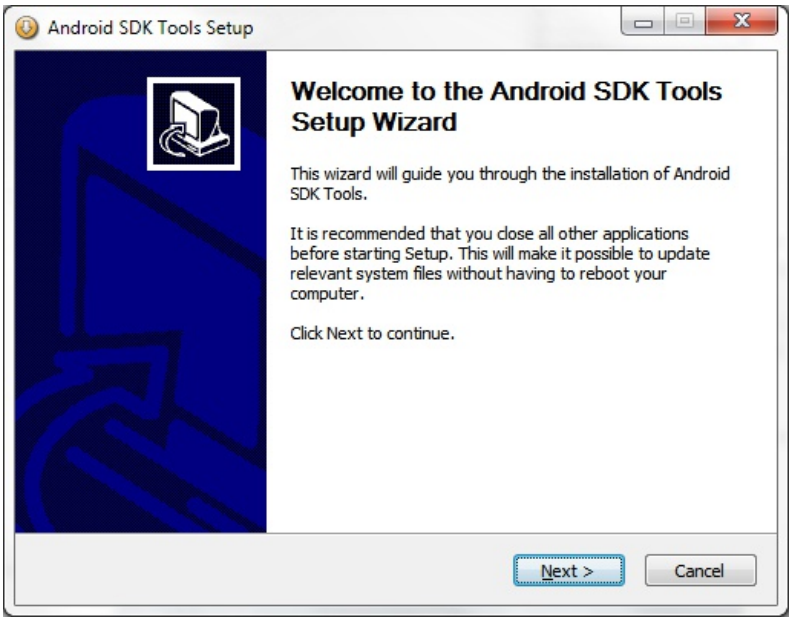

Figura 2.10: Iniciando la instalación de *Android SDK*

Durante el proceso de instalación se nos preguntará donde queremos almacenar la SDK. En nuestro caso, hemos decidido cambiar la ruta de instalación por defecto por la carpeta `C:\eclipse` a fin de tener toda la información ubicada en el mismo lugar (figura 2.11).

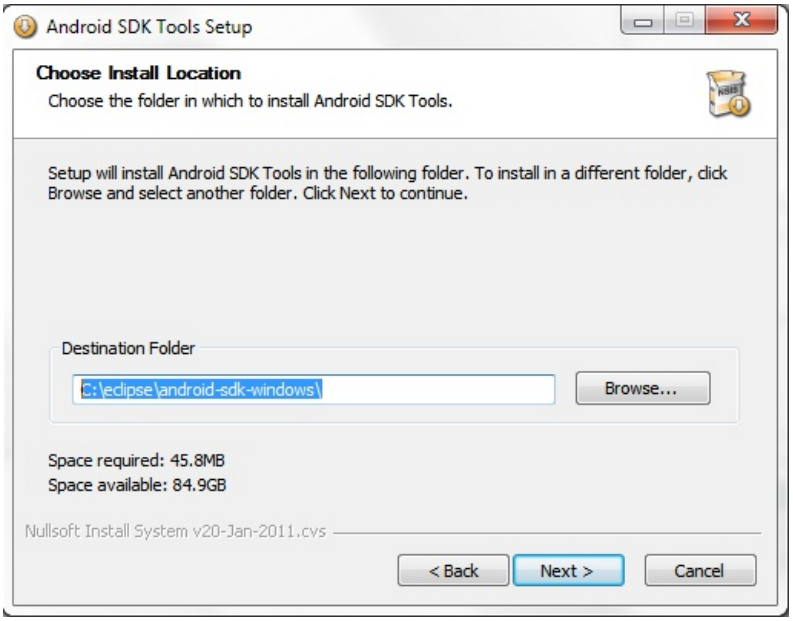

Figura 2.11: Especificando la ruta de instalación deseada para *Android SDK*

Una vez el programa copie todos los archivos y finalice este proceso, el instalador lanzará una ventana similar a la de la figura 2.12 en donde nos aparecen todas las plataformas disponibles que podemos descargarnos a fin de desarrollar nuestras aplicaciones.

Si disponemos de espacio suficiente en disco, es recomendable instalar todas las versiones de la SDK, lo que nos permitirá desarrollar aplicaciones compatibles con todas esas versiones de Android. De lo contrario, descargaremos la versión 2.3.3 completa (marcando el resto como Reject).

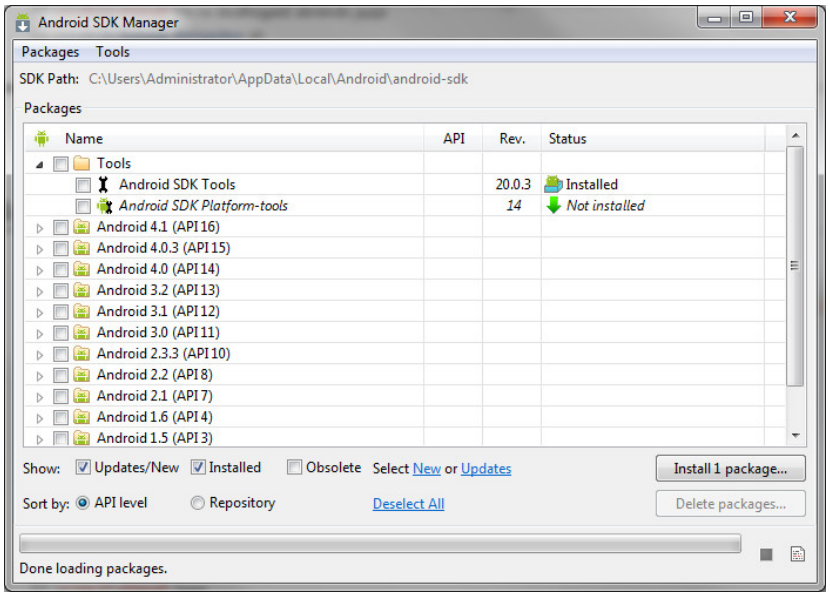

Figura 2.12: Opciones disponibles a instalar (I)

Pulsamos en Install (nos aparecerá una ventana como la que se muestra en la figura 2.13) y esperamos a que se descarguen e instalen los módulos seleccionados.

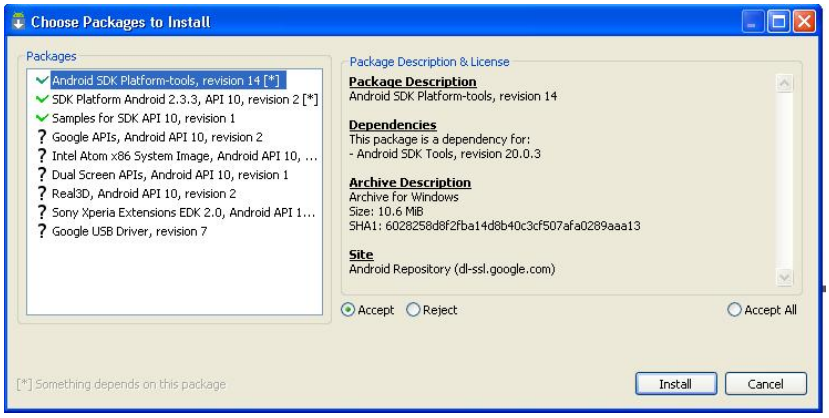

Figura 2.13: Opciones disponibles a instalar (II)

Tal y como se puede intuir por la imagen 2.12, lo que estamos haciendo es instalar diversas plataformas para el desarrollo. La razón de hacer esto viene dada porque en función del dispositivo móvil sobre el que queramos programar deberemos utilizar un entorno u otro.

Así, la mayoría de los móviles de última generación utilizan al menos la versión 2.2 de Android, mientras que los primeros dispositivos que aparecieron utilizaban versiones anteriores a la misma. Una aplicación desarrollada para Android 1.6 funciona correctamente en una versión posterior de Android, lo cual nos podría hacer pensar que lo ideal sería desarrollar siempre bajo versiones que permitan el mayor abanico de posibilidades a nivel de terminales físicos. Sin embargo, las funcionalidades que ofrece la versión 2.2 son mucho más amplias que las que ofrece la versión 1.6.

Realizar un buen análisis de requisitos y una comparativa de lo que ofrece cada versión sería importante de cara a llegar al máximo número de beneficiarios posibles. En el tema 1 ya se presentó un resumen de las capacidades de cada versión.

2.3. Instalación del plugin ADT para Eclipse

Ahora que ya tenemos correctamente instalado en nuestro ordenador tanto el entorno de desarrollo Eclipse, como el SDK de Android, el siguiente paso que debemos dar es el de unir ambos elementos. Para ello necesitaremos el Plugin ADT que extenderá la capacidad básica de Eclipse de forma que podamos crear nuevos proyectos Android, interfaces de usuario, añadir componentes, etc.

> **Nota.-** Los enlaces que se proporcionan a continuación son válidos para la versión 3.5 de Eclipse.

En primer lugar iniciamos Eclipse y dentro del menú de Help seleccionamos la opción Install New Software..., tal y como se observa en la figura 2.14.

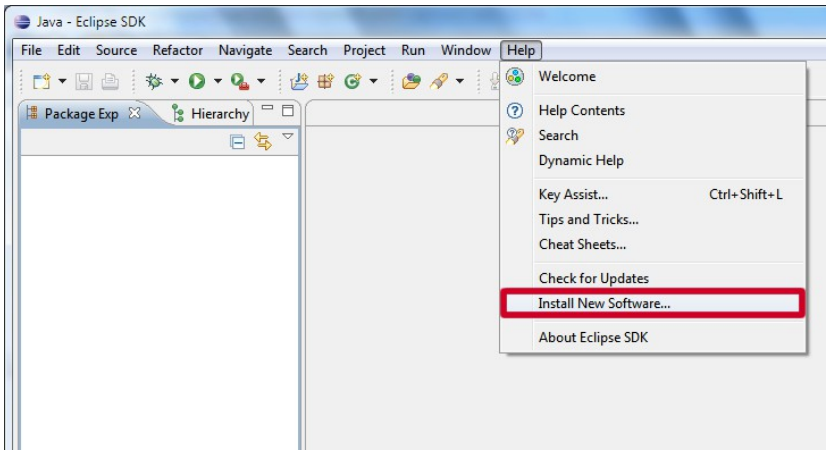

Figura 2.14: Instalando el plugin ADT en Eclipse (I)

Nos aparecerá una pantalla como la que se observa en la figura 2.15, donde debemos pulsar sobre el botón Add.

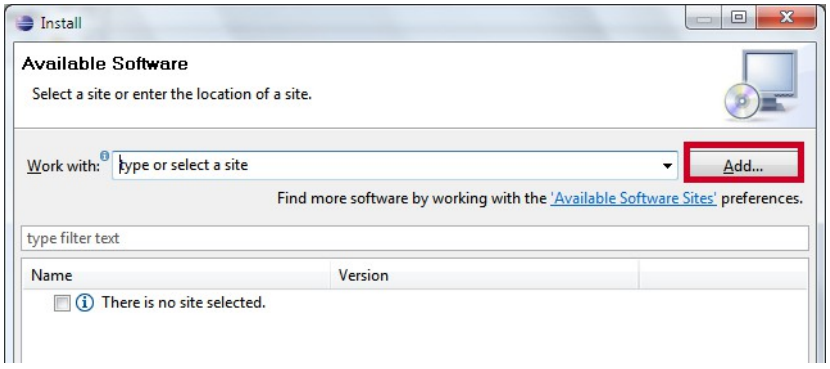

Figura 2.15: Instalando el plugin ADT en Eclipse (II)

Nuevamente, en la pantalla que nos aparece (véase la figura 2.16) insertamos en el campo Name el texto "`Android Plugin`" y en el campo Location la siguiente dirección web:

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

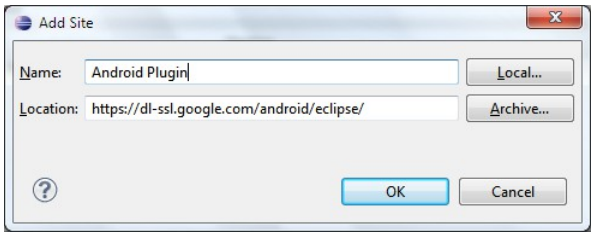

Figura 2.16: Instalando el plugin ADT en Eclipse (III)

La pantalla nos mostrará ahora una nueva entrada bajo el nombre de Developer Tools, como se muestra en la figura 2.17. Pulsamos entonces en la entrada para marcarla (si la desplegamos veremos que quedan seleccionados tanto Android DDMS, como Android Development Tools y Android Hierarchy Viewer, entre otros). Una vez hecho esto pulsamos sobre el botón Next.

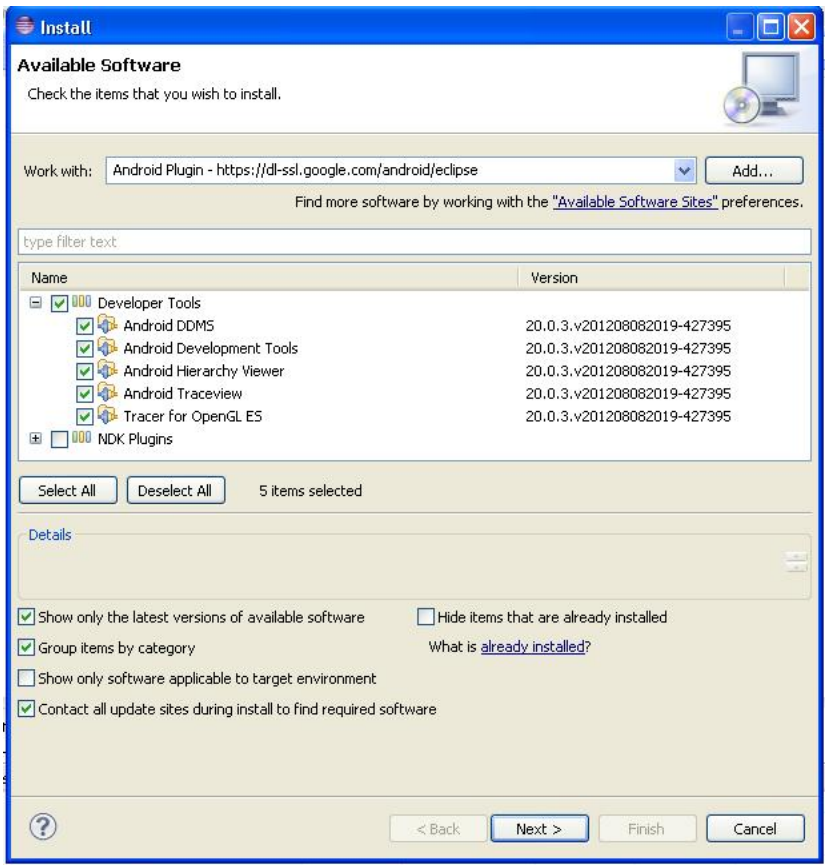

Figura 2.17: Instalando el plugin ADT en Eclipse (IV)

Nos aparecerán a continuación dos pantallas. Una de resumen de lo que vamos a instalar, donde pulsaremos sobre el botón Next, y otra en donde aparece la licencia de uso que debemos aceptar para a continuación pulsar sobre el botón Finish. El proceso de instalación del Plugin se inicia en ese momento, finalizando con la pantalla que se observa en la figura 2.18.

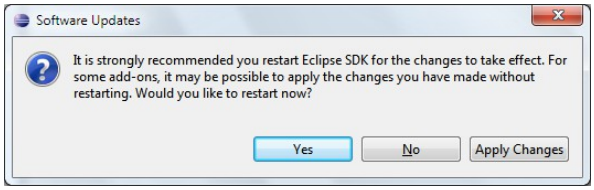

Figura 2.18: Instalando el plugin ADT en Eclipse (V)

Una vez se ha instalado el plugin ADT y hemos reiniciado Eclipse, el siguiente paso es indicarle al mismo donde está instalado Android en nuestro ordenador. Para ello, tal y como se observa en la figura 2.19, nos vamos dentro de Eclipse a Window → Preferences.

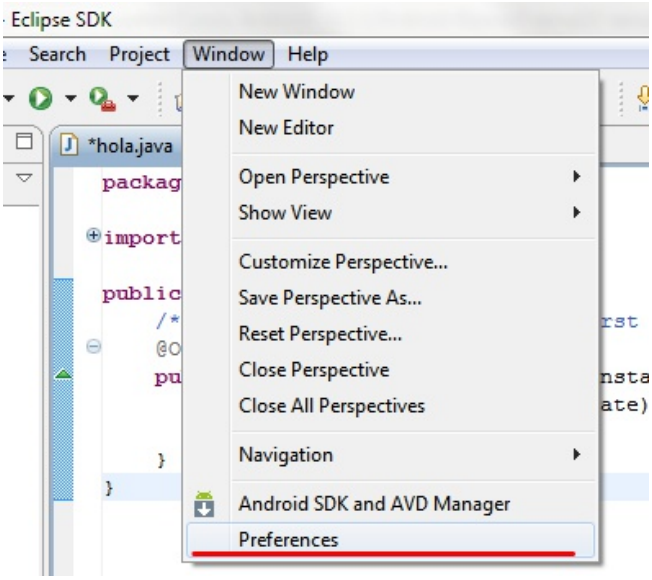

Figura 2.19: Configurando Eclipse y el plugin ADT (I)

En la pantalla que se nos abre seleccionamos la opción Android, como se puede comprobar en la figura 2.20. En el campo SDK Location debemos introducir la carpeta donde hemos guardado nuestro SDK de Android, que en nuestro caso es en `C:\eclipse\android-sdk-windows` y pulsar sobre el botón Apply para que cargue toda la instalación de Android.

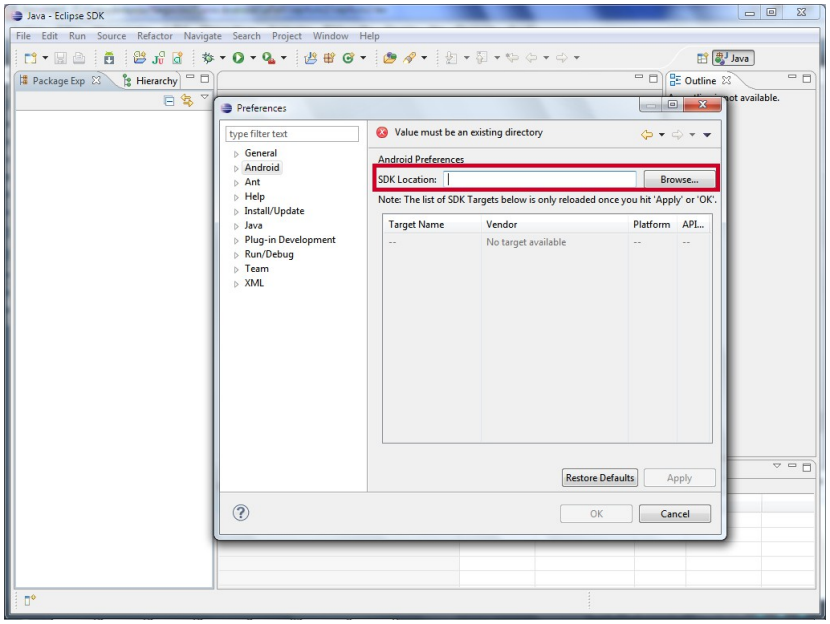

Figura 2.20: Configurando Eclipse y el plugin ADT (II)

Por último, tendremos una ventana como la que se observa en la figura 2.21, donde pulsaremos sobre el botón OK, terminando de esta manera la instalación y configuración de todos los elementos necesarios para poder empezar a desarrollar aplicaciones con Android.

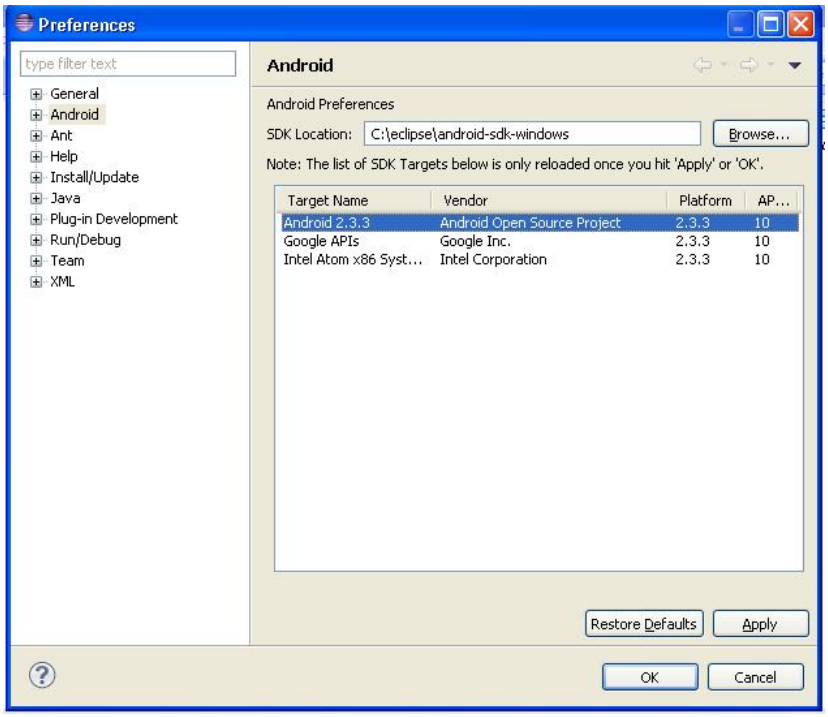

Figura 2.21: Configurando Eclipse y el plugin ADT (III)

2.4. Herramientas disponibles para el desarrollo en Eclipse

Android dispone de varias herramientas que facilitan la creación, la comprobación y la depuración de nuestros proyectos. Gracias a la instalación del plugin ADT en Eclipse podemos disponer de todas esas herramientas dentro del propio entorno de desarrollo. Lo primero que tenemos que observar en Eclipse, es que tras su instalación se ha creado un acceso directo a Android Manager con el que ya hemos trabajado anteriormente y otro para la gestión de los emuladores creados o Android Virtual Device Manager (figura 2.22).

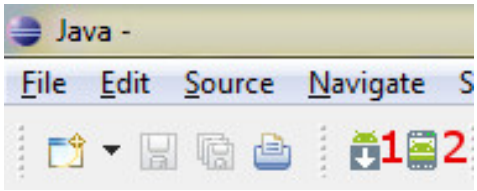

Figura 2.22: Acceso directo a Android Manager desde Eclipse

Además, cuando creemos un nuevo proyecto de Android, dispondremos de un emulador de dispositivo físico que se ejecutará directamente desde Eclipse y que crearemos de forma rápida haciendo uso del acceso directo comentado.

También dispondremos de una vista conocida como DDMS que nos permitirá observar y controlar la máquina virtual Dalvik ya comentada anteriormente mientras depuramos nuestras aplicaciones.

Por último, comentar que además de éstas, existen otras herramientas como una base de datos `SQLite3`, un conversor de clases Java a ficheros de bytes `.dex` de Android, etc.

Tema 3

Nuestro primer programa

En este tema vamos a crear nuestro primer programa para Android. Además, lo probaremos mediante el emulador que crearemos y estudiaremos las diferencias existentes entre el emulador y un terminal real.

3.1. Creación de la primera aplicación bajo Eclipse

En primer lugar, debemos pulsar en la opción File→New→Project de Eclipse, tal y como se observa en la figura 3.1.

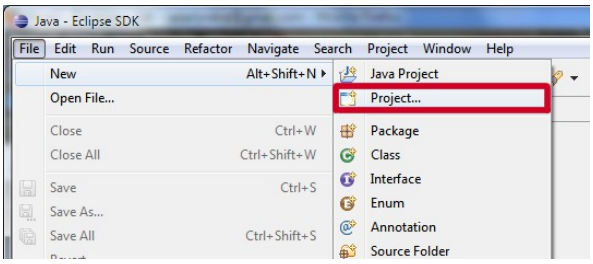

Figura 3.1: Creación de un nuevo proyecto de Android (I)

Nos aparecerá una pantalla (figura 3.2) donde seleccionamos Android Project tras abrir el desplegable con el nombre de Android. Por último, pulsamos sobre el botón Next.

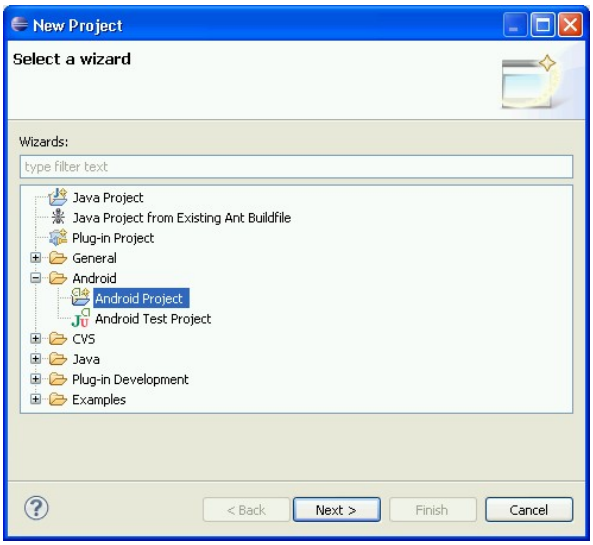

Figura 3.2: Creación de un nuevo proyecto de Android (II)

Esto nos dirige a la pantalla de especificación de los datos del proyecto, tal y como podemos ver en la figura 3.3.

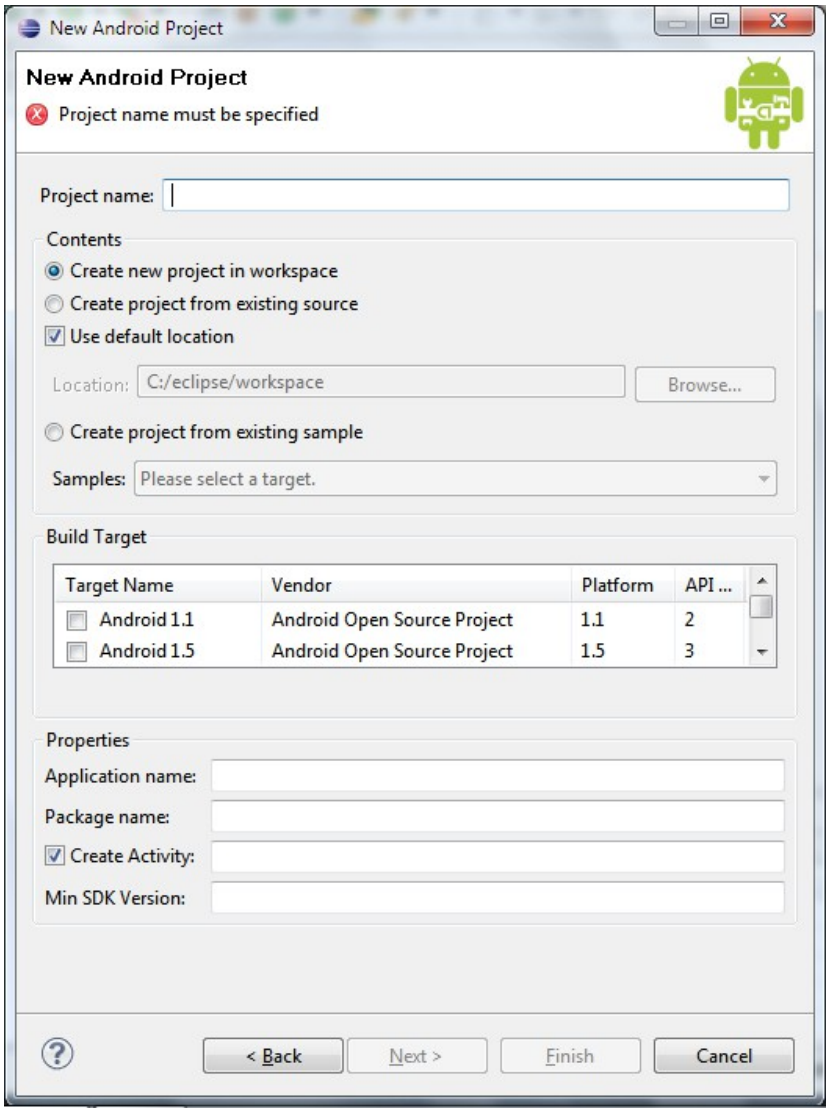

Figura 3.3: Creación de un nuevo proyecto de Android (III)

Lo primero que debemos hacer es especificar el nombre a nuestro primer programa. Como ya es tradición en el mundo informático le daremos el nombre de `HolaAndroid`.

Dejaremos la carpeta que viene dada por defecto para almacenar el proyecto pues así la tendremos localizada dentro de la carpeta de Eclipse.

A continuación indicaremos la versión de SDK con la que queremos trabajar. Dado que la mayoría de terminales que podemos encontrar en el mercado incluyen la versión 2.0 de Android o superior, elegiremos ésta para nuestro `HolaAndroid`.

En la parte inferior de la pantalla podemos observar cuatro campos de entrada. El primero de todos, Application Name nos permite definir un nombre amigable para nuestra aplicación, así que nosotros le pondremos como nombre `Hola Android`.

El segundo campo permite especificar el nombre del paquete donde se almacenarán los archivos que creemos. De forma habitual se suele especificar el nombre del paquete en dos partes, la primera la parte correspondiente a la organización a la que se pertenece y la segunda al proyecto en sí que estemos desarrollando. Nosotros hemos decidido ponerle como nombre a nuestro paquete `es.um.cursos.android`.

El tercer campo nos da la posibilidad de especificar el nombre de la clase que tendrá la responsabilidad de ser la actividad inicial. Nosotros hemos puesto `HolaAndroidActivity`.

Por último, Min SDK Version hace referencia a la compatibilidad de versiones Android para la cual tendrá valor nuestro software. Si ponemos por ejemplo 2.0 la aplicación no se ejecutará en terminales con versiones inferiores a la 2.0. En este caso lo dejamos vacío para indicar que queremos que se ejecute en todos los entornos posibles.

Una vez que está todo correctamente especificado pulsamos sobre el botón Finish y Eclipse nos creará el proyecto con todas las carpetas, tal y como se observa en la figura 3.4.

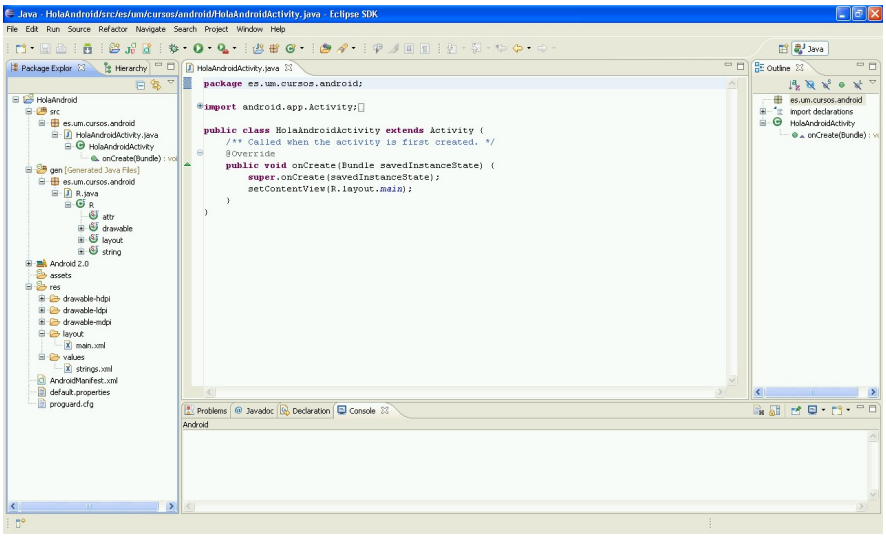

Figura 3.4: Creación de un nuevo proyecto de Android (IV)

Uno de los puntos interesantes que tiene crear el proyecto especificando el nombre de la clase en el paso anterior es que Eclipse va a crear un "HolaMundo" por defecto.

3.1.1. Configuración del emulador de Android

A continuación deberemos configurar en Eclipse el emulador de dispositivo físico que nos permita probar nuestras aplicaciones, y más concretamente la aplicación `HolaAndroid` que de forma automática se acaba de crear.

Figura 3.5: Integración Eclipse-Android SDK-Emulador

Pulsamos sobre el icono que nos abrirá el Android SDK and ADV Manager, tal y como se muestra en la figura 3.6.

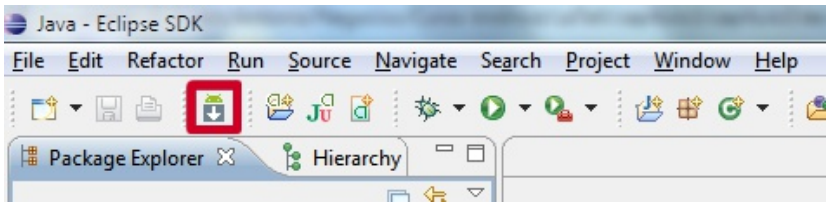

Figura 3.6: Configuración del emulador de Android (I)

Nos aparecerá entonces una pantalla como la observada en la figura 3.7, donde podemos comprobar que ya existe un dispositivo virtual de Android creado para la SDK 2.2.

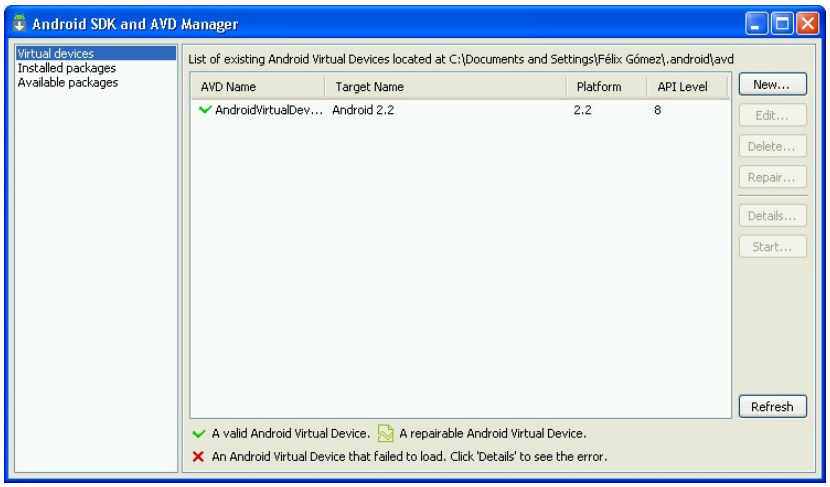

Figura 3.7: Configuración del emulador de Android (II)

Vamos a ver ahora cómo modificar las propiedades de este dispositivo virtual o emulador de Android. Para ello, seleccionamos dicho emulador y pulsamos sobre el botón Edit.... En la pantalla que nos aparece, tal y como podemos comprobar en la figura 3.8, observamos varios propiedades del emulador que pueden ser configuradas:

- El campo Name permite especificar el nombre del emulador
- En el campo Target se especifica la versión de Android que correrá en dicho dispositivo
- El campo SD Card - Size permite especificar el tamaño interno de la memoria del dispositivo
- Por último, también es posible definir el aspecto del dispositivo, así como otras características de su hardware

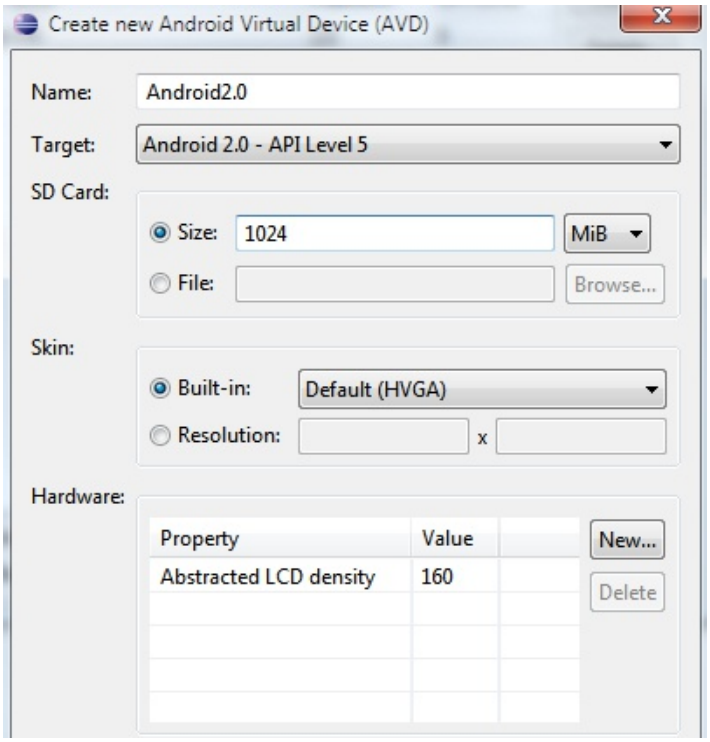

Figura 3.8: Configuración del emulador de Android (III)

3.1.2. Ejecutando nuestra aplicación

Antes de poder ejecutar finalmente nuestra aplicación en el emulador, debemos configurar el entorno de ejecución de Android en Eclipse. Para ello, pulsamos en la opción Run→Run Configurations..., tal y como se observa en la figura 3.9.

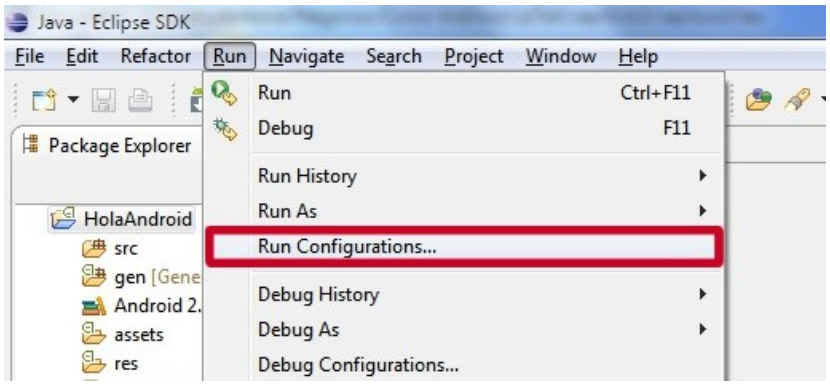

Figura 3.9: Configuración del entorno de ejecución de Android (I)

En la pantalla que nos aparece, mostrada en la figura 3.10, debemos seleccionar la opción Android Application y pulsar sobre el botón New.

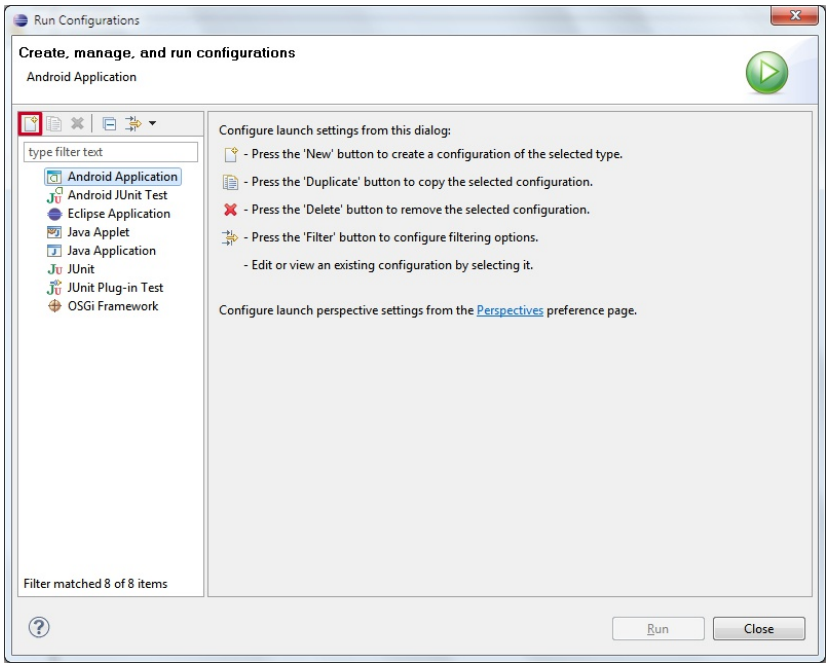

Figura 3.10: Configuración del entorno de ejecución de Android (II)

Se nos mostrará entonces la pantalla que se puede observar en la figura 3.11, donde deberemos completar la información requerida en cada una de las tres pestañas, a saber: Android, Target y Common.

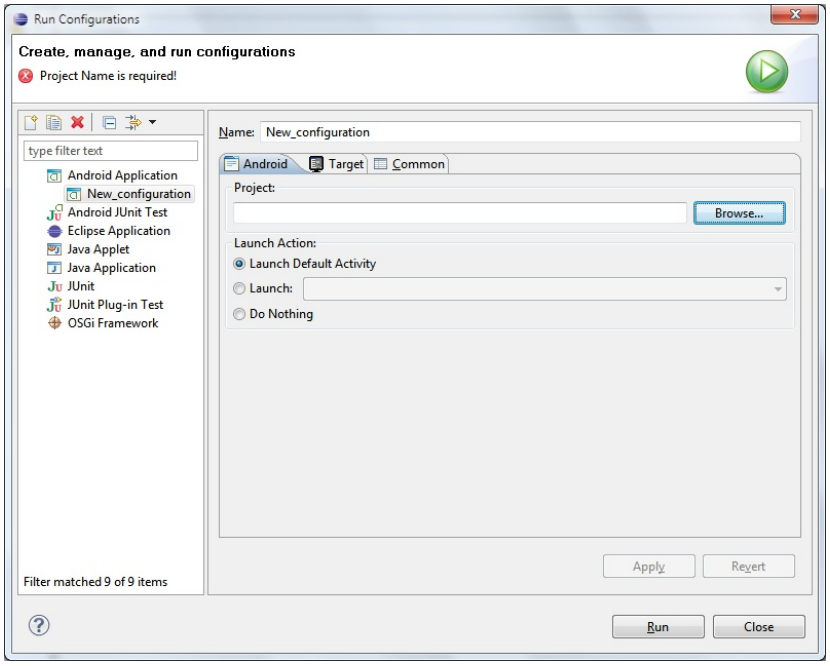

Figura 3.11: Configuración del entorno de ejecución de Android (III)

En primer lugar le damos un nombre a la configuración del entorno de ejecución. En nuestro caso será `Android2.2`. A continuación, en la pestaña Android especificamos el proyecto pulsando sobre el botón Browse y seleccionando `HolaAndroid`. En la pestaña Target especificamos el AVD Android 2.2 visto anteriormente. Y por último, en la pestaña Common no modificamos ningún parámetro.

Pulsamos entonces sobre el botón Apply y finalmente en el botón Run que ejecutará nuestra aplicación en el emulador seleccionado.

> **Nota.-** El emulador puede tardar varios minutos en ejecutarse (entre 1 y 3 minutos, aproximadamente). Sin embargo, no es necesario reiniciarlo cada vez que queramos probar un cambio en nuestra aplicación. Bastará con guardar el cambio y pulsar en el botón Run.

Y finalmente, obtenemos un resultado como el que se observa en la figura 3.12, donde se muestra la aplicación `HolaAndroid` ejecutándose sobre el emulador configurado previamente.

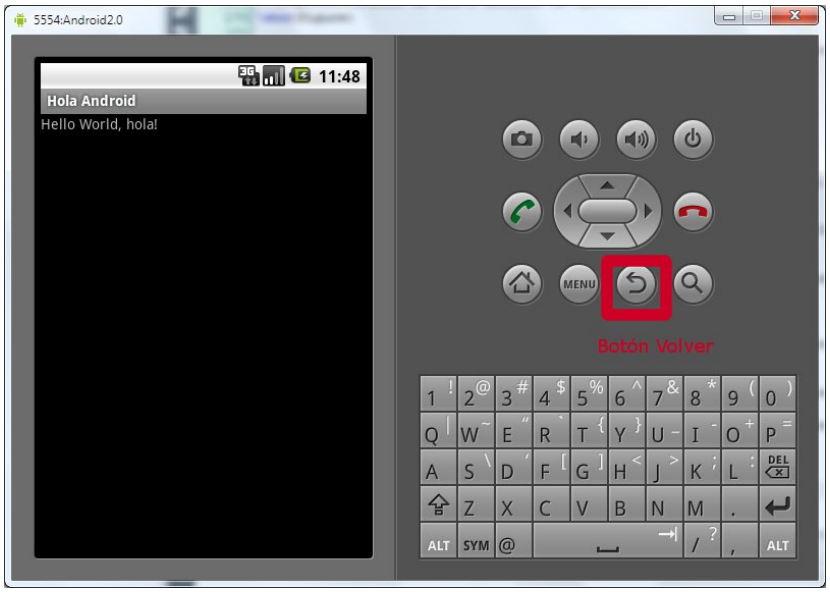

Figura 3.12: Ejecución de la aplicación HolaAndroid (I)

Actividad propuesta I.1. *Se recomienda al alumno que explore por sí mismo el entorno de desarrollo Eclipse, el Android SDK and ADV Manager, así como el emulador de Android, para familiarizarse con los mismos, si no los conocía con anterioridad.*

Figura 3.13: Ejecución de la aplicación HolaAndroid (II)

3.2. Explorando nuestra primera aplicación

Una aplicación Android consta de dos componentes fundamentales: por una parte tendremos siempre un fichero `.java` que contendrá las acciones que implementará nuestro programa, y por otra parte, también tendremos un fichero `.XML` que servirá para especificar el diseño que la interfaz gráfica del usuario tendrá.

Vamos a explicar brevemente a continuación qué hay detrás del código `.java` y del fichero `.XML` que se han generado con nuestra primera aplicación `HolaAndroid`.

3.2.1. Fichero `.java`

Comencemos con el fichero `.java`. En Eclipse, abriremos dentro del panel Package Explorer la carpeta `src/es.um.cursos.android` y dentro de ella, el fichero `hola.java`. El código que podemos ver en el mismo es el siguiente:

```
package es.um.cursos.android;

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

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

La clase `Activity`[1] es la clase base o raíz para los componentes visuales; de ahí que nuestro programa *extienda* dicha clase. Obsérvese que se debe importar la misma a través del paquete `android.app`.

De igual forma, se sobreescribe el método `onCreate()` de forma que se establezca como interfaz gráfico la ***vista*** que se ha creado de forma automática en la carpeta `res/layout` con el nombre `main.xml` (llamada al método `setContentView()`).

3.2.2. Fichero `.xml`

El contenido del fichero `main.xml` generado con nuestra aplicación `HolAndroid` es el siguiente:

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

[1] http://developer.android.com/reference/android/app/Activity.html

Cada etiqueta de este fichero XML define una parte de la interfaz gráfica del usuario y puede contar con una serie de propiedades adicionales que se especifican como atributos de cada una de dichas etiquetas, tal y como se puede ver en el fichero `main.xml`.

Así por ejemplo, un `LinearLayout`[2] representa a nivel gráfico a un espacio horizontal que puede contener objetos, ya sean texto, botones, etc., y que cuenta con las siguientes propiedades: orientación, anchura y altura.

Además, XML es un lenguaje *inclusivo* en el sentido de que una etiqueta puede estar dentro de otra etiqueta, como en el caso del ejemplo. En el que el `LinearLayout` contiene un campo de texto `TextView`[3].

Actividad propuesta I.2. *Crear, compilar y ejecutar la aplicación `HolaAndroid` tal y como se ha explicado en este tema. Se recomienda al alumno que explore por sí mismo tanto el código del fichero `.java`, como el contenido del fichero `.xml` y que pruebe a hacer pequeñas modificaciones sobre éste último para ver el efecto que tienen.*

A modo de ejemplo, a continuación se proporcionan algunos valores de entrada posibles para cada propiedad que aparece en el fichero.

Propiedad	Valores
`orientation`	{`vertical`, `horizontal`}
`layout_width`	{`fill_parent`, `wrap_content`}
`layout_heigth`	{`fill_parent`, `wrap_content`}
`text`	{Any string}

Tabla 3.1: Posibles valores para ciertas propiedades gráficas de una aplicación Android

3.3. Tipos de aplicaciones a desarrollar en Android

La mayoría de aplicaciones que podemos desarrollar en Android se enmarcan dentro de cuatro categorías principales:

- **Aplicaciones de primer plano.** Son aquellas aplicaciones que sólo serán útiles cuando se estén ejecutando en primer plano y estarán cerradas cuando no estén activas como por ejemplo un juego o un mapa.
- **Aplicaciones en segundo plano.** Serán aplicaciones con una limitada interacción por parte del usuario, que deben estar ejecutándose la mayoría del tiempo ocultas. Un ejemplo de este tipo de aplicaciones puede ser un software de recepción de correo electrónico que está comprobando periódicamente si hay nuevos emails sin intervención del usuario.
- **Aplicaciones intermitentes.** Son aplicaciones que también trabajan en segundo plano, pero que a diferencia de las anteriores requieren intervención del usuario para cambiar su estado. Como por ejemplo un reproductor multimedia.
- **Widgets.** Son aplicaciones representadas únicamente en la pantalla de inicio del usuario.

Estas categorías no están cerradas, permiten dar una visión global de los requisitos que puede tener cualquier aplicación. Es importante conocerlas, pues Android manejará los recursos en función de cómo sean los programas. De ahí que un buen análisis previo sobre la aplicación que queramos desarrollar nos sirva de gran ayuda como veremos más adelante.

[2] http://developer.android.com/reference/android/widget/LinearLayout.html
[3] http://developer.android.com/reference/android/widget/TextView.html

3.4. Diferencias entre el desarrollo en Android SDK y en terminales físicos

Para terminar este tema, en esta sección se expondrá un pequeño guión de buenas prácticas a la hora de desarrollar aplicaciones bajo Android.

Desarrollar aplicaciones Android en un potente ordenador de sobremesa nos resultará siempre cómodo y sencillo. No obstante, no debemos nunca olvidar que estamos programando no para potentes ordenadores, sino para dispositivos móviles que tienen un tamaño de pantalla limitado, una pequeña memoria, un procesador de poca potencia, un alto coste (en términos de consumo de batería) sobre las transferencias de datos, y una batería de duración limitada.

Seamos eficientes

Los fabricantes de terminales físicos generalmente valoran más (pues así lo demanda el mercado) el tener terminales pequeños con una gran batería que la mejora en las capacidades de procesamiento.

En la práctica, esto significa que debemos tener mucho cuidado cuando escribamos código, pues necesitaremos optimizar el mismo al máximo para que nuestro programa se ejecute de forma rápida y limpia si queremos que aquellos que puedan obtener nuestro software no se cansen de él rápidamente. A lo largo de lo que queda de este curso, iremos incidiendo en pequeños consejos sobre buenas prácticas de programación que se deberían tratar de seguir siempre que fuera posible.

Esperemos siempre que la memoria sea muy limitada

Un dispositivo físico puede a día de hoy albergar gran cantidad de información gracias al incremento de las capacidades en las memorias Flash o SD. No obstante, debemos ser realistas: en un porcentaje muy elevado, los usuarios de este tipo de dispositivos suelen utilizar el espacio para música u otros contenidos multimedia, por lo que pensar que podemos disfrutar de una gran capacidad de almacenamiento puede convertirse en una mala idea.

Adicionalmente, los dispositivos Android ofrecen restricciones a la hora de instalar aplicaciones en su memoria interna, de ahí que el tamaño del software compilado sea un punto importante incluso más importante que el uso correcto de los recursos.

Las técnicas para optimizar el almacenamiento de los datos de las aplicaciones que realicemos quedan fuera del alcance de este curso de iniciación a Android.

Diseña siempre para pantallas pequeñas

Si, como veíamos antes, las compañías trabajan por reducir el tamaño de los terminales, no parece una idea demasiado buena diseñar interfaces de usuario que sean grandes y sólo visibles en terminales con pantallas de gran tamaño. Por ello, diseñar la interfaz de forma intuitiva, tratando de que la misma aparezca lo más centrada posible y con el menor número de controles posibles se convierte en un reto importante de cara al desarrollador.

Por supuesto, nuestro diseño tendrá que ser acorde a la lógica del mundo informático: todo debe ser compatible. De ahí que nuestros diseños tengan que tener la **capacidad dinámica** de adaptarse a pantallas más grandes.

Intentaremos por tanto, no hacer diseños estáticos que puedan verse muy pequeños en grandes terminales.

Espera siempre bajas tasas de transferencia por Internet y altas latencias

Internet en el mundo de los teléfonos móviles no se encuentra en un estadio tan avanzado como en el de los ordenadores de sobremesa. De ahí que un buen consejo será el de hacer nuestras aplicaciones teniendo siempre en cuenta que los costes de transferencia en un terminal físico son siempre muy elevados, y que siempre que esté en nuestra mano debemos tratar de reducirlos.

El emulador de nuestro terminal físico nos permite especificar estos parámetros llegado el momento.

Los principios básicos de rendimiento según Android

Y por último, y ya para finalizar, recordemos, tal y como dice el manual de buenas prácticas de la web de Android, las dos reglas básicas para escribir código eficientemente:

- No hagas el trabajo que no necesites hacer.
- No asignes memoria si se puede evitar.

Tema 4

¿Qué es una Actividad?. Aplicaciones

4.1. Los componentes de una aplicación

Un programa escrito para Android está formado por un conjunto de componentes débilmente acoplados, vinculados por un **manifiesto**, que será el encargado de describir cada componente y la forma en la que interactúan unos con otros.

El manifiesto de la aplicación además también contendrá los metadatos de la misma y los requisitos de hardware y plataforma.

Los componentes de los que hablamos que componen la aplicación son los siguientes:

- **Actividades.** Representan a la capa de presentación de las aplicaciones. Cada pantalla de la aplicación que diseñemos debe ser una clase que herede de la clase `Activity`. Las actividades utilizan **vistas** y **layouts** para crear las interfaces gráficas de los usuarios que muestran información y se actualizan ante las acciones de los mismos. La mayoría de actividades se definen para que ocupen toda la pantalla, aunque también se pueden crear actividades flotantes o semitransparentes.

- **Servicios.** Los componentes que representan servicios se ejecutan siempre en segundo plano, y se dedican a mantener al día fuentes de datos, mostrar notificaciones, y actualizar los componentes Actividad que acabamos de ver. Se utilizan sobre todo para realizar procesos regulares que necesitan estar activos aún cuando los componentes Actividad no se encuentren visibles o activos.

- **Proveedores de contenido.** Estos componentes se utilizan fundamentalmente para interactuar con las aplicaciones de bases de datos. De igual forma también se utilizan para compartir datos entre aplicaciones. Esto significa por tanto, que nosotros podemos perfectamente crear nuestro componente proveedor de contenido de forma que permitamos a otras aplicaciones acceder a nuestra información y, de forma inversa, esto también significa que nosotros podremos acceder a la información que otras aplicaciones tengan disponible a través de sus proveedores de contenido.

- **Paso de mensajes.** Los componentes paso de mensajes se utilizan para difundir mensajes a varios componentes, o bien a un componente Actividad o Servicio indicando la intención de realizar una acción. Se conocen también como **intents**.

- **Recepción de mensajes.** Los componentes recepción de mensajes disponen de criterios para filtrar los mensajes que deben atender para realizar la acción que indique el mensaje.

- **Widgets.** Los componentes widget son componentes visuales que pueden ser añadidos a la pantalla de inicio del terminal físico.

- **Notificaciones.** Los componentes notificaciones sirven para dar avisos a los usuarios interrumpiendo lo que estén haciendo. Un ejemplo de este tipo de componente podría ser el aviso de batería agotada.

Como podemos ver son bastantes los componentes de los que disponemos a la hora de diseñar nuestra aplicación. A medida que la complejidad de la misma aumente será necesario el uso de componentes de varios tipos, pero veremos no obstante, que el punto de partida que debemos conocer y dominar se encuentra en los componentes Actividad. Veamos por tanto cómo el Manifiesto sirve de nexo de unión entre todas las piezas del puzzle.

4.2. El manifiesto de aplicación (I)

Como decíamos anteriormente, el manifiesto de la aplicación nos permite definir la estructura de nuestra aplicación, sus componentes y sus requisitos. Si nos vamos a Eclipse, a nuestro proyecto `HolaAndroid` y abrimos el fichero `Manifest.xml` desde el Package Explorer veremos que Eclipse nos muestra una pantalla como la que se observa en la figura 4.1.

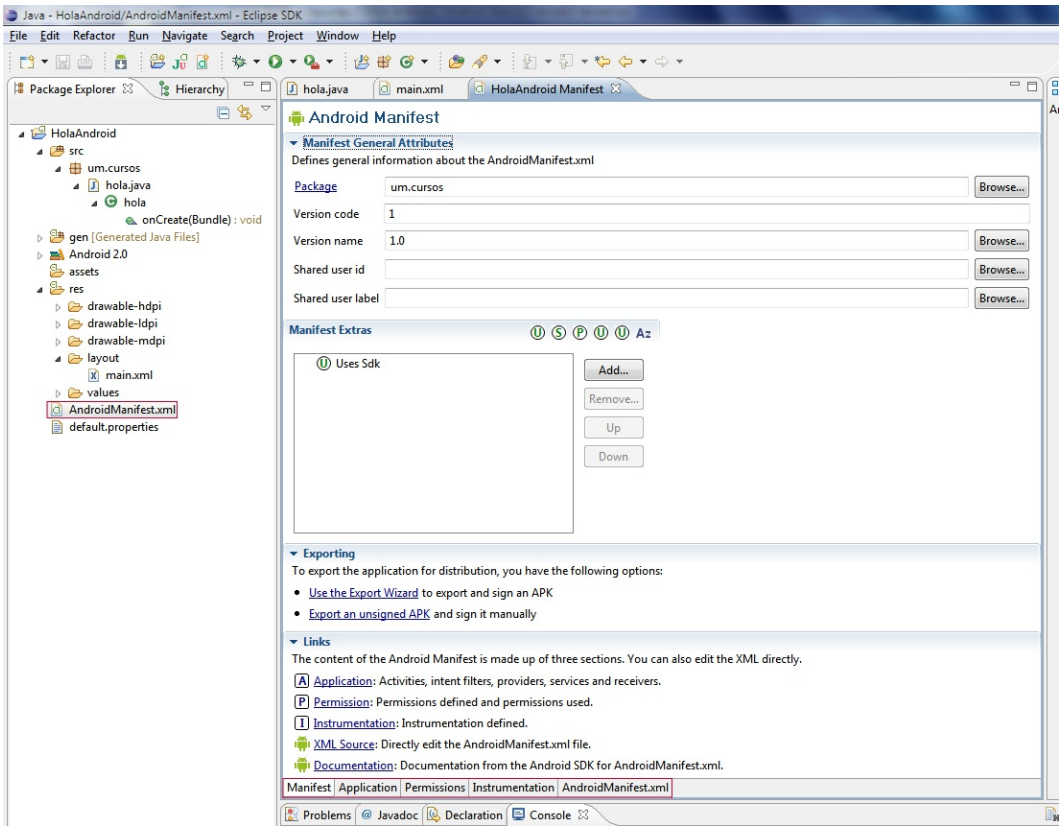

Figura 4.1: Manifiesto de la aplicación (I)

En las pestañas que hemos marcado en rojo en la imagen (salvo la última) se pueden modificar de forma amigable todas las propiedades que se muestran directamente en formato XML en la ultima pestaña AndroidManifest.xml.

No obstante, para poder utilizarlas sin tocar la última pestaña debemos entender qué suponen cada una de estas líneas de código:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="um.cursos"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".hola"
                  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>
```

De forma general, el fichero de manifiesto se divide en nodos o bloques en los que cada nodo puede (o no) contener otros nodos o bloques.

Podemos ver esta idea en la figura 4.2, donde el nodo que representa la etiqueta <manifest> contiene al nodo <application> que a su vez contiene al nodo <activity>.

Figura 4.2: Manifiesto de la aplicación (II)

Dando un vistazo rápido podemos ver que la etiqueta <application> permite propiedades como la especificación del icono que tendrá nuestra aplicación en el dispositivo o la etiqueta de texto que se mostrará debajo del mismo. En nuestro proyecto se muestra el icono por defecto para todas las aplicaciones y se muestra como nombre de la aplicación "Hola Android" tal y como le especificamos cuando creamos el mismo.

Las propiedades que se encuentran por encima de la etiqueta <application> definen en este caso la versión de nuestra aplicación en código numérico (android:versionCode=''1''), así como su correspondiente nombre (android:versionName=''1.0''). Esto nos permite ir modificando el código numérico de nuestra aplicación a medida que vayamos aplicando cambios sin tener que modificar el nombre de la versión.

La expresión xmlns:android=''http://schemas.android.com/apk/res/android'' obliga al manifiesto a seguir la estructura definida por Android para un manifiesto de aplicación (define la sintaxis del fichero XML).

Por último, el bloque definido por la etiqueta <intent-filter> permite determinar cómo se interactúa con otros componentes. Veremos esta etiqueta con más detalle posteriormente.

El manifiesto de una aplicación no se queda únicamente en lo que acabamos de ver. Puede incluir mucha más información que es importante conocer; para ello la figura 4.3 detalla una imagen que ilustra la estructura de nodos que puede contener un fichero de manifiesto.

Figura 4.3: Estructura del fichero `Manifest.xml`

A continuación se describirán brevemente cada uno de los nodos del fichero de manifiesto[1] mostrados en la figura 4.3.

4.2.1. Nodo <uses-sdk>

Esta etiqueta permite definir la versión mínima, máxima y recomendada de SDK que debe tener el dispositivo físico para que nuestra aplicación funcione correctamente. El valor que se debe introducir debe ser el `API Level` correspondiente a la versión de la plataforma, según se indica en la tabla 4.1.

Platform Version	API Level
Android 2.2	8
Android 2.1	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

Tabla 4.1: Valores posibles para el nodo <uses-sdk>

[1] http://developer.android.com/guide/topics/manifest/manifest-intro.html

Un ejemplo de uso de esta etiqueta podría ser el siguiente:

```
<uses-sdk
    android:minSdkVersion="4"
    android:targetSdkVersion="5" />
```

4.2.2. Nodo <uses-configuration>

Mediante este nodo podemos especificar los **mecanismos de entrada** que necesitará nuestra aplicación, como por ejemplo una pantalla táctil.

A modo de ejemplo, a continuación especificamos varias configuraciones soportadas posibles para la aplicación (una configuración de teclado completo y la otra con teclado de teléfono convencional):

```
<uses-configuration
    android:reqTouchScreen=["finger"]
    android:reqNavigation=["trackball"]
    android:reqHardKeyboard=["true"]
    android:reqKeyboardType=["qwerty"] />

<uses-configuration
    android:reqTouchScreen=["finger"]
    android:reqNavigation=["trackball"]
    android:reqHardKeyboard=["true"]
    android:reqKeyboardType=["twelvekey"] />
```

4.2.3. Nodo <uses-feature>

Con <uses-feature> especificamos las **características hardware** que requiere nuestra aplicación. Fijémonos en que el nodo anterior se centra en determinar las configuraciones necesarias para la entrada de datos, mientras que aquí especificaremos si necesitamos que nuestro terminal físico disponga de una cámara o bluetooth, por ejemplo. Estos ejemplos se escribirían así:

```
<uses-feature android:name="android.hardware.bluetooth" />
<uses-feature android:name="android.hardware.camera" />
```

4.2.4. Nodo <supports-screens>

Esta etiqueta ha cobrado una mayor importancia desde que los últimos dispositivos que cuentan con una pantalla WVGA(480x800) se han mezclado con los dispositivos HVGA(320x480). La tabla 4.2 muestra las resoluciones y densidades habituales de pantalla.

	Low density (120)	**Medium density (160)**	**High density (240)**
Small screen	QVGA (240x320)		
Normal screen	WQVGA400 (240x400)	HVGA (320x480)	WVGA800 (480x800)
Large screen		WVGA800 (480x800)	

Tabla 4.2: Resoluciones y densidades de pantalla habituales

Un ejemplo:

```
<supports-screens android:smallScreens=["true"]
                  android:normalScreens=["false"]
                  android:largeScreens=["false"]
                  android:anyDensity=["true"] />
```

4.2.5. Nodo <application>

Un manifiesto **sólo puede contener un nodo** <application>. Dentro de sus atributos, podemos definir como veíamos antes, el título de la aplicación, el icono que se utilizará, y el tema. Además, el nodo <application>, como ya hemos visto, también actúa de contenedor para los componentes de la aplicación. Finalmente indicar que disponemos del atributo `debuggable` que activa la posibilidad de depurar el proyecto. En este caso podemos ver el ejemplo de nuestra propia aplicación `Hola Android`.

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
```

4.2.6. Nodo <activity>

Debemos tener claro que necesitaremos un nodo activity por cada pantalla mostrada en nuestro dispositivo. Este punto es sumamente importante, pues si en nuestro fichero `manifest.xml` no quedan todas nuestras pantallas correctamente especificadas, nos encontraremos con una excepción en tiempo de ejecución. Cada nodo <activity> contendrá además una etiqueta <intent-filter> que identifica qué mensaje será el que lance la actividad y la categoría del mismo, tal y como se muestra en el siguiente ejemplo:

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

4.2.7. Nodo <service>

De igual forma que hemos dicho para los nodos <activity> debemos hacer con los nodos <service>: Uno por cada servicio que utilicemos.

```
<service android:enabled=["true"] android:name="servicio">...</service>
```

4.2.8. Nodo <provider>

Como ya dijimos anteriormente, un componente proveedor de contenido proporciona acceso a la gestión de bases de datos. Algunos de sus atributos son:

```
<provider android:authorities="list"
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:grantUriPermissions=["true" | "false"]
          android:icon="drawable resource"
          android:initOrder="integer"
          android:label="string resource"
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:permission="string"
          android:process="string"
          android:readPermission="string"
          android:syncable=["true" | "false"]
          android:writePermission="string" >
    . . .
</provider>
```

4.2.9. Nodo <receiver>

Un nodo <receiver> responde a un componente receptor de contenido de acceso a la gestión de bases de datos. Lo que se persigue conseguir con estos componentes es que actúen como si fueran *Event Listeners*. Algunos de sus atributos son:

```
<receiver android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    . . .
</receiver>
```

4.2.10. Nodo <uses-permission>

Este nodo forma parte del modelo de seguridad en Android y en él se declaran los permisos que se determinan para la aplicación de forma que ésta funcione correctamente. Debemos tener en cuenta, que el sistema operativo solicitará permisos para todas aquellas operaciones que puedan suponer un riesgo como pueda ser por ejemplo una llamada o una recepción de un mensaje SMS. Su estructura es la siguiente:

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

4.2.11. Nodo <permission>

Este nodo se utiliza para dar seguridad a aquellos componentes de nuestra aplicación que queramos hacer "compartibles" con el resto de aplicaciones. Estableciendo permisos les decimos a esas aplicaciones lo que pueden o no pueden hacer con dichos componentes.

```
<permission android:description="string resource"
            android:icon="drawable resource"
            android:label="string resource"
            android:name="string"
            android:permissionGroup="string"
            android:protectionLevel=["normal" | "dangerous" |
                                     "signature" | "signatureOrSystem"] />
```

4.2.12. Nodo <instrumentation>

El nodo instrumetation se utiliza para controlar a modo de test la aplicación. Como ejemplo podemos ver el siguiente caso:

```
<instrumentation android:label="Mi test"
                 android:name="test"
                 android:targetPackage="es.um.cursos.android.package" />
```

Actividad propuesta I.3. *Modifica el fichero `Manifest.xml` de la aplicación `HolaAndroid` para conseguir que una aplicación se ejecute bajo una versión de Android superior a la 1.6 como mínimo, no supere la versión 2.2 como máximo y que tenga en cuenta que dicha aplicación se ha diseñado para que funcione correctamente bajo la versión 2.0 de Android.*

Actividad propuesta I.4. *Modifica el fichero `Manifest.xml` de la aplicación `HolaAndroid`) para especificar que nuestra aplicación necesita utilizar Bluetooth, cámara, micrófono y WiFi.*

4.3. El manifiesto de aplicación (II)

Una vez que hemos descrito el contenido del fichero de manifiesto, y conocemos el significado de la mayoría de sus nodos, vamos a hacer uso de las facilidades que ofrece Eclipse para editar toda esa información.

Para ello vamos a ver a continuación el contenido de cada una de las demás pestañas de la ventana que muestra el manifiesto de una aplicación de Android, a saber: Manifest, Application, Permissions e Instrumentation.

4.3.1. Pestaña Manifest

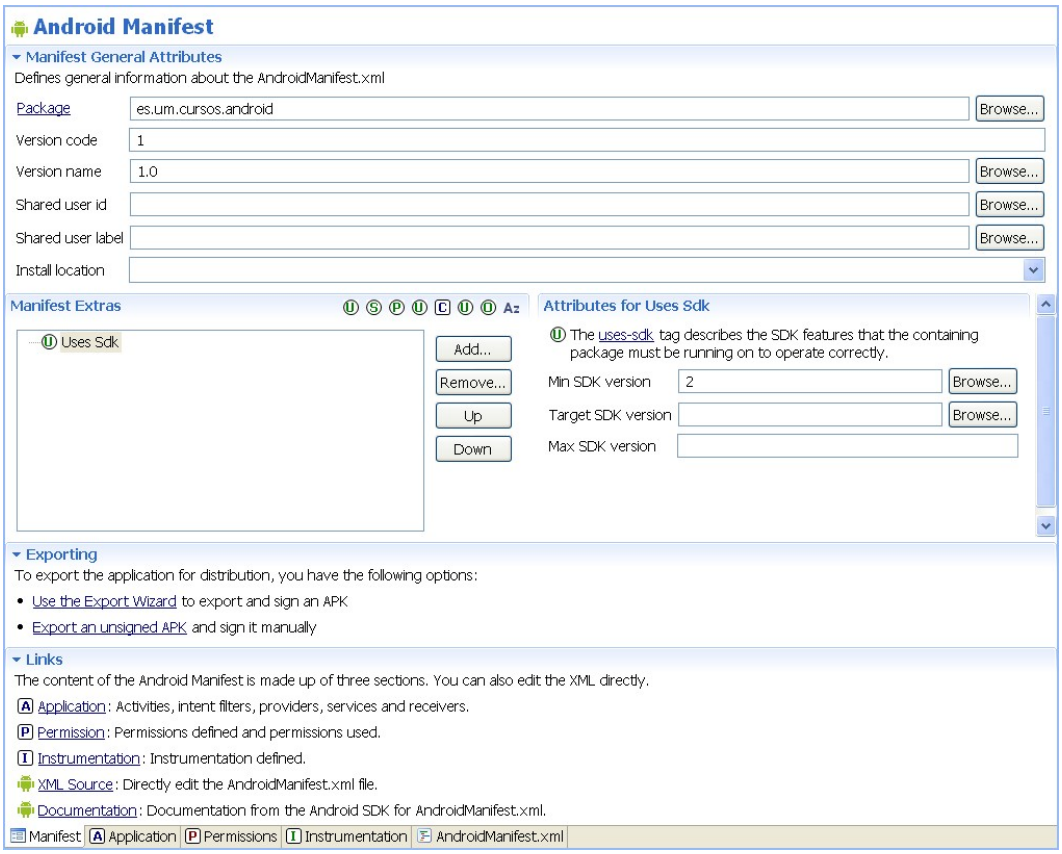

Figura 4.4: Manifiesto: Pestaña Manifest

La figura 4.4 muestra el contenido de la pestaña Manifest de la ventana de manifiesto de una aplicación de Android en Eclipse.

Como se puede observar, existen multitud de opciones que el propio entorno de desarrollo nos permite configurar.

Actividad propuesta I.5. *Se recomienda al alumno que explore por sí mismo la pestaña Manifest para familiarizarse con su contenido.*

Actividad propuesta I.6. *Repetir las actividades I.3. y I.4., haciendo uso de la interfaz gráfica de la pestaña Manifest.*

4.3.2. Pestaña **Application**

Por su parte, la pestaña Application describe los componentes de nivel de aplicación contenidos en el paquete especificado, así como atributos generales de la aplicación, tal y como se observa en la figura 4.5.

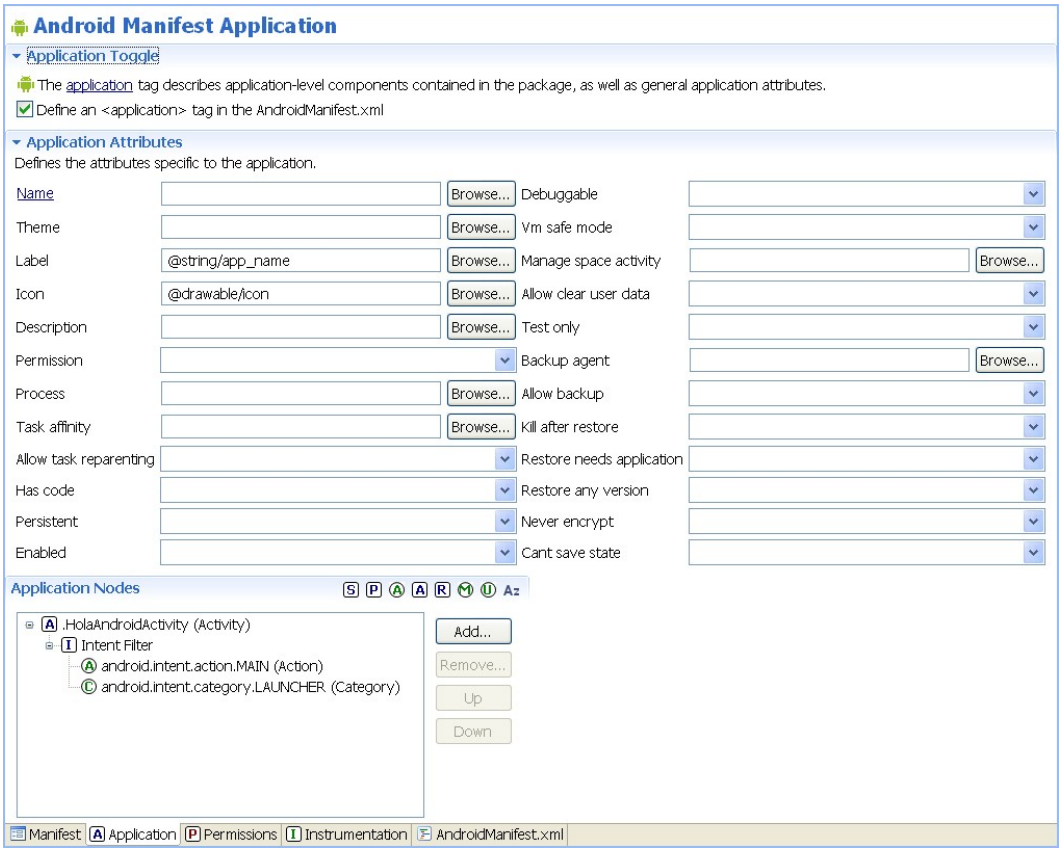

Figura 4.5: Manifiesto: Pestaña Application

Actividad propuesta I.7. *Se recomienda al alumno que explore por sí mismo la pestaña Application para familiarizarse con su contenido.*

4.3.3. Pestaña **Permissions**

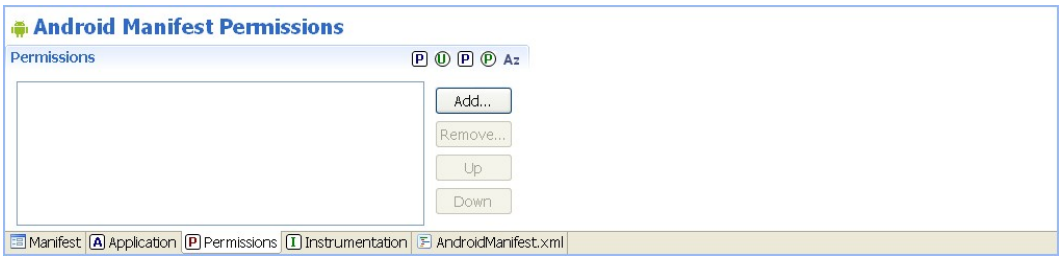

Figura 4.6: Manifiesto: Pestaña Permissions

Actividad propuesta I.8. *Se recomienda al alumno que explore por sí mismo la pestaña Permissions para familiarizarse con su contenido.*

4.3.4. Pestaña Instrumentation

Figura 4.7: Manifiesto: Pestaña Instrumentation

Actividad propuesta I.9. *Se recomienda al alumno que explore por sí mismo la pestaña Instrumentation para familiarizarse con su contenido.*

4.4. Externalizando recursos

Una vez estudiado estudiado el archivo `Manifest.xml`, que será fundamental en todas nuestras aplicaciones Android y que debemos conocer correctamente, vamos a pasar a ver ahora otra parte muy importante de cualquier aplicación: sus recursos.

Un recurso puede ser por ejemplo una imagen, un texto, un icono, una animación, o incluso el *Layout* (la plantilla) de un interfaz gráfico. Android está diseñado de forma que los recursos estén débilmente acoplados al código. Con esto conseguimos que, separando ambas partes, nuestras aplicaciones sean fácilmente actualizables a lo largo del tiempo, y también reutilizables.

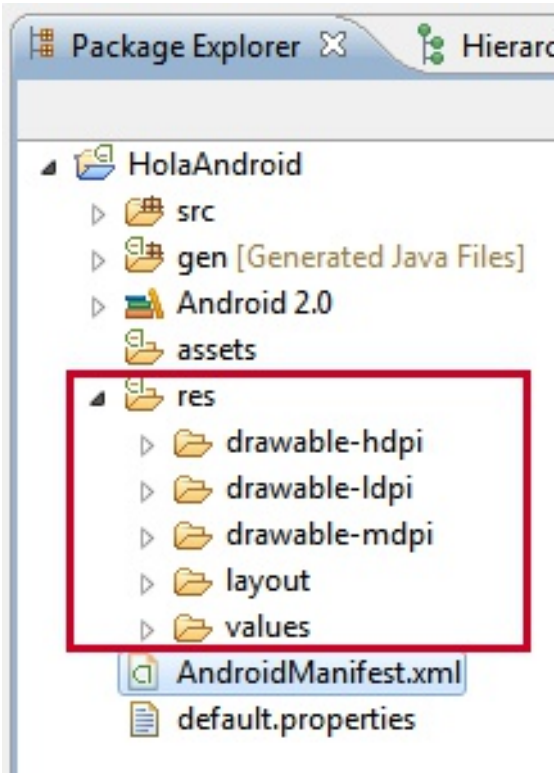

Figura 4.8: Package Explorer. Recursos

Fijémonos en el Package Explorer de la figura 4.8 de nuestro proyecto. Hemos destacado en rojo las carpetas correspondientes a los recursos. Si echamos un vistazo por ellas veremos por ejemplo que las carpetas `drawable` (baja, media y alta densidad) contienen los iconos que se utilizarán en el menú de nuestra aplicación, la carpeta `layout` contiene la plantilla que define la interfaz que vemos en nuestra aplicación y por último la carpeta `values` contiene las variables de texto (y sus valores) que se han utilizado en nuestra aplicación.

Android permite también mediante la externalización de los recursos a través de estas carpetas, la posibilidad de aplicar dinámicamente los recursos de una u otra carpeta en función del hardware que ofrezca el dispositivo, de forma que automáticamente se selecciona el árbol de recursos que contiene la configuración más idónea (por ejemplo para lenguas, países, pantallas o teclados diferentes a los habituales) sin necesidad de estar escribiendo complejas rutinas de código.

Existen 9 tipos distintos de recursos en Android, a saber:

- Valores simples
- Estilos y temas
- Imágenes o "*Drawable*"
- *Layouts*
- Animaciones
- Menús
- "*Searchable*"
- XML
- Materias primas o "*raw resources*".

Cuando se construye la aplicación, los recursos se comprimen de la forma más eficientemente posible y se incluyen dentro del paquete de la aplicación. Este proceso de construcción genera también la clase `R`[2], que contiene una referencia a cada uno de los recursos que tiene nuestro proyecto. En la figura podemos ver un ejemplo de la clase `R` en nuestro proyecto.

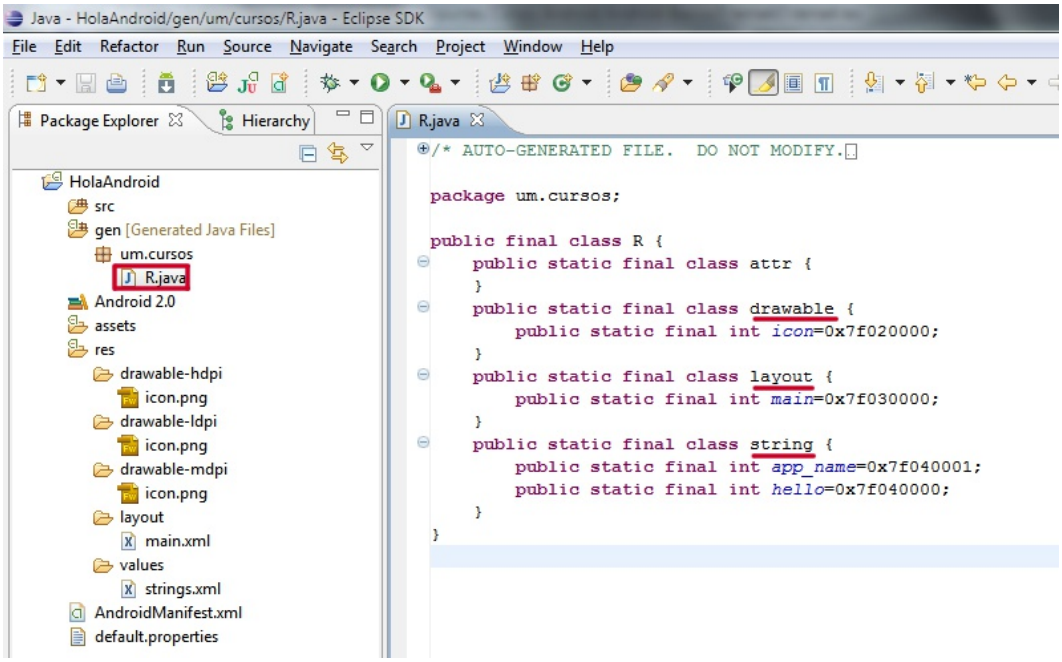

Figura 4.9: Clase R

A continuación describiremos con un poco más de detalle cada uno de los tipos de recursos que podemos encontrar en una aplicación Android.

[2] http://developer.android.com/reference/android/R.html

4.4.1. Valores simples

Un valor simple puede ser de cualquiera de los siguientes tipos: una cadena de texto (`String`), un color, un array de cadenas, una dimensión (p.ej. el ancho o alto de un *layout*), o un entero.

Si abrimos el fichero `strings.xml` dentro de la carpeta `values` del Package Explorer veremos las dos cadenas de texto que están definidas en nuestro proyecto `Hola Android`, tal y como se observa en la figura 4.10.

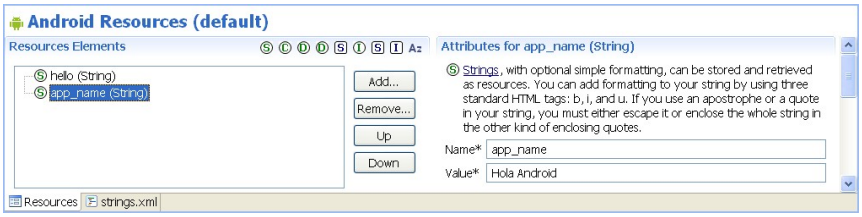

Figura 4.10: Vista del fichero `strings.xml` (I)

Eclipse ofrece dos posibilidades para la definición de nuevos recursos: mediante la pestaña Resources, tal y como se muestra en la figura 4.10, o bien editando directamente el fichero `strings.xml` desde la pestaña strings.xml, como se observa en la figura 4.11.

Figura 4.11: Vista del fichero `strings.xml` (II)

Es una práctica muy interesante la de crear un nuevo archivo por cada conjunto de valores simples que utilicemos, al igual que para el proyecto `Hola Android`, de forma automática se ha creado el fichero `strings.xml` para las cadenas. Para ello en Eclipse accedemos a File→New→Other... y en la ventana que nos aparece, dentro de la carpeta Android, seleccionamos la opción Android XML File. Pulsamos sobre el botón Next y nos aparecerá una pantalla como la de la figura 4.12.

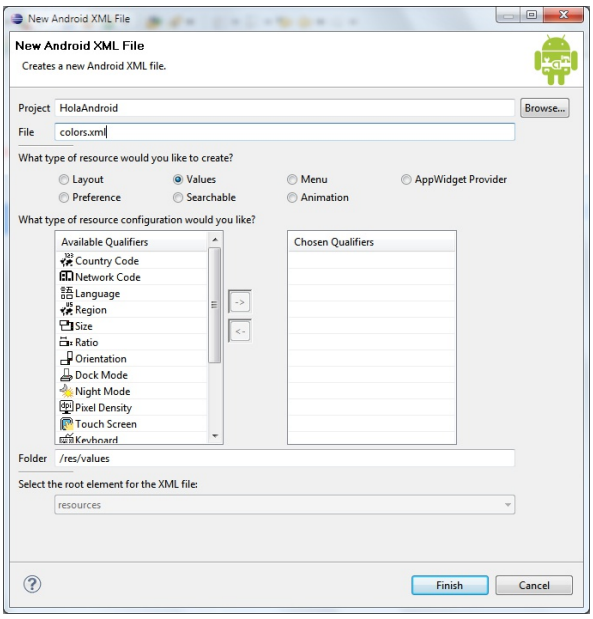

Figura 4.12: Creando el fichero `colors.xml`

Actividad propuesta I.10. *Definir nuevos recursos de tipo* `string` *y de tipo* `color` *y comprobar cómo se ha modificado la clase* `R`*.*

Strings o cadenas

Las cadenas de texto en Android no son sólo cadenas, sino que también permiten su estilización a través de comandos HTML. Así por ejemplo, si queremos que nuestras cadenas aparezcan en negrita, utilizaremos la etiqueta <b>, debiendo tener en cuenta que tendremos que editar manualmente el fichero `strings.xml`. Algunos ejemplos adicionales:

```
<string name="bold_message"><b>Mensaje en negrita</b></string>
<string name="italics_message"><b>Mensaje en cursiva</b></string>
<string name="underline_message"><b>Mensaje subrayado</b></string>
```

Actividad propuesta I.11. *Modificar la cadena* `hello` *de la aplicación* `HolaAndroid` *para que contenga palabras en negrita y en cursiva. Ejecutar la aplicación para ver el efecto.*

Colores

Los colores nos sirven para modificar el color de un recurso (por ejemplo una cadena). La notación que se utiliza para la modificación de los colores es la de HTML. Por ejemplo podemos definir el color azul oscuro mediante esta notación:

```
<color name="azul_oscuro">#00F</color>
```

Otro aspecto interesante del valor simple `color` es que permite la introducción de transparencia en el color. Para ello, especificaremos el porcentaje de transparencia delante del código del color en HTML. En el siguiente ejemplo, `77` hace referencia al porcentaje de transparencia o canal alfa y el resto (`00FF00`) al color verde:

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

Finalmente, a continuación se exponen las distintas posibilidades que existen a la hora de especificar el color:

- RGB (Ejemplo: `039`)
- RRGGBB (Ejemplo: `003399`)
- ARGB (Ejemplo: `8039`)
- AARRGGBB (Ejemplo: `80003399`)

Dimensiones

Las longitudes o dimensiones son útiles sobre todo a la hora de determinar *layouts* y estilos. Se pueden expresar en diferentes medidas tal y como se puede ver a continuación (por lo general siempre se utilizan pixels o milímetros:

- `px` (Pixels)
- `in` (Pulgadas)
- `pt` (Puntos)
- `mm` (Milímetros)
- `dp` (Densidad de píxels)
- `sp` (Escalado de píxels)

Un par de ejemplos de uso de dimensiones podrían ser:

```
<dimen name="borde_estandar">1px</dimen>
<dimen name="tamanyo_fuente">12pt</dimen>
```

4.4.2. Estilos y temas

Los estilos permiten especificar un conjunto de valores que den una apariencia concreta a las vistas. Lo más frecuente es utilizar los estilos y temas para almacenar los colores y fuentes de la aplicación. A continuación se presenta un ejemplo de definición de un par de estilos:

```
<style name="TextoNormal">
    <item name="android:textSize">12px</item>
    <item name="android:textColor">#111</item>
</style>
<style name="TextoPequenyo" parent="TextoNormal">
    <item name="android:textSize">8px</item>
</style>
```

4.4.3. Imágenes

Otro recurso interesante que Android nos permite definir son las imágenes. Los formatos soportados para las mismas son `JPEG`, `PNG` y mapas de bits, siendo `PNG` el formato preferido.

```
<drawable name="icon">icon.png</drawable>
```

4.4.4. Layouts

Los layouts permiten, como ya hemos comentado anteriormente, especificar la interfaz que tendrá nuestra aplicación sin mezclar el código de la misma con la parte interna del programa. Trabajaremos con ellas en profundidad en el próximo capítulo. Para el caso particular de nuestro proyecto `HolaAndroid` el layout que se genera es el siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

4.4.5. Animaciones

Android soporta dos tipos de animaciones: por un lado dispone de las "*Tweened animations*" o animaciones interpoladas que sirven para rotar, mover, reducir o ampliar una vista, mientras que por otro lado dispone de las animaciones habituales creadas mediante una secuencia de imágenes. Definir las animaciones como un recurso externo nos permite su reutilización en muchas partes del código y además nos da la oportunidad de presentar diferentes animaciones en función del dispositivo hardware donde se ejecuten.

4.4.6. Menús

Los menús también son otro tipo de recurso de gran valor que estudiaremos con detenimiento en el tema 6.

Tema 5

Interfaces de Usuario I. Fundamentos

5.1. Introducción

En una aplicación de Android, las interfaces de usuario se construyen utilizando objetos de las clases `View`[1] (vista) y `ViewGroup`[2] (grupo de vistas). Existen multitud de vistas y grupos de vistas, todos ellos heredando de la clase genérica `View`.

Los objetos de la clase `View` son las unidades básicas para expresar interfaces de usuario en la plataforma Android. Tal y como se observa en la figura 5.1, la clase `View` sirve como base para otras subclases llamadas "*widgets*", que ofrecen objetos de interfaz de usuario completamente desarrollados, como campos de texto o botones, por ejemplo. La clase `ViewGroup`, por su parte, sirve como base para las subclases llamadas "*layouts*", que ofrecen distintos tipos de distribución de los *widgets* en la interfaz de usuario, como lineal, tabular o relativa, entre otros.

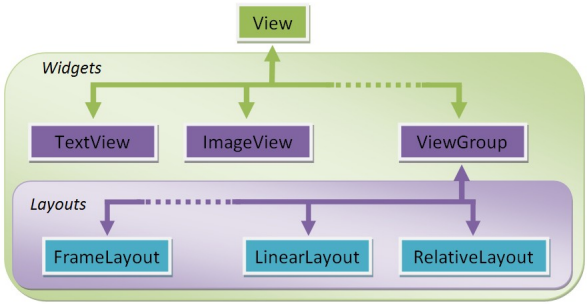

Figura 5.1: Jerarquía de la clase `View`: *Widgets* y *Layouts*

Un objeto de la clase `View` es una estructura de datos cuyas propiedades guardan la disposición y contenido para un área rectangular específica de la pantalla. Un objeto `View` maneja su propia medida y disposición, dibujado, cambio de foco, scroll y pulsaciones de teclas para el área de pantalla a la que representa.

Por otra parte, un grupo de vistas, representado por un objeto de la clase `ViewGroup`, es un tipo especial de vista cuya función es contener y organizar un subconjunto de vistas y otros grupos de vistas, tal y como se puede comprobar en la figura 5.2. Los grupos de vistas nos permiten añadir estructura a la interfaz de usuario y construir elementos de pantalla complejos que puedan ser referenciados como una sola entidad. Los grupos de vistas son al fin y al cabo contenedores de otras vistas.

[1] http://developer.android.com/reference/android/view/View.html
[2] http://developer.android.com/reference/android/view/ViewGroup.html

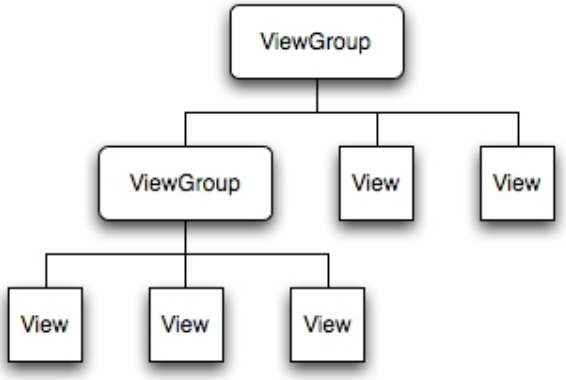

Figura 5.2: Grupo de vistas como contenedor de otras vistas

Este árbol puede ser tan simple o complejo como necesitemos, y podemos construirlo utilizando el conjunto de *widgets* y *layouts* predefinido de Android o creando los nuestros propios. Y para hacer dicho árbol visible en nuestra pantalla, el objeto de la clase `Activity` correspondiente debe llamar al método `setContentView()`, pasándole como parámetro el objeto del nodo raíz del árbol.

5.1.1. *Widgets*

Como ya hemos dicho, un *widget* es un objeto de la clase `View` que sirve como interfaz para la interacción con el usuario. Android proporciona un conjunto de *widgets* completamente desarrollados como por ejemplo botones, check boxes o campos de texto, de modo que es posible construir fácil y rápidamente una interfaz de usuario.

Otros *widgets* proporcionados por Android son algo más complejos, como un seleccionador de fechas, un reloj, o controles de zoom. Incluso es posible definir nuestros propios *widgets* complejos, pero eso queda fuera de los contenidos de este curso de iniciación a Android.

Por contra, en este tema estudiaremos con un poco más de detalle las siguientes vistas:

- `TextView`: Representa una etiqueta de sólo lectura, permitiendo el formateado del texto que contiene.
- `EditText`: Representa una entrada de texto. Permite múltiples líneas.
- `Button`: Representa un botón al cual, como veremos más adelante, nos permitirá asociarle acciones al ser pulsado.
- `ImageButton`: Representa un botón con una imagen asociada (a diferencia de `Button`, que sólo tiene texto).
- `CheckBox`: Es un botón de dos estados, que puede estar marcado o no.
- `RadioButton` Similar a un `CheckBox`, con la diferencia de que sólo un `RadioButton` de todos los pertenecientes a un mismo `RadioGroup` puede estar marcado en cada instante.
- `ImageView`: Permite mostrar imágenes.
- `ProgressBar`: Representa una barra de progreso.
- `Spinner`: Un control compuesto que muestra un `TextView` y, asociado al mismo, muestra también una `ListView` que permite seleccionar uno de los elementos que se muestra en el cuadro de texto.

Y por otra parte, también analizaremos los siguientes *layouts*:

- `FrameLayout`: Es el tipo de *layout* más simple que existe y sólo puede contener un elemento.
- `LinearLayout`: Este *layout* alinea todos los elementos que contiene vertical u horizontalmente, según le indiquemos.
- `ListView`: Una vista de grupo que contiene una lista vertical de elementos.
- `TableLayout`: Este *layout* permite colocar los elementos que contiene en filas y columnas.

5.1.2. Capturando eventos

Una vez que hayamos añadido algunos *widgets* (como veremos más adelante) a nuestra interfaz de usuario, es posible recibir notificaciones cuando el usuario interactúa con dichos *widgtes*.

Para ello, deberemos definir un *event listener*[3], y registrarlo en la vista (*widget*) sobre la que queramos recibir notificaciones.

La clase `View` contiene un conjunto de interfaces anidadas llamadas `On<accion>Listener`. Por ejemplo, `View.OnClickListener` (para manejar eventos de pulsación sobre una vista), `View.OnTouchListener` (para manejar eventos de tocar la pantalla) y `View.OnKeyListener` (para manejar eventos de pulsación de teclas o botones del dispositivo).

La clase `View` también dispone de los correspondientes métodos `setOn<accion>Listener()` para definir los *listeners* para cada tipo de acción.

5.1.3. Método `findViewById()`

Todos los objetos de la clase `View` contienen un atributo `id` de tipo entero que sirve como identificador de los mismos. Estos identificadores se suelen asignar normalmente desde el fichero `main.xml`, mediante `android:id="@+id/my_id"`, y se utilizan para encontrar vistas específicas dentro de un árbol de vistas (como el de la figura 5.2).

Y para ello se utiliza el siguiente método de la clase `Activity`:

```
public View findViewById (int id)
```

Así por ejemplo, si queremos recuperar y utilizar el objeto de la clase `TextView` que incorpora por defecto nuestra aplicación `HolaAndroid`, lo primero que tendremos que hacer será asignarle un identificador.

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

Para todas y cada una de las vistas que tengan asignado un identificador, se creará de manera automática en la clase `R` un atributo de tipo entero dentro de la clase anidada `id` con ese mismo nombre, tal y como podemos observar a continuación.

[3] http://developer.android.com/guide/topics/ui/ui-events.html

```
public final class R {
    ...
    public static final class id {
        public static final int helloTextView=0x7f050000;
    }
    ...
}
```

De esta manera, podremos recuperar de forma sencilla cualquiera de las vistas definidas en el fichero `main.xml`, desde nuestra clase `Activity` haciendo uso del método `findViewById()`, tal y como se muestra a continuación.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    TextView helloTextView = (TextView) findViewById(R.id.helloTextView);
}
```

Actividad propuesta I.12. *Asignarle un identificador al `TextView` que viene por defecto en la aplicación `HolaAndroid`, y recuperar el objeto correspondiente desde la clase `Activity`, tal y como se ha explicado.*

5.1.4. Pestaña GraphicalLayout

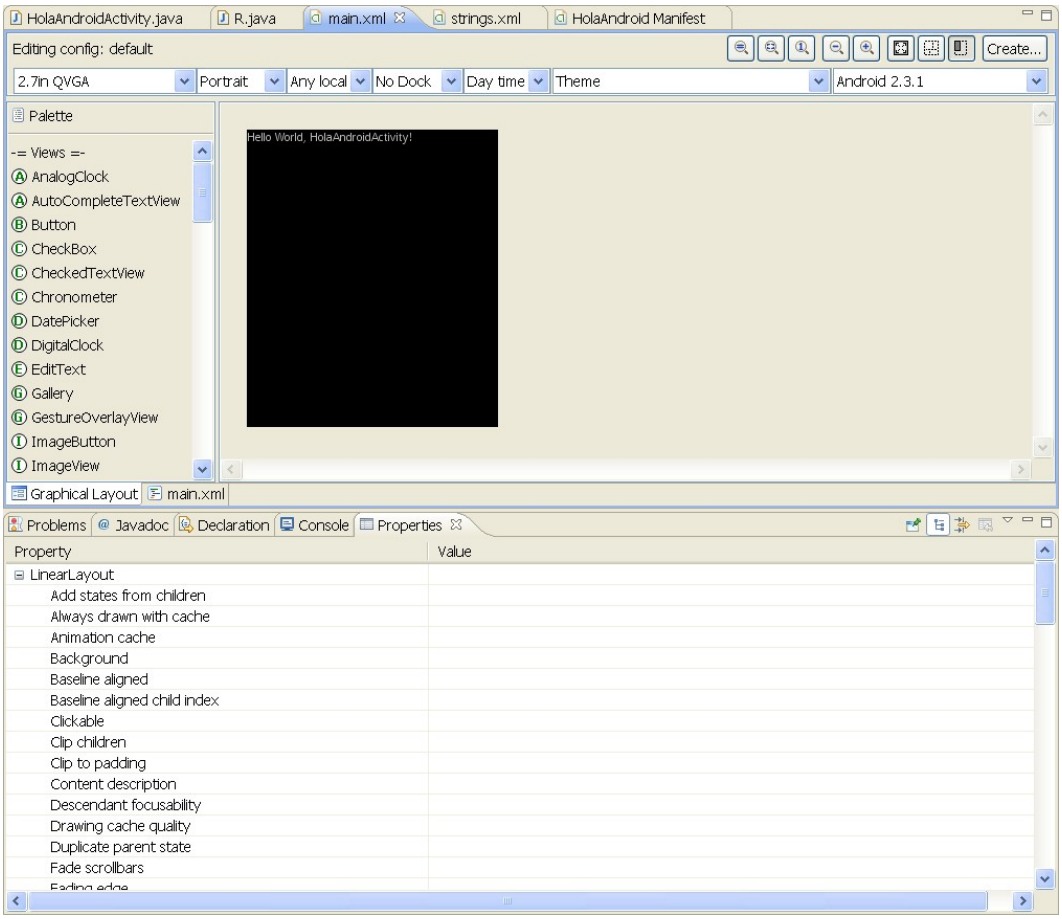

Figura 5.3: Pestaña GraphicalLayout

Una de las ventanas más útiles de Eclipse a la hora de desarrollar interfaces gráficas de usuario para Android es sin duda la que se muestra en la figura 5.3. Corresponde a la pestaña GraphicalLayout que podemos seleccionar cuando visualizamos el fichero `main.xml`.

Como podemos observar, este panel se compone, a su vez, de varios paneles. En primer lugar, en la parte superior de la pantalla, tenemos la Editing configuration, que nos permite especificar propiedades como el tamaño u orientación de la pantalla, el tema o la versión de Android a usar, entre otros.

Por otro lado, contamos con el interesante panel Palette, donde encontraremos todas las vistas y *layouts* disponibles por defecto para Android. Añadir una nueva vista o *layout* a nuestra pantalla es tan fácil como arrastrar y soltar ("*drag & drop*") dicho *widget*.

Por último en la parte inferior, tenemos la pestaña de Properties que, como su propio nombre indica, nos permite explorar y modificar las propiedades del *widget* que en cada momento tengamos seleccionado. Si no nos aparece esta pestaña de Properties, deberemos pulsar en Window→Show View→Other..., y en la ventana que nos aparece, seleccionar dentro de la carpeta General, la opción Properties.

Actividad propuesta I.13. *Se recomienda al alumno que explore por sí mismo el contenido de esta pestaña GraphicalLayout, para familiarizarse con ella.*

5.2. Vistas

En esta sección explicaremos cada una de las vistas elementales que ofrece Android, así como algunas de sus funcionalidades básicas.

5.2.1. `TextView`

La clase `TextView`[4] nos servirá para representar etiquetas de texto sencillas.

Para mostrar un ejemplo de su funcionamiento, definiremos un nuevo recurso de tipo `String` dentro del fichero `strings.xml` con nombre `goodbye` y valor `Goodbye, see you soon!`, tal y como se observa en la figura 5.4.

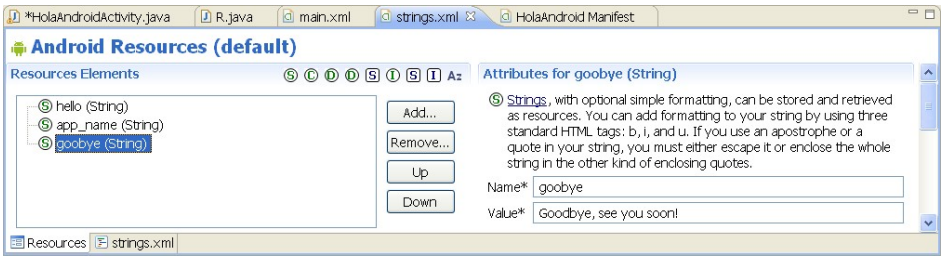

Figura 5.4: Definiendo un nuevo recurso de tipo `String`

A continuación añadiremos un nuevo `TextView` a nuestra aplicación `HolaAndroid`, con nombre `goodbyeTextView`, y le asignaremos la cadena `goodbye`, como se puede observar en la figura 5.5.

[4] http://developer.android.com/reference/android/widget/TextView.html

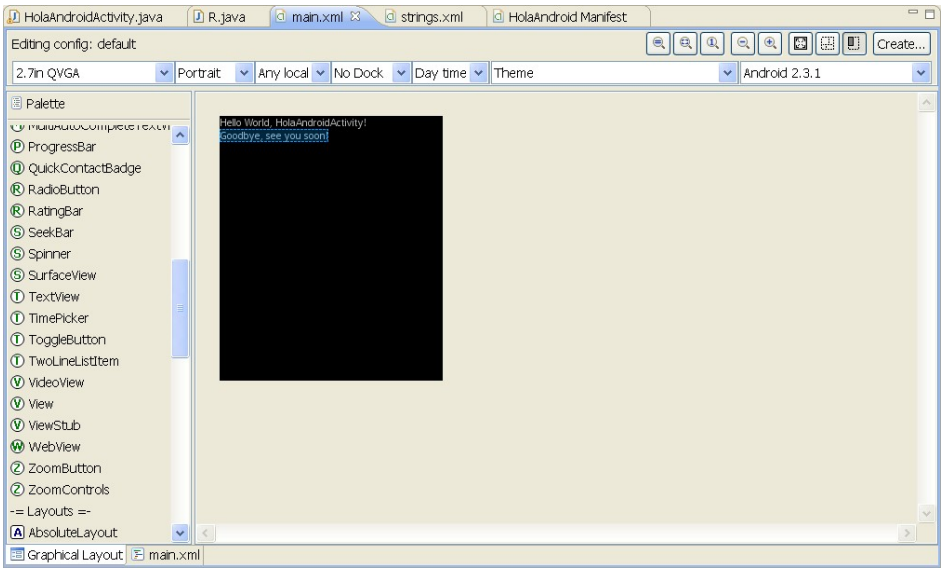

Figura 5.5: Añadiendo un nuevo `TextView`

Por último, vamos a comprobar cómo recuperar y cómo modificar el texto de un `TextView`. Para ello, vamos a desarrollar una pequeña aplicación de ejemplo que al inicio de la misma cambie automáticamente el texto asociado al `TexView goodbyeTextView` y lo convierta todo en mayúsculas.

Para conseguir este propósito, deberemos añadir el siguiente código en la clase `Activity`.

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

    TextView goodbyeTextView = (TextView) findViewById(R.id.goodbyeTextView);
    String goodbyeTextViewString = goodbyeTextView.getText().toString();
    goodbyeTextViewString = goodbyeTextViewString.toUpperCase();
    goodbyeTextView.setText(goodbyeTextViewString);
}
```

El resultado de ejecutar la aplicación se muestra en la figura 5.6.

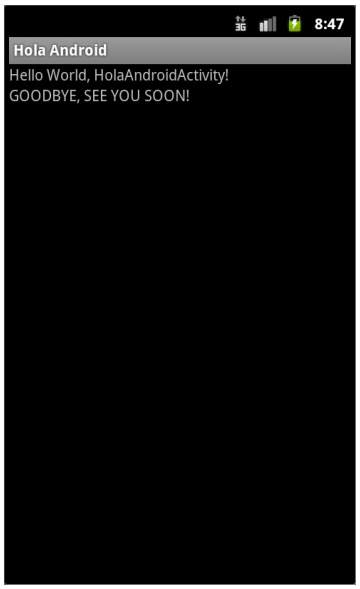

Figura 5.6: Ejemplo de uso de un `TextView`

5.2.2. `EditText`

La clase `EditText`[5] nos permitirá representar un campo de texto para que el usuario pueda introducir cualquier tipo de cadena.

Para mostrar su funcionamiento vamos a crear una nueva aplicación llamada `ConversorDeDivisas`, que contendrá, tal y como se puede observar en la figura 5.7, dos `TextView` y dos `EditText` con nombres `euroEditText` y `dolarEditText`, respectivamente.

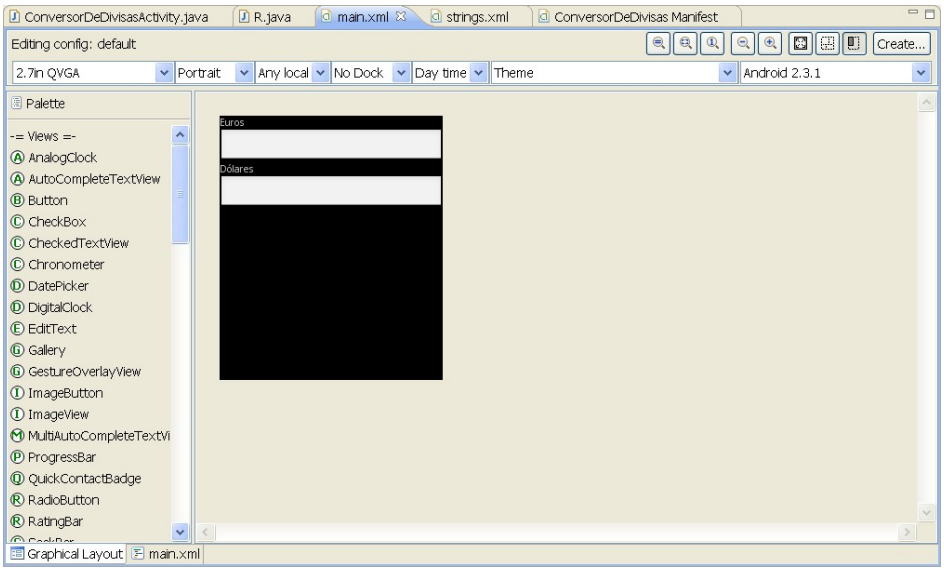

Figura 5.7: Conversor de divisas (GraphicalLayout)

A continuación vamos a darle funcionalidad a nuestro conversor de divisas mediante un par de botones.

5.2.3. `Button` e `ImageButton`

La clase `Button`[6] de Android nos permite representar botones que pueden ser pulsados por el usuario para llevar a cabo alguna acción.

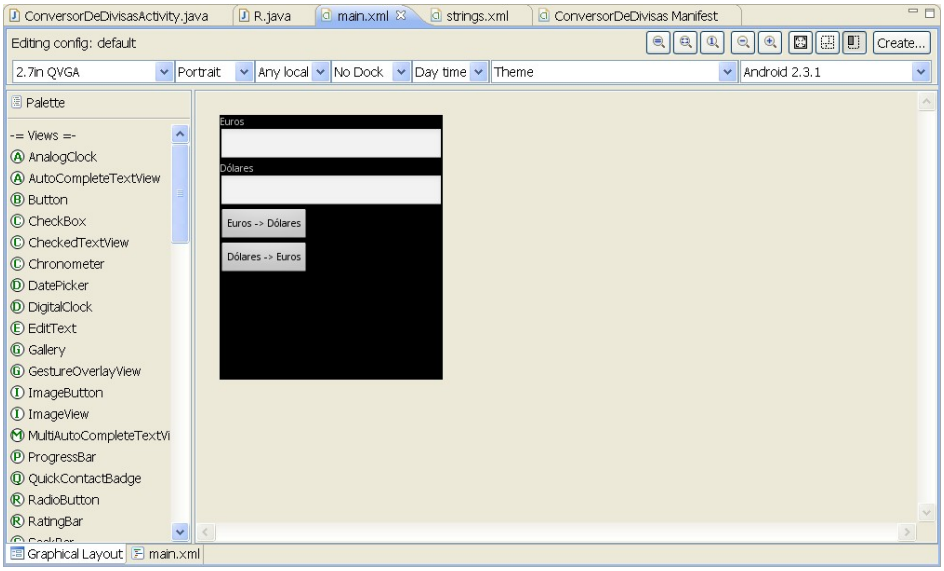

Figura 5.8: Añadiendo nuevos `Button`s

[5] http://developer.android.com/reference/android/widget/EditText.html
[6] http://developer.android.com/reference/android/widget/Button.html

Por lo tanto, completaremos nuestra aplicación `ConversorDeDivisas` mediante dos botones llamados `euro2dolarButton` y `dolar2euroButton`, respectivamente, tal y como se puede comprobar en la figura 5.8.

Como ya explicamos anteriormente, las vistas permiten capturar una serie de eventos. En nuestro caso, vamos a capturar el evento de pulsar sobre el botón. Para ello, debemos establecer y definir el *event listener* correspondiente para cada botón, especificando la acción que se llevará a cabo cuando dicho evento ocurra. Así por ejemplo, para el caso del botón `euro2dolarButton`, tendríamos el siguiente código.

```
Button euro2dolarButton = (Button) findViewById(R.id.euro2dolarButton);
euro2dolarButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        euro2Dolar();
    }
});
```

Siendo el método `euro2Dolar()` el encargado de recoger el valor del `EditText euroEditText`, convertirlo a un valor real, calcular el valor correspondiente de esa cantidad en dólares, y establecerlo como texto en el `EditText dolarEditText`. Su contenido podría ser algo como lo siguiente (el método `dolar2Euro()` es análogo):

```
public void euro2Dolar() {
    try {
        EditText euroEditText = (EditText) findViewById(R.id.euroEditText);
        String euroEditTextString = euroEditText.getText().toString();
        double euros = Double.parseDouble(euroEditTextString);
        double dolares = euros*1.36;
        String dolarEditTextString = String.valueOf(dolares);
        EditText dolarEditText = (EditText) findViewById(R.id.dolarEditText);
        dolarEditText.setText(dolarEditTextString);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Actividad propuesta I.14. *Desarrolla la aplicación `ConversorDeDivisas`, tal y como se ha explicado anteriormente.*

Por su parte, un `ImageButton`[7] permite representar un botón que en lugar de tener un texto asociado, tendrá una imagen.

Para mostrar su funcionamiento, vamos a añadir un `ImageButton` a nuestra aplicación `ConversorDeDivisas` que nos permita salir de la misma. Por lo tanto, lo primero será añadir un `ImageButton` con nombre `exitImageButton` desde la pestaña GraphicalLayout, tal y como se observa en la figura 5.9.

A continuación vamos a asignarle una imagen al `ImageButton` y vamos a establecer su color de fondo como transparente. En primer lugar, copiamos el fichero de nuestra imagen (en nuestro caso llamado `exit.png`) dentro de las tres carpetas `drawdable-hdpi`, `drawdable-ldpi` y `drawdable-mdpi`, contenidas dentro de la carpeta `res` de nuestro proyecto de Eclipse. Tras hacer esto, podemos comprobar que la clase `R` habrá incluido automáticamente la imagen como atributo de la clase anidada `drawdable`.

[7] http://developer.android.com/reference/android/widget/ImageButton.html

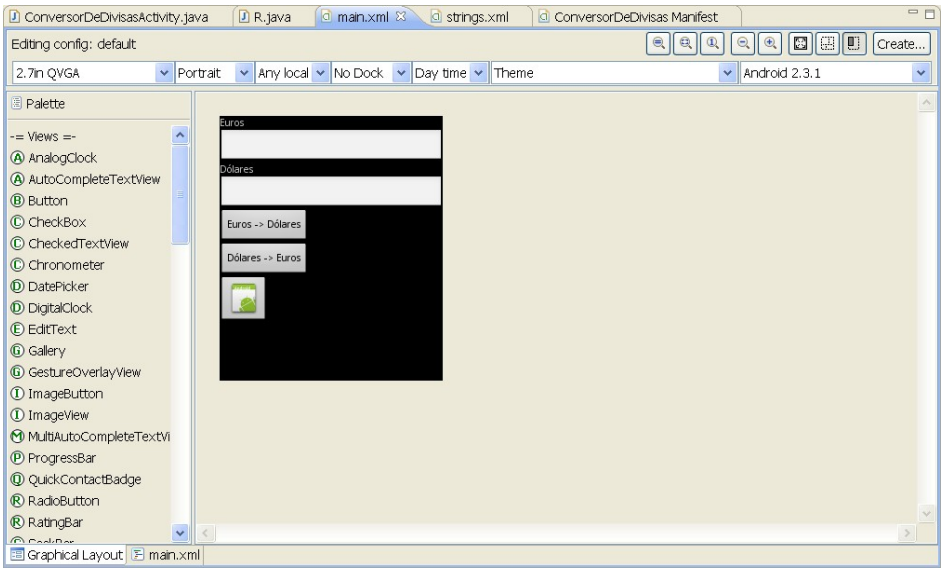

Figura 5.9: Añadiendo un nuevo `ImageButton` (I)

A continuación deberemos seleccionar el `ImageButton` y, dentro del panel Properties indicarle que el atributo Src tendrá como valor asociado el recurso `@drawdable/exit`. Por otra parte, para conseguir que el fondo del botón sea de color transparente, deberemos asignarle, dentro del panel Properties, al atributo Background, el valor `@android:color/transparent`. El resultado que obtenemos es el que se muestra en la figura 5.10.

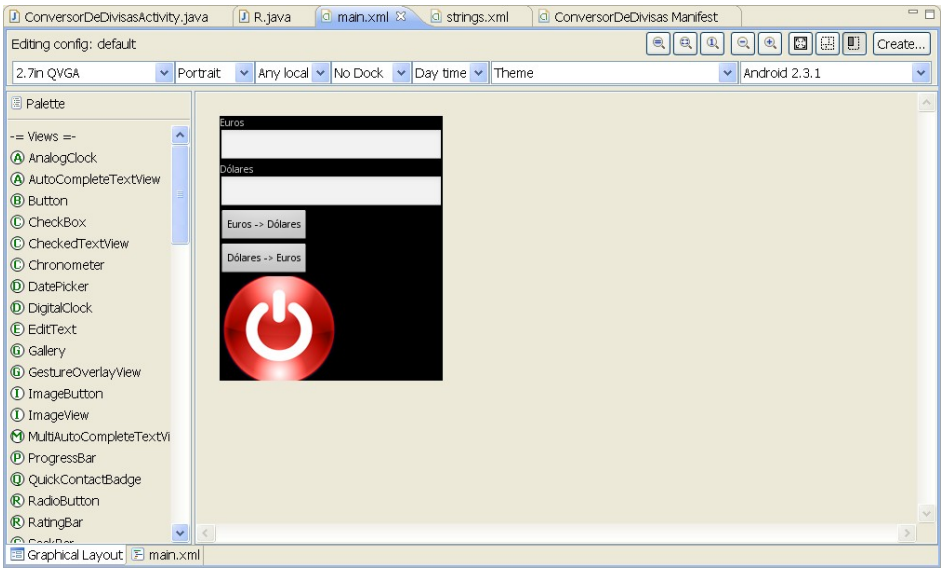

Figura 5.10: Añadiendo un nuevo `ImageButton` (II)

Por último, para asignarle la acción de finalizar la aplicación al pulsar sobre el botón, dentro de nuestra clase `Activity`, deberemos añadir el siguiente código:

```
public void onCreate(Bundle savedInstanceState) {
    ...
    ImageButton exitImageButton =
                        (ImageButton) findViewById(R.id.exitImageButton);
    exitImageButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            setResult(RESULT_OK);
            finish();
        }
    });
}
```

Actividad propuesta I.15. *Añadir a la aplicación `ConversorDeDivisas` el `ImageButton` `exitImageButton`, tal y como se ha explicado anteriormente.*

5.2.4. `CheckBox` y `RadioButton`

La clase `CheckBox`[8] nos permite representar un botón de dos estados que puede estar marcado o no marcado.

Por otro lado, la clase `RadioButton`[9] también nos permite representar un botón de dos estados que puede estar marcado o no marcado. Sin embargo, a diferencia de la clase `CheckBox`, un `RadioButton` no puede desmarcarse una vez que el usuario lo ha marcado.

Finalmente, la clase `RadioGroup`[10] nos permite agrupar varios `RadioButton`. Una particularidad de esta clase es que cuando varios `RadioButton` aparecen juntos dentro de un mismo `RadioGroup`, solamente uno de ellos podrá estar marcado en cada instante. Esto nos permite configurar, por ejemplo, encuestas de una única respuesta.

Para probar estas tres vistas vamos a crear otra sencilla aplicación llamada `RadioYCheck`, que muestre las mismas. Los objetivos de esta aplicación serán:

1. Para la vista `CheckBox` queremos que cuando se pulse en ella aparezca el mensaje "`Marcado`" o bien "`Desmarcado`", respectivamente.

2. Para la vista `RadioButton` queremos que cuando se pulse en ella aparezca el mensaje "`Ya no puede desmarcarse`".

3. Para el grupo de vistas `RadioGroup` queremos que nos señale la opción elegida en cada caso.

Por lo tanto, en primer lugar creamos una aplicación nueva llamada `RadioYCheck`, y desde la pestaña GraphicalLayout del fichero `main.xml`, cronstruimos una pantalla como la que se muestra en la figura 5.11, con varios elementos de tipo `TextView`, `CheckBox`, `RadioButton` y `RadioGroup`, usando en todos ellos los correspondientes recursos definidos en `strings.xml` para sus etiquetas.

[8] http://developer.android.com/reference/android/widget/CheckBox.html
[9] http://developer.android.com/reference/android/widget/RadioButton.html
[10] http://developer.android.com/reference/android/widget/RadioGroup.html

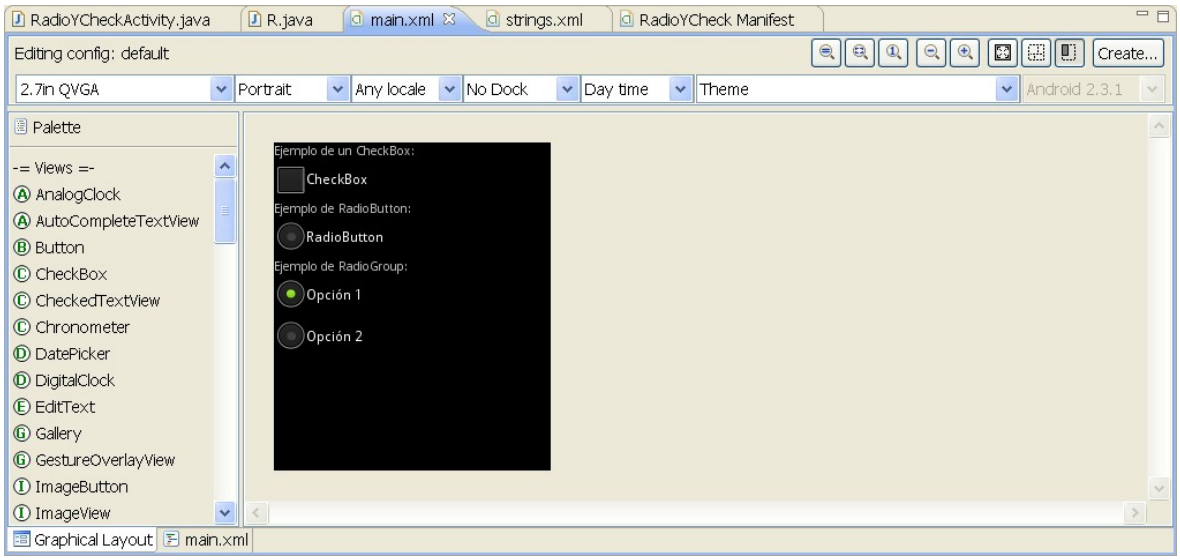

Figura 5.11: Nueva aplicación para `CheckBox`, `RadioButton` y `RadioGroup` (I)

Para el **primer objetivo**, necesitamos programar el evento *OnClick* del `Checkbox checkBox` como se puede comprobar a continuación:

```
// Ejemplo de la vista checkBox
CheckBox checkBox = (CheckBox) findViewById(R.id.checkBox);
checkBox.setOnClickListener(new View.OnClickListener() {
     public void onClick(View arg0) {
         TextView checkBoxTextView = (TextView) findViewById(R.id.checkBoxTextView);
         CheckBox checkBox = (CheckBox) arg0;
         if(checkBox.isChecked())
            checkBoxTextView.setText(getString(R.string.checkBoxLabel)+" Marcado");
         else
            checkBoxTextView.setText(getString(R.string.checkBoxLabel)+" Desmarcado");
     }
});
```

Para el **segundo objetivo**, por su parte, necesitamos programar de nuevo el evento *OnClick* del `RadioButton radioButton` de la siguiente manera:

```
// Ejemplo de la vista radioButton
RadioButton radioButton = (RadioButton) findViewById(R.id.radioButton);
radioButton.setOnClickListener(new View.OnClickListener() {
     public void onClick(View arg0) {
         TextView radioButtonTextView =
                        (TextView) findViewById(R.id.radioButtonTextView);
         radioButtonTextView.setText(
                         getString(R.string.radioButtonLabel)+" Marcado");
     }
});
```

Para el **tercer objetivo**, necesitamos programar el evento *OnCheckedChanged* del `RadioGroup` `radioGroup` tal y como podemos comprobar a continuación:

```
// Ejemplo del grupo de vistas RadioGroup;
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    public void onCheckedChanged(RadioGroup arg0, int arg1) {
         RadioButton radioButton =
                     (RadioButton) findViewById(arg0.getCheckedRadioButtonId());
         TextView radioGroupTextView =
                              (TextView) findViewById(R.id.radioGroupTextView);
         radioGroupTextView.setText(
                     getString(R.string.radioGroupLabel)+radioButton.getText());
    }
});
```

Figura 5.12: Nueva aplicación para `CheckBox`, `RadioButton` y `RadioGroup` (II)

5.2.5. `ImageView`

La clase `ImageView`[11] permite mostrar una imagen desde varias fuentes posibles: proveedores de contenido o recursos. Vamos a ver una aplicación muy sencilla a la que llamaremos `TestImagenes`, que muestre dos botones. El primero nos servirá para aplicarle a una imagen dada un valor de transparencia *alpha* igual a 50, mientras que un segundo botón restablecerá en lo posible ese valor al original. Es importante que tengamos en cuenta que una vez que aplicamos un valor de transparencia *alpha*, Android no es capaz de recuperar al 100 % el valor original.

Para asignarle una imagen concreta a nuestro `ImageView` en primer lugar copiaremos el fichero de la imagen dentro de las carpetas `drawdable-hdpi`, `drawdable-mdpi` y `drawdable-ldpi`, dentro de la carpeta `res`. A continuación, desde la pestaña GraphicalLayout, seleccionamos el `ImageView` y dentro del panel de Properties le asignamos al atributo Src el recurso asociado con la imagen que hemos copiado, tal y como se observa en la figura 5.13.

[11] http://developer.android.com/reference/android/widget/ImageView.html

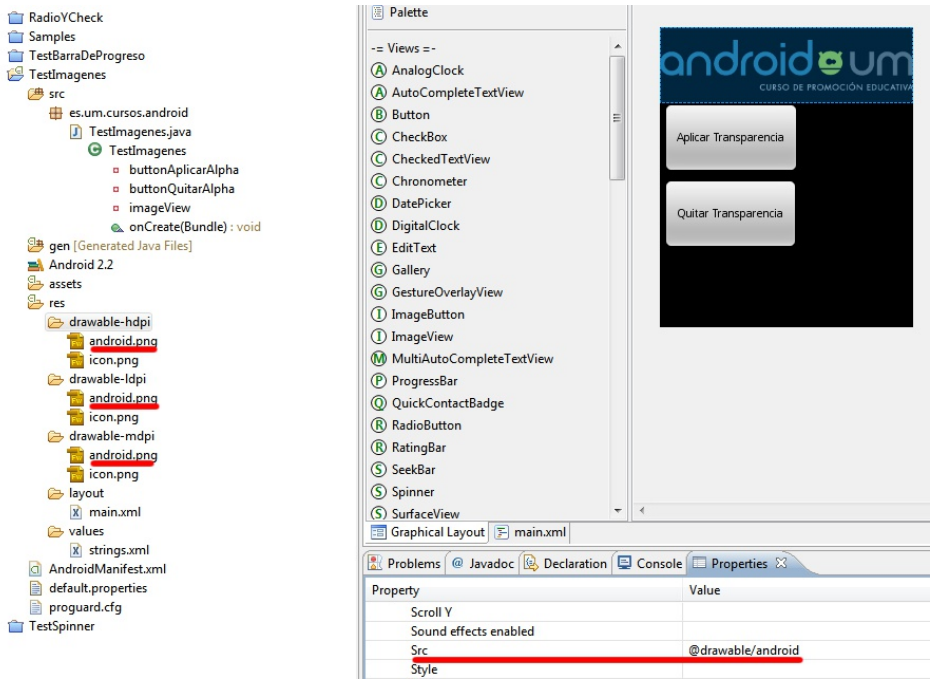

Figura 5.13: Especificando la ubicación de la imagen para un `ImageView`

Una vez hecho esto, necesitamos dotar del comportamiento anteriormente descrito a nuestros botones. Para ello, haremos uso del siguiente código:

```
ImageView imageView = (ImageView) findViewById(R.id.imageView1);
Button buttonAplicarAlpha = (Button) findViewById(R.id.button1);
Button buttonQuitarAlpha = (Button) findViewById(R.id.button2);

buttonAplicarAlpha.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View arg0) {
          imageView.setAlpha(50);
     }
});

buttonQuitarAlpha.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View arg0) {
          imageView.setAlpha(100);
     }
});
```

El resultado final de la aplicación se puede ver en la figura 5.14.

Actividad propuesta I.16. *Desarrollar la aplicación* `TestImagenes` *tal y como se ha descrito anteriormente.*

Figura 5.14: Nueva aplicación para `ImageView` llamada `TestImagenes`

5.2.6. `ProgressBar`

La clase `ProgressBar`[12] nos permite representar un indicador visual de progreso de cualquier operación.

El aspecto por defecto de un `ProgressBar` es el de una imagen circular en movimiento rotatorio. Si introducimos en nuestro *layout* principal dicho elemento, veremos que el mismo ya presenta el resultado que buscamos obtener tal y como podemos ver en la imagen 5.15.

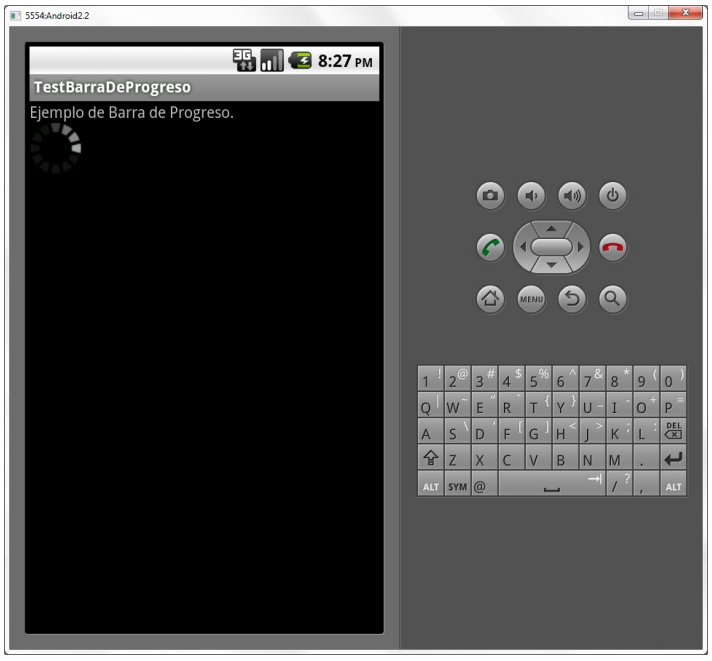

Figura 5.15: Nueva aplicación para `ProgressBar`

[12] http://developer.android.com/reference/android/widget/ProgressBar.html

No obstante, si queremos disponer de una barra de progreso más estilizada y donde se observe el estado del progreso, debemos hacer uso de hilos y manejadores junto con un estilo de barra distinto. El siguiente código muestra una aplicación de ejemplo llamada `TestBarraDeProgreso` que nos permite hacer precisamente esto.

```
public class TestBarraDeProgreso extends Activity {

    ProgressBar myProgressBar;  // Barra de progreso que utilizaremos.
    int myProgress = 0; // Medidor de progreso.

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

        myProgressBar=(ProgressBar)findViewById(R.id.progressbar_Horizontal);

        new Thread(myThread).start(); //Creamos un hilo para dar comportamiento.
    }

    private Runnable myThread = new Runnable(){

    @Override
    public void run() {
        while (myProgress<100){
           try{
               myHandle.sendMessage(myHandle.obtainMessage());
               Thread.sleep(1000);
           }
           catch(Throwable t){
           }
        }
    }

    Handler myHandle = new Handler(){

    /**
      * Hacemos uso de un manejador para trabajar con el hilo creado.
      */
      @Override
      public void handleMessage(Message msg) {
         myProgress++;
         myProgressBar.setProgress(myProgress);
      }
    };
  };
}
```

En el código anterior se observa cómo se crea una barra de progreso, que es controlada a través de un hilo que se va durmiendo cada segundo.

En Eclipse, vemos que a nivel de *layout*, debemos especificar dos valores concretos para las propiedades Max y Style de nuestro `ProgressBar`. Para el valor Max debemos especificar que queremos que como máximo la barra de progreso tenga un valor de 100. Mientras que para el valor Style debemos especificar que tenga el valor `?android: attr/progressBarStyleHorizontal` que nos sirve para que Android sepa que en lugar de la imagen circular, queremos que aplique el estilo de barra compacta.

El resultado que se consigue practicando estos cambios se puede ver en la figura 5.16

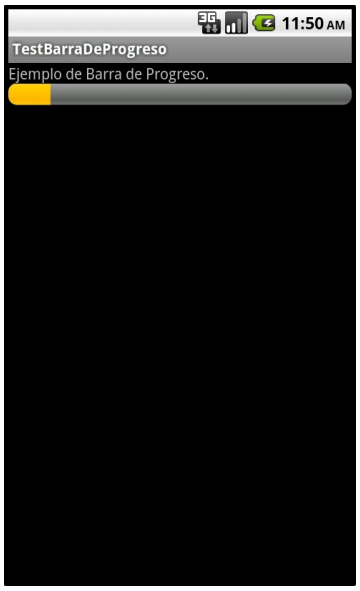

Figura 5.16: Barra de progreso estilizada

5.2.7. `Spinner`

La clase `Spinner`[13] define una lista desplegable de elementos. Muestra siempre uno de los elementos, y permite al usuario elegir entre los demás. Con esta clase podemos lograr mostrar listas desplegables como la que se muestra en la figura 5.17.

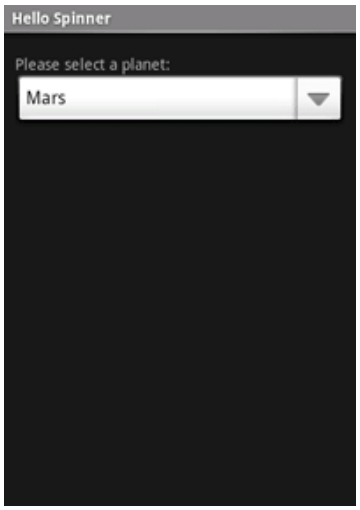

Figura 5.17: Nueva aplicación para probar el `Spinner`

[13] http://developer.android.com/reference/android/widget/Spinner.html

A continuación se muestra el código que necesitamos para trabajar con esta vista:

```
public class TestSpinner extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
                this, R.array.planets_array, android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spinner.setAdapter(adapter);
        spinner.setOnItemSelectedListener(new MyOnItemSelectedListener());

    }

    public class MyOnItemSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> parent,
            View view, int pos, long id) {
          Toast.makeText(parent.getContext(), "El planeta es: " +
              parent.getItemAtPosition(pos).toString(), Toast.LENGTH_LONG).show();
        }

        public void onNothingSelected(AdapterView<?> parent) {
          // Do nothing.
        }
    }
}
```

Si nos fijamos en este código vemos que para poder trabajar con el `Spinner` necesitamos obtener los datos desde el fichero `strings.xml` a través de un array de strings encapsulado en un Adaptador. Para crearlo tan sólo tenemos que crear los items que sean necesarios tal y como se puede ver en la figura 5.18.

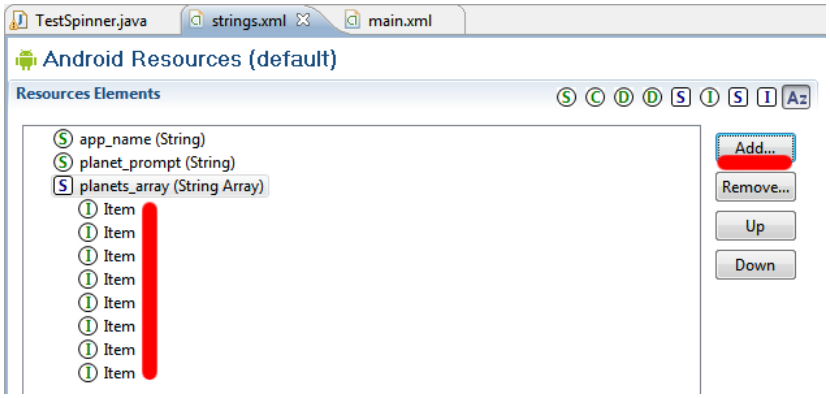

Figura 5.18: Creando el array con el interfaz gráfico de Eclipse

Además, estamos implementando la interfaz `OnItemSelectedListener` de forma que podamos mostrar una marca que muestre el planeta seleccionado. Para ello, utilizamos la clase `Toast` que permite mostrar un pequeño mensaje al usuario, con aspecto flotante, y que no sea intrusivo respecto a la aplicación principal.

5.3. Layouts

Una vez que hemos visto algunas de las vistas básicas[14] que ofrece Android, vamos ahora a explicar algunos de los *layouts*[15] más útiles y más usados. Recordemos que todos los *layouts* son subclases de la clase `ViewGroup`.

5.3.1. `FrameLayout`

Empezaremos por el más sencillo de todos, el `FrameLayout`[16]. Este layout consiste básicamente en un espacio vacío en la pantalla que posteriormente podremos rellenar con un único objeto. Dicho elemento se fijará automáticamente en la esquina superior izquierda de la pantalla.

Debemos tener en cuenta que la idea principal es la de combinar unos *layouts* dentro de otros, a modo de paneles, de forma que nuestra aplicación tenga el aspecto que deseamos.

5.3.2. `LinearLayout`

Un `LinearLayout`[17] alinea todos sus elementos en una única dirección, bien vertical o bien horizontal, dependiendo de cómo definamos el atributo `orientation`. Todos los elementos se colocan uno detrás de otro, de modo que una lista vertical sólo tendrá un elemento por fila, independientemente de lo ancho que sea, mientras que una lista horizontal tendrá una única fila, cuya altura será la altura del elemento más grande que contenga. Podemos ver un ejemplo en la figura

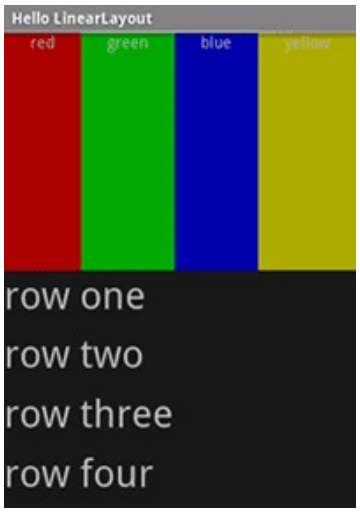

Figura 5.19: Ejemplo de uso de un `LinearLayout`

Un `LinearLayout` también permite definir pesos para los elementos que contiene. Estos pesos asignan una importancia a cada una de estas vistas y les permiten expandirse para rellenar espacio libre en el *layout* que los contiene.

Para crear un *layout* proporcionado en cuanto a tamaño en la pantalla, se recomienda crear un *layout* dándole a los atributos `layout_width` y `layout_height` el valor `fill_parent`, asignarle a los atributos `height` o `width` de los elementos que contenga el valor 0, y entonces asignar un peso (atributo `weight`) a cada elemento con un valor dependiente de la proporción que dicho elemento deba tener en la pantalla.

[14] http://developer.android.com/resources/tutorials/views/index.html
[15] http://developer.android.com/guide/topics/ui/layout-objects.html
[16] http://developer.android.com/reference/android/widget/FrameLayout.html
[17] http://developer.android.com/reference/android/widget/LinearLayout.html

5.3.3. `ListView`

Una `ListView`[18] es una vista de grupo que muestra una lista de elementos. Los elementos son añadidos a la lista a través de un objeto `ListAdapter` que envuelve a los datos que se quieren mostrar en la lista. Un `ListAdapter` puede ser por ejemplo un *Array de String*. El siguiente ejemplo muestra cómo a través del método `setListAdapter` especificamos los datos que queremos mostrar y que obtenemos del fichero `strings.xml` correspondiente.

Contenido del fichero `strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Bahrain</item>
        <item>Bangladesh</item>
        <item>Barbados</item>
        <item>Belarus</item>
        <item>Belgium</item>
        <item>Belize</item>
        <item>Benin</item>
    </string-array>
</resources>
```

Configuración del `ListAdapter`:

```
String[] countries = getResources().getStringArray(R.array.countries_array);
setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item, countries));
```

La figura 5.20 muestra un ejemplo de una `ListView`.

Figura 5.20: Ejemplo de uso de una `ListView`

Para trabajar con los elementos de la lista existen diversos métodos que permiten desde el filtrado de los elementos que contenga la lista, hasta la multiselección de elementos.

Se puede observar un ejemplo de una sencilla aplicación que muestra los países del mundo en una `ListView` en la siguiente dirección:

`http://developer.android.com/resources/tutorials/views/hello-listview.html`

[18] http://developer.android.com/reference/android/widget/ListView.html

5.3.4. `TableLayout`

Un `TableLayout`[19] es una vista de grupo que ordena a los elementos que contiene en filas y columnas. Un `TableLayout` está formado por un numero de filas representadas por objetos del tipo `TableRow` que no muestran bordes en las filas, columnas o celdas. Cada fila tiene 0 o más celdas que pueden contener un objeto de tipo `View`. Una celda puede quedarse vacía y, además, como ocurre en HTML, se pueden unir celdas contiguas para formar bloques internos más grandes. Podemos ver un ejemplo de `TableLayout` en la figura 5.21.

Figura 5.21: Ejemplo de uso de un `TableLayout`

El ancho de una columna viene definido por el ancho de la fila con celda más amplia en dicha columna. No obstante, se puede permitir que el ancho sea extensible o retráctil, de forma que no quede fijo. El ancho total del `TableLayout` viene definido por el ancho de la vista que lo contiene. Por último, indicar que podemos esconder una columna con el método `setColumnCollapsed()`.

Los elementos de una tabla no pueden especificar el atributo `layout_width`, que tendrá siempre el valor: `MATCH_PARENT`. Sin embargo, dichos elementos sí pueden definir el atributo `layout_height` que por defecto tiene el valor `WRAP_CONTENT`.

Otro aspecto importante es que la numeración de las columnas comienza siempre desde 0 y que mediante el método `addView(View child)` podemos añadir nuevos elementos a la tabla. Se puede observar un ejemplo de una sencilla aplicación que muestra un `TableLayout` en la siguiente dirección:

`http://developer.android.com/resources/tutorials/views/hello-tablelayout.html`

[19] http://developer.android.com/reference/android/widget/TableLayout.html

Tema 6

Interfaces de Usuario II. Recursos, pantallas y menús

6.1. Recursos

A la hora de trabajar con los recursos, debemos tener en cuenta que una de las ventajas más interesantes que ofrece Android es la de proporcionar la posibilidad de utilizar recursos alternativos para diferentes configuraciones. A día de hoy existen infinidad de dispositivos que trabajan sobre Android. Por ello, si disponemos de la posibilidad de ofrecer diferentes imágenes según el tamaño de la pantalla, por ejemplo, estaremos ofreciendo un valor añadido a tener muy en cuenta. La personalización de recursos no se limita al tamaño de las pantallas; existen multitud de posibilidades sobre las que trabajar para hacer que nuestra aplicación disponga de gran versatilidad. La figura 6.1 refleja esta idea.

Figure 1. Two device configurations, both using default resources.

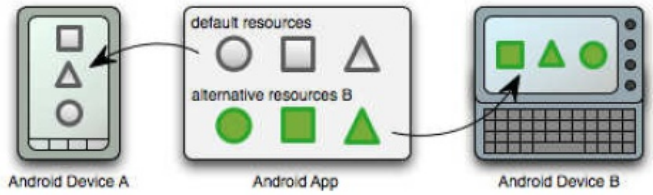

Figure 2. Two device configurations, one using alternative resources.

Figura 6.1: Diferencia entre el uso de los recursos por defecto y el uso de recursos alternativos

Así, para cada tipo de recursos se puede definir el recurso por defecto que la aplicación utilizará y múltiples recursos alternativos. Android, siempre que sea posible, seleccionará el recurso alternativo para una configuración concreta en detrimento del recurso por defecto.

De forma general, los recursos que vamos a poder especificar en cualquier aplicación para Android son los que podemos observar en la tabla 6.1. Todos ellos representan una carpeta que debe pertenecer a su vez a la carpeta `res`.

Directorio	Tipo de Recurso
`anim`	Archivos XML que definen animaciones.
`color`	Archivos XML que definen una lista de colores
`drawable`	Archivos de mapa de bits (.png, .9.png, .jpg, .gif) o bien, archivos XML.
`layout`	Archivo XML que define una interfaz de usuario.
`menu`	Archivo XML que define menús de aplicación.
`raw`	Archivos que se guardan en su formato original.
`values`	Archivos XML que contienen valores simples, como ya hemos visto.
`xml`	Ficheros XML opcionales de configuración que puedan necesitarse.

Tabla 6.1: Directorios de recursos en Android

Señalar que para el caso de la carpeta `drawable`, si el recurso se especifica a través de un fichero XML, el mismo deberá representar alguno de los siguientes tipos de dibujo[1]: Mapas de bits, mapas de bits ajustables en tamaño, listas de estado, sombras o animaciones. En este curso, sin embargo, no trabajaremos con este tipo de ficheros, sino que haremos uso de las imágenes convencionales en `.jpg` o `.gif`.

También tener en cuenta que en la carpeta `raw` podemos situar ficheros en sus formatos originales (como por ejemplo audio). Android nos ofrece la posibilidad de trabajar con ellos haciendo uso de la función `Resources.openRawResource()` junto con el ID del recurso. Dicho ID se puede localizar a través de `R.raw.filename`.

Para la carpeta `values`, existe una norma de estilo habitual que nos recomienda que para cada tipo de recurso básico utilicemos un archivo XML de forma similar a como lo hemos hecho hasta ahora. Así, lo recomendado es lo siguiente:

- `arrays.xml` para Arrays.
- `colors.xml` para valores de colores.
- `dimens.xml` para valores dimensionales.
- `strings.xml` para cadenas.
- `styles.xml` para estilos.

Una vez vistos todos los recursos que pueden aparecer en escena en una aplicación Android vamos a ver cómo Android nos ofrece un sistema para especificar las configuraciones que necesitemos en nuestra aplicación. La idea que tenemos que tener en mente es que debemos añadir el valor específico a la opción por defecto.

Por ejemplo, si queremos especificar un *layout* concreto para cuando el usuario gire 90º el terminal (en esta situación Android cambia la visualización del programa de vertical a horizontal, por lo que su aspecto puede cambiar significativamente), hemos de crear la carpeta `layout-land` como composición de la carpeta por defecto `layout` más el identificador `land`.

De forma general siempre que queramos especificar una configuración concreta seguiremos este esquema:

1. Creamos un nuevo directorio en la carpeta RES/ que tenga la siguiente forma:

   ```
   nombre_del_recurso-nombre_de_la_configuración
   ```

[1] http://developer.android.com/guide/topics/resources/drawable-resource.html

Donde:

a) "nombre del recurso" es el nombre del directorio del recurso por defecto correspondiente.

b) "nombre de la configuración" es un nombre que especifica una configuración para los recursos que van a ser utilizados.

2. Guardar los recursos alternativos en el nuevo directorio. El nombre del recurso debe llamarse exactamente igual que en la carpeta del recurso por defecto.

Un ejemplo de esta configuración podemos verla en la figura 6.2.

Figura 6.2: Ejemplo de carpetas específicas para distintas configuraciones

En dicho ejemplo se está especificando la imagen que debe aparecer para tres tipos distintos de resolución de pantalla.

En la tabla 6.3 podemos ver de forma resumida todas las distintas configuraciones para la definición de recursos en Android. Debemos respetar el orden de aparición si hacemos composiciones de configuraciones, de forma que un elemento que aparezca en la tabla en una posición superior a otra deberá aparecer antes en el nombre compuesto del directorio.

Por ejemplo:

- `drawable-port-hdpi/` sería un nombre correcto, mientras que
- `drawable-hdpi-port/` no sería un nombre válido

Característica	Valor	Descripción
MCC y MNC	mcc310 mcc310-mnc004 etc.	Código móvil de país (Mobile country code), opcionalmente seguido por el MNC o Código de red de la SIM Card. Por ejemplo: mcc208-mnc00 es Orange en Francia.
Idiomas y regiones	en fr etc.	El idioma se define en base al código de dos letras `ISO 639-1`, seguido opcionalmente por el código de región `ISO 3166-1-alpha-2`

Tamaño de pantalla	small (pequeño) normal large (grande) xlarge (extra grande)	**small:** Pantallas de baja densidad con tamaño QVGA. **normal:** Pantallas basada en el tamaño habitual de densidad media HVGA. **large:** Pantallas basadas en el espacio disponible en una pantalla VGA. **xlarge:** Pantallas que son considerablemente más grandes que las trandicionales HVGA.
Amplitud de pantalla	long (ancha) notlong (normal)	**long:** Pantallas anchas, como las `WQVGA`, `WVGA` y `FWVGA` **notlong:** Pantallas normales, como las `QVGA`, `HVGA`, and `VGA`
Método de anclaje	car desk	**car:** El dispositivo está "anclado" al coche. **desk:** El dispositivo está en el escritorio
Modo nocturno	night (activado) notnight (desactivado)	**night:** Tiempo nocturno. **notnight:** Tiempo diurno.
Densidad de pixels por pantalla (dpi)	ldpi mdpi hdpi xhdpi nodpi	**ldpi:** Pantallas de baja densidad; 120dpi. **mdpi:** Pantallas de media densidad; 160dpi. **hdpi:** Pantallas de alta densidad; 240dpi. **xhdpi:** Pantallas de muy alta densidad; 320dpi. **nodpi:** Usado para los recursos de mapa de bits que no se desea que se escalen para coincidir con la densidad del dispositivo.
Tipo de pantalla táctil	notouch stylus finger	**notouch:** El dispositivo no dispone de pantalla táctil. **stylus:** Pantalla diseñada para su uso con lápiz. **finger:** Pantalla diseñada para ser táctil.
Disponibilidad de teclado	keysexposed keyssoft	**keysexposed:** El dispositivo cuenta con teclado disponible. **keyshidden:** El dispositivo cuenta con teclado disponible pero está oculto. **keyssoft:** El dispositivo cuenta con teclado software disponible, esté o no oculto.
Método de entrada de texto	nokeys qwerty 12key	**nokeys:** El dispositivo no tiene método físico de entrada de datos. **qwerty:** El dispositivo cuenta con un teclado qwerty completo. **12key:** Hardware de 12 teclas.
Disponibilidad de teclas de navegación	navexposed navhidden	**navexposed:** Las teclas están disponibles para el usuario. **navhidden:** Las teclas no están disponibles para el usuario o están ocultas.
Método primario de navegación no táctil	nonav dpad trackball wheel	**nonav:** El dispositivo solo cuenta con la pantalla táctil. **dpad:** El dispositivo cuenta con un teclado direccional `d-pad` para la navegación. **trackball:** El dispositivo cuenta con una trackball para navegar. **wheel:** El dispositivo cuenta con una rueda direccional.

Versión del sistema (API Level)	v3 v4 v7 etc.	El nivel del API soportado por el sistema.

Tabla 6.3: Configuraciones para la definición de recursos en Android

6.1.1. Creando alias para los recursos

Cuando tengamos un recurso que nos gustaría utilizar para más de una única configuración pero no para todas, no es necesario poner el mismo recurso en cada carpeta de recurso alternativo. En vez de eso, podemos crear un recurso alternativo que actúa como un alias para un recurso almacenado en la carpeta del recurso por defecto.

Para que veamos un ejemplo sencillo, podemos fijarnos en que cuando creamos un nuevo proyecto nos aparecen dentro de la carpeta res/ las carpetas drawable-hdpi, drawable-ldpi y drawable-mdpi, conteniendo las tres carpetas la misma imagen: `icon.png`. Vamos a intentar evitar tener tres copias de la misma imagen haciendo uso de un alias.

La mecánica es muy sencilla:

1. Creamos el directorio por defecto colgado del directorio res/.
2. Introducimos en la carpeta recién creada la imagen que queremos se utilice para varias configuraciones con un nombre diferente al que tiene por defecto (en nuestro ejemplo `icon.png`).
3. Creamos un fichero XML llamado `icon.xml` en las tres carpetas que referirá a la imagen que hemos introducido, que contenga el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
 <bitmap xmlns:android="http://schemas.android.com/apk/res/android"
         android:src="@drawable/icon_resol" />
```

En el fondo lo único que estamos haciendo es crear un pequeño fichero en cada carpeta que sustituya a la imagen original que tiene un tamaño mucho mayor y que informa a la aplicación sobre donde debe buscar la imagen.

Cuando trabajemos con imágenes utilizaremos la etiqueta bitmap. Si por el contrario quisiéramos crear un alias sobre un *layout* existente utilizaríamos la etiqueta <merge> que a su vez debe tener en su interior la etiqueta <include> tal y como podemos ver a continuación.

```
<?xml version="1.0" encoding="UTF-8"?>
 <merge>
     <include layout="@layout/main_ltr"/>
 </merge>
```

Para el caso de cadenas u otros valores simples, se utilizará simplemente la etiqueta <resources>:

```
<?xml version="1.0" encoding="UTF-8"?>
 <resources>
     <string name="hello">Hello</string>
     <string name="hi">@string/hello</string>
</resources>
```

6.2. Transiciones entre pantallas

En esta sección explicaremos cómo desarrollar una aplicación con múltiples pantallas, y cómo pasar de una a otra. Para ello, los pasos genéricos a seguir son los siguientes:

1. Crear un nuevo *layout* dentro de la carpeta `res/layout` por cada pantalla.
2. Crear una clase de tipo `Activity` por cada pantalla.
3. Modificar el `Manifest.xml` de la aplicación para que incluya todas las actividades con sus correspondientes *intents*.
4. Añadir el código en sí que permite la transición de una pantalla a otra.

A continuación vamos a ir desarrollando, mediante una aplicación de ejemplo, cada uno de estos puntos.

6.2.1. Un *layout* por cada pantalla

Vamos a crear una aplicación llamada `ProbandoPantallas` que tendrá dos pantallas con un botón cada una. Cuando pulsemos en el botón de la primera pantalla nos moveremos a la segunda pantalla, y viceversa.

Una vez creada la aplicación inicial, definiremos los textos que nos harán falta dentro del fichero `strings.xml` de la carpeta `res/values`. Necesitaremos un total de cuatro cadenas de texto, tal y como podemos comprobar en la tabla 6.4.

Nombre	Valor
`screen1TextView`	`Estamos en la pantalla 1`
`screen2TextView`	`Estamos en la pantalla 2`
`screen1ButtonText`	`Ir a la pantalla 2`
`screen2ButtonText`	`Ir a la pantalla 1`

Tabla 6.4: `Strings` de la aplicación `ProbandoPantallas`

A continuación editaremos el fichero `main.xml` desde la pestaña GraphicalLayout para que contenga un `TextView` y un `Button`, asociándole las etiquetas correspondientes desde los recursos de tipo `String` previamente definidos, tal y como se observa en la figura 6.3.

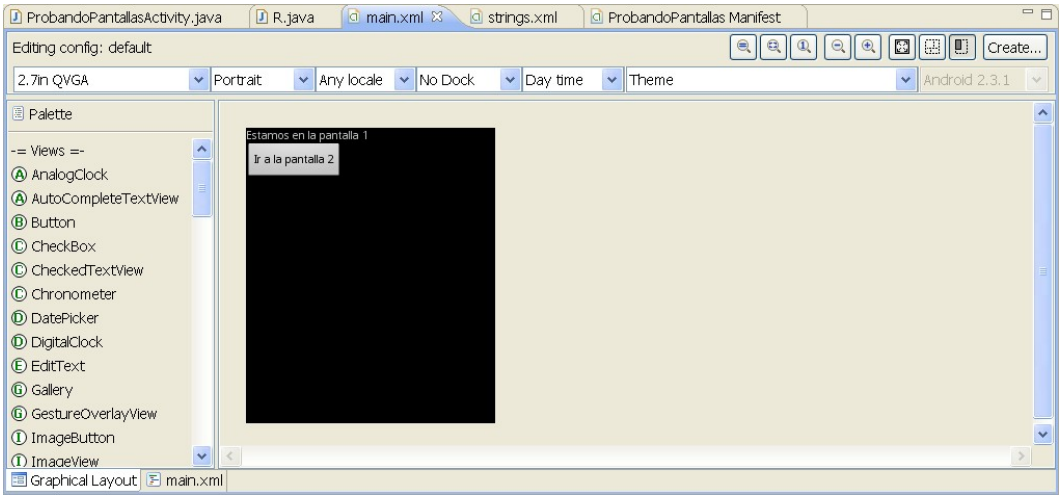

Figura 6.3: Aspecto del *layout* de la pantalla número 1

El siguiente paso a realizar consiste en crear el segundo *layout* que representará a la segunda pantalla. Para ello, pulsaremos en Eclipse sobre la opción File→New→Other..., y a continuación en Android XML File, dentro de la carpeta Android. Entonces nos aparecerá una ventana como la que se observa en la figura 6.4, donde especificaremos el nombre del nuevo recurso (`pantalla2`), el tipo (layout), la ruta (`/res/layout`), y para nuestro caso concreto, el tipo de *layout* que queremos (`LinearLayout`).

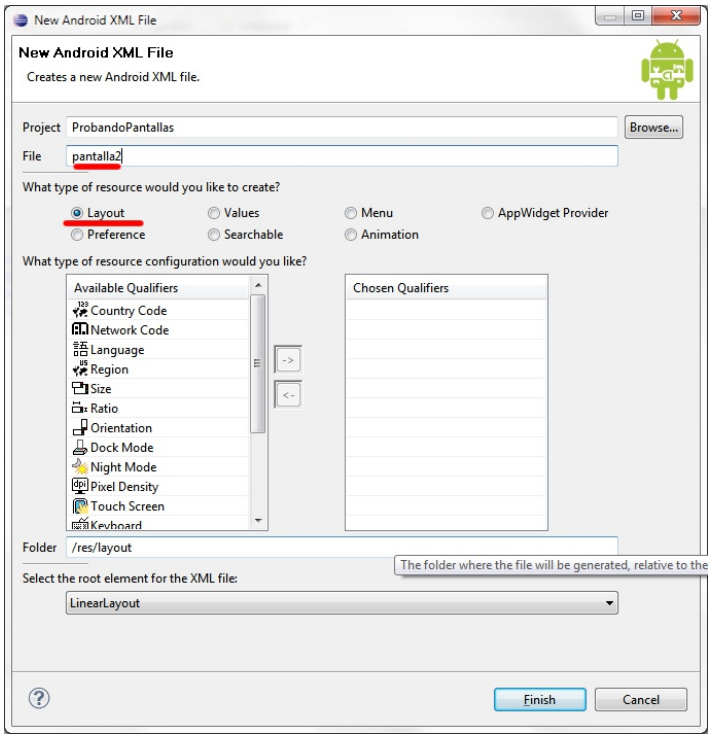

Figura 6.4: Creando el *layout* de la pantalla 2

Al igual que hicimos con el fichero `main.xml`, deberemos añadir a la pantalla representada en el fichero `pantalla2.xml`, mediante la pestaña GraphicalLayout, las vistas correspondientes, tal y como podemos comprobar en la figura 6.5.

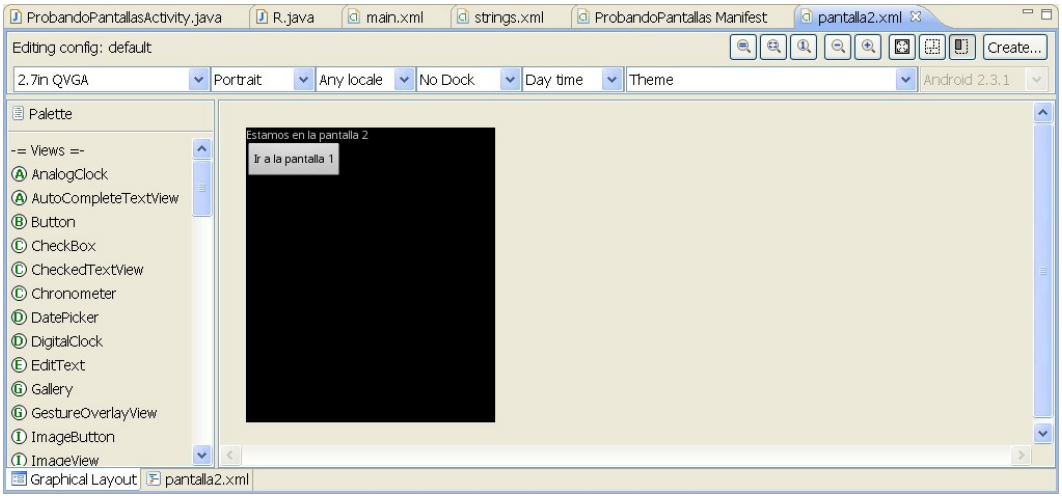

Figura 6.5: Aspecto del *layout* de la pantalla número 2

6.2.2. Una `Activity` por cada pantalla

Como ya hemos comentado anteriormente, cada pantalla tendrá asociada un objeto de la clase `Activity` donde se desarrollará su comportamiento específico.

Por lo tanto, a continuación deberemos crear una nueva clase asociada a la pantalla 2. Para ello pulsaremos sobre la opción de Eclipse File→New→Class, tras lo cual nos aparecerá una ventana como la de la figura 6.6.

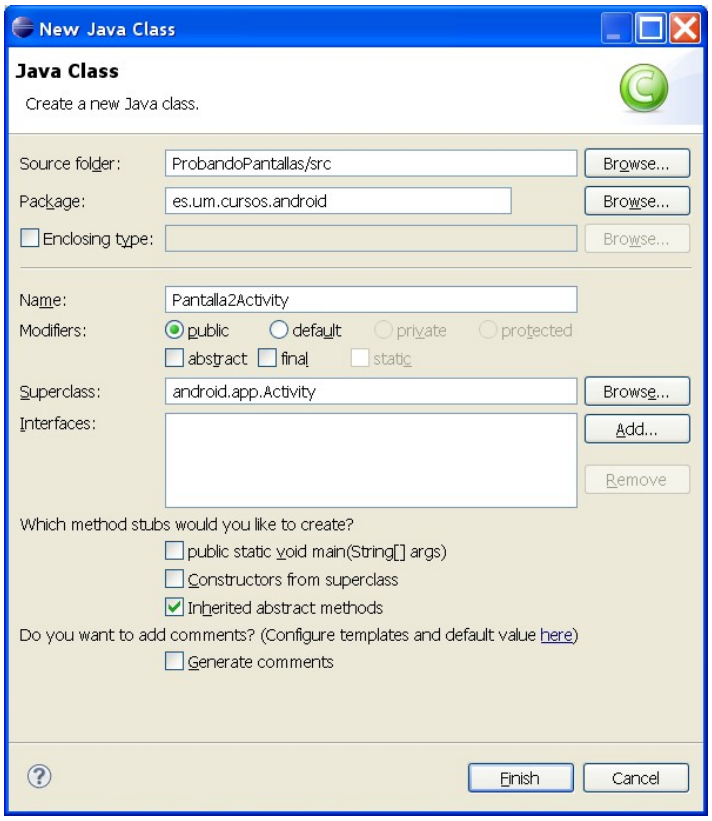

Figura 6.6: Creando la nueva actividad de la pantalla 2

Como podemos observar, deberemos especificar el nombre de la nueva clase (`Pantalla2Activity`), el paquete (`es.um.cursos.android`) y la superclase (`android.app.Activity`), tras lo cual pulsaremos sobre el botón Finish.

Por último, sobreescribiremos el método `onCreate()` de manera similar a como teníamos hecho para la primera pantalla, es decir, añadiremos el siguiente código:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.pantalla2);
}
```

Obsérvese que ahora el método `setContentView()` no recibe el *layout* `main`, sino que recibe el que se define en el fichero `pantalla2.xml`.

6.2.3. *Intents* en el `Manifest.xml`

Como ya hemos comentado anteriormente, el fichero `Manifest.xml` es responsable de contener un "índice" de las actividades (y por consiguiente de las pantallas) que contendrá nuestra aplicación. La idea que debemos tener clara es que para cada pantalla debemos contar con un elemento Activity en nuestro fichero `Manifest.xml`, y que a su vez debemos contar con un `.java` que regule el comportamiento que debe tener esa pantalla en la carpeta `src`.

Así por ejemplo, si examinamos el fichero `Manifest.xml` de nuestra aplicación `ProbandoPantallas`, podremos ver que tiene el siguiente aspecto:

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

A continuación explicaremos para qué sirven las etiquetas intent-filter y cómo declarar las actividades necesarias para nuestra aplicación en el fichero de manifiesto de la misma. De forma sencilla, si una Activity representa a una pantalla, un intent-filter representa la manera de invocar a esas pantallas. O dicho de otra forma, un `Intent` es una clase que permite especificar una `Activity` a ejecutar. Veamos un sencillo ejemplo donde se vea claro el concepto:

```
Intent intent = new Intent(getApplicationContext(),
                                ProbandoPantallasActivity.class);
startActivity(intent);
```

Mediante este sencillo código, lo que estamos haciendo es crear una instancia "`intent`" de tipo `Intent` para la cual se necesitan dos parámetros, a saber: el contexto actual, es decir, el paquete donde se encuentra la clase de la pantalla que queremos mostrar, y la clase que representa a la pantalla que queremos mostrar como tal. Mediante la invocación del método `startActivity()`, lo que hacemos es simplemente iniciar dicha actividad y por tanto mostrar la pantalla que tiene asociada.

Lo más interesante de los `Intents` se encuentra en que a un `Intent` podemos asociarle una acción determinada, unos datos y una categoría. Veámoslo con el siguiente ejemplo:

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

Aquí se está detallando, por una parte, mediante la etiqueta <`action`>, que nuestra actividad puede mostrar datos al usuario (`VIEW`). Por otra parte, mediante la etiqueta <`category`>, estamos especificando una categoría que define si la actividad será lanzada desde el lanzador de aplicaciones del terminal, desde el menú de otra aplicación o directamente desde otra actividad. Para el ejemplo expuesto, la categoría `LAUNCHER` especifica que la actividad será lanzada una vez el usuario ejecute la aplicación desde el menú de aplicaciones del terminal.

Existen muchos valores tanto para las etiquetas <`action`> como para las etiquetas <`category`>, que se pueden consultar en la siguiente dirección web:

`http://developer.android.com/reference/android/content/Intent.html`

Una vez visto esto, vamos a explicar cómo introducir en el fichero `Manifest.xml` de nuestra aplicación `ProbandoPantallas` la información necesaria para poder mostrar las dos pantallas. Si nos fijamos en la figura 6.7 veremos que en la pestaña Application podemos definir diferentes atributos junto con los nodos o Activities que queramos.

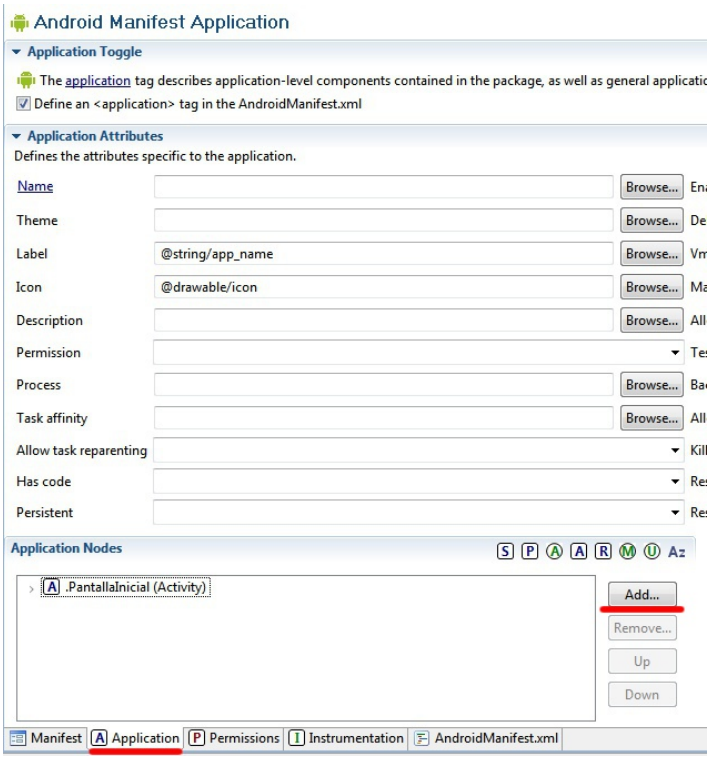

Figura 6.7: Trabajando con Intents desde modo gráfico

Vamos a añadir una nueva `Activity` con sus correspondiente `Intent`, de modo que pulsamos sobre el botón Add... y seleccionamos la opción de Activity en la ventana que nos aparecerá a continuación.

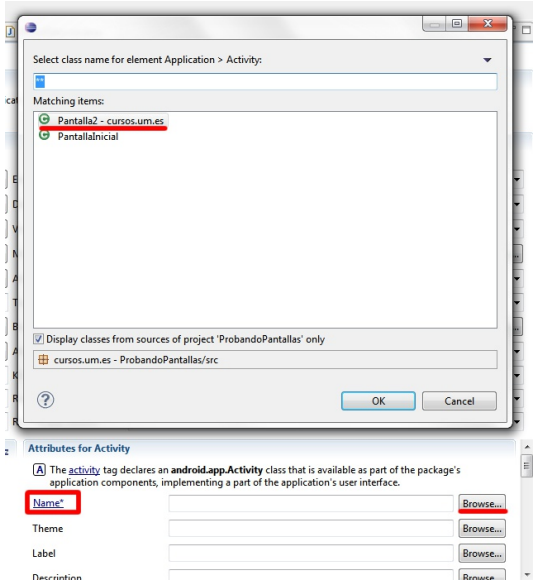

Figura 6.8: Modificando los parámetros del nuevo nodo

Como podemos observar, Eclipse nos permite especificar multitud de propiedades para cada `Activity`. Nosotros por ahora nos limitaremos a asignarle un nombre que será, a través del botón Browse, el nombre de la clase `Activity` que representa la segunda pantalla, es decir, `Pantalla2Activity`, tal y como observamos en la figura 6.8. También le asignaremos al atributo Label el texto "`Segunda Pantalla`".

Una vez añadido el nodo, añadiremos la etiqueta "Intent-filter" al mismo. Para ello, sobre el nodo .Pantalla2Activity pulsaremos con el botón derecho del ratón para que Eclipse nos muestre el menú que podemos ver en la figura 6.9

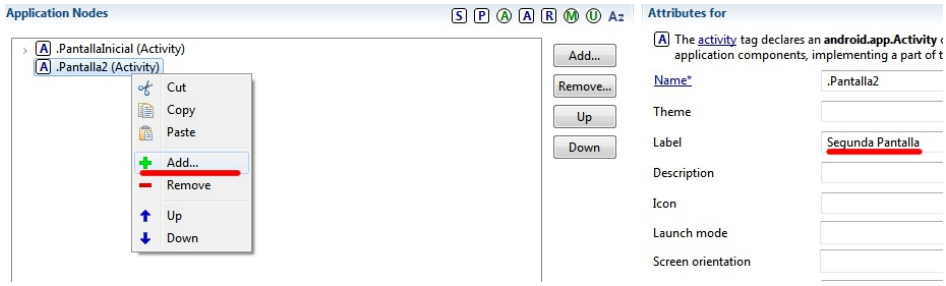

Figura 6.9: Creando la etiqueta Intent-filter

Repetiremos el proceso, dentro del nuevo elemento Intent-filter para añadir la etiqueta Action y la etiqueta Category y les daremos como valores VIEW para la etiqueta Action, y DEFAULT para la etiqueta Category, tal y como se observa en la figura 6.9

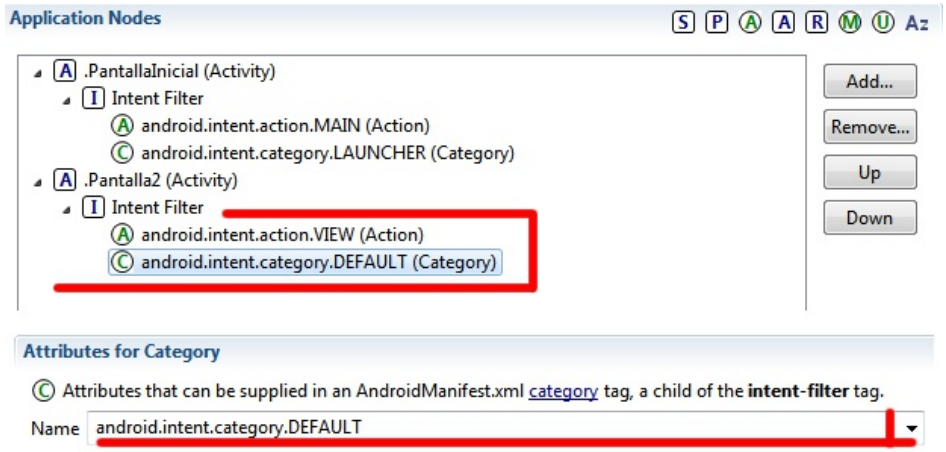

Figura 6.10: Adaptando las etiquetas en el fichero Manifest

6.2.4. Código de transición entre pantallas

Una vez que ya hemos definido un nuevo layout y una nueva Activity para la nueva pantalla, y hemos incluido dicha Activity con su correspondiente Intent en el fichero Manifest.xml de la aplicación, tan sólo nos queda insertar el código que efectivamente realizará la transición entre pantallas.

Como ya explicamos, nuestra aplicación ProbandoPantallas constará de dos pantallas con un botón cada una, que nos servirá para movernos de una a otra. Así, el código que necesitaremos en la clase `ProbandoPantallasActivity` para pasar de la primera pantalla a la segunda sería el que se muestra a continuación:

```
Button screen1Button = (Button) findViewById(R.id.screen1Button);
screen1Button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent intent =
               new Intent(getApplicationContext(), Pantalla2Activity.class);
        startActivity(intent);
    }
});
```

Mientras que para el caso de la clase Pantalla2Activity, necesitaríamos el siguiente código:

```
Button screen2Button = (Button) findViewById(R.id.screen2Button);
screen2Button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent intent =
          new Intent(getApplicationContext(), ProbandoPantallasActivity.class);
        startActivity(intent);
    }
});
```

Actividad propuesta I.17. *Desarrollar la aplicación* `ProbandoPantallas`*, tal y como se ha explicado anteriormente.*

6.3. Menús

Para finalizar, tan sólo nos resta hablar de los menús[2] en Android. De forma general la mayoría de aplicaciones de Android cuentan con un menú que les sirve para dar funcionalidad añadida a las mismas.

Para mostrar el funcionamiento de los mismos, vamos a añadir un menú a la aplicación `ConversorDeDivissas` que desarrollábamos en el tema anterior que contendrá un par de elementos para realizar la conversión de euros a dólares y viceversa.

Para ello, en primer lugar, creamos un nuevo recurso de tipo menú mediante la opción de Eclipse File→New→Other.... En la pantalla que nos aparece, deberemos seleccionar la opción de Menu, tal y como se observa en la figura 6.11.

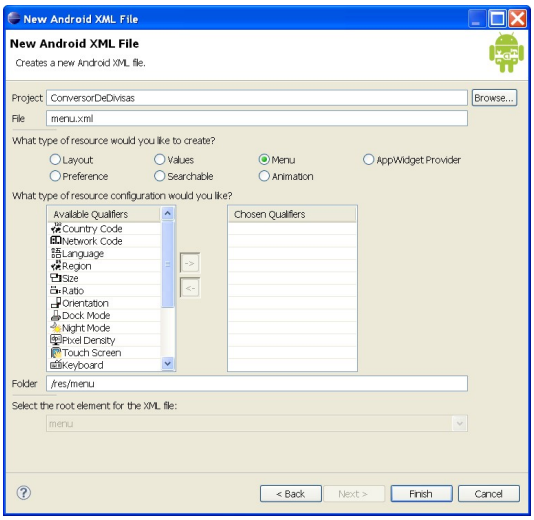

Figura 6.11: Creando un nuevo menú

[2] http://developer.android.com/guide/topics/ui/menus.html

A continuación, deberemos añadir un par de elementos (*menu items*) a nuestro menú recién creado. Para ello, seleccionamos el fichero `menu.xml` y, desde la pestaña Layout, agregamos estos ítems, tal y como se observa en la figura 6.12.

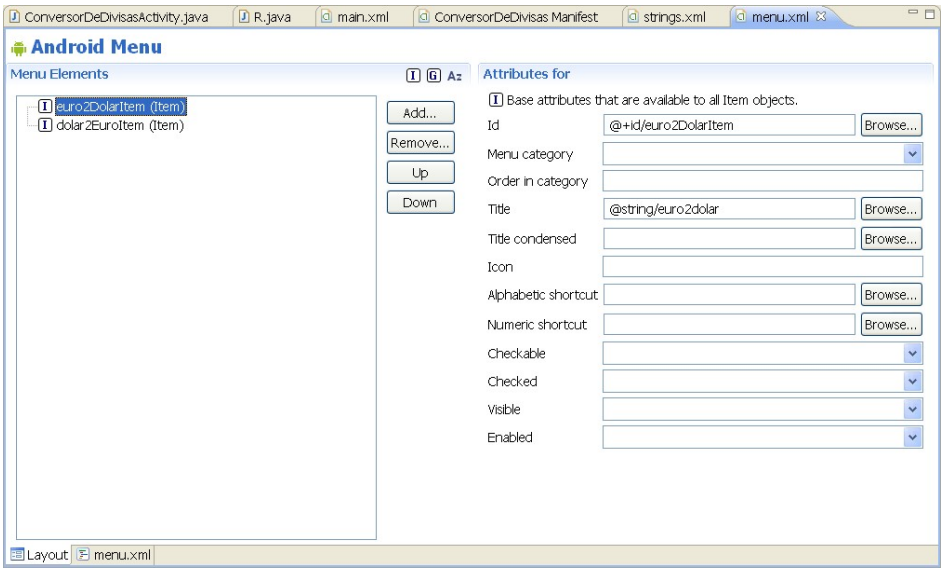

Figura 6.12: Añadiendo nuevos ítems al menú

Por último, nos queda asociar este menú que ya hemos creado con la pantalla principal, así como dotar de funcionalidad a cada uno de sus elementos. Para conseguir lo primero, haremos uso de la clase `MenuInflater`[3], que nos permitirá rellenar un objeto de la clase `Menu`[4] con el menú definido como recurso, mediante el siguiente código en la clase `Activity`:

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}
```

Por último, para que al pulsar cada elemento del menú se realice la acción que nosotros deseamos, haremos uso del siguiente código:

```
public boolean onMenuItemSelected(int featureId, MenuItem item) {
    super.onMenuItemSelected(featureId, item);
    switch (item.getItemId()) {
        case R.id.euro2DolarItem:
            euro2Dolar();
            break;
        case R.id.dolar2EuroItem:
            dolar2Euro();
            break;
    }
    return true;
}
```

Actividad propuesta I.18. *Ampliar la aplicación `ConversorDeDivisas` para que incluya un menú tal y como se ha explicado anteriormente.*

[3] http://developer.android.com/reference/android/view/MenuInflater.html
[4] http://developer.android.com/reference/android/view/Menu.html

Tema 7

Actividad final

Nota.- La entrega de la actividad final deberá ajustarse a lo indicado en la sección 5 de Metodología y Evaluación, en capítulo de Introducción de estos apuntes (página 19).

7.1. Descripción

Con todos los conocimientos adquiridos a lo largo del curso y todas las posibilidades de Android que se han mostrado, se propone al alumno realizar una aplicación que haga uso de buena parte de ellas.

A continuación se proponen algunas opciones, aunque si el alumno así lo desea, puede realizar su propia aplicación consultándolo **previamente** con los profesores (`felixgm@um.es` y `antonio.bernardez@um.es`).

- **Propuesta 1**

Diseñar e implementar una aplicación sencilla que sirva como calculadora para el móvil.

- **Propuesta 2**

Diseñar e implementar una aplicación sencilla que sirva para jugar al juego de las 3 en raya.

- **Propuesta 3**

Diseñar e implementar una aplicación que sirva como agenda de contactos, teniendo en cuenta que los datos no se guardarán al cerrar la aplicación.

- **Propuesta 4**

Diseñar e implementar una aplicación que sirva como calculadora hipotecaria, de tal modo que al introducir un importe solicitado, el número de meses al que se pide el préstamo y el interés del mismo, nos muestre la cuota mensual que deberemos pagar, así como la cantidad total de dinero que pagaremos. Mejorar esta aplicación con más opciones.

- **Propuesta 5**

Diseñar e implementar una aplicación que sirva como galería de imágenes con la posibilidad de dar una puntuación a cada una de ellas mediante un `RatingBar`. Mejorar esta aplicación con más opciones.

- **Propuesta 6**

Diseñar e implementar una aplicación que haga uso de algunas de las vistas complejas que no se han visto en este curso, como `Gallery`, `DatePicker`, `WebView`, `VideoView`, `RatingBar`, etc.

Parte II

Desarrollo Avanzado de Aplicaciones para Dispositivos Móviles *Android*

Tema 1

Ficheros y Bases de Datos

1.1. Manejo de ficheros

En esta sección veremos cómo manejar ficheros en Android, tanto en la memoria interna del dispositivo, como en la memoria externa (tarjetas SD, MMC, etc) del mismo.

1.1.1. Ficheros en la memoria interna del dispositivo

Android permite gestionar ficheros directamente en la memoria interna del dispositivo. Por defecto, los ficheros almacenados en la memoria interna son privados a la aplicación que los crea, lo que significa que ninguna otra aplicación tiene acceso a ellos, y que al desinstalar la aplicación que los creó, dichos ficheros se borran.

Las principales clases para el manejo de ficheros en Android son `FileInputStream` [1] y `FileOutputStream` [2]. Para abrir un fichero en la memoria interna del dispositivo con la intención de volcar contenido en el mismo, haremos uso del método `openFileOutput()` de la clase `Activity` (heredado en realidad de la clase abstracta `Context`).

Este método recibe como parámetros el nombre del fichero que queramos abrir, sin caracteres separadores de ruta, así como el modo concreto en el que queremos abrir o crear el fichero.

La tabla 1.1 muestra los distintos modos que podemos usar en el método `openFileOutput()`. El siguiente código muestra un ejemplo de uso de dicho método.

```
String fichero = "miFichero";
String contenido = "Cursos de Android";

try {
  FileOutputStream fos1 = openFileOutput(fichero, Context.MODE_PRIVATE);
  fos1.write(contenido.getBytes());
  fos1.close();

  FileOutputStream fos2 = openFileOutput(fichero, Context.MODE_APPEND);
  fos2.write(contenido.getBytes());
  fos2.close();
} catch (FileNotFoundException e) {
  ...
} catch (IOException e) { ... }
```

[1] http://developer.android.com/reference/java/io/FileInputStream.html
[2] http://developer.android.com/reference/java/io/FileOutputStream.html

Modo	Descripción
`MODE_APPEND`	Si el fichero existe, añadir contenido al final del mismo. Si no existe, se crea
`MODE_PRIVATE`	Modo por defecto donde el fichero sólo puede ser accedido por la aplicación que lo crea
`MODE_WORLD_READABLE`	Permite a las demás aplicaciones tener acceso de lectura al fichero creado
`MODE_WORLD_WRITEABLE`	Permite a las demás aplicaciones tener acceso de escritura al fichero creado

Tabla 1.1: Modos de creación de ficheros para el método `openFileOutput()`

Por su parte, para abrir un fichero de la memoria interna del dispositivo con la intención de leer su contenido, haremos uso del método homólogo `openFileInput()` de la clase `Activity` (también heredado de la clase abstracta `Context`). Este método solamente recibe como parámetro el nombre del fichero en cuestión que queramos abrir.

Otros métodos interesantes para el manejo de ficheros heredados de la clase `Context` son los que se muestran en la tabla 1.2.

Método	Descripción
`getFilesDir()`	Obtiene la ruta absoluta del fichero en el sistema de directorios cuando el fichero interno ha sido creado
`getDir()`	Crea (o abre si ya existe) un directorio dentro de la memoria interna
`deleteFile()`	Borra un fichero guardado en la memoria interna
`fileList()`	Devuelve un array de ficheros que están guardados en la carpeta de la aplicación

Tabla 1.2: Métodos de la clase `Context` para el manejo de ficheros

1.1.2. Aplicación `TestFicheros`

Para probar la funcionalidad del manejo de ficheros en la memoria interna del dispositivo, vamos a crear una aplicación llamada `TestFicheros`. Dicha aplicación contará con una pantalla principal que contendrá únicamente un `ListView`.

Dicho `ListView`, a su vez, contendrá tan solo dos elementos de texto: "*Leer ficheros*" y "*Escribir fichero*", de tal forma que al pulsarlos nos lleven a sendas pantallas de lectura y escritura de ficheros.

La pantalla de escritura contará con un campo para indicar el nombre del fichero que queramos crear, así como otro campo para introducir el contenido del fichero de texto. Por último, contendrá un botón para crear el fichero indicado con el contenido introducido, así como un botón para volver a la pantalla principal de la aplicación.

Por su parte, la pantalla de lectura de ficheros contendrá una lista donde se mostrarán todos los ficheros de la memoria interna asociados a nuestra aplicación, así como un campo que nos mostrará el contenido de cada fichero sobre el que pulsemos. Por último, también contendrá un par de botones: uno para borrar todos los ficheros de la memoria interna asociados a nuestra aplicación, y otro para volver a la pantalla principal.

La figura 1.1 muestra las tres pantallas de que consta la aplicación `TestFicheros`.

Figura 1.1: Pantallas de la aplicación `TestFicheros`

Actividad propuesta II.1. *Desarrollar y probar la aplicación `TestFicheros`, tal y como se ha descrito anteriormente.*

1.1.3. Ficheros en la memoria externa del dispositivo

Android también nos permite acceder y manejar los ficheros almacenados en la memoria externa del dispositivo, tales como una tarjeta SD, MMC, etc.

Para ello, en primer lugar deberemos comprobar la disponibilidad de dicha memoria externa. Para ello podemos hacer uso del método `getExternalStorageState()` de la clase `Environment` [3]. Con el siguiente fragmento de código podemos ver el estado en el que se encuentra la memoria externa del dispositivo.

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (state.equals(Environment.MEDIA_MOUNTED)) {
  // Podemos leer y escribir en la memoria externa
  mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (state.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
  // Podemos leer la memoria externa pero no escribir en ella
  mExternalStorageAvailable = true;
  mExternalStorageWriteable = false;
} else {
  // No podemos leer ni escribir la memoria externa
  mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

Una vez comprobada la disponibilidad de la memoria externa haremos uso del método `getExternalFilesDir()` de la clase `Context` para abrir el archivo el cuestión que necesitemos.

[3] http://developer.android.com/reference/android/os/Environment.html

Si queremos guardar archivos en la memoria externa que no queremos que sean específicos de nuestra aplicación y que no se borren cuando se desinstalemos la aplicación, debemos guardarlos en alguno de los directorios públicos de nuestra memoria externa. Estos directorios cuelgan de la raíz de la memoria externa y pueden ser: `Music/`, `Pictures/`, `Ringtones/` etc.

1.1.4. Sistemas de ficheros en el emulador

Para poder utilizar nuestra tarjeta de memoria SD desde nuestro emulador debemos realizar los siguientes pasos.

1. Creamos una imagen en nuestro disco duro de lo que será la memoria que usará el dispositivo mediante el comando `mksdcard`. Para ello nos desplazaremos al directorio `tools` de nuestra plataforma Android. En nuestro caso, la carpeta tools se encuentra en la siguiente ubicación:

   ```
   [...]\android-sdk\tools
   ```

2. Escribiremos el siguiente comando encargado de la creación de la imagen de la tarjeta de memoria:

   ```
   mksdcard 512M [...]/workspace/android_sdcard
   ```

3. Una vez que hayamos creado la imagen de la tarjeta de memoria iremos a la carpeta a comprobar que efectivamente se ha creado, y veremos que nos aparecerá un fichero con un tamaño de 512M.

4. Ahora debemos configurar Eclipse para que cuando lance el emulador tenga en cuenta la tarjeta de memoria que acabamos de crear. Dentro del menú Run → Run Configurations debemos acceder a la pestaña Target.

5. Elegimos el emulador virtual (si tenemos más de uno) sobre el que queremos asignarle la memoria reservada y al final de dicha pestaña vemos una opción que pone Additional Emulator Command Line Options. En esta línea debemos anotar lo siguiente (tal y como se observa en la figura 1.2):

   ```
   -sdcard /workspace/android_sdcard
   ```

6. Pulsamos en Apply para guardar los cambios y ya podemos ejecutar nuestra aplicación haciendo uso de la memoria externa.

Una vez que hemos configurado el emulador para que pueda hacer uso de la memoria externa, vamos a ver cómo podemos cargar ficheros desde nuestro ordenador a la tarjeta SD del emulador, para que nuestra aplicación pueda hacer uso de dichos ficheros (por ejemplo, un vídeo).

1. Pulsamos sobre el menú Window → Open Perspective → Other.... En la ventana que se nos abre, pulsamos sobre DDMS (ver figura 1.3).

2. Seleccionamos la pestaña File Explorer y pulsamos sobre el botón PUSH, que tiene un icono de un teléfono.

3. Seleccionamos entonces el archivo que queremos añadir a nuestra tarjeta de memoria (en nuestro caso un video). Eclipse comenzará a cargar el video en nuestra tarjeta SD virtual.

Actividad propuesta II.2. *Ampliar la aplicación desarrollada en la actividad II.1. para que también maneje ficheros de la memoria externa, cuando ésta esté disponible. Añadir manualmente algún fichero de texto en la tarjeta SD virtual para recuperar su contenido mediante la aplicación desarrollada.*

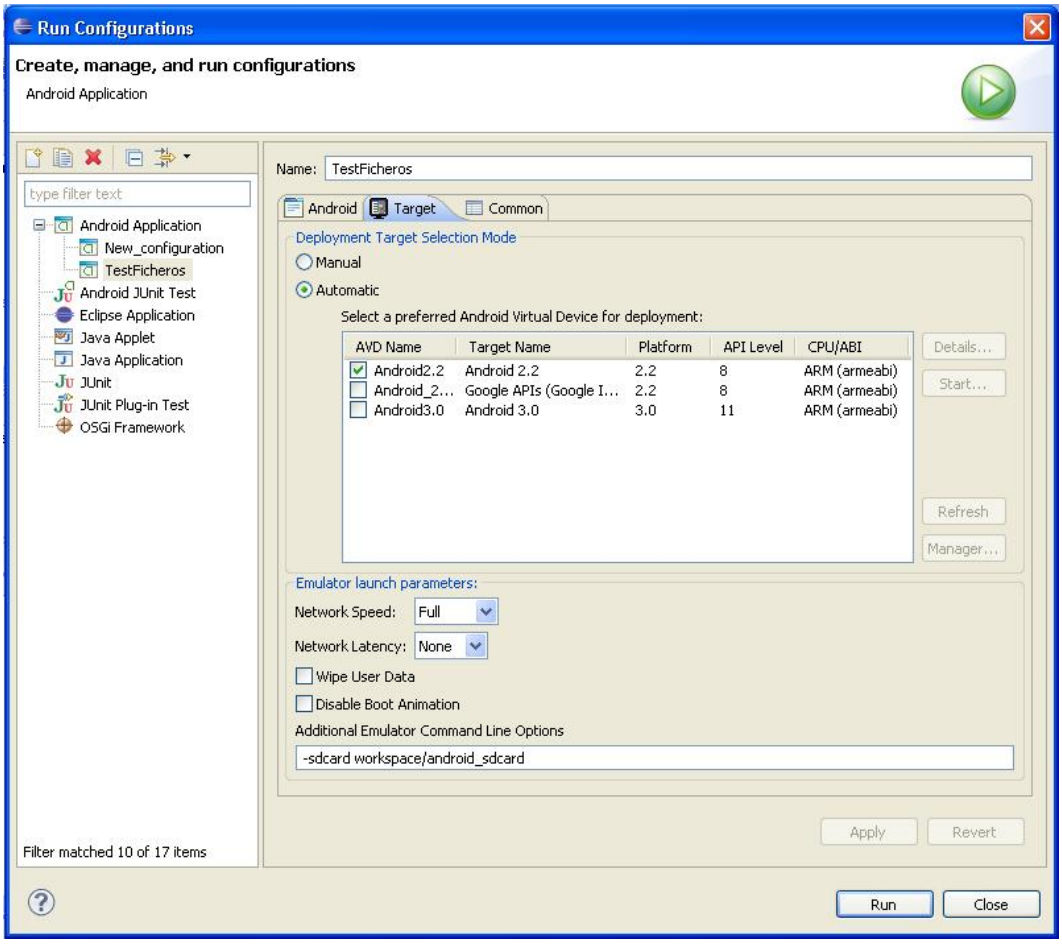

Figura 1.2: Configuración del emulador para el uso de ficheros externos

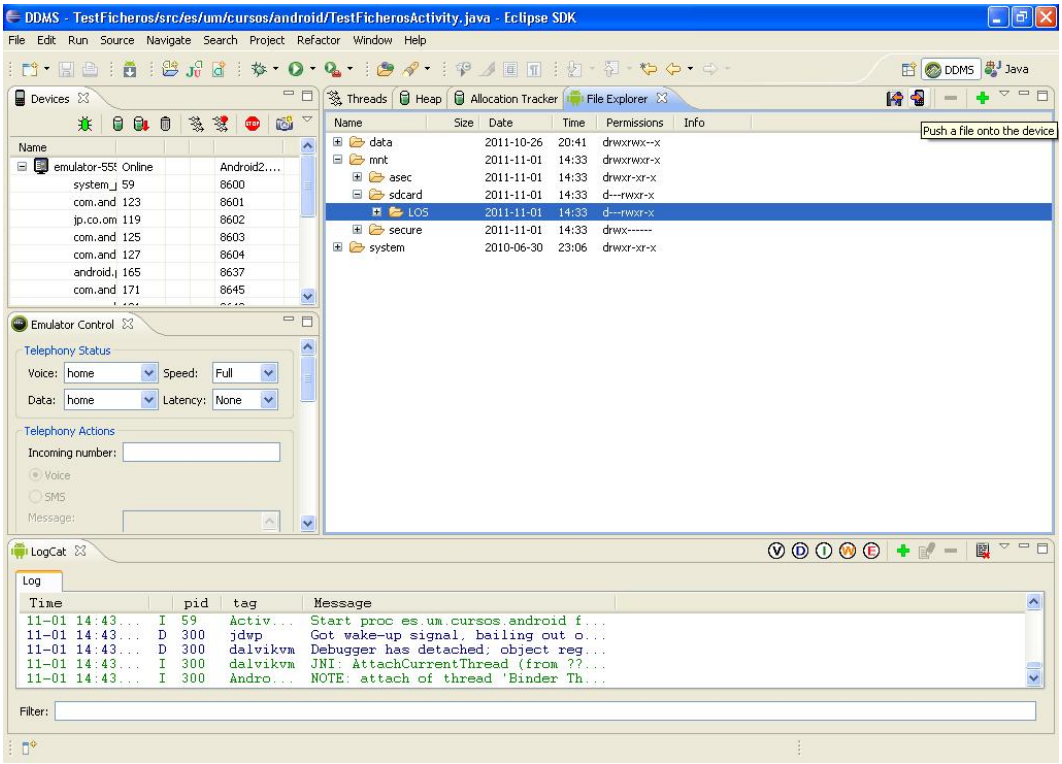

Figura 1.3: Perspectiva DDMS de Eclipse: Tarjeta SD

1.2. Manejo de Bases de Datos

En este apartado vamos a estudiar los mecanismos que Android ofrece a la hora de interactuar con sistemas de almacenamiento y consulta de datos estructurados. Tenemos dos enfoques fundamentales:

- A través de Bases de Datos `SQLite`
- A través de Content Providers [4]

Las bases de datos `SQLite` se utilizan fundamentalmente como método de almacenamiento de los datos de nuestra aplicación. En cambio, los Content Providers están pensados para facilitar la tarea de hacer visibles esos datos que maneja nuestra aplicación a otras aplicaciones, y de forma recíproca, permitir la consulta de datos publicados por terceros desde nuestra aplicación. La explicación de los Content Providers queda fuera del alcance de este curso.

1.2.1. Bases de datos `SQLite`

`SQLite` es un motor de bases de datos que cada día gana más adeptos debido fundamentalmente a que no necesita un servidor, su configuración es extremadamente simple y su tamaño muy pequeño. Además de todo esto, es de código abierto.

Android incorpora de serie una herramienta muy potente para crear y gestionar bases de datos `SQLite`. Esa herramienta viene definida a través de la clase `SQLiteOpenHelper` [5] y su función no es otra que la de comportarse como una plantilla sobre la que moldearemos la base de datos que necesitemos en nuestra aplicación.

Por esta razón, para cada aplicación crearemos una clase que herede de `SQLiteOpenHelper` y personalizaremos la misma con nuestras necesidades: una estructura determinada, cambios a efectuar ante determinados eventos, etc.

Crear una clase que herede de la clase `SQLiteOpenHelper` nos obliga a implementar los siguientes métodos: `onCreate(SQLiteBBDD)`, `onUpgrade(SQLiteBBDD, int, int)` y, opcionalmente, `onOpen(SQLiteBBDD)`.

Creación de una base de datos

Vamos a crear una base de datos de alumnos que contenga una única tabla "`Alumnos`" que, inicialmente, almacenará únicamente el nombre y apellidos de cada alumno.

Para ello creamos una clase `AlumnosBBDD` que herede de `SQLiteOpenHelper` e implemente sus métodos abstractos. Concretamente, el método `onCreate()`, que está pensado para ejecutar aquellas instrucciones que permitan la creación de la estructura de nuestra base de datos, tendrá el siguiente aspecto.

```
private String sqlCreate = "CREATE TABLE Alumnos (nombre TEXT, apellidos TEXT)";

public void onCreate(SQLiteDatabase db) {
  db.execSQL(sqlCreate);
}
```

De esta manera, en el momento de la creación de la base de datos de alumnos, se creará a su vez una única tabla que contendrá los datos antes mencionados.

Por otra parte, el constructor de esta clase deberá llamar a su vez al constructor de la clase padre, teniendo finalmente el siguiente aspecto.

[4] http://developer.android.com/guide/topics/providers/content-providers.html
[5] http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

```
public AlumnosBBDD(Context context, String name, CursorFactory factory, int version) {
  super(context, name, factory, version);
}
```

Una vez que tengamos un objeto de la clase `AlumnosBBD`, podremos recuperar la base de datos asociada, representada mediante la clase `SQLiteDatabase` [6], a través de los métodos heredados `getReadableDatabase()` y `getWritableDatabase()`, en función de si necesitamos únicamente consultar la base de datos, o si además necesitamos realizar cambios o modificaciones sobre la misma.

Así, en nuestra aplicación de prueba, a la que llamaremos `TestSQLite`, deberemos indicar en el método onCreate() de la actividad `TestSQLiteActivity`, el siguiente código para la construcción de nuestra primera base de datos.

```
AlumnosBBDD alumnosBBDD = new AlumnosBBDD(this, "dbAlumnos", null, 1);
SQLiteDatabase db = alumnosBBDD.getWritableDatabase();
```

Bases de datos en el emulador

Tras ejecutar la aplicación `TestSQLite`, con el único contenido descrito hasta ahora, podemos comprobar que la base de datos efectivamente se ha creado en el emulador de Android, consultando la perspectiva DDMS.

En este caso navegamos hasta la ruta `/data/data/es.cursos.um.android/databases` y comprobamos que la base de datos `dbAlumnos` se ha creado con éxito, tal y como se observa en la figura 1.4.

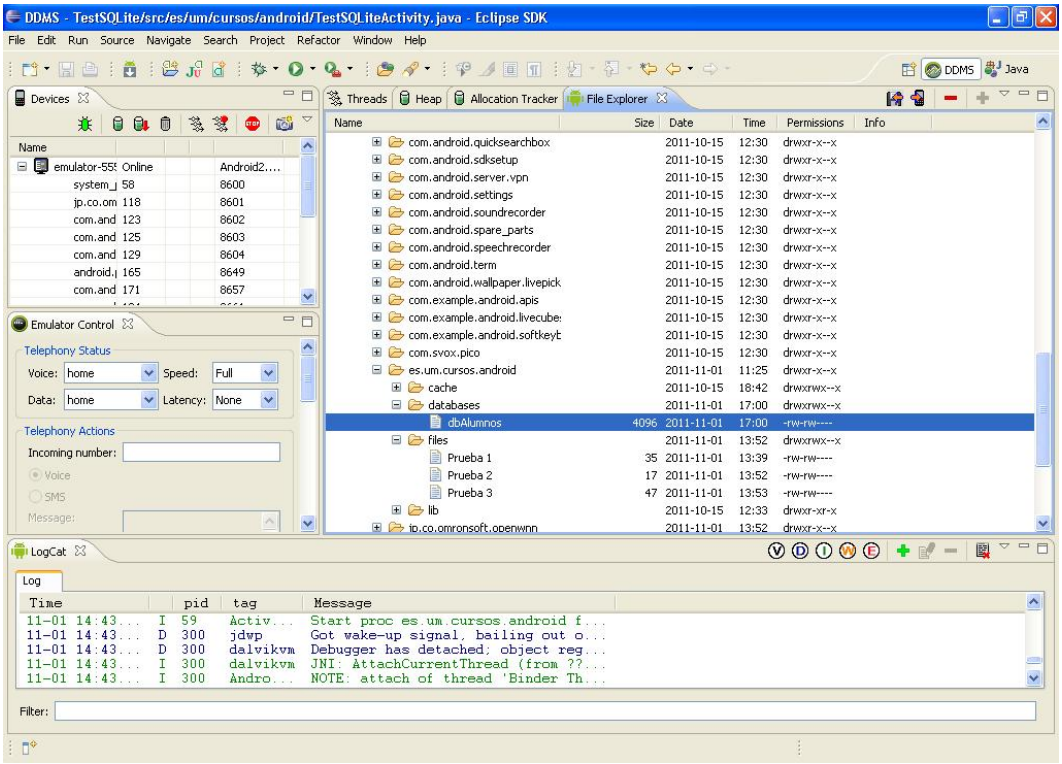

Figura 1.4: Perspectiva DDMS de Eclipse: Bases de datos

[6] http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

Actualización de una base de datos

Por otra parte, Android nos permite a la hora de diseñar una base de datos que la estructura de la misma pueda cambiar en caso de que nos surja la necesidad en un momento posterior de nuestro proyecto.

Dicho de otra manera, Android nos permite tener distintas versiones de una misma base de datos y el método `onUpgrade()` se encargará de actualizar la base de datos en caso de que se produzca una migración de una a otra.

Dicho método `onUpgrade()` se encarga por tanto de hacer los cambios oportunos automáticamente cuando intentemos abrir una versión concreta de la base de datos que aún no existe. En nuestro caso de la base de datos de alumnos, por ejemplo, podríamos tener una nueva versión de la misma que, en la tabla "`Alumnos`", además del nombre y apellidos, incluyera la edad y el correo electrónico.

Una vez que conocemos esto, podemos explicar que la creación de un objeto de la clase `SQLiteOpenHelper` tiene las siguientes consecuencias.

- Si la base de datos ya existe y su versión actual coincide con la indicada en el constructor simplemente se realizará la conexión con la misma.
- Si la base de datos existe pero su versión actual es anterior a la indicada en el constructor, se llamará automáticamente al método `onUpgrade()` para convertir la base de datos a la nueva versión y se conectará con la base de datos convertida.
- Si la base de datos no existe, entonces se llamará automáticamente al método `onCreate()` para crearla y se conectará con la base de datos creada con el nombre que especifiquemos en el constructor.

Inserción, actualización y eliminación de registros

La clase `SQLiteDatabase` tiene un método `execSQL()` que nos permite ejecutar sentencias SQL que no devuelven resultados (`INSERT`, `UPDATE`, `CREATE`, `DELETE`...).

Algunos ejemplos de su uso en nuestra aplicación podría ser los siguientes.

```
//Inserción de registros en la tabla Alumnos
db.execSQL("INSERT INTO Alumnos (nombre, apellidos) VALUES ('Juan', 'Lopez')");
db.execSQL("INSERT INTO Alumnos (nombre, apellidos) VALUES ('Pedro', 'Martinez')");
db.execSQL("INSERT INTO Alumnos (nombre, apellidos) VALUES ('Luis', 'Gonzalez')");

//Actualización de registros de la tabla Alumnos
db.execSQL("UPDATE Alumnos SET nombre='Manuel' WHERE apellidos='Martinez'");

//Eliminación de un registro de la tabla Alumnos
db.execSQL("DELETE FROM Alumnos WHERE nombre='Juan'");
```

Una alternativa a la ejecución de comandos SQL directamente mediante el método `execSQL()`, es el empleo de los siguientes métodos de la clase `SQLiteDatabase`.

```
public long insert (String table, String nullColumnHack, ContentValues values);
public int update (String table, ContentValues values, String whereClause,
                                                        String[] whereArgs);
public int delete (String table, String whereClause, String[] whereArgs);
```

Nótese que los métodos `insert()` y `update()` usan un objeto de tipo `ContentValues` [7] que representa una colección de tipo *hash*, manejando pares clave-valor.

[7] http://developer.android.com/reference/android/content/ContentValues.html

Consultas a una base de datos

Finalmente, para realizar consultas a nuestra base de datos nos encontramos con dos opciones. La primera de ellas, haciendo uso del método `rawQuery()` de la clase `SQLiteDatabase`.

```
public Cursor rawQuery (String sql, String[] selectionArgs);
```

El resultado se devuelve en una variable de tipo `Cursor` [8] que es una estructura pensada para almacenar los resultados de las consultas, similar a una lista y que se puede recorrer fácilmente.

La segunda opción es haciendo uso del método `query()` de la misma clase, que nos permitirá especificar de una forma más clara todo lo necesario para completar la consulta.

```
public Cursor query (String table, String[] columns, String selection,
String[] selectionArgs, String groupBy, String having, String orderBy, String limit);
```

Para recorrer los resultados devueltos en la variable `Cursor`, tenemos a nuestra disposición varios métodos pero destacamos dos que recorren el mismo de forma secuencial y en orden natural.

- `moveToFirst()`: mueve el puntero del cursor al primer registro devuelto.
- `moveToNext()`: mueve el puntero del cursor al siguiente registro devuelto.

Los métodos `moveToFirst()` y `moveToNext()` devuelven `true` en caso de haber realizado el movimiento correspondiente del puntero sin errores, es decir, siempre que exista un primer registro o un registro siguiente, respectivamente.

Así, el siguiente código nos permitiría consultar todos los registros de la base de datos, para comprobar que efectivamente, las operaciones de inserción, modificación y eliminación realizadas anteriormente, se han llevado a cabo con éxito.

```
String sqlQuery = "SELECT * FROM Alumnos;";
Cursor cursor = db.rawQuery(sqlQuery, null);
if (cursor.moveToFirst()) {
  do {
    String nombre = cursor.getString(0);
    String apellidos = cursor.getString(1);
    Toast.makeText(this, nombre+" "+apellidos, Toast.LENGTH_SHORT).show();
  } while (cursor.moveToNext());
}
```

Actividad propuesta II.3. *Desarrollar la aplicación `TestSQLite` que cree una base de datos conteniendo una única tabla "`Alumnos`" para almacenar el nombre y apellidos de los alumnos, inserte varios registros, actualice alguno/s de ellos, elimine alguno/s de ellos y muestre, mediante una consulta SQL, el contenido final de dicha tabla.*

[8] http://developer.android.com/reference/android/database/Cursor.html

Tema 2

SMS, MMS y Telefonía

*E*ste tema explicará de qué manera es posible enviar y recibir mensajes SMS y MMS mediante una aplicación Android. También se verá el manejo de algunas funciones de telefonía a través de Android.

2.1. Telefonía

2.1.1. Realización de una llamada

La forma mas fácil y directa de iniciar una llamada telefónica a través de una aplicación Android es mediante el uso de *Intents*. Hacer que aparezca el teclado con el número de teléfono que nosotros queramos es tan sencillo como hacer lo siguiente.

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:600123456"));
startActivity(intent);
```

Con esto, conseguimos que la aplicación llame al teclado y lo prepare para que muestre el teléfono especificado. Nótese que la llamada no se inicia automáticamente, sino que se requiere la intervención y aceptación por parte del usuario.

2.1.2. Lectura de las propiedades del dispositivo

Cuando desarrollemos nuestras aplicaciones, es importante que tengamos en cuenta que las mismas pueden ejecutarse en terminales cuya configuración de red pueda cambiar (por ejemplo por el hecho de que el usuario viaje a otro país, o porque en un pais la configuración de red sea distinta de otro) de forma que sea importante conocer las características del mismo.

En ese sentido, Android nos ofrece dentro del paquete `android.telephony` una clase llamada `TelephonyManager` [1] que nos permite obtener información sobre las características del terminal con el que trabajamos.

Para recuperar el objeto `TelephonyManager` correspondiente a nuestra actividad, deberemos utilizar el siguiente código.

```
TelephonyManager telephonyManager =
            (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

Para que nuestra aplicación pueda leer las propiedades del dispositivo sobre el que se ejecuta, es necesario que cuente con el permiso `android.permission.READ_PHONE_STATE`.

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

[1] http://developer.android.com/reference/android/telephony/TelephonyManager.html

Lectura de las propiedades del teléfono

Mediante el `TelephonyManager` podemos obtener el tipo de teléfono (GSM o CDMA), el identificador único (IMEI o MEID), la versión del software del dispositivo y el número de teléfono.

El siguiente código nos permite leer estas propiedades.

```
int phoneType = telephonyManager.getPhoneType();
switch (phoneType) {
  case (TelephonyManager.PHONE_TYPE_CDMA): break;
  case (TelephonyManager.PHONE_TYPE_GSM) : break;
  case (TelephonyManager.PHONE_TYPE_NONE): break;
}
String deviceId = telephonyManager.getDeviceId();
String softwareVersion = telephonyManager.getDeviceSoftwareVersion();
String phoneNumber = telephonyManager.getLine1Number();
```

Lectura de las propiedades de la conexión de datos

Usando los métodos `getDataState()` y `getDataActivity()` de la clase `TelephonyManager` podemos conocer el estado de la conexión de datos, así como del tráfico de datos.

El siguiente código nos permite leer estas propiedades.

```
int dataActivity = telephonyManager.getDataActivity();
int dataState = telephonyManager.getDataState();

switch (dataActivity) {
  case TelephonyManager.DATA_ACTIVITY_IN : break;
  case TelephonyManager.DATA_ACTIVITY_OUT : break;
  case TelephonyManager.DATA_ACTIVITY_INOUT : break;
  case TelephonyManager.DATA_ACTIVITY_NONE : break;
}

switch (dataState) {
  case TelephonyManager.DATA_CONNECTED : break;
  case TelephonyManager.DATA_CONNECTING : break;
  case TelephonyManager.DATA_DISCONNECTED : break;
  case TelephonyManager.DATA_SUSPENDED : break;
}
```

Lectura de las propiedades de la red

Cuando estemos conectados a una red, podemos usar el `TelephonyManager` para conocer los códigos de red y de país (MCC+MNC), el código ISO del país y el tipo de red al que estamos conectados.

El siguiente código nos permite leer estas propiedades.

```
String networkCountry = telephonyManager.getNetworkCountryIso();
String networkOperatorId = telephonyManager.getNetworkOperator();
String networkName = telephonyManager.getNetworkOperatorName();
int networkType = telephonyManager.getNetworkType();
switch (networkType) {
  case (TelephonyManager.NETWORK_TYPE_1xRTT)    : break;
  case (TelephonyManager.NETWORK_TYPE_CDMA)     : break;
```

```
   case (TelephonyManager.NETWORK_TYPE_EDGE)    : break;
   case (TelephonyManager.NETWORK_TYPE_EVDO_0)  : break;
   case (TelephonyManager.NETWORK_TYPE_EVDO_A)  : break;
   case (TelephonyManager.NETWORK_TYPE_GPRS)    : break;
   case (TelephonyManager.NETWORK_TYPE_HSDPA)   : break;
   case (TelephonyManager.NETWORK_TYPE_HSPA)    : break;
   case (TelephonyManager.NETWORK_TYPE_HSUPA)   : break;
   case (TelephonyManager.NETWORK_TYPE_UMTS)    : break;
   case (TelephonyManager.NETWORK_TYPE_UNKNOWN) : break;
 }
```

Lectura de las propiedades de la SIM

Por último, se pueden consultar los detalles de la SIM para obtener el código ISO del país, el código del operador y el MCC y MNC del operador.

El siguiente código nos permite leer estas propiedades.

```
int simState = telephonyManager.getSimState();
switch (simState) {
  case (TelephonyManager.SIM_STATE_ABSENT): break;
  case (TelephonyManager.SIM_STATE_NETWORK_LOCKED): break;
  case (TelephonyManager.SIM_STATE_PIN_REQUIRED): break;
  case (TelephonyManager.SIM_STATE_PUK_REQUIRED): break;
  case (TelephonyManager.SIM_STATE_UNKNOWN): break;
  case (TelephonyManager.SIM_STATE_READY): {
    String simCountry = telephonyManager.getSimCountryIso();
    String simOperatorCode = telephonyManager.getSimOperator();
    String simOperatorName = telephonyManager.getSimOperatorName();
    String simSerial = telephonyManager.getSimSerialNumber();
    break;
  }
}
```

2.1.3. Aplicación `TestTelefonia`

Vamos a crear ahora una aplicación llamada `TestTelefonia`, que nos permitirá probar tanto la realización de llamadas a través de Android, como la consulta de propiedades del dispositivo sobre el que se ejecute nuestra aplicación.

Para ello, crearemos una aplicación con un *layout* `main` como el que se observa en la figura 2.1. Como se puede observar, la aplicación cuenta con un `EditText` donde podremos introducir el número de teléfono al que queramos llamar, así como un botón para iniciar dicha llamada.

Por otra parte, también contará con un `RadioGroup` que nos permitirá seleccionar el tipo de propiedad que queremos consultar (del teléfono, de los datos, de la red o de la SIM), además de un botón para efectivamente consultar dichas propiedades y mostrarlas finalmente en otro `EditText` en la parte inferior de la pantalla.

Actividad propuesta II.4. *Desarrollar y probar la aplicación `TestTelefonia`, tal y como se ha descrito anteriormente.*

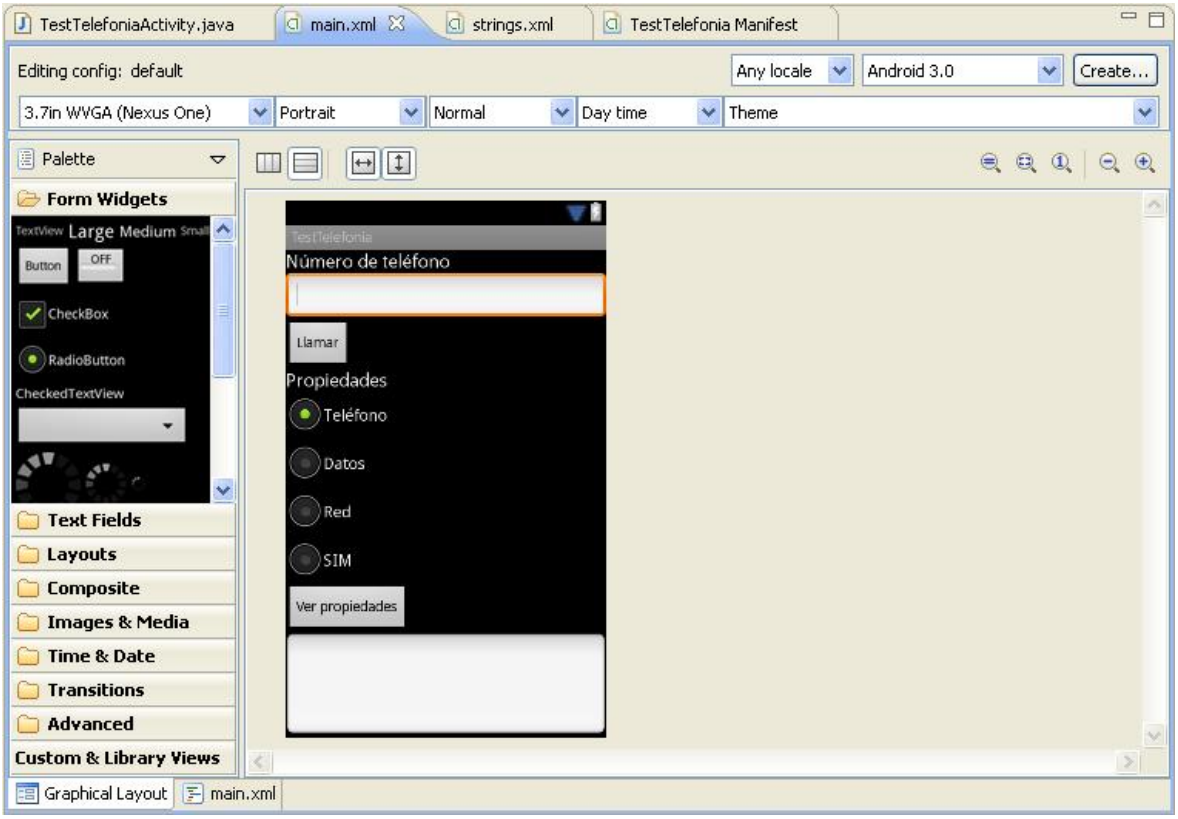

Figura 2.1: *Layout* `main` de la aplicación `TestTelefonia`

Figura 2.2: Aplicación `TestTelefonia`

2.2. Mensajería SMS

2.2.1. Mensajes SMS a través de *intents*

Del mismo modo que hacíamos para iniciar una llamada telefónica, Android nos ofrece la posibilidad de enviar un mensaje SMS, mediante un *intent* tan sencillo como el que se muestra a continuación.

```
Intent smsIntent = new Intent(Intent.ACTION_SENDTO, Uri.parse("sms:600123456"));
smsIntent.putExtra("sms_body",
          "¡Hola! Nos vemos esta noche junto a la puerta del bar de siempre.");
startActivity(smsIntent);
```

La figura 2.3 muestra un ejemplo de aplicación que hace uso de ese código.

Figura 2.3: Mensajes SMS a través de *intents*

2.2.2. Mensajes SMS a través de la clase `SmsManager`

Inicialmente, y hasta la versión 1.6 de Android, la clase `SmsManager` era la encargada de manejar los mensajes a bajo nivel en las aplicaciones. Se encontraba dentro del paquete `android.telephony.gsm` pero dado que esta clase no daba soporte a las infraestructuras de red CDMA ampliamente utilizada en Norte América se sustituyó por la clase que se encuentra dentro del paquete `android.telephony.SmsManager` [2] que sí ofrece ese soporte.

Siempre que trabajemos con el gestor de mensajería SMS tendremos que solicitar los permisos pertinentes en el manifiesto, a saber, `SEND_SMS` y `RECEIVE_SMS`.

Para obtener una referencia al gestor de SMS debemos utilizar el método estático de la clase `SmsManager.getDefault()`.

[2] http://developer.android.com/reference/android/telephony/SmsManager.html

Mandar un mensaje de texto utilizando el gestor de SMS es muy sencillo. Tan sólo necesitamos hacer uso del método `sendTextMessage()` especificando correctamente los siguientes parámetros:

- `destinationAddress`: Número de teléfono del destinatario.
- `scAddress`: Centro de Servicio alternativo al de nuestra compañía telefónica para tramitar los mensajes.
- `text`: El texto del mensaje en cuestión.
- `sentIntent`: Permite especificar un *Pending Intent* que quede a la espera de la recepción de un *Ok* o un *Failure* en el envío correcto del mensaje.
- `deliveryIntent`: Similar al parámetro anterior pero en este caso para confirmar la entrega del mensaje.

Para rastrear la correcta transmisión y entrega de mensajes SMS salientes, deberemos implementar y registrar *Broadcast Receivers* que escuchen a las acciones indicadas a la hora de crear los *Pending Intents* que se le pasan al método `sendTextMessage()`. El siguiente código permite enviar un mensaje SMS mediante el `SmsManager`, y muestra un mensaje emergente indicando el éxito o fallo a la hora de enviar dicho mensaje, así como una notificación cuando el mensaje ha sido correctamente entregado al receptor.

```
SmsManager smsManager = SmsManager.getDefault();

String SENT_SMS_ACTION = "SENT_SMS_ACTION";
String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";

String sendTo = "600123456";
String myMessage = "Ejemplo de mensaje creado con la clase SmsManager";

Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI =
    PendingIntent.getBroadcast(getApplicationContext(), 0, sentIntent, 0);

Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI =
    PendingIntent.getBroadcast(getApplicationContext(), 0, deliveryIntent, 0);

registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context _context, Intent _intent) {
    switch (getResultCode()) {
      case Activity.RESULT_OK:
        Toast.makeText(_context, "Mensaje enviado con éxito", Toast.LENGTH_SHORT).show();
        break;
      case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
        Toast.makeText(_context, "Error al enviar el mensaje", Toast.LENGTH_SHORT).show();
        break;
      case SmsManager.RESULT_ERROR_RADIO_OFF:
        Toast.makeText(_context, "El teléfono está desactivado", Toast.LENGTH_SHORT).show()
        break;
      case SmsManager.RESULT_ERROR_NULL_PDU:
```

```
        Toast.makeText(_context, "Error en el PDU", Toast.LENGTH_SHORT).show();
        break;
    }
  }
}, new IntentFilter(SENT_SMS_ACTION));

registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context _context, Intent _intent) {
    Toast.makeText(_context, "Mensaje recibido con éxito", Toast.LENGTH_SHORT).show();
  }
}, new IntentFilter(DELIVERED_SMS_ACTION));

smsManager.sendTextMessage(sendTo, null, myMessage, sentPI, deliverPI);
```

2.2.3. Aplicación `TestSMS`

Vamos ahora a desarrollar una aplicación denominada `TestSMS` que nos servirá para mostrar la funcionalidad de la mensajería SMS descrita hasta ahora. Será muy sencilla y contará, tal y como se puede comprobar en la figura 2.4, únicamente con un campo para introducir el destinatario del mensaje, otro campo para escribir el mensaje en sí, y un botón para enviar el mensaje introducido al destinatario indicado.

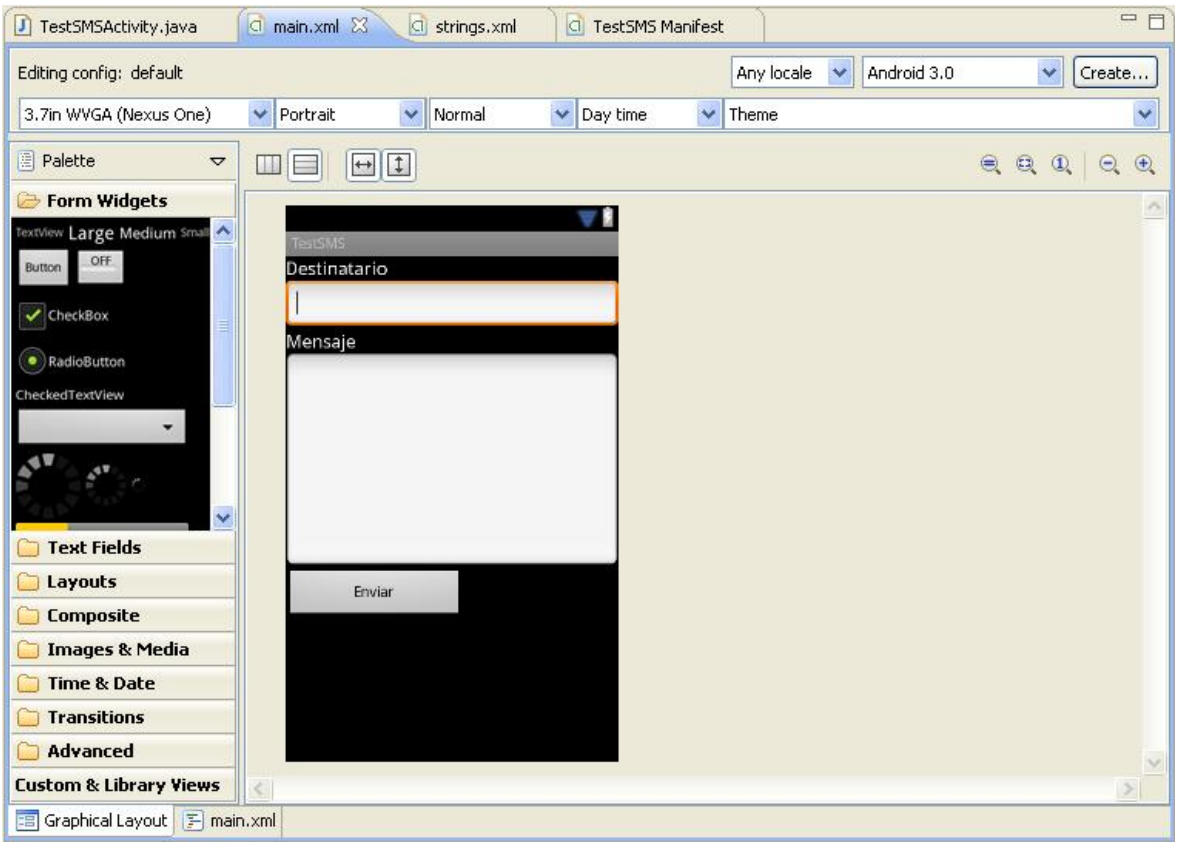

Figura 2.4: *Layout* `main` de la aplicación `TestSMS`

La figura 2.5 muestra la ejecución de la aplicación `TestSMS`.

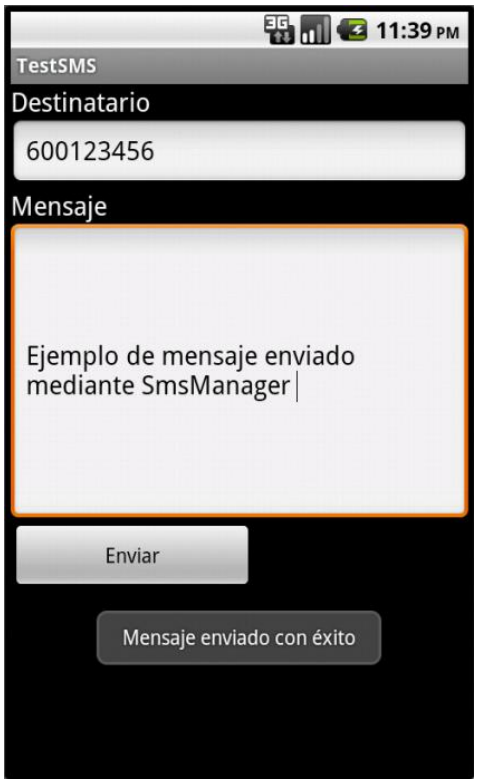

Figura 2.5: Aplicación `TestSMS`

Actividad propuesta II.5. *Desarrollar y probar la aplicación `TestSMS`, tal y como se ha descrito anteriormente.*

2.3. Mensajería MMS

2.3.1. Mensajes MMS a través de *intents*

Los mensajes multimedia o MMS nos permiten enviar contenido multimedia (audio, imágenes, vídeo, etc) en nuestros mensajes de texto tradicionales. Al igual que con los SMS y las llamadas telefónicas, la forma de enviar mensajes multimedia en Android es muy sencilla, mediante el uso de *intents*.

Para ello deberemos adjuntar el contenido multimedia mediante su correspondiente URI, tal y como se muestra a continuación.

```
Uri uri = Uri.parse("content://mnt/sdcard/download/bici");
Intent mmsIntent = new Intent(Intent.ACTION_SEND, uri);
mmsIntent.putExtra("sms_body", "Mi nueva bicicleta");
mmsIntent.putExtra("address", "600123456");
mmsIntent.putExtra(Intent.EXTRA_STREAM, uri);
mmsIntent.setType("image/png");
startActivity(mmsIntent);
```

Tema 3

Conexiones HTTP y el WebBrowser

*E*n este tema se dará al alumno los conocimientos necesarios para establecer conexiones HTTP mediante el lenguaje Android, al tiempo que se explicará cómo visualizar páginas web en un navegador en este tipo de dispositivos, usando Android.

3.1. Recursos de Internet

A la hora de desarrollar una aplicación en Android con acceso a recursos en la web, uno podría plantearse si realmente merece la pena el esfuerzo, o si sería simplemente más sencillo acceder a ese recurso (página web, fichero, etc), directamente desde el navegador del dispositivo móvil Android.

Sin embargo, existe un buen número de motivos por los cuales puede interesar desarrollar un cliente nativo en Android con acceso a la web, antes que acceder directamente desde el navegador del dispositivo Android. Algunos de estos motivos son, por ejemplo:

- **Ancho de banda**. Algunos recursos estáticos, tales como imágenes, música o vídeo pueden suponer un consumo de ancho de banda elevado en dispositivos con limitaciones en el acceso a Internet. Mediante la creación de una aplicación nativa en Android, es posible controlar el consumo de ancho de banda.
- **Caching**. Una solución basada en navegador web con una mala conexión a Internet puede resultar en una disponibilidad intermitente de la aplicación. Sin embargo, una aplicación nativa es capaz de cachear ciertos datos para proporcionar tanta funcionalidad como sea posible sin necesidad de una conexión constante a Internet.
- **Propiedades nativas**. Los dispositivos Android son mucho más que meros navegadores web móviles. Incluyen servicios basados en localización, notificaciones, cámara, acelerómetros, etc. La creación de una aplicación nativa permite combinar los datos disponibles online con las capacidades de hardware disponibles en el dispositivo.

Los dispositivos móviles ofrecen hoy en día diferentes alternativas para acceder a Internet. Desde un punto de vista más amplio, los dispositivos Android proporcionan dos técnicas principales para la conexión a Internet, ambas de forma transparente a la capa de aplicación:

- **Internet móvil**. Las operadoras que ofrecen tarifas de datos a sus clientes, proporcionan acceso a Internet a través de conexiones GPRS, EDGE y 3G.
- **WiFi**. Una conexión Wi-Fi asegura a menudo una mejor calidad en la conexión de datos a Internet.

3.2. Conexiones HTTP

3.2.1. Permiso para la conexión a Internet

En primer lugar, cualquier aplicación de Android que desee realizar conexiones a Internet, necesita especificar el correspondiente permiso en su fichero de manifiesto. Este permiso es:

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

3.2.2. Aplicación `HTTPImageDownloader`

A continuación vamos a describir los pasos necesarios para desarrollar una aplicación en Android, a la que llamaremos `HTTPImageDownloader`, que descargará un recurso de Internet, concretamente una imagen, y la mostrará en la pantalla del dispositivo.

En primer lugar, como hemos comentado anteriormente, deberemos especificar en el fichero de manifiesto el permiso concreto para que nuestra aplicación pueda acceder a Internet, tal y como se observa en la figura 3.1.

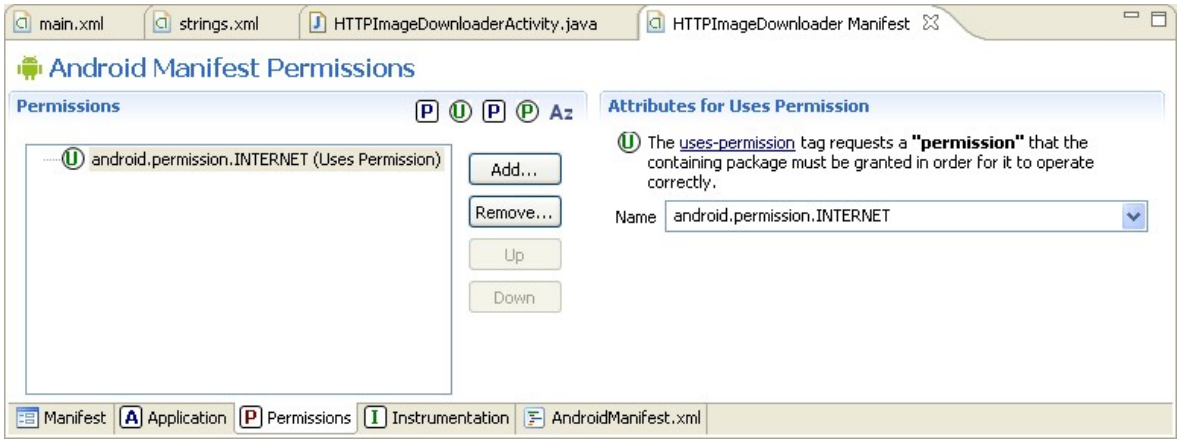

Figura 3.1: Permiso `INTERNET` en el manifiesto de `HTTPImageDownloader`

A continuación editaremos el *layout* `main` de la aplicación `HTTPImageDownloader`, tal y como se observa en la figura 3.2, de tal forma que contendrá un `TextView` donde introducir la URL de la imagen que queremos descargar, un `Button` para iniciar dicha descarga, y por último un `ImageView` donde mostraremos la imagen descargada.

Por último, para dotar de funcionalidad al botón de descarga, deberemos añadir el siguiente código en la clase `HTTPImageDownloaderActivity`:

```
public void onCreate(Bundle savedInstanceState) {
    ...
    urlTextView = (TextView) findViewById(R.id.urlTextView);
    descargarButton = (Button) findViewById(R.id.descargarButton);
    imageView = (ImageView) findViewById(R.id.imageView);

    descargarButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            downloadImage();
        }
    });
}
```

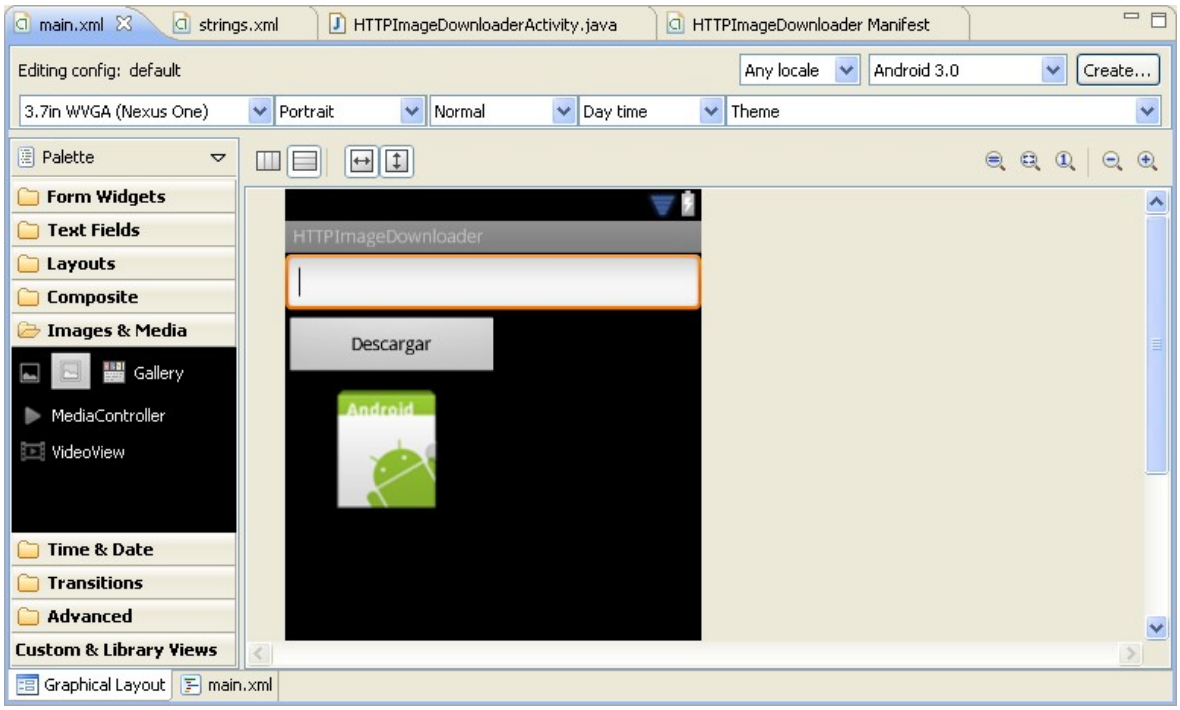

Figura 3.2: *Layout* `main` de la aplicación `HTTPImageDownloader`

```
public void downloadImage() {
    try {
        URL url = new URL(urlTextView.getText().toString());
        URLConnection connection = url.openConnection();
        HttpURLConnection httpConnection = (HttpURLConnection) connection;
        int responseCode = httpConnection.getResponseCode();
        if (responseCode == HttpURLConnection.HTTP_OK) {
            InputStream in = httpConnection.getInputStream();
            BufferedInputStream buf = new BufferedInputStream(in);
            Bitmap bMap = BitmapFactory.decodeStream(buf);
            imageView.setImageBitmap(bMap);
            if (in != null)
                in.close();
            if (buf != null)
                buf.close();
        }
    } catch (MalformedURLException e) {
        Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT);
    } catch (IOException e) {
        Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT);
    }
}
```

Como se puede observar, las principales clases para abrir una conexión HTTP y descargar un recurso son `URL`[1], `URLConnection`[2] y `HttpURLConnection`[3].

[1] http://developer.android.com/reference/java/net/URL.html
[2] http://developer.android.com/reference/java/net/URLConnection.html
[3] http://developer.android.com/reference/java/net/HttpURLConnection.html

El resultado de ejecutar la aplicación `HTTPImageDownloader` se puede observar en la figura 3.3.

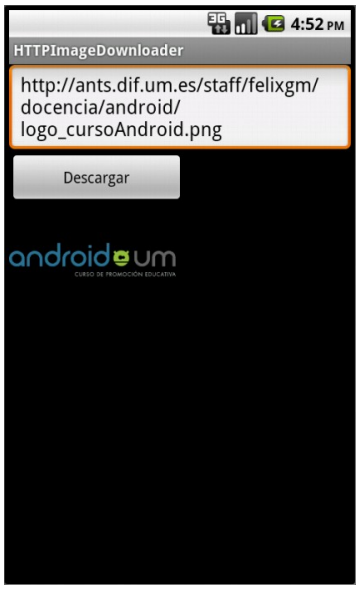

Figura 3.3: Aplicación `HTTPImageDownloader`

Actividad propuesta II.6. *Desarrollar la aplicación `HTTPImageDownloader` tal y como se ha explicado anteriormente.*

Actividad propuesta II.7. *Desarrollar una aplicación similar a `HTTPImageDownloader` llamada `HTTPResourceDownloader`, que permita descargar un recurso de Internet (imagen, texto, web, etc) y almacenarlo en el dispositivo móvil.*

3.3. Aplicación `WebBrowser`

Por otra parte, si lo que nosotros deseamos es que nuestra aplicación cuente con un navegador web integrado, podemos hacer uso de la vista `WebView`[4].

Para mostrar su funcionalidad vamos a desarrollar una nueva aplicación llamada `WebBrowser`. Esta aplicación contará, tal y como se observa en la figura 3.4, de un `TextView` donde introducir la URL de la página web que deseemos visualizar, un `ImageButton` que al pulsarlo nos permitirá visualizar la web que hayamos indicado, y por último un `WebView`, que contendrá la página web en sí que queramos visualizar.

> **Nota.-** Esta aplicación también requiere el permiso `android.permission.INTERNET` en su manifiesto, tal y como se indicó en la sección 3.2.1.

[4] http://developer.android.com/reference/android/webkit/WebView.html

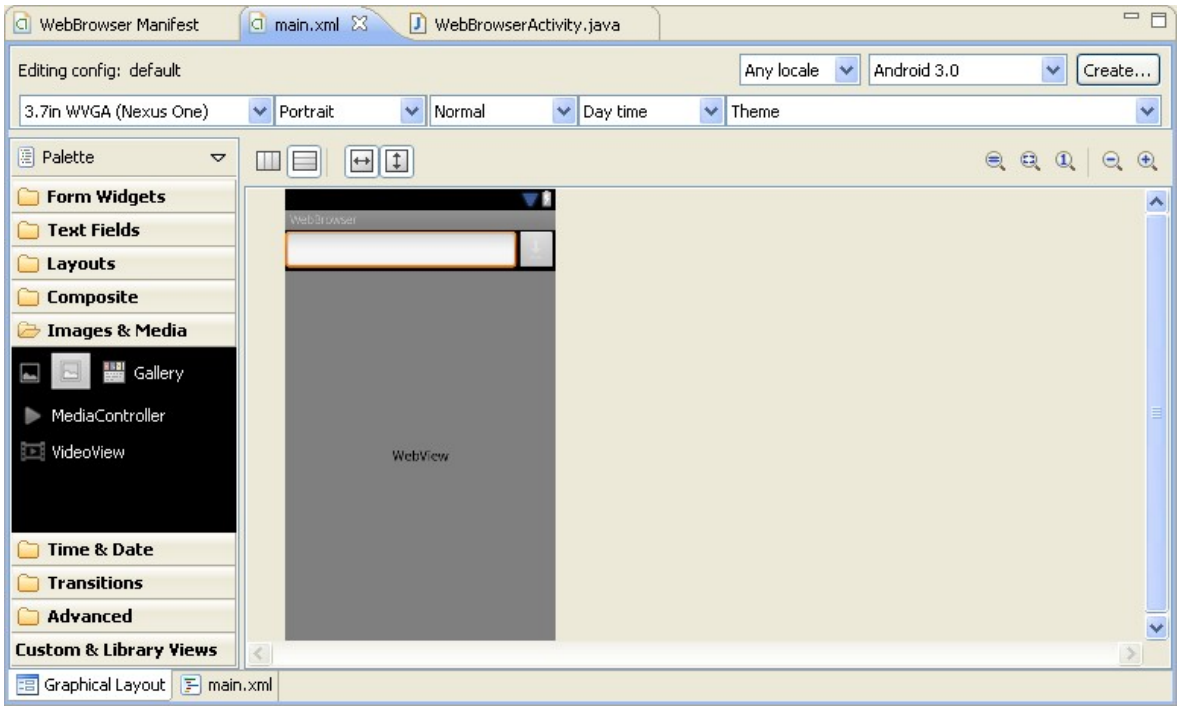

Figura 3.4: *Layout* `main` de la aplicación `WebBrowser`

A continuación, dotaremos de funcionalidad al botón de nuestra aplicación, añadiendo el siguiente código a la clase `WebBrowserActivity`:

```
public void onCreate(Bundle savedInstanceState) {
    ...
    urlTextView = (TextView) findViewById(R.id.urlTextView);
    downloadWebImageButton = (ImageButton) findViewById(R.id.downloadWebImageButton);
    webView = (WebView) findViewById(R.id.webView);

    downloadWebImageButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            downloadWeb();
        }
    });
}

public void downloadWeb() {
    String url = urlTextView.getText().toString();
    webView.loadUrl(url);
}
```

El resultado de ejecutar la aplicación `WebBrowser` puede observarse en la figura 3.5.

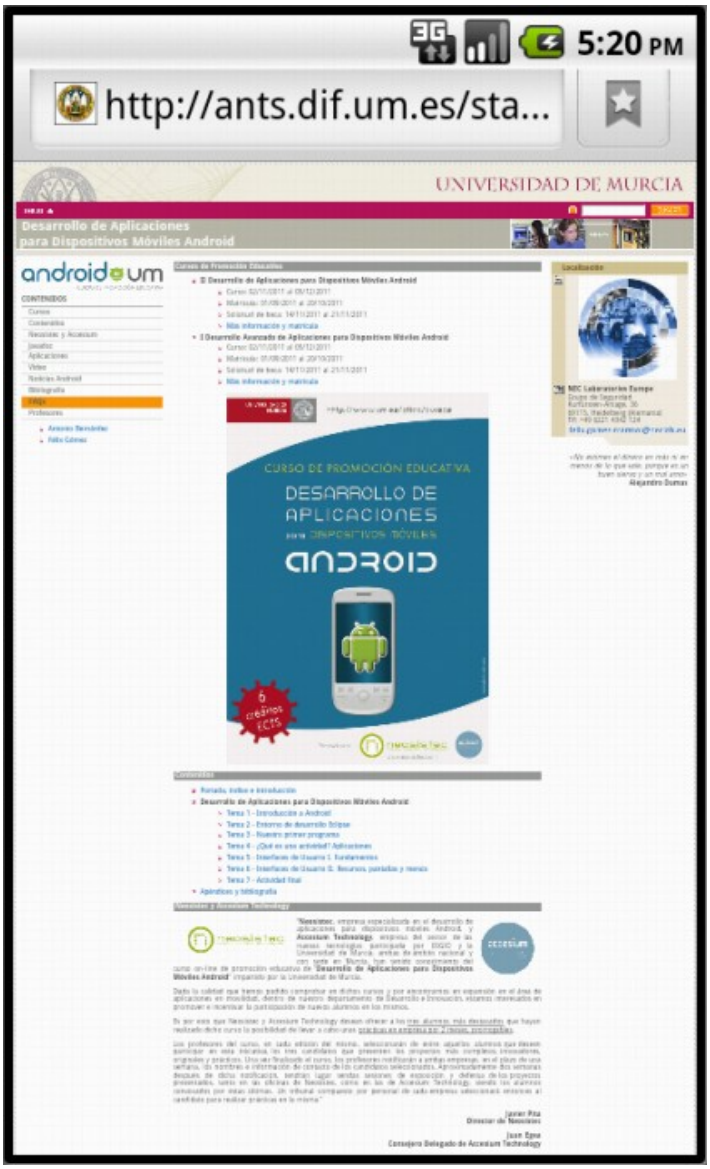

Figura 3.5: Aplicación `WebBrowser`

Actividad propuesta II.8. *Desarrollar la aplicación* `WebBrowser` *tal y como se ha explicado anteriormente.*

Otro método interesante de la clase `WebView` es `void loadData(String data, String mimeType, String encoding)`, que permite cargar, entre otras cosas, contenido html directo, como en el siguiente ejemplo:

```
webView.loadData("<html><body>¡Hola, mundo!</body></html>","text/html","UTF-8");
```

Tema 4

Audio, Vídeo y Cámara

4.1. Introducción

El auge de las tecnologías multimedia ha propiciado que los dispositivos móviles cuenten con los más avanzados dispositivos con los que poder grabar videos, reproducir música o hacer fotografías.

Independientemente del uso más directo que se le puede dar a estas utilidades, están surgiendo de forma cada vez más habitual aplicaciones profesionales que se basan en la utilización de estas características para ofrecer servicios avanzados.

Sin ir más lejos, el Ayuntamiento de Barcelona dispone de un software que permite a los operarios de mantenimiento del Puerto Marítimo de Barcelona capturar los defectos mediante la cámara de su dispositivo móvil y generar un informe en tiempo real que llegará a la persona encargada de su resolución en el momento.

Con ese objetivo en este tema presentamos las diferentes formas de trabajar con estas características. Vamos a ver cómo Android nos presenta siempre un enfoque básico, en donde se utilizan las aplicaciones ya creadas para hacer uso del audio, vídeo o cámara, y cómo, de igual forma, nos ofrece una forma más avanzada con la que poder personalizar nuestras aplicaciones hasta un punto mucho más profundo.

4.2. Audio

4.2.1. Formatos de audio soportados en Android

Android es capaz de reproducir múltiples formatos de audio. De forma general será capaz de reproducir la práctica totalidad de los formatos existentes en la actualidad. La tabla 4.1 muestra los formatos de audio soportados por Android.

Formato	**Tipo de fichero**
AAC LC/LTP	.3gp, .mp4, .m4a, .aac, .ts
HE-AACv1 (AAC+)	
HE-AACv2 (enhanced AAC+)	
AMR-NB/AMR-WB	.3gp
FLAC	.flac
MP3	.mp3
MIDI	.mid, .xmf, .mxmf, .rttl, .rtx, .ota, .imy
Vorbis	.ogg, .mkv
PCM/WAVE	.wav

Tabla 4.1: Formatos de audio soportados en Android

4.2.2. Máquina de estados del Media Player

El reproductor multimedia o Media Player de Android gestiona todo el proceso de inicio y reproducción de un fichero de audio como si fuera una máquina de estados, tal y como se observa en la figura 4.1.

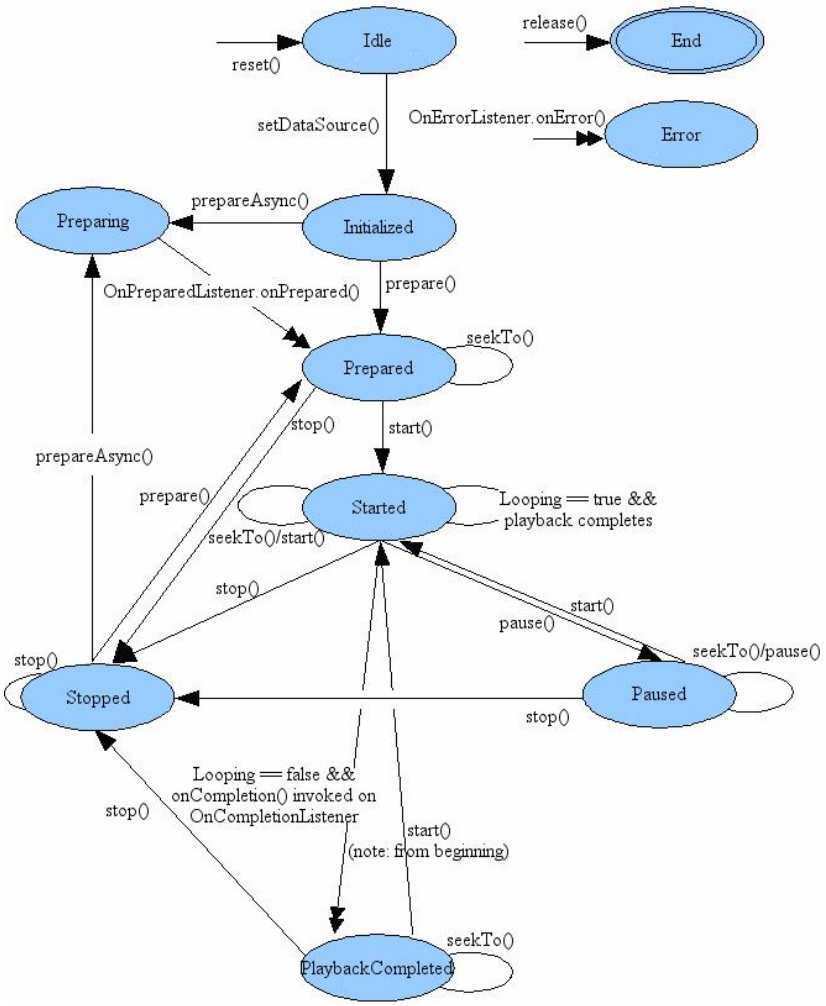

Figura 4.1: Máquina de estados del Media Player de Android

Para el caso de la reproducción del audio, las transiciones a través de la máquina de estados son las siguientes:

1. Inicializamos el Media Player con un archivo a reproducir. (`Idle` → `Initialized`)
2. Preparamos el Media Player para su reproducción. (`Initialized` → `Prepared`)
3. Iniciamos la reproducción (`Prepared` → `Started`)
4. Pausamos o paramos la reproducción. (`Started` → `Stopped` o bien `Started` → `Paused`)
5. Completamos la reproducción. (`Stated` → `Playback Completed`)

El diagrama presenta igualmente posibles transiciones de estados que se pueden producir ante determinados eventos como puedan ser la colocación en un lugar concreto de la reproducción mediante el método `seekTo()` o la reproducción continua del fichero mediante la condición `(Looping == true && playback completes)`.

4.2.3. Clase `MediaPlayer`

A la hora de reproducir un archivo de audio, tenemos dos formas de hacerlo a través de la clase `MediaPlayer`[1].

La primera forma es hacer uso del método `create()`. Este método es un método sobrecargado cuyas cabeceras podemos ver a continuación y cuyo cometido es el de preparar la fuente de origen del audio y de dejar listo el reproductor para su ejecución.

```
public static MediaPlayer create (Context context, Uri uri, SurfaceHolder holder)
public static MediaPlayer create (Context context, int resid)
public static MediaPlayer create (Context context, Uri uri)
```

El método aceptará por tanto una ruta de un archivo, un proveedor de contenido URI, una dirección HTTP, o un descriptor de archivo. Además, el método `create()` invoca cuando se ejecuta al método `prepare()`, necesario para la correcta ejecución del Media Player.

Otra alternativa para iniciar el reproductor es ejecutar el método `setDataSource()` a partir de una instancia que tengamos creada de la clase `MediaPlayer`.

4.2.4. Aplicación `TestAudio`

Para probar la funcionalidad del audio, vamos a desarrollar una aplicación llamada `TestAudio`. Tal y como se observa en la figura 4.2, esta aplicación contará simplemente con tres botones que nos permitirán comenzar, pausar y terminar la reproducción de un fichero de audio.

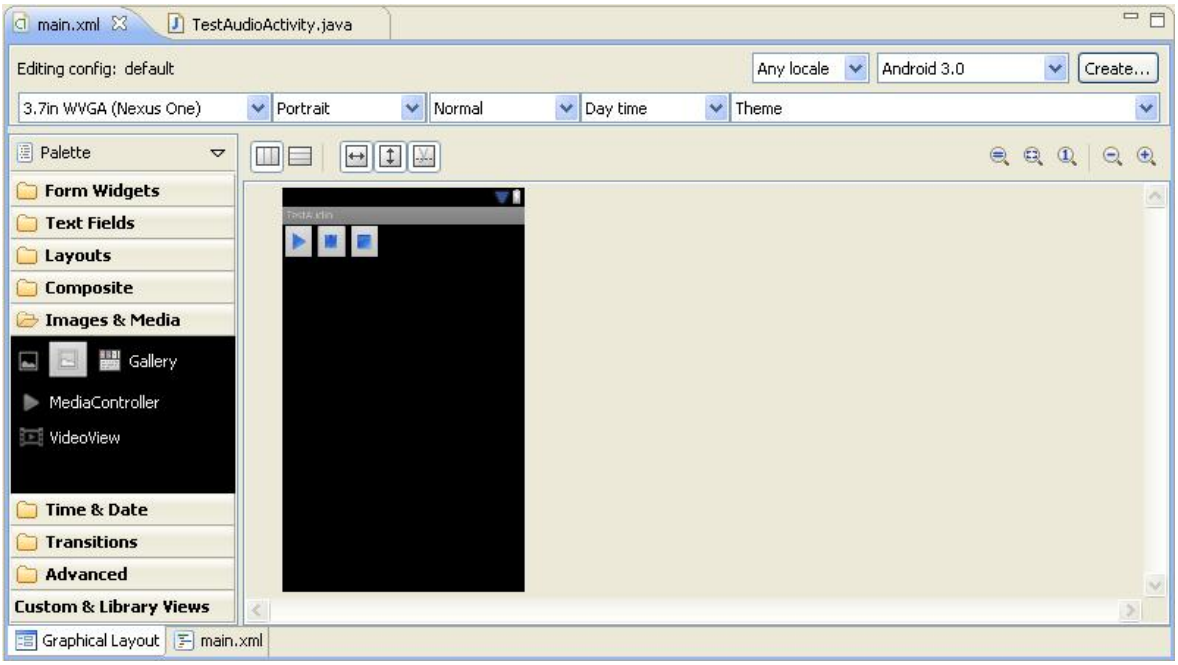

Figura 4.2: *Layout* `main` de la aplicación `TestAudio`

A continuación deberemos dotar de funcionalidad a cada uno de esos botones (mediante llamadas a los métodos `start()`, `pause()`, `stop()` y `release()` de la clase `MediaPlayer`, respectivamente). Pero antes tendremos que inicializar el objeto de la clase `MediaPlayer` que nos permitirá realizar las acciones antes mencionadas.

Así por lo tanto, la actividad `TestAudioActivity` tendrá el siguiente aspecto.

[1] http://developer.android.com/reference/android/media/MediaPlayer.html

```
private MediaPlayer mediaPlayer;

public void onCreate(Bundle savedInstanceState) {
  ...
  try {
    mediaPlayer = new MediaPlayer();
    mediaPlayer.setDataSource(
      "http://server3.pianosociety.com/protected/bach-bwv772-stahlbrand.mp3");
    mediaPlayer.prepare();
  } catch (IllegalArgumentException e) { ... }
  ...
}
```

Actividad propuesta II.9. *Desarrollar y probar la aplicación* `TestAudio`*, tal y como se ha descrito anteriormente.*

4.3. Vídeo

La reproducción de vídeo en Android es algo más compleja que la de Audio que se acaba de explicar. A fin de mostrar un vídeo, debemos especificar la superficie donde se mostrará el mismo. Para ello, existen dos formas en Android.

La primera, encapsulando la creación de una interfaz de visualización a través de la vista `VideoView` y guardando y preparando el contenido de vídeo mediante el Media Player.

Y la segunda, especificando la interfaz de visualización directamente y acto seguido, interactuando con una instancia del Media Player. En esta sección estudiaremos ambas formas de visualizar un vídeo en Android.

4.3.1. Clase `VideoView`

La vista `VideoView`[2] incluye una interfaz en donde el vídeo es visualizado y encapsula el manejo del Media Player para la reproducción del mismo. Tan sólo es necesario realizar una llamada a los métodos `setVideoUri()` o `setVideoPath()` para especificar la ruta a un proveedor de contenido o página web, o bien la ruta a un archivo local, respectivamente. También cuenta con otros métodos interesantes como `start()`, `pause()`, `resume()` o `seekTo()`, entre otros.

Para probar la funcionalidad de la vista `VideoView` vamos a desarrollar una aplicación llamada `TestVideo`, que simplemente contendrá una `VideoView`. La clase `TestVideoActivity` deberá contener el siguiente código:

```
private VideoView videoView;

public void onCreate(Bundle savedInstanceState) {
  ...
  videoView = (VideoView) findViewById(R.id.vistaVideo);
  videoView.setKeepScreenOn(true); // Mantenemos la pantalla encendida.
  videoView.setVideoURI(Uri.parse(
  "http://ants.dif.um.es/staff/felixgm/docencia/android/resources/techno-chicken.3gp"));
  videoView.start();
}
```

La figura 4.3 muestra el resultado de ejecutar la aplicación `TestVideo`.

[2] http://developer.android.com/reference/android/widget/VideoView.html

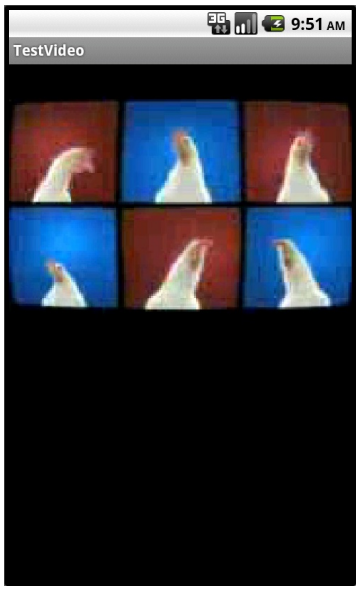

Figura 4.3: Aplicación `TestVideo`

Actividad propuesta II.10. *Desarrollar y probar la aplicación* `TestVideo`*, tal y como se ha descrito anteriormente.*

Actividad propuesta II.11. *Modificar la aplicación* `TestVideo` *desarrollada en la aplicación II.10. para que utilice el método* `setVideoPath()` *y reproduzca un vídeo almacenado en la tarjeta SD.*

4.3.2. Usando el Media Player

Cuando queremos tener una interfaz personalizada donde mostrar nuestro video tenemos que hacer uso de una "*Surface*" o superficie. El reproductor Media Player no podrá ejecutarse correctamente si no dispone de un objeto de la clase `SurfaceHolder`[3] donde poder mostrar el video. Dicho objeto `SurfaceHolder` deberá ser asignado mediante el método `setDisplay()` de la clase `MediaPlayer`. Android nos ofrece una vista llamada `SurfaceView`[4] que permite cubrir nuestras necesidades.

Para mostrar su manejo desarrollaremos una nueva aplicación llamada `TestVideoSurface`, que contendrá en su *layout* `main`, una `SurfaceView`, tal y como se observa en la figura 4.4.

El código de la clase `TestVideoSurfaceActivity` tendrá el siguiente aspecto:

```
public class TestVideoSurfaceActivity extends Activity
                                        implements SurfaceHolder.Callback{
    public void onCreate(Bundle savedInstanceState) {
        ...
        SurfaceView surface = (SurfaceView)findViewById(R.id.surfaceView);
        SurfaceHolder holder = surface.getHolder();
        holder.addCallback(this);
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
    ...
}
```

[3] http://developer.android.com/reference/android/view/SurfaceHolder.html
[4] http://developer.android.com/reference/android/view/SurfaceView.html

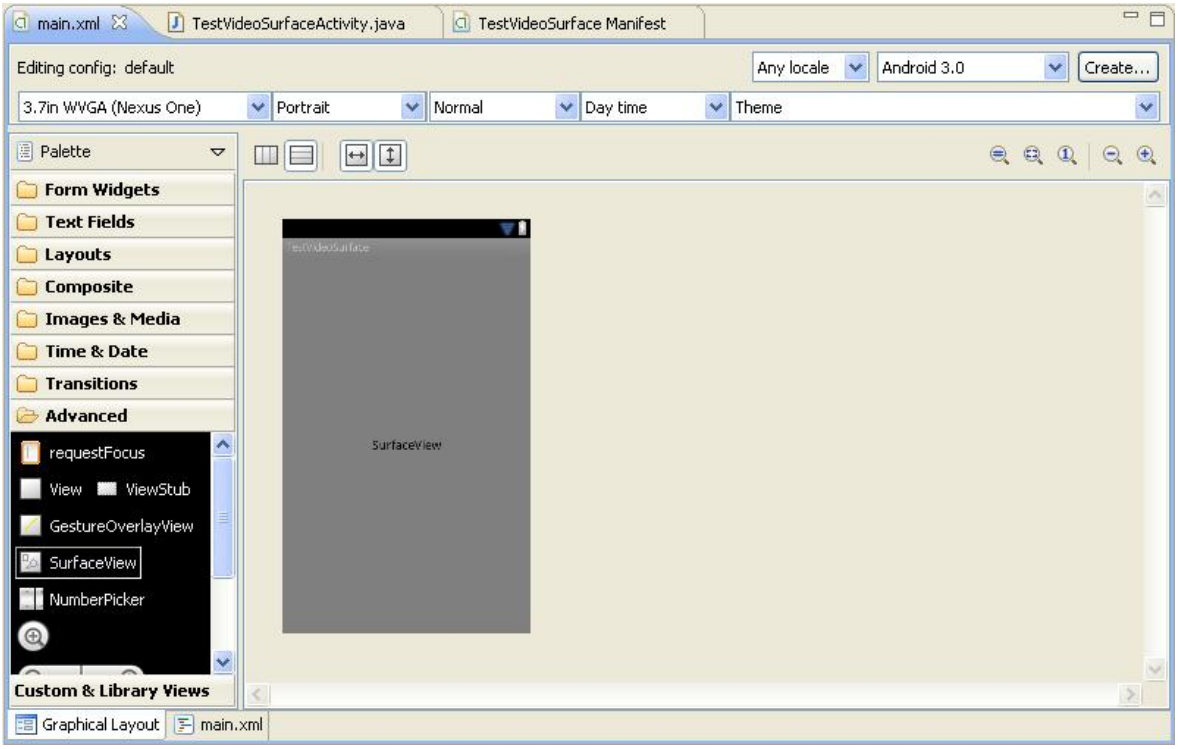

Figura 4.4: *Layout* `main` de la aplicación `TestVideoSurface`

Como podemos observar, nuestra actividad implementa la interfaz `SurfaceHolder.Callback`[5], que cuenta con los siguientes métodos:

```
public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
  ...
}

public void surfaceCreated(SurfaceHolder holder) {
  try {
    mediaPlayer.setDisplay(holder);
    mediaPlayer.setDataSource("/sdcard/techno-chicken.3gp");
    mediaPlayer.prepare();
    mediaPlayer.start();
  } catch (IllegalArgumentException e) {
    ...
  }
}

public void surfaceDestroyed(SurfaceHolder holder) {
  mediaPlayer.release();
}
```

Actividad propuesta II.12. *Desarrollar y probar la aplicación* `TestVideoSurface`*, tal y como se ha descrito anteriormente.*

[5] http://developer.android.com/reference/android/view/SurfaceHolder.Callback.html

4.4. Cámara

Para poder capturar y manejar imágenes mediante la cámara de nuestro dispositivo móvil Android, podremos hacer uso de los *intents*.

Para mostrar esta funcionalidad, vamos a crear una aplicación llamada `TestCamara`, que contará únicamente con dos botones y un `ImageView`, donde mostraremos la imagen capturada.

El primer botón nos permitirá tomar una imagen mediante la cámara del dispositivo móvil Android, mientras que el segundo botón nos servirá para almacenar la imagen capturada en nuestro dispositivo.

La figura 4.5 muestra el *layout* `main` de la aplicación `TestCamara`.

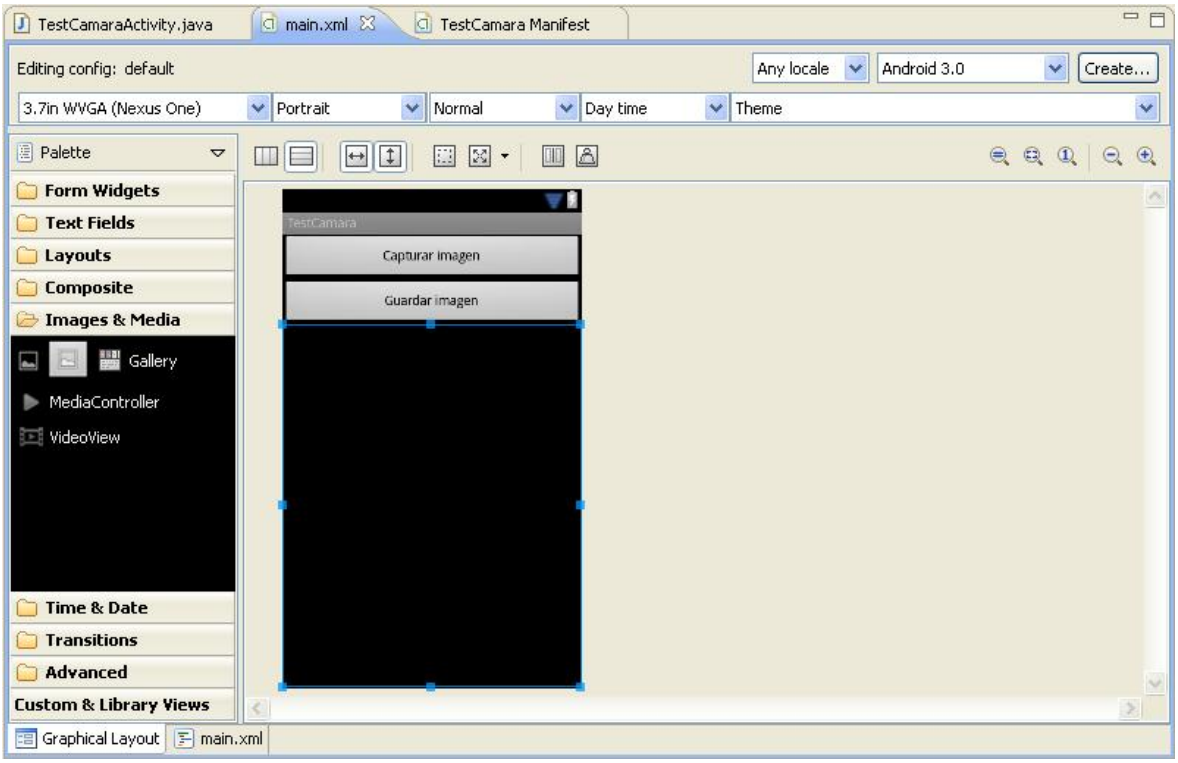

Figura 4.5: *Layout* `main` de la aplicación `TestCamara`

> **Nota.-** El emulador de dispositivos móviles Android de Eclipse no permite simular el comportamiento de la cámara, de modo que si queremos testear el correcto funcionamiento de la aplicación `TestCamara`, deberemos hacerlo en un dispositivo móvil Android real.

Por su parte, la clase `TestCamaraActivity` tendrá el siguiente aspecto, donde, como hemos comentado, cada uno de los botones tendrá asignado un código que, mediante el uso de *intents*, permitirá llevar a cabo las acciones deseadas (a saber, capturar una imagen con la cámara y almacenar una imagen capturada).

```
public final int CAMERA_RESULT = 0;
public final int TAKE_PICTURE = 1;
private String pictureFilePath;
...

public void onCreate(Bundle savedInstanceState) {
  ...
  pictureFilePath = Environment.getExternalStorageDirectory().getAbsolutePath()
                              + "/tmp_image.jpg";

  takePictureButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
      Intent intent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
      startActivityForResult(intent, TAKE_PICTURE);
    }
  });

  savePictureButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
      Uri imageFileUri = Uri.fromFile(new File(pictureFilePath));
      Intent intent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
      intent.putExtra(android.provider.MediaStore.EXTRA_OUTPUT, imageFileUri);
      startActivityForResult(intent, CAMERA_RESULT);
    }
  });
}
...
```

A este código tan sólo nos faltará añadirle el comportamiento que deseemos para el resultado de cada uno de los anteriores *intents*. Esto lo conseguimos mediante el siguiente método `onActivityResult()`.

```
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
  super.onActivityResult(requestCode, resultCode, intent);
  switch (requestCode) {
    case CAMERA_RESULT:
      if (resultCode == RESULT_OK) {
        Bitmap bmp = BitmapFactory.decodeFile(pictureFilePath);
      }
      break;

    case TAKE_PICTURE:
      if (intent !=null)
        if (intent.hasExtra("data")){
          Bitmap bmp = intent.getParcelableExtra("data");
          imageView.setImageBitmap(bmp);
        }
  }
}
```

Actividad propuesta II.13. *Desarrollar y probar la aplicación* `TestCamara`*, tal y como se ha descrito anteriormente.*

Tema 5

Bluetooth

*E*ste tema explicará los conceptos básicos para establecer una comunicación Bluetooth entre terminales Android, haciendo uso de dicho lenguaje de programación.

Nota.- Las librerías de Bluetooth para Android solamente están disponibles desde la versión 2.0 (SDK API level 5). También merece la pena recordar que no todos los dispositivo móviles Android contarán necesariamente con la capacidad de establecer comunicaciones mediante Bluetooth.

5.1. Introducción

Bluetooth es un protocolo de comunicación diseñado para comunicaciones de corto alcance, con bajo ancho de banda y de par a par (P2P). En Android, los dispositivos y comunicaciones Bluetooth se manejan y gestionan mediante las siguientes clases, principalmente:

- `BluetoothAdapter`[1]. El Adaptador de Bluetooth representa al dispositivo local de Bluetooth, esto es, el dispositivo Android sobre el que ejecutamos nuestras aplicaciones
- `BluetoothDevice`[2]. Esta clase representa a cada dispositivo remoto con el que establecer una comunicación a través de Bluetooth
- `BluetoothSocket`[3]. Un Socket Bluetooth se obtiene mediante la llamada al método `createRfcommSocketToServiceRecord()` sobre el objeto que representa al dispositivo Bluetooth remoto. Dicho Socket Bluetooth nos permitirá realizar peticiones de conexión a dispositivos remotos, así como iniciar la comunicación con los mismos
- `BluetoothServerSocket`[4]. La creación de un Socket Servidor Bluetooth (mediante la invocación del método `listenUsingRfcommWithServiceRecord()`) en el Adaptador Bluetooth local nos permitirá escuchar peticiones de conexión entrantes procedentes de dispositivos remotos, a través de Sockets Bluetooth

[1] http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html
[2] http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html
[3] http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html
[4] http://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html

5.2. Búsqueda de dispositivos

5.2.1. Clase `BluetoothAdapter`

Como ya hemos comentado, el dispositivo Bluetooth local se controla mediante la clase `BluetoothAdapter`. Y para recuperar el objeto de dicha clase que representa el adaptador de Bluetooth por defecto en el dispositivo local, haremos uso del método estático `getDefaultAdapter()`.

Este adaptador Bluetooth nos permitirá, entre otras cosas, conocer por ejemplo la dirección Bluetooth, así como el "*friendly-name*" del dispositivo local. El siguiente código es un ejemplo de ello:

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
String address = bluetooth.getAddress();
String name = bluetooth.getName();
Toast.makeText(this, name+" : "+address, Toast.LENGTH_LONG).show();
```

Por otra parte, para poder leer cualquier propiedad del adaptador local de Bluetooth, iniciar el descubrimiento de otros dispositivos, o buscar dispositivos emparejados, es necesario que nuestra aplicación cuente con el permiso `BLUETOOTH`. Y si lo que queremos es poder modificar alguna de las propiedades del dispositivo local, entonces necesitaremos también la propiedad `BLUETOOH_ADMIN`.

5.2.2. Activación de Bluetooth en el dispositivo

El adaptador de Bluetooth se encuentra por defecto desactivado. Para saber si el adaptador se encuentra activado o no, podemos hacer uso del método `isEnabled()` de la clase `BluetoothAdapter`. También podemos conocer en qué estado concreto se encuentra el adaptador Bluetooth mediante el método `getState()`, de esa misma clase, que nos devolverá alguno de los valores que se observan en la tabla 5.1.

Constante	Descripción
`STATE_TURNING_ON`	El adaptador se está activando
`STATE_ON`	El adaptador está activado
`STATE_TURNING_OFF`	El adaptador se está desactivando
`STATE_OFF`	El adaptador está desactivado

Tabla 5.1: Constantes de estado de la clase `BluetoothAdapter`

Si lo que queremos es activar el adaptador de Bluetooth del dispositivo tendremos que ejecutar una sub-actividad del sistema utilizando la constante estática `ACTION_REQUEST_ENABLE` de la clase `BluetoothAdapter` como acción para el método `startActivityForResult()`, tal y como se observa a continuación.

```
startActivityForResult(new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE), 0);
```

Esta sub-actividad muestra un mensaje al usuario y requiere su confirmación para que active el adaptador de Bluetooth. Si el usuario acepta, la sub-actividad terminará y volverá a la actividad desde donde se hubiera hecho la llamada, una vez que el adaptador se haya activado. Si por el contrario el usuario cancela la activación, entonces la sub-actividad se cerrará y el flujo de ejecución volverá inmediatamente a la actividad desde donde se hubiera hecho la llamada.

Puesto que la activación y desactivación del adaptador de Bluetooth son operaciones costosas y asíncronas, conviene que nuestra aplicación registre un receptor broadcast que escuche los cambios de estado del adaptador (`ACTION_STATE_CHANGED`).

El siguiente código nos permite activar el adaptador de Bluetooth así como observar los cambios de estado producidos en el mismo.

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
BroadcastReceiver bluetoothState = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String stateExtra = BluetoothAdapter.EXTRA_STATE;
    int state = intent.getIntExtra(stateExtra, -1);

    String toastText = "";
    switch (state) {
      case (BluetoothAdapter.STATE_TURNING_ON) : {
        toastText = "Bluetooth turning on";
        break;
      }
      case (BluetoothAdapter.STATE_ON) : {
        toastText = "Bluetooth on";
        unregisterReceiver(this);
        break;
      }
      case (BluetoothAdapter.STATE_TURNING_OFF) : {
        toastText = "Bluetooth turning off";
        break;
      }
      case (BluetoothAdapter.STATE_OFF) : {
        toastText = "Bluetooth off";
        break;
      }
      default:
        break;
    }
    Toast.makeText(this, toastText, Toast.LENGTH_LONG).show();
  }
};

if (!bluetooth.isEnabled()) {
  String actionStateChanged = BluetoothAdapter.ACTION_STATE_CHANGED;
  String actionRequestEnable = BluetoothAdapter.ACTION_REQUEST_ENABLE;
  registerReceiver(bluetoothState, new IntentFilter(actionStateChanged));
  startActivityForResult(new Intent(actionRequestEnable), 0);
}
```

5.2.3. Descubrimiento de dispositivos remotos

Para que un dispositivo Android remoto pueda descubrir nuestro dispositivo a través de Bluetooth, el adaptador local debe ser "descubrible". Para conocer si el adaptador local de Bluetooth puede ser descubierto por otros dispositivos remotos podemos hacer uso del método `getScanMode()` de la clase `BluetoothAdapter`. Dicho método devolverá alguna de las constantes mostradas en la tabla 5.2.

Constante	Descripción
`SCAN_MODE_CONNECTABLE_DISCOVERABLE`	El dispositivo puede ser descubierto por cualquier otro dispositivo realizando una búsqueda
`SCAN_MODE_CONNECTABLE`	El dispositivo puede ser descubierto sólo por aquellos otros dispositivos con los que se hubiera emparejado anteriormente, pero no por nuevos dispositivos realizando búsquedas
`SCAN_MODE_NONE`	El dispositivo no puede ser descubierto por ningún otro dispositivo remoto

Tabla 5.2: Constantes de descubrimiento de la clase `BluetoothAdapter`

Para permitir que el dispositivo local pueda ser descubierto por otros dispositivos remotos (que por defecto no lo es, por razones de privacidad), podemos lanzar una nueva actividad usando la acción `ACTION_REQUEST_DISCOVERABLE`, tal y como se muestra a continuación:

```
startActivityForResult(
  new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE), DISCOVERY_REQUEST);
```

Por otra parte, si lo que queremos es descubrir dispositivos remotos a través de Bluetooth, en primer lugar podremos comprobar si nuestro adaptador local se encuentra en este momento buscando dispositivos cercanos mediante el método `isDiscovering()` de la clase `BluetoothAdapter`. El proceso de descubrimiento se inicia mediante la llamada al método `startDiscovery()` y se cancela haciendo uso del método `cancelDiscovery()`.

El siguiente código permite monitorizar los cambios en el proceso de descubrimiento de otros dispositivos Bluetooth remotos, mediante un receptor broadcast que escucha los intents `ACTION_DISCOVERY_STARTED` y `ACTION_DISCOVERY_FINISHED`:

```
BroadcastReceiver discoveryMonitor = new BroadcastReceiver() {
  String dStarted = BluetoothAdapter.ACTION_DISCOVERY_STARTED;
  String dFinished = BluetoothAdapter.ACTION_DISCOVERY_FINISHED;

  @Override
  public void onReceive(Context context, Intent intent) {
    if (dStarted.equals(intent.getAction())) {
      // El proceso de descubrimiento ha comenzado
      ...
    } else if (dFinished.equals(intent.getAction())) {
      // El proceso de descubrimiento ha terminado
      ...
    }
  }
};
```

```
registerReceiver(discoveryMonitor,
  new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_STARTED));
registerReceiver(discoveryMonitor,
  new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED));
```

Para recuperar los dispositivos Bluetooth remotos encontrados podemos hacer uso del siguiente código, que registra un receptor broadcast que escucha el intent `ACTION_FOUND`. Cada intent broadcast incluye el nombre del dispositivo remoto en un extra indexado como `BluetoothDevice.EXTRA_NAME`, así como una representación inmutable del dispositivo Bluetooth remoto en un objeto de la clase `BluetoothDevice` almacenado en el extra `BluetoothDevice.EXTRA_DEVICE`.

```
BroadcastReceiver discoveryResult = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String remoteDeviceName =
           intent.getStringExtra(BluetoothDevice.EXTRA_NAME);
    BluetoothDevice remoteDevice =
           intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
    ...
  }
};
registerReceiver(discoveryResult, new IntentFilter(BluetoothDevice.ACTION_FOUND));

if (!bluetooth.isDiscovering())
  bluetooth.startDiscovery();
```

5.3. Establecimiento de la conexión

5.3.1. Servidor

Un socket servidor Bluetooth se utiliza para escuchar peticiones entrantes de conexión se sockets Bluetooth provenientes de dispositivos Bluetooth remotos. Para que dos dispositivos Bluetooth se conecten, uno de ellos debe actuar como servidor (escuchando y aceptando peticiones entrantes), mientras que el otro debe actuar como cliente (iniciando la petición para conectarse al servidor).

Una vez que ambos se han conectado, la comunicación entre servidor y cliente se gestiona mediante un socket Bluetooth en los dos extremos.

Para recuperar el socket servidor Bluetooth que nos permitirá escuchar peticiones de conexión entrantes debemos llamar al método `listenUsingRfcommWithServiceRecord()` del adaptador local Bluetooth, pasándole un nombre para identificar al servidor, así como un UUID. Dicho método nos devolverá un objeto `BluetoothServerSocket`, sobre el cual deberemos invocar a su vez al método `accept()`, para efectivamente comenzar a escuchar peticiones de conexión.

Si una petición de conexión entrante es aceptada, entonces el método `accept()` devolverá un `BluetoothSocket` que nos permitirá transferir datos con el cliente, como veremos más adelante.

> **Nota.-** La operación `accept()` es bloqueante, por lo que es recomendable escuchar peticiones de conexión entrantes en un hilo en segundo plano, en lugar de bloquear el hilo principal hasta que se realice una conexión.

El siguiente código recupera un `BluetoothServerSocket` a partir de un UUID y lanza un hilo en background para escuchar peticiones de conexión entrantes sobre dicho socket servidor:

```
startActivityForResult(
  new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE), DISCOVERY_REQUEST);

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   if (requestCode == DISCOVERY_REQUEST) {
      boolean isDiscoverable = resultCode > 0;
      int discoverableDuration = resultCode;
      if (isDiscoverable) {
         UUID uuid = UUID.fromString("b35d90c2-f22c-26ed-6c85-08002009c666");
         String name = "Servidor Bluetooth";

         final BluetoothServerSocket btserver =
            bluetooth.listenUsingRfcommWithServiceRecord(name, uuid);

         Thread acceptThread = new Thread(new Runnable() {
            public void run() {
               try {
                  BluetoothSocket serverSocket = btserver.accept();
                  ...
               } catch (IOException e) {
                  ...
               }
            }
         });
         acceptThread.start();
      }
   }
}
```

5.3.2. Cliente

Para obtener un BluetoohSocket que nos permita comunicarnos con un dispositivo Bluetooth remoto deberemos recuperar el objeto de la clase BluetoothDevice que represente a dicho dispositivo remoto. Como vimos en la sección 5.2.3, podemos obtener dicho objeto mediante un proceso de descubrimiento de dispositivos remotos.

Sin embargo, existen otras alternativas, como por ejemplo recuperar directamente dicho dispositivo remoto a partir de su dirección hardware, a través del adaptador local de Bluetooth.

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
BluetoothDevice remoteDevice = bluetooth.getRemoteDevice("01:23:77:35:2F:AA");
```

Otra opción puede ser recuperar la lista de dispositivos previamente emparejados con el adaptador local de Bluetooth. Para ello utilizaremos el método `getBondedDevices()`.

Una vez tenemos el `BluetoothDevice` que represente al dispositivo remoto al cual queramos conectarnos, deberemos llamar al método `createRfcommSocketToServiceRecord()` pasándole el UUID del socket servidor Bluetooth aceptando peticiones entrantes.

Este método nos devolverá un `BluetoothSocket` que nos servirá para iniciar la comunicación, mediante la llamada al método `connect()`.

> **Nota.-** La operación `connect()` es bloqueante, por lo que es recomendable iniciar una conexión Bluetooth en un hilo en segundo plano, en lugar de bloquear el hilo principal hasta que se realice dicha conexión.

El siguiente código busca un dispositivo remoto Bluetooth cuya dirección hardware es `01:23:77:35:2F:AA`, y que además esté previamente emparejado con el adaptador local. Si lo encuentra, envía una petición de conexión entrante, usando el UUID donde está escuchando el servidor. La conexión se realiza en un hilo en segundo plano.

```
final BluetoothDevice device = bluetooth.getRemoteDevice("01:23:77:35:2F:AA");
final Set<BluetoothDevice> bondedDevices = bluetooth.getBondedDevices();

BroadcastReceiver discoveryResult = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      BluetoothDevice remoteDevice =
            intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
      if (remoteDevice.equals(device) &&
                     bondedDevices.contains(remoteDevice)) {
         UUID uuid = UUID.fromString("b35d90c2-f22c-26ed-6c85-08002009c666");
         try {
            final BluetoothSocket clientSocket =
                 remoteDevice.createRfcommSocketToServiceRecord(uuid);
            Thread connectThread = new Thread(new Runnable() {
               public void run() {
                  try {
                     clientSocket.connect();
                     ...
                  } catch (IOException e) {
                     ...
                  }
               }
            });
            connectThread.start();
         } catch (IOException e) {
           ...
         }
      }
   }
};
registerReceiver(discoveryResult,
        new IntentFilter(BluetoothDevice.ACTION_FOUND));

if (!bluetooth.isDiscovering())
  bluetooth.startDiscovery();
```

5.4. Transferencia de datos

Una vez se haya establecido una conexión Bluetooth entre cliente y servidor tendremos un `BluetoothSocket` en ambos extremos. Dicho socket podremos utilizarlo para la comunicación entre los dispositivos, bien leyendo o escribiendo datos en el mismo.

Para ello, es posible recuperar sendos objetos `InputStream`[5] y `OutputStream`[6], tal y como se muestra a continuación:

```
BluetoothSocket socket = ...
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
```

5.5. Aplicación `TestBluetooth`

> **Nota.-** Desafortunadamente, el emulador de Android no permite simular la funcionalidad Bluetooth, por lo que cualquier aplicación que haga uso de la misma deberá ser testeada en dispositivos móviles reales.

Con todo lo visto hasta ahora, vamos a crear nuestra aplicación de prueba llamada `TestBluetooth`. Como vimos en la sección 5.2.1, en primer lugar deberemos darle los permisos de `BLUETOOTH` y `BLUETOOTH_ADMIN` a nuestra aplicación, tal y como se observa en la figura 5.1.

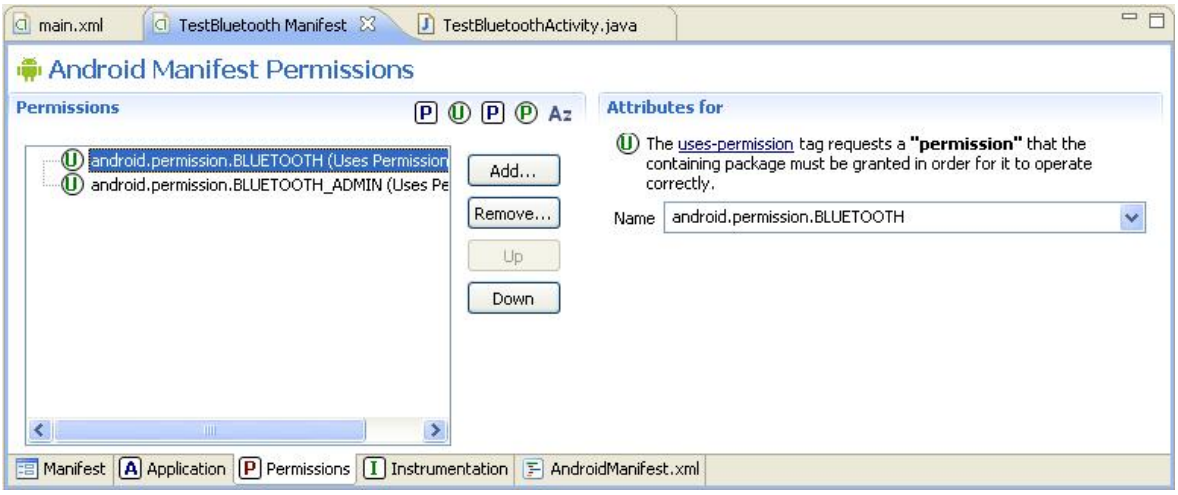

Figura 5.1: Permisos `BLUETOOTH` y `BLUETOOTH_ADMIN` en el manifiesto de `TestBluetooth`

El *layout* `main` de esta aplicación, tal y como se puede ver en la figura 5.2, consta de los siguientes elementos (por este orden): un `Button` llamado `searchButton` que nos permitirá iniciar la búsqueda de dispositivos Bluetooth remotos, otro `Button` llamado `helloButton`, que enviará un mensaje informativo a uno de los dispositivos encontrados, un `TextView` llamado `messagesTextView`, que utilizaremos para mostrar mensajes al usuario, y por último, una `ListView` llamada `discoveredDevicesListView` que mostrará aquellos dispositivos encontrados en la búsqueda.

[5] http://developer.android.com/reference/android/java/io/InputStream.html
[6] http://developer.android.com/reference/android/java/io/OutputStream.html

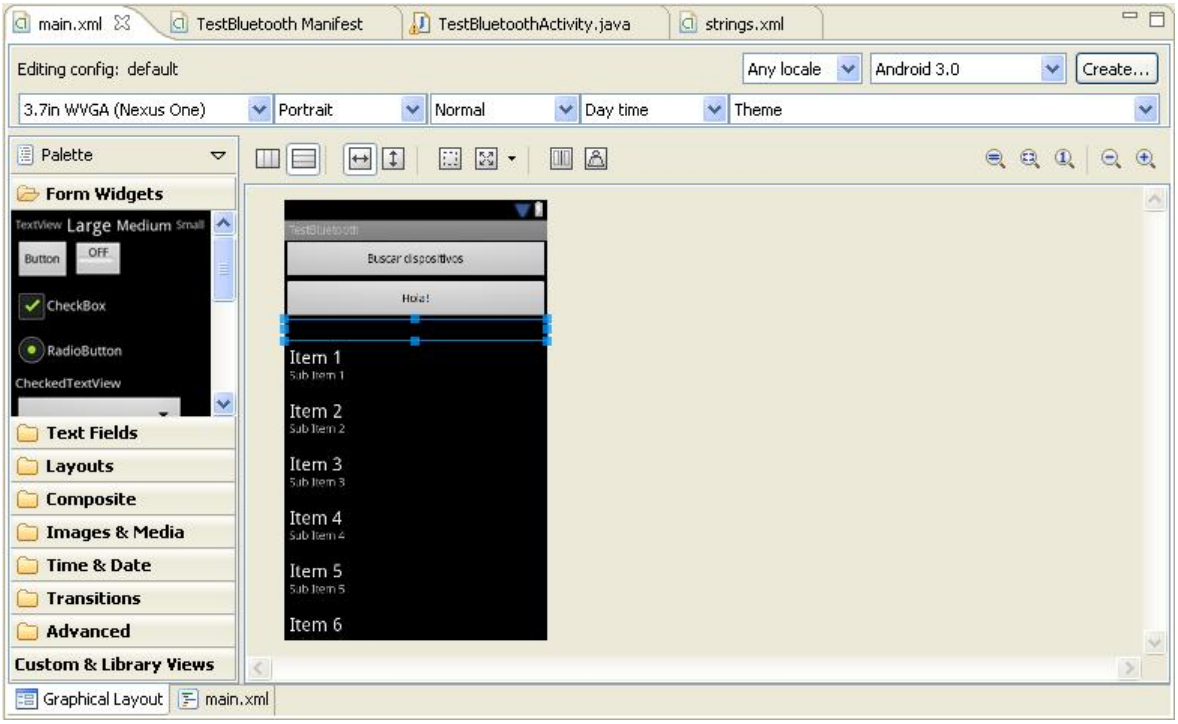

Figura 5.2: *Layout* `main` de la aplicación `TestBluetooth`

Actividad propuesta II.14. *Descargar la aplicación* `TestBluetooth` *de la web del curso y examinar su código para entenderlo. Si se dispone de más de un dispositivo Android, se puede instalar la aplicación en los mismos y probarla.*

Tema 6

Acelerómetro y Brújula

*E*n este tema se verán los conocimientos necesarios para utilizar el acelerómetro y la brújula en dispositivos móviles Android.

6.1. Acelerómetro

6.1.1. Introducción

Muchos de los dispositivos Android cuentan con multitud de sensores como los de campo magnético, presión, temperatura, luz, etc.

Dentro de esos sensores encontramos el acelerómetro, cuyo uso se ha extendido ampliamente en los últimos tiempos debido a que amplía enormemente las posibilidades que nos ofrecen los terminales.

Sin ir más lejos, la actualización de la información de la pantalla en función de los cambios de movimiento han cambiado radicalmente el enfoque de los desarrollos, puesto que la pantalla deja de ser un elemento estático, y la misma puede mostrar más o menos información en función de su orientación.

Pero, ¿qué es un acelerómetro? Según Wikipedia se denomina acelerómetro a cualquier instrumento destinado a medir aceleraciones. Dicho de una forma más coloquial, un acelerómetro es un dispositivo (sensor) instalado dentro del terminal móvil que sirve para detectar los movimientos del mismo y su posición.

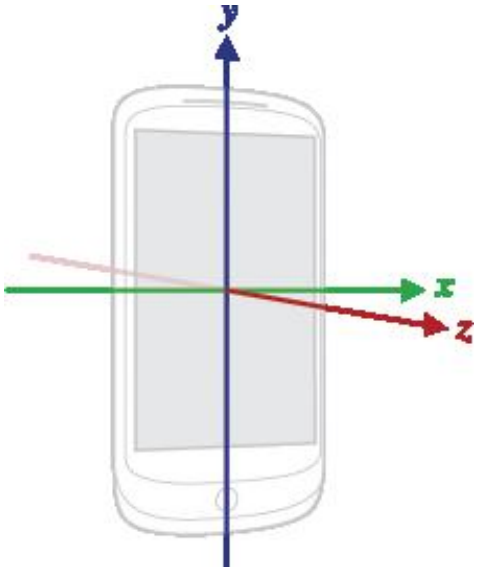

Figura 6.1: Ejes de coordenadas para el acelerómetro

Si puede detectar movimiento, el acelerómetro necesita trabajar con coordenadas. De esta forma, el acelerómetro tiene en cuenta el esquema de orientación mostrado en la figura 6.1.

En esta sección veremos cómo acceder a las coordenadas que vemos en la figura 6.1 de forma que podamos tener en cuenta los posibles cambios de orientación del dispositivo a la hora de desarrollar nuestras aplicaciones en Android.

6.1.2. `SensorManager`, `SensorEventListener` y `SensorEvent`

Para poder gestionar todos esos sensores, Android ofrece una clase llamada `SensorManager` [1] que es la encargada de permitir el acceso a los sensores del dispositivo. Para ello, en nuestra aplicación tendremos que obtener una instancia de esta clase mediante la llamanda al método `Context.getSystemService()` con el argumento `SENSOR_SERVICE`.

Un aspecto muy importante a tener en cuenta cuando trabajemos con sensores, es que debemos deshabilitar los sensores en nuestra aplicación cuando no los necesitemos, especialmente cuando nuestra aplicación esté pausada, puesto que sino lo hacemos consumiremos la batería del terminal en muy poco tiempo. Android por defecto no deshabilita los sensores automáticamente cuando la pantalla se apaga.

Para gestionar esa deshabilitación, tendremos que hacer uso de los métodos `onResume()` y `onPause()` en nuestra `Activity`. Para el método `onResume()` le diremos al gestor de los sensores que vuelva a registrar el sensor, mientras que para el método `onPause()` haremos justo lo contrario: pedirle al gestor que elimine de la lista de *listeners* a nuestra aplicación.

```
private final SensorManager sensorManager;
private final Sensor accelerometer;

public TestAcelerometroActivity() {
  sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
  accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}

public void onCreate(Bundle savedInstanceState) {
  ...
  sensorManager.registerListener(this, accelerometer,
                                       SensorManager.SENSOR_DELAY_NORMAL);
}

protected void onResume() {
  super.onResume();
  sensorManager.registerListener(this, accelerometer,
                                       SensorManager.SENSOR_DELAY_NORMAL);
}

protected void onPause() {
  super.onPause();
  sensorManager.unregisterListener(this);
}
```

Adicionalmente cuando desarrollemos nuestra aplicación, deberemos implementar la interfaz denominada `SensorEventListener` [2] que se encargará del manejo de las actualizaciones de los sensores. Dicha interfaz incluye los dos siguientes métodos.

[1] http://developer.android.com/reference/android/hardware/SensorManager.html
[2] http://developer.android.com/reference/android/hardware/SensorEventListener.html

- `onAccuracyChanged(Sensor sensor, int accuracy)`
- `onSensorChanged(SensorEvent event)`

El método `onAccuracyChanged()` es llamado cuando la precisión del sensor cambia, mientras que el método `onSensorChanged()` se invoca cuando los valores medidos por el sensor cambian.

Es precisamente en estos métodos donde tendremos que programar el comportamiento que queramos para nuestra aplicación haciendo uso de la información que nos proporcione el acelerómetro.

Los eventos de la clase `SensorEvent` [3] registran cuatro propiedades básicas de un evento producido en un sensor, tal y como se observa en la tabla 6.1.

Propiedad	Descripción
`sensor`	El objeto `Sensor` [4] que ha capturado el evento
`accuracy`	La precisión que tenía el sensor cuando capturó el evento (baja, media, alta, o no detectada)
`values`	Los valores detectados en forma de array en punto flotante
`timestamp`	El instante (en nanosegundos) en que ocurrió el evento

Tabla 6.1: Propiedades de la clase `SensorEvent`

Por su parte, los valores almacenados en el array `values`, para el caso de que el sensor sea el acelerómetro son los siguientes.

- `values[0]`: Lateral (eje X)
- `values[1]`: Longitudinal (eje Y)
- `values[2]`: Vertical (eje Z)

6.1.3. Aplicación `TestAcelerometro`

Vamos a desarrollar ahora una aplicación llamada `TestAcelerometro` que nos servirá para mostrar y probar el manejo del acelerómetro en Android. Esta aplicación contará únicamente con cuatro `TextViews`, tal y como se observa en la figura 6.2.

Como explicábamos anteriormente, queremos que cuando se produzca un movimiento en el terminal se actualice la pantalla de nuestra aplicación. Así, el contenido del método `onSensorChanged()` será el siguiente.

```
public void onSensorChanged(SensorEvent event) {
  if(accelerometer != null){
    float x = event.values[0];
    float y = event.values[1];
    float z = event.values[2];

    textViewX.setText(getResources().getString(R.id.textViewX)+x);
    textViewY.setText(getResources().getString(R.id.textViewY)+y);
    textViewZ.setText(getResources().getString(R.id.textViewZ)+z);
  }
}
```

[3] http://developer.android.com/reference/android/hardware/SensorEvent.html

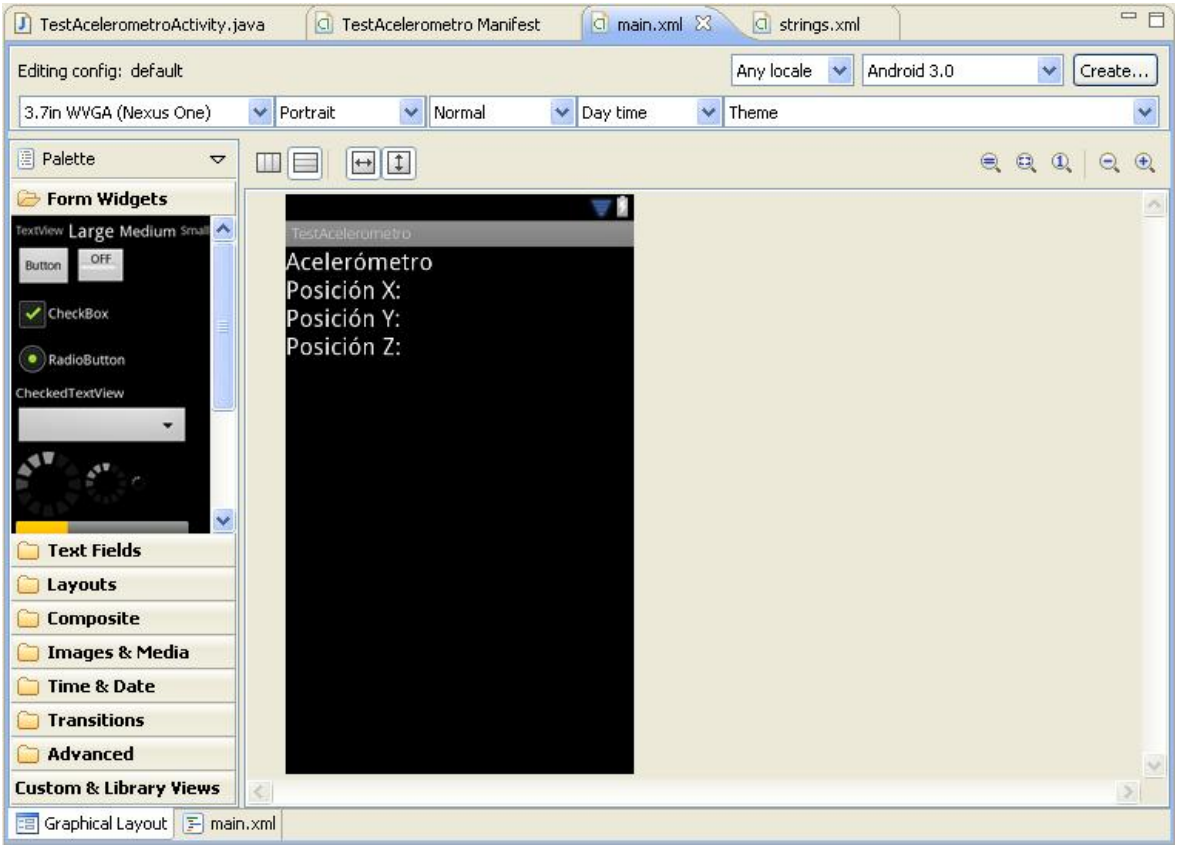

Figura 6.2: *Layout* `main` de la aplicación `TestAcelerometro`

Actividad propuesta II.15. *Desarrollar y probar la aplicación* `TestAcelerometro`*, tal y como se ha descrito anteriormente.*

6.2. Brújula

A la hora de consultar la información sobre las coordenadas Norte-Sur-Este-Oeste de nuestro dispositivo podemos hacer uso del sensor de orientación de la misma forma que hemos visto en el apartado del acelerómetro.

Por lo tanto, para obtener el objeto de la clase `Sensor` que represente a la brújula, podemos hacer uso del siguiente código.

```
Sensor compass = sensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
```

Mientras que si queremos registrar nuestra actividad para que escuche los eventos de la brújula, entonces deberemos usar un código similar al que se muestra a continuación.

```
sensorManager.registerListener(this, compass, SensorManager.SENSOR_DELAY_NORMAL);
```

Por otra parte, los valores almacenados en el array `values` (véase la tabla 6.1), para el caso de la brújula son los siguientes, tal y como se muestran en la figura 6.3.

- `values[0]`: Azimuth (eje vertical)
- `values[1]`: Pitch (eje transversal)
- `values[2]`: Roll (eje longitudinal)

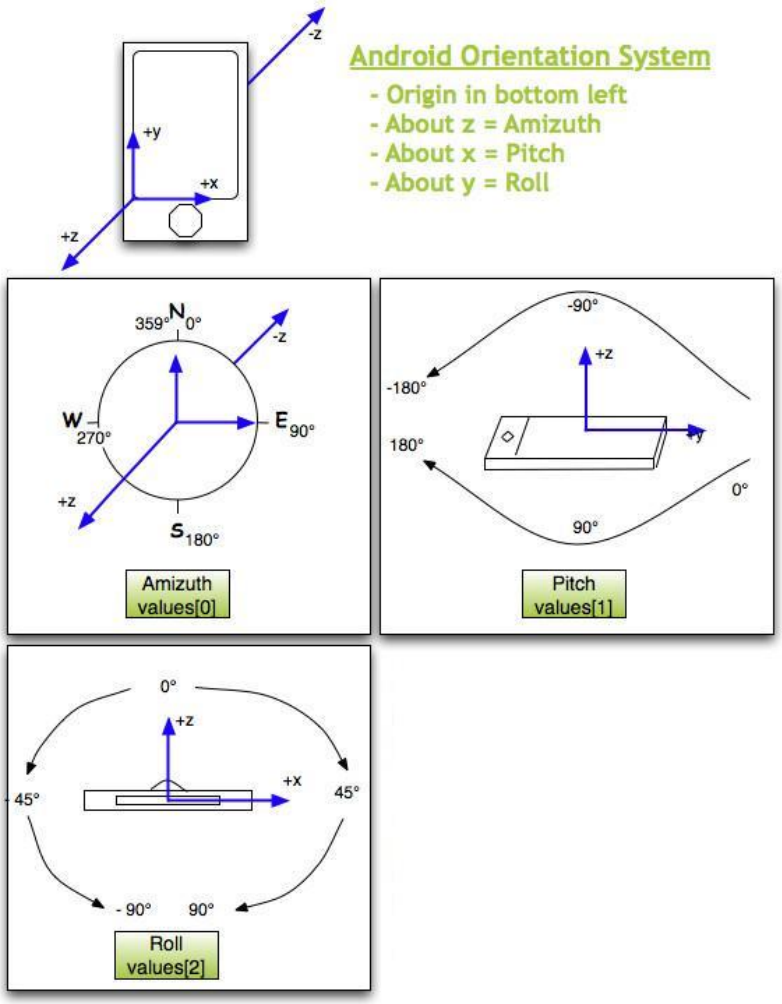

Figura 6.3: Sistema de orientación de Android

Como podemos ver en la figura 6.4, el *azimuth* nos indicará la orientación del dispositivo (Norte, Sur, Este u Oeste). Por su parte, el *pitch* mide el grado de inclinación del dispositivo respecto al eje mayor, mientras que el *roll* hace lo propio con respecto al eje menor.

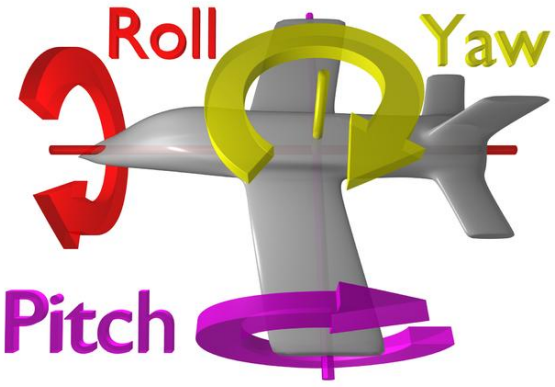

Figura 6.4: *Yaw* (*Azimuth*), *Pitch* y *Roll*

6.2.1. Aplicación `TestBrujula`

Por último vamos a desarrollar una aplicación sencilla, llamada `TestBrujula`, similar a la aplicación `TestAcelerometro` desarrollada anteriormente, que simplemente mostrará por pantalla los valores de *azimuth*, *pitch* y *roll*, así como sus variaciones, conforme el dispositivo se mueva en el espacio.

La figura 6.5 muestra el *layout* `main` de la aplicación `TestBrujula`.

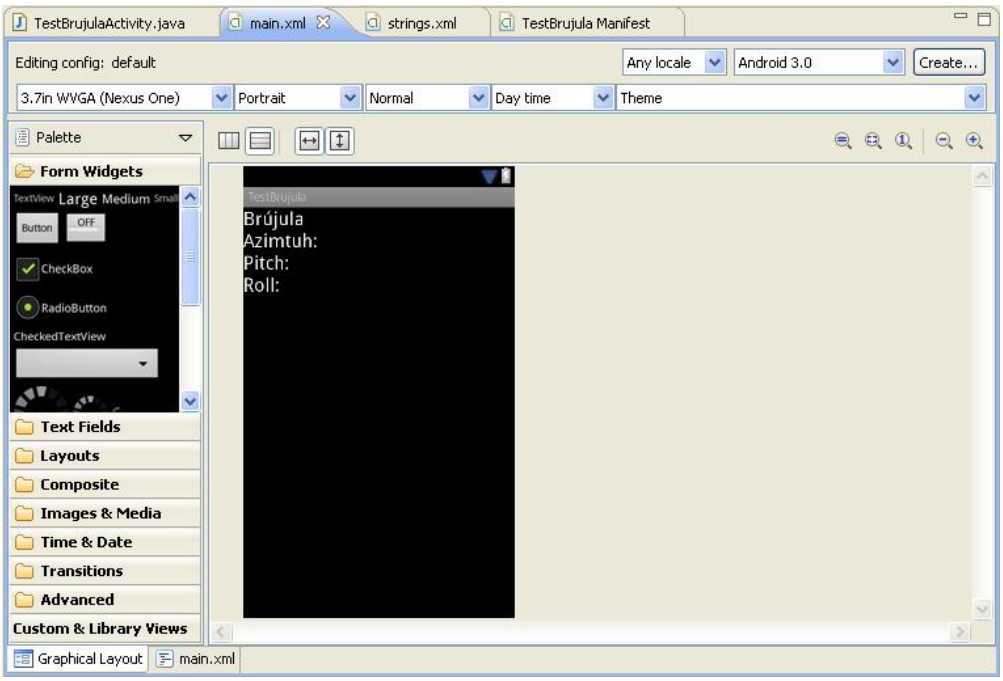

Figura 6.5: *Layout* `main` de la aplicación `TestBrujula`

Por su parte, el método `onSensorChanged()` tendrá el siguiente aspecto.

```
public void onSensorChanged(SensorEvent event) {
  if (compass != null) {
    String azimtuhText = getResources().getString(R.id.azimuthTextView)+event.values[0];
    if (event.values[0] > 0.00 && event.values[0] < 90.00) {
      azimtuhText += " (orientación Noreste)";
    } else if (event.values[0] > 90.00 && event.values[0] < 180.00) {
      azimtuhText += " (orientación Sureste)";
    } else if (event.values[0] > 180.00 && event.values[0] < 270.00) {
      azimtuhText += " (orientación Suroeste)";
    } else if (event.values[0] > 270.00 && event.values[0] < 360.00) {
      azimtuhText += " (rientación Noroeste)";
    }
    azimuthTextView.setText(azimtuhText);

    pitchTextView.setText(getResources().getString(R.id.pitchTextView)+event.values[1]);
    rollTextView.setText(getResources().getString(R.id.rollTextView)+event.values[2]);
  }
}
```

Actividad propuesta II.16. *Desarrollar y probar la aplicación `TestBrujula`, tal y como se ha descrito anteriormente.*

Tema 7

GPS y Google Maps

*E*n este tema se mostrará cómo utilizar el GPS integrado en los dispositivos móviles Android, al tiempo que se verá cómo integrar Google Maps en nuestras aplicaciones Android.

Nota.- A la hora de desarrollar cualquier aplicación que use los mapas de Google debemos tener en cuenta que su utilización conlleva la aceptación de una serie de términos y condiciones que debemos conocer, sobretodo en caso de que desarrollemos aplicaciones comerciales, pues en este caso su utilización ya no es gratuita.

7.1. Configuración de Eclipse para el uso de mapas

En primer lugar deberemos asegurarnos de tener instalado el paquete de mapas en Eclipse que nos permitirá interactuar con los mismos. El nombre habitual de este paquete será:

`Google APIs by Google, Android API x, revisión y`

Para comprobar si lo tenemos correctamente instalado, y en caso negativo poder instalarlo debemos acceder al Android SDK and AVD Manager y observar si aparece en la sección de paquetes instalados, tal y como se observa en la figura 7.1.

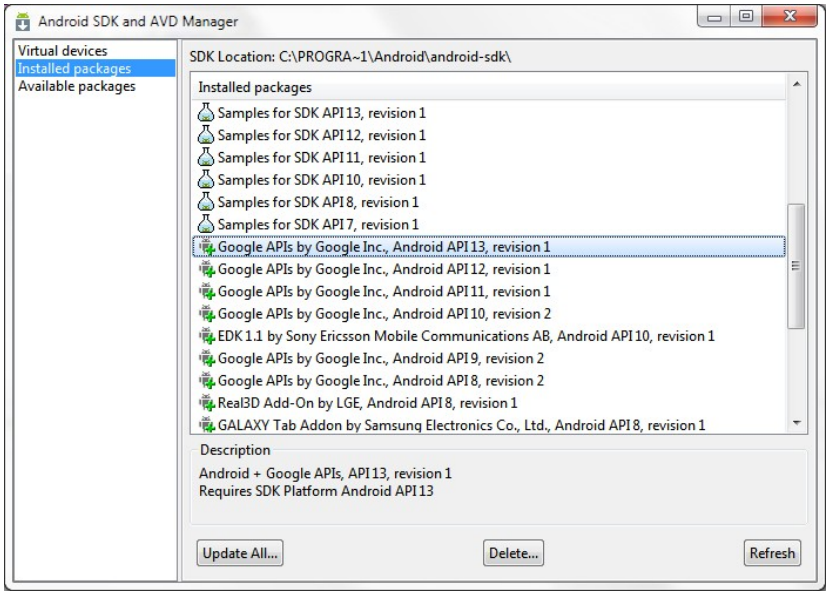

Figura 7.1: Android SDK and AVD Manager: `Google API`

Por otra parte, si queremos probar la aplicación en el emulador tenemos también que crear un nuevo AVD que utilice el paquete en cuestión como "target", tal y como podemos ver en la figura 7.2.

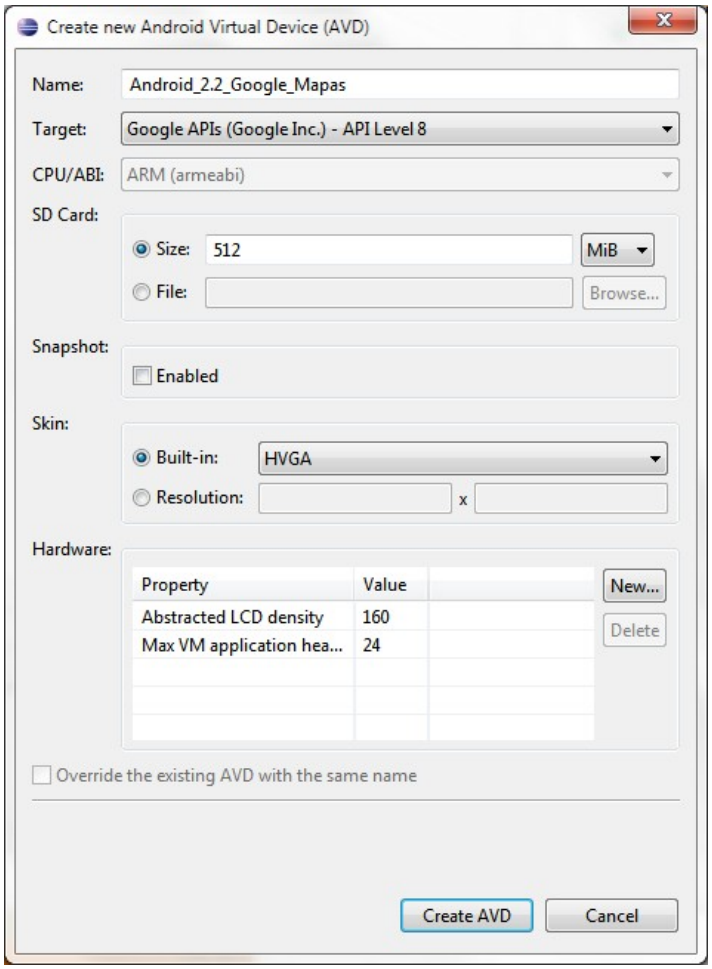

Figura 7.2: Creando un nuevo ADV usando la *Google API*

Por último, al crear el proyecto en Eclipse también tendremos que seleccionarlo como "target", en las propiedades del mismo, tal y como podemos comprobar en la figura 7.3. En ella se observa que vamos a crear una nueva aplicación llamada `TestMapas`, que nos permitirá comprobar algunas de las funcionalidad de los mapas en aplicaciones desarrolladas para dispositivos Android.

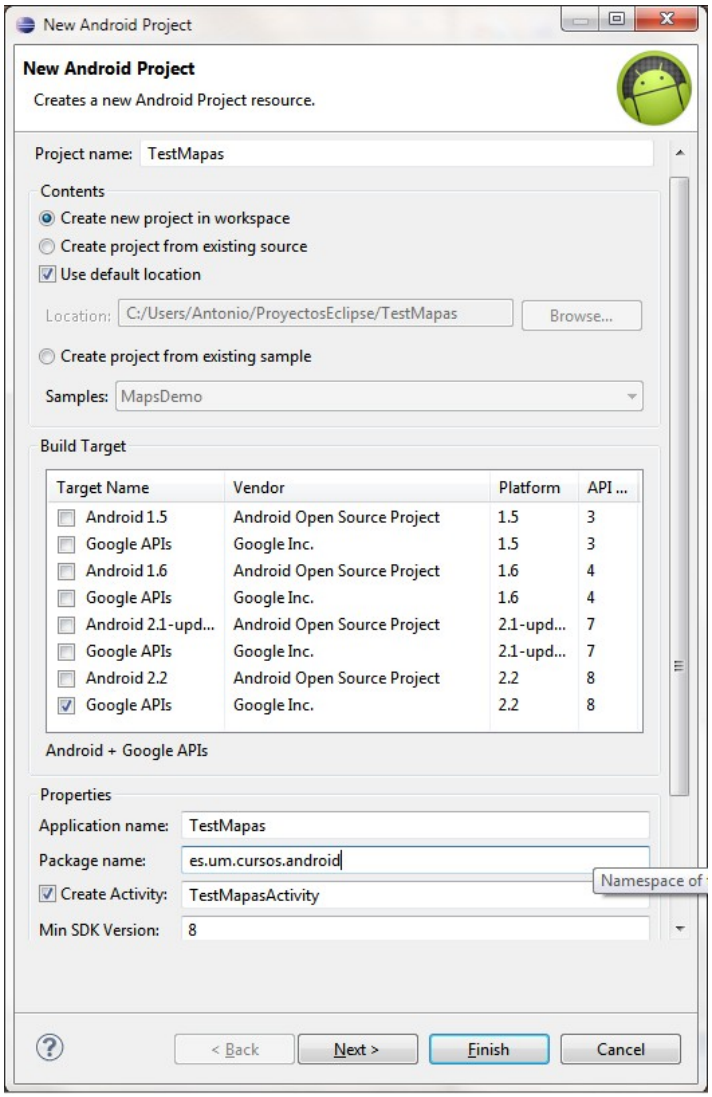

Figura 7.3: Creando el proyecto `TestMapas`

7.1.1. Generación de la clave para la `Android Maps API`

Android exige que toda aplicación instalada en un dispositivo (y que por tanto se pueda comercializar en Android Market) esté firmada digitalmente con un certificado digital cuya clave privada sea propiedad del desarrollador. De esta forma se consigue que el autor de la aplicación esté correctamente identificado.

Así, para poder hacer uso del `Android Maps API` necesitamos que Google nos proporcione una clave para dicho API, la cual estará asociada al certificado con el que firmaremos digitalmente nuestras aplicaciones que utilicen mapas.

Para solicitar dicha clave debemos acceder a la siguiente dirección web, tal y como se observa en la figura 7.4:

`http://code.google.com/intl/es-ES/android/maps-api-signup.html`

Como podemos comprobar, para obtener dicha clave necesitamos proporcionarle a Google la huella digital de nuestro certificado. Para ello localizaremos el fichero donde se almacenan los datos del certificado de depuración creado por defecto, llamado `debug.keystore`.

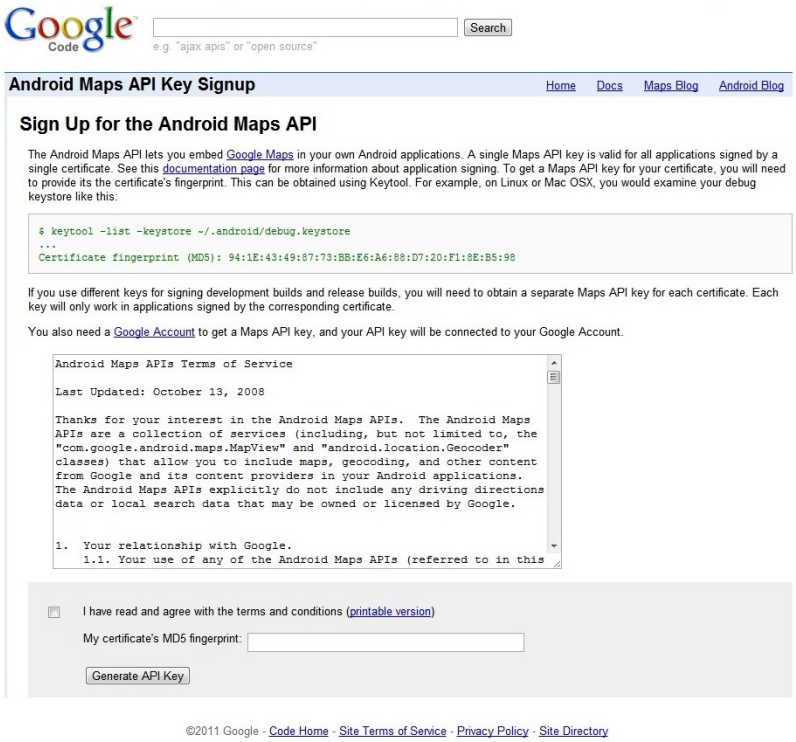

Figura 7.4: Obtención de la clave para la `Android Maps API` (I)

A su vez, para conocer la localización de dicho certificado, abriremos la ventana Windows → Preferences de Eclipse, y dentro del apartado Android → Build encontraremos la ruta del certificado en cuestión, tal y como se puede comprobar en la figura 7.5.

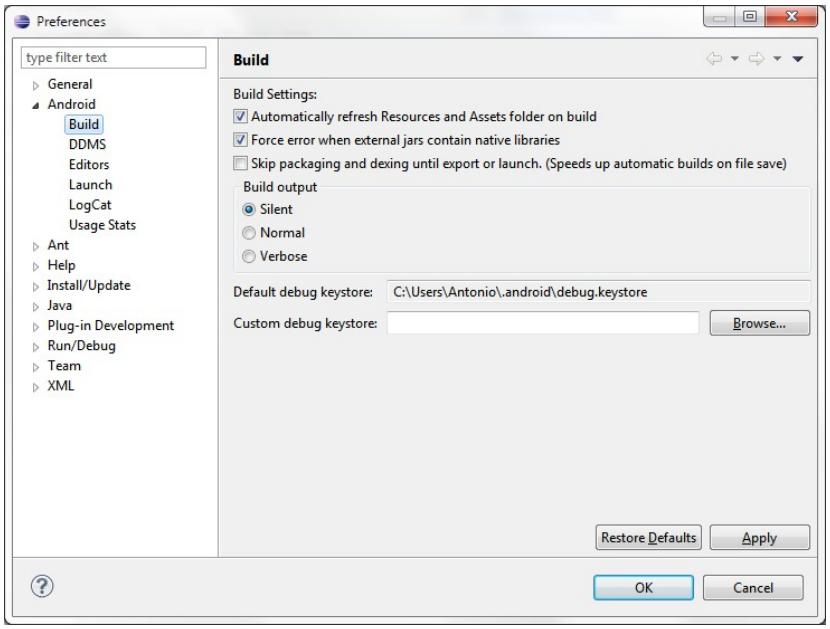

Figura 7.5: Obtención de la clave para la `Android Maps API` (II)

Una vez conocemos la ruta de nuestro certificado de depuración por defecto, debemos obtener su huella digital mediante el siguiente comando, en la línea de comandos:

`keytool -list -alias androiddebugkey -keystore ‘‘`Default debug keystore path`’’ -storepass android -keypass android`

Por último copiamos dicha huella digital que hemos obtenido en la web de solicitud de Google, tal y como observamos en la figura 7.6.

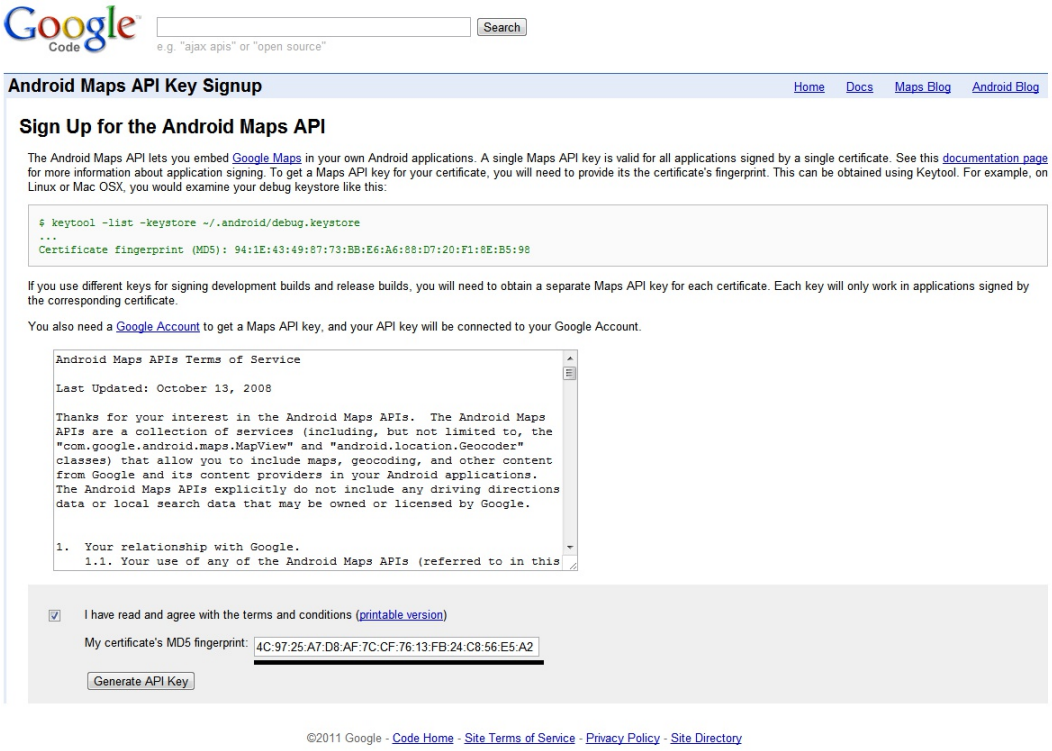

Figura 7.6: Obtención de la clave para la `Android Maps API` (III)

En ese momento ya podemos solicitar a Google que nos genere una clave para poder utilizar la `Android Maps API`, tal y como se muestra en la figura 7.7.

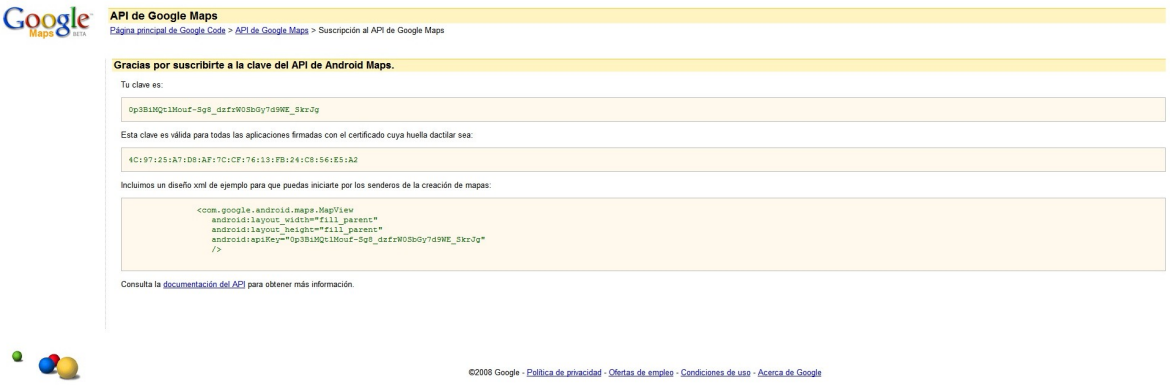

Figura 7.7: Obtención de la clave para la `Android Maps API` (IV)

7.2. Aplicación `TestMapas`

7.2.1. Clase `MapView`

En primer lugar, la vista `MapView`[1] nos permite mostrar y gestionar un mapa dentro del *layout* de nuestra aplicación. Sin embargo, puesto que Eclipse no lo incluye en la vista Graphical Layout, deberemos editar directamente el fichero `main.xml`, tal y como se muestra a continuación:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 ...
 <com.google.android.maps.MapView
   android:id="@+id/mapa"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:apiKey="0p3BiMQtlMouf-Sg8_dzfrW0SbGy7d9WE_SkrJg"
   android:clickable="true" />
</LinearLayout>
```

La propiedad `apiKey` se actualiza con el clave que hemos obtenido anteriormente en la sección 7.1.1, y añadimos además la propiedad `clickable` para poder interactuar con el mapa (por ejemplo desplazarnos).

Por otra parte, la clase `MapView` también incluye los siguientes métodos para controlar el tipo de mapa que queramos mostrar.

```
setSatellite(boolean on)
setStreetView(boolean on)
setTraffic(boolean on)
```

Podemos igualmente consultar el tipo de mapa que en ese momento está desplegado mediante los siguientes métodos

```
isSatellite()
isStreetView()
isTraffic()
```

7.2.2. Clase `MapController`

Todos los mapas en android cuentan con un controlador que nos permite, entre otras acciones, realizar zoom y enfoque sobre los mismos. El método `getController()` de la clase `MapView` nos devolverá el controlador de dicho mapa, representado en un objeto de la clase `MapController`[2]. Algunos de los métodos más interesantes con los que cuenta esta clase son los siguientes.

```
setCenter(GeoPoint point)
setZoom(int zoomLevel)
```

También es posible el paso de una localización a otra en el mapa mediante efectos gráficos que pueden resultar muy interesantes, haciendo uso de los siguientes métodos.

```
animateTo(GeoPoint point)
zoomIn()
```

[1] http://code.google.com/intl/es-ES/android/add-ons/google-apis/reference/com/google/android/maps/MapView.html
[2] http://code.google.com/intl/es-ES/android/add-ons/google-apis/reference/com/google/android/maps/MapController.html

7.2.3. Clase MapActivity

Si queremos que nuestra aplicación haga uso de los mapas de Google, es necesario que nuestra actividad principal extienda la clase abstracta `MapActivity`[3]. Dicha clase contiene el método abstracto `isRouteDisplayed()` que deberemos implementar y que devolverá un booleano que tomará el valor `true` si vamos a representar algún tipo de información de ruta sobre el mapa y `false` en caso contrario.

Así, la clase `TestMapasActivity` tendrá el siguiente aspecto (donde el método de la clase `MapView setBuiltInZoomControls()` nos permitirá mostrar u ocultar los controles habituales de zoom sobre el mapa):

```
public class TestMapasActivity extends MapActivity {

    private MapView mapa = null;

    public void onCreate(Bundle savedInstanceState) {
        ...
        mapa = (MapView)findViewById(R.id.mapa);
        mapa.setBuiltInZoomControls(true);
    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

7.2.4. Manifiesto de la aplicación TestMapas

Por último, y antes de poder ejecutar nuestra aplicación satisfactoriamente en el emulador, debemos modificar el manifiesto de nuestra aplicación, de modo que le concedamos a la misma los permisos de `INTERNET` y `ACCESS_FINE_LOCATION`, tal y como se observa en la figura 7.8.

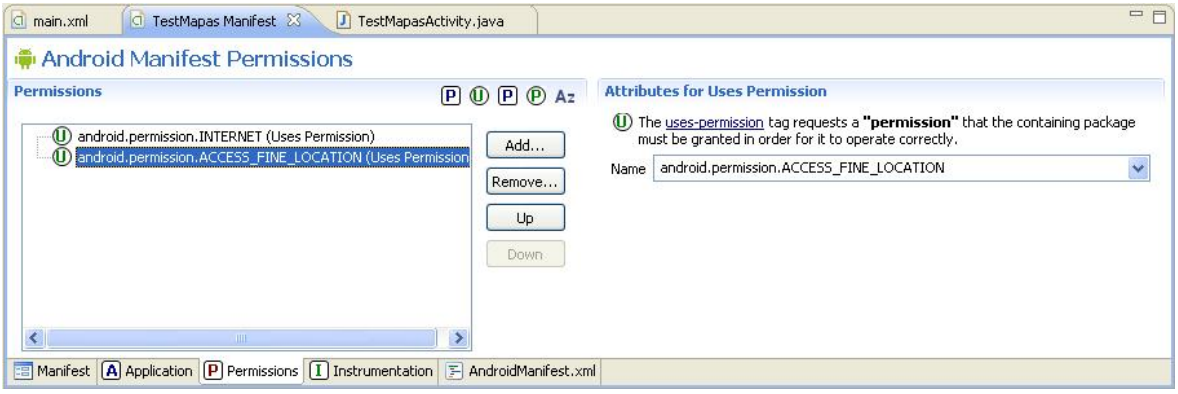

Figura 7.8: Permisos `INTERNET` y `ACCESS_FINE_LOCATION` en el manifiesto de `TestMapas`

Por otra parte, en la pestaña Application del manifiesto, en la sección de Application Nodes, debemos indicar que nuestra aplicación hará uso de la librería de mapas de Google, tal y como se indica en la figura 7.9.

[3] http://code.google.com/intl/es-ES/android/add-ons/google-apis/reference/com/google/android/maps/MapActivity.html

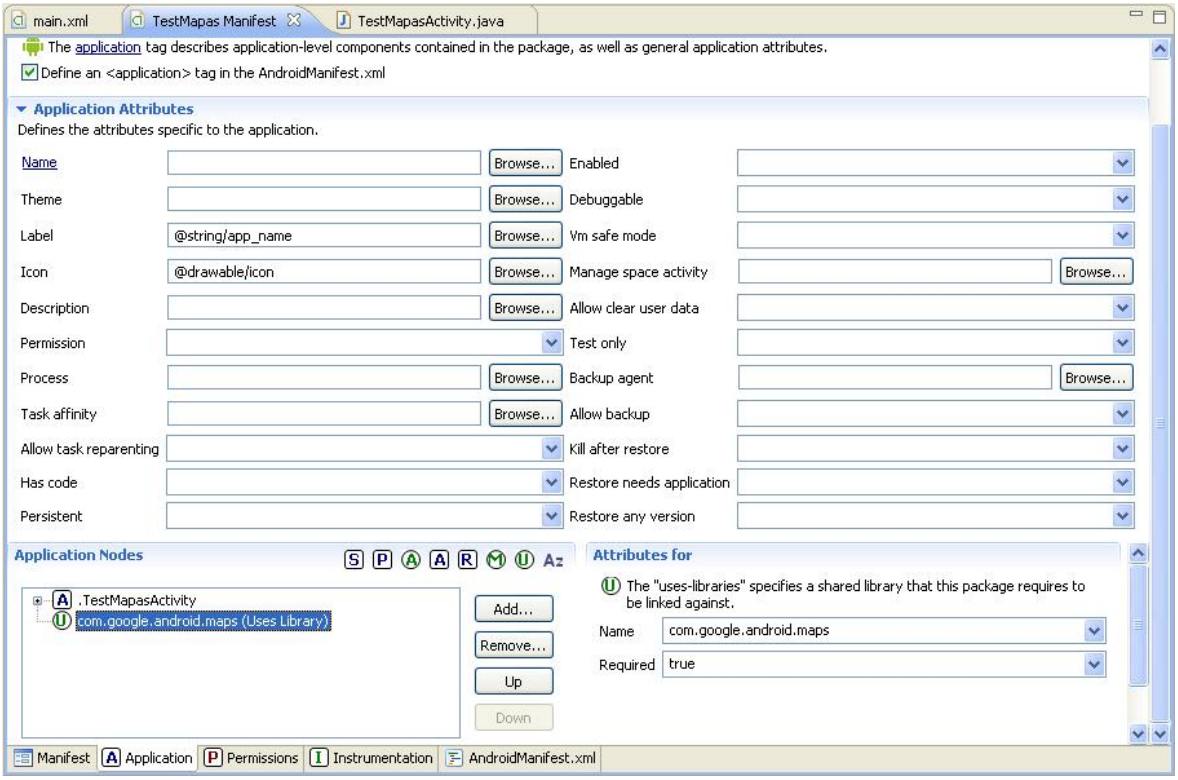

Figura 7.9: Librería `com.android.google.maps` en el manifiesto de `TestMapas`

Una vez seguidos todos estos pasos, podemos ejecutar nuestra aplicación `TestMapas`, obteniendo un resultado como el que se muestra en la figura 7.10.

Figura 7.10: Aplicación `TestMapas`

Actividad propuesta II.17. *Desarrollar y probar la aplicación* `TestMapas`*, tal y como se ha descrito anteriormente.*

Actividad propuesta II.18. *Mejorar la aplicación* `TestMapas` *para añadir un botón que cambie alternativamente el tipo de mapa y otro botón que centre el mapa en Murcia (latitud 38.01*1E6, longitud -1.13*1E6).*

Actividad propuesta II.19. *Modificar la aplicación desarrollada en la actividad II.18. para que el centrado del mapa en Murcia se haga de forma animada.*

7.3. Posicionamiento mediante GPS

7.3.1. Permisos

En primer lugar, para que nuestra aplicación pueda conocer la localización del dispositivo Android sobre el que se ejecuta, necesitará disponer del permiso `ACCESS_FINE_LOCATION`. Puesto que vamos a integrar dicha localización con la `Android Maps API`, necesitaremos también el permiso `INTERNET`, tal y como se muestra en la figura 7.11.

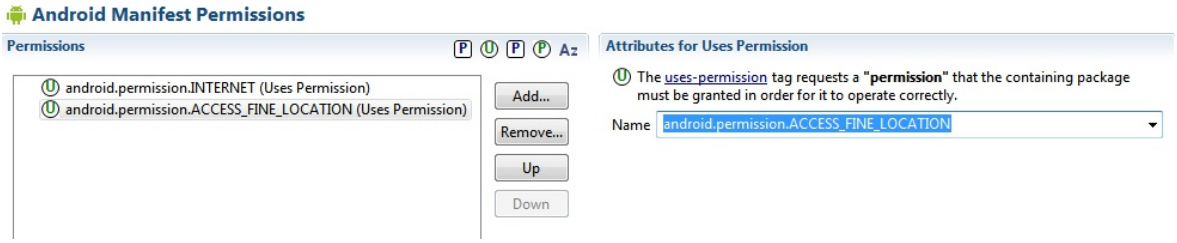

Figura 7.11: Permisos `INTERNET` y `ACCESS_FINE_LOCATION` en el manifiesto de `TestGPS`

7.3.2. Interfaz `LocationListener`

El siguiente paso consistirá en la implementación de la interfaz `LocationListener`[4]. Dicha interfaz contiene los siguientes cuatro métodos.

```
public void onLocationChanged(Location location) {
  updateLocation(location);
}

public void onProviderDisabled(String provider) {
  Intent intent =
     new Intent(android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
  startActivity(intent);
}

public void onProviderEnabled(String provider) { }

public void onStatusChanged(String provider, int status, Bundle extras) { }
```

El método `updateLocation()`, referenciado dentro del método `onLocationChanged()`, lo veremos más adelante en la sección 7.3.5, y como podemos imaginar, nos permitirá actualizar nuestra posición actual en el mapa .

[4] http://developer.android.com/reference/android/location/LocationListener.html

En cuanto al método `onProviderDisabled()`, como podemos observar lo que se hace es activar, mediante un *intent*, el funcionamiento del GPS, para que nuestra aplicación pueda conocer la localización del dispositivo sobre el que se ejecuta.

7.3.3. Clase `Overlay`

La clase `Overlay`[5] nos permitirá superponer capas encima de nuestros mapas. Esto será muy útil, por ejemplo, para ir dibujando y superponiendo un marcardor que indique nuestra posición en cada instante.

Para ello crearemos nuestra propia clase `MyOverlay`, que heredará de `Overlay`, y que sobreescribirá su método `draw()`, tal y como podemos observar a continuación.

```
public class MyOverlay extends Overlay {

 private GeoPoint point;
 private Resources resources;

 public MyOverlay(GeoPoint point, Resources resources) {
   super();
   this.point = point;
   this.resources = resources;
 }

 public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
   super.draw(canvas, mapView, shadow);

   Point scrnPoint = new Point();
   mapView.getProjection().toPixels(this.point, scrnPoint);
   Bitmap marker = BitmapFactory.decodeResource(resources, R.drawable.icon);
   canvas.drawBitmap(marker, scrnPoint.x - marker.getWidth() / 2,
                             scrnPoint.y - marker.getHeight() / 2, null);
   return true;
 }
```

Como se puede observar, el cometido del método `draw()` no es otro que el de convertir el punto que representa la localización del dispositivo (mediante la clase `GeoPoint`[6]) en un punto en el mapa (mediante los métodos `getProjection()` y `toPixels()`). A continuación se crea un marcardor a partir de un recurso de la aplicación y se dibuja éste en el punto del mapa obtenido anteriormente.

[5] http://code.google.com/intl/es-ES/android/add-ons/google-apis/reference/com/google/android/maps/Overlay.html
[6] http://code.google.com/intl/es-ES/android/add-ons/google-apis/reference/com/google/android/maps/GeoPoint.html

7.3.4. Clase `LocationManager`

Por último, la clase `LocationManager`[7] nos permitirá conocer la posición concreta del dispositivo sobre el que se ejecuta nuestra aplicación (mediante el método `getLastKnownLocation()`), así como establecer la frecuencia con la que se actualizará la posición del marcador en el mapa (mediante el método `requestLocationUpdates()`).

Por lo tanto, nuestra clase `TestGPSActivity` tendrá el siguiente aspecto:

```
public class TestGPSActivity extends MapActivity implements LocationListener {
  ...
  public void onCreate(Bundle savedInstanceState) {
    ...
    LocationManager locationManager =
              (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    updateLocation(
       locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER));
    locationManager.requestLocationUpdates(
                            LocationManager.GPS_PROVIDER, 6000, 50, this);
    ...
  }
  ...
}
```

7.3.5. Aplicación `TestGPS`

Finalmente, para que nuestra aplicación `TestGPS` esté del todo completa, solamente nos faltará implementar el método `updateLocation()`, tal y como se observa a continuación.

```
protected void updateLocation(Location location){
  GeoPoint point = new GeoPoint((int) (location.getLatitude() * 1E6),
                                (int) (location.getLongitude() * 1E6));
  mapController.animateTo(point);
  mapController.setZoom(15);

  List<Overlay> mapOverlays = mapView.getOverlays();
  MyOverlay marker = new MyOverlay(point, getResources());
  mapOverlays.add(marker);
  mapView.invalidate();
}
```

Como podemos comprobar, este método actualiza la posición central del mapa con la localización recibida como parámetro, al tiempo que añade una nueva capa a nuestro mapa, en forma de `Overlay`, de manera que el marcador de posición también se actualice con la posición actual del dispositivo.

Actividad propuesta II.20. *Desarrollar y probar la aplicación `TestGPS`, tal y como se ha descrito anteriormente.*

[7] http://developer.android.com/reference/android/location/LocationManager.html

Tema 8

Actividad final

Nota.- La entrega de la actividad final deberá ajustarse a lo indicado en la sección 5 de Metodología y Evaluación, en capítulo de Introducción de estos apuntes (página 19).

8.1. Descripción

Con todos los conocimientos adquiridos a lo largo del curso y todas las posibilidades de Android que se han mostrado, se propone al alumno realizar una aplicación que haga uso de buena parte de ellas.

A continuación se proponen algunas opciones, aunque si el alumno así lo desea, puede realizar su propia aplicación consultándolo **previamente** con los profesores (`felixgm@um.es` y `antonio.bernardez@um.es`).

- **Propuesta 1**

Diseñar e implementar una aplicación sencilla que sirva como agenda de contactos, guardando dichos contactos en el sistema de ficheros.

- **Propuesta 2**

Diseñar e implementar una aplicación sencilla que sirva como agenda de contactos, guardando dichos contactos como una base de datos.

- **Propuesta 3**

Diseñar e implementar una aplicación sencilla que sirva como mediateca, guardando imágenes, música y vídeos en el sistema de ficheros.

- **Propuesta 4**

Diseñar e implementar una aplicación sencilla que sirva para jugar al Sudoku, y que descargue los tableros de un servidor a través de HTTP.

- **Propuesta 5**

Diseñar e implementar una aplicación sencilla que lea de un fichero una lista de números de teléfono asociados cada uno de ellos con un nombre (pares del tipo <teléfono, nombre>) y envíe un SMS de publicidad a cada uno de esos números de teléfono.

Apéndice A

Protocolo Bluetooth

A.1. Arquitectura Bluetooth

La unidad básica de un sistema Bluetooth [**?**] es una *piconet*, que consta de un nodo maestro (*master*) y hasta siete nodos esclavos (*slave*) activos a una distancia de 10 metros. Varias *piconets* pueden conectarse a través de nodos puente, como se muestra en la figura A.1, formando lo que se denomina una *scatternet*.

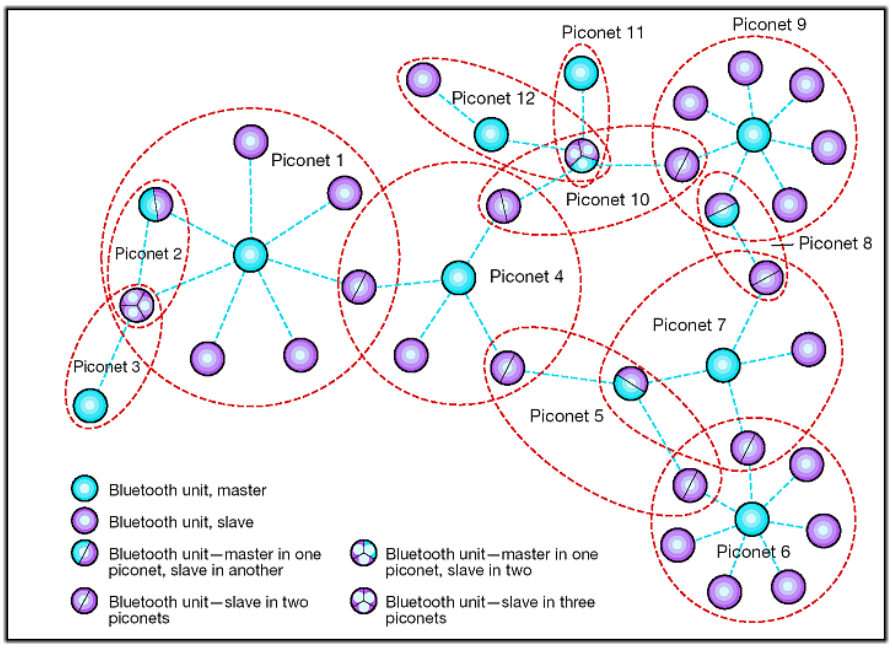

Figura A.1: *Scatternet* formada por 12 *piconets*

Además de los siete nodos esclavos activos de una *piconet*, puede haber hasta 255 nodos estacionarios en la red. Éstos son dispositivos que el maestro ha cambiado a un estado de bajo consumo para reducir el desgaste innecesario de sus baterías. Lo único que un dispositivo en estado estacionario puede hacer es responder a una señal de activación por parte del maestro.

En esencia, una *piconet* es un sistema TDM centralizado, en el cual el maestro controla el reloj y determina qué dispositivo se comunica en un momento determinado. Todas las comunicaciones se realizan entre el maestro y el esclavo; no existe comunicación directa de esclavo a esclavo.

A.2. Perfiles de Bluetooth

Para poder utilizar Bluetooth, un dispositivo debe ser compatible con ciertos perfiles de Bluetooth [32]. El SIG de Bluetooth define y adopta los perfiles mostrados en la tabla A.1.

Perfil	**Descripción**
Advanced Audio Distribution Profile	Transferencia de un flujo de audio estéreo
Audio/Video Remote Control Profile	Interfaz estándar para control remoto de TVs, Hi-Fi,...
Basic Imaging Profile	Envío de imágenes
Basic Printing Profile	Envío de texto, e-mails, ... a impresoras
Common ISDN Access Profile	Acceso a servicios, datos y señales ofrecidas por un ISDN
Cordless Telephony Profile	Uso de teléfonos inalámbricos con Bluetooth
Device ID Profile	Identificación de un dispositivo
Dial-up Networking Profile	Acceso a internet y otros servicios de marcación sobre Bluetooth
Fax Profile	Interfaz entre un teléfono móvil y un PC con software para FAX
File Transfer Profile	Acceso al sistema de ficheros de otro dispositivo
General Audio/Video Distribution Profile	Es la base para A2DP y VDP
Generic Access Profile	Es la base para todos los demás perfiles
Generic Object Exchange Profile	Es la base para otros perfiles de transferencia de datos
Hard Copy Cable Replacement Profile	Alternativa inalámbrica a la conexión por cable entre un dispositivo y una impresora
Hands-Free Profile	Comunicación entre los kits manos libres de los coches y los teléfonos móviles
Human Interface Device Profile	Soporte para dispositivos como ratones, joysticks, teclados, etc.
Headset Profile	Soporte para auriculares Bluetooth usados con teléfonos móviles
Intercom Profile	Permite llamadas de voz entre dos auriculares Bluetooth
Objet Push Profile	Envío de "objetos" como imágenes, citas, etc.
Personal Area Networking Profile	Permite el uso del protocolo de encapsulación de red de Bluetooth
Phone Book Access Profile	Intercambio de objetos de la agenda entre dispositivos
Serial Port Profile	Emulación de una conexión a través de un cable serie
Service Discovery Application Profile	Permite averiguar los perfiles ofrecidos por un dispositivo
SIM Access Profile	Conexión entre dispositivos como teléfonos de coche y módulos SIM en teléfonos con Bluetooth
Synchronization Profile	Sincronización de elementos del PIM
Video Distribution Profile	Transporte de un flujo de vídeo
Wireless Application Protocol Bearer	Soporte de WAP sobre PPP sobre Bluetooth

Tabla A.1: Perfiles de Bluetooth

A.3. La pila de protocolos de Bluetooth

En la figura A.2 se observa la pila de protocolos de Bluetooth.

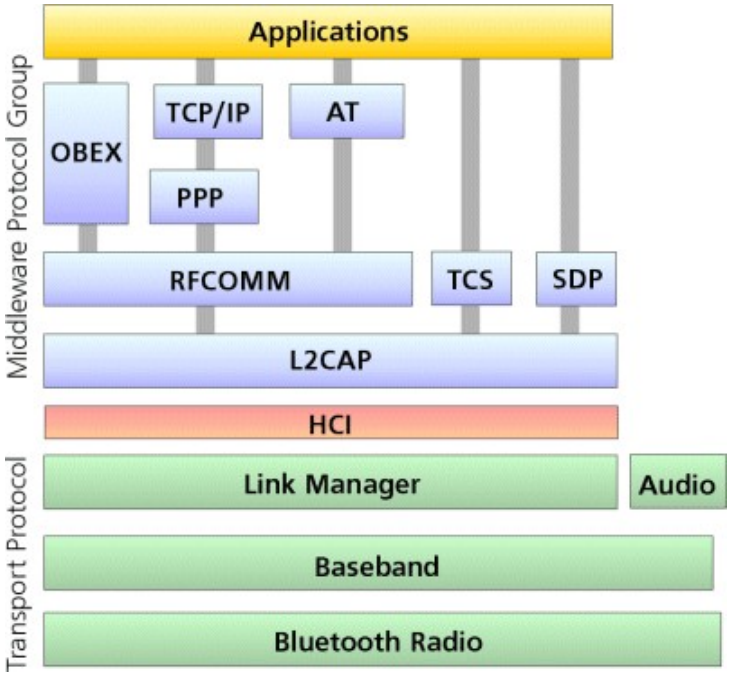

Figura A.2: Pila de protocolos de Bluetooth

A continuación describiremos las distintas capas y protocolos que forman la pila de protocolos de Bluetooth [**?**, 33].

A.3.1. La capa de radio

La capa de radio define los requisitos para un transmisor-receptor de radio Bluetooth, el cual opera en la banda de los 2'4 GHz. Define los niveles de sensibilidad de dicho transmisor-receptor, establece los requisitos para utilizar las frecuencias del espectro expandido y clasifica a los dispositivos Bluetooth en tres clases, según se indica en la tabla A.2.

Clase	**Potencia máxima permitida**		**Potencia mínima permitida**		**Distancia**
	mW	**dBm**	**mW**	**dBm**	
Clase 1	100 mW	20 dBm	1 mW	0 dBm	100 metros
Clase 2	2'5 mW	4 dBm	0'25 mW	-6 dBm	10 metros
Clase 3	1 mW	0 dBm	N/A	N/A	1 metros

Tabla A.2: Clases de dispositivos Bluetooth

A.3.2. La capa de banda base

La capa de banda base representa a la capa física de Bluetooth. Se usa como controladora de enlace, la cual trabaja junto con el *link manager* para llevar a cabo operaciones tales como la creación de conexiones de enlace con otros dispositivos.

Controla el direccionamiento de dispositivos, el control del medio (cómo los dispositivos se buscan unos a otros), operaciones de ahorro de energía, así como control de flujo y sincronización entre dispositivos.

A.3.3. *Link Manager*

El administrador de enlaces o *Link Manager*, se encarga de establecer canales lógicos entre dispositivos, incluyendo administración de energía, autenticación y calidad de servicio.

A.3.4. *Host Controller Interface*

El HCI [34] permite el acceso mediante línea de comandos a la capa de banda base y al LMP para controlar y recibir información acerca del estado. Se compone de tres partes:

1. El *firmware* HCI, o programa oficial del fabricante, actualmente forma parte del hardware Bluetooth.
2. El controlador HCI, o *driver*, que se encuentra en el software del dispositivo Bluetooth.
3. El *Host Controller Transport Layer*, que conecta el *firmware* con el *driver*.

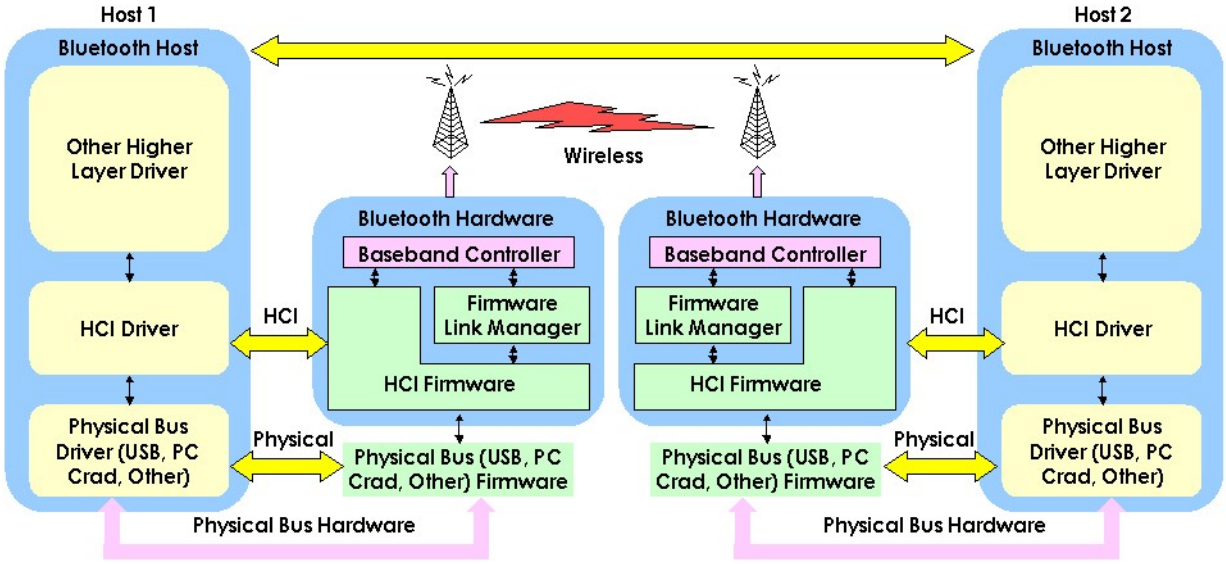

Figura A.3: *Host Controller Interface*

A.3.5. Protocolo de adaptación y control de enlaces lógicos L2CAP

El protocolo L2CAP aísla a las capas superiores de los detalles de transmisión. Es análogo a la subcapa LLC del estándar 802, pero difiere de ésta en el aspecto técnico. Proporciona servicios orientados y no orientados a la conexión a las capas superiores.

L2CAP permite a los protocolos de alto nivel y a las aplicaciones enviar y recibir paquetes de datos de hasta 64 Kb. Buena parte de su tiempo la dedica a la segmentación y reemsablado.

A.3.6. RFCOMM

RFCOMM es el protocolo que emula el puerto serie estándar RS232 de los ordenadores para la conexión de teclados, ratones y módems, entre otros dispositivos. Su propósito es que dichos dispositivos no tenga por qué saber nada acerca de Bluetooth.

A.3.7. TCS y SDP

El protocolo de telefonía TCS es un protocolo de tiempo real y está destinado a los tres perfiles orientados a voz (HFP, HSP e ICP). También se encarga del establecimiento y terminación de llamadas. Por su parte, el protocolo de descubrimiento de servicios (SDP) se emplea para la detección automática de dispositivos dentro de la red, así como de servicios ofrecidos por dichos dispositivos.

A.4. La capa de radio de Bluetooth

La capa de radio transfiere los bits del maestro al esclavo o viceversa. Es un sistema de baja potencia con un rango de entre 1 y 100 metros que opera en la banda ISM de los 2'4 GHz. La banda se divide en 79 canales de 1 MHz cada uno. La modulación es por división en frecuencia, con 1 bit por Hz, lo cual da una tasa de datos aproximada de 1 Mbps, pero gran parte de este espectro la consume la sobrecarga.

Para asignar los canales de manera equitativa, el espectro de saltos de frecuencia se utiliza a 1600 saltos por segundo y un tiempo de permanencia de 625 μseg. Todos los nodos de una *piconet* saltan de manera simultánea, y el maestro establece la secuencia de salto.

Debido a que tanto el 802.11 como Bluetooth operan en la banda ISM de 2'4 GHz en los mismos 79 canales, interfieren entre sí. Puesto que Bluetooth salta mucho más rápido que 802.11, es más probable que un dispositivo Bluetooth dañe las transmisiones del 802.11 que en el caso contrario.

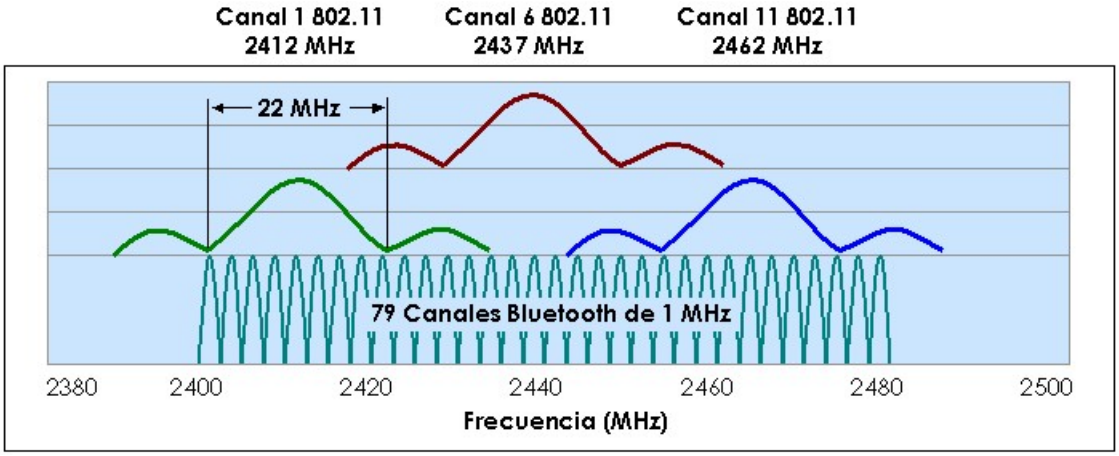

Figura A.4: Coexistencia de Bluetooth y 802.11 en la banda de los 2'4 GHz

A.5. La capa de banda base de Bluetooth

La capa de banda base de Bluetooth es lo más parecido a una subcapa MAC. Esta capa convierte el flujo de bits puros en tramas. En la forma más sencilla, el maestro de cada *piconet* define una serie de ranuras de tiempo de 625 μseg y las transmisiones del maestro empiezan en las ranuras pares, y las de los esclavos en las impares. Ésta es la tradicional multiplexación por división de tiempo, en la cual el maestro acapara la mitad de las ranuras y los esclavos comparten la otra mitad. Las tramas pueden tener 1, 3 ó 5 ranuras de longitud.

Cada trama se transmite por un canal lógico llamado enlace entre el maestro y un esclavo. Hay dos tipos de enlaces:

- ACL, que se utiliza para datos conmutados en paquetes disponibles a intervalos irregulares. Estos datos provienen de la capa L2CAP en el nodo emisor y se entregan en la capa L2CAP en el nodo receptor. No hay garantías de entrega del tráfico ACL. Las tramas se pueden perder y tienen que retransmitirse. Un esclavo sólo puede tener un enlace ACL con su maestro.

- SCO, para datos en tiempo real, como ocurre en las conexiones telefónicas. A este tipo de canal se le asigna una ranura fija en cada dirección. Las tramas que se envían a través de SCO nunca se retransmiten. En vez de ello se puede usar la corrección de errores hacia adelante para conferir una alta fiabilidad al sistema. Un esclavo puede establecer hasta tres enlaces SCO con su maestro. Cada enlace de este tipo puede transmitir un canal de audio PCM de 64000 bps.

A.6. La capa L2CAP de Bluetooth

La capa L2CAP tiene tres funciones principales:

1. Acepta paquetes de hasta 64 KB provenientes de las capas superiores y los divide en tramas para transmitirlos. Las tramas se reensamblan nuevamente en paquetes en el otro extremo.

2. Maneja la multiplexación y desmultiplexación de múltiples fuentes de paquetes. Cuando se reensambla un paquete, la capa L2CAP determina qué protocolo de las capas superiores lo manejará, por ejemplo, RFCOMM o el de telefonía.

3. Se encarga de la calidad de servicio, tanto al establecer los enlaces como durante la operación normal. Asimismo, durante el establecimiento de los enlaces se negocia el tamaño máximo de carga útil permitido, para evitar que un dispositivo que envíe paquetes grandes sature a uno que recibe paquetes pequeños.

A.7. Estructura de la trama de Bluetooth

En la figura A.5 se muestra el formato de una trama de Bluetooth.

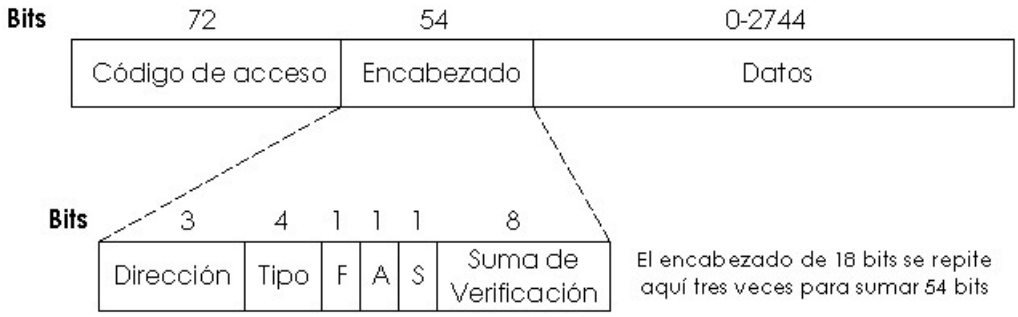

Figura A.5: Estructura de la trama de Bluetooth

Como se puede observar, las tramas comienzan con un código de acceso, el cual identifica al maestro de cada *piconet* y cuyo propósito es que los esclavos que se encuentren en el rango de alcance de dos maestros sepan qué tráfico está destinado a ellos.

A continuación se encuentra un encabezado de 54 bits que contiene campos comunes de la subcapa MAC. Y por último está el campo de datos, de hasta 2744 bits (para una transmisión de cinco ranuras). Para una sola ranura de tiempo, el formato es el mismo excepto que el campo de datos es de 240 bits.

Profundizando un poco en el encabezado, el campo *Dirección* identifica a cuál de los ocho dispositivos activos está destinada la trama. El campo *Tipo* indica el tipo de trama (ACL, SCO, de sondeo o nula), el tipo de corrección de errores que se utiliza en el campo de datos y cuántas ranuras de longitud tiene la trama.

Un esclavo establece el bit *F* (de flujo) cuando su búfer está lleno y no puede recibir más datos. Ésta es una forma primitiva de control de flujo. El bit *A* (de confirmación de recepción o *Acknowledgement*) se utiliza para incorporar un ACK en la trama. Por último, el bit *S* (de secuencia) sirve para numerar las tramas con el propósito de de detectar retransmisiones. El protocolo es de parada y espera, por lo que 1 bit es suficiente.

A continuación se encuentra el encabezado *Suma de verificación* de 8 bits. Todo el encabezado de 18 bits se repite tres veces para formar el encabezado de 54 bits. En el receptor se comprueban las tres copias de cada bit y si coinciden, se acepta. De lo contrario se impone la opinión de la mayoría.

Apéndice B

Acrónimos

3G, 3rd Generation

ADT, Android Development Tools

API, Application Programming Interface

AVD, Android Virtual Device

CDMA, Code Division Multiple Access

DDMS, Dalvik Debug Monitor Server

DPI, Dots Per Inch

EDGE, Enhanced Data Rates for GSM Evolution

FM, Frequency Modulation

FWVGA, Full Wide Video Graphics Array

GSM, Global System for Mobile Communications

GPS, Global Positioning System

HTML, HyperText Markup Language

HVGA, Half-size Video Graphics Array

IDE, Integrated Development Environment

IEEE, Institute of Electrical and Electronics Engineers

IMEI, International Mobile Equipment Identity

IP, Internet Protocol

IPC, Inter-Process Communication

ISO, International Organization for Standardization

JDKC, Java Development Kit

JPG/JPEG, Joint Photographic Experts Group

MCC, Mobile Country Code

MEID, Mobile Equipment Identifier

MMC, Multi Media Card

MNC, Mobile Network Code

MP3, MPEG-2 Audio Layer 3

MPEG, Moving Picture Experts Group

NDK, Native Development Kit

NFC, Near Field Communication

OpenGL, Open Graphics Library

P2P, Peer to Peer

PDA, Personal Digital Assistant

PIN, Personal identification number

PUK, Personal Unlocking Key

QVGA, Quarter Video Graphics Array

SD, Secure Digital

SDK, Software Development Kit

SIM, Subscriber Identity Module

SIP, Session Initiation Protocol

SMS, Short Message Service

SQL, Structured Query Language

VGA, Video Graphics Array

VoIP, Voice over IP

WQVGA, Wide Quarter Video Graphics Array

WVGA, Wide Video Graphics Array

WWW, World Wide Web

Bibliografía

[1] Reto Meier. "*Proffesional Android 2 Application Development*", Wiley Publishing, Inc., 2010, ISBN: 978-0-470-56552-0.

[2] Mark L. Murphy, "*Beginning Android 2*", Apress, , 2010, ISBN 978-1-4302-2629-1.

[3] "*Android Developers.*", Google Inc., 2011.
`http://developer.android.com/index.html`

[4] "*The Developer's Guide*", Google Inc., 2011.
`http://developer.android.com/guide/index.html`

[5] "*Package Index*", Google Inc., 2011.
`http://developer.android.com/reference/packages.html`

[6] "*Developer Resources*", Google Inc., 2011.
`http://developer.android.com/resources/index.html`

[7] "*Android SDK. Android Developers*", Google Inc., 2011.
`http://developer.android.com/sdk/index.html`

[8] "*Android 2.2 Platform Highlights*", Google Inc., 2011.
`http://developer.android.com/sdk/android-2.2-highlights.html`

[9] "*Android 2.0 Platform Highlights*", Google Inc., 2011.
`http://developer.android.com/sdk/android-2.0-highlights.html`

[10] "*Android 1.6 Platform Highlights*", Google Inc., 2011.
`http://developer.android.com/sdk/android-1.6-highlights.html`

[11] "*Android 1.5 Platform Highlights*", Google Inc., 2011.
`http://developer.android.com/sdk/android-1.5-highlights.html`

[12] "*Activity — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/app/Activity.html`

[13] "*LinearLayout — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/LinearLayout.html`

[14] "*TextView — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/TextView.html`

[15] "*The AndroidManifest.xml File*", Google Inc., 2011.
`http://developer.android.com/guide/topics/manifest/manifest-intro.html`

[16] "*R — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/R.html`

[17] "*View — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/view/View.html`

[18] "*View — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/view/ViewGroup.html`

[19] "*Handling UI Events — Android Developers*", Google Inc., 2011.
`http://developer.android.com/guide/topics/ui/ui-events.html`

[20] "*EditText — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/EditText.html`

[21] "*Button — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/Button.html`

[22] "*ImageButton — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/ImageButton.html`

[23] "*CheckBox — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/CheckBox.html`

[24] "*RadioButton — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/RadioButton.html`

[25] "*RadioGroup — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/RadioGroup.html`

[26] "*ImageView — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/ImageView.html`

[27] "*ProgressBar — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/ProgressBar.html`

[28] "*Spinner — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/Spinner.html`

[29] "*Hello Views — Android Developers*", Google Inc., 2011.
`http://developer.android.com/resources/tutorials/views/index.html`

[30] "*Common Layout Objects — Android Developers*", Google Inc., 2011.
`http://developer.android.com/guide/topics/ui/layout-objects.html`

[31] "*FrameLayout — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/FrameLayout.html`

[32] "*LinearLayout — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/LinearLayout.html`

[33] "*List View — Android Developers*", Google Inc., 2011.
`http://developer.android.com/resources/tutorials/views/hello-listview.html`

[34] "*ListView — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/ListView.html`

[35] "*Table Layout — Android Developers*", Google Inc., 2011.
`http://developer.android.com/resources/tutorials/views/hello-tablelayout.html`

[36] "*TableLayout — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/widget/TableLayout.html`

[37] "*Drawable Resources — Android Developers*", Google Inc., 2011.
`http://developer.android.com/guide/topics/resources/drawable-resource.html`

[38] "*Intent — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/content/Intent.html`

[39] "*Menus — Android Developers*", Google Inc., 2011.
`http://developer.android.com/guide/topics/ui/menus.html`

[40] "*Menu — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/view/Menu.html`

[41] "*MenuInflater — Android Developers*", Google Inc., 2011.
`http://developer.android.com/reference/android/view/MenuInflater.html`

[42] "*Providing Resources — Android Developers*", Google Inc., 2011.
`http://developer.android.com/guide/topics/resources/providing-resources.html`

[43] "*Eclipse.org home*", Eclipse Foundation, 2011.
`http://www.eclipse.org`

[44] "*Eclipse Downloads*", Eclipse Foundation, 2011.
`http://www.eclipse.org/downloads/`

[45] "*Eclipse (software) - Wikipedia, la enciclopedia libre*", Wikimedia Foundation Inc., 2011.
`http://es.wikipedia.org/wiki/Eclipse_(software)`

[46] "*Android - Wikipedia, la enciclopedia libre*", 2011.
`http://es.wikipedia.org/wiki/Android`

[47] Oracle Inc. "*Java SE Downloads*", 2011.
`http://www.oracle.com/technetwork/java/javase/downloads/index.html`