Android

Aplicaciones Móviles
Curso de Adaptación
Grado en Ingeniería de Sistemas Audiovisuales

Celeste Campo - Carlos García Rubio
celeste, cgr@it.uc3m.es

Índice

Introducción

- Solución software completa para dispositivos móviles.
- Engloba:
  - Sistema operativo (basado en Linux).
  - Entorno de ejecución basado en Java.
  - Librerías de bajo y medio nivel.
  - Conjunto inicial de aplicaciones para el usuario final.

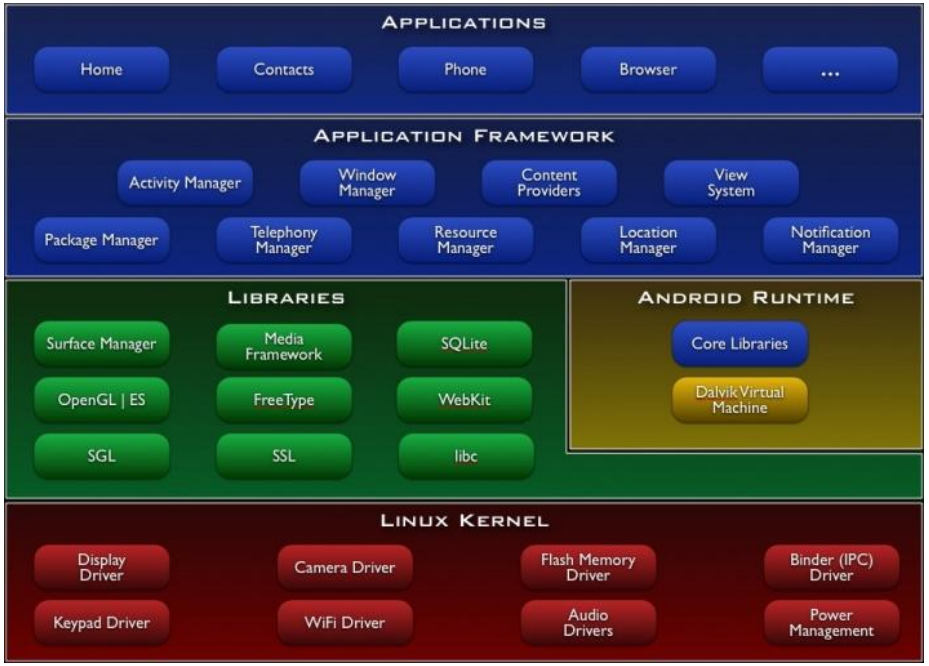


Imagen obtenida de Developer Android (http://developer.android.com/)

Introducción

- ¿Qué es Android?
  - Plataforma abierta para desarrollar aplicaciones móviles.
- ¿Qué no es Android?
  - Una implementación de Java ME.
  - Parte de Linux Phone Standards Forum (LiPS) o de Open Mobile Alliance (OMA).
  - Una capa de aplicación (S60 o UIQ).
  - Un teléfono móvil.
  - La respuesta de Google al iPhone.

Componentes

- Las aplicaciones en Android están basadas en componentes.
- Existen cuatro tipos de componentes:
  - *Activities*
  - *Services*
  - *Broadcast receivers*
  - *Content providers*
- Una aplicación Android será siempre una combinación de uno o más de estos componentes:
  - Deberán ser declarados de forma explícita en el fichero `AndroidManifiest.xml`

Componentes

- *Activity*:
  - Modela una actividad llevada a cabo por una aplicación.
  - Normalmente, lleva asociado una ventana o elemento de interfaz gráfica, pero no se asocia sólo con él.
  - Una aplicación puede tener una o varias actividades:
    - Existe una actividad inicial que es la que se le presenta al usuario cuando se arranca la aplicación.
  - Se implementan como subclase de la clase `Activity`.
  - Ej. Listar en pantalla los contactos de la agenda.
- *Service*:
  - Modela una actividad ejecutada en segundo plano sin interfaz gráfico.
  - Se implementa como subclase de la clase `Service`.
  - Ej. Reproducir música obtenida/seleccionada por una actividad.

Componentes

- *Broadcast receiver*:
  - No realizan ninguna acción, sólo reciben y reaccionan ante anuncios enviados por difusión.
  - Normalmente reaccionan lanzando alguna actividad que procese la información recibida.
  - La mayoría de anuncios por difusión son generados por el propio sistema (zona horaria ha cambiado, la batería está baja, …)
  - Las aplicaciones pueden generar sus propios anuncios y difundirlos para avisar a otras aplicaciones.
  - Extienden la clase `BroadcastReceiver`.
  - Ej. Una llamada entrante, recepción de un SMS.
- *Content provider*:
  - Modela un conjunto de datos de una aplicación que pueden estar disponibles para otras aplicaciones.
  - Los datos pueden estar almacenados en el sistema de ficheros, en una base de datos, o cualquier otro almacenamiento.
  - Las aplicaciones no interacción directamente con ellos deben manejarlos a través de `ContentResolver`.
  - Extiende la clase `ContentProvider`.

Componentes

- Activación de componentes:
  - *Activities*, *services* y *broadcast receivers* se activan con *Intents*:
    - Los *Intents* modelan mensajes asíncronos (clase `Intent`).
      - Para *activities* y *services* nos indican la acción solicitada y una URI indicando dónde están los datos a los que se le aplica la acción.
      - Para *broadcast receivers* nos indica la acción que debe ser anunciada.
  - *Content provider* se activan a través de `ContentResolver`.
- Parada de componentes:
  - Los *content providers* y *broadcast receivers* tienen por definición un tiempo limitado de actividad, no necesitan pararse explícitamente.
  - Android proporciona métodos para parar de forma ordenada *activities* y *services* que están en ejecución.
  - Cualquier componente puede ser finalizado por el sistema cuando se detecta que no va a utilizarse o se necesitan recursos para activar nuevos componentes.

Componentes: Fichero manifiesto

- El fichero manifiesto (`AndroidManifest.xml`):
  - Fichero XML que debe existir en toda aplicación Android y nos indica:
    - Componentes de la aplicación y relaciones entre ellos.
    - Bibliotecas necesarias (además de las de Android).
    - Permisos que necesita la aplicación para su ejecución.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

Componentes: Fichero manifiesto

- Algunos elementos del manifiesto:
  - Nivel 1: elemento `manifest`:
    - Atributo `package`: nombre del paquete java que es la base de la aplicación.
  - Nivel 2:
    - Elemento `uses-permission`: permisos que necesita la aplicación para funcionar correctamente.
    - Elemento `permission`: permisos de seguridad para limitar el acceso a componentes específicos de la aplicación.
    - Elemento `instrumentation`: permite monitorizar la interacción de la aplicación con el sistema.
    - Elemento `application`: declaración de la aplicación:
      - Elemento `activity` (activities).
      - Elemento `service` (services).
      - Elemento `receiver` (broadcast receivers).
      - Elemento `provider` (content providers).

Componentes: Ciclo de vida

- En Android cada aplicación se ejecuta en su propio proceso:
  - Beneficios de seguridad, gestión de memoria, CPU.
  - El sistema es el que gestiona estos procesos de la forma más eficiente posible.
    - El tiempo de vida de un proceso no está controlado por la aplicación, por lo tanto, por el programador.
- En Android las *activities*, *services* y *broadcast receivers* tienen sus propios ciclos de vida.

Componentes: Ciclo de vida

- Ciclo de vida *Activity*:
  - Una *Activity* tiene tres estados:
    - **Activa o en ejecución**: está en primer plano y tiene el foco de interacción con el usuario.
    - **Pausada**: ha perdido el foco pero sigue siendo visible para el usuario. Sigue siendo una actividad "viva" pero podría ser eliminada por el sistema en cualquier momento (normalmente, situaciones extremas de baja memoria).
    - **Parada**: está totalmente oculta por otra actividad. Se mantiene información de su estado por si es necesario recuperarla. El sistema frecuentemente elimina estas actividades para liberar memoria del sistema.
  - Cuando una *Activity* cambia de estado se le informa invocando los siguientes métodos:
    - `void onCreate(Bundle savedInstanceState)`
    - `void onStart()`
    - `void onRestart()`
    - `void onResume()`
    - `void onPause()`
    - `void onStop()`
    - `void onDestroy()`

Componentes: Ciclo de vida

- `onCreate()`, `onDestroy()`:
  - Abarcan todo el ciclo de vida.
  - Representan el principio y fin de la activity.
- `onStart()`, `onStop()`:
  - Representan la parte visible del ciclo de vida.
  - Es visible aunque puede no tener el foco de acción con el usuario.
  - Pueden invocarse múltiples veces.
- `onResume()`, `onPause()`:
  - Abarcan la parte útil.
  - Es visible y además tiene el foco de acción con el usuario.

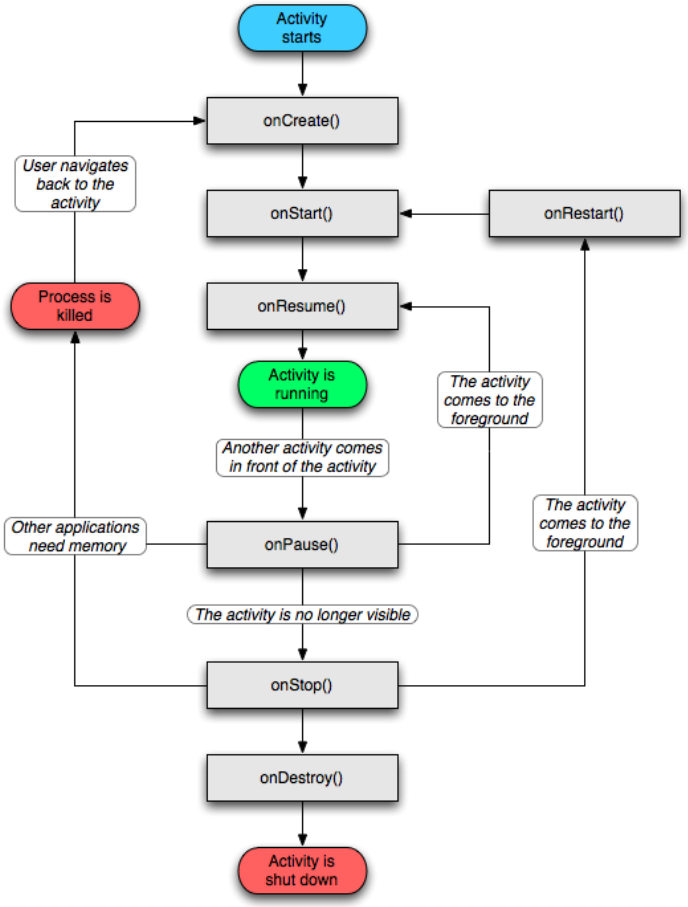


Imagen obtenida de Developer Android (http://developer.android.com/)

Componentes: Ciclo de vida

- Ciclo de vida *Service*:
  - Un *Service* puede ser utilizado de dos formas:
    - Arrancando y parándolo según se necesite.
    - Conectándose y desconectándose del servicio cuando se necesite.
  - Cuando una *Service* cambia de estado se le informa invocando los siguientes métodos:
    - `void onCreate()`
    - `void onStart(Intent intent)`
    - `void onDestroy()`

Componentes: Ciclo de vida

- `onCreate()`, `onDestroy()`:
  - Abarcan todo el ciclo de vida.
- `onStart()`:
  - El servicio está activo y en ejecución.
- `onBind()`, `onUnbind()`, `onRebind()`:
  - Permite conectarse al servicio estableciendo un canal de comunicación.
  - Independientemente de cómo se haya lanzado el servicio un cliente puede conectarse a él.

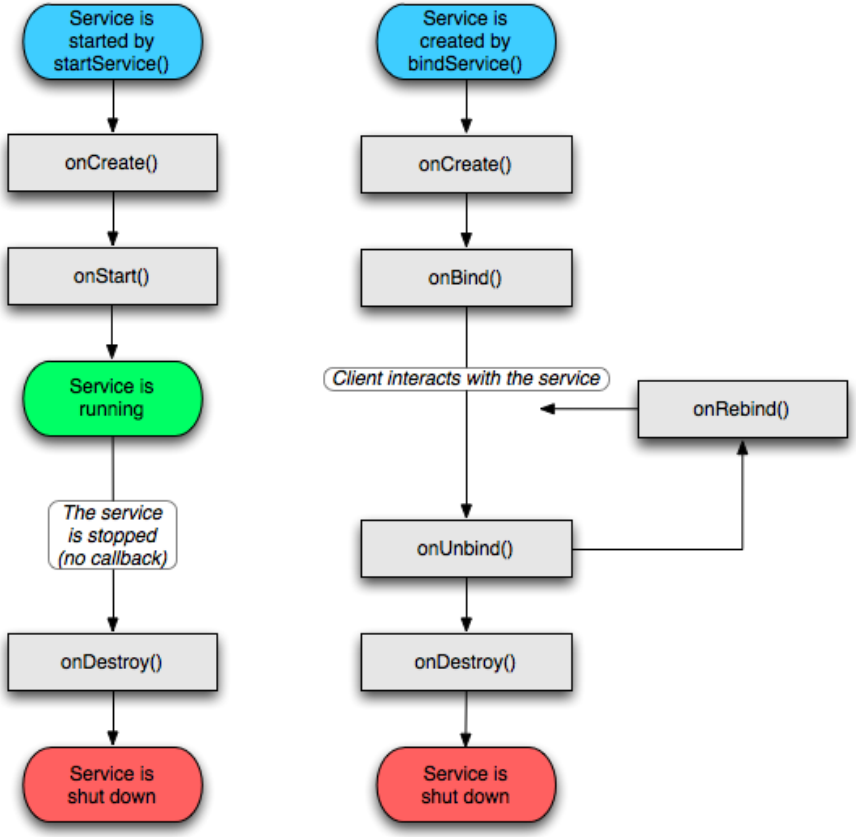
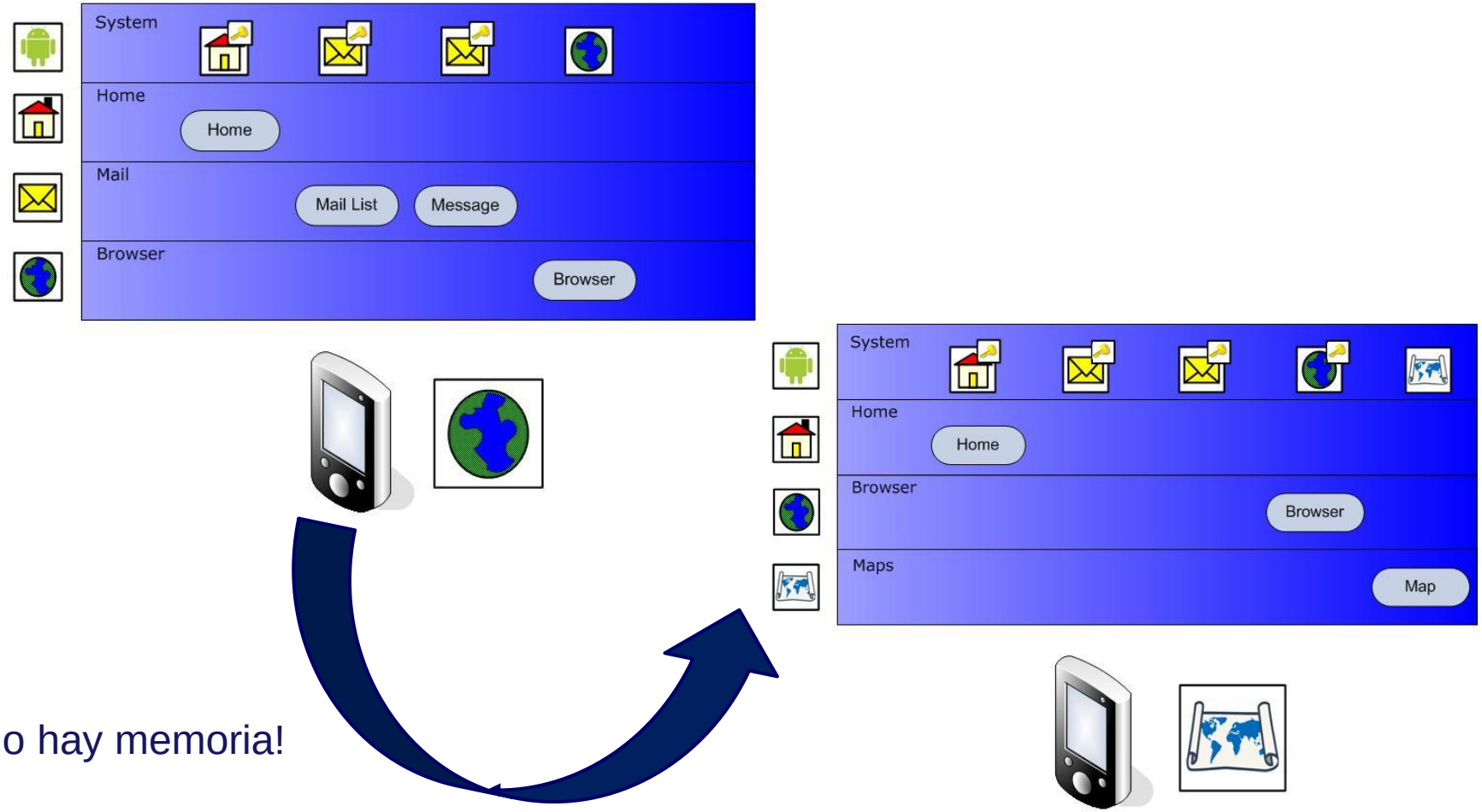


Imagen obtenida de Developer Android (http://developer.android.com/)

Componentes: Ciclo de vida

- Ciclo de vida *Broadcast receiver*:
  - Sólo tiene un método:
    - `void onReceive(Context curContext, Intent broadcastMsg)`
  - Es invocado por el sistema cuando se recibe un mensaje.
  - Se considera que un *broadcast receiver* está activo mientras se está ejecutando este método, cuando se sale de él se considera que ya no está en ejecución.

Componentes: Ciclo de vida



¡No hay memoria!

Imagen obtenida del PFC de Jaime Aranaz Tudela

Componentes: Ciclo de vida

- Política de eliminación de procesos:
  - No usar los componentes de una aplicación de forma adecuada puede llevar a que el sistema los elimine cuando se estaba realizando una tarea importante.
  - Android construye una jerarquía de procesos según un orden de importancia:
    - **Procesos en primer plano**: pertenece a esta categoría si se cumple alguna de la siguiente condiciones:
      - Tiene un componente *Activity* en ejecución interactuando con el usuario.
      - Tiene un componente *Broadcast Receiver* ejecutándose.
      - Ha lanzado algún otro proceso que tiene un componente *Service* ejecutándose.
    - **Procesos visibles**: tiene un componente *Activity* ejecutándose aunque no con el foco de actividad.
    - **Procesos de servicio**: tiene un componente *Service* ejecutándose en segundo plano.
    - **Procesos en segundo plano**: tiene un componente *Activity* que no es visible para el usuario.
    - **Procesos vacíos**: no ejecutan ninguna actividad pero se mantiene en memoria por si el usuario quiere volver a interaccionar con ellos recuperar el estado más rápidamente.

Desarrollo de aplicaciones Android

- Antes de desarrollar aplicaciones Android es importante tener claro con qué versión de la plataforma vamos a trabajar para saber a qué nivel de API se corresponde.
  - El nivel de API (*API level*) es un valor entero que identifica unívocamente la revisión del *framework API* soportada por una versión determinada de la plataforma Android.
  - Un *framework API* consiste en:
    - Conjunto de clases y paquetes.
    - Conjunto de elementos y atributos XML que pueden declararse en el fichero manifiesto.
    - Conjunto de elementos y atributos XML que pueden declararse para acceder a recursos.
    - Conjunto de *Intents*.
    - Conjunto de permisos que puede requerir una aplicación, así como los permisos del sistema.

Desarrollo de aplicaciones Android

- En la actualidad existen las siguiente correspondencias entre versiones de la plataforma Android y niveles del API.

Platform Version	API Level
Android 3.0	11
Android 2.3.3	10
Android 2.3	9
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

- Para facilitar la compatibilidad es mejor desarrollar la aplicación para la versión más baja posible:
  - En general, existe compatibilidad hacia delante pero no está garantiza la compatibilidad hacia atrás.
- Lo mejor es trabajar filtrando el API por nivel.
  - http://developer.android.com/reference/classes.html

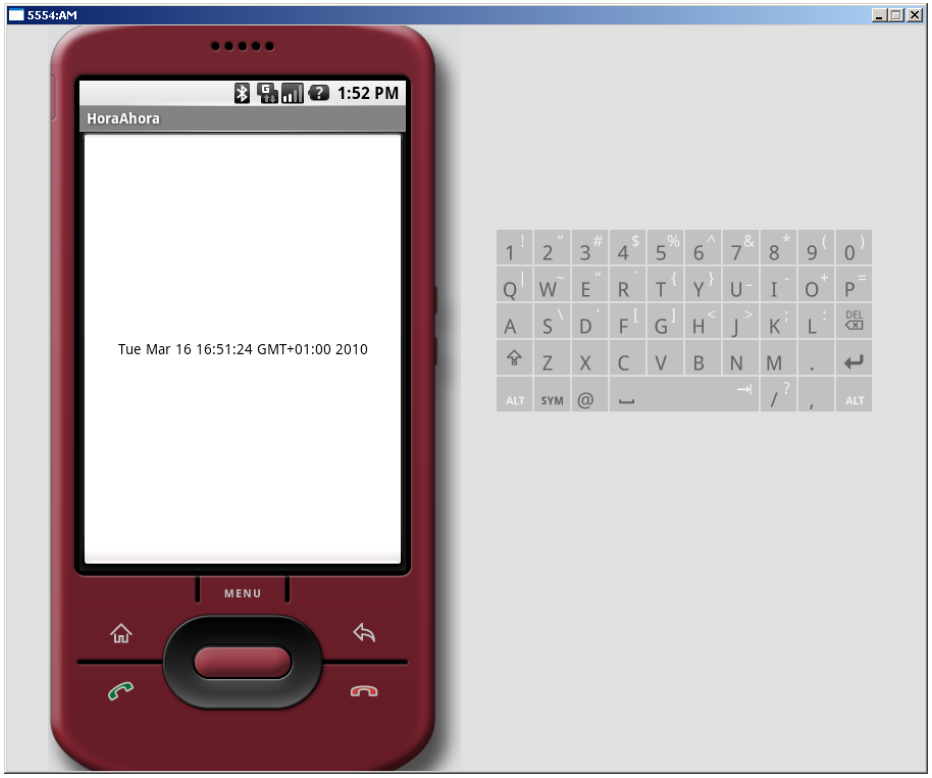
Desarrollo de aplicaciones Android

- Cada versión de la plataforma almacena su Nivel de API internamente:
  - Este valor se tiene en cuenta a la hora de instalar una aplicación.
- Elementos del manifiesto para indicar nivel del API:
  - Elemento `uses-sdk`, con atributos:
    - `android:minSdkVersion`
      - Indica el mínimo nivel del API en el que se puede ejecutar la aplicación. Valor por defecto: 1.
    - `android:targetSdkVersion`
      - Indica el nivel de API en el que se ha diseñado la aplicación.
    - `android:maxSdkVersion`
      - Indica el mayor nivel del API en el que se puede ejecutar la aplicación.

```
<manifest>
  <uses-sdk android:minSdkVersion="5" />
  ...
</manifest>
```

Desarrollo: ejemplo *Activity*

- Vamos a explicar el desarrollo de una *Activity* sencilla:
  - `HoraAhora`, cada vez que se pulsa la pantalla se refresca la hora y fecha mostrada.



Desarrollo: ejemplo *Activity*

- Declaración de paquetes necesarios:

```
package am.example;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
```

  - Las clases específicas de Android están en los paquetes `android`.
  - No todos los paquetes de Java SE están disponibles en Android.
    - Es necesario chequearlo en el API.

Desarrollo: ejemplo *Activity*

```
public class HoraAhora extends Activity implements View.OnClickListener {

  Button btn;

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

      btn = new Button(this);
      btn.setOnClickListener(this);
      updateTime();
      setContentView(btn);
    }

    public void onClick(View view) {
      updateTime();
    }

    private void updateTime() {
      btn.setText(new Date().toString());
    }
}
```

HoraAhora.java

Desarrollo: ejemplo *Activity*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="am.example"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HoraAhora"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="2" />

</manifest>
```

AndroidManifest.xml

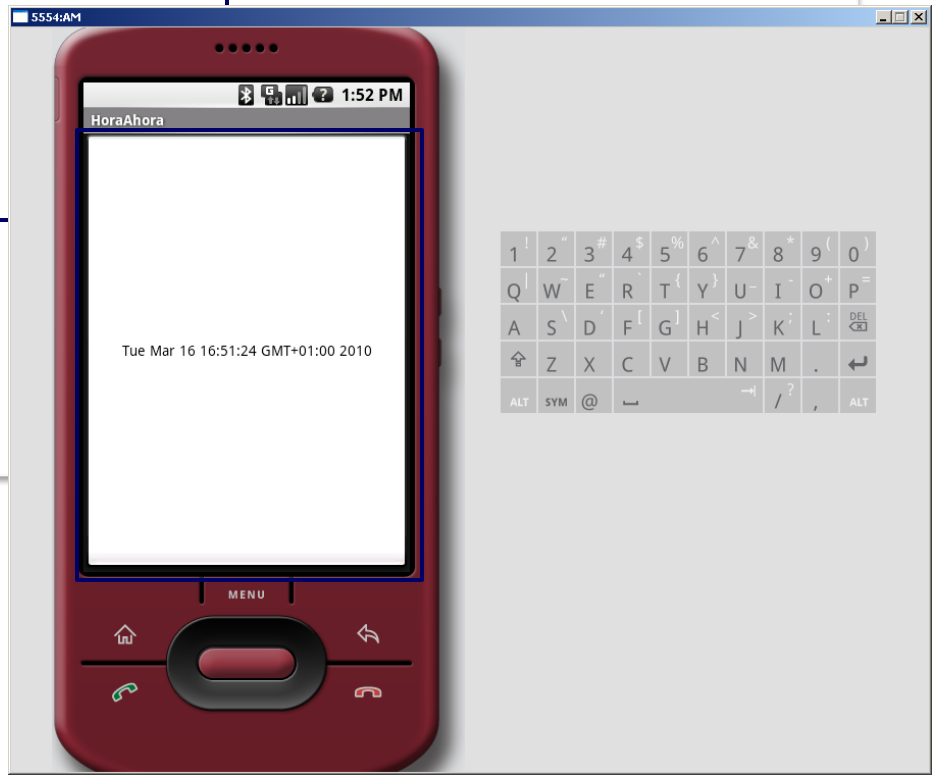
Desarrollo: ejemplo *Activity (con XML)*

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

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

</LinearLayout>
```

layout/main.xml



Desarrollo: ejemplo *Activity (con XML)*

```
public class HoraAhora extends Activity implements View.OnClickListener {

  Button btn;

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

      setContentView(R.layout.main);
      btn = (Button)findViewById(R.id.Button01);

      btn.setOnClickListener(this);
      updateTime();
    }

    public void onClick(View view) {
      updateTime();
    }

    private void updateTime() {
      btn.setText(new Date().toString());
    }
}
```

HoraAhora.java

Índice

Intent e *Intent Filters*

- Hasta ahora hemos visto ejemplos de una sola actividad que se lanzaba con la aplicación principal:
  - Pero si la aplicación tiene varios componentes (lo habitual) ¿cómo se pasa de unos a otros?:
    - Con *Intent* e *Intent Filters.*
- ¿Qué son los *Intent*?
  - Son mensajes que activan componentes de aplicaciones.
  - Contiene la siguiente información:
    - Nombre del componente.
    - **Acción**
    - **Datos**
    - Categoría
    - Extras
    - Flags

Intent e *Intent Filters*

- Nombre del componente (opcional):
  - Indica el componente que debe gestionar el *Intent*, dando el paquete y el nombre de clase.
  - Si no se indica, el sistema deberá seleccionar en base a unos criterios qué componente debe gestionar el *Intent*.
- **Acción**:
  - String que indica la acción a realizar.
  - Existen unos valores predefinidos en la clase `Intent` (algunos en la tabla).
  - Un desarrollador puede definir las suyas propias en sus aplicaciones.

CONSTANT	TARGET COMPONENT	ACTION
`ACTION_EDIT`	activity	Visualiza datos para que el usuario los edite.
`ACTION_MAIN`	activity	Lanza la actividad inicial de una tarea, sin datos de entrada ni datos de salida.
`ACTION_SYNC`	activity	Sincroniza datos en un servidor con datos en el dispositivo móvil.
`ACTION_DIAL`	activity	Marca el número indicado y espera que el usuario confirme la acción
`ACTION_CALL`	activity	Inicia una llamada de teléfono.
`ACTION_SEARCH`	activity	Realiza una búsqueda.

Listado completo: http://developer.android.com/reference/android/content/Intent.html

Intent e *Intent Filters*

- Datos:
  - La URI de los datos y su tipo MIME.
  - Ejemplo: en un `ACTION_EDIT`, la URI nos indicará dónde están los datos para editar.
- Categoría:
  - String con información sobre qué tipo de componente debe gestionar el *Intent*.
  - Existen unos valores predefinidos en la clase `Intent` (algunos en la tabla).

CONSTANT	SIGNIFICADO
CATEGORY_BROWSABLE	La actividad objetivo puede ser invocada por el navegador para mostrar los datos referenciados por un enlace (imagen, correo electrónico, etc…)
CATEGORY_GADGET	La actividad puede ser embebida en otra actividad que alberga gadgets.
CATEGORY_HOME	La actividad muestra la pantalla de HOME, la primera pantalla que el usuario ve cuando enciende el dispositivo o cuando pulsa la tecla HOME.
CATEGORY_LAUNCHER	La actividad es la actividad inicial de una tarea y está listada en el nivel superior del lanzador de aplicaciones.
CATEGORY_PREFERENCE	La actividad objetivo es un panel de preferencias.

Listado completo: http://developer.android.com/reference/android/content/Intent.html

Intent e *Intent Filters*

- Extras:
  - Son pares clave-valor que nos proporcionan información adicional que debe ser proporcionada al componente que gestiona el *Intent*.
  - Ejemplos:
    - Valor de "*time-zone*“ para `ACTION_TIMEZONE_CHANGED`
    - Valor de “*state*” para `ACTION_HEADSET_PLUG`
- Flags:
  - Indican cómo debe ser controlado el *Intent*.
  - Existen unos valores predefinidos en la clase `Intent` (`FLAG_ACTIVITY_NEW_TASK`, `FLAG_DEBUG_LOG_RESOLUTION`, `FLAG_FROM_BACKGROUND`).

Intent e *Intent Filters*

- ¿Qué componente gestiona un Intent?
  - Los Intents pueden dividirse en dos grupos:
    - Explícitos:
      - En su definición incluye el nombre del componente que tiene que gestionarlo.
      - Se utilizan para mensajes internos de una aplicación (actividades que son lanzadas por una actividad).
    - Implícitos:
      - En su definición no se incluye el nombre del componente.
      - Se utilizan frecuentemente para activar componentes de otras aplicaciones (normalmente las del sistema).
      - El sistema Android debe elegir cuál es el componente que debe gestionar este *Intent*:
        - Compara los valores del objeto `Intent` con los **Intent Filters** definidos en las diferentes aplicaciones:
          - Utiliza para ellos los campos acción, datos y categoría.
          - Los valores de extras y flags no se tienen en cuenta.

Intent e *Intent Filters*

- *Intent Filters*:
  - Debe conocerlos el sistema Android antes de lanzar un componente:
    - Se definen en el `AndroidManifiest.xml` en elementos `<intent-filter>`.
  - Para entregar un *Intent* a un componente, uno de sus `<intent-filter>` debe pasar tres tests:
    - Test de acción:
      - Subelemento `<action>` del elemento `<intent-filter>`
      - Puede tener varias acciones y al menos, siempre una.
      - Pasa el test si una de las acciones definidas coincide con la especificada en el objeto Intent.
    - Test de categoría:
      - Subelemento `<category>` del elemento `<intent-filter>`
      - Pasa el test si cada categoría definida en el objeto `Intent` coincide con una categoría indicada en el `<intent-filter>`
      - *Intent* implícitos se crean siempre con la categoría `android.intent.category.DEFAULT`
    - Test de datos:
      - Subelemento `<data>` del elemento `<intent-filter>`:
        - Puede definir una URI y un tipo de datos (MIME).
      - En un *Intent Filter* se pueden definir los elementos de una URI de forma independiente (`scheme`, `host`, `port` y `path`):
        - Se comparan las partes de la URI del *Intent* con las partes definidas en el filtro.
      - Pasa el test:
        - Si el objeto `Intent` no especifica ni URI ni tipo, el filtro no debe incluir ni URI, ni tipo.
        - Si el objeto `Intent` especifica una URI pero no un tipo, el filtro debe tener una URI que coincida, pero no tener definido un tipo.
        - Si el objeto `Intent` especifica un tipo pero no una URI, el filtro debe tener un tipo que coincida pero no tener definida una URI.
        - Si el objeto `Intent` especifica un tipo y una URI, el filtro debe tener un tipo que coincida.

Intent e *Intent Filters*

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```

Ejemplos obtenidos de Developer Android (http://developer.android.com/)

Intent e *Intent Filters*

```
<activity android:name="NoteEditor"
          android:theme="@android:style/Theme.Light"
          android:label="@string/title_note" >
   <intent-filter android:label="@string/resolve_edit">
     <action android:name="android.intent.action.VIEW" />
     <action android:name="android.intent.action.EDIT" />
     <action android:name="com.android.notepad.action.EDIT_NOTE" />
     <category android:name="android.intent.category.DEFAULT" />
     <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
   </intent-filter>
   <intent-filter>
     <action android:name="android.intent.action.INSERT" />
     <category android:name="android.intent.category.DEFAULT" />
     <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
   </intent-filter>
</activity>
```

Ejemplos obtenidos de Developer Android (http://developer.android.com/)

Intent e *Intent Filters*

- *Intent* para lanzar aplicaciones preinstaladas en dispositivos Android.

APP	INTENT URI	INTENT ACTION
Browser	*http://web_address* *https://web_address* *"" (empty string)* *http://web_address* *https://web_address*	VIEW WEB_SEARCH
Dialer	*tel:phone_number* *tel:phone_number* *voicemail:*	CALL DIAL
Google Maps	*geo:latitude,longitude* *geo:latitude,longitude?z=zoom* *geo:0,0?q=my+street+address* *geo:0,0?q=business+near+city*	VIEW
Google StreetView	*google.streetview:cbll=lat,lng&cbp=1,yaw,,pitch,zoom&mz=mapZoom*	VIEW

Lanzar varias actividades

- ¿Cómo se utilizan los *Intent* para lanzar varias actividades?
  - Sabemos que actividad queremos que se lance:
    - Es otra actividad de nuestra propia aplicación.
  - No sabemos que actividad queremos que se lance:
    - Tenemos unos datos y queremos hacer algo con ellos.
- Siempre se realiza en dos pasos:
  - Construir el objeto `Intent`.

```
Intent i = new Intent(this, HelpActivity.class)
```

```
Uri uri = Uri.parse("geo:" + lat.toString() + "," + lon.toString());
Intent i = new Intent (Intent.ACTION_VIEW, uri)
```

  - Lanzar la actividad.

```
startActivity(…)

startActivityForResult(…);
```


Lanzar varias actividades

- ¿Por qué existen dos métodos para lanzar una actividad?
  - `startActivity` no permite saber cuándo finaliza la actividad que lanzas.

```
startActivity(i)
```

  - `startActivityForResult` permite saber cuándo finaliza la actividad (método de callback `onActivityResult`)

```
startActivityForResult(i, ID1)

…

protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
  if (requestCode == ID1)
    if (resultCode == RESULT_OK)

}
```


Lanzar varias actividades

- Ejemplo `ExampleActivities`.

Lanzar varias actividades

- Ejemplo `ExampleActivities`

```
public class FirstActivity extends Activity implements View.OnClickListener {
  static final int ActivityID = 1111;
  Button btn;
  @Override
  public void onCreate(Bundle savedInstanceState) {
  …
  }

  public void onClick(View view) {
    startActivityForResult(new Intent(this, SecondActivity.class), ActivityID);
  }

  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
     if (requestCode == ActivityID) {
       if (resultCode == RESULT_OK){
         startActivity(data);
       }
      }
  }
```

FirstActivity.java

Lanzar varias actividades

- Ejemplo `ExampleActivities`

```
public class SecondActivity extends Activity implements View.OnClickListener {

  Button btn;
  EditText et;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    …
  }

  public void onClick(View view) {
    setResult(RESULT_OK, new Intent(Intent.ACTION_VIEW,
              Uri.parse("geo:" + et.getText())));
    finish();
  }

}
```

SecondActivity.java

Broadcast Receiver e *Intents*

- Los Broadcast Receiver reaccionan antes Intents enviados por broadcast.
  - Un desarrollador de aplicaciones puede enviar su propios *Intent* por difusión para que lo procese otras de sus aplicaciones o aplicaciones de terceros.
  - Android define sus propios *Intents* que se envían por difusión para que las aplicaciones puedan estar al tanto de eventos que se producen a nivel de sistema: nivel de batería, conexiones de red, y llamadas entrantes.

CONSTANT	TARGET COMPONENT	ACTION
`ACTION_BATTERY_LOW`	broadcast receiver	La batería está baja.
`ACTION_TIME_CHANGED`	broadcast receiver	La fecha del sistema ha sido cambiada.
`ACTION_POWER_CONNECTED`	broadcast receiver	El dispositivo se ha conectado a la corriente eléctrica.
`ACTION_SCREEN_OFF`	broadcast receiver	La pantalla se ha apagado
`ACTION_NEW_OUTGOING_CALL`	broadcast receiver	Se va a realizar una llamada.
`ACTION_SEARCH`	broadcast receiver	Realiza una búsqueda.

Broadcast Receiver e *Intents*

- Crear un *Intent* y enviarlo por difusión:

```
public static final String NEW_LIFEFORM_DETECTED = "com.paad.action.NEW_LIFEFORM"

Intent intent = new Intent(NEW_LIFEFORM_DETECTED);
intent.putExtra("lifeformName", lifeformType);
intent.putExtra("longitud", currentLongitud);
Intent.putExtra("latitude", currentLatitude);
sendBroadcast(intent);
```

- Implementar un Broadcast Receiver:
  - Extender la clase BroadcastReceiver:

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class MyBroadcastReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
   // Reaccionar al Intent recibido
  }
}
```


Código obtenido de Professional Android 2 Application Development. Reto Meier. Wiley Publishing, Inc. 2010.

Broadcast Receiver e *Intents*

- Registrar el Broadcast Receiver en la aplicación a través del manifiesto:

```
<receiver android:name="nombre_clase"
  <intent-filter>
    <action android:name="com.paad.action.NEW_LIFEFORM"/>
  </intent-filter>
</receiver>
```

Broadcast Receiver e Intents

```
public class LifeformDetectedBroadcastReceiver extends BroadcastReceiver {

public static final String BURN = "com.paad.alien.action.BURN_IT_WITH_FIRE";

@Override
public void onReceive(Context context, Intent intent) {
  // Obtener la forma de vida.
  Uri data = intent.getData();
  String type = intent.getStringExtra("type");
  double lat = intent.getDoubleExtra("latitude", 0);
  double lng = intent.getDoubleExtra("longitude", 0);
  Location loc = new Location("gps");
  loc.setLatitude(lat);
  loc.setLongitude(lng);
  if (type.equals("alien")) {
    Intent startIntent = new Intent(BURN, data);
    startIntent.putExtra("latitude", lat);
    startIntent.putExtra("longitude", lng);
    context.startActivity(startIntent);
  }
 }
}
```


Actividades en *background*

- En Android cuando se tienen que hacer actividades que consumen un tiempo importante:
  - Utilizar un hilo en segundo plano.
  - Utilizar un servicio en segundo plano.
- La manera más sencilla de utilizar hilos en Android es creando una instancia de una subclase de `Handler`:
  - Sólo se necesita un objeto `Handler` por actividad.
  - La comunicación con el `Handler` se puede hacer:
    - Utilizando objetos `Runnable`.
    - Utilizando mensajes.

Utilizando hilos

- Vamos a ver un ejemplo utilizando mensajes:
  - Para enviar un mensaje a un objeto Handler hay que realizar dos pasos:
    - Invocar `obtainMessage()` para obtener un objeto de tipo `Message`.
    - Enviar el objeto `Message` al objeto `Handler`.
      - `sendMessage(Message msg)`:
        - Pone el mensaje inmediatamente en la cola.
      - `sendMessageAtFrontOfQueue(Message msg)`:
        - Pone el mensaje inmediatamente en la cola, ocupando la primera posición en la cola, por lo tanto con mayor prioridad.
      - `sendMessageAtTime(Message msg, long uptimeMillis)`:
        - Pone el mensaje en la cola en el tiempo indicado (valor absoluto).
      - `sendMessageDelayed(Message msg, long delayMillis)`:
        - Pone el mensaje en la cola en el tiempo indicado (valor relativo al instante actual).

Utilizando hilos

- Ejemplo `HandlerDemo`:

```
<?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"
    />

<ProgressBar
    android:id="@+id/progress"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

</LinearLayout>
```

main.xml

Código obtenido de http://www.java2s.com/Code/Android/Development/ThreadsHandlerDemo.htm

Utilizando hilos

- Ejemplo `HandlerDemo`:

```
public class HandlerDemo extends Activity {
  ProgressBar bar;

  Handler handler=new Handler() {
    @Override
    public void handleMessage(Message msg) {
       bar.incrementProgressBy(5);
    }
  };

  AtomicBoolean isRunning=new AtomicBoolean(false);

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    bar=(ProgressBar)findViewById(R.id.progress);
  } // end onCreate
```

HandlerDemo.java

Código obtenido de http://www.java2s.com/Code/Android/Development/ThreadsHandlerDemo.htm

Utilizando hilos

- Ejemplo `HandlerDemo`:

```
public void onStart() {
  super.onStart();
  bar.setProgress(0);
  Thread background=new Thread(new Runnable() {
    public void run() {
      try {
        for (int i=0;i<20 && isRunning.get();i++) {
          Thread.sleep(1000);
          handler.sendMessage(handler.obtainMessage());
        }
      } catch (Throwable t) {
        // just end the background thread
      }
    }
  });
  isRunning.set(true);
  background.start();
} // end onStart
```

HandlerDemo.java

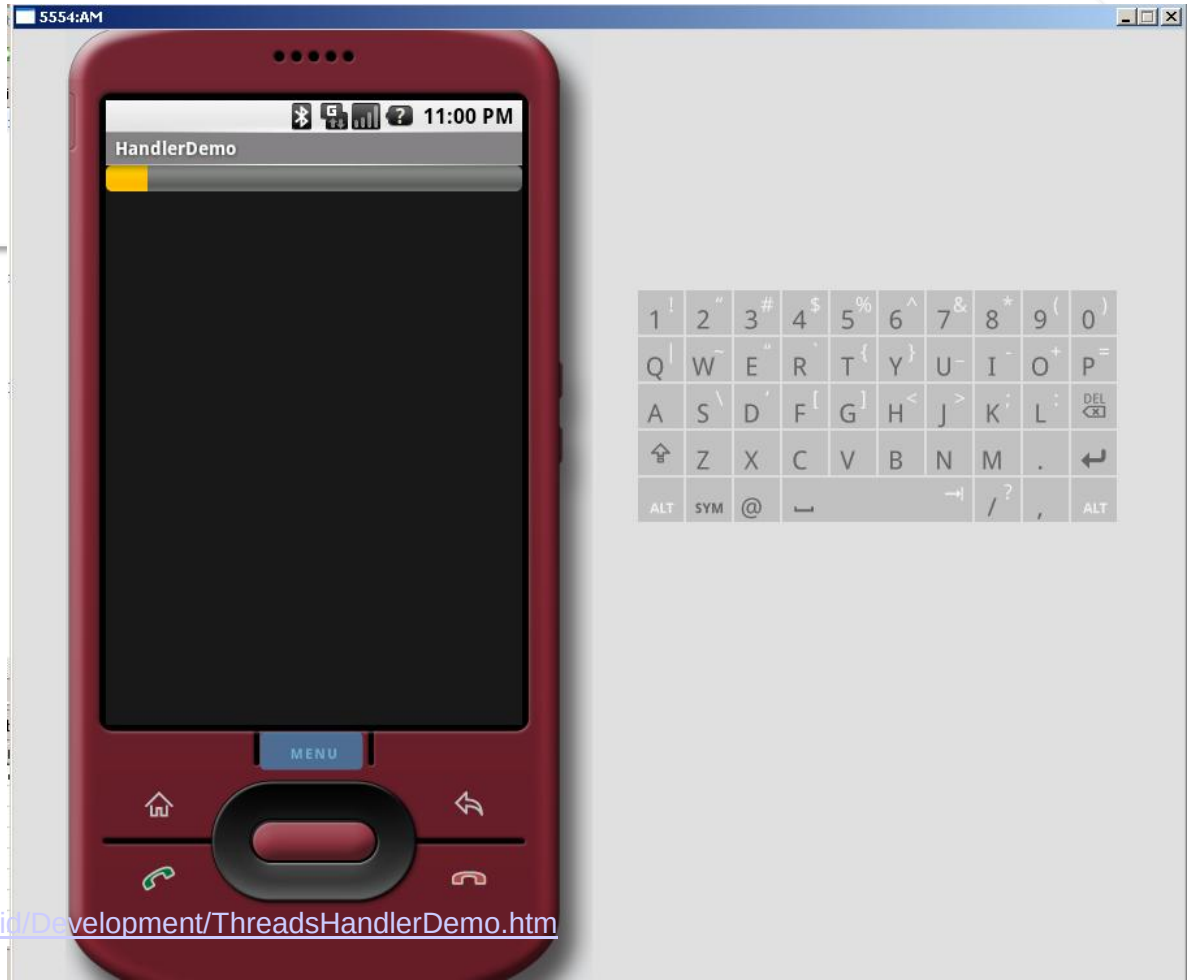
Código obtenido de http://www.java2s.com/Code/Android/Development/ThreadsHandlerDemo.htm

Utilizando hilos

- Ejemplo `HandlerDemo`:

```
  public void onStop() {
    super.onStop();
    isRunning.set(false);
  } // end onStop

} // end class HandlerDemo
```



Código obtenido de http://www.java2s.com/Code/Android/Development/ThreadsHandlerDemo.htm

Creando un servicio

- Los pasos para crear un servicio son:
  - Extender la clase `Service`.
  - Se pueden sobreescribir los métodos del ciclo de vida:
    - `onCreate()`:
      - El método que se invoca cuando se crea el servicio.
    - `onStart()`:
      - El método que se invoca cuando algún componente arranca un servicio de forma “manual”.
      - Normalmente no se sobreescribe, sólo si es necesario pasar datos a través de un objeto `Bundle`.
      - A partir del API de nivel 5 este método se ha sustituido por:
        - `onStartCommand()`
    - `onDestroy()`:
      - El método que se invoca cuando se para el servicio.
    - `onBind()`:
      - El método que devuelve un canal de comunicación con el servicio.
      - Si no se permite que un componente se enlace al servicio, se devuelve `null`.

Creando un servicio

- Ejemplo `ExampleService`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="am.example"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ActivityService"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:name="MyService"></service>

    </application>
    <uses-sdk android:minSdkVersion="2" />
</manifest>
```

AndroidManifest.xml

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:

```
public class MyService extends Service {

  private Timer timer = new Timer();
  private static final long UPDATE_INTERVAL = 1000;
  public static ServiceUpdateUIListener UI_UPDATE_LISTENER;
  private int count = 0;

  public static void setUpdateListener(ServiceUpdateUIListener l) {
    UI_UPDATE_LISTENER = l;
  }

  private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
      UI_UPDATE_LISTENER.update(count);
    }
  };
```

MyService.java

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:

```
@Override
public IBinder onBind(Intent arg0) {
  return null;
}

@Override
public void onCreate() {
  super.onCreate();
  _startService();
}

@Override
public void onDestroy() {
  super.onDestroy();
  _shutdownService();
}
```

MyService.java

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:

```
  private void _startService() {
    timer.scheduleAtFixedRate(
    new TimerTask() {
      public void run() {
        count++;
        handler.sendEmptyMessage(0);
      }
    },
    0,
    UPDATE_INTERVAL);
    }

  private void _shutdownService() {
    if (timer != null) timer.cancel();
  }

} // end class MyService
```

MyService.java

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:

```
public class ActivityService extends Activity implements ServiceUpdateUIListener {

  TextView text;

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

    text = (TextView) findViewById(R.id.count)
    Button startButton = (Button) findViewById(R.id.start);

    startButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        startService();
      }
    });
```

ActivityService.java

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:

```
  Button stopButton = (Button) findViewById(R.id.stop);

  stopButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
      stopService();
    }
  });

  MyService.setUpdateListener(this);
} // end onCreate

private void startService() {
  Intent svc = new Intent(this, MyService.class);
  startService(svc);
}

private void stopService() {
  Intent svc = new Intent(this, MyService.class);
  stopService(svc);
}
```

ActivityService.java

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:

```
    @Override
    public void update(int count) {
      text.setText("Contador: " + count);
    }
} // end class ActivityService
```

ActivityService.java

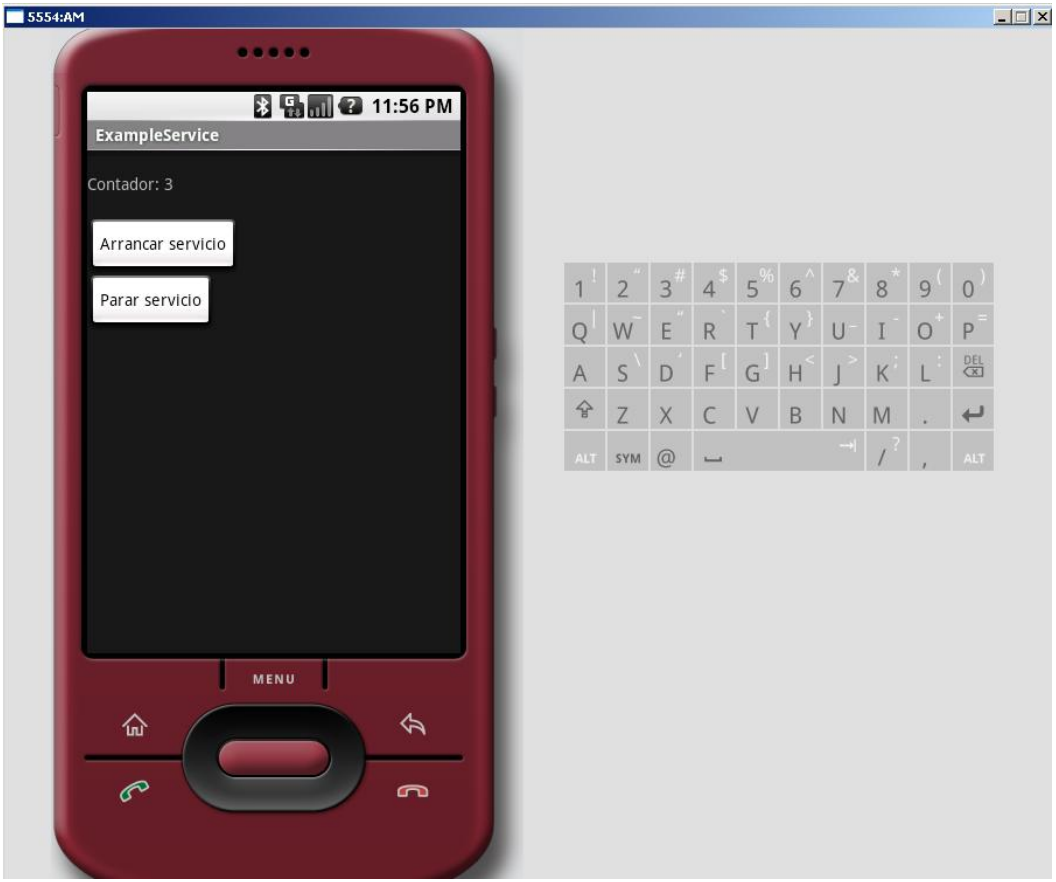
```
public interface ServiceUpdateUIListener {
  public void update(int count);
}
```

ServiceUpdateUIListener.java

Código obtenido de http://android.javielinux.com/pruebaservicio.php

Creando un servicio

- Ejemplo `ExampleService`:



API de Android

- El API de Android consta de múltiples clases en varios paquetes:
  - `android.*`
    - Específicas de Android.
  - `java.*` y `javax.*`
    - Comunes con Java SE (no todas).
  - `org.*`
    - Relacionadas con apache, xml y w3c.
  - `junit.*`
    - Para la realización de pruebas unitarias basadas en `junit`.
  - `dalvik.*`
    - Relacionadas con la máquina virtual Dalvik.

API de Android – Interfaces gráficas

- Android define los siguientes elementos para interfaces gráficas:
  - Vistas (*Views*): es la clase base de todos los elementos de un interfaz gráfico.
    - Clase `android.view`.
  - Grupos de vistas (*View groups*): son extensiones de las vistas que nos permite trabajar con un conjunto de ellas.
    - Clase `android.view.ViewGroup`.
  - Actividades (*Activities*): representan la pantalla. Para mostrarle a un usuario una interfaz lo que hacemos es asignar una vista a una actividad.
    - Clase `Activity`.

API de Android – Interfaces gráficas

- Las vistas más habituales son:
  - `android.widget.TextView`:
    - Etiqueta de texto de sólo lectura.
  - `android.widget.EditText`:
    - Un campo de texto editable.
  - `android.widget.ListView`:
    - Un `ViewGroup` que nos permite crear y gestionar una lista vertical de vistas.
  - `android.widget.Spinner`:
    - Nos permite seleccionar un elemento de una lista.
  - `android.widget.Button`:
    - Un botón.
  - `android.widget.RadioButton`:
    - Un botón con dos estados, activado o desactivado.
  - `android.widget.ViewFlipper`:
    - Un `ViewGroup` que nos permite realizar una animación entre diferentes vistas, sólo una de ellas será visible al mismo tiempo.
  - `android.widget.QuickContactBadge`:
    - Nos permite visualizar información de un contacto visualizando su foto e información de contacto.
  - `android.view.Menu`:
    - Un menú de opciones.

API de Android – Interfaces gráficas

- Los elementos del interfaz gráfico se introducen en “layouts”:
  - Los gestores de “layouts” son extensiones de `ViewGroup`.
- Las clases de Layout más importantes de Android son:
  - `android.widget.FrameLayout`:
    - Un marco que ocupa toda la pantalla, todos los elementos se colocan a partir de la esquina superior izquierda por lo que unos pueden quedar ocultos detrá de otros.
  - `android.widget.LinearLayout`:
    - Los elementos se van colocando uno detrás del otro.
    - Podemos indicar si es horizontal o vertical la ordenación.
  - `android.widget.RelativeLayout`:
    - Los elementos se van colocando de forma relativa a otro elemento.
  - `android.widget.TableLayout`:
    - Los elementos se van colocando como una tabla, en filas, y cada fila puede tener varios elementos (columnas).
  - `android.widget.Gallery`:
    - Los elementos se colocan en un sola fila que se pueden ir visualizando de forma horizontal.

API Android – Internet

- Todas las aplicaciones que tienen que acceder a recursos de Internet deben especificar el siguiente permiso en su manifiesto:

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

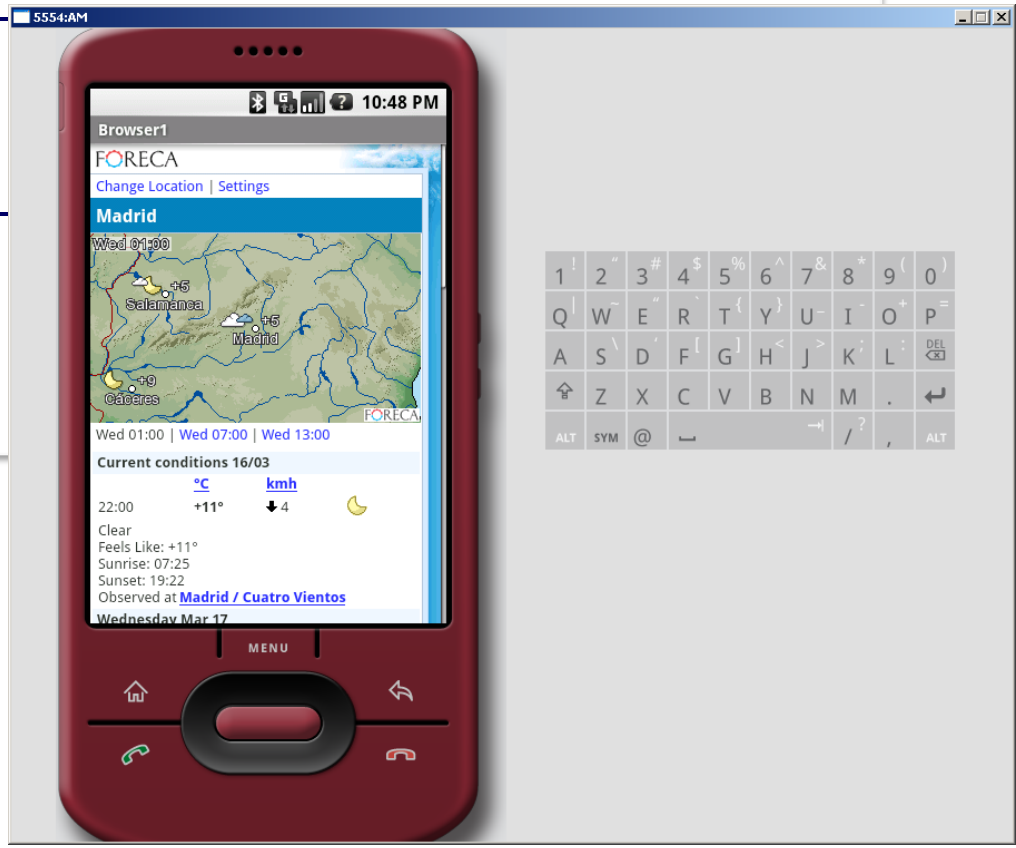
- Las clases habituales para conexiones están:
  - `android.net.*`
  - `java.net.*`
  - `org.apache.http.*`
- Android también nos proporciona el paquete android.webkit para tener utilidades de navegador dentro de nuestra aplicación.
  - La clase `WebView` nos permite visualizar de forma sencillas páginas web.

Internet: ejemplo WebKit

```
<?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"
    >
<WebView
    android:id="@+id/webkit"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
 />

</LinearLayout>
```

layout/main.xml



Internet: ejemplo WebKit

```
package am.example;

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

public class Browser1 extends Activity {

  WebView browser;

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

      setContentView(R.layout.main);
      browser = (WebView)findViewById(R.id.webkit);

      browser.loarUrl("http://www.foreca.mobi");

    }

}
```

Browser1.java

Internet: ejemplo WebKit

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

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

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Browser1"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="2" />

</manifest>
```

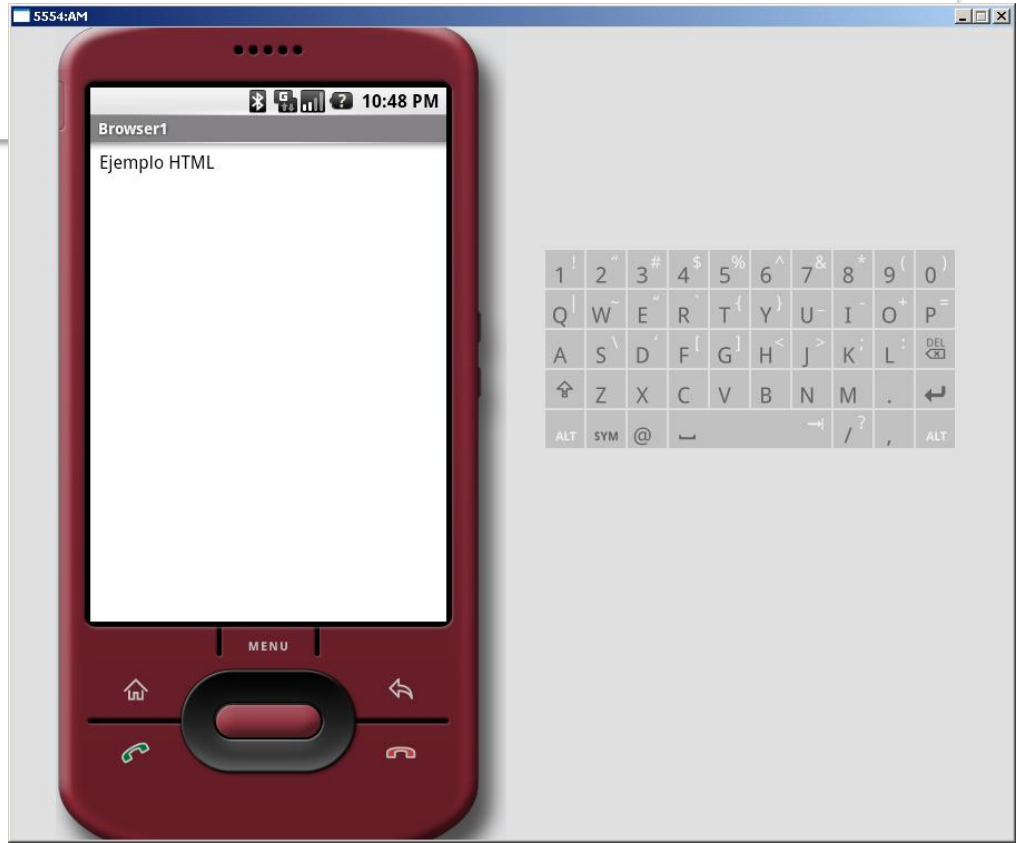
AndroidManifest.xml

Desarrollo: ejemplo WebKit

```
…
setContentView(R.layout.main);
browser = (WebView)findViewById(R.id.webkit);

browser.loadData("<html><body>Ejemplo HTML</body></html>", "text/html", "UTF-8"));
…
```

Browser1.java



Referencias

- "*The Developer's Guide*". Android.
  - http://developer.android.com/guide/index.html
- "*Professional Android 2 Application Development*". Reto Meier. 2010.
  - Biblioteca UC3M: L/D 004.451.9 ANDROID MEI.
- "*The Busy Coder's Guide to Android Development*". Mark L. Murphy. July 2008.
  - Biblioteca UC3M: L/D 004.42 MUR
- *Andbook!. Android Programming*.
  - http://andbook.anddev.org/