UNIDAD 6

SERVICIOS, SENSORES, PERMISOS Y RECEPTORES

Diseño de Interfaces Web

UD6. Sensores, Servicios, Permisos y Receptores

1. Servicios
2. Sensores
3. Permisos
4. Receptores

UD6. Sensores, Servicios, Permisos y Receptores

1. **Servicios**
2. Sensores
3. Permisos
4. Receptores

1. Servicios

- **Un servicio es una sub-aplicación que se ejecuta en segundo plano**, normalmente sin interacción con el usuario, y que realiza una serie de tareas dependiendo de la activación de determinados eventos externos, como la recepción de un email, la activación de un temporizador, etc.
- Crearemos un servicio cuando sea necesario añadir un nuevo componente a tu aplicación para ejecutar algún tipo de acción que se ejecute en segundo plano, es decir, que no requiera una interacción directa con el usuario; pero que queramos que permanezca activo aunque el usuario cambie de actividad.

1. Servicios

Un servicio puede funcionar de dos modos:

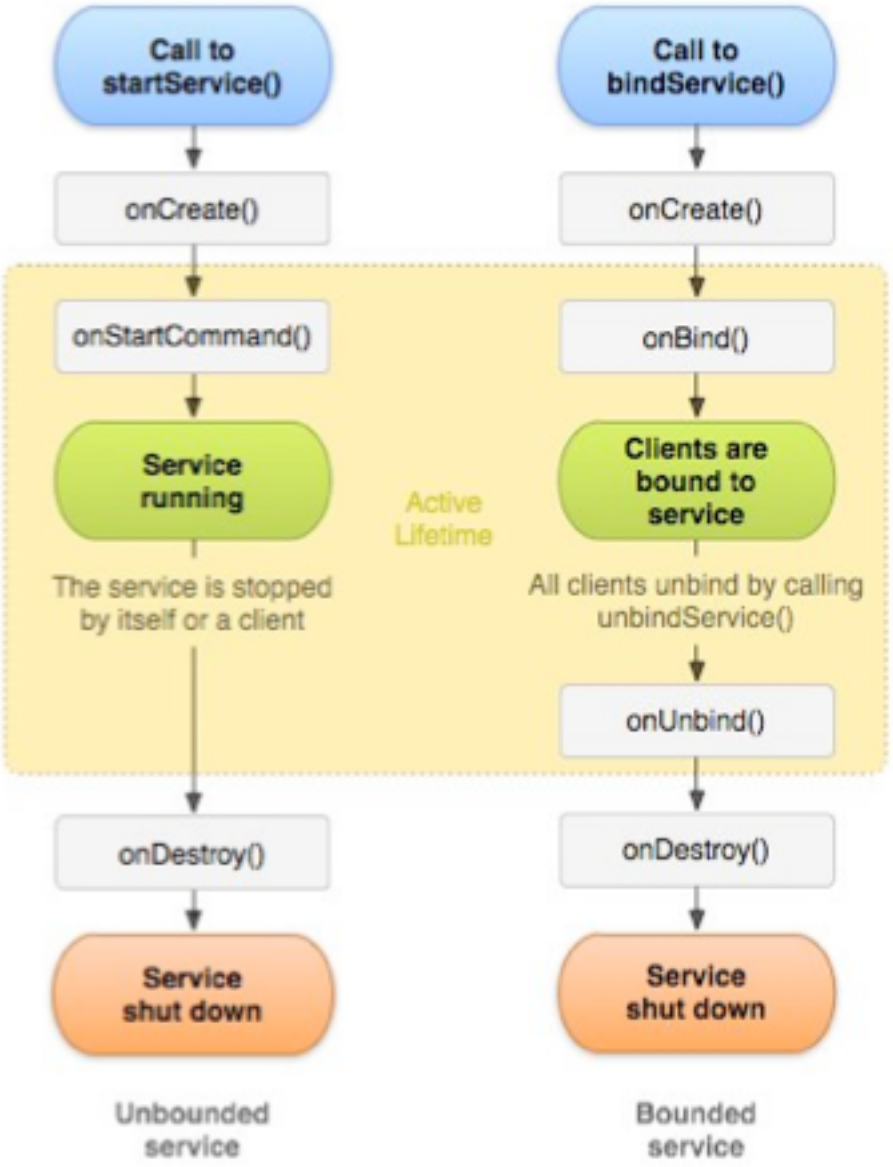
- **Autónomo**: cuando un componente de la aplicación, por ejemplo, una actividad, inicia el servicio mediante el método ***StartService ()***. Una vez arrancado, el servicio puede ejecutarse en segundo plano de forma indefinida, incluso si el componente que lo inició se destruye. Normalmente, un servicio iniciado de esta forma realiza una única operación y no devuelve el resultado al componente que lo inicia. Por ejemplo, puede descargar de Internet un archivo o cargarlo. Cuando la operación finaliza, el servicio debe detenerse.

1. Servicios

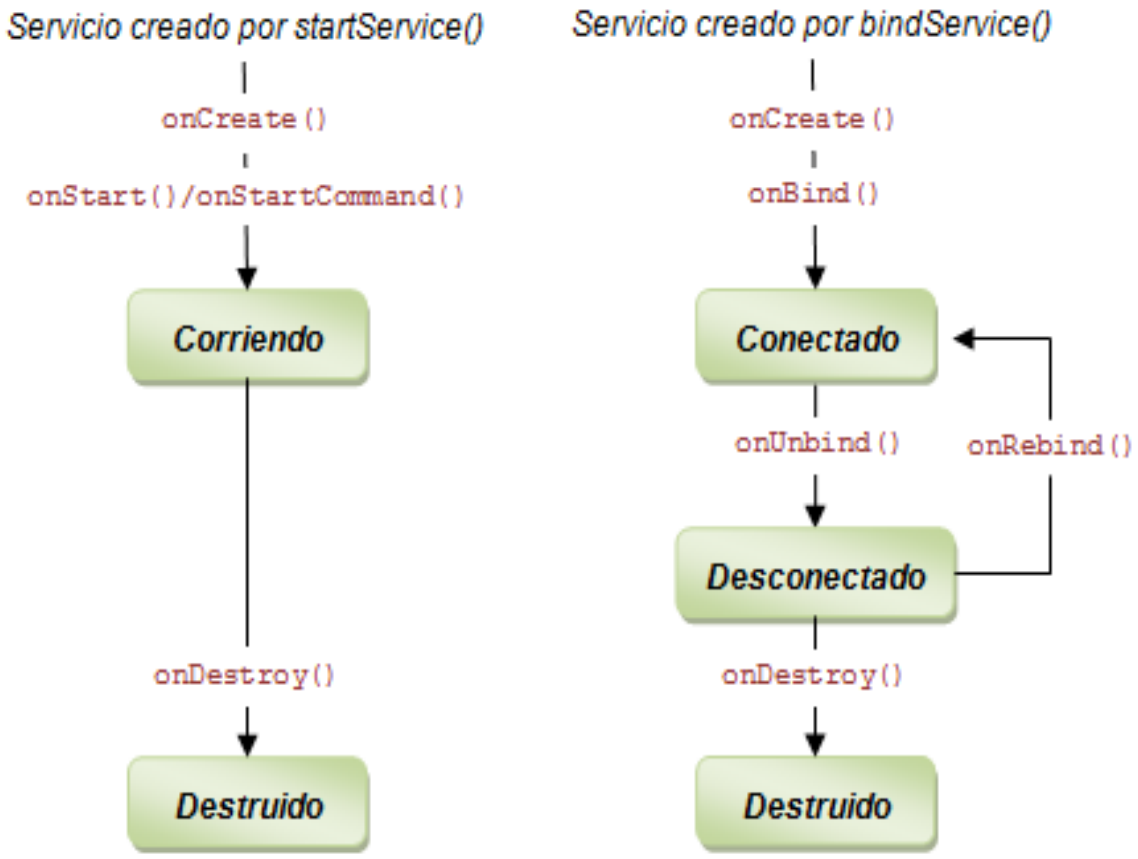
Un servicio puede funcionar de dos modos:

- **Dependiente o Ligado** (en inglés a este modo se le denomina "bind"): cuando un componente de la aplicación se une al servicio mediante el método **bindService()**. Un servicio ligado ofrece una interfaz de tipo cliente-servidor que permite a los componentes de una aplicación interactuar con él enviando peticiones y recibiendo su resultado. Un servicio ligado sólo se ejecuta mientras otro componente de la aplicación está unido a él. Es posible unir un mismo servicio a varios componentes de una o de varias aplicaciones al mismo tiempo; sin embargo, cuando todos ellos se “desligan”, el servicio se destruye.

1. Servicios


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service
Active Lifetime

1. Servicios


Servicio creado por startService()
onCreate()
onStart()/onStartCommand()
Corriendo
onDestroy()
Destruido
Servicio creado por bindService()
onCreate()
onBind()
Conectado
onUnbind()
onRebind()
Desconectado
onDestroy()
Destruido

1. Servicios

- Una Actividad puede iniciar un servicio en modo Autónomo a través del método **StartService()** y detenerlo mediante el método **StopService()**.
- Cuando lo hacemos, Android invoca su método **onCreate()**; después, se invoca el método **onStartCommand()** con los datos proporcionados por la Intención de la actividad.
- En el método **startService()** también podemos indicar como parámetro el comportamiento del ciclo de vida de los servicios:
  - **START_STICKY**: se utiliza para indicar que el servicio debe ser explícitamente iniciado o parado.
  - **START_NOT_STICKY**: el servicio termina automáticamente cuando el método **onStartCommand()** finaliza su ejecución.

1. Servicios

- Cuando se inicia un servicio para realizar alguna tarea en segundo plano, el proceso donde se ejecuta podría ser eliminado ante una situación de baja memoria.
- Podemos configurar la forma en que el sistema reaccionará ante esta circunstancia según el valor que devolvamos en *onStartCommand*().
  - Devolveremos START_STICKY si queremos que el sistema trate de crear de nuevo el servicio cuando disponga de memoria suficiente.
  - Devolveremos START_NOT_STICKY si queremos que el servicio sea creado de nuevo solo cuando llegue una nueva solicitud de creación.

1. Servicios

Creación de un servicio

- Al igual que ocurre con las actividades, los servicios deben definirse en el *manifiesto* de nuestra aplicación.
  - <service android:name=".ServiceClassName" />

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
   <application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name=".MainActivity"
           android:label="@string/title_activity_main" >
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER"/>
           </intent-filter>
       </activity>
       <service android:name=".MyService" />
   </application>
</manifest>
```

1. Servicios

Creación de un servicio

- Hay que terner en cuenta que la clase que implemente el servicio deberá heredar de la superclase *Service*.

```
public class MyService extends Service {

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    //TODO do something useful
    return Service.START_NOT_STICKY;
  }

  @Override
  public IBinder onBind(Intent intent) {
  //TODO for communication return IBinder implementation
    return null;
  }
}
```

1. Servicios

Instanciación de un servicio

- Al igua que ocurre con las activiades, no pueden co**hexistir en el sistema dos servicios de la misma clase, po**r lo que antes de instanciar un servicio debemos comprobar si éste ya se está ejecutando.

  startService(new Intent((context, service_class));
  bindService(new Intent((context, service_class))

```
// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);
// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
context.startService(i);
```

1. Servicios

Instanciación de un servicio

- Al igua que ocurre con las activiades, no pueden co**hexistir en el sistema dos servicios de la misma clase, po**r lo que antes de instanciar un servicio debemos comprobar si éste ya se está ejecutando.

  startService(new Intent((context, service_class));

```
// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);
// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
context.startService(i);
```

1. Servicios

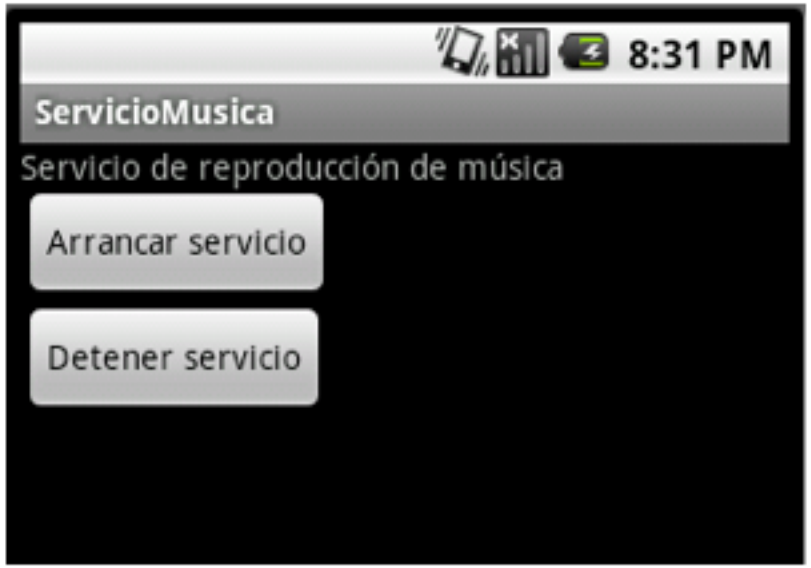
Destrucción de un servicio

- La activación de un evento provoca que una porción de código asociada a un servicio se ejecute, pero éste permanece residente en memoria, a la espera de que le mismo u otro evento se active, hasta que el servicio que es explícitamente destruido, lo cual puede hacerse mediante la llamada *stopService()* (a la que se le pasa como parámetro el tipo de servicio a detener) o mediante la llamada al procedimiento *stopSelf* asociada al servicio.
  - stopService(new Intent(this, MyService.class));

1. Servicios

Práctica:

Crea un servicio que reproduzca una canción/sonido en segundo plano



UD6. Sensores, Servicios, Permisos y Receptores

1. Servicios
2. Sensores
3. **Permisos**
4. Receptores

3. Permisos

- Android es un sistema operativo que separa privilegios entre aplicaciones que se ejecutan simultáneamente usando identidades diferentes del sistema, es decir, con un ID de usuario de grupo de Linux diferente (cada aplicación se ejecuta con un usuario distinto).
- Determinadas partes del sistema operativo también se separan usando diferentes identidades.
- Linux aísla así las aplicaciones entre sí y del sistema operativo.
- Además, se proporcionan características adicionales de seguridad más específicas a través del mecanismo de "permiso" que impone restricciones a las operaciones concretas que un proceso en particular puede realizar.

3. Permisos

- La filosofía del diseño de la arquitectura de seguridad de Android consiste en que, por defecto, una aplicación no tiene permisos para realizar cualquier operación que pueda afectar negativamente a otras aplicaciones, al sistema operativo o al usuario.
- Esto incluye la lectura o escritura de datos privados del usuario (como los contactos o correos electrónicos), leer o escribir archivos de otra aplicación, acceder a Internet, etcétera.

3. Permisos

- Todas las aplicaciones Android (archivos .apk) deben estar firmadas con un certificado que identifica al autor de la aplicación.
- El propósito de los certificados de Android es distinguir a los autores de las aplicaciones.
- Esto permite al sistema conceder o denegar solicitudes de acceso para compartir la identidad de Linux que tenga otra aplicación.
- Cualquier dato almacenado por la aplicación será asignado a su usuario Linux, por lo que normalmente no tendrán acceso otras aplicaciones.

3. El usuario Linux y acceso a los ficheros

- Cuando crees un fichero puedes usar los modos MODE_WORLD_READABLE y/o MODE_WORLD_WRITEABLE para permitir que otras aplicaciones puedan leer o escribir en el fichero.
- Aunque otras aplicaciones puedan escribir el fichero, el propietario siempre será el usuario asignado a la aplicación que lo creó.

3. Permisos

- Una aplicación básica de Android no tiene permisos asociados, por lo que no puede hacer nada que afectara al usuario o a cualquier aplicación instalada en el dispositivo.
- Para hacer uso de las características protegidas del dispositivo, se debe incluir en el fichero AndroidManifest.xml del proyecto una o más etiquetas **<uses-permission>** que declaren los permisos que necesita la aplicación.

3. Permisos

*AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.androidandmobileservices"
    android:versionCode="1"
    android:versionName="1.0" >

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

3. Permisos

- **CALL_PHONE** - *Servicios por los que tienes que pagar* – llamar directamente a números de teléfono (muy alta). Permite realizar llamadas sin la intervención del usuario. Nunca solicites este permiso en tus aplicaciones. Si has de realizar una llamada, es mejor realizarla por medio de una intención. A diferencia de la llamada directa, no necesitas ningún permiso, pero el usuario ha de pulsar el botón de llamada para que comience.

- **SEND_SMS** - *Servicios por los que tienes que pagar* – Enviar mensaje de texto (muy alta). Permite a la aplicación mandar SMS. Por iguales razones, a no ser que tu aplicación tenga que mandar SMS sin la intervención del usuario, resulta más conveniente enviarlos por medio de una intención.

3. Permisos

- **READ_PHONE_STATE** - *Llamadas de teléfono* – leer ID y estado del teléfono (alta). Muchos juegos piden este permiso para ponerse en pausa cuando recibes una llamada. Sin embargo, también permite el acceso al IMEI (identificador de teléfono GSM), IMSI (identificador tarjeta SIM) y al identificador único de 64 bits que Google asigna a cada terminal. En las primeras versiones de SDK este permiso no era necesario

- **WRITE_EXTERNAL_STORAGE** - *Almacenamiento* – modificar/eliminar contenido del almacemaniento USB (alta) Permites el borrado y modificación de archivos en la memoria exterior. Lo ha de solicitar toda aplicación que necesite escribir un fichero en la memoria externa. Ejemplo: exportar datos en XML. Pero al permitirlo también podrán modificar/eliminar ficheros externos creados por otras aplicaciones. (desde versión 1.6)

3. Permisos

- **READ_EXTERNAL_STORAGE** - *Almacenamiento leer contenido del almacenamiento USB* - Permite leer archivos en la memoria exterior. Este permiso se ha introducido para un futuro uso. En la actualidad todas las aplicaciones pueden leer en la memoria externa. Por lo tanto, has de tener cuidado con la información que dejas en ella.

- **ACCESS_FINE_LOCATION** - *Tu ubicación*– Localización GPS detallada (moderada). Localización basada en satélites GPS.

3. Permisos

- **ACCESS_COARSE_LOCATION** - *Tu ubicación*– Localización no detallada (basada en red) (moderada). Localización basada en telefonía móvil (Cel-ID) y Wi-Fi. Aunque en la actualidad esta tecnología nos suele ofrecer menos precisión que el GPS, no siempre es así. Por ejemplo, se está aplicando en el interior de aeropuertos y museos con precisiones similares.

- **READ_OWNER_DATA** - *Tu información personal* – leer datos de contacto (alta). Permiten leer información sobre el propietario del teléfono (nombre, correo electrónico, número de teléfono). Muy peligroso, algunas aplicaciones podrían utilizar esta información de forma no lícita.

3. Permisos

- **READ_CALENDAR** - *Tu información personal* – leer datos de calendario (moderada). Solo da este permiso si consideras que la aplicación realmente lo necesita.

- **WRITE_CALENDAR** - *Tu información personal* – escribir datos de calendario (moderada). Por las mismas razones que el anterior permiso, hay que plantearse si una aplicación nos pide este permiso con sentido o no.

3. Permisos

- **INTERNET** - *Comunicación de red*– Acceso a Internet sin límites (muy alta). Permite establecer conexiones a través de Internet. Este es posiblemente el permiso más importante, en el que más hay que fijarse a quién se le otorga. La mayoría de las aplicaciones lo piden, pero no todas lo necesitan. Cualquier *malware* necesita una conexión para poder enviar datos de nuestro dispositivo.

- **ACCESS_WIFI__STATE** - *Comunicación por red* – ver estado de conexión, ver estado Wi-Fi (baja). Información sobre redes WiFi disponibles.

3. Permisos

- **BLUETOOTH** - *Comunicación de red crear conexión* - Bluetooth (baja). Conexión con otro dispositivo Bluetooth.

- **WAKE_LOCK** - *Herramientas del sistema* – Impedir que el teléfono entre en modo de suspensión (baja). Algunas aplicaciones pueden necesitar de este permiso, y realmente a lo único que puede afectar es a nuestra batería.

3. Permisos

- **CHANGE_CONFIGURATION** - *Herramientas del sistema* – Modificar la configuración global del sistema (moderada). Permite cambiar la configuración. Pese a que es importante en sí, es muy común, así por ejemplo los widgets lo necesitan.

- **READ_SYNC_SETTINGS** - *Herramientas del sistema* – leer ajustes de sincronización (baja). Tan solo permite saber si tienes sincronización en segundo plano con alguna aplicación (como con un cliente de Twitter o Gmail).

3. Permisos

- **WRITE_APN_SETTINGS** - *Herramientas del sistema* – Escribir configuración del Punto de Acceso (moderada). Permite la configuración del punto de acceso a la red. Por ejemplo, encender y a apagar tu conexión de red o Wi-Fi.

- **MANAGE_APP_TOKENS** - *Herramientas del sistema* – recuperar aplicaciones en ejecución (moderada). Permite saber qué aplicaciones están corriendo en segundo plano y cambiar el orden.

3. Permisos

- **SET_PREFERRED_APPLICATIONS** - *Herramientas del sistema* – establecer aplicaciones preferidas (moderada). Permite la asignación a una aplicación para que haga determinada tarea. Por ejemplo, la reproducción de vídeos.

- **VIBRATE** - *Controles de hardware* – control de la vibración (baja). Permite hacer vibrar al teléfono. Los juegos lo suelen utilizar bastante.

3. Permisos

- **CAMERA -** *Controles de hardware* – realizar fotografías (baja). Permite acceso al control de la cámara y a la toma de imágenes.

- **RECORD_AUDIO -** *Controles de hardware* – grabar audio (moderada). Permite grabar sonido desde el micrófono del teléfono.

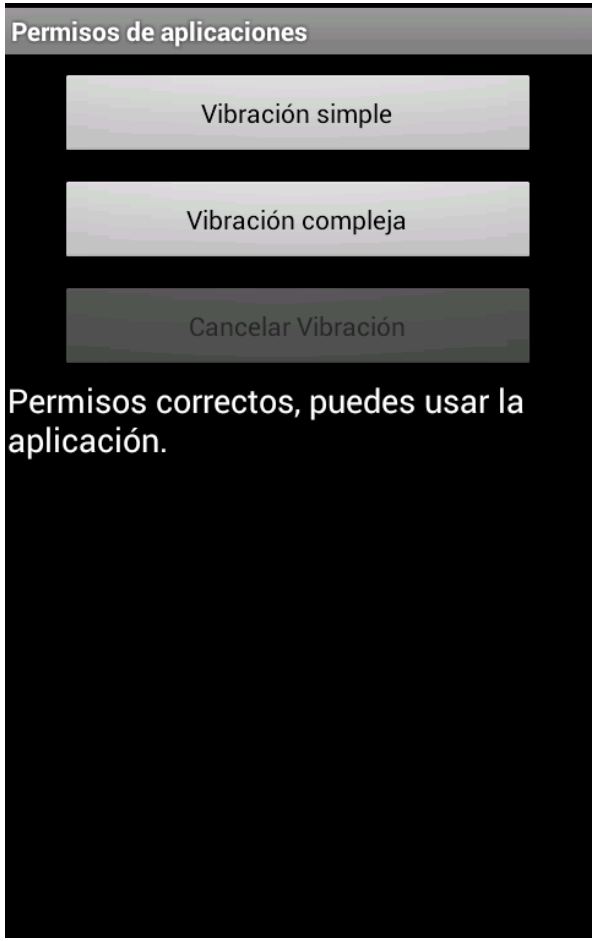
3. Permisos

- Si nos olvidáramos de declarar los permisos necesarios de una aplicación en el fichero AndroidManifest y no controlamos en tiempo de ejecución los permisos que tiene asignados, al ejecutar la , veríamos el siguiente mensaje y la aplicación finalizaría, por ejemplo:

3. Permisos

Práctica:

Crea una aplicación que requiera el alta de permisos



UD6. Sensores, Servicios, Permisos y Receptores

1. Servicios
2. Sensores
3. Permisos
4. **Receptores**

4. Receptores

- Existen **mensajes de difusión** (**Broadcast**) para comunicar eventos entre servicios, como por ejemplo *Batería baja, llamada entrante,… .*
- Estos mensajes son, en realidad, **Intents**.
- Para ello usamos la clase Receptor de mensajes de difusión (**BroadcastReceiver**), que debemos declarar en el archivo **AndroidManifest.xml**.
- Esta clase puede recibir Intenciones (**Intents**), es decir, mensajes enviados por otro componente de Android mediante el método **sendBroadcast()** de la clase **Context** (contexto de la aplicación).
- La clase **BroadCastReceiver** define el único método **onReceive ()** donde se recibe el mensaje de difusión; por lo tanto, fuera de este método no se puede realizar ninguna operación asíncrona porque el mensaje de difusión ya no está activo.

4. Receptores

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


      <receiver android:name="ReceptorLlamadas">

      <intent-filter>

           <action android:name="android.intent.action.PHONE_STATE">

           </action>

      </intent-filter>

      </receiver>

</application>

...


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

</uses-permission>
```

4. Receptores

- La clase **ReceptorLlamadas** es subclase de BroastReceiver e implementa el receptor de mensajes de difusión contiene las siguientes sentencias:

```
@Override
public void onReceive(Context context, Intent intent) {
   Bundle extras = intent.getExtras();
   if (extras != null) {
     String estado = extras.getString(TelephonyManager.EXTRA_STATE);
     Log.w("ESTADO TELEFONO", estado);
     if (estado.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
          String numeroTelefono= extras
                 .getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
          Log.w("NUMERO TELEFONO", numeroTelefono);
     }
   }
 }
}
```

4. Receptores

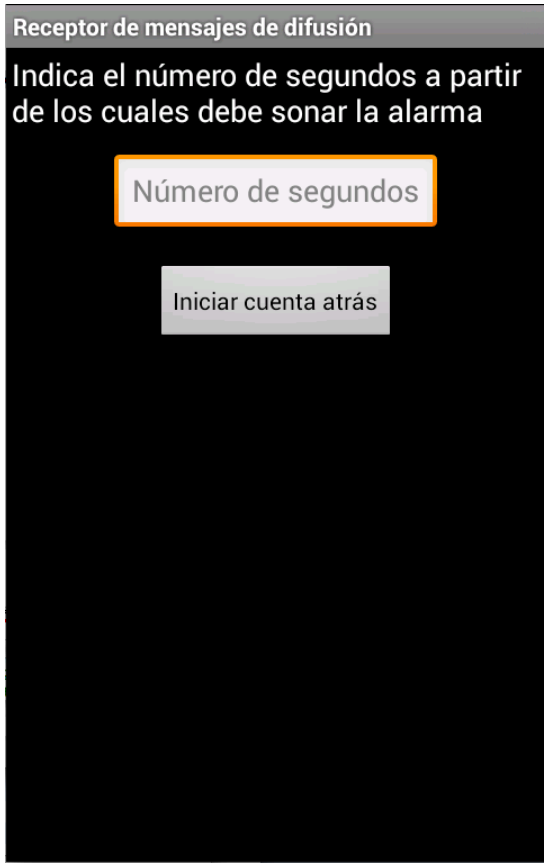
- La clase **ReceptorLlamadas** es subclase de BroastReceiver e implementa el receptor de mensajes de difusión contiene las siguientes sentencias:

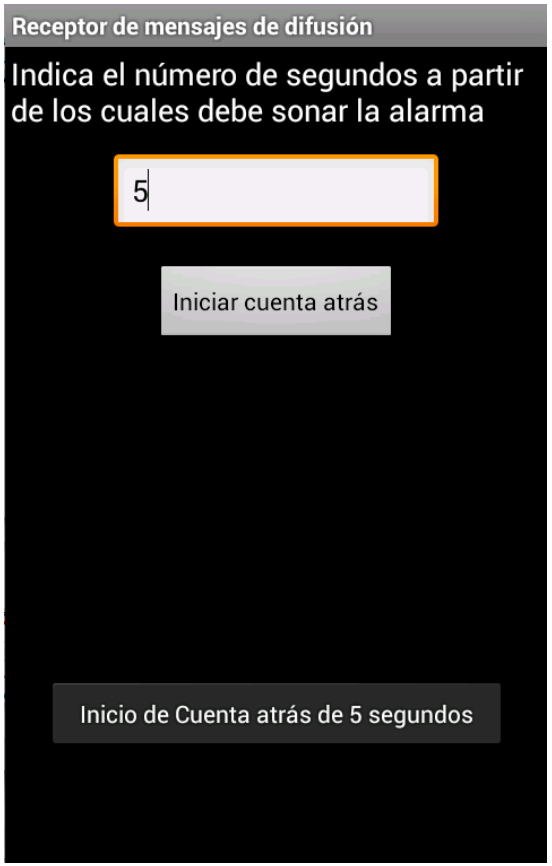
```
@Override
public void onReceive(Context context, Intent intent) {
   Bundle extras = intent.getExtras();
   if (extras != null) {
     String estado = extras.getString(TelephonyManager.EXTRA_STATE);
     Log.w("ESTADO TELEFONO", estado);
     if (estado.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
          String numeroTelefono= extras
                 .getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
          Log.w("NUMERO TELEFONO", numeroTelefono);
     }
   }
}
}
```

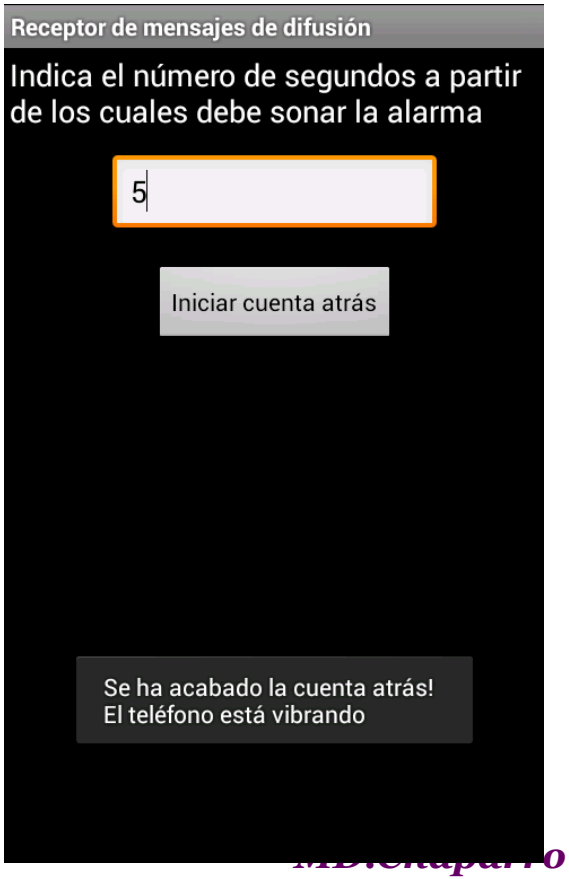
4. Receptores

Práctica:

Crear un receptor de alarma

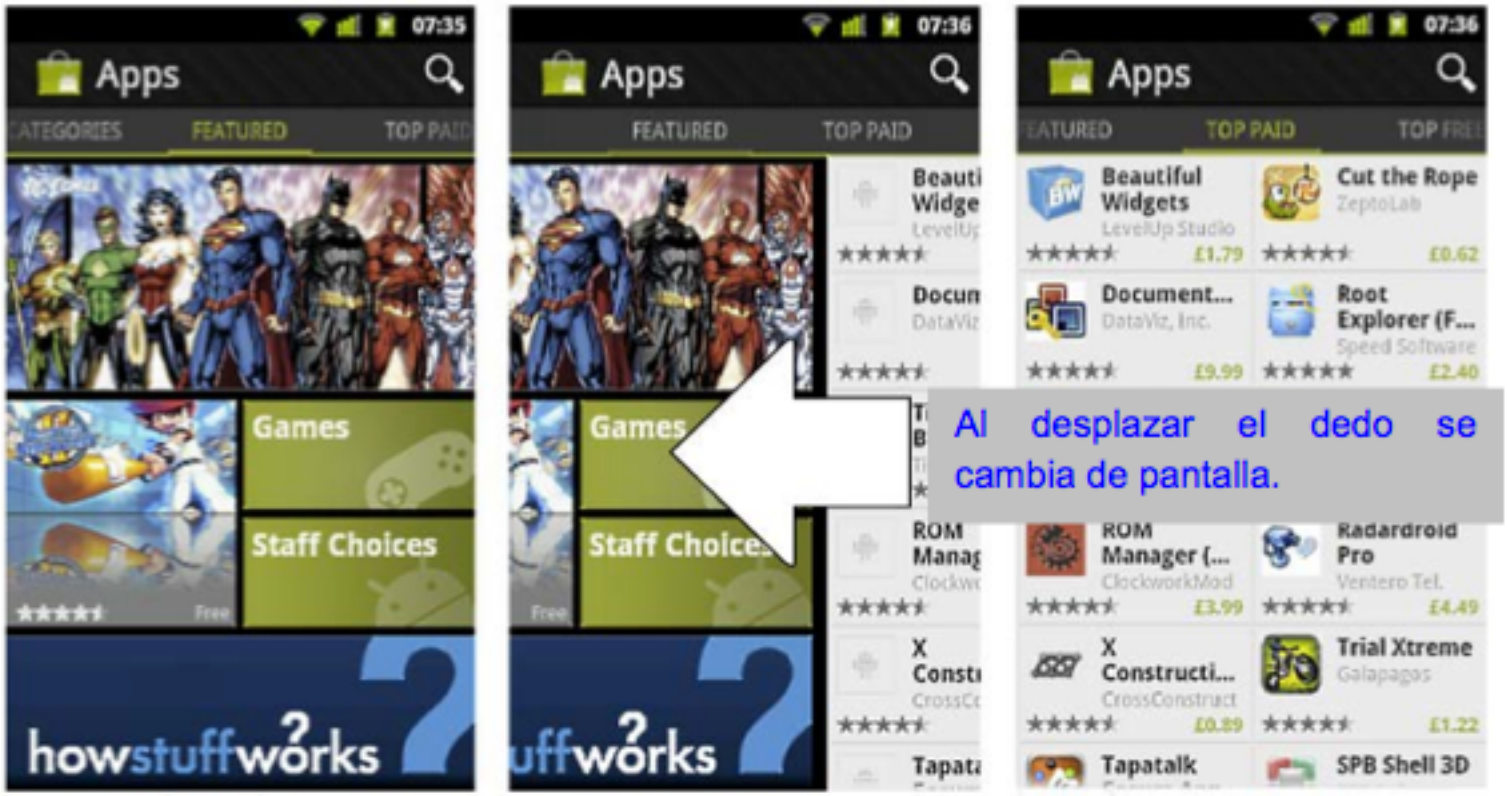






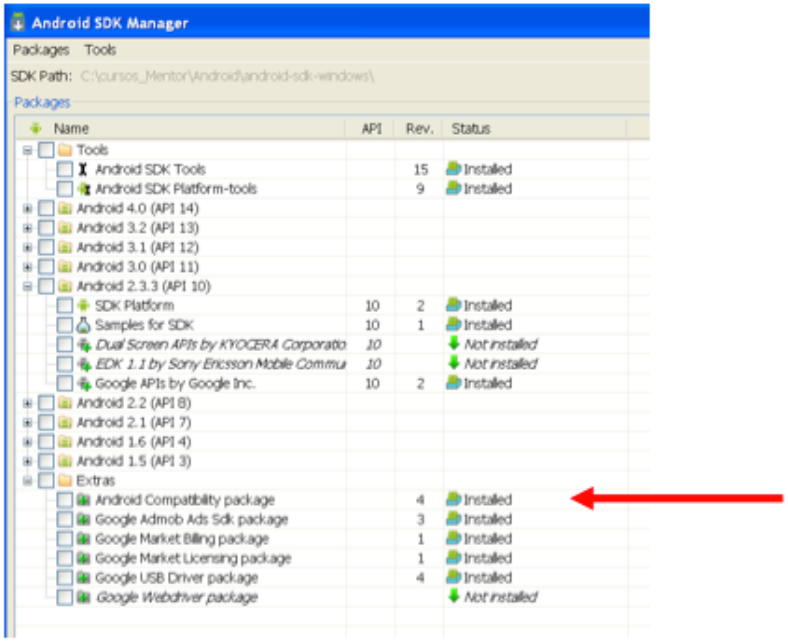
5. ViewPager

- Un **ViewPager** (heredado de ViewGroup)permiten desplazar páginas deslizando el dedo horizontalmente sobre la pantalla.



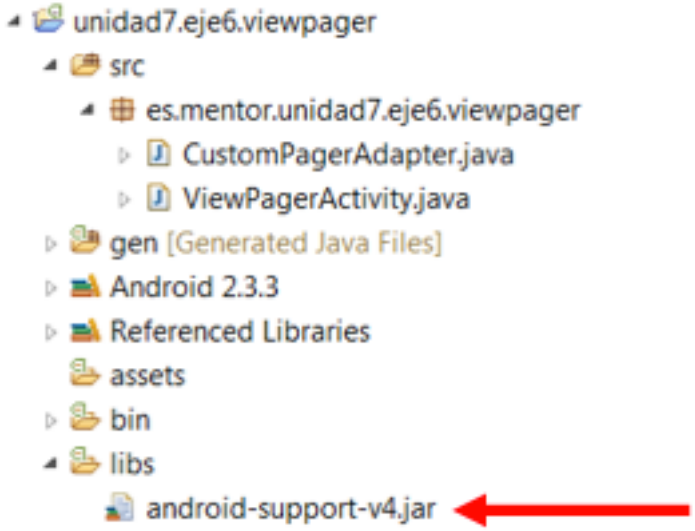
5. ViewPager

- Este componente puede no formar parte de las clases por defecto del SDK de Android.
- Está incluido en el paquete externo de **Compatibilidad** de Android que deberías haber añadido al instalar el SDK de Android en Eclipse. Para comprobar que está bien añadido, haz clic en el botón "**Opens the Android SDK Manager**" de Eclipse:



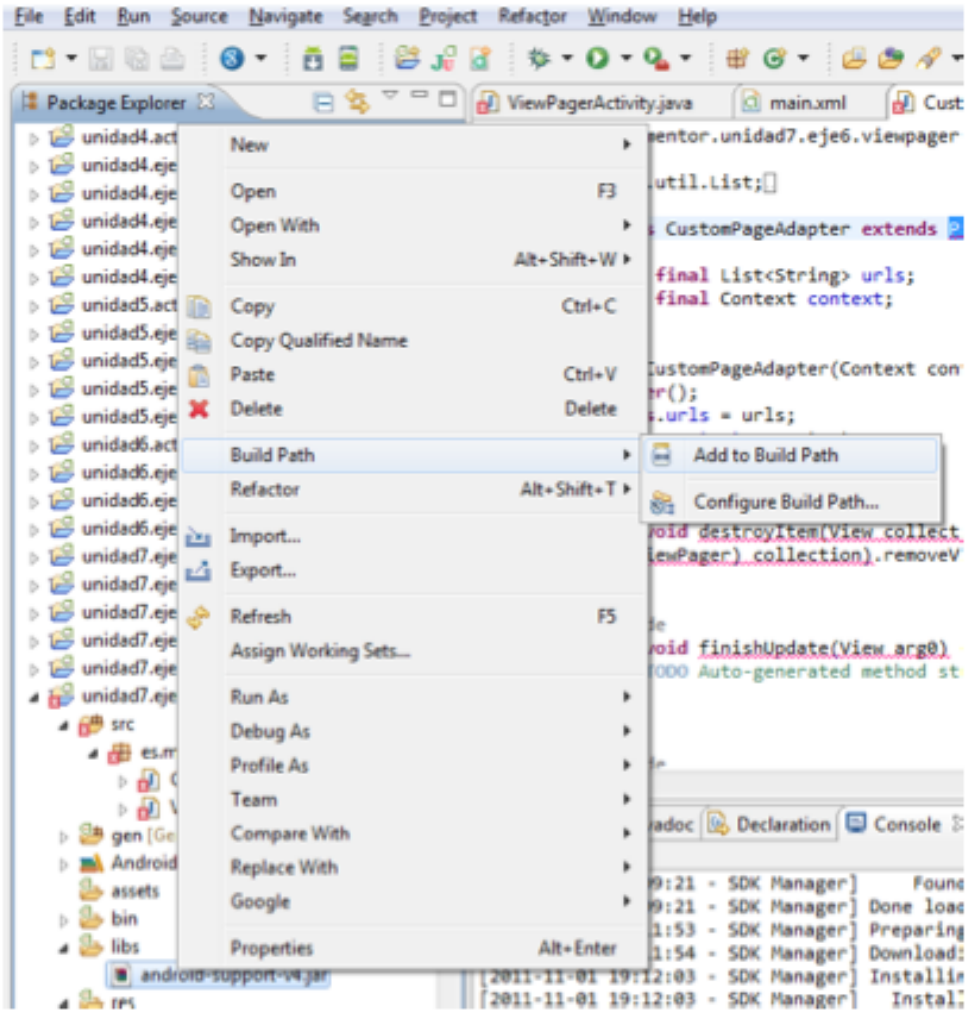
5. ViewPager

- Una vez que hemos comprobado que tenemos las librerías extra de compatibilidad de Android, procedemos a incluirlas en el proyecto.
- En este proyecto hemos creado la carpeta "**libs**" y copiado dentro el archivo **android- support-v4.jar** del directorio donde se encuentre la librería



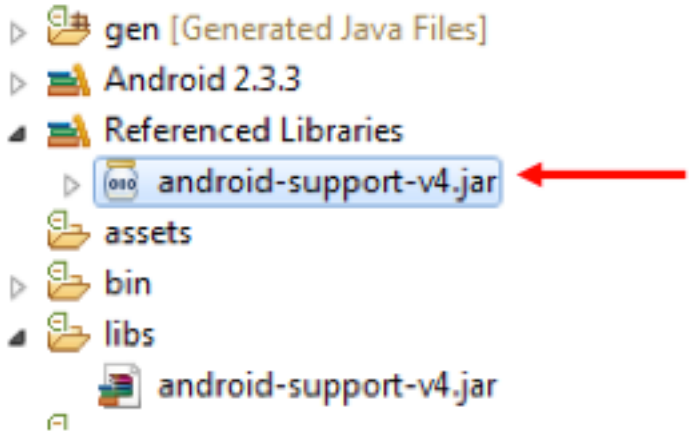
5. ViewPager

- A continuación, añadimos la librería al **Build Path** haciendo clic con el botón derecho del ratón sobre el archivo de la librería y eligiendo la opción "**Build Path->Add to Build Path**" del menú desplegable:



5. ViewPager

- Para comprobar que hemos incluido la librería correctamente en Eclipse, debe aparecer como Librería referenciada ("Referenced Libraries"):



5. ViewPager

- Para generar las páginas contenidas en este **ViewPager** es necesario usar un objeto **PagerAdapter**, que se encarga de alimentar de páginas al componente **ViewPager**.

```
private static int NUMERO_VIEWS = 10;
private ViewPager vPager;
private CustomPagerAdapter vPagerAdapter;
```

```
<?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"
    android:background="#a4c639">
<android.support.v4.view.ViewPager
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/vPager"/>
</LinearLayout>
```

5. ViewPager

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    vPager = (ViewPager) findViewById(R.id.vPager);

    vPagerAdapter = new CustomPagerAdapter(NUMERO_VIEWS, this);
    vPager.setAdapter(vPagerAdapter);

    // Definimos el evento de cambio de pagina en el ViewPager
    vPager.setOnPageChangeListener(new ViewPager.OnPageChangeListener() {
        @Override
        public void onPageSelected(int position) {
            Toast.makeText(getBaseContext(), "Has cambiado a la pantalla " + (position+1), 1).show();
        }

        @Override
        public void onPageScrollStateChanged(int arg0) {
            // No definimos nada en el evento al hacer scroll en la pagina
        }

        @Override
        public void onPageScrolled(int arg0, float arg1, int arg2) {
            // No definimos nada en el evento al hacer scroll en la pagina
        }
    }); // end setOnPageChangeListener
}
```

5. ViewPager

Práctica:

Crear una aplicaciónc on ViewPager