

# Desarrollo de Aplicaciones para Android

## Sesión 8: Servicios avanzados



# Puntos a tratar

- Servicios en segundo plano
- Notificaciones
- AppWidgets
- Publicación de software



# Servicios

- `Services`, analogía con los demonios de GNU/Linux.
- No necesitan una actividad abierta para seguir ejecutándose.
- La manera de controlarlos desde actividades
  - `startService(new Intent(main, MiServicio.class));`
  - `stopService(new Intent(main, MiServicio.class));`
- El servicio puede detenerse a si mismo con
  - `selfStop()`



# Servicios propios

- Heredan de la clase `Service`
- Implementan obligatoriamente el método `IBinder onBind(Intent)`
  - sirve para comunicación entre servicios y necesita que se defina una interfaz AIDL (Android Interface Definition Language).
  - Devolviendo `null` estamos indicando que no implementamos tal comunicación.
- Se declaran en el `AndroidManifest.xml`
- También se suelen sobrecargar los métodos
  - `onCreate()`, `onStartCommand(...)`, `onDestroy()`



# Servicios propios

```
public class MiServicio extends Service {

    @Override
    public void onCreate() {
        super.onCreate();
        //Inicializaciones necesarias
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //Comenzar la tarea de segundo plano
        return Service.START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        //Terminar la tarea y liberar recursos
    }

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



# Declarar el servicio

- En el `AndroidManifest.xml`

```
...  
    <service android:name=".MiServicio" />  
</application>
```

- Si el servicio se encontrara declarado dentro de otra clase, el `android:name` contendría: `.MiOtraClase$MiServicio`.

# Notificaciones

- Las `Notification` son el mecanismo típico que utilizan los servicios para comunicarse con el usuario.
- No roban el foco a la aplicación actual.
- Permanecen el tiempo que haga falta hasta ser vistas o descartadas
- Permiten mostrar distintos tipos de información
- Pueden responder a la pulsación
- Se pueden actualizar





# Notification Manager

```
Notification notification;  
int id = 1;  
NotificationManager notificationManager;  
notificationManager = (NotificationManager)getSystemService(  
    Context.NOTIFICATION_SERVICE);  
notification = new Notification(R.drawable.icon,  
    "Mensaje evento", System.currentTimeMillis());  
notificationManager.notify(id, notification);
```

- El `id` sirve para actualizar la misma notificación o poner una nueva, si es un `id` nuevo.
- También se puede modificar la información de una instancia ya creada de una `Notification`:

```
notification.setLatestEventInfo(getApplicationContext(),  
    "Texto", contentIntent);  
notificationManager.notify(id, notification);
```





# Contestar a la pulsación

```
notification.setLatestEventInfo(getApplicationContext(),  
                                "Texto", contentIntent);
```

- `contentIntent` es un `Intent` que se puede usar para abrir una actividad
- Al abrirse la actividad conviene cerrar la notificación que ya no será necesaria. Se puede hacer desde el método `onResume()` de la actividad:

```
@Override  
protected void onResume() {  
    super.onResume();  
    notificationManager.cancel(MiTarea.NOTIF_ID);  
}
```



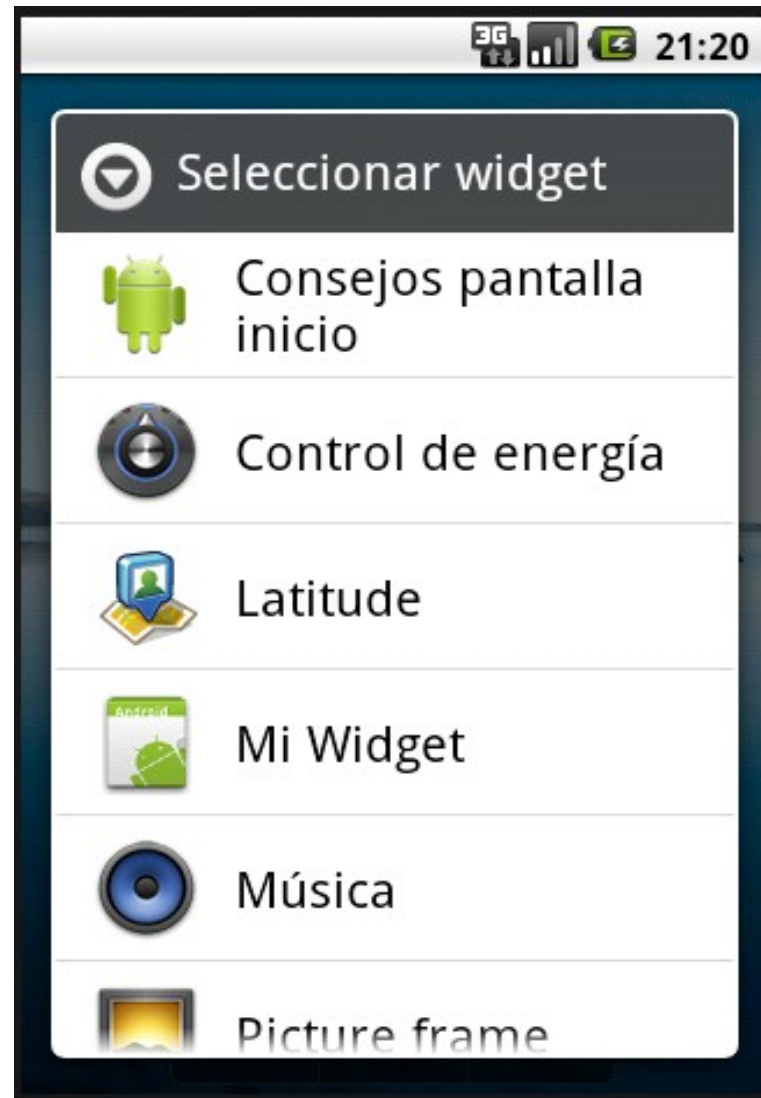
# Notificaciones con AsyncTask

```
private class MiTarea extends AsyncTask<String, String, String>{
    public static final int NOTIF_ID = 1;
    Notification notification;
    NotificationManager notificationManager;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        notificationManager = (NotificationManager) getSystemService(
            Context.NOTIFICATION_SERVICE);
        notification = new Notification(R.drawable.icon,
            "Mensaje evento", System.currentTimeMillis());
    }
    @Override
    protected String doInBackground(String... params) {
        while(condicionSeguirEjecutando){
            if(condicionEvento)
                publishProgress("Información del evento");
        }
        return null;
    }
    @Override
    protected void onProgressUpdate(String... values) {
        Intent notificationIntent = new Intent(
            getApplicationContext(), MiActividadPrincipal.class);
        PendingIntent contentIntent = PendingIntent.getActivity(
            getApplicationContext(), 0, notificationIntent, 0);
        notification.setLatestEventInfo(getApplicationContext(),
            values[0], contentIntent);
        notificationManager.notify(NOTIF_ID, notification);
    }
}
```

# AppWidgets

- También conocidos como widgets
- Ocupan determinado área del home (escritorio)
- Se refrescan con determinada frecuencia





# Componentes permitidos

- layouts
  - FrameLayout
  - LinearLayout
  - RelativeLayout
- views
  - AnalogClock
  - Button, Chronometer
  - ImageButton
  - ImageView
  - ProgressBar
  - TextView
- EditText no está permitido: emular con actividad al pulsar

# Android 3 y 4

- Nuevos componentes
  - GridView
  - ListView
  - StackView
  - ViewFlipper
  - AdapterViewFlipper





# Definir un widget

- Declaración necesaria en
  - En un recurso XML
  - En el `AndroidManifest.xml`
- Clase que herede de `AppWidgetProvider`
  - Normalmente habrá un servicio actualizando el widget.



# XML del widget

- Declara:
  - Área que ocupa
  - Layout que lo define
  - Período de actualización en milisegundos

```
<?xml version="1.0" encoding="utf-8"? >
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dip"
    android:minHeight="72dip"
    android:updatePeriodMillis="600000"
    android:initialLayout="@layout/miwidget_layout"
/>
```



# Widget y AndroidManifest.xml

- Declarar el widget, su recurso XML y, si procede, el servicio que lo actualiza

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.ua.jtech.ajdm.appwidget"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver android:name=".MiWidget" android:label="Mi Widget">
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                android:resource="@xml/miwidget" />
        </receiver>
        <service android:name=".MiWidget$UpdateService" />
    </application>
    <uses-sdk android:minSdkVersion="8" />
</manifest>
```





# Widget con servicio

```
public class MiWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
        // Inicio de nuestro servicio de actualización:
        context.startService(new Intent(context, UpdateService.class));
    }
    public static class UpdateService extends Service {
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
            ...
            ...
            return Service.START_STICKY;
        }
        @Override
        public IBinder onBind(Intent intent) {
            return null;
        }
    }
}
```



# RemoteViews

- Los views definidos en el layout del widget se actualizan a través de RemoteViews
- Ejemplo de layout básico:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/miwidget"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

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

</LinearLayout>
```



# RemoteViews

- Desde el servicio:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    RemoteViews updateViews = new RemoteViews(
        getPackageName(), R.layout.miwidget_layout);

    //Aqui se actualizarían todos los tipos de Views que hubiera:
    updateViews.setTextViewText(R.id.TextView01,
        "Valor con el que refrescamos");

    // ...

    //Y la actualización del widget con el updateViews creado:
    ComponentName thisWidget = new ComponentName(
        this, MiWidget.class);
    AppWidgetManager.getInstance(this).updateAppWidget(
        thisWidget, updateViews);

    return Service.START_STICKY;
}
```



# RemoteViews

- Si pulsamos algún componente, lanzar una actividad:

```
// ...
//Además, ¿qué hacer si lo pulsamos? Lanzar alguna actividad:
Intent defineIntent = new Intent(...);
PendingIntent pendingIntent = PendingIntent.getActivity(
    getApplicationContext(), 0, defineIntent, 0);
updateViews.setOnClickPendingIntent(R.id.miwidget, pendingIntent);

//Y la actualización del widget con el updateViews creado:
ComponentName thisWidget = new ComponentName(
    this, MiWidget.class);
AppWidgetManager.getInstance(this).updateAppWidget(
    thisWidget, updateViews);
```



# Publicar el software

- Para empaquetar la aplicación debemos
  - Poner el nombre de la aplicación, icono y versión.
  - Deshabilitar debugging en el AndroidManifest.xml (atributo `android:debuggable="false"` del tag de application).
  - Eliminar cualquier mensaje de Log.
  - Pedir sólo los permisos que de verdad la aplicación use, y no más de los necesarios.
  - Por supuesto, haber probado la aplicación en terminales reales, a ser posible en más de uno.



# Publicar el software

- Generar el paquete
  - Con el plug-in de Eclipse (menú contextual del proyecto > Android Tools > Export Signed Application Package)
  - Paquete con extensión .apk
  - Para publicar el paquete, este deberá ir firmado (signed).



# Firmar el paquete

- Conseguir una firma digital
  - De una autoridad certificadora conocida
  - O bien sin autoridad, auto-firmado (self-signed)
    - Se puede hacer con la herramienta `keytool`:

```
keytool -genkey -v -keystore myandroidapplications.keystore  
-alias myandroidapplications -keyalg RSA -validity 10000
```

- Firmar el paquete
  - Con Eclipse
  - O con la herramienta `jarsigned`

```
jarsigned -verbose -keystore myandroidapplications.keystore  
miaplicacion.apk myandroidapplications  
jarsigned -verbose -certs -verify miaplicacion.apk
```



# Certificados y versiones

- Los updates de la aplicación asumen que el certificado no va a cambiar.
- Si cambiamos el certificado la aplicación deberá ser desinstalada por completo (manualmente) antes de instalar la nueva versión con el nuevo certificado.
- De lo contrario, al hacer update, dará error de certificado porque no lo actualiza, sino que comprueba el antiguo.





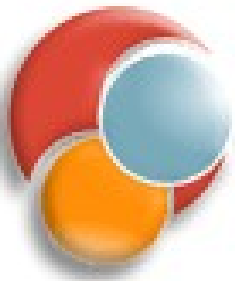
# Android Market

- Para publicar en Android Market debemos darnos de alta como desarrollador
  - 25\$ (a través de Google Checkout)
  - Cuenta de Google Checkout Merchant
- Subir la aplicación
  - Pedirá información en varios idiomas, países en los que funciona, tipo, categoría, precio, información de copyright, contacto de soporte.



# Publicación independiente

- Siempre se puede colgar el paquete.apk en un servidor propio.
- Para que el usuario lo pueda instalar tras descargarlo tendrá que habilitar la opción de instalar software desde fuentes desconocidas.



# ¿Preguntas...?