Notificaciones y AppWidgets - Ejercicios

Índice

1. Servicio con notificaciones: Números primos

El proyecto `android-av-notificaciones` de la plantilla tiene un servicio con una tarea que va calculando números primos a un ritmo lento. Vamos a mostrar una `Notification` en la barra de tareas cada vez que se descubra un primo y la iremos actualizando con la llegada de cada nuevo número. Si salimos de la aplicación sin parar el servicio, seguirán apareciendo notificaciones, y si pulsamos sobre la notificación, volverá a lanzarse la actividad, cerrándose la notificación que hemos pulsado.

- Dentro del servicio `MiNumerosPrimosServicio` se encuentra declarada la `AsyncTask` llamada `MiTarea`. En ella tenemos como campos de la clase una `Notification` y un `NotificationManager`. Hay que darles valores en el método `onPreExecute()`.
- El método `doInBackground(...)` ejecutará un bucle que irá incrementando `i` mientras su valor sea menor de `MAXCOUNT`. En cada iteración, si el número es primo (función incluida en la plantilla), pedirá que se muestre el progreso, pasándole como parámetro el nuevo primo encontrado.
- Implementar el método `onProgressUpdate(...)` para que muestre la notificación. Para ello habrá que actualizar la notificación con el método `setLatestEventInfo`, al cuál le pasaremos en un String la información del último primo descubierto y le pasaremos un `PendingIntent` para que al pulsar sobre la notificación, nos devuelva a la actividad de la aplicación, por si la hemos cerrado. Para crear el PendingIntent utilizaremos el método `PendingIntent.getActivity(...)` al cuál le tenemos que pasar un `new Intent(getApplicationContext(),Main.class)`.
- La aplicación debería funcionar en este punto, mostrando las notificaciones y relanzando la aplicación si son pulsadas, pero no cerrándolas al pulsarlas. Para ello simplemente tenemos que llamar al método `cancel(id)` del `notificationManager` y pasarle la constante NOTIF_ID para que la notificación no se muestre como una nueva, sino como actualización de la que ya habíamos puesto. Una manera de hacerlo es en un método estático del `MiNumerosPrimosServicio`, que ya está creado en las plantillas del ejercicio y se llama `cerrarMiNotificacion(NotificationManager nm)`. Debes invocar este método desde el `Main.onResume()`.

Notificación del servicio de números primos

2. IP AppWidget

Vamos abrir el proyecto `android-av-appwidget` para construir un AppWidget de Android, que nos muestre una frase célebre y la hora.

En el proyecto pulsamos con el boton derecho y añadimos un nuevo Android XML File, de tipo AppWidget Provider, que se llame `miwidget.xml`. El editor nos permite pulsar sobre el AppWidget Provider y editar sus atributos. Ponemos los siguientes:

```
    android:minWidth="146dip"
    android:minHeight="72dip"
    android:updatePeriodMillis="4320000"
    android:initialLayout="@layout/miwidget_layout"
```

El layout `miwidget_layout.xml` no lo tenemos que crear porque ya está incluido en el proyecto.

Creamos una clase `MiWidget` que herede de `AppWidgetProvider`, en el paquete `es.ua.jtech.av.appwidget`. Sobrecargamos su método `onUpdate(...)` y su método `onReceive(...)`. En este último comprobaremos si se ha recibido un intent con una acción personalizada, la `es.ua.jtech.av.ACTION_WIDGET_CLICK`:

```
public class MiWidget extends AppWidgetProvider {
        public static final String ACTION_WIDGET_CLICK =
"es.ua.jtech.av.ACTION_WIDGET_CLICK";

        @Override
        public void onUpdate(Context context, AppWidgetManager
appWidgetManager,
                        int[] appWidgetIds) {
```

```
                        //TODO actualizar
        }

        @Override
        public void onReceive(Context context, Intent intent) {
                super.onReceive(context, intent);
                if(intent.getAction().equals(ACTION_WIDGET_CLICK)){
                        //TODO actualizar
                }
        }
}
```

En ambos casos la actualización sería la misma, así que vamos a extraerla en un método privado de la clase, `private void actualizar(Context context)` que llamaremos desde `onUpdate()` y desde `onReceive()`. El método utilizará los `RemoteViews` para actualizar el campo de texto. Lo actualizará con una frase aleatoria de entre las definidas dentro de un array de strings en el recurso `strings.xml` que viene incluido en el proyecto de las plantillas. También se asignará un comportamiento al hacer click sobre el widget que consistirá en enviar un broadcast intent con la acción que hemos definido:

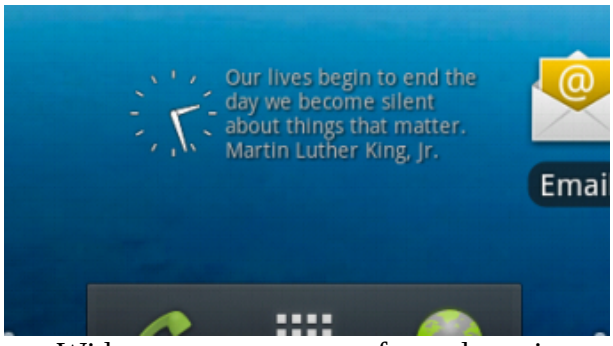
```
RemoteViews updateViews = new RemoteViews(context.getPackageName(),
R.layout.miwidget_layout);

//Seleccionar frase aleatoria
String frases[] = context.getResources().getStringArray(R.array.frases);
updateViews.setTextViewText(R.id.TextView01,
frases[(int)(Math.random()*frases.length)]);

//Comportamiento del botón
//On-click listener que envía un broadcast intent
Intent intent = new Intent(ACTION_WIDGET_CLICK);
PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0,
intent, 0);
updateViews.setOnClickPendingIntent(R.id.miwidget, pendingIntent);

ComponentName thisWidget = new ComponentName(context, MiWidget.class);
AppWidgetManager.getInstance(context).updateAppWidget(thisWidget,
updateViews);
```

En el `AndroidManifest.xml`, dentro de `<application>` declararemos el receiver de nuestro widget con dos intent filters, uno para la acción del sistema `android.appwidget.action.APPWIDGET_UPDATE`, y otro para la que utilizamos para forzar la actualización desde la clase `MiWidget`:

```
        <receiver android:name=".MiWidget" android:label="Frases Widget">
            <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <intent-filter>
                <action android:name="es.ua.jtech.av.ACTION_WIDGET_CLICK"
/>
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                    android:resource="@xml/miwidget" />
        </receiver>
```

Ejecutamos el widget desde Eclipse, como aplicación android, y comprobamos que no ocurra ningún error en la consola de Eclipse. Ya se puede añadir el widget en el escritorio, efectuando una pulsación larga sobre una porción de área libre del escritorio, y seleccionando nuestro widget.

En Android 4.0 la pulsación larga no nos muestra la opción de añadir widget sino que hay que ir al menú de aplicaciones y seleccionar la pestaña de widgets. Para insertar uno se tiene que pulsar prolongadamente para arrastrarlo a una zona libre del escritorio. Añadimos el widget y observamos el resultado:



Widget que muestra una frase aleatoria

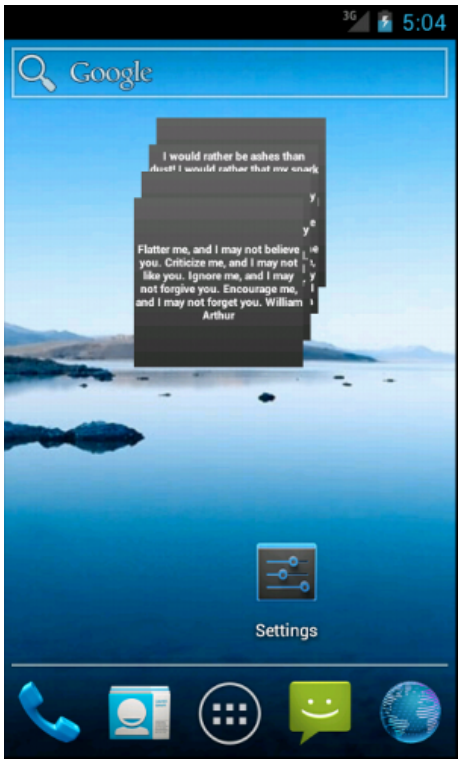
3. StackWidget (*)

Vamos a probar una de las características novedosas de los widgets a partir de Android 3.0, se trata del `StackView` que nos permite pasar "tarjetas" hacia delante y hacia atrás. Ejecuta el proyecto `android-av-stackwidget` de las plantillas y comprueba que funciona correctamente en un emulador con alguna de las últimas versiones de Android.

Vamos a modificar el widget para que muestre las frases del ejercicio anterior. Para ello tenemos que copiar y pegar el array de strings del recurso `strings.xml`. A continuación editamos la clase `StackWidgetProvider` y le añadimos un campo `private String [] frases`. Las podemos inicializar en en el constructor:

```
frases = context.getResources().getStringArray(R.array.frases);
```

En el método `onCreate` sustituiremos el bucle con las llamadas a `mWidgetItems.add(new WidgetItem(...))` por un bucle que añada todas las cadenas de `frases`. El resultado debe quedar así:



StackWidget con frases