Servicios - Ejercicios

Índice

1. Servicio con proceso en background. Contador

Los servicios se utilizan para ejecutar algún tipo de procesamiento en background. En este ejercicio vamos a crear nuestro propio proceso asociado a un servicio, que ejecute determinada tarea, en este caso, contar desde 1 hasta 100, deteniéndose 5 segundos antes de cada incremento. En cada incremento mostraremos un `Toast` que nos informe de la cuenta.

En las plantillas tenemos el proyecto `android-av-serviciocontador` que ya incluye la declaración del servicio en el manifest, la actividad que inicia y detiene el servicio, y el esqueleto del servicio `MiCuentaServicio`.

- En el esqueleto que se proporciona, viene definida una extensión de `AsyncTask` llamada `MiTarea`. Los métodos `onPreExecute`, `doInBackground`, `onProgressUpdate` y `onCancelled` están sobrecargados pero están vacíos. Se pide implementarlos, el primero de ellos inicializando el campo `i` que se utiliza para la cuenta, el segundo ejecutando un bucle desde 1 hasta 100, y en cada iteración pidiendo mostrar el progreso y durmiento después 5 segundos con `Thread.sleep(5000)`. El tercer método, `onProgressUpdate` mostrará el `Toast` con el progreso, y por último el método de cancelación pondrá el valor máximo de la cuenta para que se salga del bucle.
- En los métodos del servicio, `onCreate`, `onStartCommand` y `onDestroy`, introduciremos la creacion de la nueva `MiTarea`, su ejecución (método `execute()` de la tarea) y la cancelación de su ejecución (método `cancel()` de la tarea).

Una vez más, el servicio deberá seguir funcionando aunque se salga de la aplicación y podrá ser parado entrando de nuevo en la aplicación y pulsando Stop.

2. Dialer. Iniciar una actividad con un evento broadcast (*)

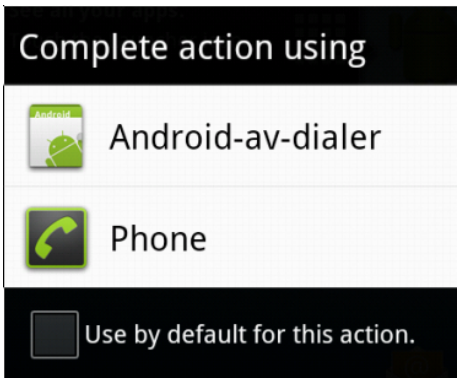
Se pide crear un proyecto `android-av-dialer` con el paquete `es.ua.jtech.av.dialer` y una activdad `Main` que muestre el texto "Dialer personalizado".

Esta actividad se iniciará cuando se inicie un marcado de llamada telefónica. Para ello debes añadir otro intent filter al `AndroidManifest.xml` para que además de iniciarse ante el evento de lanzamiento de la aplicación, se inicie ante un evento `DIAL`:

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

Al ejecutar la aplicación desde Eclipse no ocurrirá nada más que su instalación en el dispositivo. Para ejecutarla se debe iniciar un marcado telefónico. Es posible que no funcione en el emulador con Android 4.0. Además, a partir de Android 3.1 es necesario

ejecutar la actividad de la aplicación al menos una vez para que el BroadcastReceiver quede registrado.



Diálogo que ofrece realizar la llamada con nuestro dialer

Para implementar un programa marcador de verdad, necesitarás algunos de los permisos siguientes:

```
<uses-permission android:name="android.permission.CALL_PRIVILEGED" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"
/>
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Se pide añadir dos botones para marcar dos números de teléfono favoritos (preferiblemente inventados). El marcado se puede iniciar de la siguiente manera:

```
Intent dialIntent = new
Intent(Intent.ACTION_DIAL,Uri.parse("tel:+3400000000"));
startActivity(dialIntent);
```

3. Arranque. Iniciar servicio con evento broadcast

En el proyecto `android-av-onboot` tenemos un servicio que se lanza al inicar la actividad, mostrando Toasts del 1 al 5 y finalizando.

Se pide registrar el servicio para que se inicie cuando el móvil haya terminado de arrancar. Para ello añadiremos el un permiso al `AndroidManifest.xml`:

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

También registraremos un receptor de broadcast `MiBroadcastReceiver` con el intent filter para `android.intent.action.BOOT_COMPLETED`.

Implementaremos el `MiBroadcastReceiver` y en su método `onReceive()`

introduciremos el código:

```
if( "android.intent.action.BOOT_COMPLETED".equals(intent.getAction())) {
        ComponentName comp = new ComponentName(context.getPackageName(),
                                        MiServicio.class.getName());
        ComponentName service = context.startService(
                                        new Intent().setComponent(comp));
        if (null == service){
                Log.e(MiBroadcastReceiver.class.toString(),
                  "Servicio no iniciado: " + comp.toString());
        }
} else {
        Log.e(MiBroadcastReceiver.class.toString(),
          "Intent no esperado " + intent.toString());
}
```

Tras ejecutar el proyecto desde Eclipse habrá que reiniciar el emulador para observar que el servicio se inicia al finalizar el arranque. Es necesario reiniciarlo sin cargar desde un estado anterior guardado para que el sistema Android arranque desde cero.

4. Localizador de móvil desaparecido

No sería fácil detectar por software que nuestro móvil ha desaparecido, pero podemos programar un servicio que responda a determinado evento que nosotros causemos a través de la red móvil. Lo más sencillo, para el caso de que no haya red, sería que al recibir un mensaje SMS con un contenido determinado, el teléfono respondiera con un SMS (a ese mismo número o a otro que especifiquemos) indicando su localización.

En las plantillas de la sesión contamos con el BroadcastReceiver `Localizador` que podemos añadir a un nuevo proyecto que llamaremos `android-av-localizador` con una actividad principal `Main` en el paquete `es.ua.jtech.av.localizador`.

Registraremos el receptor de broadcast `Localizador` a través del `AndroidManifest.xml`. Dejaremos también la actividad `Main` para que pueda ser lanzada desde el menú. Es necesaria a partir de Android 3.1 para que quede efectuado el registro. Podríamos poner algún texto explicativo, "Se ha registrado el servicio de localizador".

También harán falta los permisos RECEIVE_SMS, SEND_SMS, ACCESS_COURSE_LOCATION, ACCESS_FINE_LOCATION.

Para probar el programa se pueden abrir dos emuladores y enviar SMS al número de puerto que viene indicado en la barra de cada emulador. Para establecer una localización simulada podemos utilizar Eclipse con la vista DDMS/Emulator control, o bien

```
telnet localhost <puerto>
geo fix <longitude> <latitude> [<altitude>]
```

5. AIDL (*)

Crea un proyecto llamado `android-av-aidl-calculadora`. con un servicio `ServicioCalculadora` que ofrezca los métodos `float suma(float, float)` y `float eleva(float, int)`.

La actividad principal `CalcActivity` debe mostrar interfaz gráfica muy sencilla que nos permita llamar al servicio del otro proyecto. Es importante hacer el `unbindService()` al cerrarse la actividad.