Desarrollo para dispositivos móviles con Android: app multiactividades.

Contenido

1 Introducción

En realidad es extraño que una aplicación Android tenga tan solo una actividad. Solo aquellas más sencillas pueden diseñarse así. En este tema se explicará con detalle cómo crear nuevas actividades, que son llamadas desde otra actividad ya existente, y bloquean a esta hasta que terminan su ejecución.

2 Creando una nueva actividad

Una nueva **Activity** (actividad) precisa de varios componentes para llevar a cabo sus funciones: un layout (bajo *res/layout*), una entrada descriptiva en *AndroidManifest.xml*, y una clase *Java* que derive de **Activity**, de tal forma que reescriba al menos el método **Activity**.*onCreate()*.

Para especificar la actividad en el manifiesto, tan solo es necesario añadir una línea como la que sigue:

```
<activity android:name=".Actividad2"/>
```

Con la línea superior, se especifica que que existe una actividad identificada por la clase Java **Actividad2**. Así, un AndroidManifest.xml con una actividad principal y otra subactividad quedaría como sigue.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest …

    <application
        …
        <activity android:name=".view.MainActivity">
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
        </activity>
        <activity android:name=".view.Actividad2"></activity>
    </application>
</manifest>
```

En el ejemplo más arriba, se especifica que la clase **Actividad2** se encuentra en el package *View* bajo el pacjage principal de la aplicación. Como ya se ha discutido previamente, es muy recomendable efectuar una separación en paquetes de aquel código específico de Android y la lógica de la aplicación.

Solo queda crear el *layout* (*layout_actividad2.xml*), y enlazarlo con la clase de la siguiente forma:

```
public class Actividad2 extends Activity {
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate( savedInstanceState );

                // Indica el layout para esta actividad.
                // El archivo de layout se guarda en res/layout/layout_actividad2.xml
                this.setContentView( R.layout.layout_actividad2 );
        }
}
```

3 Lanzando la actividad

Existen dos manera de ejecutar la actividad: de manera independiente, de forma que esta permite al usuario realizar una tarea sobre la que la actividad principal no precisa ser informada en cuanto a su resultado (método **Activity**.*startActivity(intent)*), o al contrario, obteniendo un resultado (método **Activity**.*startActivityForResult(intent, codigo)*), con lo que se puede saber, por ejemplo, si el usuario llevó a cabo la tarea en la actividad o prefirió, en caso contrario, fue cancelada.

Un ejemplo de actividad independiente podría ser una actividad que mostrara la información legal de la aplicación o de ciertos servicios de la empresa, es decir, a título meramente informativo. El resto de posibilidades caen ya dentro de la necesidad de obtener un resultado: un ejemplo típico sería la edición de ciertos datos (se quiere saber si el usuario va a guardar realmente esas modificaciones o no).

3.1 Actividades independientes

Es la forma más sencilla de lanzar una actividad. Solamente es necesario crear un objeto **Intent**, que contendrá una referencia a la actividad que lanza la subactividad y la clase de la subactividad que se va a lanzar (en el ejemplo, **Activity2**). Este **Intent** se pasa como argumento a **Activity**.*startActivity()*. En el siguiente ejemplo, se muestra cómo se realiza el lanzamiento de la segunda actividad, así como la vuelta desde la actividad principal. Ambas tareas se realizan en respuesta a la interacción del usuario mediante un botón.

```
// MainActivity
class MainActivity extends Activity {
	protected void onCreate(…
	{
		// más cosas…

		Button btLanza = (Button) this.findViewById(…
		btLanza.setOnClickListener( new OnClickListener…
			public void onClick(…
			{
				this.startActivity( new Intent( this, Actividad2.class ) );
			}
		);
	}

	// más cosas...
}

// Actividad2
class Actividad2 extends Activity {
	protected void onCreate(…
	{
		// más cosas…

		Button btVuelve = (Button) this.findViewById(…
		btVuelve.setOnClickListener( new OnClickListener…
			public void onClick(…
			{
				this.finish();
			}
		);
	}

	// más cosas…
}
```

La subactividad (en este ejemplo, **Activity2**), una vez ha terminado su trabajo, puede llamar a **Activity**.*finish()*, con lo que el control retorna a la actividad original. Por ejemplo, se puede crear un botón (*btVuelve* en el ejemplo) en pantalla con el texto “terminar” o similar, e incluir una llamada a **Activity**.*finish()* en su **OnClickListener**. El usuario siempre tendrá la opción de pulsar la tecla de retorno en el teléfono.

3.2 Códigos de retorno válidos

Los códigos de retorno válidos que puede devolver una subactividad son los siguientes. Existen varias constantes en la clase **Activity** que pueden ser usadas. Como se verá en la sección siguiente, estos códigos se pueden especificar como resultado desde una subactividad (utilizando **Actividad**.*setResult(codigo)*) para indicar, por ejemplo, la intención del usuario (lo más común), o cualquier otro dato.

Código	Constante	Significado
-1	RESULT_OK	Correcto
0	RESULT_CANCEL	Cancelado
1	RESULT_FIRST_USER	Código de usuario

El valor 1 (constante RESULT_FIRST_USER) , es el primer valor que el usuario puede utilizar de manera particular para satisfacer cualquier interés de su aplicación. Es decir, mientras los valores 0 y -1 están reservados, es posible devolver 125, por poner un ejemplo, para indicar un resultado especial a devolver por la subactividad. Nótese que, de manera predeterminada, al pulsar el botón "atrás" del teléfono desde una subactividad, el código devuelto es el 0 (**Activity.**RESULT_CANCEL).

3.3 Obteniendo un resultado

En este caso, es necesario indicar, con **Activity**.*setResult()*, el resultado de la ejecución de la subactividad. Es la forma más parecida de utilizar una actividad como un diálogo, si bien pudiendo ocupar toda la pantalla. Por ejemplo, se pueden crear dos botones, uno con el texto "Guarda", y otro con el botón "Cancela", de forma que en sus **OnClickListener**, respectivamente, se indique el resultado -1 (**Activity.**RESULT_OK) y 0 (**Activity.**RESULT_CANCEL).

El lanzamiento de la actividad se hace ahora con **Activity**.*startActivityForResult(intent, requestCode)*. Se pueden lanzar varias actividades distintas también con distintas intenciones, de ahí que se incluya un código de petición (*request code*) que permita diferenciar entre ellas.

En el siguiente ejemplo, se lanza una actividad esperando que retorne un valor según ha realizado la tarea esperada, o no. El resultado se obtiene en la actividad origen, de manera que es necesario reescribir el método **Activity**.*onActivityResult(requestCode, resultCode, intent)*. Dado que este método es una forma centralizada de tratar el resultado de todas las subactividades, el código incluido en la llamada se emplea para reconocer dicho resultado. Así, comprobando mediante un *if* si *resultCode* es Activity.RESULT_OK, y *requestCode* se corresponde con la subactividad en cuestión, se pueden aplicar los cambios realizados por el usuario en la subactividad.

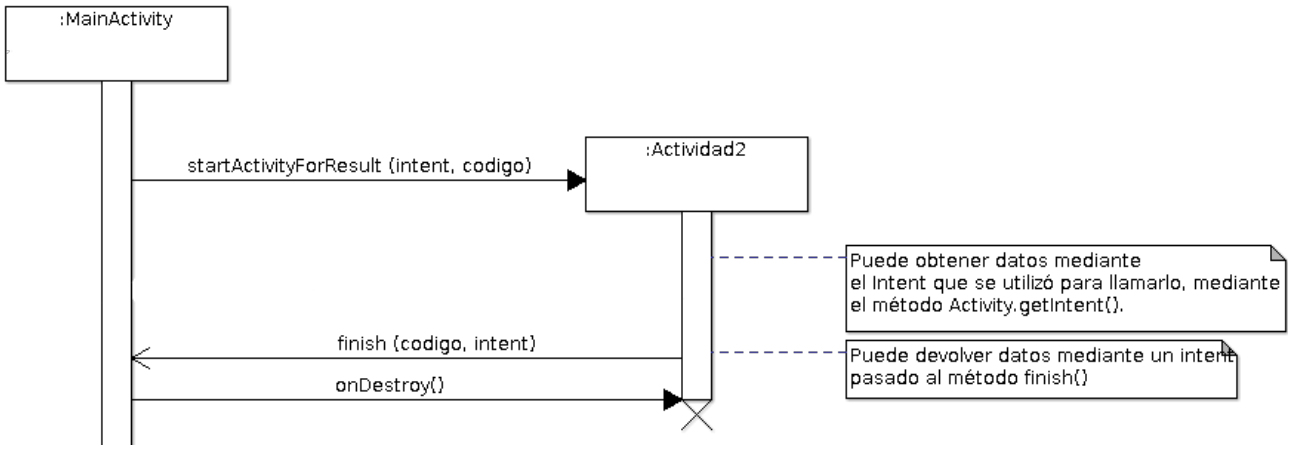
Como se verá en la sección siguiente, el usuario puede devolver datos mediante un objeto **Intent**, de ahí el parámetro final de *onActivityResult()*.

```
// MainActivity
class MainActivity extends Activity {
        private final int REQUEST_CODE = 1;

        protected void onCreate(…
        {
                // más cosas…

                Button btLanza = (Button) this.findViewById(…
                btLanza.setOnClickListener( new OnClickListener…
                        public void onClick(…
                        {
                                this.startActivityForResult(
                                        new Intent( this, activity2.class ), REQUEST_CODE );
                        }
                );
        }

        public void onActivityResult(int requestCode, int resultCode, Intent data)
        {
            if ( requestCode == REQUEST_CODE
              && resultCode == RESULT_OK )
            {
                 // …
            }
        }

        // más cosas...
}

// Actividad2
class MainActivity extends Activity {
        protected void onCreate(…
        {
                // más cosas…

                Button btVuelve = (Button) this.findViewById(…
                btVuelve.setOnClickListener( new OnClickListener…
                        public void onClick(…
                        {
                                this.setResult( RESULT_OK );
                                this.finish();
                        }
                );
        }

        // más cosas…
}
```

4 Compartiendo datos

La forma más sencilla de compartir datos es la sección “extra” del objeto **Intent**. Mediante esta sección extra, de una manera muy parecida al objeto **SharedPreferences** (*extra* es, de hecho un objecto **Bundle**), es posible guardar cadenas, enteros, etc., en formato de pares clave-valor. Estos datos se pueden “pasar” a la subactividad, y así mismo, la subactividad puede crear un nuevo **Intent** para devolver sus propios datos: creando un nuevo objeto **Intent** y utilizando sus extras. Para ello, se devuelven los datos como un parámetro extra de **Activity**.*setResult(**int** codigoResultado, **Intent** datosRetorno)*.

```
// MainActivity
class MainActivity extends Activity {
        private final int REQUEST_CODE = 1;

        protected void onCreate(…
        {
                // más cosas…

                Button btLanza = (Button) this.findViewById(…
                btLanza.setOnClickListener( new OnClickListener…
                        public void onClick(…
                        {
                                Intent myIntent = new Intent( v.getContext(), activity2.class );
                                myIntent.putExtra( “msg”, “hola” );
                                this.startActivityForResult( myIntent, REQUEST_CODE );
                        }
                );
        }

        public void onActivityResult(int requestCode, int resultCode, Intent datosDevolver)
        {
            if ( requestCode == REQUEST_CODE
              && resultCode == RESULT_OK )
            {
                String datos = datosDevolver.getExtras().get( “data” );

                // más cosas…

            }
        }

        // más cosas...
}

// Actividad2
class MainActivity extends Activity {
        protected void onCreate(…
        {
                // más cosas…

                Button btVuelve = (Button) this.findViewById(…
                btVuelve.setOnClickListener( new OnClickListener…
                        public void onClick(…
                        {
                                Intent datosDevolver = new Intent();
                                datosDevolver.putExtra( "data", edData.getText().toString() );

                                this.setResult( RESULT_OK, datosDevolver );
                                this.finish();
                        }
                );
        }

        // más cosas…
}
```



5 Sustituyendo la clase Application por defecto

Para aquellos datos que sea necesario mantener vivos y accesibles durante toda la ejecución de la aplicación, es conveniente crear nuestra propia clase derivada de **Application**, que sustituirá a la clase **Application** por defecto. Para hacer esto, solamente es necesario crear una clase como la que se muestra a continuación. Debe tenerse en cuenta que la clase derivada de **Application** debe ser lo más rápida posible, es decir, debe emplear tan poco tiempo como sea posible en su método *onCreate()*. Este método es, con diferencia, el más útil de la clase **Application**. Si bien existe un método **Application**.*onTerminate()*, este solamente es invocado cuando la aplicación se ejecuta en un emulador, por lo que no es realmente de interés.

```
import android.app.Application;
import ...core.Datos;

/**
 * Reemplazando el Application por defecto
 */
public class MiApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        this.datos = new Datos();
    }

    public Datos getDatos() {
        return this.datos;
    }

    private Datos datos;
}
```

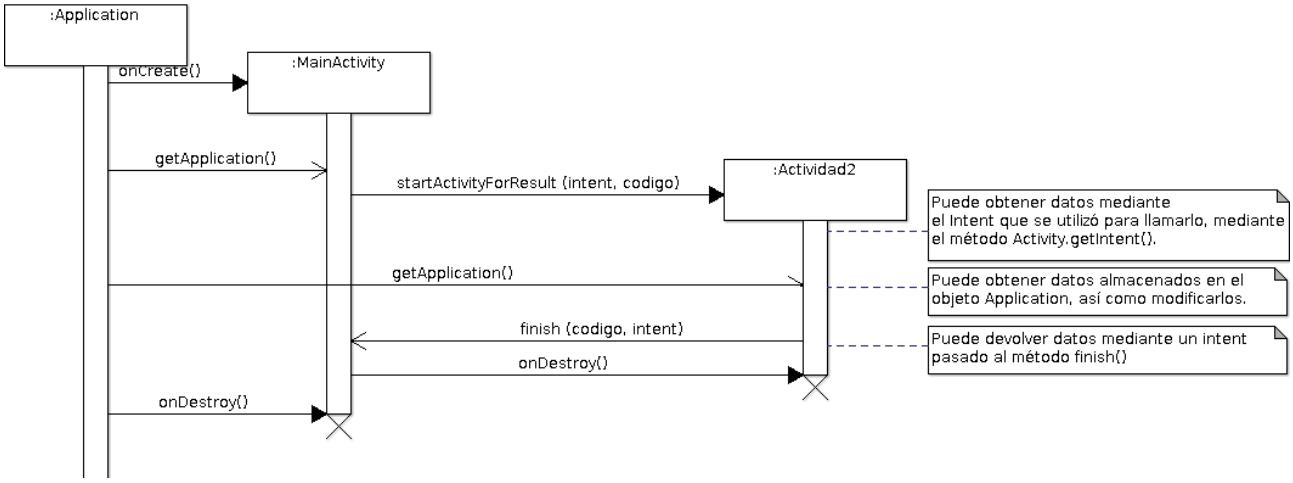
Además, será necesario cambiar el archivo *AndroidManifest.xml* para indicar cuál será nueva la clase aplicación. Esta información se indica mediante el atributo *android:name* dentro de la etiqueta *Application*, que ya existe por defecto en cualquier *AndroidManifest.xml*. Es decir, solamente es necesario añadir *android:name=”MiApp”* a la etiqueta *Application*. Por ejemplo:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"...
    ...
    <application android:name=".MiApp"
                 android:label="@string/app_name"
                 android:icon="@drawable/ic_launcher">
    ...
</manifest>
```

La clase anterior crea un objeto **Datos** que será accesible durante toda la aplicación, desde cualquier actividad, como se puede ver más abajo, mediante el método **Activity**.*getApplication()*.

```
// MainActivity
class MainActivity extends Activity {
        protected void onCreate(…
        {
                // más cosas…

                this.datos = ( (MiApp) this.getApplication() ).getDatos();

                // más cosas…

                Button btLanza = (Button) this.findViewById(…
                btLanza.setOnClickListener( new OnClickListener…
                        public void onClick(…
                        {
                                this.startActivity( new Intent( this, Actividad2.class ) );
                        }
                );
        }

        // más cosas…
        private Datos datos;
}
```

```
// Actividad2
class Actividad2 extends Activity {
        protected void onCreate(…
        {
                super.onCreate( savedInstanceState );
                 this.setContentView( R.layout.manager );

                 this.datos = ( (MiApp) this.getApplication() ).getDatos();

                // más cosas…

                Button btVuelve = (Button) this.findViewById(…
                btVuelve.setOnClickListener( new OnClickListener…
                        public void onClick(…
                        {
                                this.finish();
                        }
                );
        }

        // más cosas…

        private Datos datos;
}
```



6 La aplicación “Lista de la compra”

La aplicación de la lista de la compra sigue un esquema sencillo: permite añadir y borrar elementos en una lista que es siempre visible en la actividad principal. En este momento, cuando es necesario añadir o modificar un artículo a comprar, la interacción con el usuario se realiza a través de un diálogo. Si bien los diálogos son correctos cuando la información a manejar es pequeña, se quedan pronto cortos cuando se maneja una cantidad de información importante.

Es entonces cuando es interesante diseñar el trabajo a realizar en una segunda actividad. En esencia, las subactividades se comportan como diálogos a pantalla completa que detienen la ejecución de la actividad principal.

6.1 Lista de la compra pasando datos en un Intent

En esta aproximación a la lista de la compra, se utilizan los extras para compartir los datos, Así, se utilizan **Activity**.*startActivityForResul()* y **Actitvity**.*onActivityResult()* para lanzar la aplicación pasando los datos, y recoger los datos resultado, respectivamente.

A continuación se muestra el archivo *AndroidManifest.xml* con el resalte especial de la línea dedicada a crear la nueva actividad. En este caso, esta se usa para dos funciones distintas: dar de alta y modificar un item de la compra.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.baltasarq.listacompra2">

    <application
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".View.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".View.ItemEditionActivity"></activity>
    </application>
</manifest>
```

La clase Item se muestra a continuación. Con ella se representan los ítems a comprar especificando un nombre y una cantidad.

```
public class Item {
    private String nombre;
    private int num;

    public Item(String n)
    {
        this.nombre = n;
    }

    public int getNum() {
        return num;
    }

    public void setNum(int num) {
        this.num = num;
    }

    public String getNombre() {
        return nombre;
    }

    @Override
    public String toString()
    {
        return this.getNombre() + ". Num.: " + this.getNum();
    }
}
```

Las partes más relevantes de la actividad principal se muestran a continuación. Desde ella se llama a la subactividad **ItemEditionActivity** con el propósito de añadir o modificar un ítem. Para añadir, se pulsa en el botón btInserta, mientras que para modificar se realiza una pulsación larga en un elemento determinado de la lista.

```
public class MainActivity extends AppCompatActivity {
    protected static final int CODIGO_EDICION_ITEM = 100;
    protected static final int CODIGO_ADICION_ITEM = 102;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // más cosas…
        ListView lvLista = (ListView) this.findViewById( R.id.lvLista );
        Button btInserta = (Button) this.findViewById( R.id.btInserta );

        // Inserta
        btInserta.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent subActividad = new Intent( MainActivity.this, ItemEditionActivity.class );

                subActividad.putExtra( "nombre", "" );
                subActividad.putExtra( "cantidad", 1 );
                MainActivity.this.startActivityForResult( subActividad, CODIGO_ADICION_ITEM );
            }
        });
```

```
        // Modifica
        lvLista.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
                Intent subActividad = new Intent( MainActivity.this, ItemEditionActivity.class );
                Item item = MainActivity.this.adaptadorItems.getItem( i );

                subActividad.putExtra( "nombre", item.getNombre() );
                subActividad.putExtra( "cantidad", item.getNum() );
                subActividad.putExtra( "pos", i );
                MainActivity.this.startActivityForResult( subActividad, CODIGO_EDICION_ITEM );

                return true;
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        if ( requestCode == CODIGO_ADICION_ITEM
          && resultCode == Activity.RESULT_OK )
        {
            Item item = new Item( data.getExtras().getString( "nombre").toString() );
            item.setNum( data.getExtras().getInt( "cantidad" ) );
            this.adaptadorItems.add( item );
            this.updateStatus();
        }

        if ( requestCode == CODIGO_EDICION_ITEM
          && resultCode == Activity.RESULT_OK )
        {
            int pos = data.getExtras().getInt( "pos" );
            Item item = new Item( data.getExtras().getString( "nombre").toString() );

            item.setNum( data.getExtras().getInt( "cantidad" ) );
            this.items.set( pos, item );
            this.adaptadorItems.notifyDataSetChanged();
        }

        return;
    }

    // más cosas…

    private ArrayAdapter<Item> adaptadorItems;
    private ArrayList<Item> items;
}
```

Al lanzar la actividad, se pasan los datos necesarios. En el caso de la adición de nuevos ítems, el nombre que se pasa es la cadena vacía, y el número de artículos, 1. En caso de edición, se pasa el nombre del ítem y el número asociado, para cambiarlos.

En **Activity**.*onActivityResult()* se hace la recepción. En el caso de la adición, el ítem se crea mediante los datos que se obtienen de la subactividad, y se añade a la lista. En caso de edición, se habrán pasado los datos del ítem y su posición en la lista, de manera que solo será necesario cambiar los datos de esa posición en la recepción.

A continuación se muestran las partes más relevantes de la subactividad, en la que se pueden editar los datos de un ítem.

```
public class ItemEditionActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        // más cosas…

        final Intent datosEnviados = this.getIntent();
        final Button btGuardar = (Button) this.findViewById( R.id.btGuardar );
        final Button btCancelar = (Button) this.findViewById( R.id.btCancelar );
        final EditText edCantidad = (EditText) this.findViewById( R.id.edCantidad );
        final EditText edNombre = (EditText) this.findViewById( R.id.edNombre );

        edNombre.setText( datosEnviados.getExtras().getString(( "nombre" ) ) );
        edCantidad.setText( Integer.toString( datosEnviados.getExtras().getInt(( "cantidad" ) ) ) );

        btCancelar.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                ItemEditionActivity.this.setResult( Activity.RESULT_CANCELED );
```

```
                ItemEditionActivity.this.finish();
            }
        });

        btGuardar.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Intent datosRetornar = new Intent();

                datosRetornar.putExtra( "nombre", edNombre.getText().toString() );
                datosRetornar.putExtra( "cantidad",
                       Integer.parseInt( edCantidad.getText().toString() ) );
                datosRetornar.putExtra( "pos",
                       ItemEditionActivity.this.getIntent().getExtras().getInt( "pos" ) );
                ItemEditionActivity.this.setResult( Activity.RESULT_OK, datosRetornar );
                ItemEditionActivity.this.finish();
            }
        });

        // más cosas...
        });
    }
}
```

6.2 Lista de la compra sustituyendo la clase Application

En esta aproximación, se emplea una clase que sustituye a la clase Application, desde la que se comparten los datos en todas las posibles actividades, haciendo innecesario el paso de datos mediante los extras del **Intent**. Como hasta ahora, las partes más relevantes de las clases involucradas se comentan a continuación. No se muestran ni la clase **Item** ni el archivo *AndroidManifest.xml* por ser iguales al caso anterior.

La clase **ListaCompra3App** guarda la lista de ítems en la lista de la compra, y tiene varios métodos auxiliares: *addItem(n, x)* y *modifyItem(i, n, x)*, permitiendo la adición sencilla de nuevos ítems y la modificación de los ya existentes, respectivamente). Estos métodos no son estrictamente necesarios, ya que el método *getItemList()* devuelve la lista completa (debido a la necesidad de crear el adaptador), pero hacen el código más sencillo.

```
public class ListaCompra3App extends Application {
    public void onCreate()
    {
        this.items = new ArrayList<>();
    }

    public List<Item> getItemList() {
        return this.items;
    }

    public void addItem(String nombre, int num) {
        Item item = new Item( nombre );
        item.setNum( num );

        this.items.add( item );
    }

    public void modifyItem(int pos, String nombre, int num) {
        Item item = new Item( nombre );
        item.setNum( num );

        this.items.set( pos, item );
    }

    private List<Item> items;
    private int pos;
}
```

A continuación, se muestra la clase MainActivity, correspondiente a la actividad principal. El código se simplifica, pues al compartir la lista de ítems entre actividades, no es necesario pasar o devolver datos entre actividades mediante los extras de los **Intent**. A excepción, eso sí, de la posición del ítem a modificar. En caso de ser una adición, se pasa -1, de forma que la subactividad pueda distinguir entre una adición y una modificación.

De hecho, **Activity**.*onActivityResult()* tan solo se emplea para actualizar los datos de la lista.

```
public class MainActivity extends AppCompatActivity {
    protected static final int CODIGO_EDICION_ITEM = 100;
    protected static final int CODIGO_ADICION_ITEM = 102;

    protected void onCreate(Bundle savedInstanceState) {
        // más cosas...
        final ListaCompra3App app = (ListaCompra3App) this.getApplication();

        ListView lvLista = (ListView) this.findViewById( R.id.lvLista );
        Button btInserta = (Button) this.findViewById( R.id.btInserta );

        // Lista
        this.adaptadorItems = new ArrayAdapter<Item>(
                this,
                android.R.layout.simple_selectable_list_item,
                app.getItemList() );
        lvLista.setAdapter( this.adaptadorItems );

        // Inserta
        btInserta.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Intent subActividad = new Intent( MainActivity.this, ItemEditionActivity.class );

                subActividad.putExtra( "pos", -1 );
                MainActivity.this.startActivityForResult( subActividad, CODIGO_ADICION_ITEM );
            }
        });

        // Modifica
        lvLista.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
                Intent subActividad = new Intent( MainActivity.this, ItemEditionActivity.class );

                subActividad.putExtra( "pos", i );
                MainActivity.this.startActivityForResult( subActividad, CODIGO_EDICION_ITEM );

                return true;
            }
        });
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        if ( requestCode == CODIGO_ADICION_ITEM
          && resultCode == Activity.RESULT_OK )
        {
            this.adaptadorItems.notifyDataSetChanged();
            this.updateStatus();
        }

        if ( requestCode == CODIGO_EDICION_ITEM
          && resultCode == Activity.RESULT_OK )
        {
            this.adaptadorItems.notifyDataSetChanged();
        }

        return;
    }

    private void updateStatus()
    {
        TextView lblNum = (TextView) this.findViewById( R.id.lblNum );
        lblNum.setText( Integer.toString( this.adaptadorItems.getCount() ) );
    }

    private ArrayAdapter<Item> adaptadorItems;
}
```

La clase **ItemEditionActivity** es la que se encarga directamente de realizar el trabajo de edición o de adición de los datos introducidos por el usuario. Sí se devuelve el código adecuado para indicar si el usuario finalmente confirmó la modificación o adición, pero nada más.

```
public class ItemEditionActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        // más cosas…

        final Button btGuardar = (Button) this.findViewById( R.id.btGuardar );
        final Button btCancelar = (Button) this.findViewById( R.id.btCancelar );
        final EditText edCantidad = (EditText) this.findViewById( R.id.edCantidad );
        final EditText edNombre = (EditText) this.findViewById( R.id.edNombre );
        final ListaCompra3App app = (ListaCompra3App) this.getApplication();

        Intent datosEnviados = this.getIntent();
        final int pos = datosEnviados.getExtras().getInt( "pos" );
        String nombre = "";
        int cantidad = 1;

        if ( pos >= 0 ) {
            nombre = app.getItemList().get( pos ).getNombre();
            cantidad = app.getItemList().get( pos ).getNum();
        }

        edNombre.setText( nombre );
        edCantidad.setText( Integer.toString( cantidad ) );

        btCancelar.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                ItemEditionActivity.this.setResult( Activity.RESULT_CANCELED );
                ItemEditionActivity.this.finish();
            }
        });

        btGuardar.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                final String nombre = edNombre.getText().toString();
                final int cantidad = Integer.parseInt( edCantidad.getText().toString() );

                if ( pos >= 0 ) {
                    app.modifyItem( pos, nombre, cantidad );
                } else {
                    app.addItem( nombre, cantidad );
                }

                ItemEditionActivity.this.setResult( Activity.RESULT_OK );
                ItemEditionActivity.this.finish();
            }
        });

       // más cosas...
    }
}
```

7 Referencias

- Documentación y recursos de Android para desarrolladores (accedido en sept. 2016) http://developer.android.com/
- Actividades https://developer.android.com/guide/components/activities.html
- Ciclo de vida de una actividad https://developer.android.com/training/basics/activity-lifecycle/
- App “Lista de la compra” con multiactividades. https://github.com/Baltasarq/ListaCompra2
- App “Lista de la compra” sustituyendo la clase **Application**. https://github.com/Baltasarq/ListaCompra3