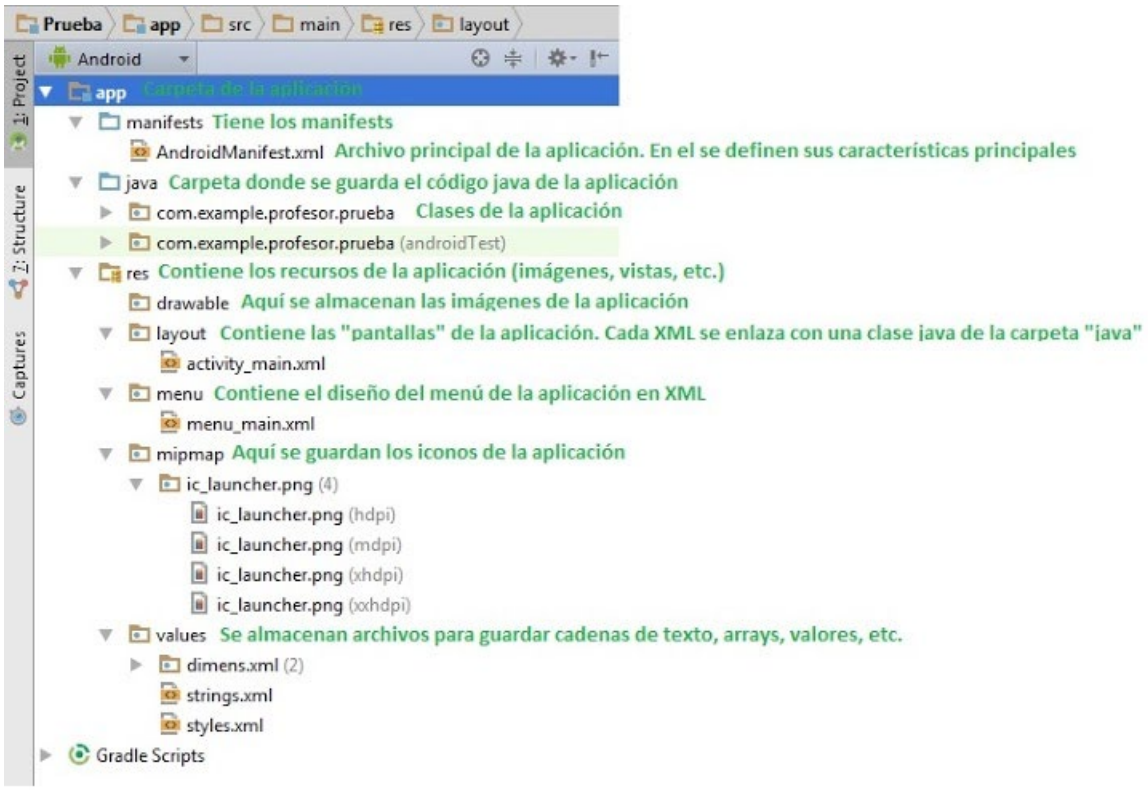
UNIDAD 2: Programación de aplicaciones para dispositivos móviles

El material adicional de esta unidad, está pensado para orientar y reforzar los conceptos que tenéis que poner en práctica en la tarea de la unidad.

ÍNDICE

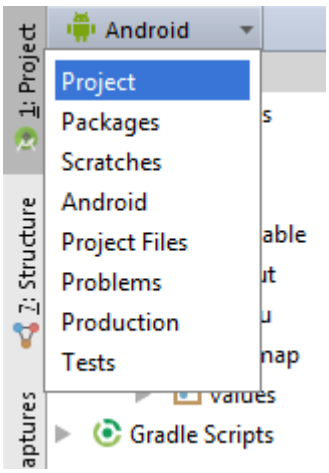
1 Estructura de un proyecto Android

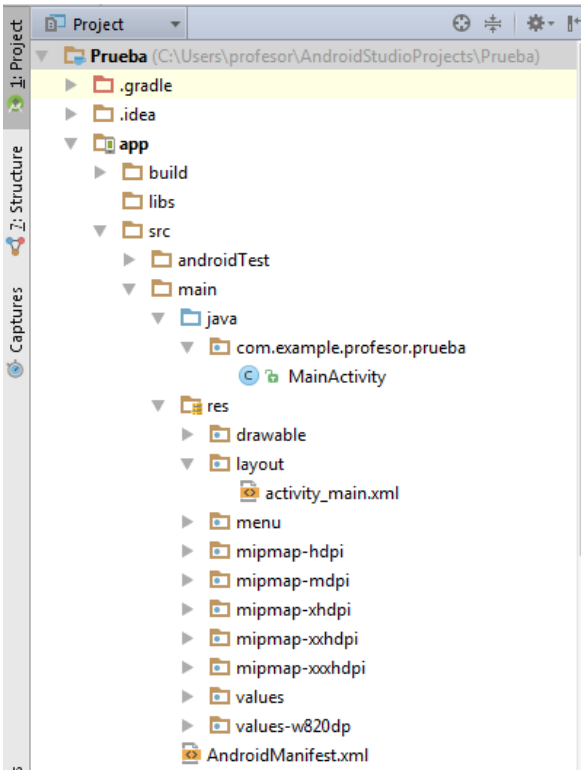
Esta es la estructura de un proyecto Android:



Donde más trabajaremos será sobre la carpeta “java”, donde está el código fuente y sobre “res/layout”.

Esta estructura no se corresponde con la estructura en disco del proyecto. Podemos cambiar de vista haciendo clic en el proyecto donde pone “Android” y seleccionando “Project”. Ahora sí vemos la estructura de carpetas física del proyecto.





2 ConstraintLayout

ConstraintLayout permite crear diseños de forma rápida, fácil y flexible. Está disponible a partir de la API 9. Permite posicionar los elementos de la interfaz sin tener que usar layouts anidados.

Consulta la documentación oficial de Android Studio para saber cómo trabajar con ellos: https://developer.android.com/training/constraint-layout?hl=es-419.

3 Activity

3.1 Consideraciones Generales

Las activities son el núcleo de la programación android. Una activity representa la actividad que se desarrolla en una pantalla. **En android sólo una activity puede estar activa en cada momento**.

Toda actividad dereda de la clase **AppCompatActivity** y deben implementar al menos el método **onCreate** y llamar al constructor de la superclase.

El código de una activity es el siguiente:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); //Carga los componentes definidos en el layout res/activity_main.xml
    }
}
```

Se pueden definir los componentes de la interfaz gráfica desde código Java o cargar un recurso layout ya definido. En la siguiente imagen se muestra el código para cada caso. En las tareas que vamos a realizar, vamos a trabajar con layouts definidos como recursos.

Codigo Java	Recurso xml

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
LinearLayout ll = new LinearLayout(this);
ll.setOrientation(LinearLayout.*VERTICAL*);
TextView tvMensaje = new TextView(this);
tvMensaje.setText("Hola");
ll.addView(tvMensaje);
setContentView(ll);
}
	@Override
	protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.*activity_main*);
	}

3.2 Regristro de un activity en AndroidManifest

Todas las actividades de una aplicación deben estar registradas en el fichero AndroidManifest.xml. Si creamos las actividades desde el asistente, se registrará automáticamente, pero si las creamos como una clase, hay que registrarlas para que la aplicación funcione. En la siguiente imagen, se muestra el AndroidManifest de un proyecto con dos actividades. La segunda actividad está marcada en amarillo.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.importar2021">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Importar2021"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".Actividad2"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

3.3 Acceso a los recursos

En el desarrollo de una aplicación resulta casi imprescindible el acceso a los recursos definidos en archivos xml (en la carpeta res) desde código java.

Hay que tener en cuenta que nuestro IDE genera automáticamente una clase R.java dentro del directorio gen. **El acceso a estos recursos se realiza a través del nombre generado, nunca directamente**.

3.3.1 Acceso a los componentes de un Layout

Los layouts se almacen en ficheros dentro de la carpeta **res/layout**. Para acceder desde una actividad a los elementos definidos en un layout debemos dar estos pasos:

1. Cargar el layout de la actividad con el método **setContentView**.
2. Declaramos objetos de la clase a enlazar.
3. Enlazamos con el recurso utilizando el método **findViewById**: El código generado en R.java para los componentes tiene el siguiente formato: **R.id.IddelComponente**.

Una vez hecho esto ya podemos trabajar con el componente desde código.

La siguiente imagen muestra un ejemplo de un un layout que contiene un elemento de tipo EditText cuyo id es etNombre:

```
/res/layout/activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ... >
    <EditText
          android:id="@+id/etNombre"
          .../>
    ...
</LinearLayout>
```

La siguiente imagen muestra cómo enlazamos un objeto definido en la actividad con el elemento definido en el layout para asignarle como texto *Pepito Perez Lopez*.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    EditText etNombre = findViewById(R.id.etNombre);
    etNombre.setText("Pepito Perez Lopez");
}
```

Debemos tener cuidad con el ámbito de definición de los objetos. En este ejemplo, el objeto etNombre está declarado dentro del método onCreate, es por ello que solamente tendremos acceso dentro de este método. Podríamos haberlo declarado a nivel global, como atributo de la clase.

3.3.2 Acceso a los recursos string

Los string se almacenan en fichero **res/values/string.xml**. Para acceder a ellos debemos hacer referencia al componente definido en R.java. El código generado en R.java para los string tiene el siguiente formato: ***R.string.IDdelString***.

Definición del string en el fichero string.xml.

```
<string name="texto">Datos del empleado</string>
```

Acceso al string desde java:

```
String texto= getString(R.string.texto);
```

3.3.3 Acceso a los recursos array de strings

Se realiza con el siguiente formato: ***R.array.IddelArray***.

Definición del array:

```
<resources>
    <string-array name="nombres">
        <item>María</item>
        <item>Javier</item>
        <item>Pablo</item>
    </string-array>
</resources>
```

```
String[] nombres= getResources().getStringArray(R.array.nombres);
```

Acceso al array desde Java:

3.3.4 Acceso a los recursos array de enteros

Se realiza con el siguiente formato: R.array.IddelArray.

Definición del array de enteros:

```
<integer-array name="numeros">
    <item>1</item>
    <item>2</item>
    <item>3</item>
</integer-array>
```

Acceso al array desde Java:

```
int[] numeros = getResources().getIntArray(R.array.numeros);
```

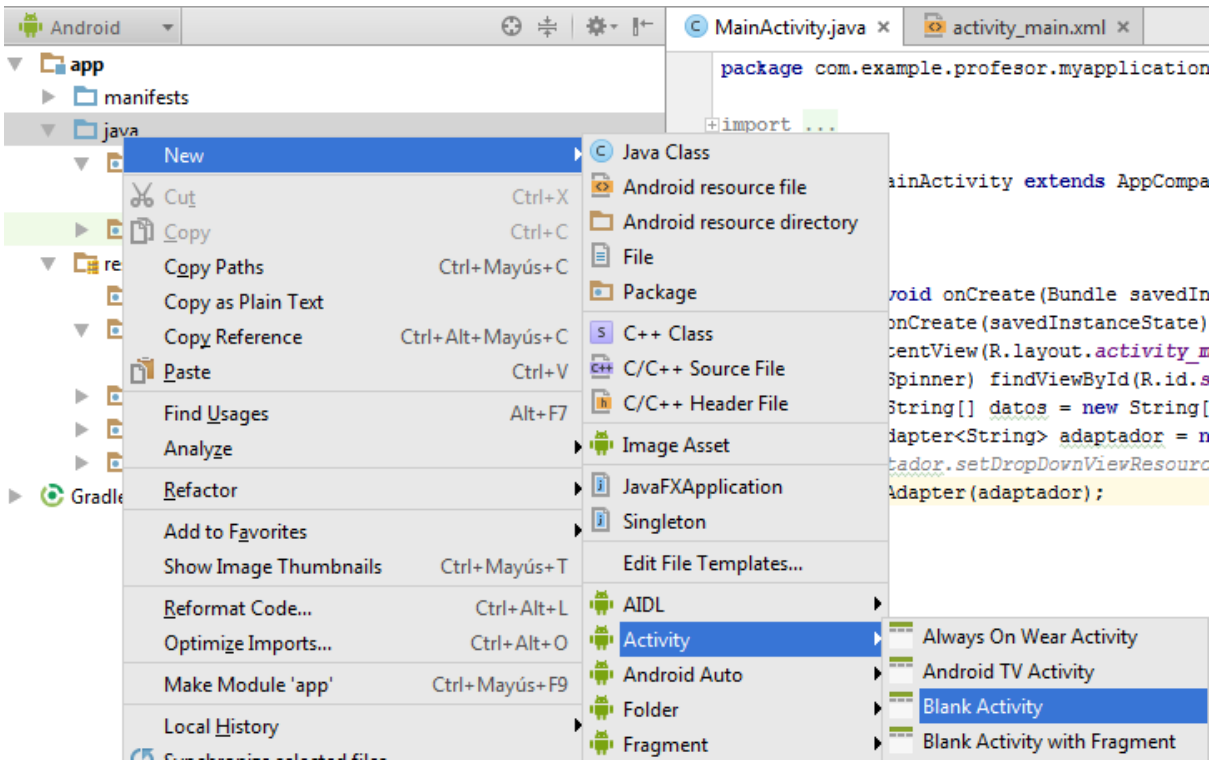
3.3.5 Acceso a los recursos drawable

Los archivos de imágenes se ubican en la carpeta **res/drawable**. Se accede a ellos con ***R.drawable.IddelDrawable*** (el id va a ser el nombre del archivo) .

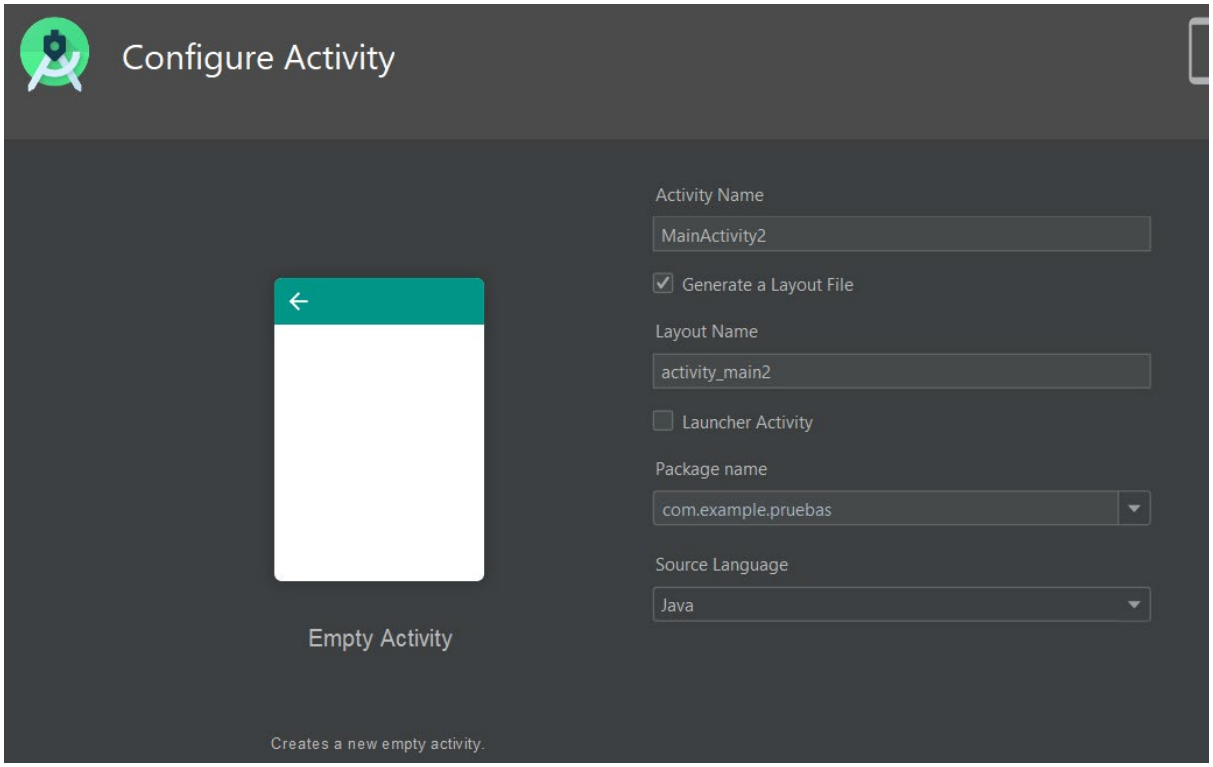
Drawable logo =getResources().getDrawable(R.drawable.logo);

3.4 Añadir nueva Activity al proyecto

Para crear una nueva Activity en un proyecto pulsamos botón derecho del ratón sobre la carpeta java y Seleccionamos New/Activity/Empty Activity.



La siguiente ventana permite especificar las características: nombre, título, nombre del layout...



La nueva activity aparece reflejada en el Manifest:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Pruebas"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity2"></activity>
    <activity android:name=".MainActivity">
        <intent-filter>
```

La activity que se lanza al inicio de la aplicación es la que tiene el código XML marcado en rojo:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Pruebas"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity2"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

3.5 Eventos

3.5.1 Control del click

Siendo los dispositivos móviles Android mayoritariamente táctiles y carentes de teclado físico, es especialmente importante el control de las pulsaciones sobre la pantalla.

Android soporta métodos generales de tratamiento de estas pulsaciones, pero en esta ocasión nos centraremos en un método específico, la pulsación sobre un objeto Button.

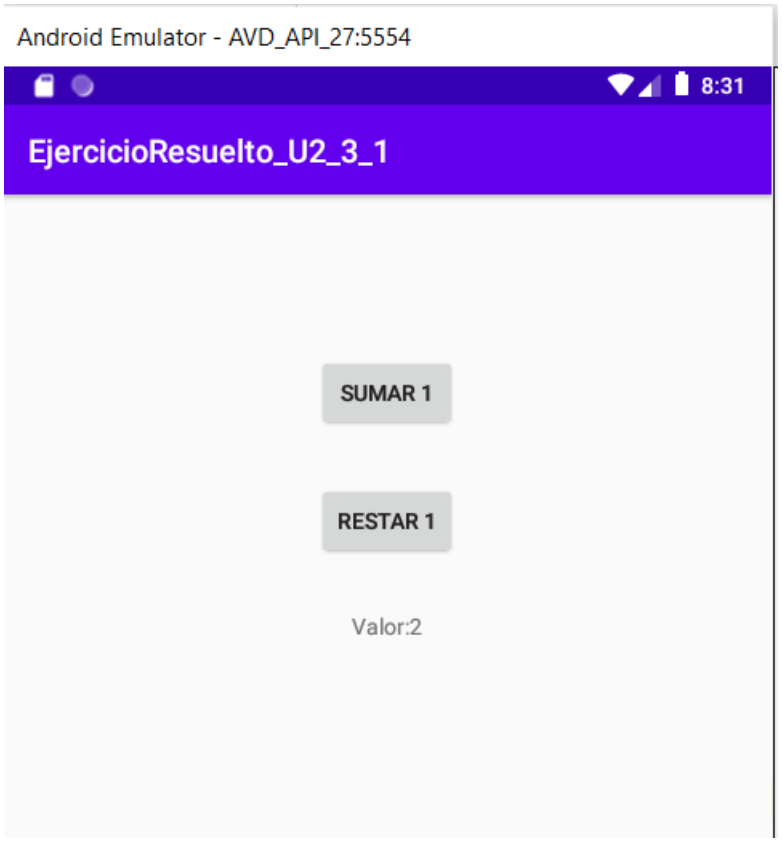
El control del click sobre un objeto Button lo podemos realizar de tres maneras diferentes:

- Definiendo un atributo **android:click** al definir el componente Button.
- Implementando la interfaz **OnClickListener**.
- Implementando el método **setOnClickListener** de este button en cuestión.

```
boton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        ...
    }
});
```

3.6 Ejercicio Resuelto U2_3_1

Realiza una pequeña App que permita sumar y restar 1 a un valor que inicialmente es 0:



Solución

4 Contexto de una Aplicación

Una definición simple del contexto sería es el “lugar” donde se encuentra la aplicación en un momento determinado de la ejecución de la misma.

A la hora de crear algunos objetos, su constructor nos pide el contexto:

- Toast
- Llamadas a otras actividades (intents)
- Otros...

El ejemplo más básico es en la generación de un mensaje emergente o **Toast**:

```
Toast.makeText(contexto,"Mensaje 1",Toast.LENGTH_SHORT).show();
```

La clase Toast va a utilizar “contexto” para saber dónde mostrar el mensaje. Ese “lugar” debe ser la activity actual que se está ejecutando en ese momento. Lo que nos lleva al problema de cómo obtener el contexto.

Hay diferentes forma de obtener el contexto, según dónde nos encontremos:

- this

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Toast.makeText( context: this, text: "Mensaje 1",Toast.LENGTH_SHORT).show();
```

- getApplicationContext(): Lo usamos cuando no podemos usar this, por ejemplo, dentro de un evento de un botón.

```
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText(getApplicationContext(), text: "Mensaje 2",Toast.LENGTH_SHORT).show();
    }
});
```

- NombreClase.this: Con este método nos vamos a evitar cualquier problema, ya que estamos indicando explícitamente el contexto que queremos utilizar.

```
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText( context: MainActivity.this, text: "Mensaje 2",Toast.LENGTH_SHORT).show();
    }
});
```

- Utilizar una constante: Almacenamos el contexto al inicio de la Activity en una constante de tipo “Context” Método muy útil si vamos a trabajar mucho con el contexto.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final Context contexto=this;

    Toast.makeText(contexto, text: "Mensaje 1",Toast.LENGTH_SHORT).show();

    boton=findViewById(R.id.button);
    boton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast.makeText(contexto, text: "Mensaje 2",Toast.LENGTH_SHORT).show();
        }
    });
```

5 Depuración de aplicaciones

5.1 Localizar la línea en la que se ha producido el error

https://www.youtube.com/watch?v=zUiJ4Qig_Ic

5.2 Depurar en Android Studio

https://www.youtube.com/watch?v=9oKuxBXs1SQ&feature=youtu.be

5.3 LogCat para depurar

https://www.youtube.com/watch?v=VjzAufVT_CM&feature=youtu.be

5.4 Ejercicio U2_5_1

- Abre el ejercicio U2_3_1.
- Pon un breakpoint al inicio del onCreate. Ejecuta paso por paso (tecla F8) todas las instrucciones.
- Quita el breakpoint anterior y pon otro dentro del evento de click del botón. Observa el valor que va tomando la variable “contador”.

5.5 Ejercicio U2_5_2

Modifica el ejercicio U2_5_1 y muestra en el Logcat los valores de “contador” cada vez que se produzca un cambio.

5.6 Ejercicio U2_5_3

Comenta la línea de enlazar el botón suma con el id:

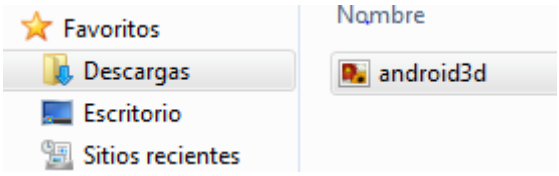
```
//suma = findViewById(R.id.botonSuma);
```

Al ejecutar la aplicación fuerza cierre. Pon un breakpoint en el onCreate y analiza la línea en la que se produce el fallo.

6 Imágenes en Android Studio

6.1 Añadir imágenes a un proyecto

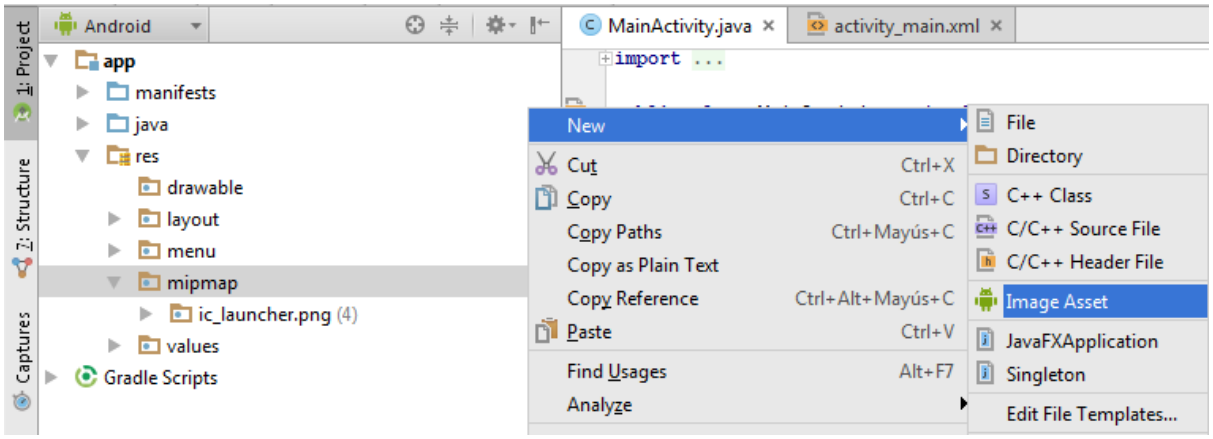
Lo primero que tenemos que hacer es localizar el archivo de imagen. Os recomiendo que los nombres de los archivos no contengan letras mayúsculas, ni tildes, ni caracteres especiales para evitar problemas.



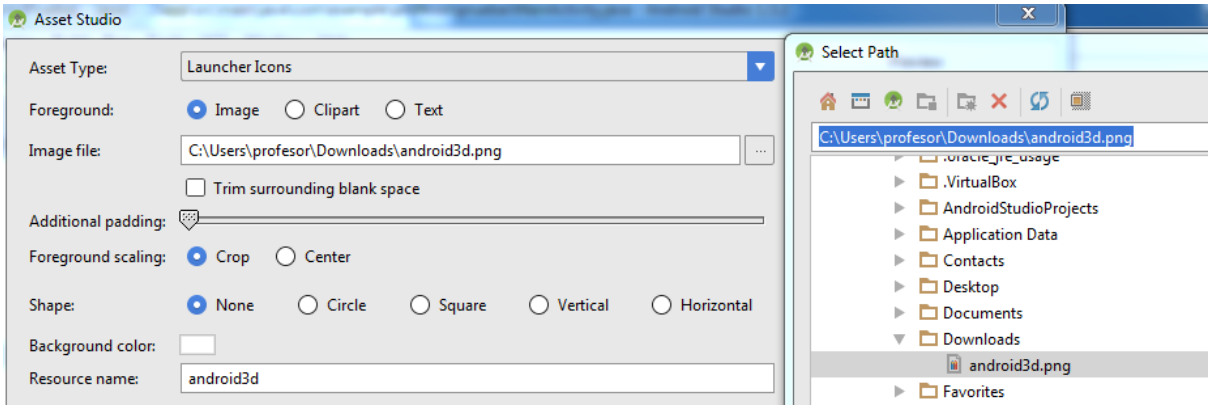
Hay varias formas de añadir imágenes a un proyecto, lo podemos hacer a la carpeta mipmap o a la carpeta drawable.

6.1.1 Añadir a la carpeta mipmap

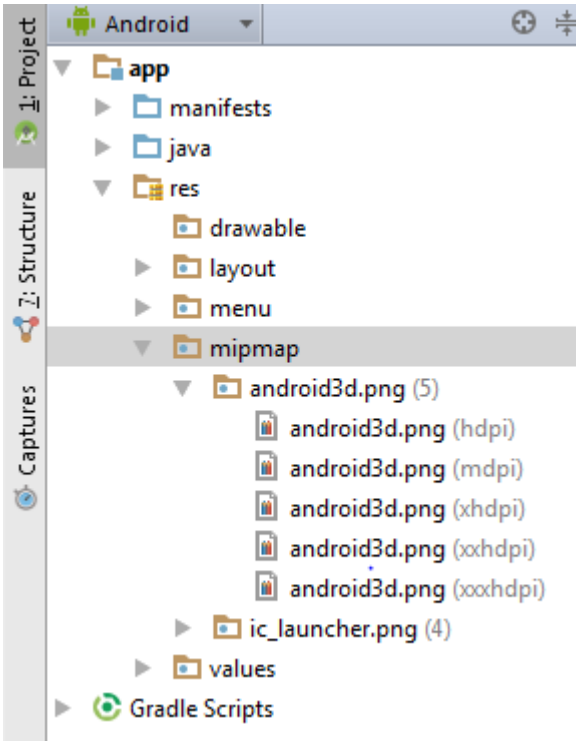
1. Dentro de Android Studio, pulsamos botón derecho sobre mipmap y seleccionamos New/Image Asset.



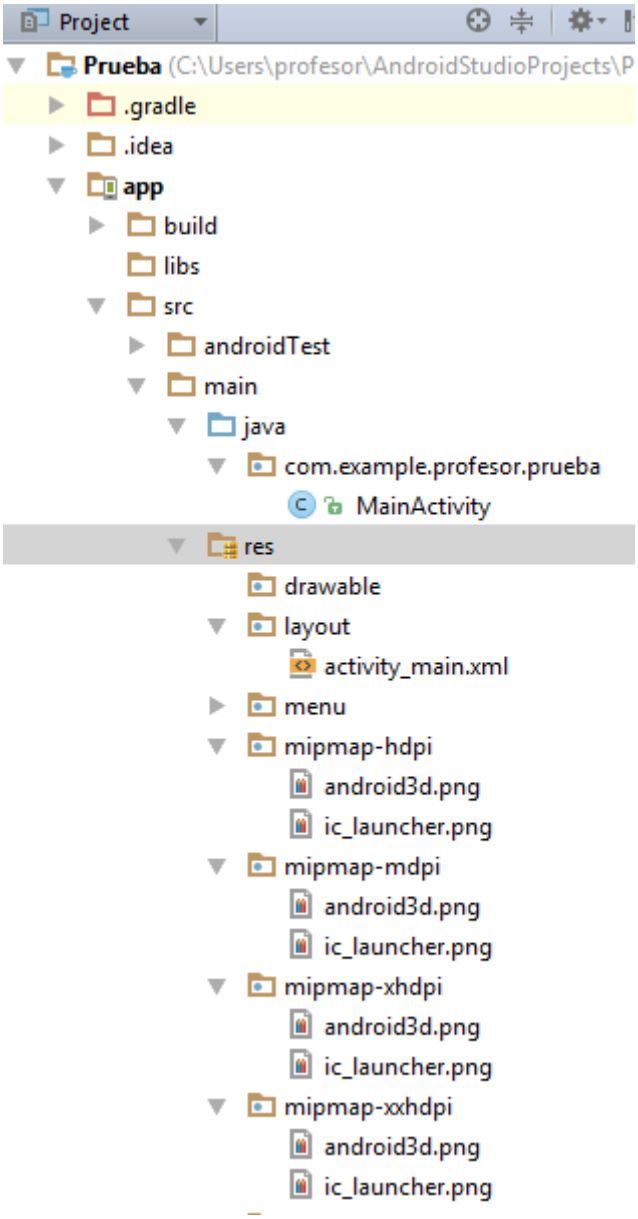
2. En “Image file:” buscamos la ruta de la imagen. En “Resource name” le ponemos un nombre descriptivo a esa imagen.



El resultado es que en nuestro proyecto tenemos la imagen cargada. Nos ha creado una imagen con una definición específica dependiendo de la calidad de la pantalla del teléfono. Nosotros no nos tenemos que preocupar por eso, solo utilizaremos la imagen y Andoid elige cuál usar.



Si cambiamos a la vista de proyecto, vemos la estructura real de las carpetas. Nos ha creado un archivo en cada carpeta mipmap, según la calidad de la imagen.

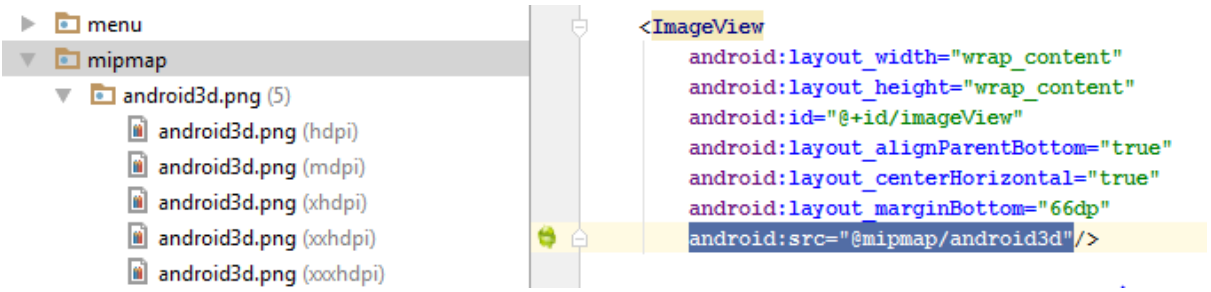


6.1.2 Añadir a la carpeta drawable

Otra forma de añadir más rápido imágenes al proyecto consiste en arrastrarlas directamente a la carpeta “drawable” (se puede usar ctrl+c del fichero de la imagen y ctrl+v sobre la carpeta drawable).

6.2 Uso de imágenes

El objeto que nos va a permitir cargar una imagen dentro de un Layout es un **ImageView**. Tenemos que añadir la propiedad ***android:src*** para indicar el origen de la imagen. Su valor será ***@nombre_carpeta/nombre_de_la_imagen***.



Si queremos cambiar la imagen desde la clase Java, utilizamos el método **setImageResource** del objeto **ImageView**. El método recibe como parámetro la imagen: ***R.mipmap.nombre_imagen*** o ***R.drawable.nombre_imagen***, dependiendo de la carpeta en que se ubique.

```
public class MainActivity extends AppCompatActivity {
ImageView imagen;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imagen= findViewById(R.id.imageView);
        imagen.setImageResource(R.mipmap.android3d);
```

7 Eventos de Checkbox y RadioButtons

Los objetos de tipo Checkbox y Radiobuttons permiten al usuario seleccionar opciones de un conjunto. La diferencia está en que los checkbox permiten una selección múltiple y los radiobuttons no. Los radiobuttons siempre se definen dentro de un radiogroup de forma que, la selección de un radiobutton dentro de un mismo radiogroup es mutuamente exclusiva.

Cuando el usuario hace click sobre un **CheckBox** o un **RadioGroup** se lanza el evento **OnClick**. Pero puede que el CheckBox cambie de estado por otros motivos que no dependan de la interacción del usuario con la aplicación (algo que no sea el usuario utiliza el método **setChecked** del checkbox)**,** por ejemplo:

- Otro CheckBox cambie su estado
- Un botón cambie su estado
- Otro evento cambie su estado

Por tanto, para estos dos tipos de objetos necesitamos capturar el evento **onCheckedChange** si queremos programar un cambio en su estado. El evento **onCheckedChange** se activa cuando el **CheckBox** o de alguno de los **RadioButtons** de un **RadioGroup** cambia de estado, no cuando el usuario hace click sobre ellos.

Cuando el usuario pulsa sobre el checkbox se lanzan dos eventos:

- Onclick
- Oncheckedchange

7.1 EjercicioResuleto_U2_7_1

Realiza la siguiente aplicación:

Siempre que se marque uno de los elementos de la cesta de la compra, deberá mostrar un texto informando de ello. Lo mismo cuando se seleccione una opción de pago.

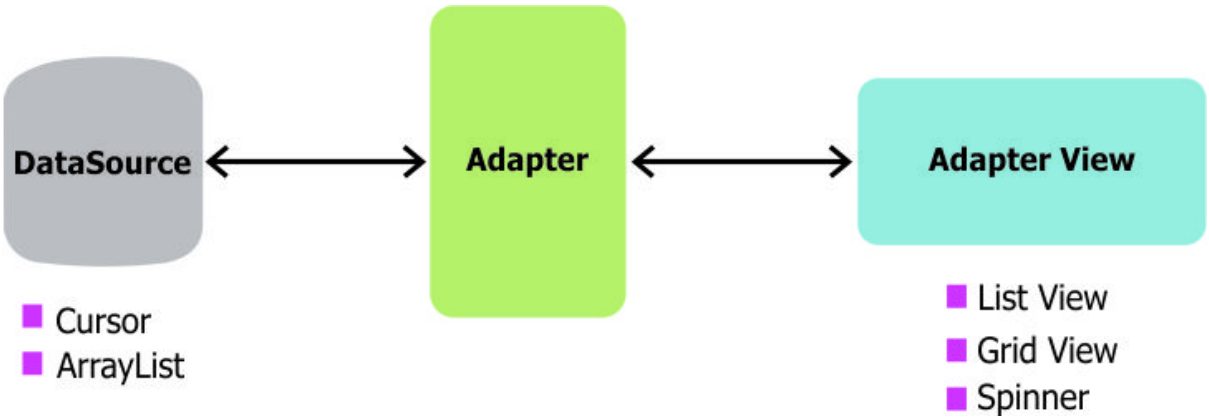
Los mensajes también deben mostrarse cuando la selección de los items se haga a a través de los botones *Marcar Todo* y *Desmarcar Todo*.

Solución

8 Spinners y Lista. Apdaptadores

Tanto los spinners como las listas son controles que permiten realizar una selección de los elementos que contienen. Este tipo de controles acceden a los datos que contienen a través de un adaptador. El adaptador se encarga de:

- Proveer de datos a los controles visuales
- Especificar cómo se mostrarán los datos dentro del control de selección.



Existen varios tipos de adaptadores: ArrayAdapter, SimpleAdpater, SimpleCursorAdapter. Nosotros vamos a trabajar con **ArrayAdapter**.

8.1 ArrayAdapter.

Es el más sencillo de todos los adaptadores, y provee de datos a un control de selección a partir de un array de objetos.

Creación de ArrayAdapter con origen en un array estático definido en Java

En este ejemplo el array está definido como un array estático en el código, pero podría haberse definido en un fichero de recursos. También se podrían usar Arrays dinámicos como los ArraysList.

```
//1. Definición del array que contiene los datos
final String[] datos=new String[]{"Elem1","Elem2","Elem3","Elem4","Elem5"};

//Creamos el adaptador. Le pasamos tres parámetros:
//El contexto. Es una referencia a la activity donde se crea el adaptador
//ID del layout. Especifica cómo se muestra el spinner cuando no está desplegado
//El array con los datos
ArrayAdapter<String> adaptador=
        new ArrayAdapter<String>(getApplicationContext(),
                android.R.layout.simple_spinner_item, datos);

//2. Indica cómo se muestra cada elemento del spinner cuando está desplegado
adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

//3. Aplica el adaptador al spinner para que se muestren los datos
cmbOpciones.setAdapter(adaptador);
```

Creación de ArrayAdapter con origen en un array estático definido en string.xml

Fichero string.xml

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

<resources>
   <string-array name="valores_array">
      <item>Mercury</item>
      <item>Venus</item>
      <item>Earth</item>
      <item>Mars</item>
      <item>Jupiter</item>
      <item>Saturn</item>
      <item>Uranus</item>
      <item>Neptune</item>
   </string-array>
</resources>
```

```
//Creación de un ArrayAdapter a través de recursos en el fichero strings.xml
ArrayAdapter<CharSequence> adaptador=
        ArrayAdapter.createFromResource(getApplicationContext(),
                R.array.valores_array,android.R.layout.simple_spinner_item);
```

8.2 Responder a las selecciones

Cuando se produce una selección de un elemento de una lista o de un spinner, se genera un evento que debemos capturar para programar la respuesta que queremos darle al usuario. Dicho evento es diferente si estamos trabajando con una lista o con un spinner.

- Spinner: el evento que debemos capturar es ***setOnItemSelectedListener***.
- Lista: el evento que debemos capturar es ***setOnItemClickListener***.

```
//Para capturar las selecciones hechas por el usuario, implementamos el evento onItemSelected
cmbOpciones.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    //"parent" es el elemento(AdapterView) donde el usuario ha pinchado (el spinner)
    //"view" es la vista dentro del AdapterView donde el usuario ha pinchado (fila del spinner)
    //"position" nos da el índice del elemento seleccionado en el adaptador empezando desde 0
    //"id" indica la fila que el usuario ha seleccionado en el spinner. Empieza en 0
   public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        //Mostramos el valor del elemento seleccionado
        Toast.makeText(getApplicationContext(),"Has seleccionado: "+datos[position],Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {

    }
});
```

8.3 EjercicioResuelto_U2_8_1

- Añade un spinner vacío al proyecto. Ejecuta la app en el emulador.
- Crea un spinner con los días de la semana. Utiliza como origen un Array definido en el fichero string.xml.
- Crea un spinner con los meses del año. Utiliza como origen de datos un ArrayList.
- Cambia el layout del spinner d *setDropDownViewResource* por simple_List_item_multiple_choice.
- Cada vez que se seleccione un día de la semana o un mes, muestra un mensaje con la opción seleccionada.

Solución

8.4 EjercicioResuelto_U2_8_2

- Añade un listView vacío al proyecto.
- El origen de datos del listview será un array definido en un fichero de recursos con los nombres de los planetas del sistmea solar.
- Cada vez que se seleccione un planeta, muestra un mensaje con la opción seleccionada.
- Modifica el atributo ***choiceMode*** del listview para que admita una selección múltiple.
- Añade un botón que al pulsarlo muestre un mensaje con las opciones seleccionadas en el listView.

Solución

8.5 EjercicioResuelto_U2_8_3

Modifica el ejercicio anterior para que al hacer una pulsación larga sobre un elemento de la lista, se borre dicho elemento y se actualice tanto el adaptador como la lista.

Debes cambiar el origen de datos a un ArrayList, en lugar del array estático definido en el fichero de recursos.

Solución.

8.6 Profundización

8.6.1 Layout personalizado

Para el ejercicio anterior, crea un layout que contenga un texto de color rojo y tamaño 30sp, alineado a la derecha. Utiliza ese layout para mostrarlo en un spinner (o lista) a través del adaptador.

Pista: Añadir un parámetro más al ArrayAdapter.

8.6.2 Lista con dos líneas

Investiga cómo trabajar con una lista con dos líneas utilizando el layout android.R.layout.simple_list_item_2.

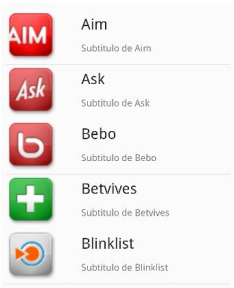
Pista: hay que utilizar un SimpleCursorAdapter.

8.6.3 Lista con Scroll

Investiga cómo se puede hacer una Lista "Scrollable".

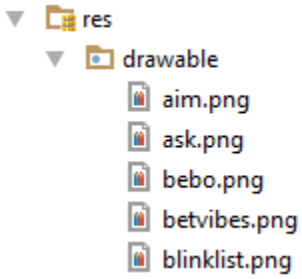
8.6.4 ListView o Spinner Personalizados

Es posible hacer listviiew/Spinner con diseños personalizados, tal y como se muestra en la imagen de la derecha.



Los pasos que hay que dar para ello son:

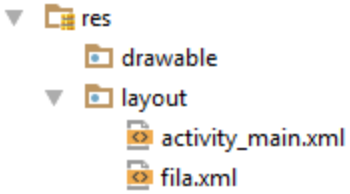
1. Importar las imágenes al proyecto



2. Crear también los arrays con los títulos y subtítulos en strings.xml

```
<string-array name="titulos">
    <item>Aim</item>
    <item>Ask</item>
    <item>Bebo</item>
    <item>Betvives</item>
    <item>Blinklist</item>
</string-array>
<string-array name="subtitulos">
    <item>Subtitulo de Aim</item>
    <item>Subtitulo de Ask</item>
    <item>Subtitulo de Bebo</item>
    <item>Subtitulo de Betvives</item>
    <item>Subtitulo de Blinklist</item>
</string-array>
```

3. Crear un Layout que tendrá el aspecto que queramos para cada fila del ListView o Spinner.



4. Crear una clase Adaptadora que extienda de ***BaseAdapter.***
    a. Crear primero una clase contenedora de objetos que almacena las imágenes y los dos textos de cada celda de la lista.

```
//Clase contenedora. Almacenamos un título, un subtítulo y una imagen
    class filas{
        String titulo;
        String subtitulo;
        int imagen;

        filas(String titulo, String subtitulo, int imagen) {
            this.titulo=titulo;
            this.subtitulo=subtitulo;
            this.imagen=imagen;
        }

    }
```

    b. Crea la clase adaptadora.

```
class MyCustomAdapter extends BaseAdapter
{


    public int getCount() {}


    public Object getItem(int arg0) {}


    public long getItemId(int position) {}


    public View getView(int position, View convertView, ViewGroup parent) {}
```

    c. El constructor se encarga de crear un ArrayList de objetos de la clase contenedora (de cada fila de la lista).

```
//Clase adaptadora. Extiende de BaseAdapter
    class adaptador extends BaseAdapter{


        ArrayList<filas> lista;
        //Almacenamos el contexto de la aplicación
        Context contexto;

        //Constructor
        adaptador(Context c){
            //Recogemos el contexto que nos pasan como parámetro desde la llamada
            contexto=c;
            //Definimos un arraylist donde almacenamos la terna de elementos (objetos de la clase contenedora)
            lista=new ArrayList<filas>();

            //Definimos un objeto Resources para poder acceder a los arrays de strings y a los drawables
            Resources res=c.getResources();
            String[] titulos=res.getStringArray(R.array.Titulos);
            String[] subtitulos=res.getStringArray(R.array.Subtitulos);
            int[] imagenes={R.drawable.a1,R.drawable.a2,R.drawable.a3,R.drawable.a4,R.drawable.a5};

            //Creamos un araylist de objetos de la clase contenedora a partir de los arrays
            for (int i=0; i<5; i++){
                lista.add(new filas(titulos[i],subtitulos[i],imagenes[i]));
            }
        }
```

d. Implementa los métodos getCount, getItem y getItemId.

```
@Override
//Método que devuelve el número de elementos
public int getCount() {
    // TODO Auto-generated method stub
    return lista.size();
}

@Override
//Devuelve el item indicado por la posición
public Object getItem(int position) {
    // TODO Auto-generated method stub
    return lista.get(position);
}

@Override
//Devuelve la posición
public long getItemId(int position) {
    // TODO Auto-generated method stub
    return position;
}
```

e. Implementa el método getView es el que se encarga de mostrar cada fila.

```
//Método getview
public View getView(int position, View convertView, ViewGroup parent) {
    // TODO Auto-generated method stub

    //Definimos un LayoutInflater para poder cargar el custom layout de la fila
    LayoutInflater inflar=(LayoutInflater) contexto.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    //Inflamos el custom layout
    View list=inflar.inflate(R.layout.filas, parent, false);
    //Enlazamos a través de la clase R los elementos del custom layout
    TextView titulo=(TextView) list.findViewById(R.id.textView1);
    TextView subtitulo=(TextView) list.findViewById(R.id.textView2);
    ImageView imagen= (ImageView) list.findViewById(R.id.imageView1);

    //Mostrar el elemento definido en position
    //En un objeto de la clase contenedora recogemos el elemento n
    filas temporal=lista.get(position);
    //Y asignamos su contenido a los elementos del custom layout
    titulo.setText(temporal.titulo);
    subtitulo.setText(temporal.subtitulo);
    imagen.setImageResource(temporal.imagen);

    return list;

}
```

5. Usa el adaptador.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    lw1=(ListView) findViewById(R.id.listv1);
    lw1.setAdapter(new adaptador(this));

}
```

6. Responder a las selecciones del usuario.

```
//Responder a las selecciones del usuario.Evento onItemClick
lw1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        //getItemAtPosition nos devuelve el objeto de la clase filas de la posición seleccionada
        filas fil = (filas) lw1.getItemAtPosition(position);
        //Con fil podemos sacar el título y el subtítulo
        Toast.makeText(getApplicationContext(),
        "Título: "+ fil.titulo + "Subtítulo: "+ fil.subtitulo,
        Toast.LENGTH_SHORT).show();

    }
});
```

9 Intents

Un Intent es un objeto que facilita la comunicación entre Activities y permite que se puedan iniciar otras Activities. El Intent describe la actividad que se debe iniciar y contiene los datos necesarios para ello.

Existen dos tipos de intents:

- **Explícitas**: especifican qué aplicación las administrará, ya sea incluyendo el nombre del paquete de la app de destino o el nombre de clase del componente completamente calificado. Normalmente, el usuario usa una intent explícita para iniciar un componente en su propia aplicación porque conoce el nombre de clase de la actividad o el servicio que desea iniciar. Por ejemplo, puedes utilizarla para iniciar una actividad nueva en respuesta a una acción del usuario o iniciar un servicio para descargar un archivo en segundo plano. Ejemplo:

```
Intent miIntent = new Intent(this, Actividad2.class);
startActivity(miIntent);
```

- **Implícitas**: Los intents implícitos permiten abrir una activity externa (de otra aplicación).

9.1 Intent Implícitas

La llamada a estas activities suelen requerir el uso de permisos en el archivo Android Manifest. El comportamiento del sistema después de declarar un permiso depende de qué tan sensible sea este. Algunos se consideran "normales", por lo que el sistema los otorga inmediatamente después de la instalación. Otros se consideran "peligrosos", por lo que el usuario debe otorgar el acceso a tu app de manera explícita.

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

La siguiente imagen muestra el permiso para enviar un SMS.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mkyong.android"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

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

9.1.1 Abrir Página Web en Navegador

```
Intent intento = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.es"));
startActivity(intento);
```

9.1.2 Realizar una búsqueda web

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
intent.putExtra(SearchManager.QUERY,  value: "IES Augustóbriga");
startActivity(intent);
```

9.1.3 Mostrar Contactos

```
Intent intento = new Intent(Intent.ACTION_VIEW, Uri.parse("content://contacts/people/"));
startActivity(intento);
```

9.1.4 Mostrar el teléfono con un número para llamar

Ejemplo1:

```
Intent intento = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:123456789"));
startActivity(intento);
```

Ejemplo 2:

```
Intent intento = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:123456789"));
startActivity(intento);
```

9.1.5 Hacer llamada automáticamente

```
Intent intento = new Intent(Intent.ACTION_CALL, Uri.parse("tel:123456789"));
startActivity(intento);
```

Necesita solicitar permiso en Manifest. Está considereado dentro del nimvel y en la aplicación de manera explícita (Ajustes/Aplicaciones/Permisos).

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

9.1.6 Mostrar un mapa

Ejemplo 1:

```
Intent intento = new Intent(Intent.ACTION_VIEW,
        Uri.parse("geo:39.890467,-5.532791"));
startActivity(intento);
```

Ejemplo 2:

```
String latitud = "0";
String longitud = "0";
String uri = "geo:"+ latitud + "," + longitud +
        "?q=71, Antonio Concha, Navalmoral de la Mata";
Intent intento = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
startActivity(intento);
```

9.1.7 Iniciar Cámara

```
Intent intento = new Intent( action: "android.media.action.IMAGE_CAPTURE");
startActivity(intento);
```

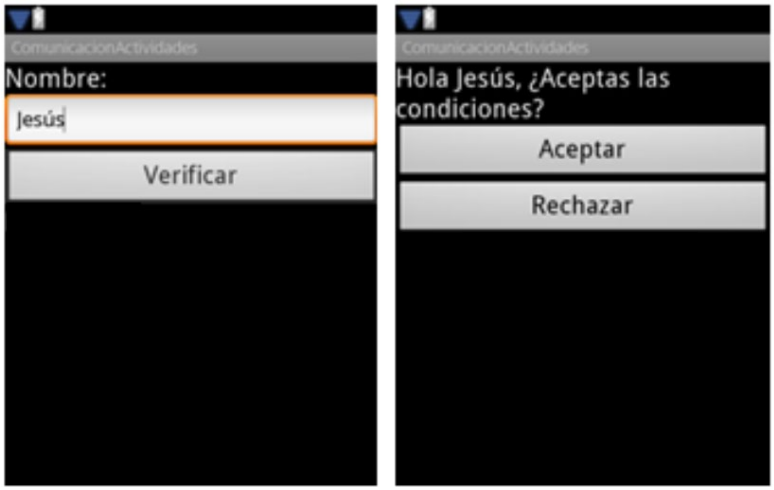
9.1.8 Enviar SMS

```
Intent intento = new Intent(Intent.ACTION_SENDTO, Uri.parse("smsto:1234424"));
intento.putExtra( name: "sms_body", value: "Texto a enviar");
startActivity(intento);
```

9.2 Comunicación entre Activities

Los Intents establecen un mecanismo de comunicación entre actividades. Este mecanismo está basado en intents.

Un ejemplo sencillo de comunicación entre dos actividades es el siguiente:



La primera actividad crea un Intent y le pasa el valor del EditText Nombre:

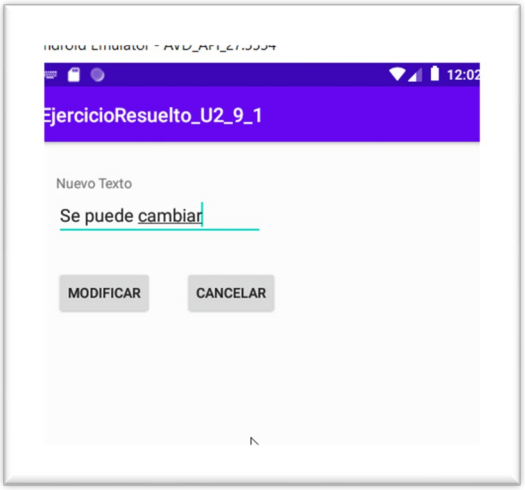
```
Intent i = new Intent( packageContext: this, MainActivity2.class);
i.putExtra( name: "nombre",Nombre.getText().toString());
startActivity(i);
```

La actividad destino, recupera los datos que le ha pasado la actividad llamante del Intent:

```
Bundle extras = getIntent().getExtras();
String s= extras.getString( key: "nombre");
Toast.makeText( context: this, s,Toast.LENGTH_LONG).show();
```

9.3 EjercicioResuelto_U2_9_1

Realiza la siguiente Aplicación:

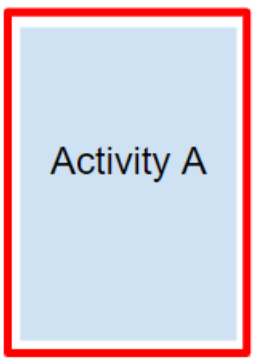


Solución

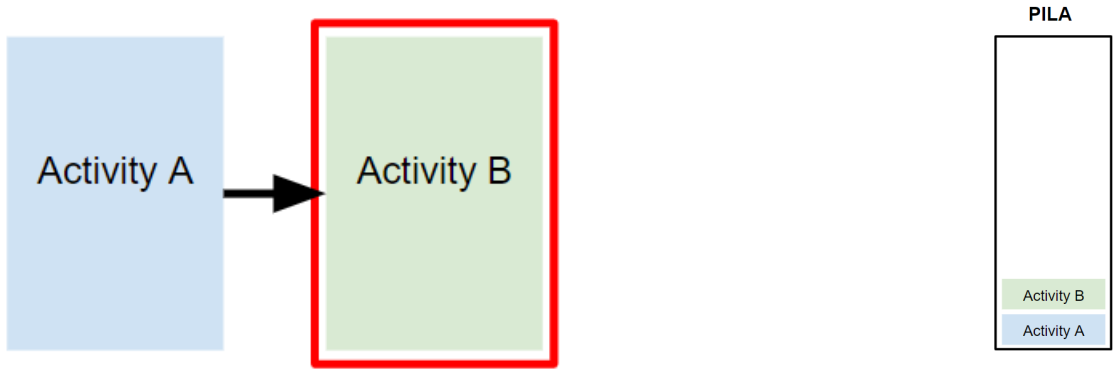
9.4 Pila de Activities (Control del stack)

Las actividades de una aplicación se organizan en una pila (la pila de actividades) en el orden en que se abre cada actividad. Antes de iniciar la aplicación, no hay ninguna Activity activa y la pila está vacía.

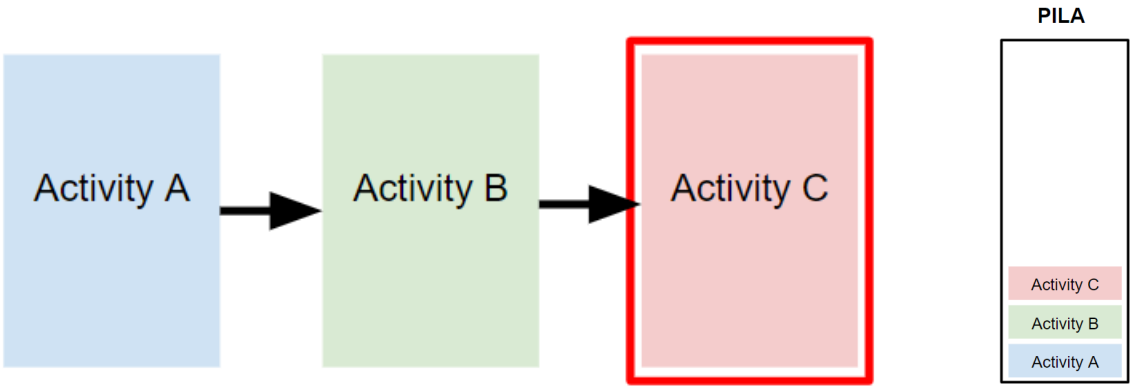
Cuando se inicia la Activity A. Al ser la activa, pasa a la parte superior de la pila.



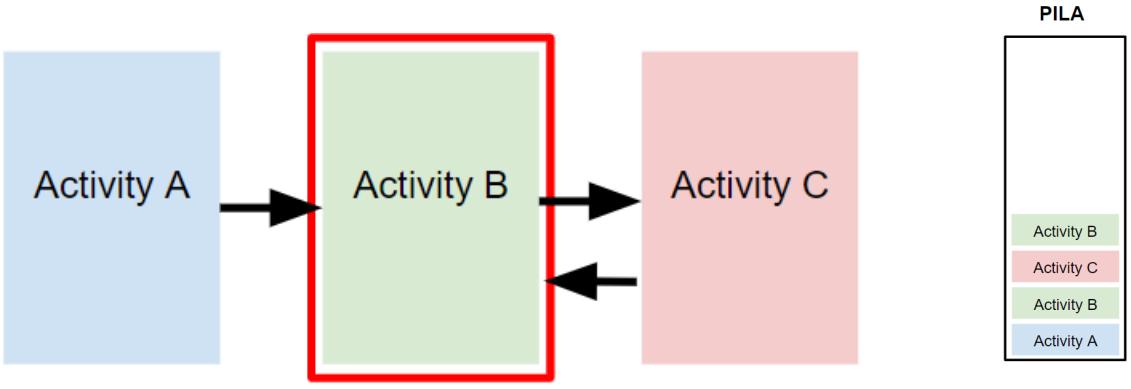
La Activity A llama a la Activity B con un Intent:



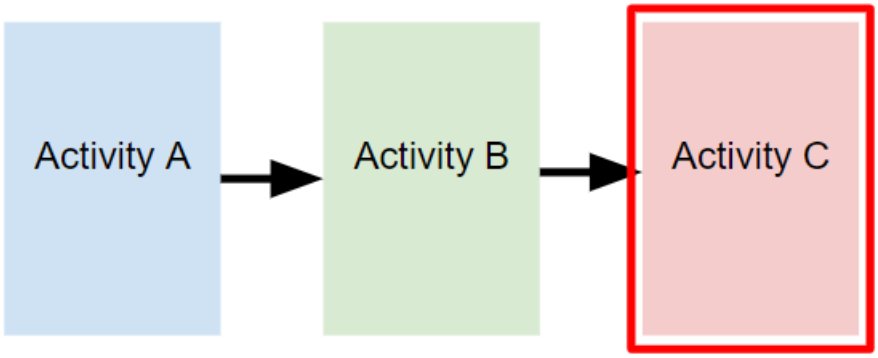
La Activity B llama a la Activity C con un Intent:

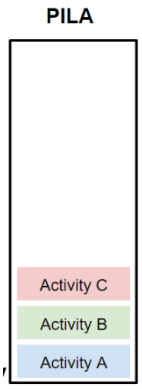


La Activity C llama a la Activity B con un Intent:

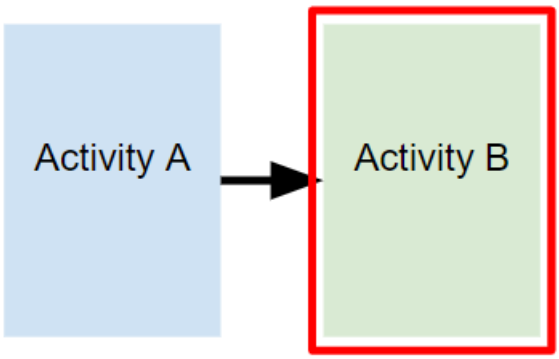


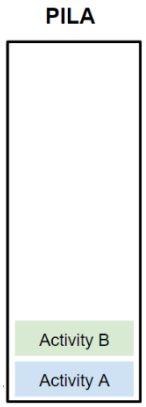
El comportamiento esperado por el programador es que al pulsar “back button” desde la Activity B actual, pase a la Activity A. El comportamiento real es diferente y al pulsar “back button” las activities se muestran según se van desapliando y se mostraría la actividad C:



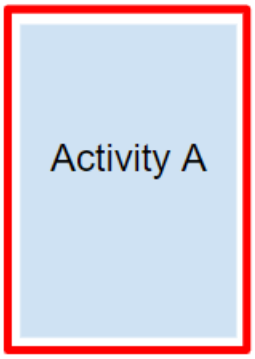


Pulsamos “back button” desde la Activity C y se activa la Activity B:

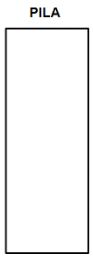




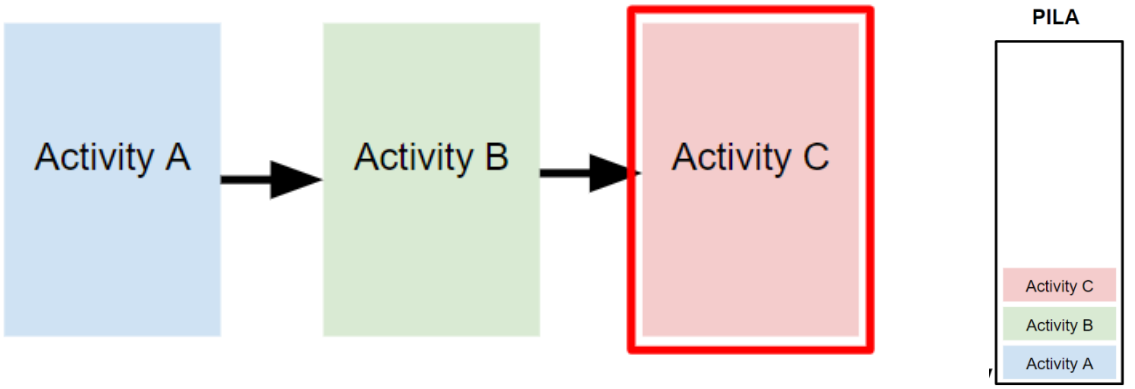
Pulsamos “back button” desde la Activity B y se activa la Activity A:



Pulsamos “back button” desde la Activity A y finaliza la ejecución de la aplicación:



Partiendo de la siguiente situación:

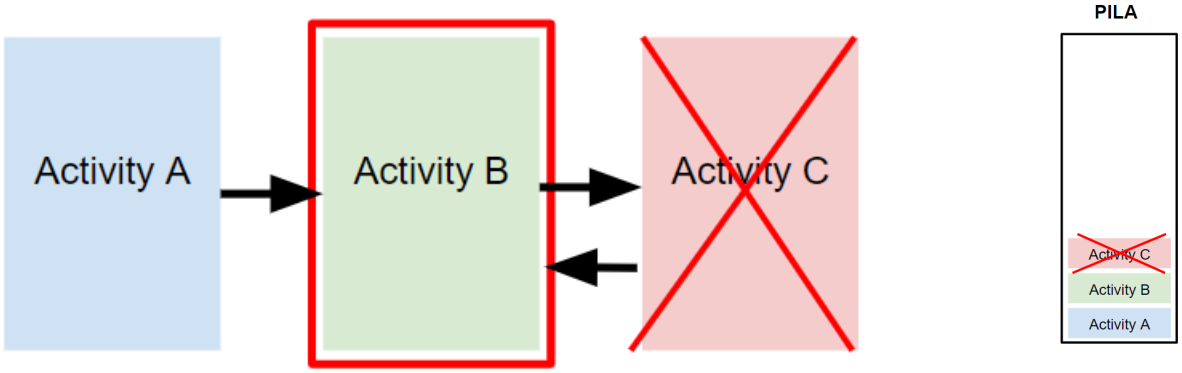


Podemos hacer que el comportamiento de la aplicación sea el esperado, y que se desapile la Actividad C pasando a ser la activa la B.

Al hacer el intent a la Activity B desde la C, debemos indicarle que no se apile con FLAG_ACTIVITY_CLEAR_TOP:

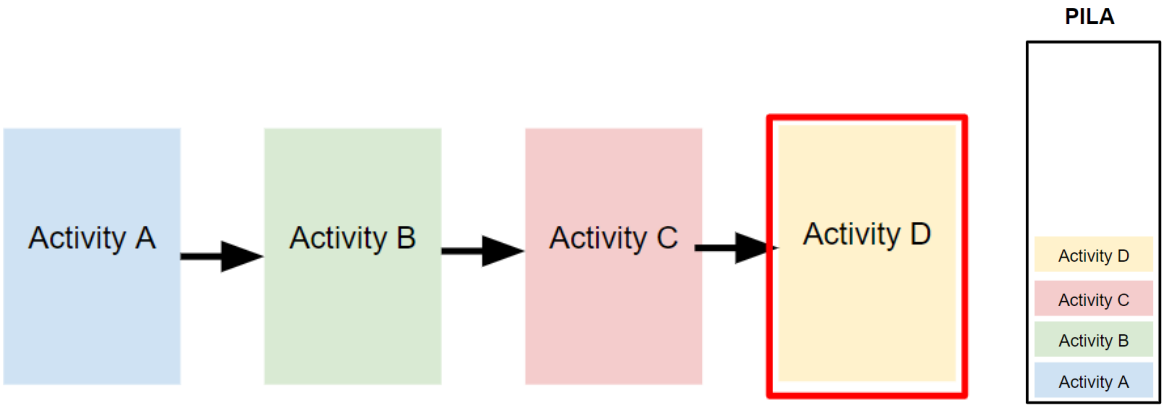
```
Intent intento=new
Intent(ActivityC.class,ActivityB.class);
intento.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intento);
```

Lo cual limpia la pila hasta que se encuentra con la Activity B. Ahora, al pulsar “back button” desde B, se activa A:

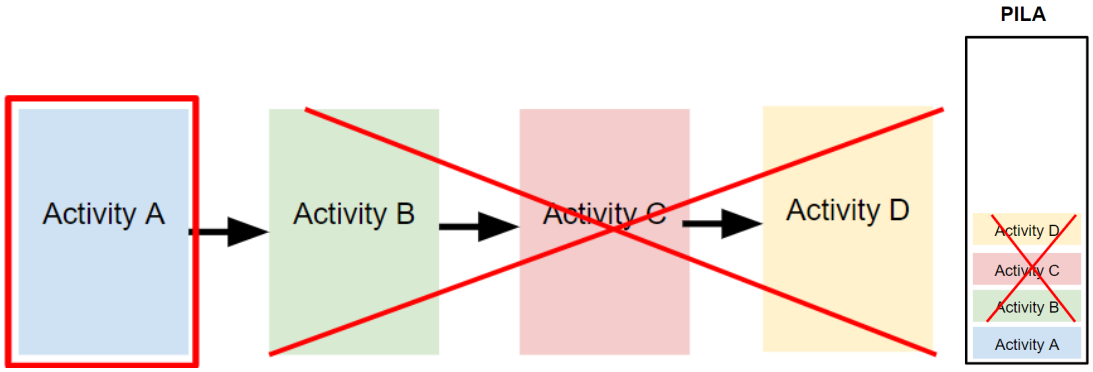


Ejemplo más complejo. ¿Qué pasa al hacer este intent desde Activity D hacia A?

```
Intent intento=new
Intent(ActivityD.class,ActivityA.class);
intento.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intento);
```



Al hacer este intent desde Activity D, se desapilan todas las Activities hasta que se encuentre la A.



10 Menús de opciones

El menú de opciones es la colección principal de elementos de menú de una actividad. Es donde debes colocar las acciones que tienen un impacto global en la app, como "Buscar", "Redactar correo electrónico" y "Configuración". Los pasos a dar para incorporar un menú a tu aplicación son:

1. Crear un fichero xml con los elementos del menú en la carpeta ***res/menu***.
2. Cargar el menú en la actividad donde debe mostrarse redefiniendo el método ***onCreateOptionsMenu()***:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.mi_menu, menu);
    return true;
}
```

3. Programar la acción de cada opción del menú redefiniendo el método ***onOptionsItemSelected()***.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.idOpcion1:
           …
            return true;
        case R.id.idOpcion2:
           …
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

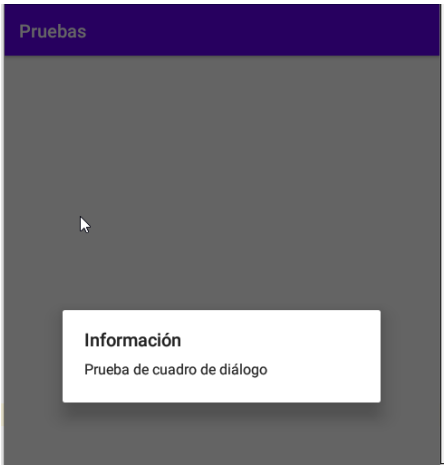
Para ampliar este contenido visita la documentación oficial Menús.

11 Cuadros de diálogo

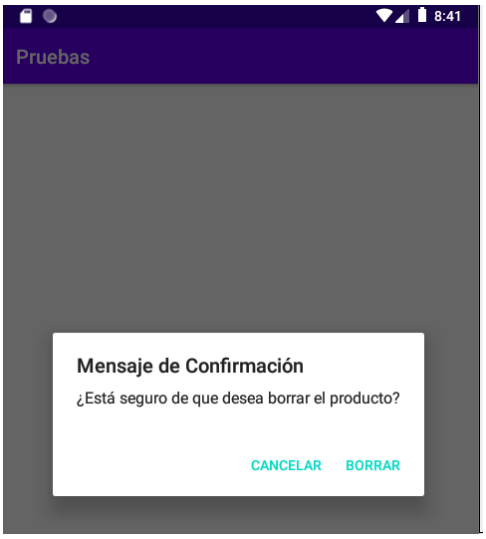
11.1 Simples

Aquí tienes el código de un cuadro de diálogo con un solo botón:

```
AlertDialog.Builder builder = new
AlertDialog.Builder(this);
builder.setTitle("Información");
builder.setMessage("Prueba de cuadro de diálogo");
builder.create();
builder.show();
```



11.2 Con dos Botones




```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

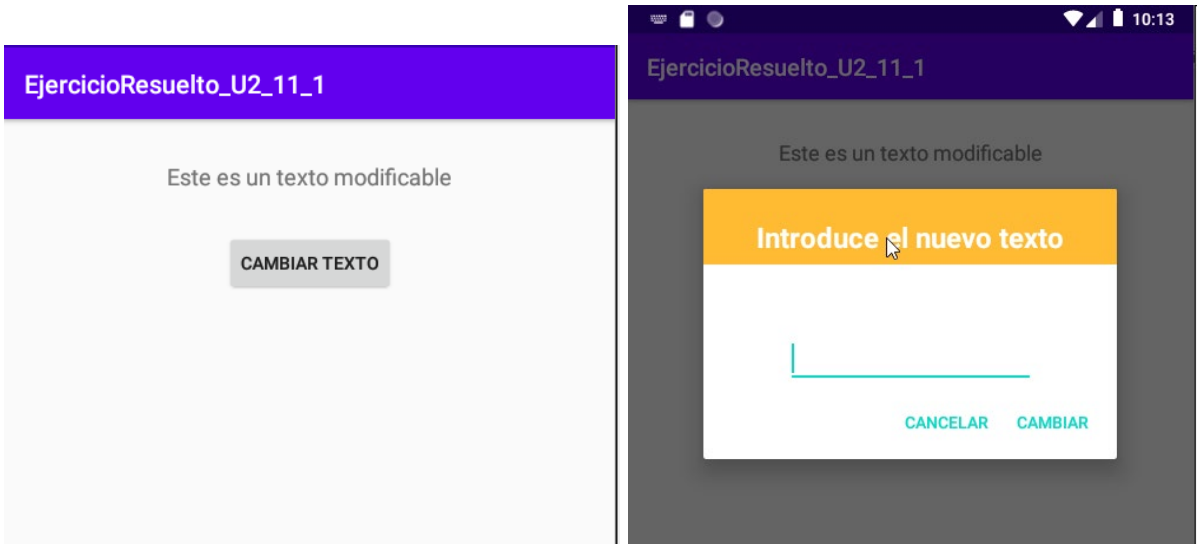
    AlertDialog.Builder dialogo1 = new
AlertDialog.Builder(this);
    dialogo1.setTitle("Mensaje de Confirmación");
    dialogo1.setMessage("¿Está seguro de que desea
borrar el producto?");
    dialogo1.setCancelable(false);
    dialogo1.setPositiveButton("Borrar", new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface
dialogo1, int id) {
            BotonAceptar();
        }
    });
    dialogo1.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialogo1, int id) {
            BotonCancelar();
        }
    });
    dialogo1.show();

}
public void BotonAceptar() {
    Toast t=Toast.makeText(this,"Producto borrado", Toast.LENGTH_SHORT);
    t.show();
}
```

```
public void BotonCancelar() {
    finish();
}
```

11.3 EjercicioResuelto_U2_11_1 Input Dialog

Aquí tienes un ejemplo para crear un cuadro de diálogo personalizado que solicita al usuario que introduzca un valor en un campo dentro del cuadro de diálogo.



Solución

12 Bases de datos SQLITE

Cuando trabajemos con aplicaciones que accedan a bases de datos sqlite, vamo a partir del siguiente esquema:

- Clase **BaseDatos.java**, en la que se define la estructura de la BD. Contiene los métodos de creación y de actualización de la estructura de la BD. Hereda de la clase ***SQLiteOpenHelper***.
- ```
  public class BaseDatos extends SQLiteOpenHelper {
      Context contexto;

      public BaseDatos(@Nullable Context context, @Nullable String name) {
          super(context, "NombreBD", null, 1);
          contexto = context;
      }

      @Override
      //onCreate se ejecuta cuando se CREA la BD, solamente una vez para cada
  versión de la BD
      public void onCreate(SQLiteDatabase db) {
          try {
  ```
  ```

```
                //Creamos la tabla
                db.execSQL("CREATE TABLE ...");
            }
            catch (Exception e){
                //Mensaje de error si no se ha ejecutado correctamente
                Toast.makeText(contexto,"Error al crear la base da
    datos"+e,Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        //onUpgrade se ejecuta cuando se ACTUALIZA la versíon de la BD
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion) {
            try {
                //Eliminamos la tabla anterior (si existe)
                db.execSQL("DROP TABLE IF EXISTS .....");
                //Llamamos a onCreate para que cree la tabla con las nuevas
    especificaciones
                onCreate(db);
            }
            catch (Exception e){
                //Mensaje de error si no se ha ejecutado correctamente
                Toast.makeText(contexto,"Error al actualziar la base de
    datos"+e,Toast.LENGTH_SHORT).show();
            }
        }
```

- Clase **AccesoDatos.java**, en la que se definen los métodos de acceso a la bd, donde se implementarán las operaciones de CRUD.

```
public class AccesoDatos {
    private Context contexto;
    BaseDatos miBD;

    public AccesoDatos(Context contexto) {
        this.contexto = contexto;
        miBD = new BaseDatos(contexto);
    }

    public void ejemplMetodoConsultar(){
        SQLiteDatabase accesoLectura = miBD.getReadableDatabase();
        //Hacer consulta seleccion
    }
    public void ejemploMetodoModificacion(){
        SQLiteDatabase accesoEscritura = miBD.getWritableDatabase();
        ////Hacer consulta inserción/modificación/borrado
    }

}
```

- Las activities, accederan a la BD a través de un objeto de la clase AccesoDatos.java.

```
public class MainActivity extends AppCompatActivity {

    AccesoDatos baseDatos;
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        baseDatos = new AccesoDatos(this);

        baseDatos.metodoConsultar();
        baseDatos.metodoModificacion();
    }
}
```

12.1 Ejecución de consulta

La forma más general de ejecutar una consulta, independientemente del tipo de consulta que sea, es mediante la sentencia ***execSQL***:

```
public void metodoModificacion(){
    SQLiteDatabase accesoEscritura = miBD.getWritableDatabase();
    ////Hacer consulta inserción/modificación/borrado
    accesoEscritura.execSQL("insert into tabla(campo1,campo2) values('valor
campo1', 'valor campo2')");
}
```

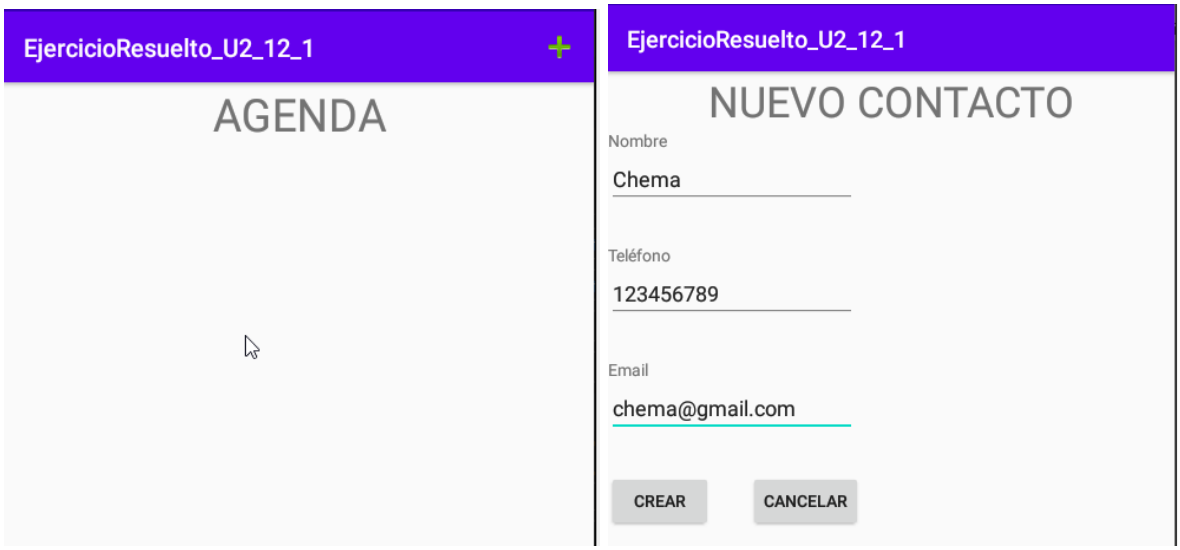
12.2 Inserción

Sentencia ***insert***: Permite controlar el resultado del insert (**devuelve -1 si el insert falla y el ID de la fila si el insert se ejecuta correctamente**):

```
public void metodoModificacion(){
    SQLiteDatabase accesoEscritura = miBD.getWritableDatabase();
    ////Hacer consulta inserción/modificación/borrado
    //Se crea un registro donde se especificará cada campo y su valor
    ContentValues registro = new ContentValues();
    registro.put("campo1", valor1);
    registro.put("campo2",valor2);
    //Resultado permite comprobar si el insert se ha realizado o no
    long resultado = accesoEscritura.insert(tabla,null,registro);
    if(resultado!=-1){
        Toast.makeText(contexto,"Se ha creado el registro con id
"+resultado,Toast.LENGTH_LONG).show();
    }
}
```

12.3 EjercicioResuelto_U2_12_1

Crea la siguiente aplicación que simula una agenda de contactos. Cuando se pulse en insertar (+), se pedirán los datos del contacto a crear y se insertará en la bd.



Solución

12.4 Consulta de selección

Sentencia query: Para realizar una consulta de selección debemos definir qué columnas se van a recuperar, ejecutar la consulta y recorrer el resultado que se devuelve en un cursor. El método query admite como parámetros: La tabla, las columnas a recuperar, las expresiones de la cláusulas where, groupby, having y orderby:

```
query(String table,String[] columns, String selection, String[] selectionArgs,
String groupBy, String having, String orderBy)
```

Ejemplo:

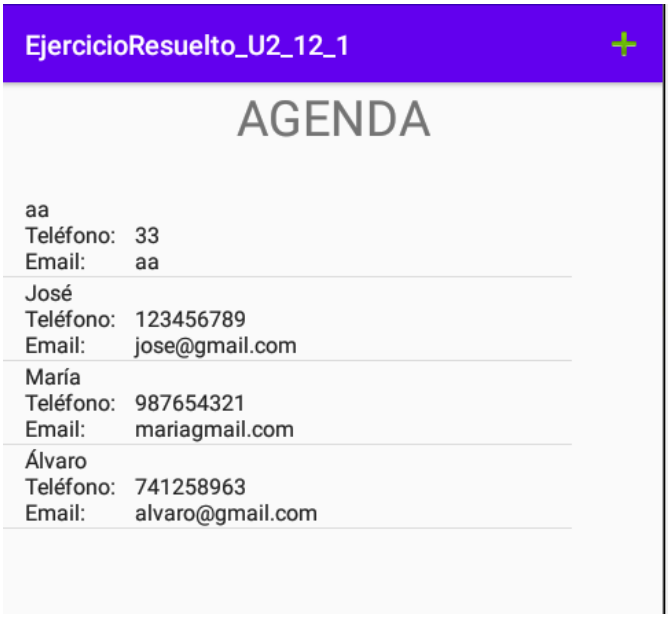
```
//Definimos un array de string con las columnas a recuperar en el select
String[] columnas = {"campo1","campo2"};

//Creamos un acceso para lectura
SQLiteDatabase accesoLectura = miBD.getReadableDatabase();
//Ejecutamos la consulta. Indicamos la tabla y las columnas a recuperar. El
resultado se almacena en un cursor.
Cursor cursor =
accesoLectura.query("tabla",columnas,"campo1="+valor,null,null,null,null);
//Recorremos cada registro que devuelve la consulta
while(cursor.moveToNext()){
    …
}
```

12.5 EjercicioResuelto_U2_12_2

Modifica la aplicación anterior para que en la actividad principal se muestre una lista con los contactos que hay en la agenda. El adaptador de la lista, debe rellenarse con los datos que se obtienen de la base de datos.

Además, debes controlar que cuando se cree un nuevo contacto, la lista se actualice.



Solución.

12.6 Actualización

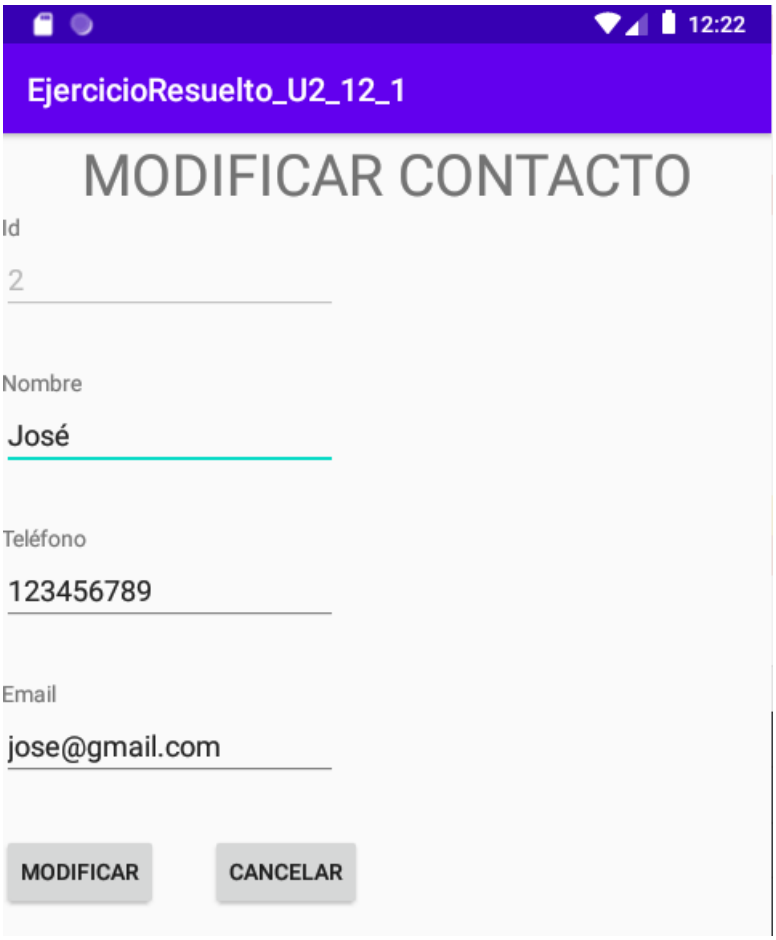
Sentencia ***update***: : Hay que indicar la tabla, los campos a modificar y la claúsula where. Permite controlar el resultado del insert (**devuelve un valor con el número de registros afectados por el update)**:

```
public void metodoModificacion(){
    SQLiteDatabase accesoEscritura = miBD.getWritableDatabase();
    ////Hacer consulta inserción/modificación/borrado
    //Se crea un registro donde se especificará cada campo a modificar y su valor
    ContentValues registro = new ContentValues();
    registro.put("campo1", valor1);
    registro.put("campo2",valor2);
    //Resultado permite comprobar si el update se ha realizado o no
    int resultado = accesoEscritura.update("tabla",registro,"campo=valor",null);

        if(resultado>0){
            Toast.makeText(contexto,"Se han modificado los resgistros
   "+resultado,Toast.LENGTH_LONG).show();
        }
   }
```

12.7 EjercicioResuelto_U2_12_3

Modifica la aplicación anterior para que cuando se seleccione un contacto en la lista, nos lleve a una nueva actividad que permita modificar el contacto seleccionado. Una vez modificado, se debe actualizar la lista de la pantalla principal.



Solución

12.8 Borrado

Sentencia ***delete***: Hay que indicar la tabla y la claúsula where. Permite controlar el resultado del insert (**devuelve un valor con el número de registros afectados por el delete)**:

```
//Creamos un acceso para escritura
SQLiteDatabase accesoEscritura = miBD.getWritableDatabase();
//Resultado contiene el número de registros afectados por el select, en este caso,
debería ser 1
long resultado = accesoEscritura.delete("tabla","campo=valor",null);
```

12.9 EjercicioResuelto_U2_12_4

Modifica la aplicación anterior para que, al hacer una pulsación larga sobre la lista, se borre el contacto seleccionado. El listado deberá actualizarse tras el borrado.

Solución

13 Bases de datos remotas

Con SQLite solo podemos trabajar con una Base de Datos local. Para poder acceder a una Base de Datos remota (por ejemplo, en internet) tenemos que utilizar web services.

Un servicio web es una interfaz escrita en un lenguaje de programación (php en nuestro caso) que es la que realmente accede a la Base de Datos (por ejemplo, mySQL). La información devuelta debe estar en un formato lo más estándar posible: habitualmente JSON o XML.

Los pasos a seguir para realizar una app que acceda a un servidor de BD remoto son:

1. En la parte servidor:
   a. Instalación del servidor web+php+mySQL y creación de la Base de Datos y tablas.
   b. Creación del servicio web
2. En la parte cliente (aplicación Android):
   a. Utilización del servicio web a través del protocolo http
   b. Manejo de los datos devueltos por el servicio web

Ejemplo Inserción de datos desde una app usando un servicio web.

Ejemplo de consulta de datos desde una app usando un servicio web.