5. Programación avanzada en Android.

Contenido

1. Introducción

La programación de aplicaciones para dispositivos móviles, en general, consta de más funcionalidades que unas simples pantallas, una de las características más importantes del interfaz es el **paso de información entre Actividades**. Una vez tenemos claro el esquema de la aplicación y la transición entre pantallas, es el momento de ir más allá y atribuirle una serie de particularidades específicas.

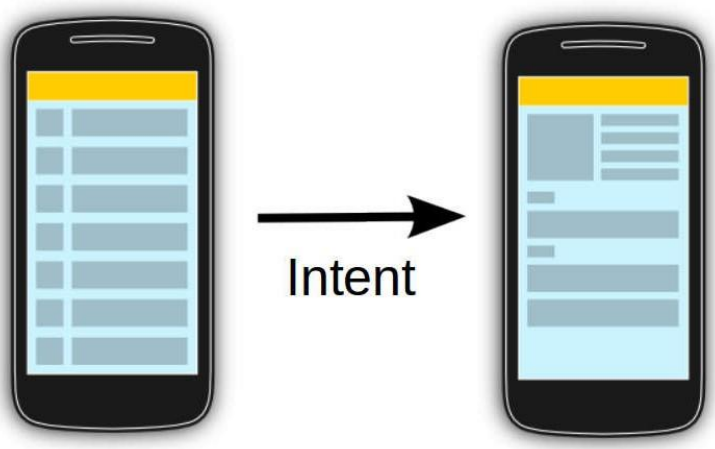
Android nos proporciona una API y unas librerías de clases que dan al programador la oportunidad de crear aplicaciones completas. Aparte de la programación de la interfaz gráfica de la aplicación (con todas sus complejidades), existen otros elementos que pueden enriquecer su aplicación como la **persistencia de datos.**

En el apartado "Programación avanzada" se verán técnicas avanzadas de programación en Android y se tratarán diferentes formas de obtener persistencia de datos mediante bases de datos.

2. 'Intents'

Los *intents* son mensajes enviados entre los diferentes elementos que conforman las aplicaciones de Android. Las actividades, servicios y broadcast receivers (receptores de broadcast) son activados a través de estos mensajes. Los **intents le indican a una actividad que se inicie, o a un servicio que comience o se detenga, o son simplemente mensajes de broadcast** (dirigidos a todas las aplicaciones del dispositivo). En definitiva, sirven para comunicar componentes de la misma aplicación u otros.

Los intents sirven para comunicar diferentes componentes de la misma aplicación o aplicaciones diferentes. Generalmente hacen referencia a una acción a realizar y van acompañados de dos elementos: la acción que ejecutará y la información que ésta necesita.



El intent es un objeto de la clase *Intent* y contiene la información de la operación que se quiere realizar o, en el caso de los broadcast, la descripción de algún evento que ha tenido lugar en el sistema. Existen diferentes maneras de enviar los intents en función del tipo de componente al que van dirigidos. Para dirigir un intent a una actividad se pueden utilizar los métodos:

- Context.startActivity(Intent intent)
- Activity.startActivityForResult(Intent intent, intrequestCode)

dependiendo de si se espera que la actividad devuelva alguna información o no.

Por ejemplo, para iniciar otra actividad:

```
# Start the activity connect to the
# specified class

Intent i = new Intent(this, ActivityTwo.class);
startActivity(i);
```

Context : *es el contexto del estado actual de la aplicación, permite a los nuevos objetos entender que está sucediendo. Normalmente se utiliza para obtener información de otra parte del programa, acceder recursos y clases específicos de la aplicación, lanzar actividades, broadcasting y recibir Intents. Puede invocarse mediante getApplicationContext(), getContext(), getbaseContext() o this (desde la actividad).*

2.1. Tipos de "intents'

Hay dos tipos de intents diferentes:

- **Explícitos:** donde se especifica el componente explícitamente por su nombre (el campo nombre del componente del Intent tiene un valor introducido). Generalmente, no podemos saber el nombre específico de los componentes de otras aplicaciones, por eso los intents explícitos generalmente se utilizan para llamar componentes de **nuestra propia aplicación**.

  Ejemplo:

```
Intent i = new Intent(this, ActivityTwo.class);
```

  Si queremos que nuestra actividad pueda ser llamada desde otra aplicación deberemos incluir en el manifest un **filtro de intent** (ver punto 3.3) exponiendo nuestra actividad:

```
<intent-filter>
      <action android:name=”com.example.miguel.intents.ACTIVIDAD2" />
      <category    android:name="android.intent.category.DEFAULT"    />
</intent-filter>
```

  De esta manera podemos llamar a esta actividad desde nuestra aplicación o desde cualquier otra, utilizando el nombre de la acción indicada en el intent:

```
startActivity (new Intent ("com.example.miguel.intents.Actividad2"));
```

- **Implícitos:** los intents implícitos especifican la acción que debe realizarse y opcionalmente datos que proporciona contenido para la acción. Si el intent implícito se envía al sistema Android, busca todos los componentes que están registrados para la acción específica y el tipo de datos apropiado. Si sólo se encuentra un componente, Android inicia este componente directamente. Si varios componentes están identificados por el sistema Android para esa acción, el usuario obtendrá un diálogo de selección y puede decidir qué componente se debe utilizar.

Por ejemplo, el siguiente le indica al sistema Android que abra una página web. Todos los navegadores web instalados deben estar registrados a través de un **filtro de intent**, éstos no tienen definido el nombre del componente al que van dirigidos. Generalmente, se usan para activar componentes de otras aplicaciones.

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

Como los intents implícitos no especifican cuál es el componente al que va dirigido el intent, el sistema tiene que encontrar el mejor componente para tratar el intent. Por hacerlo, compara los contenidos definidos en el objeto Intent con unas estructuras definidas en los componentes que sirven para anunciar al sistema las capacidades del componente. Estas estructuras sirven para decir al sistema qué tipo de intents es capaz de tratar el componente y se llaman intent filters (filtros de intents).

Los intent filters sirven para decir al sistema las capacidades del componente. El sistema sabrá qué tipo de intent puede responder el componente comparando las características del intent con el filtro de intent.

2.2. Filtros de "intents'

Por ejemplo, se puede crear una actividad que sirva para ver páginas web. Puede anunciar al sistema que su actividad está abierta a recibir intents para visitar sitios web. La próxima vez que se haga un intent para visitar una web desde el dispositivo, su actividad será una de las candidatas para abrir la web (en el caso de que haya varios componentes para atender intent, el sistema pedirá al usuario qué quiere usar).

Los **filtros permiten que los componentes puedan recibir intents implícitos.** Si un componente no tiene intent filters, únicamente puede recibir intents explícitos (con su nombre). Un componente que define filtros de intents puede recibir intents tanto implícitos como explícitos.

Un componente tendrá un filtro diferente para cada trabajo que puede hacer. Por ejemplo, una aplicación multimedia puede tener diferentes tipos de filtros para anunciar que puede abrir diferentes tipos de archivos (fotos, audio, vídeo, etc.).

Un filtro es una instancia de la clase IntentFilter. El sistema compruebalos componentes para ver cuál puede recibir el intent implícito, de ahí que deba conocer las capacidades del componente antes de ponerlo en marcha. Por eso los filtros se han definir en el archivo **AndroidManifest.xml** como elementos **<intent-filter>.**

Un filtro tiene campos similares a los del objeto Intent: acción, datos y categoría. Los intents implícitos se comparan con los campos del filtro. Para que el intent sea enviado al componente, la comparación con lostres campos del filtro debe ser válida. Se realiza un test con cada campo por separado:

- Acción
- Categoría
- Datos (el URI y el tipo de datos)

Ejemplo, declarar un intent para recibir mensajes de texto :

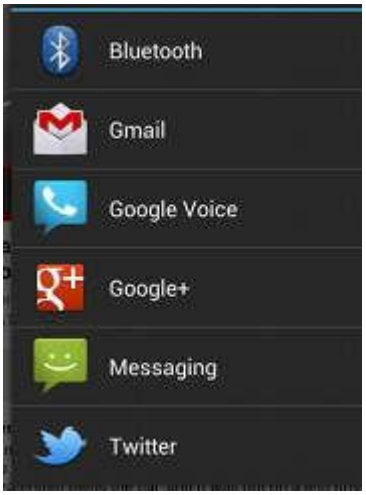
```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Para llamar a dicha actividad :

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Como normalmente tendremos registradas muchas aplicaciones capaces de recoger un mensaje, Android nos mostrará un diálogo para que elijamos que Actividad queremos que lo abra.



2.3. Paso de información entre actividades

Los extras son **pares clave-valor** que aportan información adicional que se ha enviar al componente que trate el intent. Así, diferentes URI tienen asociados algunos extras en particular. Por ejemplo, la acción ACTION_HEADSET_PLUG, que corresponde a un cambio en el estado de la conexión de los auriculares del dispositivo, tiene un campo extra indicando si los auriculares se han enchufado o no.

Ejemplo:

```
Intent i = new Intent(this, ActivityTwo.class);
i.putExtra("Value1", "This value one for ActivityTwo ");
i.putExtra("Value2", "This value two ActivityTwo");
```

También se puede usar un objeto del tipo Bundle para añadir extras. Un objeto de la clase Bundle (literalmente del inglés, 'paquete' o 'haz') permite almacenar una serie de datos (que pueden ser de distinto tipo). Las variables se pueden añadir al Bundle simplemente dando un nombre y la variable que se quiere guardar. En cierto modo, recuerda los arrays con que se pasan variables entre losformularios en PHP: un array de tuplas [nombre, valor]. Puede utilizar los métodos putExtras (Bundle extras) y getExtras () para añadir y obtener extras del intent.

```
Intent i = new Intent(this, ActivityTwo.class);
Bundle extras = new Bundle();
extras.putString("Value1", "This value one for ActivityTwo ");
extras.putString("Value2", "This value two ActivityTwo");
i.putExtras(extras);
startActivity(i);
```

Para recoger los datos enviados a través del Intent en la nuevaactividad (en su método OnCreate)

```
Intent intent = getIntent();
String value = intent.getStringExtra("Value1");

//o bien si es un bundle
Bundle bundle = this.getIntent().getExtras();
String nombre= bundle.getString("NOMBRE"));
```

Si se inicia la actividad con la llamada al método **startActivityForResult(),** se espera que la retroalimentación de la sub-actividad. Una vez que termina la sub-actividad, el método onActivityResult () en la subactividad se llama y se puede realizar acciones basadas en el resultado.

En el método startActivityForResult () se puede especificar un código de resultado para determinar qué actividad comenzó. Este código de resultado se devuelve en el método onActivityResult. La actividad comenzó también puede establecer un código de resultado que la persona que llama puede utilizar para determinar si la actividadse ha cancelado o no.

```
        Intent i = new Intent(this, ActivityTwo.class);

        // set the request code to any code you like,
        // you can identify the callback via this code
        startActivityForResult(i, REQUEST_CODE);
```

La actividad ActivityTwo le devuelve los datos a través del métodofinish():

```
@Override
public void finish() {
        Intent data = new Intent();
        data.putExtra("returnKey1", "Datos de retorno. ");
        setResult(RESULT_OK, data);
        super.finish();
}
```

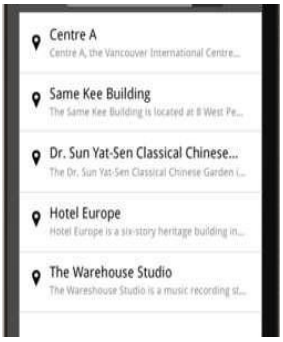
La actividad original puede recibir los datos con el método onActivityResult():

```
@Override
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
  if (resultCode == RESULT_OK && requestCode ==
REQUEST_CODE) {
      if (data.hasExtra("returnKey1")) {
          Toast.makeText(this,
                  data.getExtras().getString("returnKey1"),
                  Toast.LENGTH_SHORT).show();
      }
  }
}
```

3. Interfaz de usuario avanzada

Vamos a revisar algunos de los elementos de la interfaz de usuario máscaracterísticos de Android, algunos de ellos son de recientes introducción y han tenido una gran acogida apareciendo en la mayoríade las aplicaciones actuales.

3.1. ListViews



Existen muchos elementos que podemos utilizarpara mejorar el UI (Interfaz de usuario) en Android, uno de los más utilizados son los ListView que muestran una serie de elementos con una disposición similar a un LinearLayout.

Cada elemento del ListView está definido en un layout (común a todos los elementos) donde podemos definir qué y cómo se va a mostrar su contenido. Para insertar el contenido en los ListView se utilizan los adapters. *Un* **adaptador** gestiona el modelo de datos y lo adapta a lasentradas individuales en el widget. Un adaptador extiende laclase BaseAdapter.

Cada línea en el widget que muestra los datos consiste en un diseño que puede ser tan complejo como usted desee. Una línea típica de una lista
tiene una imagen en el lado izquierdo y dos líneas de texto en el mediocomo se muestra en el siguiente gráfico.

Normalmente necesitaremos los siguientes elementos para utilizar un ListView :

- Control ListView en un layout.
- Layout de una línea del layout (veáse listview_layout.xml).
- Una clase para almacenar los datos que se mostrarán en ellayout.
- Un adaptador (una clase que extiende de ArrayAdapter) para gestionar la conexión entre los datos y la presentación.

Un archivo de diseño (layout) de una línea de este tipo podría ser similar a la siguiente (listview_layout.xml)

```
<RelativeLayout      xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip" >
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentBottom="true"
        android:layout_alignParentTop="true"
        android:layout_marginRight="6dip"
        android:contentDescription="TODO"
        android:src="@drawable/ic_launcher" />
    <TextView
        android:id="@+id/lblTitulo"
        android:layout_width="fill_parent"
        android:layout_height="26dip"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_toRightOf="@id/icon"
        android:ellipsize="marquee"
        android:singleLine="true"
        android:text="Description"  android:textSize="12sp"
        />
    <TextView
        android:id="@+id/lblSubtitulo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/secondLine"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_alignWithParentIfMissing="true"
        android:layout_toRightOf="@id/icon"
        android:gravity="center_vertical"
        android:text="Example                    application"
        android:textSize="16sp" />
</RelativeLayout>
```

Necesitaremos una clase para almacenar los datos de cada fila, por ejemplo **Titular.java** :

```
public class Titular
{
    private String titulo;
    private String subtitulo;

    public Titular(String tit, String sub){
        titulo = tit;
        subtitulo = sub;
    }
    public String getTitulo(){
        return titulo;
    }
    public String getSubtitulo(){
        return subtitulo;
    }
}
```

También necesitaremos una clase adaptador, por ejemplo AdaptadorTitulares.java. La clase ArrayAdapter puede manejar una lista o una matriz de objetos Java como entrada. Cada objeto Java se asigna a una fila.

```
class AdaptadorTitulares extends ArrayAdapter<Titular> {
    private Titular[] datos;//la clase titular almacena los datos de
                            //  cada línea
    public AdaptadorTitulares(Context context, Titular[]
                datosEnviados) {
        super(context, R.layout.listview_layout, datosEnviados);
        datos=datosEnviados;//recibimos el array de objetos que contienen los
datos a mostrar en el listview
    }

    //este método se llama una vez por cada elemento del listview
    public View getView(int position, View convertView, ViewGroup
                parent)                                                    {
        LayoutInflater inflater = LayoutInflater.from(getContext());
        View item = inflater.inflate(R.layout.listview_layout, null);
        //obtenemos las referencias a los elementos del
        // listview_layout y les introducimos los datos
        TextView lblTitulo =
            (TextView)item.findViewById(R.id.LblTitulo);
        lblTitulo.setText(datos[position].getTitulo());
        TextView lblSubtitulo =
             (TextView)item.findViewById(R.id.LblSubTitulo);
        lblSubtitulo.setText(datos[position].getSubtitulo());
        return(item);
    }
}
```

El adaptador “infla” el diseño para cada fila en su getView() método yasigna los datos de los puntos de vista individuales en la fila.

El adaptador se asigna a la ListView a través del **método setAdapter** en el objeto ListView.

Desde la **actividad principal** se crean los datos para el listview y sele asigna el adapter al listview :

```
// array de objetos de tipo Titular declarados como propiedad del activity_main
private Titular[] datosTitular =
        new Titular[]{
                new Titular("Título 1", "Subtítulo largo 1"),
                new Titular("Título 2", "Subtítulo largo 2"),
                new Titular("Título 3", "Subtítulo largo 3"),
                new Titular("Título 4", "Subtítulo largo 4");
….

@Override
protected void onCreate(Bundle savedInstanceState) {
….
ListView lstOpciones = (ListView)findViewById(R.id.LstOpciones);
//instanciación del objeto adaptador al cual le pasamos los datos a mostrar
AdaptadorTitulares adaptadorTitulares =
        new AdaptadorTitulares(this, datosTitular);
//asignamos el adaptador a nuestro listview
lstOpciones.setAdapter(adaptadorTitulares);
….
```

El resultado final será similar a la siguiente imagen:

Los Adaptadores no sólo son utilizados por ListView, sino también por **otros controles** que extienden la clase AdapterView, como, por ejemplo, Spinner GridView Gallery y StackView.

Para reaccionar a las selecciones en la lista, establezca un OnItemClickListeneren el ListView

```
listView.setOnItemClickListener(new OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view,
    int position, long id) {
    Toast.makeText(getApplicationContext(),
      "Pulsado el ítem número: " + position, Toast.LENGTH_LONG)
      .show();
  }
});
```

3.2. RecycleView

Recientemente el componente ListView ha sido sustituido por el RecyclerView, por tanto ListView, aunque sigue siendo posible utilizarlo, se ha marcado como obsoleto.

El funcionamiento es muy similar, lo que más cambia es el adaptador dado que se añade el concepto de **ViewHolder**, es una forma que tiene de optimizar la visualización de la lista reutilizando las celdas. Si quisiéramos modificar el ejemplo anterior, el adaptador quedaría así:

```
class AdaptadorTitulares extends
        RecyclerView.Adapter<AdaptadorTitulares.ViewHolder>  {

    private List<Titular> datos;
    public AdaptadorTitulares(List<Titular> datosEnviados) {
        datos=datosEnviados;//Guardamos la lista en un atributo de la clase
    }

    @Override //Creamos el layout y se lo pasamos al ViewHolder
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View v =
         LayoutInflater.from(parent.getContext()).
                      inflate(R.layout.listview_layout, parent, false);
        ViewHolder viewHolder = new ViewHolder(v);
        return viewHolder;
    }

    @Override //Metemos los datos en el layout, en el elemento position
    public void onBindViewHolder(ViewHolder holder, int position) {
        final String titulo = datos.get(position).getTitulo();
        holder.title.setText(titulo);

        String subtitulo = datos.get(position).getSubtitulo();
        holder.subtitle.setText(subtitulo);

   }

    @Override
    public int getItemCount() return datos.size();

    public static class ViewHolder extends RecyclerView.ViewHolder {
        private TextView title;
        private TextView subtitle;
        public ViewHolder(View v) {
            super(v);
            title = (TextView) v.findViewById(R.id.LblTitulo);
            subtitle = (TextView) v.findViewById(R.id.LblSubTitulo);
        }
    }
```

Como puede observarse lo que en el ListView se realizaba en el getViewpasa a realizarse en el ViewHolder, recoge referencias a los controles de la línea, y en el onBindViewHolder donde se introducen los datos enlos controles.

En el MainActivity lo mantendríamos igual, teniendo en cuenta que envez de un array hay que pasarle una lista al adaptador y que debemosindicar el tipo de layout que se va a utilizar en el RecyclerView, el máshabitual en el lineal.

En nuestro caso:

```
// use a linear layout manager
 miRecycler.setLayoutManager(new LinearLayoutManager(this));
```

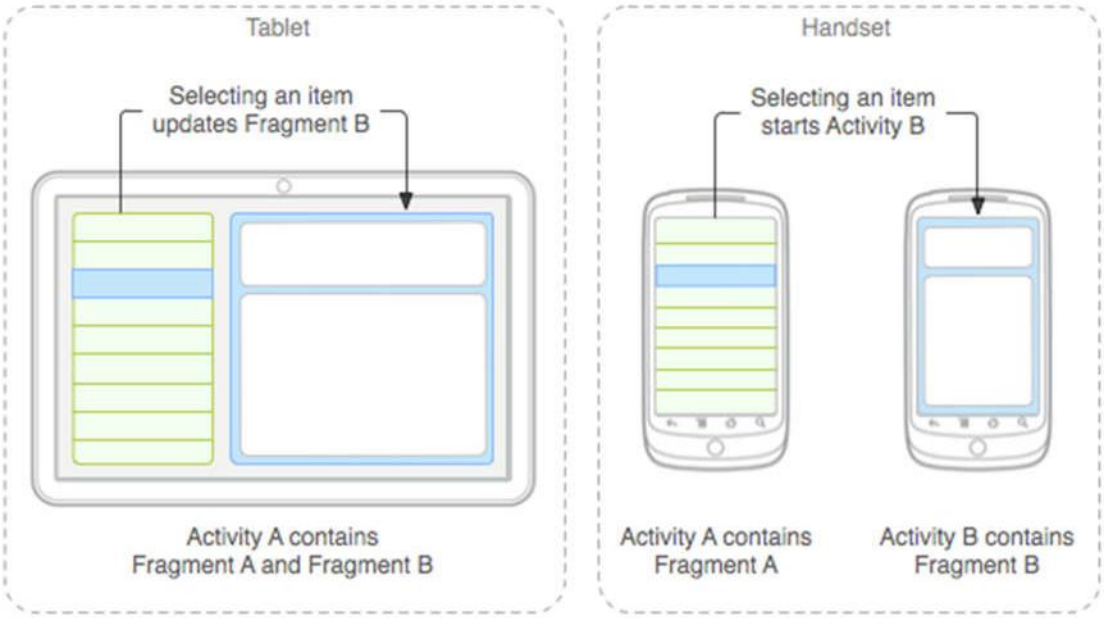
3.3. Fragments

Un fragmento es una clase reutilizable que implementa una parte de una actividad. Un Fragmento normalmente define una parte de una interfaz de usuario. Los fragmentos deben estar integrados en las actividades; no pueden funcionar independientemente de las actividades.

Para entender los fragmentos:

- Un fragmento es una combinación de un archivo de diseño XML y una clase de Java similar a una actividad.
- Utilizando la biblioteca de soporte, los fragmentos son compatibles con todas las versiones relevantes de Android (se introdujeron en el API 11).
- Los fragmentos encapsulan vistas y lógica para que sea más fácil de reutilizar dentro de las actividades. Permiten adaptar lainterfaz a todo tipo de pantallas.

- Los fragmentos son componentes independientes que pueden contener vistas, eventos y lógica.



Creación de un Fragmento (clase que hereda de fragment). Puedecrearse con un asistente New->Fragment->Fragment(Blank).

```
public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.miFragment_view, container, false);
    }
}
```

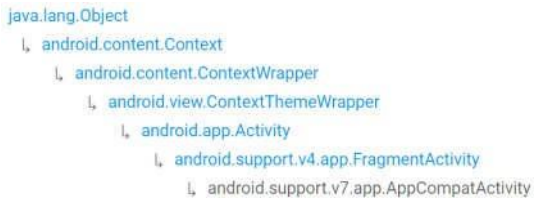
Añadir un fragmento de forma **estática** a una activity a través del xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment class="com.example.android.fragments.miFragment"
              android:id="@+id/mi_fragment"
              android:layout_weight="2"
              android:layout_width=" match_parent "
              android:layout_height="match_parent" />

</LinearLayout>
```

Cargamos el layout en nuestro activity, heredamos de AppCompatActivity que hereda de FragmentActivity y nos proporciona compatibilidad con versiones anteriores del API 11, en caso contrario podemos usar la clase Activity como siempre:




```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
    }
}
```

Si deseamos **cargar el Fragmento de forma dinámica**, para por ejemplo adaptarlo a distintos tamaños de pantalla, debemos usar la clase FragmentManager. En el layout de nuestra actividad estableceremos una marca donde se insertará el fragmento en tiempode ejecución:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

  <FrameLayout
       android:id="@+id/mi_fragmento"
       android:layout_width="match_parent"
       android:layout_height="match_parent">
  </FrameLayout>

</LinearLayout>
```

En la actividad principal:

```java
// Comienza la transacción
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
// Reemplaza el contenido del contenedor con el nuevo fragmento
ft.replace(R.id.your_placeholder, new FooFragment());
// Completa los cambios
ft.commit();
```

Otra forma si deseamos **cargar los Fragmentos de forma que se adapten al tamaño de la pantalla**, es crear varios layout con el mismo nombre y los guardaremos en distintas carpetas, por ejemplo, el activity_main.xml en las carpetas :

- */res/layout :* para pantallas de smartphones
- */res/layout-large :* para pantallas de tablets
- */res/layout-large-port :* para pantalla de tablets en modo vertical

En cada una de ellas se establecerá un layout diferente con los fragmentos que consideres.

Ejemplo: activity_main.xml en carpeta /res/layout con solo un fragmento.

```xml
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="com.example.roberto.correo.FragmentListado"
    android:id="@+id/FrgListado"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Ejemplo: activity_main.xml en carpeta /res/layout-large con dos fragmentos (al ser mayor la pantalla pueden verse los dos simultáneamente).

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

    <fragment class="com.example.roberto.correo.FragmentListado"
        android:id="@+id/FrgListado"
        android:layout_weight="30"
        android:layout_width="0px"
        android:layout_height="match_parent" />

    <fragment class="com.example.roberto.correo.FragmentDetalle"
        android:id="@+id/FrgDetalle"
        android:layout_weight="70"
        android:layout_width="0px"
        android:layout_height="match_parent" />

</LinearLayout>
```

En la actividad principal podemos recoger la referencia al fragmentomediante:

```
FragmentListado frgListado
        =(FragmentListado)getSupportFragmentManager()
        .findFragmentById(R.id.FrgListado);
```

Comunicando fragmentos a través de un listener:

Si un fragmento necesita comunicar eventos a la actividad, el fragmento debe **definir una interfaz** como un tipo interno y es necesario que la actividad implemente esta interfaz.

Por ejemplo dentro de nuestro fragmento FragmentListado:

```
public interface CorreosListener {
    void onCorreoSeleccionado(Correo c);
}

public void setCorreosListener(CorreosListener listener) {
    this.listener=listener;
}
```

Y en la actividad principal debemos implementar el listener declaradoen el fragmento:

```
public class MainActivity extends AppCompatActivity
        implements FragmentListado.CorreosListener{
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    FragmentListado frgListado
            =(FragmentListado)getSupportFragmentManager()
            .findFragmentById(R.id.FrgListado);

    frgListado.setCorreosListener(this);
}

@Override
public void onCorreoSeleccionado(Correo c) {
    boolean hayDetalle =
            (getSupportFragmentManager().findFragmentById(R.id.FrgDetalle) !=
null);

    if(hayDetalle) {
        ((FragmentDetalle)getSupportFragmentManager()

.findFragmentById(R.id.FrgDetalle)).mostrarDetalle(c.getTexto());
    }
    else {
        Intent i = new Intent(this, DetalleActivity.class);
        i.putExtra(DetalleActivity.EXTRA_TEXTO, c.getTexto());
        startActivity(i);
    }
}
```

Como vemos en las últimas líneas, se comprueba si existe otro fragmento (FragmentDetalle) y en caso de que exista se le llama pasándole unos datos a través de un Intent con extras. Podemos **comunicar datos entre fragmentos** igual que lo hacíamos con las actividades.

3.4. ToolBar

La ToolBar en la sucesora de la AppBar (también conocida como ActionBar) y se introdujo en el API 21, es uno de los elementos de diseño más importantes en las actividades, ya que proporciona una estructura visual y elementos interactivos que son familiares para los usuarios.

Es un View que puede colocarse en cualquier lugar del layout XML, laToolBar es una generalización de la ActionBar, las diferencias másimportantes respecto a esta son:

- ToolBar es un View en un layout.
- Al ser un View puede colocarse, animarse y controlarse más fácilmente.
- Podemos tener varias Toolbar es una actividad.

Usando una ToolBar como ActionBar

Debes asegurarte de utilizar la librería de soporte AppCompat-v7, paraello en el fichero build.gradle:

```
dependencies {
  ...
  compile 'com.android.support:appcompat-v7:24.2.0'
}
```

Debes desactivar el tema por defecto y extender del tema Theme.AppCompat.NoActionBar dentro del fichero res/styles.xml:

```
<resources>
  <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
  </style>
</resources>
```

Ahora debes añadir una ToolBar al layout de tu actividad o fragmento,puedes colocarlo en cualquier sitio del layout. Vamos a colocarlo en laparte superior de un LinearLayout como haríamos con el ActionBar:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
      android:id="@+id/toolbar"
      android:minHeight="?attr/actionBarSize"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      app:titleTextColor="@android:color/white"
      android:background="?attr/colorPrimary">
    </android.support.v7.widget.Toolbar>
</LinearLayout>
```

En la actividad o fragmento vamos a establecer el ToolBar para queactúe como un ActionBar llamando a setSupportActionBar(Toolbar).

```
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;

public class MyActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

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

Tenemos que asegurarnos de tener los ítems que utilizamos es un fichero de recursos de menú en un fichero como *res/menu/menu_main.xml* que se "infla" en el método *onCreateOptionsMenu*:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/miCompose"
        android:icon="@mipmap/ic_compose"
        app:showAsAction="ifRoom"
        android:title="Compose">
    </item>
    <item
        android:id="@+id/miProfile"
        android:icon="@mipmap/ic_profile"
        app:showAsAction="ifRoom|withText"
        android:title="Profile">
    </item>
</menu>
```

Las imágenes ic_compose e ic_profile son recursos que se han introducido utilizando New->ImageAsset sobre la carpeta drawable. Elresultado :

El **titulo** mostrado es la propiedad label de la aplicación (android:label="@string/app_name")dentro del fichero manifest.

Recoger el evento click sobre la Toolbar

Podemos hacerlo de forma similar a un botón utilizando la propiedad Android: onClick en el Xml :

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/item1"
        android:icon="@drawable/ic_compose"
        android:onClick="onComposeAction"
        app:showAsAction="ifRoom"
        android:title="Compose">

    </item>
</menu>
```

Que codificaríamos en la actividad:

```
public class MainActivity extends AppCompatActivity {
  public void onComposeAction(MenuItem mi) {
     // handle click here
  }
}
```

Otra manera sería utilizando el método onOptionItemSelected() :

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.miCompose:
            composeMessage();
            return true;
        case R.id.miProfile:
            showProfileView();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

3.5. Navigation View

El panel lateral de navegación es un panel en el que se muestran las principales opciones de navegación de la app en el borde izquierdo dela pantalla. La mayor parte del tiempo está oculto, pero aparece cuando el usuario desliza un dedo desde el borde izquierdo de la pantalla o, mientras está en el nivel superior de la app, el usuario toca el icono de la app en la barra de acciones.

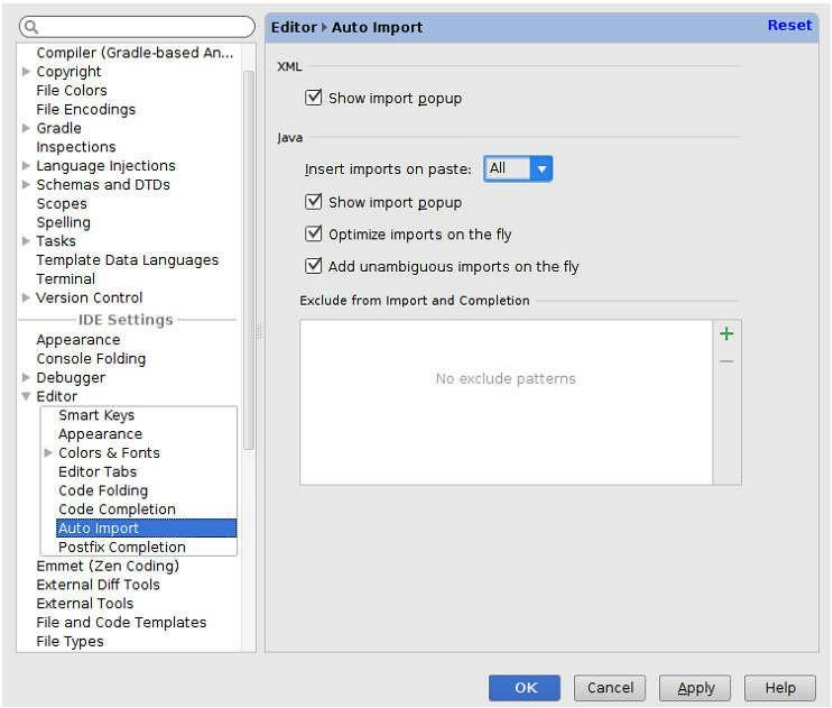
4. Programación avanzada

Cuando se haya creado el esquema de una sencilla aplicación, es el momento de añadir funcionalidades avanzadas, como una interfaz de usuario avanzada, el acceso a los datos de la aplicación (mediante bases de datos o proveedores de contenidos) o la persistencia de los datos de la misma. Asimismo, cuando piensa que tiene la aplicación suficientemente desarrollada, es el momento de publicar la aplicación.

4.1. Importación en Android Studio

Para facilitar la programación con el Android Studio modificaremos las preferencias para automatizar las *importaciones a* medida que vamos escribiendo o que pegamos código, así obtendremos una mayor agilidad a la hora de escribir código.

Accederemos a las preferencias desde *File/Settings* y en el submenú *Editor/General/Auto Import* tendremos que marcar los *checkboxes Optimize importes on the fly* y *Add unambiguous*



importes on the fly. Además, podemos modificar el valor de *Insert imports* donde parte de *Ask All* (véase la figura).

Con estas modificaciones el Android Studio hará *todos* los imports automáticamente, es por eso que siempre tendremos que comprobar que el import sea el correcto para evitar comportamientosno deseados.

4.2. Persistencia

Es probable que desee que su aplicación pueda guardar algunos datos entre las diferentes ejecuciones de la aplicación. Por ejemplo, puede querer guardar las preferencias de la aplicación para que la próxima vez que lo ejecute tenga la misma apariencia que la última vez que se ejecutó. Existen diferentes formas de obtener persistencia en los datosde la aplicación en diferentes ejecuciones:

- Un mecanismo ligero llamado preferencias compartidas (shared preferences) para guardar pequeñas cantidades de datos.
- El sistema de archivos tradicional.
- Una base de datos relacional SQLite.

Para guardar **poca información**, la mejor forma de hacerlo es con las preferencias compartidas. Android incorpora el objeto **SharedPreferences**, que sirve para guardar y leer datos persistentes en la forma clave-valor de datos primitivos. Puede utilizar SharedPreferences para guardar cualquier tipo de datos primitivos: *boolean, float, int,* long y *string.* Estos datos se mantendrán entre sesiones, aunque su aplicación se haya cerrado. Además, serán guardadas automáticamente en un fichero XML.

Para obtener un objectSharedPreferences a su aplicación puede utilizar el método getSharedPreferences() con dos argumentos: el nombre delfichero de preferencias y el modo de operación.

Por ejemplo:

```
1 private SharedPreferences prefs;
2 // Obtener el objeto SharedPreferences
3 prefs = getSharedPreferences ("FicheroPreferences", MODE_PRIVATE);
```

En caso de que desee utilizar un único archivo de preferencias, puede llamar al método getPreferences (), donde no es necesario especificarel nombre del archivo.

Para **escribir valores** en el archivo de preferencias:

1. Llamar al método edit() para obtener un objeto SharedPreferences.Editor.
2. Agregue valores con los métodos que le permiten escribir valores primitivos, como putBoolean () o putString (). Estos métodos tienen dos argumentos. El primero es un *string* que define la clave, y el segundo es el valor que se quiere guardar.
3. Confirme los valores con commit().

Por ejemplo:

```
1 SharedPreferences.Editor editor = prefs.edit ();
2
3 editor.putInt ("Edad", 23);
4 editor.putString ("NombreUsuario”, "Fidel");
```

Para leer los valores de preferencias puede utilizar los métodos análogos getBoolean (), getString () o getInt () de la clase SharedPreferences.

Estos métodos tienen dos argumentos: el primero es el nombre de la clave que está buscando en el archivo de preferencias. El segundo es el valor por defecto que se utilizará en caso de que no se encuentre la clave en el archivo de preferencias. Por ejemplo:

```
1// Cargar las preferencias
2 SharedPreferences prefs = getSharedPreferences
        ("FicheroConfiguracion", MODO_PRIVATE);
3
4 int Edad = prefs.getInt ("Edad", 18);
5 String nombre = prefs.getString ("NombreUsuario", "Usuario");
```

El lugar más adecuado para cargar y guardar las preferencias de la aplicación serían los métodos onCreate () y onStop () (el momento enque la aplicación se pone en marcha y el momento en que se cierra).

4.3. Trabajando con bases de datos

Android proporciona un sistema de bases de datos relacionales basado en **SQLite** que puede utilizar en sus aplicaciones. Cuando la cantidad de datos a guardar es importante, o bien se quieren realizar búsquedas sobre los datos, o la información está relacionada entre sí, es más adecuado tenerla estructurada en forma de una base de datos.

Por ejemplo, podría tener una base de datos con información de diferentes fabricantes relacionada con los diferentes productos que estos producen. Usando bases de datos se puede asegurar la integridad de los datos especificando relaciones entre los diferentes conjuntos de datos.

La base de datos que puede crear para su aplicación únicamente estará disponible para la propia aplicación. El resto de aplicaciones del dispositivo no podrán acceder, por lo tanto **no puede utilizar la base de datos para compartir datos entre aplicaciones.** Trabajar con bases de datos Android puede ser complicado. Por este motivo, debe de crear una clase que servirá para encapsular el acceso a las bases de datos y, así, simplificar el código de su aplicación (que tendrá un acceso a los datos transparente a su implementación, a través de esta clase).

Decimos que una aplicación tiene un acceso transparente a los datos cuando podemos acceder a la información a través de métodos sin tener que conocer su implementación. Esta manera de trabajar nos permite modificar la estructura interna de la información sin que las aplicaciones que la usan deban modificar su código para funcionar correctamente.

4.3.1. Clase DBInterface

Crea un nuevo proyecto con los siguientes datos:

- **Application name:** BasesDeDatos

Deja el resto de opciones con los valores por defecto.
Ahora se creará una clase que servirá de interfaz con el acceso a las bases de datos de la aplicación. Esta clase tendrá el nombre DBInterface. Crea una aplicación, y dentro de esta, crea una nueva clase haciendo clic con el botón derecho dentro del paquete de su proyecto y seleccionando *New/Class,* esta clase creará, abrirá, utilizará y cerrará una base de datos SQLite.

Se creará una base de datos llamada BDClients que contendrá una única tabla, *contactos.* Esta tabla tendrá únicamente tres campos: _id,nombre y email, tal como se puede ver en la tabla:

_id	nom	email
1	John	jcoltrane@atlantic.com
2	Miles	mdavis@bluenote.com

En primer lugar, hay que definir una serie de constantes de texto que servirán para establecer algunos identificadores y campos (y así no será necesario tener que repetir por todo el código, con el peligro deequivocarse en algún momento).

```
public class DBInterface {
    // Constantes
    public static final String CAMPO_ID = "_id";
    public static final String CAMPO_NOMBRE = "nombre";
    public static final String CAMPO_EMAIL = "email";
    public static final String TAG = "DBInterface";

    public static final String BD_NOMBRE = "BDContactos";
    public static final String BD_TABLA = "contactos";
    public static final int VERSION = 1;

    public static final String BD_CREATE =
            "create table " + BD_TABLA + "(" + CAMPO_ID + " integer primary
key autoincrement, "+ CAMPO_NOMBRE + " text not null," + CAMPO_EMAIL + " text
not null); ";

    private final Context contexto;
    private AyudaDB ayuda;
    private SQLiteDatabase bd;
```

Concretamente, la constante BD_CREATE contiene la cadena que se utilizará para crear la tabla *contactos* dentro de la base de datos.

La clase AyudaBD, la creará más tarde. El constructor de nuestra clase, lo único que hace es crear un objeto AyudaBD y guardar en una variable el contexto en que se está ejecutando la clase:

```
public DBInterface (Context con)
{
    this.contexto = con;
    Log.w(TAG, "creando ayuda" );
    ayuda = new AyudaDB(contexto);
}
```

Context es una clase implementada por el sistema Android que da acceso a recursos y clases específicos de la aplicación.

En Android existe clase SQLiteOpenHelper, que es una clase que sirve de ayuda para gestionar la creación de bases de datos y gestión de versiones.

Creará la clase AyudaBD que hereda de ésta:

```
public class AyudaDB extends SQLiteOpenHelper {

    public AyudaDB(Context con){
        super (con, BD_NOMBRE, null, VERSION);
        Log.w(TAG, "constructor de ayuda");
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            Log.w(TAG, "creando la base de datos "+BD_CREATE );
            db.execSQL(BD_CREATE);
        } catch (SQLException e) {
            e.printStackTrace ();
        }
    }

    @Override
    public void onUpgrade (SQLiteDatabase db,
                           int VersionAntigua, int VersionNueva) {
        Log.w(TAG, "Actualizando Base de datos de la versión" +
                VersionAntigua + "A" + VersionNueva + ". Destruirá todos los
datos");
        db.execSQL("DROP TABLE IF EXISTS" + BD_TABLA);

        onCreate(db);
    }
}
```

El constructor de AyudaDB llama al constructor de SQLiteOpenHelper, el cual crea un objeto de ayuda para crear, abrir y gestionar la base dedatos. Vea la documentación de SQLiteOpenHelper para obtener ayudasobre sus métodos.

Esta clase se ocupa de abrir la base de datos si esta existe o crearla encaso contrario, y actualizarla si es necesario. El método onCreate () crea una nueva base de datos. El método onUpgrade () es llamado cuando se ha de actualizar la base de datos. Lo que hace es eliminarla(hacer un *drop de* la tabla) y volverla a crear.

De vuelta a la clase DBInterface, el siguiente paso es definir los diferentes métodos para abrir y cerrar la base de datos.

```
public DBInterface abre () throws SQLException {
    Log.w(TAG, "abrimos base de datos" );
    bd = ayuda.getWritableDatabase();
    return this;
}

// Cierra la BD
public void cierra () {
    ayuda.close();
}
```

El método getWritableDatabase () crea y / o abre una base de datos. La primera vez que se llama se abre la base de datos y se llama a onCreate(). Aquí es donde se crea la base de datos, llamando a execSQL () con la cadena de creación de la base de datos. Una vez creada, la base dedatos queda en *caché,* y por tanto se puede llamar este método para abrirla.

A continuación, define los métodos para modificar la base de datos. Para insertar un contacto se utilizará el método insert (String table, String nullColumnHack, ContentValues values). Los argumentos son estos:

- **String table**: tabla donde se quiere insertar un elemento.
- **String nullColumnHack**: un argumento opcional que dejaremos a *null.* Puede consultar *la* Guía *del desarrollador* de Android para consultar su uso.
- **ContentValues values**: un objeto de la clase ContentValues que sirve para almacenar valores que pueden ser procesados porun ContentResolver.

Este método devuelve el ID de la fila que se ha insertado o un -1 si ha habido un error. Nuestro método para insertar un contacto creará un ContentValues con los valores de la fila insertar y hará la llamada a insert ().

```
public long insertarContacto(String nombre, String email)
{
    ContentValues initialValues = new ContentValues ();
    initialValues.put(CAMPO_NOMBRE, nombre);
    initialValues.put(CAMPO_EMAIL, email);
    return bd.insert(BD_TABLA, null, initialValues);
}
```

Para borrar un elemento de la tabla utilizará el método delete (String table, String whereClause, String [] whereArgs). El significado de los argumentos es el siguiente:

- **String table**: tabla de la que se borrará el registro.
- **String whereClause:** la cláusula WHERE que se aplicará para borrar de la base de datos. Si se le pasa *null,* borrará todas las filas de la tabla.
- **String [] whereArgs:** argumentos de la cláusula WHERE. Este método devuelve la cantidad de filas afectadas por la cláusula WHERE.

```
public long borrarContacto(long id)
{
    return bd.delete(BD_TABLA, CAMPO_ID + "=" + id, null);
}
```

Su método devolverá un valor booleano que indica si se ha borrado algún elemento o no. Para devolver un contacto utilizará el método query (), que tiene los siguientes argumentos:

- **boolean distinct:** será *true* si desea que cada fila sea única, o *false* en caso contrario.
- **String table:** define la tabla respecto a la que desea ejecutar la sentencia de *query.*
- **String [] columns:** admite una lista de las columnas de la tabla que devolverá el método.
- **String selection:** establece un filtro que define qué filas devolver, con el formato de cláusula de SQL, WHERE (sin incluirla palabra WHERE en el *string).*

- **String [] selectionArgs:** permite añadir los argumentos de la selección, si no les ha introducido directamente en la cadena.
- **String groupBy:** establece un filtro que define cómo se agrupanlas filas. Tiene el mismo formato que la cláusula de SQL: GROUP BY (sin incluir las palabras GROUP BY). Si se le pasa un *null,* las filas que se devuelvan no estarán agrupadas.
- **String having:** establece un filtro que declara qué grupos de filas incluye el cursor. Tiene el mismo formato que la cláusula deSQL: HAVING (sin incluir la palabra HAVING). Si se le pasaun *null,* se incluirán todos los grupos.
- **String orderby:** indica cómo ordenar las filas. Tiene el mismo formato que la cláusula de SQL: ORDER BY (sin incluir las palabras ORDER BY). Si se le pasa un *null* devuelve las filas conel orden predeterminado.
- **String límite:** especifica el límite de filas devueltas por *query,* con el formato de la cláusula de SQL: LIMIT. Si se le pasa un *null,* no existe límite.

Este método devuelve un objeto de la clase **Cursor**, que proporciona acceso de lectura y escritura en el resultado devuelto por una consulta(un *query)* en la base de datos.

```
// Devuelve los datos de un contacto a través de su id
public Cursor obtenerContacto(Long id){
    return bd.query(BD_TABLA, new String[]
            {CAMPO_NOMBRE, CAMPO_EMAIL}, CAMPO_ID + "=" + id , null, null,
null,null);
}
```

Para obtener todos los contactos, utiliza otra versión de *query* que noincluya el primer booleano.

```
// Devuelve todos los Contactos
public Cursor obtenerContactos(){
    return bd.query(BD_TABLA, new String []
            { CAMPO_ID,CAMPO_NOMBRE, CAMPO_EMAIL}, null,null, null, null,
null);
}
```

Ten en cuenta que Android utiliza un objeto de la clase Cursor por valor de retorno de las consultas a la base de datos(las *queries).* **Considera el Cursor como un apuntador al conjunto de resultados obtenidos de la consulta en la base de datos**. El uso de un Cursor permite Android gestionar de una forma más eficiente las filas y las columnas.

Finalmente, para actualizar un registro de la tabla utilizará el método Update () con los siguientes argumentos:

- **String table:** establece la tabla a actualizar.
- **ContentValues values:** introduce un objeto de la clase ContentValues con el valor de las columnas a actualizar.
- **String whereClause**: incluye la cláusula WHERE opcional que se aplicará cuando se hace la actualización. Si se le pasa *null* actualizará todas las filas.
- **String [] whereArgs**: establece los argumentos del WHERE.

```
public long modificaContacto(long id,String nombre, String email)
{
    ContentValues newValues = new ContentValues ();
    newValues.put(CAMPO_NOMBRE, nombre);
    newValues.put(CAMPO_EMAIL, email);
    return bd.update(BD_TABLA, newValues, CAMPO_ID + "=" + id, null);
}
```

4.3.2. Utilizando la clase "DBInterface"

Cuando haya creado la clase que le servirá de ayuda para trabajar con las bases de datos, es el momento de usarla. El siguiente ejemplo muestra una aplicación que utiliza la clase de interfaz de base de datos. Es muy sencilla, pero sirve para ilustrar la forma de uso de esta clase.

La actividad principal tiene un *layout* con cinco botones, uno para cada opción de la aplicación:

- Añadir
- Obtener
- Obtener Todos
- Actualizar
- Borrar

La aplicación activará una actividad para cada uno de los botones, equivalente a las diferentes "pantallas" de la aplicación. Estas diferentes actividades deben estar definidas en el AndroidManifest.xml.

```
<activity
    android:name=".Activity_anadir"
    android:label="@string/title_activity_activity_anadir" >
</activity>
```

La actividad principal implementa la interfaz OnClickListener para gestionar las acciones que se derivan cuando el usuario pulsa los botones. A continuación, tenéis el código parcial que corresponde a la definición de la clase y de las variables que usaremos a la actividad:

```
public class MainActivity extends AppCompatActivity implements
View.OnClickListener{

    Button btnAñadir, btnObtener, btnObtenerTodos, btnActualizar, btnBorrar;
```

El método onCreate () sencillamente carga el *layout,* crea el objeto de interfaz de la base de datos y crea los *listeners* de los botones.

A continuación, implementaremos cada una de las acciones que corresponden a los botones en **una actividad separada**. Vamos a veralgunas de estas acciones en detalle:

Añadir

En primer lugar, hay que definir el *listener* del botón:

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.btnAñadir:
            startActivity(new Intent(this,Activity_anadir.class));
            break;
…
```

Esto lanzará *un* intento para la actividad que permitirá añadirelementos en la base de datos. Esta actividad tiene un diseño muy sencillo, incorpora tan sólo los *widgets* necesarios para añadir un nuevo elemento, como se puede ver en la figura :

Este es el código de la actividad donde a partir del *listener* del botón *Añadir* se insertan los datos en la base de datos :

```
public class Activity_anadir extends AppCompatActivity {

    EditText nombre;
    EditText email;

    DBInterface dbInterface;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_anadir);

        dbInterface = new DBInterface(this);
        dbInterface.abre();

        nombre = (EditText)findViewById(R.id.txtNombre);
        email = (EditText)findViewById(R.id.txtEmail);
```

```
        Button añadir = (Button)findViewById(R.id.btnAñadir);
        añadir.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if ( dbInterface.insertarContacto(
                        nombre.getText().toString(),
                        email.getText().toString()) == -1) {
                        Toast.makeText(getBaseContext(),"ERROR : en la
inserción", Toast.LENGTH_LONG).show();
                }
                else {
                        Toast.makeText(getBaseContext(),"Se ha insertado
correctamente", Toast.LENGTH_LONG).show();
                }
```

Básicamente, lo que se hace es:

- Abrir la base de datos.
- Insertar un elemento con los datos que obtiene de las cajas de texto. Fijaos cómo comprueba el valor de retorno de insertarContacto() para saber si se ha podido insertar el contactonuevo correctamente o no.
- Cerrar la base de datos.
- Cerrar la actividad.

Obtener

Se compone de una actividad en la que los únicos *widgets* serán una caja de texto y el botón para obtener el contacto, como se puede ver en la figura.

El código de la actividad lo único que se programa es el *listener* delbotón *Obtener.* El código es el siguiente:

```
public class Activity_Obtener extends AppCompatActivity {

    EditText id;
    DBInterface dbInterface;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_obtener);
        dbInterface = new DBInterface(this);
        dbInterface.abre();

        id = (EditText)findViewById(R.id.txtID);


        Button consulta = (Button)findViewById(R.id.btnObtener);
        consulta.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Cursor c;
                c= dbInterface.obtenerContacto(
Long.parseLong(id.getText().toString()));
                if ( c == null || c.getCount()==0  ) {
                    Toast.makeText(getBaseContext(), "ERROR : en la
consulta", Toast.LENGTH_LONG).show();
                }
                else {
                    c.moveToFirst();
                    Toast.makeText(getBaseContext(),"Nombre:"+c.getString(0)
+ "\nEmail:"+c.getString(1), Toast.LENGTH_LONG).show();
                }
                finish();
            }
        });
    }

    @Override
    protected void onStop(){
        super.onStop();
        dbInterface.cierra();
    }
```

Lo que hemos hecho es:

1. Abrir la base de datos
2. Obtener el identificador que está escrito en la caja de texto
3. Llamar la base de datos
4. Comprobar si ha habido algún resultado

5. Mostrar el contacto
6. Cerrar la actividad
7. Cerrar la base de datos

Lo normal sería mostrar el contacto a través de otra actividad diseñada para mostrar contactos, pero para hacer el programa más sencillo hemos optado por mostrar el contacto con un mensaje por pantalla.

Obtener todos

La opción *Obtener todos* mostrará todos los contactos por pantalla, uno a uno. En una aplicación lo más habitual sería introducir los contactos en un ListView o un ContentProvider, pero en este ejemplo queremos lo haremos a través de un toast. Dado que el programa no necesita ninguna otra información extra para mostrar todos los contactos, esta opción no abrirá otra actividad, sino que mostrará los contactos uno a uno por pantalla. Dentro del *listener* del botón obtenemos todos los contactos de la base de datos y hacemos un recorrido con el cursor que se nos retorna para ir mostrando los contactos.

```
//Para visualizar los datos en un Toast
DBInterface dbInterface = new DBInterface(this);
dbInterface.abre();
Cursor c= dbInterface.obtenerContactos();

// Movemos el cursor en la primera posición
if (c == null) {
    Toast.makeText(getBaseContext(), "Tabla vacía",
Toast.LENGTH_LONG).show();
} else {
    String contactos = "";
    if (c.moveToFirst()) {
        do {
            contactos = contactos + "\n Nombre: " + c.getString(0) + " Email:
" + c.getString(1);
            //  Mientras podamos pasar al siguiente contacto
        } while (c.moveToNext());
    }
    Toast.makeText(getBaseContext(), contactos, Toast.LENGTH_LONG).show();
}

dbInterface.cierra();
```

Lo que hemos hecho es:

1. Abrir la base de datos
2. Llamar la BD para obtener todos los contactos
3. Mover el cursor en la primera posición
4. Mostrar los contactos mientras se pueda
5. Cerrar la base de datos
6. Mostrar un mensaje por pantalla cuando se haya terminado de mostrar los contactos.