Tema 3

Cambio de Activities, menús y listas

Índice

1. Introducción 1
2. Uso de varios Activities. La clase Intent. 1
   2.1. Lanzar una Activity 1
   2.2. Enviar datos a una activity secundaria. 2
   2.3. Devolución de un valor por una activity secundaria 3
      2.3.1. Acciones en la activity secundaria 4
      2.3.2. Acciones en la activity inicial 5
   2.4. Intents implícitos 9
3. Menús / Actionbar 11
   3.1. Definir el menú 11
   3.2. Activar el menú 13
   3.3. Gestionar la pulsación de una opción 13
   3.4. Cambiar las propiedades de ActionBar 14
      3.4.1. Título y subtitulo de la activity 14
      3.4.2. Añadir un icono al ActionBar 14
      3.4.3. Botón volver 15
      3.4.4. Ocultar y mostrar la ActionBar 16
      3.4.5. Personalizar los elementos de un menu 17
   3.5. Toolbar 19
4. AdapterViews 20
   4.1. ListView 20
   4.2. ListView con selección 23
   4.3. RecyclerView 24
      4.3.1. Creación de un RecyclerView 25
      4.3.2. Elementos opcionales 33
      4.3.3. Pulsación de un elemento del RecyclerView 35
      4.3.4. Cambios en el contenido del RecyclerView 36
5. Spinner 39
6. Menú contextual 41
7. Menús emergentes 43
8. Apéndice 1: Pulsación alternativa de un elemento del RecyclerView 46
9. Apéndice 2: ListView con varios datos por entrada. Adapter a medida. .. 48
10. Apéndice 3: Uso de Intents implícitos 53
11. Bibliografía 54

1. Introducción

A lo largo de este tema veremos cómo navegar entre Activities así como otros elementos que se podrán incluir en la interfaz de usuario.

Como se comento en el tema pasado, no se pueden ver una por una todas las características de cada elemento por lo que es parte del aprendizaje el saber manejarse con la documentación de las distintas clases usadas.

También es interesante consultar, en la documentación de Android, la sección sobre Marerial Design donde se pueden consultar recomendaciones de cómo y cuándo usar los distintos componentes: https://material.io/design/

2. Uso de varios Activities. La clase Intent.

Un Intent es un elemento de comunicación asíncrono entre componentes de una aplicación Android. Es una clase que sirve, entre otras cosas, para interaccionar con el sistema Android. Con un Intent una activity puede pedir que se le dé un contacto, que saque una foto, realizar una llamada, ...

Además, que es lo que se va a tratar en este punto, se puede usar para lanzar nuevas activities pasándole cierta información y recoger resultados producidos por ellas. Se puede ver como un objeto para comunicar activities en tiempo de ejecución.

Es una clase realmente compleja por lo que, a lo largo del curso, se irá viendo distintos usos según se vayan usando.
En este punto se verán siguientes elementos:

- Lanzar una activity secundaria.
- Enviar datos a una activity secundaria.
- Recoger datos que una activity secundaria envía de vuelta.
- Lanzar Intents implícitos (Android decide que activity va a lanzar).

Si queremos ver una descripción más completa de los Intents se puede consultar los siguientes enlaces:

- https://developer.android.com/guide/components/intents-filters.html?hl=es
- https://developer.android.com/reference/android/content/Intent.html?hl=es

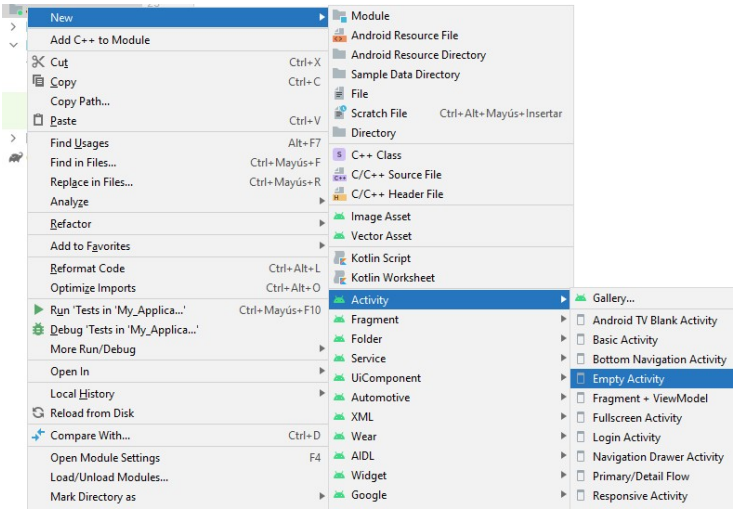
2.1.Lanzar una Activity

En este punto se verá como desde la activity actual pulsando un botón, podría ser con cualquier otra acción, se lance una nueva activity. En este punto solo se lanzará la activity sin pasarle datos ni recoger los datos que pueda enviar de vuelta.

Lo primero que hay que hacer es crear la activity que se va a lanzar. Es recomendable crear las activities mediante el asistente de Android Studio ya que así la activity se añade automáticamente al archivo AndroidManifest.xml. En caso de crearla de forma manual la tendríamos que añadir nosotros.

Para usar el asistente pulsaremos con el botón derecho sobre la estructura del proyecto (o desde el menú file) y se escoge la opción:

New → Activity → Empty Activity



A esta nueva activity se le debe introducir un nombre, en nuestro ejemplo se llamara Secundaria.

Gestionaremos la pulsación de un botón en la primera activity, de cualquiera de las formas vistas en el tema pasado, añadiéndole el siguiente código:

```
Intent intent=new Intent(MainActivity.this, Secundaria.class);
startActivity(intent);
```

Donde:

- En la primera línea se crea un Intent con uno de sus múltiples constructores. El que nos interesa se utiliza para lanzar una activity al que se le pasa:
  1. El contexto de la activity actual. Si estamos en una clase inner (OnClickListener) es necesario usar el identificador MainActivity antes del this.

     Se puede usar también el método getApplicationContext() en lugar del this. Ambos son válidos.

     El contexto es un objeto que mantiene información del entorno de funcionamiento de la aplicación: clases usadas, activities, recursos, ...
  2. La clase de la activity que queremos lanzar. En nuestro ejemplo es Secundaria.class.
- Para lanzar una activity hay que llamar al método startActivity usando como parámetro el intent que acabamos de crear.

En este punto si se ejecuta el programa se debería lanzar, al pulsar el botón, la segunda activity

2.2.Enviar datos a una activity secundaria.

Para enviar datos de una activity a otra se utiliza el mimo intent definido en el punto anterior. A este se le puede añadir una colección de pares clave-valor denominada Bundle. El intent, con los dados que posee en esta colección, es lo que se le pasa a la activity lanzada.

Seguiremos con el ejemplo anterior pero en este caso la activity principal, al pulsar el botón, enviará una cadena a la activity secundaria y esta visualizara este valor en un campo de texto.

Para ello seguiremos los siguientes pasos:

- Añadimos un editText en la activity primaria. El valor de este editText es el que se enviará a la secundaria al pulsar el botón.
- Añadimos un TextView a la activity secundaria que se encargará de visualizar el valor que se envía desde la activity primaria.
- Modificaremos el código del apartado anterior para añadirle el dato al intent.

```
Intent intent=new Intent(MainActivity.this, Secundaria.class);
// Se añade al intent el dato con clave NOMBRE y valor el contenido del editText
intent.putExtra( "NOMBRE" , editNombre.getText().toString() );
startActivity(intent);
```

El método putExtra se encarga de añadir el valor (el parámetro de la derecha) al intent con una clave (parte izquierda) de tipo cadena sensible al caso.

- En la activity secundaria, se recupera el intent enviado mediante el método getIntent. Una vez obtenido el intent se recupera el dato o datos que interesen con el método get que se corresponda con el tipo de dato que se quiera recuperar: getFloatExtra, getStringExtra, getIntExtra, ...

```
@Override  // Método onCreate de la activity secundaria
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_secundaria);

      // Recuperamos el intent enviado desde la activity primaria
      Intent intent=getIntent();

      // TextView de la activity secundaria
      final TextView nombre=findViewById(R.id.textViewMensaje);

       // Se establece el valor del TextView con el dato pasado en el intent. En
       // este caso NOMBRE
      nombre.setText("Hola "+ intent.getStringExtra("NOMBRE"));
}
```

2.3.Devolución de un valor por una activity secundaria

Ahora veremos como una activity secundaria puede devolver algún valor a la activity que la ha lanzado. Para eso dividiremos el proceso en dos partes:

- Acciones que realizaremos en la activity secundaria para devolver datos.
- Acciones a realizar en la activity inicial:
  - Como tenemos que lanzar la activity secundaria para que esta pueda devolver datos.
  - Pasos tenernos que realizar para capturar los datos devueltos por la activity secundaria.

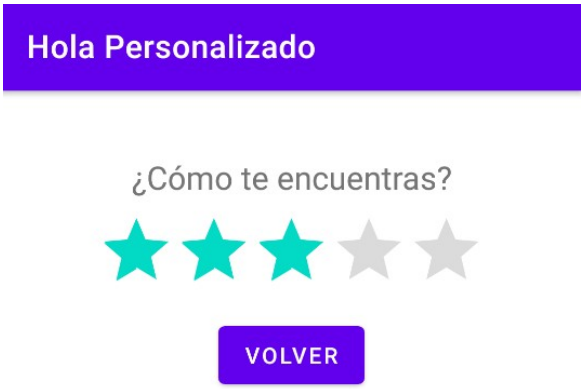
2.3.1.Acciones en la activity secundaria

Los pasos que deberemos realizar en la activity secundaria son:

1. Crear un Intent vacio con el que se devolverán los datos.
2. Introducir en el intent, mediante sus métodos put los datos que se quieran devolver. Pare recuperar el valor deberemos luego usar el método get que se corresponda con el tipo de dato enviado.
3. Establecer el tipo de resultado que se produce mediante setResult. Los posibles valores son:
   - RESULT_OK: el resultado es correcto.
   - RESULT_CANCELED: se cancela la obtención de un resultado.
   - O un valor personalizado definido por nosotros (no lo usaremos).
4. Finalizar la activity con finish. Finish provoca que se finalice la activity actual, en este caso secundaria, y volviendo a la activada desde que se lanzo ejecutándose, de forma automática, al método encargado de recupera el resultado de la actividad principal.

Para comprobar cómo funciona la devolución de valores añadiremos a la activity secundaria los siguientes elementos:

- Un textview explicativo con la frase "¿Cómo te encuentras?".
- Un componente tipo RatingBar de 5 estrellas con 3 marcadas por defecto y un tamaño de selección de 0.5.
- Un botón con la palabra Volver.



Se añade en el método onCreate el evento de respuesta del botón:

```
final Button btnVolver = findViewById(R.id.btnVolver);
final RatingBar ratingBar = findViewById(R.id.ratingBar);

btnVolver.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        1 Intent intentSec=new Intent();
        2 intentSec.putExtra("numEstrellas", ratingBar.getRating()); // float
        3 setResult(RESULT_OK, intentSec);
        4 finish();
    }
});
```

2.3.2.Acciones en la activity inicial

En este apartado veremos las acciones que tenemos que llevar a cabo para lanzar la activity secundaria de forma que esta pueda devolver datos y una vez que esta devuelva los datos poder obtenerlos para trabajar con ellos.

En este punto tenemos dos opciones:

- Usar la nueva la API de Activity Result[1] que se introdujo en AndroidX. Esta opción es la actualmente recomendada por Android.
- Usar startActivityForResult() y onActivityResult() tal y como se venía haciendo antes de AndroidX.

Usando la API de Activity Result

Puede ocurrir que cuando se lanza una actividad para obtener un resultado, ya sea esta una activity de tu propia aplicación u otra aplicación diferente (como puede ser la cámara) tu aplicación se destruya (por ejemplo por motivos de memoria) reconstruyéndose luego.

Por este motivo la API de Activity Result permite separar el lugar desde donde se inicia una actividad del lugar donde se registra la devolución de la llamada, con los resultados, de esta nueva activity. Esto es así ya que la devolución de la llamada debe registrarse siempre cada vez que se crea o recrea la activity inicial.

El ejemplo siguiente muestra el código que permite registrar la devolución de la llamada a una actividad:

```
1  ActivityResultLauncher<Intent> launcher=registerForActivityResult(new
2      ActivityResultContracts.StartActivityForResult(), new ActivityResultCallback<ActivityResult>() {
3      @Override
4      public void onActivityResult(ActivityResult result) {
5          if (result.getResultCode()==RESULT_OK){
6              Intent intent=result.getData();
7              int numEstrellas=(int)intent.getFloatExtra("numEstrellas", 0f);
8              switch(numEstrellas){
9                  case 0: Log.i(TAG, "Por los suelos");  break;
10                 case 1: Log.i(TAG, "De bajón");        break;
11                 case 2: Log.i(TAG, "Triste");          break;
12                 case 3: Log.i(TAG, "Normal");          break;
13                 case 4: Log.i(TAG, "Contento");        break;
14                 case 5: Log.i(TAG, "Feliz");           break;
15             }
           }
       }
   });
```

Donde:

[1] https://developer.android.com/training/basics/intents/result#java

- Línea 1: lanzaActivity es un objeto del tipo ActivityResultLauncher que permite lanzar una actividad. Su definición es ActivityResultLauncher<T> donde T es tipo de entrada necesario para lanzar otra actividad. Usaremos su método launch, que toma como parámetro un tipo de dato T, para lanzar la actividad.
- Línea 1: registerForActivityResult[2] permite para registrar la devolución de llamada de resultados por parte de una actividad. Este método solo registra la devolución de una llamada pero no inicia la otra actividad. Mediante este sistema evitamos tener que usar un parámetro requestCode para identificar de que actividad provienen los datos como tendremos que hacer en la versión que veremos a continuación de esta.
- Línea 2: La sobrecarga que usaremos de registerForActivityResult tiene dos parámetros.
  - El primero es de tipo ActivityResultContract. Este tiene dos parámetros: uno de entrada (un intent) y otro de salida (un objeto de tipo ActivityResult). Se puede definir un ActivityResultContract personalizado o usar uno de los disponibles en ActivityResultContracts (en plural). En este caso usaremos StartActivityForResult que está presente en los predefinidos de Android.
  - El Segundo es la interfaz ActivityResultCallback cuyo tipo debe coincidir con el tipo de salida que devuelve ActivityResultContract. Deberemos implementar el método onActivityResult que tiene como parámetro un objeto del tipo indicado antes. Es en este método donde gestionaremos la devolución de valores por la activity que ha sido invocada.
- Línea 2: ActivityResultContracts[3]: una colección de ActivityResultContract predefinidos proporcionados por Android para utilizar en las acciones intent básicas como pueden ser llamar a otra activity, tomar una foto, solicitar permisos, ...
- Línea 4: implementación del método onActivityResult.
- Línea 5: mediante el parámetro result obtenemos los resultados producidos.
- Línea 6: usando el método getData de result se puede obtener el intent enviado de vuelta.
- Línea 7: una vez conseguido el intent podemos obtener sus valores tal y como hemos hecho hasta ahora.

Se pueden definir tantos objetos ActivityResultLauncher como se desean para recuperar valores devueltos desde distintas activities.

Vamos a ver ahora como lanzar una actividad para obtener los valores que devuelva. Para el ejemplo gestionaremos esto dentro del listener de un botón.

[2] registerForActivityResult

[3] ActivityResultContracts

```
1 boton.setOnClickListener(new View.OnClickListener() {
2    @Override
3    public void onClick(View view) {
4       Intent intent=new Intent(MainActivity.this, Secundaria.class);
5       intent.putExtra("valor",111);
6       launcher.launch(intent);
7    }
8 });
```

El único elemento diferente con lo visto anteriormente es la línea 6 en la que usamos el método launch (en vez de startActivity) del objeto lanzaActivity generado anteriormente para lanzar la actividad. Toma como parámetro un objeto del tipo definido en ActivityResultLauncher.

Usando startActivityForResult() y onActivityResult

Para habilitar a que la activity secundaria pueda devolver un dato esta se deberá lanzar con el método startActivityForResult en vez de con startActivity. Con esto indicamos que la activity que se lanza va a devolver algún dato.

```
// este valor es un valor único utilizado para identificar cada activity que se invoque
int MIREQUESTCODE=1;
startActivityForResult(intent, MIREQUESTCODE);
```

Cuenta con dos parámetros:

- El primero es el intent con el que se indica que actividad se va a lanzar (este intent se define de la misma forma que las vistas en el apartado 2.1 cuando veíamos como lazar una activity).
- MIREQUESTCODE se utilizar para identificar la activity que devuelve los datos cuando se ejecute onActivityResult.

En la activity desde donde le lanza la actividad secundaria, MainActivity en nuestro ejemplo, se deberá implementar el método onActivityResult. Este método permite capturar los valores devueltos por una activity que se ha iniciado con startActivityForResult y se ha finalizado con setResult y finish.

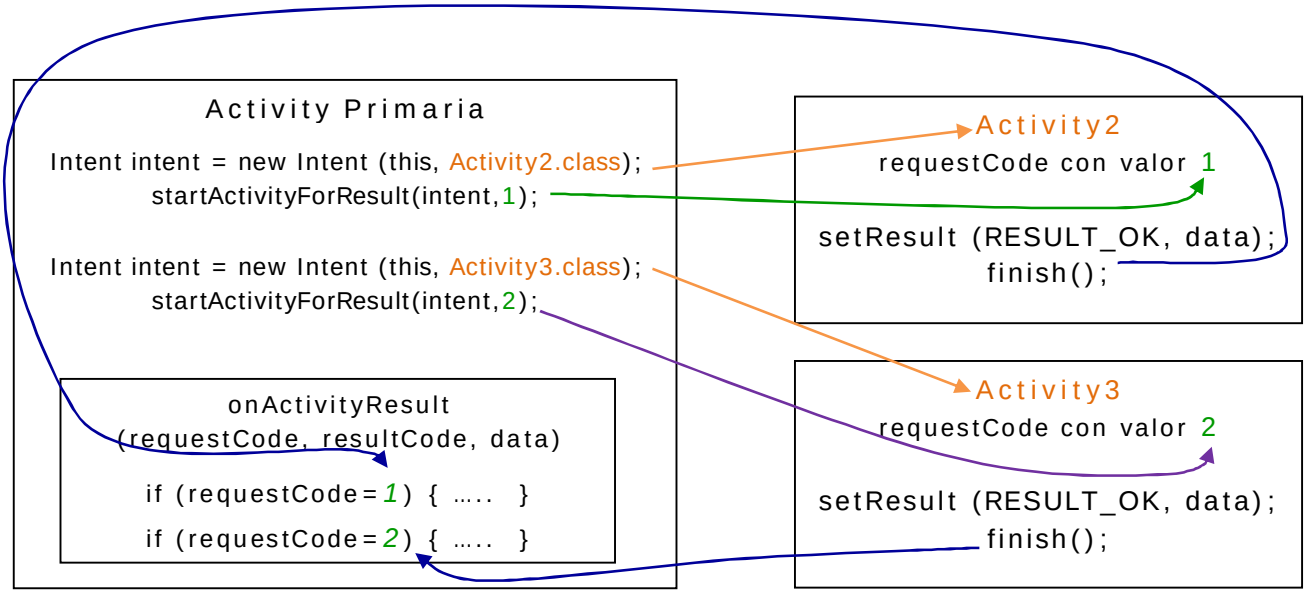
```
@Override
 // data es el intent devuelto por la actividad secundaria
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == MIREQUESTCODE && resultCode == RESULT_OK){
      switch((int)data.getFloatExtra("ESTADO", 0f)){
         case 0:Log.i(TAG, "Por los suelos");  break;
         case 1:Log.i(TAG, "De bajón");        break;
         case 2:Log.i(TAG, "Triste");          break;
         case 3:Log.i(TAG, "Normal");          break;
         case 4:Log.i(TAG, "Contento");       break;
         case 5:Log.i(TAG, "Feliz");           break;
      }
   }
}
```

Los parámetros de este método son:

- requestCode: se utiliza para identificar cual es la activity que está devolviendo el resultado ya que pueden ser varias.
- resultCode se utiliza para comprobar el éxito o no de las operaciones realizadas en la otra activity.
- data: es el intent, devuelto por la activity secundaria, que contiene los datos

Si resultCode es correcto y no se ha cancelado se recoge el dato con alguna variante de getExtra del intent. Para el caso de tipos que no sean simples, en lugar de putExtra se puede usar setData.

El siguiente ejemplo muestra un ejemplo de invocación y de retorno de valores cuando estos proceden de más de una activity.



Ejercicio 1

Amplía el ejemplo del tema pasado para que si se pulsa el imageButton envíe y muestre en una segunda Activity, llamada Secundaria, el nombre introducido en el editText y el valor del RatingBar.

En la segunda activity se visualizará este valor en un ratingBar con un paso de rating de 0.5.

En la activity secundaria añadiremos un botón que al pulsarse devuelve el valor del RatingBar a la activity principal mostrándose en el textView a la derecha del imageButton, en el RatingBar y en el logcat.

Además debe funcionar tanto si se pulsa el botón Back estándar de Android como si se pulsa el botón volver de la actividad secundaria (usa onBackPressed).

2.4. Intents implícitos

Para lanzar nuevas activities vía intent existen dos opciones:

- De forma explícita: indicando la activity a lanzar tal y como se ha hecho en el punto anterior.
- De forma implícita: de forma que a partir de un Intent y a partir de los datos que se lo pasan como parámetro Android deduzca el componente de otra aplicación que va a lanzar.

Veamos esta posibilidad con un ejemplo.

En MainActivity añade otro botón de Posición y el siguiente código asociado a su evento de pulsación:

```
final Button btnPosicion =(Button)findViewById(R.id.button);
btnPosicion.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        Double lat = 42.237109;
        Double lon = -8.723474;
        int zoom = 22;
        String label = "MiPunto";

        //  try {  label=URLEncoder.encode("label","UTF-8");
        //      } catch (UnsupportedEncodingException e){}

        String uri = String.format(Locale.US,
             "geo:%f,%f?z=%d&q=%f,%f(%s)", lat, lon, zoom, lat, lon, label);
        intent.setData(Uri.parse(uri));

        // Si existe una Activity que es capaz de gestionar los datos esta se lanza
        if (intent.resolveActivity(getPackageManager()) != null)
            startActivity(intent);
        }
});
```

Al iniciar el intent no se le indica una clase si no que se indica una acción a realizar. Luego a partir de los datos que se le pasan, Android llama a unos activities predefinidos o a otros.

En este caso la acción es mostrar, ACTION_VIEW, pero podrían ser otros como ACTION_SEND, ACTION_EDIT, IMAGE_CAPTURE, … y los datos se especifican a través de una URI (Uniform Resource Identifier), que es una forma estándar de indicar un protocolo de datos de información como puede ser http:, ftp:, file:, geo:, …

- La lista completa de acciones se puede consultar la siguiente dirección: https://developer.android.com/reference/android/content/Intent.html?hl=es#standard-activity-actions
- Para ver ejemplos y explicación de la utilización de Intents implícitos se puede consultar la página: https://developer.android.com/reference/android/content/Intent.html

- Lista extensa de Intents comunes que se pueden utilizar: https://developer.android.com/guide/components/intents-common?hl=es#java
- Se puede ver la lista completa de tipos de URIs en: http://es.wikipedia.org/wiki/URI_scheme.

En el ejemplo anterior los datos son de tipo geo por lo que Android lanza la activity predefinida para visualizar mapas.

Actividad: Prueba las siguientes llamadas a Intents implícitos.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "¡Hola Mundo!");

if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}

// Se requiere el permiso com.android.alarm.permission.SET_ALARM
 Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)
        .putExtra(AlarmClock.EXTRA_MESSAGE, "¡¡ Hora de levantarse!!")
        .putExtra(AlarmClock.EXTRA_HOUR, 7)
        .putExtra(AlarmClock.EXTRA_MINUTES, 30);
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

Modifica la definición del Intents para probar los siguientes:

```
intent = new Intent("android.media.action.IMAGE_CAPTURE");
intent = new Intent(Intent.ACTION_VIEW, Uri.parse("content://contacts/people/"));
intent = new Intent(Intent.ACTION_EDIT, Uri.parse("content://contacts/people/1"));
```

Además se pueden conseguir datos del intent lanzado de las dos mismas formas que las vistas en el punto 2.3 de este tema.

Para mostrar un ejemplo más completo de su funcionamiento podemos ver un ejemplo en el apéndice 3

Ejercicio 2

Continua con el ejercicio 1 añadiendo el botón Llamar, con su correspondiente imagen, que, al pulsarlo, permita mostrar el teclado de llamada, no llamar, por teléfono con un número que hayamos introducido en un campo de texto.

Si se quisiera que el botón llamase directamente. ¿Qué necesitaríamos hacer? intent.

3. Menús / Actionbar [4] [5] [6]

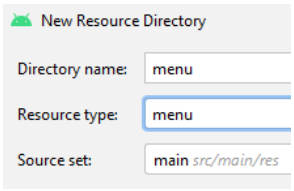
En este punto explicaremos como crear menús para nuestra aplicación. Como otros elementos, se pueden definir tanto mediante un fichero XML como mediante código. Nosotros veremos solo su definición usando ficheros XML.

Para la utilización de los menús deberemos seguir tres pasos:

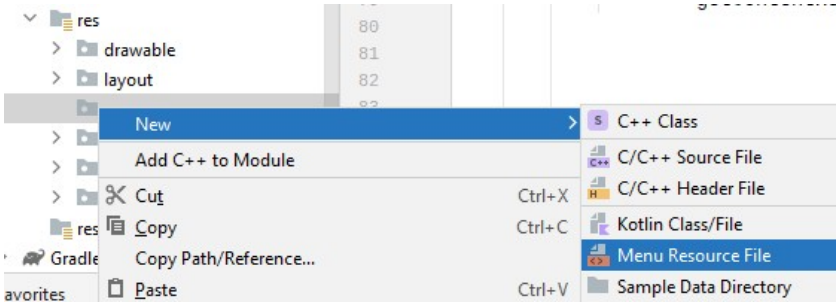
1. Definir el menú en un fichero XML.
2. Expandir el menú en una activity.
3. Gestionar la pulsación de una opción.

3.1. Definir el menú

Los menús en Android se definen como recursos dentro del directorio res/menu. Si el directorio no existe en nuestro proyecto deberemos crearlo mediante new → Android resource directory y escoger como tipo de recurso menú. Dejaremos el nombre del directorio por defecto.



Una vez creado el directorio definiremos dentro de de él un nuevo fichero de recursos de menú: Menu resource file.



Una vez creado el fichero XML asociado al menú podremos editarlo en el propio fichero XML o mediante el asistente gráfico tal y como se hace con los layouts.

Básicamente la estructura de un menú es la siguiente: Tenemos un elemento <menu> del que cuelgan los <item>, que son cada opción que posee el menú. Cada item puede tener una serie de propiedades e incluso otros menús, que serian los submenús.

Sus principales propiedades son:

- title: es el texto que se mostrará en cada opción.
- icon: permite asociar una imagen a la opción.
- checkable: se muestra un checkbox seleccionable.

[4] https://developer.android.com/guide/topics/ui/menus
[5] https://developer.android.com/training/appbar/
[6] https://developer.android.com/guide/topics/resources/menu-resource

- showAsAction: indica como el elemento ítem creado se muestra en la ActionBar (menú superior). Nos interesan las siguientes propiedades:
  - always: la opción del menú se muestra siempre en la actionBar en vez del menú desplegable (menú que se muestra al pulsar el icono de los tres puntos).
  - never: la opción del menú se muestra siempre en el menú desplegable.
  - ifRoom: la opción del menú se muestra en la actionBar si tiene sitio, sino se muestra en el menú. Si hay más elementos de menú que sitio disponible en la actionBar la prioridad de la opción de menú a mostrar en la actionBar se especifica mediante la propiedad orderInCategory.
  - withText: si la opción se muestra en la actionBar se muestra también con el título (solo en horizontal).

La definición del menú que se corresponde con la página anterior es:

```
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id="@+id/meditar"
      android:icon="@drawable/ic_edit"
      android:title="Editar"
      app:showAsAction="always" />
   <item
      android:id="@+id/mborrar"
      android:icon="@drawable/ic_add"
      android:title="Borrar"
      app:showAsAction="always" />
   <item
      android:id="@+id/motro"
      android:title="Otro">
      <menu>
         <item
            android:id="@+id/msubmenu"
            android:title="Submenu" />
      </menu>
   </item>
</menu>
```

Para que la opción showAsAction funcione se deberá, si ya no está, añadir la siguiente línea a la declaración del menú en el fichero XML:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

Y modificar la opción showAsAction de android:showAsAction a app:showAsAction.

Se puede ver una lista de iconos que ofrece Material Design para usarse como iconos en:

https://material.io/tools/icons/?style=baseline

3.2.Activar el menú

Para que una activity pueda mostrar el menú creado tendremos que sobrecargar el método onCreateOptionsMenu y expandir el menú (convertir el fichero XML en código java) tomando como referencia el id del fichero (que coincide con el nombre del fichero xml con el menú) que contiene el menú.

Si el fichero del menú creado se llama menu_principal el código quedaría:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater menuInflater=getMenuInflater();
    menuInflater.inflate(R.menu.menu_principal,menu);
    return true;
}                                          // nombre del menú creado
```

3.3.Gestionar la pulsación de una opción

Para gestionar la pulsación de una opción de menú tenemos dos opciones:

- Se sobrecarga el método onOptionsItemSelected en la activity que muestra el menú, que según la opción escogida realizara una acción u otra. En nuestro caso solo se mostrará un Toast informativo.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.meditar:
         Toast.makeText(this, "Editar", Toast.LENGTH_SHORT).show();
         return true;

      case R.id.mborrar:
         Toast.makeText(this, "Borrar", Toast.LENGTH_SHORT).show();
         return true;

      case R.id.motro:
         Toast.makeText(this, "Otro", Toast.LENGTH_SHORT).show();
         return true;

      case R.id.msubmenu:
         Toast.makeText(this, "Submenú 1", Toast.LENGTH_SHORT).show();
         return true;

      default:
         return super.onOptionsItemSelected(item);
   }
}
```

- Especificamos el nombre de un método en la propiedad onClick de un ítem. Este método tiene se tiene que implementar en la activity que visualiza el menú no puede devolver nada y toma como parámetro una opción de menú. Un ejemplo puede ser el siguiente:

```
public void menuEditar(MenuItem item) {
     Toast.makeText(this, "Editar onClick", Toast.LENGTH_SHORT).show();
}
```

Ahora en en el item de menú identificado por R.id.meditar establecemos el nombre de este método en la propiedad onClick. En el fichero xml quedaría:

```
android:onClick="menuEditar"
```

El tipo de parámetro del método que se usa en onClick varía si es un componente de la interfaz (View) o si es una opción de menú (MenuItem).

Ejercicio 3:

Continua con el ejercicio 1 implementando, en la aplicación, un menú con las siguientes opciones (cada una de ellas con su correspondiente icono).

- Nuevo: se muestra solo en la actionBar con texto. Se lanza un nueva activity Terciaria en la que en el log se muestran el checkBox seleccionado.
- Borrar: se muestra, en la actionBar, el texto y el icono solo si hay espacio. Se vacían todos los TextViews y se pone a cero el SeekBar.
- Editar: solo se muestra solo en el menú desplegable. Se vacía el editText del nombre.
- Sub: tiene un submenú con dos opciones: opc1 y opc2 donde cada una de ellas deberá mostrar un toast indicado que ha sido pulsado.

3.4.Cambiar las propiedades de ActionBar

Para modificar las propiedades del ActionBar de un Activity tenemos que, primero, obtener el ActionBar de la activity:

```
ActionBar actionBar = getSupportActionBar();
```

Para posteriormente utilizar este objeto para modificar sus propiedades.

3.4.1.Título y subtitulo de la activity

Para cambiar el título y añadir/modificar el subtítulo de un Activity podemos usar:

```
actionBar.setTitle("Título");
actionBar.setSubtitle("Subtítulo");
```

Podemos ocultar el título y su subtitulo con:

```
actionBar.setDisplayShowTitleEnabled(false);
```

3.4.2.Añadir un icono al ActionBar

Para añadir un icono a la izquierda del título/subtitulo usaremos el siguiente código:

```
actionBar.setDisplayShowHomeEnabled(true);
actionBar.setDisplayUseLogoEnabled(true);
actionBar.setLogo(android.R.drawable.ic_menu_camera);
```

En estos momentos Material Design no recomienda usar esta opción.

3.4.3.Botón volver

Android permite establecer, en la parte izquierda de la activity, un botón para, en un principio, volver al Activity anterior, aunque podemos establecerle el comportamiento que deseemos.

Se habilita mediante el código:

```
actionBar.setDisplayHomeAsUpEnabled(true);
```

Si se desea cambiar el icono del botón volver se puede usar el método:

```
ab.setHomeAsUpIndicator(id de un recurso o directamente un drawable);
```

Para gestionar la pulsación tenemos dos opciones:

- Tratando este botón como una opción de menú más usando el id de botón: android.R.id.home:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            onBackPressed();
            return true;
        }
     return super.onOptionsItemSelected(item);
}
```

- Añadiendo las siguientes líneas a la definición de la Activity en el archivo AndroidManifests.xml:

```
<activity  <!-- El botón se ha añadido a la actividad secundaria -->
   android:name=".Secundaria"
   android:parentActivityName=".MainActivity"/>
```

Si queremos dar soporte a versione anteriores a Android 4.1 (API level 16) tendríamos que añadir el siguiente elemento meta-data:

```
<activity
   android:name=".Secundaria"
   android:parentActivityName=".MainActivity">
   <meta-data
      android:name="android.support.PARENT_ACTIVITY"
      android:value=".MainActivity" />
</activity>
```

3.4.4.Ocultar y mostrar la ActionBar

Para ocultar la ActionBar se puede usar su método hide mientras que usando show se puede volver a mostrar:

```
actionBar.hide();    // oculta la actionBar

actionBar.show();    // muestra la actionBar
```

Otra opción es evitar que la AcctionBar esté presente. Para ello se puede usar, en el Manifest (API mínima de 11), un tema sin ActionBar, ya sea a nivel de aplicación o de activity. El siguiente ejemplo muestra un tema de los predefinidos de Android sin ActionBar:

```
// si no existiese en el punto siguiente se ve cómo crearlo
android:theme="@style/Theme.AppCompat.NoActionBar"
```

Si el estilo @style/AppTheme.NoActionBar no está presente o se quiere conservar el tema actual se pueden añadir, en el fichero themes.xml, las siguientes líneas al tema actual:

```
<item name="windowNoTitle">true</item>
<item name="windowActionBar">false</item>

<!-- true: hide notificacion bar-->
<item name="android:windowFullscreen">false</item>
```

Ejercicio 4:

Continua con el ejercicio anterior:

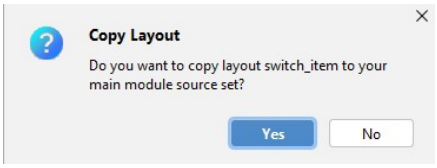
- Un botón que permita ocultar y mostrar, de forma alternativa, la ActionBar.
- En la activity secundaria y terciaria añade y gestiona la pulsación del botón volver en la ActionBar.
- Añade un subtitulo a la ActionBar de la actividad primaria mostrando el valor de la SeekBar. Este valor debe estar siempre actualizado.

3.4.5.Personalizar los elementos de un menu

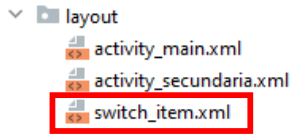
Además de los elementos vistos a un menú se puede añadir distintos componentes de una interfaz definidos en un fichero XML.

Por ejemplo añadiremos un botón Switch y un botón normal al menú. Comenzaremos arrastramos la opción Switch item a nuestro menú.

En este momento Android Studio nos pregunta si queremos crear el layout que contendrá el Switch. En nuestro caso contestaremos que si para que cree el layout con el nombre switch_item.xml en el directorio res/layout.

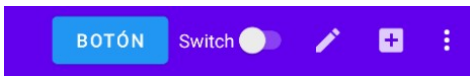


Es en el item del Switch definido en el fichero XML del menú donde se establece, en la propiedad `app:actionLayout="@layout/switch_item"`.



Y es en ese layout donde editaremos las propiedades individuales de los componentes que mostraremos: el ide, el texto que muestra y el color del botón y en el caso del Switch su id y su propiedad text que es el texto que se muestra en el menú a lado del Switch.

No es obligatorio usar el layout creado por Android Studio ya que, en la propiedad `app:actionLayout`, podremos especificar un layout diferente creado por nosotros. Como estos son layouts normales se pueden añadir cualquier componente como botones, toggles buttons, … para que se visualicen al el menú.



El layout definido es:

```
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/LayoutItemmenu"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">

   <Switch
      android:id="@+id/menuswitch"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Switch"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="@+id/menuboton" />
```

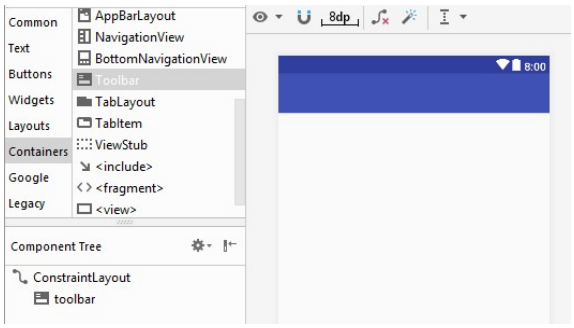
```
    <Button
        android:id="@+id/menuboton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:backgroundTint="#2196F3"
        android:text="Botón"
        app:layout_constraintEnd_toStartOf="@+id/menuswitch"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

La pulsación de los elementos definidos en un menú dentro de su propio layout se gestiona en onCreateOptionsMenu y no en onOptionsItemSelected.

Un ejemplo es el siguiente:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater menuInflater=getMenuInflater();
    menuInflater.inflate(R.menu.menu_principal,menu);

                    // Id del item del menú con el layout que se quiere usar
                                          //id del componente definido en el layout
    Switch switchView = menu.findItem(R.id.app_bar_switch).getActionView().
                                                    findViewById(R.id.menuswitch);

    // Se puede establecer su estado inicial
    switchView.setChecked(true);

    // Se puede cambiar el texto inicial
    switchView.setText("Texto prueba"); mostrado mediante código

    // Se gestiona la pulsación del switch
    switchView.setOnCheckedChangeListener(new
                                    CompoundButton.OnCheckedChangeListener() {
       @Override
       public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
          Toast.makeText(MainActivity.this, "Estado:"+b, Toast.LENGTH_SHORT).show();
          switchView.setText(b?"Activado":"Desactivado");
       }
    });

    Button bu=menu.findItem(R.id.app_bar_switch).getActionView().
                                                    findViewById(R.id.menuboton);

    // Se gestiona la pulsación del switch
    bu.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View view) {
          Toast.makeText(MainActivity.this, "Pulsato", Toast.LENGTH_SHORT).show();
       }
    });

    return true;
}
```

3.5.Toolbar[7]



En las últimas versiones de Android se recomienda, por temas de compatibilidad, no usar la AcctionBar y en vez de ello usar un componente Toolbar. Introduciremos aquí su utilización aunque se verá en más profundidad en el tema siguiente.

Para su utilización se deben de realizar los siguientes pasos:

1. (Solo en versiones antiguas de Android Studio) Usar la librería de compatibilidad V7 appcompat (Android Studio la agrega de forma automática)
2. La actividad donde se desea usar debe heredar de AppCompatActivity.
3. Configurar la Activity para que no use la AcctionBar que se añade por defecto indicando, en el archivo AndroidManifest, que se haga uso de un estilo sin AcctionBar como hemos visto en el punto anterior. Existen dos posibilidades:
   - Si solo alguna Activity va a usar Toolbar se puede indicar en la definición de la Activity en particular.
   - Se puede indicar en la definición de la aplicación si todas las Activities van hacer uso de Toolbar.
4. Se añade un componente Toolbar en la parte superior de la Actividad.
   Una propiedad interesante es android:elevation que indica la altura que tiene Toolbar sobre el componente inferior. Material design recomienda 4dp.
5. En onCreate establecemos la barra de herramientas creada como barra de herramientas de la actividad.

```
Toolbar barraDeHerramientas= findViewById(R.id.toolbar);
setSupportActionBar(barraDeHerramientas);
```

   Hay que tener especial cuidado con los imports. Puesto que se debe importar el componente androidx.appcompat.widget.Toolbar y no android.widget.Toolbar.

6. Para trabajar con la barra de herramientas se puede conseguir la ActionBar y usar los métodos y funciones indicados en puntos anteriores:

```
ActionBar ab = getSupportActionBar();
```

Para no tener que redefinir en cada Activity la Toolbar se puede definir un fichero layout que solo contenga la barra de herramientas y luego incluirla en los layouts de las actividades mediante la clausula include.

```
<include android:id="@+id/appbar"  layout="@layout/toolbar" />
```

En este caso el ID de la barra de herramientas se define en el include.

[7] https://developer.android.com/training/appbar/

4. AdapterViews

Son Groupviews (como lo son LinearLayout y RelativeLayout) que son manejados por una adaptador. Esto permite manejar una fuente de datos que serán gestionados por esta vista.

4.1. ListView

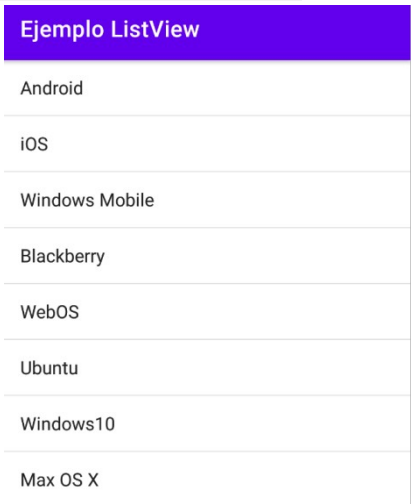
Es una lista de elementos seleccionables con scroll. Puede crearse con ella una lista simple de elementos o llegar incluso a montar un layout personalizado en cada uno de sus entradas.

Para poder utilizar un ListView tenemos dos opciones:

- Opción 1: Usar ListView como un componente más de la interfaz. Para comprobar su funcionamiento insertaremos ListView al layout que ocupe toda la pantalla asegurándonos que posea un ID.

El layout podría ser:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <ListView
      android:id="@+id/listView"
      android:layout_width="0dp"
      android:layout_height="0dp"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



Como fuente de datos del ListView usaremos una lista, o un array, que rellenaremos con unos datos de ejemplo. La lista de datos que se le pasa puede ser de cualquier tipo dato ya que en cada fila del ListView lo que se visualizara es la cadena que devuelve el método toString sobre cada valor de la lista. La decisión de usar una lista o un array depende si el ListView va a cambiar de tamaño o no ya que en un array no se puede cambiar mientras que en una lista sí.

A continuación tenemos que establecer el adaptador. Este es el objeto que se encarga actualizar el contenido del ListView a partir de los datos que se le pasan.

Para realizar el control de la pulsación de un elemento del ListView deberemos establecer el listener correspondiente. En este caso será: setOnItemClickListener.

Un código de ejemplo de su utilización puede ser:

```
public class MainActivity extends AppCompatActivity {

   final ArrayList<String> lista = new ArrayList<>();  // Fuente de datos
   ArrayAdapter<String> adapter;          // Adaptador para el ListView
   ListView lv;                           // listView

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

      lv=findViewById(R.id.listView);
      String[] sistemas = new String[] { "Android", "iOS", "Windows Mobile",
       "Blackberry", "WebOS", "Ubuntu","Windows10", "Max OS X", "OpenBSD",
       "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2", "Ubuntu",
       "Windows 11", "Solaris", "Linux", "Android", "iOS", "FreeBSD", "Haiku" };

      // se añaden los elementos del array a la lista
      lista.addAll(Arrays.asList(sistemas));
      for (int i = 0; i < sistemas.length; ++i) {
         lista.add(sistemas[i]);
      }
      // Se define el adaptador usando el contexto, el tipo de listView
      // y la colección de datos.
      adapter=new ArrayAdapter<String>(this,
                                  android.R.layout.simple_list_item_1, lista);

      // se añaden elementos al adaptador mediante código
      adapter.addAll("Windows 8", "Unix");

      // se asigna el adaptador al ListView
      lv.setAdapter(adapter);

      // Se gestiona la pulsación de un elemento del ListView
      lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
         @Override   // Representa el elemento del listView pulsado
                     // donde position es la posición pulsada
         public void onItemClick(AdapterView<?> adapterView, View view,
                                                    int position, long id) {
            // Se obtiene los datos de la fila pulsada de dos formas diferentes
            String cad="Posición pulsada: "+ position + "\nEn adaptador: " +
              adapter.getItem(position) + "\nEn la lista: " + lista.get(position);

            // Se muestran los datos mediante un Toast
            Toast.makeText(getApplicationContext(), cad,
                                              Toast.LENGTH_SHORT).show();
         }
      });
   }
}
```

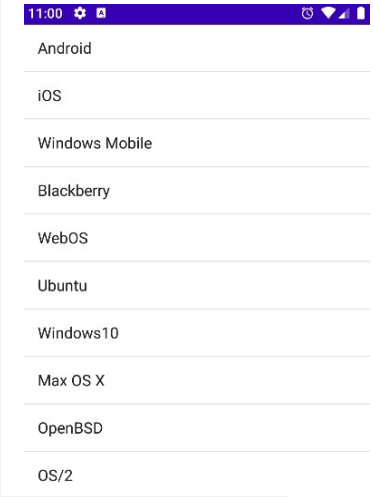
- Opción 2: Cambiando la herencia de la activity principal para que herede de ListActivity (en ve de AppCompatActivity) y eliminado la llamada a setContentView, si no fallará (ya no se va a representar un layout de un fichero XML sino que el layout será el propio listView). En esta versión no sirve de nada añadir componentes al layout debido que este no se visualizara sino que es el ListView ocupara toda la pantalla.

Se utiliza setListAdapter para asignar el adaptador al ListView.

Las pulsaciones se gestionan con el método onListItemClick que tendremos que sobrescribir.

El código quedará de la siguiente manera:

```
public class MainActivity extends ListActivity {
   final ArrayList<String> lista = new ArrayList<>();// Fuente de datos
   ArrayAdapter<String> adapter;  // Adaptador para el ListView

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      //setContentView(R.layout.activity_main); No poner
      //ListView lv=findViewById(R.id.listView); No poner

      String[] sistemas = new String[] { "Android", "iOS", "Windows Mobile",
       "Blackberry", "WebOS", "Ubuntu","Windows10", "Max OS X", "OpenBSD",
       "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2", "Ubuntu",
       "Windows 11", "Solaris", "Linux", "Android", "iOS", "FreeBSD", "Haiku" };

      // se añaden los elementos del array a la lista
      lista.addAll(Arrays.asList(sistemas));
      for (int i = 0; i < sistemas.length; ++i) {
         lista.add(sistemas[i]);
      }
      // Se define el adaptador usando el contexto,
      // el tipo de listView y la colección de datos.
      adapter=new ArrayAdapter<String>(this,
               android.R.layout.simple_list_item_1,  lista);

      // se añaden elementos al adaptador mediante código
      adapter.addAll("Windows 8", "Unix");

      // se asigna el adaptador al ListView
      setListAdapter(adapter);
}
   @Override
   protected void onListItemClick(ListView l, View v, int position, long id) {
     super.onListItemClick(l, v, position, id);
           String cad="Posición pulsada: "+ position + "\nEn adaptador: " +
             adapter.getItem(position) + "\nEn la lista: " + lista.get(position);

           // Se muestran los datos mediante un Toast
           Toast.makeText(getApplicationContext(), cad,
                                          Toast.LENGTH_SHORT).show();
   }
}
```



4.2.ListView con selección.

Ahora veremos cómo seleccionar elementos en el ListView.

Para ello deberemos establecer el modo de selección en el ListView:

- CHOICE_MODE_NONE: no se selecciona nada.
- CHOICE_MODE_SINGLE: solo se puede seleccionar una fila.
- CHOICE_MODE_MULTIPLE: se pueden seleccionar más de una fila.

Dependiendo de la opción elegida en el punto anterior podemos escoger una de las dos opciones siguientes:

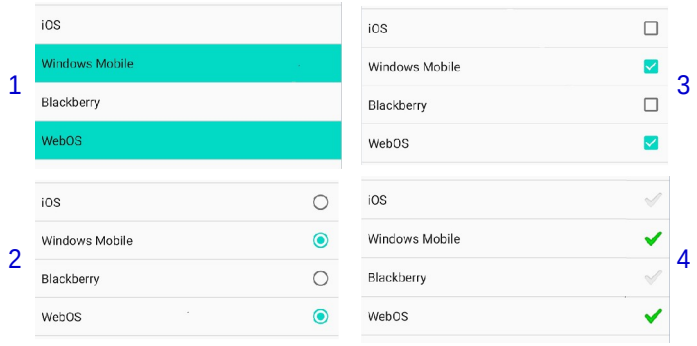
- El ListView está como componente
```
lv.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
```
- La clase hereda de ListView
```
this.getListView().setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
```

Además cambiaremos el layout establecido en el adapter:

```
adapter=new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, lista);
```

Por uno de los siguientes:

1. simple_list_item_activated_1
2. simple_list_item_single_choice
3. simple_list_item_multiple_choice
4. simple_list_item_checked



Para eliminar los elementos marcados podemos crear un botón o un menú en el que se establece su atributo onClick a un método que crearemos. Será en este método donde se realice el borrado de los elementos.

Mediante el método notifyDataSetChanged se avisa al adaptador que el origen de datos se ha modificado y que el listView se actualice.

Su código es el siguiente:

```
public void clickEliminar(MenuItem item) {
      @SuppressWarnings ("unchecked")
      // Obtenemos el adaptador
      ArrayAdapter<String> adapter=(ArrayAdapter<String>) lv.getAdapter();
      // Recorremos la lista en orden inverso para borrar sin problema
      for (int i= lv.getCount() - 1; i >= 0; i--) {
            // Si un elemento está marcado lo eliminamos
            if (lv.isItemChecked(i)) lista.remove(i);
      }
      // Eliminamos las marcas de la lista
      lv.getCheckedItemPositions().clear();
      // Informamos que ha habido cambios en los datos del adaptador
      // para actualizar la lista
      adapter.notifyDataSetChanged();
}
```

En el ejemplo se usa lista.remove(i) para eliminar un elemento, siendo i la posición de un elemento seleccionado.

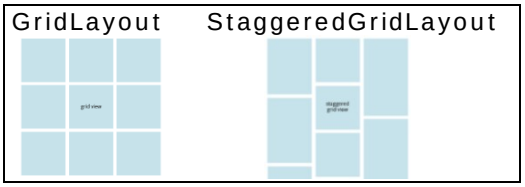
4.3.RecyclerView [8] [9] [10]

RecyclerView es un componente que, al igual que ListView, permite mostrar colecciones de datos pero de una forma más flexible. Se le denomina Recycler por que es capaz de rehusar posiciones sin tener que volver a generarlas mejorando, con ello, la eficiencia. RecyclerView se ha creado como un substituto de ListView.

Cada celda del RecyclerView va a estar representada por un layout definido en un fichero XML dentro del directorio layout del directorio de recursos.

RecyclerView para su funcionamiento se apoya en los siguientes elementos:

- Adapter: En un adaptador que se define como una clase que hereda de RecyclerView.Adapter. Se encarga de suministrar los datos al RecyclerView y manejar los elementos que se visualizan en el.
- ViewHolder: representa cada uno de los elementos que RecyclerView va a visualizar (lo podemos equiparar como cada una de las celdas de una tabla). Son instancias de una clase que definiremos y que debe heredar de RecyclerView.ViewHolder. Estos elementos son manejados por el adaptador según sus necesidades.
- LayoutManager[11]: es el encargado de indicarle a RecyclerView la posición de los elementos a mostrar, a ocultar, a rehusar, ... Tenemos disponibles:
  - LinearLayoutManager[12] (1Dimensión): ofrece una funcionalidad similar a ListView mostrando los datos de forma lineal ya sea esta horizontal o vertical.
  - GridLayoutManager[13] (2D): muestra los datos en forma de cuadricula con celdas del mismo tamaño en la misma fila o columna.
  - StaggeredGridLayoutManager[14] (2D): muestra los datos en una cuadricula con celdas de tamaño variable o incluso se puede crear una distribución propia.



- ItemDecoration[15]: permite definir aspectos visuales de RecyclerView como pueden los divisores. No es obligatorio.
- ItemAnimator[16]: indica cómo se anima RecyclerView cuando se producen cambios sobre él. Estos cambios pueden producirse cuando se añaden, mueven, modifican o se eliminan elementos del mismo. No es obligatorio. Por defecto se usa DefaultItemAnimator par las animaciones pero podemos crear muestre propia animación.

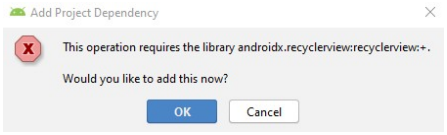
[8] https://developer.android.com/guide/topics/ui/layout/recyclerview#java
[9] https://developer.android.com/reference/androidx/recyclerview/widget/package-summary
[10] https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.html
[11] https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView. LayoutManager
[12] https://developer.android.com/reference/androidx/recyclerview/widget/LinearLayoutManager
[13] https://developer.android.com/reference/androidx/recyclerview/widget/GridLayoutManager
[14] https://developer.android.com/reference/androidx/recyclerview/widget/StaggeredGridLayoutManager
[15] https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.ItemDecoration.html
[16] https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.ItemAnimator.html

Cuando se van a visualizar los elementos de un RecycleView Android no crea de inicio todos los elementos sino que solo crea, de forma dinámica, los elementos que se visualizando en pantalla más alguno más para visualizar los próximos elementos que pueden aparecen, por ejemplo al realizar un scroll. Asi cuando un elemento desaparece de pantalla pasa a ser marcado como reutilizable para los nuevos elementos que van aparecer. Con ello consigue una mejor eficiencia. Incluso almacena los últimos elementos que han desaparecido al realizar el scroll, así si cambiamos la dirección del scroll estos elementos se pueden rehusar, sin tener que volver a generarlos (esto es costoso).

4.3.1.Creación de un RecyclerView

Los pasos para utilizar RecyclerView son los siguientes.

1. Añadir el componente RecyclerView al Layout.

   En versiones antiguas de Android Studio para poder usar RecyclerView en nuestros proyectos se deberá incorporar previamente la dependencia necesaria. Este paso lo realiza automáticamente Android Studio la primera vez que se añade el componente a nuestro proyecto.

   También se puede añadir esta dependencia a mano añadiéndola en la sección dependencies del fichero de configuración del módulo principal build.gradle (Module: app):

   ```
   implementation 'androidx.recyclerview:recyclerview:1.1.0'
   ```

   En las últimas versiones de Android Studio las acciones anteriores ya no son necesarias.

   Una vez añadido genera un código XML similar al siguiente:

   ```
   <androidx.recyclerview.widget.RecyclerView
    android:id="@+id/mi_recycler_view"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
   ```

   Donde se deberá, si no lo tiene ya, especificar su id correspondiente y sus constraints.

2. En cada fila del RecyclerView vamos a visualizar el nombre, año y logo de una serie de sistemas operativos. Deberemos crear la clase, SistemasOperativos, que servirá como contenedora de estos datos.

   ```
   public class SistemaOperativo {

       private String nombre;      // Nombre del sistema operativo
       private String ano;         // Año de salida
       private int logo;           // Int que representa la imagen en el fichero R
   ```

```
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getAno() {
        return ano;
    }
    public void setAno(String ano) {
        this.ano = ano;
    }
    public int getLogo() {
        return logo;
    }
    public void setLogo(int logo) {
        this.logo = logo;
    }
    public SistemaOperativo(String nombre, String ano, int logo) {
        this.nombre = nombre;
        this.ano = ano;
        this.logo = logo;
    }
}
```

3. En el fichero colors.xml definimos un color que usaremos como color de fondo de cada celda del RecyclerView. Esta parte es opcional.

```
<color name="colorcelda">#7CB342</color>
```

4. Cada posición del RecyclerView visualizará un Layout. Este deberá poseer los componentes necesarios para mostrar la información que se desee. En nuestro ejemplo el Layout se llamará celda.xml y tendrá un ImageView, dos TextView para mostrar contenido y un TextView como interfaz.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="1dp"
    android:background="@color/colorcelda"
    android:padding="4dp">

  <ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:adjustViewBounds="true"
    tools:srcCompat="@tools:sample/avatars" />
```

```
    <TextView
        android:id="@+id/textViewfondo"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:alpha="0.5"
        android:background="#000000"
        android:text="TextView"
        app:layout_constraintBottom_toBottomOf="@+id/imageView"
        app:layout_constraintEnd_toEndOf="@+id/imageView"
        app:layout_constraintStart_toStartOf="@id/imageView"
        app:layout_constraintTop_toTopOf="@+id/textViewNombre"/>

    <TextView
        android:id="@+id/textViewNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="4dp"
        android:layout_marginBottom="4dp"
        android:textColor="#FFFFFF"
        android:textSize="16sp"
        app:layout_constraintBottom_toTopOf="@+id/textViewYear"
        app:layout_constraintStart_toStartOf="@id/imageView"/>

    <TextView
        android:id="@+id/textViewYear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="4dp"
        android:layout_marginBottom="8dp"
        android:textColor="#FFEB3B"
        android:textSize="14sp"
        app:layout_constraintBottom_toBottomOf="@id/imageView"
        app:layout_constraintStart_toStartOf="@id/imageView"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

5. Deberemos crear el adaptador. Para ello realizaremos los siguientes pasos:

   a) Crearemos una clase Java que contendrá el adaptador. En nuestro ejemplo la llamaremos MiAdaptador. Tendra como atributo un colección con los datos que se visualizara en el RecyclerView. En nuestro ejemplo se usara un ArrayList de sistemas operativos.

```
public class MiAdaptador {
    // Puede ser cualquier estructura de datos
    ArrayList<SistemaOperativo> sistemasOperativos;
}
```

   b) Implementar, normalmente dentro del adaptador, una clase que herede de RecyclerView.ViewHolder. Esta clase se instanciará para cada celda del RecyclerView y desde la cual accederemos a cada uno de los elementos que se definen en el layout, que definimos en el paso 4, que representa cada calda. En nuestro caso la llamaremos MyViewHolder pero puede ser cualquiera.

```
public class MiAdaptador{
    // Puede ser cualquier estructura de datos
    ArrayList<SistemaOperativo> sistemasOperativos;
    public class MyViewHolder extends RecyclerView.ViewHolder {
        // Elementos que queremos mostrar en cada posición del RecyclerView,
        // normalmente se corresponderán con los definidos en el layout
        private TextView nombre;
        private TextView year;
        ImageView logo;
        // Constructor: asocia cada atributo de la clase con su
        // correspondiente componente definido en la layout (ViewElemento)
        public MyViewHolder(View viewElemento) {
            super(viewElemento);
            this.nombre=viewElemento.findViewById(R.id.textViewNombre);
            this.year=viewElemento.findViewById(R.id.textViewYear);
            this.logo=viewElemento.findViewById(R.id.imageView);
        }
        public TextView getNombre() {
            return nombre;
        }
        public TextView getYear() {
            return year;
        }
        public ImageView getLogo() {
            return logo;
        }
    }
}
```

c) Al adaptador se le añade un constructor con el que se inicializará la colección de datos a mostrar en el RecyclerView.

```
// Constructor
public MiAdaptador(ArrayList<SistemaOperativo> sistemasOperativos) {
   this.sistemasOperativos = sistemasOperativos;
}
```

d) Para que la clase que hemos creado (MiAdaptador) sea un adaptador de un RecyclerView debe heredar de RecyclerView.Adapter (toma como parámetro la clase creada en el punto anterior, en nuestro caso MyViewHolder). Al añadir esta herencia se tienen que implementar los siguientes tres métodos abstractos:

- onCreateViewHolder: se encarga de crear los elementos, que heredan de ViewHolder, que necesite para representar el contenido del RecyclerView.
- onBindViewHolder: cuanto reutilizamos un ViewHolder se encarga de actualizar su contenido con los nuevos datos que le corresponde con su posición.
- onItemCount: muestra el tamaño del la colección de datos que va a mostrar RecyclerView.

Por lo que, con estos métodos, el adaptador completo quedaría:

```
public class MiAdaptador extends RecyclerView.Adapter<MiAdaptador.MyViewHolder> {

   // Puede ser cualquier estructura de datos
   ArrayList<SistemaOperativo> sistemasOperativos;

   // Constructor
   public MiAdaptador(ArrayList<SistemaOperativo> sistemasOperativos) {
      this.sistemasOperativos = sistemasOperativos;
   }

   // Crea nuevos elementos expandiendo el layout definido en el fichero R.layout.celda
  // que usamos para crear el Holder que devolveremos
   @NonNull
   @Override
 public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
      View elemento= LayoutInflater.from(parent.getContext()).inflate(R.layout.celda,
                                                                      parent, false);
      MyViewHolder mvh = new MyViewHolder(elemento);
      return mvh ;
      // return new MyViewHolder(elemento);
   }

   // Establece al objeto holder los valores de la colección de datos que
   // están en la posición position.
   @Override
   public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
      SistemaOperativo so=this.sistemasOperativos.get(position);
      holder.getNombre().setText(so.getNombre());
      holder.getYear().setText(so.getAno());
      holder.getLogo().setImageResource(so.getLogo());

      if (selectedPos == position)
          holder.itemView.setBackgroundResource(R.color.seleccionado);
      else holder.itemView.setBackgroundResource(R.color.colorcelda);
}


   // Número de elementos en la colección de datos. Será el número de elementos
   // que se tendrá el RecycleView
   @Override
   public int getItemCount() {
      return this.sistemasOperativos.size();
   }

   public class MyViewHolder extends RecyclerView.ViewHolder {

      // Elementos que queremos mostrar en cada posición del RecyclerView,
      // normalmente se corresponderán con los definidos en el layout
      private TextView nombre;
      private TextView year;
      ImageView logo;
```

```
        // Constructor: asocia cada atributo de la clase con su
        // correspondiente componente definido en la layout (ViewElemento)
        public MyViewHolder(View viewElemento) {
            super(viewElemento);
            this.nombre=viewElemento.findViewById(R.id.textViewNombre);
            this.year=viewElemento.findViewById(R.id.textViewYear);
            this.logo=viewElemento.findViewById(R.id.imageView);
        }
        public TextView getNombre() {
            return nombre;
        }
        public TextView getYear() {
            return year;
        }
        public ImageView getLogo() {
            return logo;
        }
    }
}
```

6. Se rellenan, en onCreate de la actividad principal, la colección con los datos a mostrar. Instanciamos el adaptador con esta colección y obtenemos el objeto RecyclerView del layout.

```
public class MainActivity extends AppCompatActivity {
  ArrayList<SistemaOperativo> sistemas;
  RecyclerView rv;
  MiAdaptador miAdaptador;

  public void rellenaDatos(int vueltas){
    sistemas= new ArrayList<SistemaOperativo>();
    for (int i=1;i<=vueltas;i++) {
      sistemas.add(new SistemaOperativo("Ubuntu 14 "+i, "2014", R.drawable.ubuntu14));
      sistemas.add(new SistemaOperativo("MacOS X "+i, "2004", R.drawable.maxx));
      sistemas.add(new SistemaOperativo("Windows 95 "+i, "1995", R.drawable.w95));
      sistemas.add(new SistemaOperativo("Debian "+i, "1993", R.drawable.debian));
      sistemas.add(new SistemaOperativo("Windows 98 "+i, "1998", R.drawable.w98));
      sistemas.add(new SistemaOperativo("Linux Mint 15 "+i, "2013", R.drawable.mint));
      sistemas.add(new SistemaOperativo("Windows 10 "+i, "2016", R.drawable.w10));
      sistemas.add(new SistemaOperativo("Android "+i, "2006", R.drawable.android));
      sistemas.add(new SistemaOperativo("iOS 8 "+i, "2014", R.drawable.ios8));
      sistemas.add(new SistemaOperativo("Windows Vista "+i, "2007", R.drawable.wvista));
      sistemas.add(new SistemaOperativo("Windows XP "+i, "2001", R.drawable.wxp));
      sistemas.add(new SistemaOperativo("Elementary "+i,"2014", R.drawable.elementary));
      sistemas.add(new SistemaOperativo("Ubuntu 20 "+i, "2020", R.drawable.ubuntu20));
      sistemas.add(new SistemaOperativo("Windows 11 "+i, "2021", R.drawable.w11));
      }
    }
```

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

        rellenaDatos(10);

        miAdaptador=new MiAdaptador(sistemas);
        rv=findViewById(R.id.mi_recycler_view);
    }
}
```

7. Se define el LayoutManager que va a usar. Este determina cómo se distribuyen las celdas en el RecyclerView. En un mismo RecyclerView cambiando el LayoutManager las celdas se distribuirán de forma totalmente distinta. Como se comento al principio del punto Android proporciona, por defecto, tres LayoutManager: LinearLayoutManager, GridLayoutManager y StaggeredGridLayoutManager.

   Se comienza definiendo un atributo de instancia que contendrá el LayoutManager que se usará para asignarle uno de los LayoutManager predefinidos de Android.

   ```
   RecyclerView.LayoutManager miLayoutManager;
   ```

   - LinearLayoutManager: los elementos se distribuyen en una única fila, si es horizontal, o en una única columna, si es vertical. Es decir tiene una única dimensión. Se dispone de dos constructores:

     ```
     miLayoutManager = new LinearLayoutManager(this);
     miLayoutManager = new LinearLayoutManager(this,
                                 LinearLayoutManager.HORIZONTAL, false);
     ```

     El primero creara un LinealLayoutManager vertical mientras que en el segundo se puede modificar su distribución. Donde:

     - En ambos casos el primer parámetro es el contexto de la aplicación.
     - El segundo parámetro indica la orientación del Layout. Las variaciones son LinearLayoutManager.HORIZONTAL y LinearLayoutManager.VERTICAL.
     - El tercer parámetro, si es verdadero, indica que la lista se invierta, es decir comience por el final.

   - GridLayoutManager: los elementos se distribuyen en forma de cuadricula, teniendo todas las celdas el mismo tamaño. En caso de ser horizontal se le puede especificar el número de filas y en caso de ser vertical el número de columnas. Se dispone de dos constructores:

     ```
     miLayoutManager = new GridLayoutManager(this, 3);
     miLayoutManager = new GridLayoutManager(this, 3,
                                     GridLayoutManager.VERTICAL, true);
     ```

El primero creara un GridLayoutManager vertical mientras que en el segundo se puede modificar su distribución. Donde:

- El primer parámetro es el contexto de la aplicación.
- El segundo parámetro, según la orientación, indica el número de filas (si es horizontal) o columnas (si es vertical).
- El tercer parámetro indica la orientación del Layout. Las variaciones son GridLayoutManager.HORIZONTAL y GridLayoutManager.VERTICAL.
- El cuarto parámetro, si es verdadero, indica que la lista se invierta, es decir comience por el final.

- StaggeredGridLayoutManager: los elementos se distribuyen en forma de cuadricula escalonada, pudiendo tener las celdas distinto tamaño. En caso de ser horizontal se le puede especificar el número de filas y en caso de que sea vertical el número de columnas. Usaremos el constructor:

```
miLayoutManager = new StaggeredGridLayoutManager(3,
StaggeredGridLayoutManager.VERTICAL);
```

Donde:

- El primer parámetro es el contexto de la aplicación.
- El segundo parámetro indica la orientación del Layout. Las variaciones son StaggeredGridLayoutManager.HORIZONTAL y StaggeredGridLayoutManager.VERTICAL.

En vez de usar un objeto de tipo RecyclerView.LayoutManager se puede usar directamente el tipo de LayoutManager que queramos usar. Por ejemplo en el caso del un GridLayoutManager podríamos usar:

```
GridLayoutManager miLayoutManager =new GridLayoutManager(this, 3,
GridLayoutManager.VERTICAL, true);
```

Dependiendo del tiempo de objeto tendremos unos métodos u otros. Por ejemplo en el caso de GridLayoutManager o del StaggeredGridLayoutManager tendríamos los métodos:

- getSpanCount:obtiene el número de filas o de columnas del LayoutManager
- setSpanCount: permite establecer el número de filas o de columnas del LayoutManager.

8. Asignamos el LayoutManager creado al RecyclerView:

```
rv.setLayoutManager(miLayoutManager);
```

9. Mediante el método setAdapter se asigna el adaptador creado a RecyclerView.

```
rv.setAdapter(miAdaptador);
```

El resultado de los pasos anteriores seria el siguiente RecuclerView.

4.3.2.Elementos opcionales

ItemDecorations

De forma opcional se pueden usar ItemDecorations a RecyclerView para añadir elementos gráficos a los elementos que posee. Ejemplos típicos son mostrar divisores entre los elementos, marcar elementos, agrupar elementos, … siendo dibujados en el orden en que se añaden al RecyclerView.

Por eficiencia se recomienda usar elementos ItemDecoration como divisores antes que los divisores que proporcionan los propios layouts.

Debido a lo laborioso de definir un ItemDecoration desde cero solo usaremos los divisores horizontales y verticales que vienen definidos en Android.

Solo hay que definirlos y añadirlos al RecyclerView definido:

- Divisor horizontal:

```
rv.addItemDecoration(new DividerItemDecoration(this,
                                    DividerItemDecoration.HORIZONTAL));
```

- Divisor vertical:

```
rv.addItemDecoration(new DividerItemDecoration(this,
                                      DividerItemDecoration.VERTICAL));
```

- Divisor dinámico: Detecta lo orientación de LayoutMananer y se establece el divisor correspondiente.

```
rv.addItemDecoration(new DividerItemDecoration(this,
          ((StaggeredGridLayoutManager)miLayoutManager).getOrientation()));
```

StaggeredGridLayoutManager se debe sustituir por el LayoutMananer que estemos usando.

- Un ejemplo sencillo de creación de un ItemDecoracion a medida se puede encontrar en: https://gist.github.com/nesquena/db922669798eba3e3661

```
public class SpacesItemDecoration extends RecyclerView.ItemDecoration {
      private final int mSpace;
      public SpacesItemDecoration(int space) {
            this.mSpace = space;
      }
      @Override
      public void getItemOffsets(Rect outRect, View view,
                                 RecyclerView parent, RecyclerView.State state) {
            outRect.left = mSpace;
            outRect.right = mSpace;
            outRect.bottom = mSpace;
            // Add top margin only for the first item to avoid
            if (parent.getChildAdapterPosition(view) == 0)
                  outRect.top = mSpace;  // double space between items
      }
}
```

Se utiliza de la siguiente forma:

```
SpacesItemDecoration decoration = new SpacesItemDecoration(3);
rv.addItemDecoration(decoration);
```

ItemAnimators

Opcionalmente usando ItemAnimators se pueden añadir animaciones a RecyclerView. ItemAnimator realiza animaciones cuando se añaden, mueven, modifican o se eliminan elementos del RecyclerView. Usaremos las animaciones que por defecto que proporciona Android usando para ello DefaultItemAnimator.

```
rv.setItemAnimator(new DefaultItemAnimator());
```

Efecto de centrado

Se puede añadir un efecto de centrado de forma que se fuerce a que una de las columnas o filas mostradas quede siempre centrada en pantalla. Para ello se utiliza el siguiente código:

```
SnapHelper snapHelper = new LinearSnapHelper();
snapHelper.attachToRecyclerView(rv);
```

Optimización

Una optimización, de rendimiento, que admite RecyclerView es indicar que cambios en su contenido no afectaran al tamaño de los elementos que posee.

```
rv.setHasFixedSize(true);
```

4.3.3.Pulsación de un elemento del RecyclerView

Al contrario que con ListView Android no posee el método setOnItemClickListener para gestionar las pulsaciones en un RecyclerView. Por lo deberemos añadírselo nosotros. Existen diferentes aproximaciones de las cuales veremos la siguiente (otra opción se podría ver en el siguiente enlace).

1. Se crea, en el fichero colors.xml, un nuevo color para el color de fondo del elemento seleccionado.

```
<color name="seleccionado">#E98E6D</color>
```

2. Se añade el atributo de instancia selectedPos inicializándolo a RecyclerView. NO_POSITION (internamente tiene un valor -1). Este atributo nos permite controlar el número del elemento que ha sido seleccionado. Añadimos también su getter y su setter.

```
int selectedPos=RecyclerView.NO_POSITION;

public int getSelectedPos() {
   return selectedPos;
}

public void setSelectedPos(int nuevaPos) {

   // Si se pulsa sobre el elemento marcado
   if (nuevaPos == this.selectedPos){

      // Se establece que no hay una posición marcada
      this.selectedPos=RecyclerView.NO_POSITION;

      // Se avisa al adaptador para que desmarque esa posición
      notifyItemChanged(nuevaPos);

   } else {  // El elemento pulsado no está marcado

      if (this.selectedPos >=0 ) {  // Si ya hay otra posición marcada
         // Se desmarca
         notifyItemChanged(this.selectedPos);
      }

      // Se actualiza la nueva posición a la posición pulsada
      this.selectedPos = nuevaPos;

      // Se marca la nueva posición
      notifyItemChanged(nuevaPos);
   }
}
```

3. En el método onBindViewHolder y mediante código se establece el color de fondo de un elemento dependiendo si este está seleccionado o no.

```
if (selectedPos == position) {
     holder.itemView.setBackgroundResource(R.color.seleccionado);
} else {
     holder.itemView.setBackgroundResource(R.color.colorcelda);
}
```

4. En contructor de la clase que representa cada elemento del RecycleView, MyViewHolder en nuestro ejemplo, establecemos un listener para gestionar la pulsación de cada celda del RecycleView.

```
public MyViewHolder(View viewElemento) {
   super(viewElemento);

   this.nombre=viewElemento.findViewById(R.id.textViewNombre);
   this.year=viewElemento.findViewById(R.id.textViewYear);
   this.logo=viewElemento.findViewById(R.id.imageView);

   viewElemento.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {

         // getAdapterPosition devuelve la posición del view en el adaptador
         setSelectedPos(getAdapterPosition());

         // Si hay una posición marcada se muestra un  Toast
         if (selectedPos>RecyclerView.NO_POSITION){
            Toast.makeText(view.getContext(), sistemasOperativos.
                  get(selectedPos).getNombre(),Toast.LENGTH_SHORT).show();
         }
      }
   });
}
```

4.3.4.Cambios en el contenido del RecyclerView

Al igual que sucedía con ListView cuando se realizan modificaciones en al contenido de la colección de datos asociada con RecyclerView se ha de avisar al adaptador que se han realizado esas modificaciones.

Para ello, el adaptador, dispone de los siguientes métodos específicos que aceptan como parámetro un entero:

- notifyItemInserted(posición): Avisa que se ha insertado un elemento en la posición indicada.
- notifyItemRemoved(posición): Avisa que se ha eliminado un elemento en la posición indicada.
- notifyItemChanged(posición): Avisa que se ha modificado un elemento en la posición indicada.
- notifyItemMoved(posiciónOriginal, posiciónNueva): Avisa que el elemento que estaba en la posición posiciónOriginal se ha movido a posiciónNueva.
- notifyDataSetChanged(): Avisa que se han modificado los datos del RecyclerView. Es menos eficiente que los anteriores ya que asume que todos los elementos no son validos por lo que fuerza al LayoutManager a actualizar y recolocar todos estos elementos.

Una vez realizados los cambios existe la posibilita de moverse a una posición determinada, usando el parámetro posición, mediante los siguientes métodos de RecyclerView:

- scrollToPosition(posición)
- smoothScrollToPosition(posición)

Veamos, en un ejemplo, cómo gestionar estas operaciones sobre el elemento seleccionado en el RecyclerView mediante la pulsación de opciones del menú.

1. Creamos un menú con cuatro botones para realizar las cuatro acciones sobre el elemento marcado. Un ejemplo del menu puede ser:

```
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id="@+id/madd"
      android:icon="@drawable/ic_add"
      android:title="Add"
      app:showAsAction="always" />
   <item
      android:id="@+id/mdel"
      android:icon="@drawable/ic_delete"
      android:title="Del"
      app:showAsAction="always" />
   <item
      android:id="@+id/medit"
      android:icon="@drawable/ic_edit"
      android:title="Edit"
      app:showAsAction="always" />
   <item
      android:id="@+id/mmove"
      android:icon="@drawable/ic_move"
      android:title="Move"
      app:showAsAction="always" />
</menu>
```

2. Se implementa el método onOptionsItemSelected que gestiona la pulsación de las opciones del menú:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   int pos=miAdaptador.getSelectedPos();
   ActionBar ab=getSupportActionBar();

   // Si hay un elemento pulsado
   if (miAdaptador.getSelectedPos()>=0){
      switch (item.getItemId()) {
         // Se añade un nuevo elemento con la imagen new_item
         case R.id.madd:
            // Se añade un elemento en la posición a insertar
            sistemas.add(pos,new SistemaOperativo("prueba" + pos,
                                 String.valueOf(pos), R.drawable.new_item));
```

```
            // Se actualiza esa posición con los nuevos datos
            miAdaptador.notifyItemInserted(pos);

           // Se desmarca la antigua posición marcada
            miAdaptador.notifyItemChanged(pos+1);
            break;

         // Se borra el elemento marcado
         case R.id.mdel:
            // Borramos el elemento indicado
            sistemas.remove(pos);

            // Indicamos al adaptador que el elemento se ha borrado
            miAdaptador.notifyItemRemoved(pos);

            // Indicamos que no hay elementos marcados
            miAdaptador.setSelectedPos(RecyclerView.NO_POSITION);
            break;

         // Editamos el nombre del sistema operativo a modificado
         case R.id.medit:
            sistemas.get(pos).setNombre("Modificado");

            // Se actualiza esa posición con los nuevos datos
            miAdaptador.notifyItemChanged(pos);
            break;

         // Se mueve el elemento seleccionado a la posición 1
         // (la primera posición es 0)
         case R.id.mmove: // mover:
            SistemaOperativo aux=sistemas.get(pos);

            // Nueva posición a mover
            int newPos=1;

            // Eliminamos el elemento de su posición original
            sistemas.remove(pos);

            // Lo movemos a su nueva posición
            sistemas.add(newPos, aux);

            // Se actuliza la posiciones original y nueva para que
            // reflejen los cambios
            miAdaptador.notifyItemChanged(pos);
            miAdaptador.notifyItemMoved(pos,newPos);

            // Indicamos que la posición seleccionada es newPos
            miAdaptador.setSelectedPos(newPos);
            break;
      }
   }else {
      Toast.makeText(this, "Pulsa posición", Toast.LENGTH_SHORT).show();
   }
   ab.setTitle("Tam: "+sistemas.size());
   ab.setSubtitle("Pos: "+miAdaptador.getSelectedPos());
   return true;
}
```

5. Spinner[17]

Un Spinner permite la selección de un elemento textual entre varios. Se puede equiparar a un elemento Select en Html. Su creación es prácticamente igual a la de un ListView.

Para probar su funcionamiento crearemos un proyecto en el cual se añadirán un TextView y un Spinner.

Los valores que va a mostrar el Spinner se obtendrán, en nuestro ejemplo, desde un array de Strings definido en el fichero strings.xml.

```
<string-array name="provincias">
   <item >La Coruña</item>
   <item >Lugo</item>
   <item >Orense</item>
   <item >Pontevedra</item>
</string-array>
```

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

Al igual que un ListView el contenido del Spinner se establece mediante un adaptador:

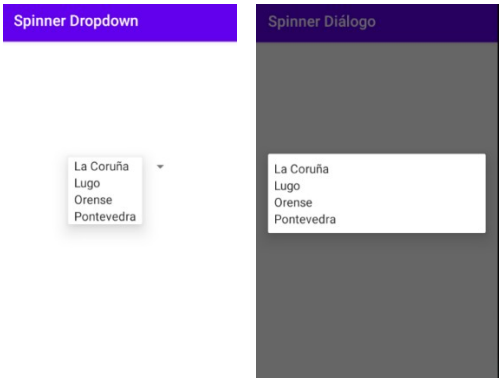
```
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
                              android.R.layout.simple_spinner_item, provincias);
```

En el que se establece el tipo de layout que visualizará a:

```
android.R.layout.simple_spinner_item
```

Mediante la propiedad spinnerMode se puede escoger el tipo de Spinner que se va a utilizar:

- dropdown: se muestra un menú desplegable. En la propiedad popupBackground se le puede establecer una imagen o un color de fondo.
- dialog: se muestra un diálogo ocupando toda la pantalla. En la propiedad prompt se le puede especificar un título.



Para gestionar la selección, o no, de una opción se usara el siguiente listener: setOnItemSelectedListener.

[17] https://developer.android.com/guide/topics/ui/controls/spinner

Por lo que nuestro ejemplo podría quedar de la siguiente manera:

```
boolean muestra=false;

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

   final String[] provincias=getResources().getStringArray(R.array.provincias);
   Spinner spinner=findViewById(R.id.spinner);

   ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
                                android.R.layout.simple_spinner_item, provincias);
   spinner.setAdapter(adapter);

   spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
      @Override
      public void onItemSelected(AdapterView<?> adapterView, View view,
                                                                  int i, long l) {

         // Se usa la variable booleana para impedir que muestre el resultado
         // seleccionado por defecto cuando se accede al activity. Solo mostramos
         // cuando seleccionamos una opción.
         if (!muestra) muestra=true;
         else {
            Toast.makeText(getApplicationContext(), "Has selecionado: " +
                                    adapterView.getItemAtPosition(i).toString(),
                                                Toast.LENGTH_SHORT).show();

            setTitle(adapterView.getItemAtPosition(i).toString());
         }
      }

      // Se ejecuta cuando se pulsa fuera del Spinner. Es decir no se
      // selecciona nada
      @Override
      public void onNothingSelected(AdapterView<?> adapterView) {

      }

   });
}
```

6. Menú contextual[18]

Este menú es el que aparece al mantener presionado durante cierto tiempo un componente y que debe presentar acciones relacionadas con dicho componente. Por ejemplo, en un listview pueden aparecer cosas como editar, eliminar, ... el componente correspondiente.

Crea en la carpeta menu el archivo contextual.xml con un menú determinado. Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:title="Editar"   android:id="@+id/editar"/>
   <item android:title="Copiar"   android:id="@+id/copiar"/>
   <item android:title="Pegar"    android:id="@+id/pegar"/>
</menu>
```

En onCreate de indica cual va a ser el listener para el componente que utiliza el menú contextual. Por ejemplo si fuera un botón:

```
final Button btn = findViewById(R.id.button);
btn.setOnCreateContextMenuListener(this);  // registerForContextMenu(btn);
```

Pero también podría ser un ListView.

```
final ListView lv = findViewById(R.id.listview);
lv.setOnCreateContextMenuListener(this);  // registerForContextMenu(lv);
```

Para que funcione debemos implementar la función onCreateContextMenu, en la propia activity que funciona como listener, permitiendo la creación del menú especificado.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
     super.onCreateContextMenu(menu, v, menuInfo);
     MenuInflater inflater = getMenuInflater();
     inflater.inflate(R.menu.contextual, menu);
}
```

El parámetro ContextMenuInfo da información añadida sobre el menú contextual. Por ejemplo se usa por AdaptersView para indicar el item de, por ejemplo, un listview.

Para gestionar la pulsación de una opción en el menú contextual usaremos el método onContextItemSelected. En este caso gestionaremos la pulsación sobre un ListView.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterView.AdapterContextMenuInfo info =
                        (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
```

[18] https://developer.android.com/guide/topics/ui/menus#context-menu

```
    // Position solo en el caso se listas
    // String cad=item.getTitle()+" -> "+info.position;
    String cad=item.getTitle().toString();

    switch (item.getItemId()) {
      case R.id.editar: // Se realizaria la ccción asociada con editar
         Toast.makeText(this, cad, Toast.LENGTH_SHORT).show();
         return true;
      case R.id.copiar: // Se realizaria la ccción asociada con copiar
         Toast.makeText(this, cad, Toast.LENGTH_SHORT).show();
         return true;
      case R.id.pegar: // Se realizaria la ccción asociada con pegar
         Toast.makeText(this, cad, Toast.LENGTH_SHORT).show();
         return true;
      default:
         return super.onContextItemSelected(item);
   }
}
```

Si se gestiona la pulsación del botón se devuelve true sino se pasa el elemento del menú a la implementación de la superclase.

Además de gestionar la pulsación con el método onContextItemSelected se pueden gestionar también mediante atributo onClick de la propia opción de menú. El método a implementar debe tener un único atributo de tipo MenuItem. Por ejemplo:

```
public void pulsaBoton(MenuItem menuItem){
    Toast.makeText(this, menuItem.getTitle(), Toast.LENGTH_SHORT).show();
}
```

En un mismo Activity se pueden definir más de un menú contextual sobre distintos elementos, según el ejemplo anterior tenernos dos listeners uno sobre un botón y otro sobre la lista:

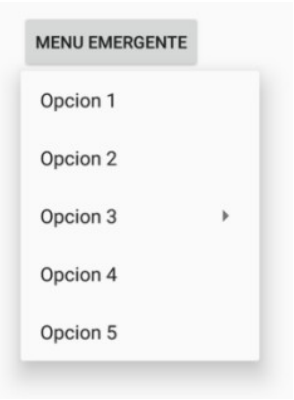
```
btn.setOnCreateContextMenuListener(this);
lv.setOnCreateContextMenuListener(this);
// Es cuando se crea el menú que tenemos que especificar cuál de
// los dos menús deberemos crear. Por ejemplo:
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
      super.onCreateContextMenu(menu, v, menuInfo);
      MenuInflater inflater=getMenuInflater();

      // Se crea el menu contextual del botón
      if (v.getId()==btn.getId()){
            inflater.inflate(R.menu.contextual,menu);
            menu.setHeaderTitle("Menu botón");
      } else { // Se crea el menu contextual del ListView
            inflater.inflate(R.menu.mlistview,menu);
            menu.setHeaderTitle("Menu listview");
      }
}
```

También, al igual que con el resto de componentes, se puede gestionar la creación del menú contextual mediante el siguiente listener en el propio componente:

```
btn.setOnCreateContextMenuListener(new View.OnCreateContextMenuListener() {
    @Override
    public void onCreateContextMenu(ContextMenu contextMenu, View view,
                                    ContextMenu.ContextMenuInfo contextMenuInfo) {
        getMenuInflater().inflate(R.menu.contextual,contextMenu);
    }
});
```

Las pulsaciones de todos los menús contextuales se realizan en el método onContextItemSelected de la misma forma que en los menús normales y para todos los menús contextuales definidos.

7. Menús emergentes[19]



Un menú emergente, PopupMenu, es un menú modal que está anclado a un componente de la Activity. Aparece debajo del componente o en el caso de no haber espacio se visualiza sobre este. Al escoger una opción o pulsar fuera del menú este desaparece.

Se debe usar para:

- Proporcionar una serie de acciones que estar relacionadas con un componente. Esta acción es diferente de un menú contextual ya que en este las opciones afectan el componente seleccionado.
- Añade diferentes acciones a un componente. Por ejemplo ante el botón pegar puede mostrar diferentes opciones de pegar.
- Proporciona un menú desplegable similar a un Spinner pero que no mantiene una selección persistente.

La forma de crear el menú es la siguiente:

- El menú se define en un archivo de menú XML como los definidos hasta ahora.
- Una vez definido se ha de asociar con un componente. Se especifica, en la propiedad onClick del componente, el nombre de un método que mostrara el menú emergente.
- Se gestiona la pulsación de una opción del menú.

[19] https://developer.android.com/guide/topics/ui/menus#PopupMenu

Un ejemplo asociado con un botón seria:

```
public void menuEmergente (View view){
   PopupMenu popup = new PopupMenu(this, view);
   popup.inflate(R.menu.menupopup);

   popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
      @Override
      public boolean onMenuItemClick(MenuItem item) {
         switch (item.getItemId()) {
            case R.id.opc1:
               Toast.makeText(getApplicationContext(), item.getTitle()+" -> opc1",
                                                      Toast.LENGTH_SHORT).show();
               return true;
            case R.id.opc2:
               Toast.makeText(getApplicationContext(), item.getTitle()+" -> opc2",
                                                      Toast.LENGTH_SHORT).show();
               return true;

           case R.id.opc3:
               Toast.makeText(getApplicationContext(), item.getTitle()+" -> opc3",
                                                      Toast.LENGTH_SHORT).show();
               return true;
            default:
         }
         return false;
      }
   });
   popup.show();
}
```

Una opción alternativa es separar la definición del menú emergente del tratamiento de las pulsaciones. Esto nos permite compartir el método OnMenuItemClickListener entre los menús existentes.

La clase debe implementar la interfaz PopupMenu.OnMenuItemClickListener y registrarla mediante setOnMenuItemclickListener.

```
public class MainActivity extends AppCompatActivity
                          implements PopupMenu.OnMenuItemClickListener {
```

Luego definimos el método que expande el menú y registramos el escuchador.

```
public void menuEmergente (View view){
   PopupMenu popup = new PopupMenu(this, view);
   popup.inflate(R.menu.menupopup);
   popup.setOnMenuItemClickListener(MainActivity.this);
   popup.show();
}
```

Ahora implementamos el método que gestiona las pulsaciones:

```
@Override
public boolean onMenuItemClick(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.opc1:
         Toast.makeText(getApplicationContext(), item.getTitle()+" -> opc1",
                                                 Toast.LENGTH_SHORT).show();
         return true;

      case R.id.opc2:
         Toast.makeText(getApplicationContext(), item.getTitle()+" -> opc2",
                                                 Toast.LENGTH_SHORT).show();
         return true;

      case R.id.opc3:
         Toast.makeText(getApplicationContext(), item.getTitle()+" -> opc3",
                                                 Toast.LENGTH_SHORT).show();
         return true;
   }
   return false;
}
```

8. Apéndice 1: Pulsación alternativa de un elemento del RecyclerView

Al contrario que con ListView Android no posee el método setOnItemClickListener para gestionar las pulsaciones en un RecyclerView. Por lo deberemos añadírselo nosotros. Existen diferentes aproximaciones de las cuales veremos la siguiente (otra opción se podría ver en el siguiente enlace).

1. Se crea, en el fichero colors.xml, un nuevo color para el color de fondo del elemento seleccionado.

```
<color name="seleccionado">#E98E6D</color>
```

2. Se añade la interfaz View.OnClickListener, al adaptador, para gestionar las pulsaciones sobre los distintos elementos, views, que contiene.

```
public class MiAdaptador extends RecyclerView.Adapter
                <MiAdaptador.MyViewHolder> implements View.OnClickListener
```

   Lo que obliga a implementar el método abstracto onClick que completaremos más adelante.

3. Se crea, en el adaptador, un atributo de instancia para contener el listener que detectara que se ha pulsado una posición y un setter para este listener.

```
private View.OnClickListener listener;

// el nombre del método es indiferente
public void setOnClickListener(View.OnClickListener listener) {
     this.listener = listener;
}
```

4. En el método onClick del adaptador, se ejecuta el método onClick del listener creado en el punto 3. Este listener se creará en la Activity que invoca el RecycleView. En view está el elemento pulsado.

```
if(listener != null) listener.onClick(view);
```

5. En el método onCreateViewHolder se establece quien gestionara las pulsaciones sobre un elemento que se expande. En este caso es la propia clase.

```
elemento.setOnClickListener(this);
```

6. Se añade el atributo de instancia selectedPos inicializándolo con valor -1. Este atributo nos permite controlar el número del elemento que ha sido seleccionado.

```
int selectedPos=RecyclerView.NO_POSITION;
```

7. En el método onBindViewHolder y mediante código se establece el color de fondo de un elemento dependiendo si este está seleccionado o no.

```
if (selectedPos == position)
holder.itemView.setBackgroundResource(R.color.seleccionado);
else holder.itemView.setBackgroundResource(R.color.elemento);
```

8. Se crea el getter y setter de selectedPos. Es en el setter donde establecemos si un elemento esta seleccionado o no avisando, al adaptador, que se ha modificado esa posición.

```
public int getSelectedPos() {
   return selectedPos;
}

public void setSelectedPos(int selectedPos) {
   // Si pulso sobre el elemento marcado
   if (selectedPos==this.selectedPos){
      // Se avisa para que desmarque esa posición
      notifyItemChanged(this.selectedPos);
      this.selectedPos=RecyclerView.NO_POSITION;
   } else {  // El elemento pulsado no está marcado
      // Si hay otra posición marcada se desmarca
      if (this.selectedPos >=0 ) notifyItemChanged(this.selectedPos);
      this.selectedPos = selectedPos;
      // Se marca la nueva posición
      notifyItemChanged(this.selectedPos);
   }
}
```

9. En mainActivity se crea el escuchador y se registra sobre el adaptador utilizando el setter creado. Es en este método onClick donde realmente se gestiona la acción que se asocia con la pulsación. Es el ejemplo se consigue la posición del elemento pulsado y se muestra un Toast con la información de esa posición. No obstante, se podría realizar cualquier acción, como por ejemplo lanzar una activity enviándole esos datos.

```
View.OnClickListener listener = new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      int pos=rv.getChildAdapterPosition(view);
      miAdaptador.setSelectedPos(pos);
      // Si marco una posición
      if (miAdaptador.getSelectedPos()>=0){
         Toast.makeText(MainActivity.this, "Pos: " +pos+"\n" + sistemas.
                  get(pos).getNombre() + "\n" +sistemas.get(pos).getAno(),
                                          Toast.LENGTH_SHORT).show();
      }
   }
};
// Se establece el escuchado del RecycleView
miAdaptador.setOnClickListener(listener);
```

9. Apéndice 2: ListView con varios datos por entrada. Adapter a medida.

En este caso vamos a realizar un ListView de forma que cada fila tenga más información y distribuida en más componentes que en el caso anterior. Creamos un nuevo proyecto denominado ListaAMedida.

Vamos a mostrar un listado de sistemas operativos o distribuciones con una imagen y año de salida del mismo. Usaremos para ello la arquitectura MVC.

Para ello lo primero es definir el Modelo, es decir, la estructura de datos que va a manejar lo mismo en la colección o en la base de datos que usemos.

Puede ser algo así:

```
public class SistemaOperativo {
   private String nombre;    // Nombre del sistema operativo
   private String ano;        // Año de salida
   // Int que representa al drawable correspondiente en el fichero R
   private int logo;

   public String getNombre() {
      return nombre;
   }

   public void setNombre(String nombre) {
      this.nombre = nombre;
   }

   public String getAno() {
      return ano;
   }

   public void setAno(String ano) {
      this.ano = ano;
   }

   public int getLogo() {
      return logo;
   }

   public void setLogo(int logo) {
      this.logo = logo;
   }

   public SistemaOperativo(String nombre, String ano, int logo) {
      super();
      this.nombre = nombre;
      this.ano = ano;
      this.logo = logo;
   }
}
```

Una vez hecho el modelo hacemos la Vista, mediante XML establecemos un layout que será el layout de cada una de las filas. Le llamaremos fila.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
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="wrap_content">
```

```
    <ImageView
        android:id="@+id/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/txtSistema"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:textSize="24sp"
        app:layout_constraintBottom_toTopOf="@+id/txtFecha"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/logo"
        app:layout_constraintTop_toTopOf="@+id/logo" />

    <TextView
        android:id="@+id/txtFecha"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="@+id/logo"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/logo"
        app:layout_constraintTop_toBottomOf="@+id/txtSistema" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

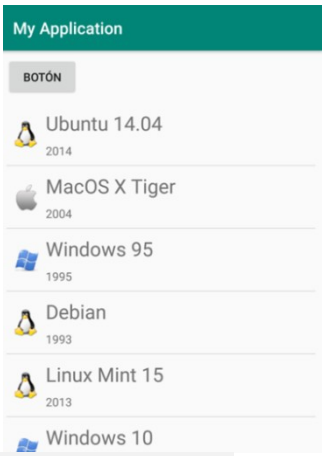
Finalmente haremos el Adapter que es nuestro controlador en la arquitectura MVC. Para poder hacer un adaptador se hereda de la clase BaseAdapter.

```
public class  AdaptadorSistemas extends BaseAdapter {
     ArrayList<SistemaOperativo> lista;
     LayoutInflater inflador;

     public AdaptadorSistemas(Context contexto, ArrayList<SistemaOperativo>
lista) {
          this.inflador = LayoutInflater.from(contexto);
          this.lista = lista;
     }

     @Override
     public int getCount() {
          return lista.size();
     }
```

```
    // Devuelve el elemento asociado con la posición en el ListView
    @Override
    public Object getItem(int position) {
       return lista.get(position);
    }
   // A partir de la posición en la ListView da la posición en el arraylist
    @Override
    public long getItemId(int position) {
          return position;
    }
    // Sobrescribimos el método getView() de nuestro Adapter para utilizar
    // el Contenedor
    @Override
    public View getView(int position, View view, ViewGroup parent) {
          Contenedor contenedor = null; // Aquí se gestionan las filas
          /* Si la vista ya existe en pantalla no se crea otra si no que se reutiliza
             la que hay. Así si es null significa que hay que crearla,
            si no es null, se coge una existente  */
          if (view == null){
                // "Inflamos" la vista
                view = inflador.inflate(R.layout.fila, null);

                // Creamos el contenedor e instanciamos los recursos
                contenedor = new Contenedor();
                contenedor.logotipo =(ImageView) view.findViewById(R.id.logo);
                contenedor.txtSistema = (TextView) view.findViewById(
                                                          R.id.txtSistema);
                contenedor.txtAno=(TextView) view.findViewById(
                                                            R.id.txtFecha);

                // Asignamos el Contenedor a la vista
                view.setTag(contenedor);
          }
          else  contenedor=(Contenedor)view.getTag();   // obtengo el contenedor
          // Obtenemos el dato a mostrar y lo colocamos en los componentes
          SistemaOperativo so = (SistemaOperativo) getItem(position);
          contenedor.logotipo.setImageResource(so.getLogo());
          contenedor.txtSistema.setText(so.getNombre());
          contenedor.txtAno.setText(so.getAno());
          return view;
    }
    // Clase para gestionar los componentes de la fila
    class Contenedor {
          ImageView logotipo;
          TextView txtSistema, txtAno;
    }
}
```

Quizá en esta parte lo más complejo de entender es que por un tema de eficiencia, si Android tiene una lista de 40 elementos y en pantalla solo se ven 7 filas, cuando el usuario hace scroll no se crean nuevas filas si no que se reutilizan las que van desapareciendo y se colocan en la posición correcta con los datos adecuados.

De esta forma en el getView, que es de donde se obtiene la fila con los datos, se comprueba que no haya ninguna fila disponible, en ese caso se llama al inflater. Si la hay se coge dicha fila. Finalmente se colocan los datos.



Finalmente en nuestro MainActivity quedaría de la siguiente manera:

```
public class MainActivity extends AppCompatActivity{
   AdaptadorSistemas adapter;
   ArrayList<SistemaOperativo> sistemas;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      ListView lv=findViewById(R.id.listview);
sistemas= new ArrayList<SistemaOperativo>();
sistemas.add(new SistemaOperativo("Ubuntu 14.04", "2014", R.drawable.tux));
sistemas.add(new SistemaOperativo("MacOS X Tiger", "2004", R.drawable.apple));
sistemas.add(new SistemaOperativo("Windows 95", "1995", R.drawable.windows));
sistemas.add(new SistemaOperativo("Debian", "1993", R.drawable.tux));
sistemas.add(new SistemaOperativo("Linux Mint 15", "2013", R.drawable.tux));
sistemas.add(new SistemaOperativo("Windows 10", "2016", R.drawable.windows));
sistemas.add(new SistemaOperativo("Android", "2006", R.drawable.android));
sistemas.add(new SistemaOperativo("iOS 8", "2014", R.drawable.apple));
sistemas.add(new SistemaOperativo("Windows XP", "2001", R.drawable.windows));
sistemas.add(new SistemaOperativo("Elementary OS", "2014", R.drawable.tux));
sistemas.add(new SistemaOperativo("MacOS 9", "1999", R.drawable.apple));

      adapter=new AdaptadorSistemas(this, sistemas);
      lv.setAdapter(adapter);
      lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
         @Override
         public void onItemClick(AdapterView<?> adapterView,
                                 View view, int position, long id) {
            SistemaOperativo so = (SistemaOperativo) adapter.getItem(position);
            Toast.makeText(getApplicationContext(), "Borrada posición: " +
               position + " -> " + so.getNombre(), Toast.LENGTH_SHORT).show();
            sistemas.remove(adapter.getItem(position));
            adapter.notifyDataSetChanged();
         }
      });
   }

   /* En caso de que herede de ListView
   @Override
      protected void onListItemClick(ListView l, View v, int position, long id) {
         SistemaOperativo so=(SistemaOperativo)adapter.getItem(position);
         Toast.makeText(this, "Borrada posición: "+position+" -> "+
                        so.getNombre(), Toast.LENGTH_SHORT).show();
         sistemas.remove(adapter.getItem(position));
         adapter.notifyDataSetChanged();
      }
   */
}
```

Ejercicio Adapter a medida:

Modifica el ejemplo anterior para que:

- En las filas que aparece el sistema operativo Windows no se visualice el logo.
- En las filas pares tenga un color de fondo rojo claro.

10.Apéndice 3: Uso de Intents implícitos

El siguiente ejemplo de uso de Intent implícitos está basado en el ejemplo mostrado en:
https://developer.android.com/guide/components/intents-common?hl=es#PickContactDat

que permite conseguir los datos de un contacto determinado.

Ejemplo: Datos de un contacto

Se lanza la lista de contactos desde un botón:

```
Button bu=findViewById(R.id.button);
bu.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType(ContactsContract.CommonDataKinds.Phone.CONTENT_TYPE);
      if (intent.resolveActivity(getPackageManager()) != null) {
         startActivityForResult(intent, 1);
      }
   }
});
```

Se muestran los datos obtenidos:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   if (requestCode == 1 && resultCode == RESULT_OK) {
      // Get the URI and query the content provider for the phone number
      Uri contactUri = data.getData();

      String[] projection = new String[]{
          ContactsContract.CommonDataKinds.Phone.NUMBER,
          ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME
       };

      Cursor cursor= getContentResolver().query(contactUri, projection, null, null, null);

     // If the cursor returned is valid, get the phone number
     if (cursor != null && cursor.moveToFirst()) {
          int index = cursor.getColumnIndex(
                                    ContactsContract.CommonDataKinds.Phone.NUMBER);

          Toast.makeText(this, cursor.getString(index)+
                                  " "+cursor.getString(1), Toast.LENGTH_SHORT).show();

     }

     cursor = getContentResolver().query(contactUri, null, null, null, null);

     // If the cursor returned is valid, get the phone number
     if (cursor != null && cursor.moveToFirst()) {
         for (int i=0;i < cursor.getColumnCount(); i++){
            if (cursor.getType(i)==Cursor.FIELD_TYPE_STRING) {
               Log.i("Datos ", cursor.getColumnName(i) + " -> " + cursor.getString(i));
            }
         }
      }
   }
}
```

11.Bibliografía

1. Marerial Design
2. Intents
    1. Uso de Intents
    2. Clase Intent
    3. Devolución de un valor por una activity secundaria
    4. API de Activity Result
    5. RegisterForActivityResult
    6. ActivityResultContracts
    7. Intents implicitos
        1. La lista completa de acciones
        2. Ejemplos y explicación de utilización de intents implícitos
        3. Lista extensa de intents comunes
        4. Lista completa de tipos de URIs
        5. Ejemplo de consultar datos de un contacto
3. Menús
    1. Guia de desarrollo de menús
    2. Toolbar
    3. Recurso de menú
    4. Lista de iconos para los menús
4. RecyclerView
    1. RecyclerView
    2. Paquete RecyclerView
    3. Clase RecyclerView
    4. LayoutManager
    5. LinearLayoutManager
    6. GridLayoutManager
    7. StaggeredGridLayoutManager
    8. ItemDecoration
    9. Ejemplo de ItemDecoration
    10. ItemAnimator
5. Spinner
6. Menú contextual
7. Menús emergentes (Popup menús)

Ejercicio

6 Realiza una aplicación de gestión de una base de datos de películas de estreno. La información por cada película que se almacenará será:

Titulo, director, duración, sala de cine, sinopsis, fecha de estreno, imagen con la portada, imagen con la clasificación por edades:

http://es.wikipedia.org/wiki/Clasificación_por_edades_(cine)

La activity principal constará de un RecyclerView en el que se mostrará el titulo de la película, el director y las imágenes de calificación por edades y portada (ver propiedades adjustViewBounds y scaleType). Tendrá una sección fija en la parte superior que mostrara la película seleccionada. Añade un botón en la parte inferior que mostrara o ocultara la ActionBar.

Además dispondrá de un menú con tres opciones: Listado completo, Favoritos y Añadir estreno. Listado completo deberá aparecer siempre como una imagen en la ActionBar.

Listado completo tendrá un RecyclerView de forma que en cada fila aparezca: caratula, director, fecha de estreno, duración, sala de cine, imagen de calificación por edades (deberá reducirse su tamaño a la mitad en el eje x e y) e icono indicando si es favorita o no.

Al pulsar en una película se mostrará un nuevo activity con la caratula en grande, ocupando la mitad de la pantalla, la sinopsis y como titulo de la activity el titulo de la película. Si se pulsa en la caratula se mostrara el trailer de la película en youtube.

En listado favoritos aparecerá un ListView con selección múltiple que muestra los títulos de las películas, y permitirá indicar si una película es favorita o no.

En añadir se permitirá meter una nueva película con los siguientes componentes: Titulo, Director, duración se harán con EditView. Sala con un Spinner (hay una cantidad fija de salas) y para la clasificación un radiobutton. Fecha de estreno se rellena o con un DatePicker o con un CalendarView que se muestra en un Activity diferente. Mediante el menú se aceptan o se cancelan estas modificaciones. En estas películas se ha de mostrar una caratula genérica indicando que no tienen caratula.

Ejercicios extra

Las aplicaciones deben estar al menos en 2 idiomas

1. Crea una aplicación En la que se permita introducir 2 números en la activity principal y mediante un botón de cálculo pase a otra activity donde de resultados de la suma, resta multiplicación y división de dichos números.

   El botón tendrá el siguiente comportamiento:

   Si se realiza un pulsación normal sobre el mostrará un toast con el siguiente mensaje "Aprieta más tiempo"

   Si se realiza una pulsación larga se mostrara la activity con los resultados.

   Si la operación no se puede realizar (no se introdujeron bien los datos) informa de ello mediante un toast.

2. Haz una aplicación que permita manejar de forma simple una colección de libros. Cada libro tendrá los siguientes atributos: Título, autor, puntuación.

   Al iniciarse muestra en la activity principal el título y autor del primer libro de la colección o nada si esta está vacía. Presentará además cinco botones (siguiente, anterior, nuevo y buscar) con las siguientes características:

   Siguiente: muestra el siguiente libro de la colección (si existe) en la misma activity.

   Anterior: muestra el anterior libro de la colección si existe.

   Nuevo: llama a una activity secundaria donde se puede introducir un nuevo libro y puntuarlo. En dicho activity no se manejará la colección si no que devolverá los datos.

   Borrar: que al llamarlo se borra el libro actual.

   Buscar: que al llamarlo abra un activity implícito con la búsqueda en Google del libro actual.

   Se deberá mostrar un campo de texto, en la activity principal, con el número de libros.

   Implementa la posibilidad de modificación de un libro.