
	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

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	3
2.3.2.	Acciones en la activity inicial.....	4
2.4.	Intents implícitos.....	7
3.	Menús / ActionBar	9
3.1.	Definir el menú.....	9
3.2.	Activar el menú	10
3.3.	Gestionar la pulsación de una opción	10
3.3.1.	Gestionar opción de menú Switch.....	11
3.4.	Cambiar las propiedades de ActionBar	11
3.4.1.	Título y subtítulo de la activity	12
3.4.2.	Añadir un icono al ActionBar.....	12
3.4.3.	Botón volver	12
3.4.4.	Ocultar ActionBar	12
3.5.	Toolbar.....	13
4.	AdapterViews	14
4.1.	ListView.....	14
4.2.	ListView con selección.	15
4.3.	RecyclerView	16
4.3.1.	Creación de un RecyclerView	17
4.3.2.	Elementos opcionales.....	23
4.3.3.	Pulsación de un elemento del RecyclerView.....	24
4.3.4.	Cambios en el contenido del RecyclerView	25
5.	Spinner	26
6.	Menú contextual.....	27
7.	Menús emergentes.....	29
8.	Apéndice 1: Pulsación alternativa de un elemento del RecyclerView.....	31
9.	Apéndice 2: ListView con varios datos por entrada. Adapter a medida.	33

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 *Material 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 interactuar 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

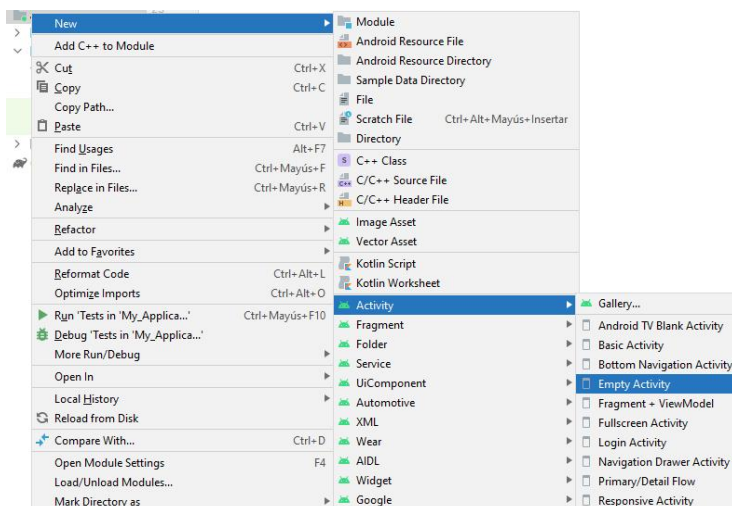
En este punto se verá como desde la *activity* actual pulsando un botón, podría ser con cualquier otra acción, se lance una nueva *activity*. En este punto solo se lanzará la *activity* sin pasarle datos ni recoger los datos que pueda enviar de vuelta.

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 se pulsa 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*.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Gestionaremos la pulsación del botón de 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* creado.

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 *intent* definido en el punto anterior. A estos se le pueden insertar una colección de pares clave-valor denominada *Bundle*. El *intent*, con los datos 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 secundaria y esta visualizará 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);
intent.putExtra("NOMBRE", editNombre.getText().toString()); // se añade al intent el dato con clave NOMBRE y
startActivity(intent);                                     // valor el contenido del editText editNombre
```

El método *putExtra* se encarga de añadir el valor al *intent* con una clave de tipo texto sensible al caso.

- En la *activity* secundaria, se recupera el *intent* enviado con 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
protected void onCreate(Bundle savedInstanceState) { // Método onCreate de la activity secundaria
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_secundaria);

    Intent intent=getIntent(); // Recuperamos el intent enviado desde la activity primaria
    final TextView nombre=findViewById(R.id.textViewMensaje); // TextView de la activity secundaria
    nombre.setText("Hola " + intent.getStringExtra("NOMBRE")); // Se establece el valor del TextView con el dato
                                                                // pasado en el intent. En este caso NOMBRE.
}
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 llamado. 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 para lanzar la *activity* secundaria de manera que nos pueda devolver datos y una vez devueltos que pasos tenemos que realizar para capturar estos valores.

2.3.1. Acciones en la activity secundaria

Los pasos que deberemos realizar en la *activity* secundaria son:

- 1 • Crear un *Intent* con el que se devolverán los datos.
- 2 • Introducir en el *intent* todos los datos que se quieran devolver.
- 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*. Con esto la aplicación vuelve a la *activity* que la lanzo.

Finish provoca que se finalice la *activity* secundaria invocándose, 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, un color de estrellas amarillo 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("ESTADO", ratingBar.getRating());
        3 setResult(RESULT_OK, intentSec);
        4 finish();
    }
});
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 Result¹ 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

Debido a que cuando se lanza una actividad para obtener un resultado, ya sea una *activity* de tu aplicación como otra aplicación diferente (por ejemplo la cámara) puede ser que tu aplicación se destruya, por ejemplo por motivos de memoria, y luego se reconstruya.

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> launchActivity = registerForActivityResult (
2      new 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                  Toast.makeText(getApplicationContext(), "Valor: " + intent.getStringExtra("val"), Toast.LENGTH_SHORT).show();
8              }
9          }
10     }
11 );


```

Donde:

- Línea 1: `launchActivity` 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 tiene de `registerForActivityResult` tiene dos parámetros.
 - El primero es de tipo `ActivityResultContract`. Este tiene dos parámetros: uno de entrada y otro de salida. Se puede definir un `ActivityResultContract` personalizado o usar uno de los disponibles en `ActivityResultContracts`. En este caso usamos `StartActivityForResult` que está presente en los predefinidos de Android que toma como parámetro de entrada un *Intent* y produce como salida un objeto de tipo `ActivityResult`.
 - El Segundo es la interfaz `ActivityResultCallback` cuyo tipo debe coincidir con el tipo de salida que devuelve `ActivityResultContract`. Debemos implementar el método `onActivityResult` que tiene

¹ <https://developer.android.com/training/basics/intents/result#java>

² [registerForActivityResult](#)

	RAMA:	Informática	CICLO:	Desenvolvemeto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

como parámetro un objeto del tipo indicado antes. Este método gestiona 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 el tipo de resultado producido.
- 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 se lanzas una actividad para obtener los valores que devuelva. Para el ejemplo gestionaremos esto dentro del *listener* de un botón.

```

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         lanzaActivity.launch(intent);
7     }
8 });

```

El único elemento diferente con lo visto anteriormente es la línea 6 en la que usamos el método *launch* 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.

```

int MIREQUESTCODE=1; // este valor es un valor único para identificar cada activity que se invoque
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 lanzar 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.getIntExtra("ESTADO", 0)){
            case 0: Log.i(TAG, "Por los suelos"); break;

```

³ <https://developer.android.com/reference/androidx/activity/result/contract/ActivityResultContracts>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

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;
}
}
}

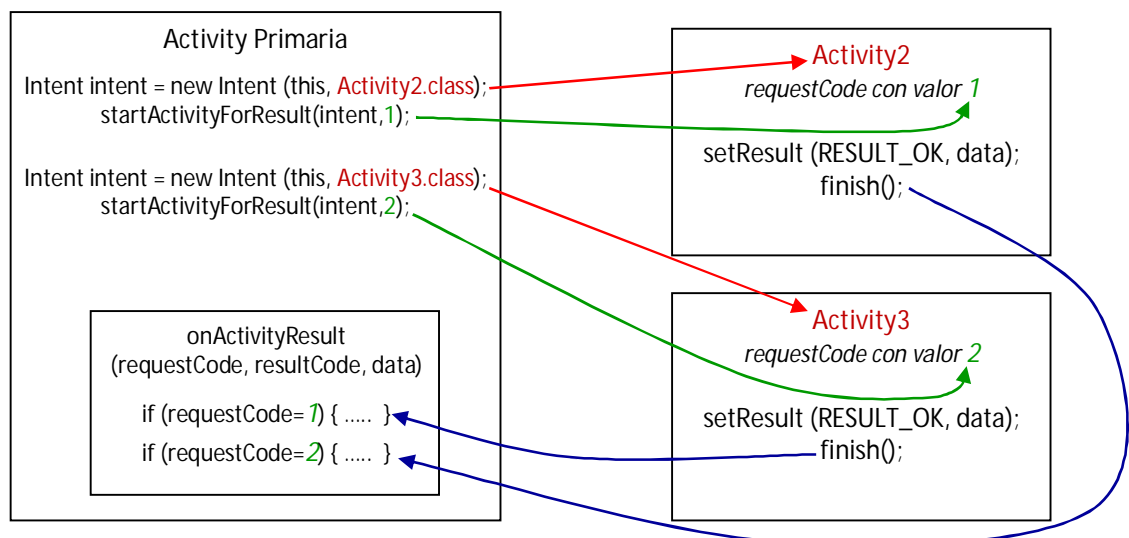
```

Los parámetros de este método son:

- o requestCode: se utiliza para identificar cual es la *activity* que está devolviendo el resultado ya que pueden ser varias.
- o resultCode se utiliza para comprobar el éxito o no de las operaciones realizadas en la otra *activity*.
- o 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 del *editText* y el valor establecido en el *RatingBar*.

En la segunda *activity* se visualizará este valor en un *ratingBar* con un paso de *rating* de 0.5. Existirá además un botón que al pulsarse devuelve el valor establecido a la actividad primaria mostrándose ese valor tanto en el *textView* a la derecha del *ImageButton* como en el *RatingBar* y en el *logcat*.

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

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

2.4. Intents implícitos

Para lanzar nuevas *activities* via *intent* existen dos opciones:

- De forma explícita: indicando la actividad a lanzar tal y como se ha hecho en el caso anterior.
- De forma implícita: de forma que a partir del *Intent* y de los datos que se pasan como parámetro *Android* deduzca componente de otra aplicación 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.btnPosicion);
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));
        if (intent.resolveActivity(getPackageManager()) != null) // Si existe una Activity que es capaz de gestionar los datos
            startActivity(intent); // Se lanza
    }
});
```

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

Además podemos indicar que una *activity* nuestra se lance si un *intent* requiere determinados criterios usando los *intent filters*.

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 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 este caso los datos son de tipo *geo* por lo que *Android* lanza la *activity* predefinida para visualizar mapas.

Ejemplo: Llamadas a Intents implícitos.

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

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"));
startActivity(intent);
```


	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Además se pueden conseguir datos del *intent* lanzado de la misma forma que la vista anteriormente.

- Se lanza el *intent* con *startActivityForResult*
- Se recogen los datos en *onActivityResult*.

Para mostrar su funcionamiento veremos un ejemplo basado en el ejemplo mostrado en:

<https://developer.android.com/guide/components/intents-common?hl=es#PickContactData> 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) {
        Uri contactUri = data.getData(); // Get the URI and query the content provider for the phone number
        String[] projection = new String[]{ContactsContract.CommonDataKinds.Phone.NUMBER,
                                           ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME};
        Cursor cursor = getContentResolver().query(contactUri, projection, null, null, null);
        if (cursor != null && cursor.moveToFirst()) { // If the cursor returned is valid, get the phone number
            int index = cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER);
            Toast.makeText(this, cursor.getString(index)+" "+cursor.getString(1), LENGTH_SHORT).show();
        }
        cursor = getContentResolver().query(contactUri, null, null, null, null);
        if (cursor != null && cursor.moveToFirst()) { // If the cursor returned is valid, get the phone number
            for (int i=0; i<cursor.getColumnCount(); i++){
                if (cursor.getType(i)==Cursor.FIELD_TYPE_STRING) {
                    Log.i("Datos contacto", cursor.getColumnName(i) + " -> " + cursor.getString(i));
                }
            }
        }
    }
}
```

Ejercicio 2

Continúa 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.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

3. Menús / Actionbar ^{4 5 6}

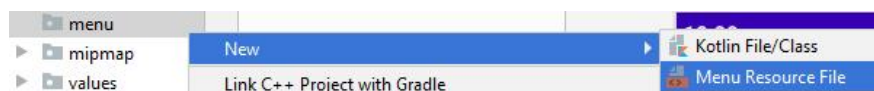
En este punto explicaremos como crear menús para nuestra aplicación. Como otros elementos, se pueden definir tanto mediante un fichero *XML* como mediante código. Nosotros veremos solo su definición usando ficheros *XML*.

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 posteriormente definir dentro de él un nuevo fichero de recursos de menú: *Menu resource file*.

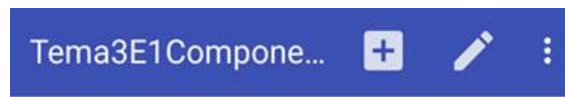


Una vez creado el fichero *XML* asociado al menú podremos editarlo directamente en el propio fichero *XML* o mediante el asistente gráfico tal y como se hace con los *layouts*.

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 serían los submenús.

Las 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.
- **showAsAction**: indica como el elemento *item* 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).



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>

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>

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 contiene el menú.

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

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

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.mnuevo:
            Toast.makeText(this, "Nuevo", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.mnuevob1:
            Toast.makeText(this, "Submenú 1", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.mborrar:
            Toast.makeText(this, "Borrar", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.meditar:
            Toast.makeText(this, "Editar", Toast.LENGTH_SHORT).show();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

- Especificamos el nombre de un método en la propiedad *onClick* de un *item*. Este método tiene que estar implementado en la *activity* que visualiza el menú:

Por ejemplo en el *item* identificado por *R.id.meditar* establecemos la siguiente propiedad: *android:onClick="menuEditar"*. Por lo que tendríamos que tener implementado el método *menuEditar*:

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

El 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*).

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

3.3.1. Gestionar opción de menú Switch

Para añadir un botón *Switch* al menú 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 fichero *XML* del menú donde se establece, en la propiedad `app:actionLayout="@layout/switch_item"`, el *layout* que se visualizara para ese *item*.

Es en los componentes definidos en ese *layout* donde editaremos las propiedades individuales de los elementos, en nuestro caso el *Switch*. Por ejemplo estableceremos su *id* o su propiedad *text* que será el texto que muestre 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.

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

Un ejemplo es el siguiente:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    // Id de item del menú con el layout que se quiere usar id del componente
    final Switch switchView = menu.findItem(R.id.app_bar_switch).getActionView().findViewById(R.id.idswitch);
    switchView.setChecked(true); // Se puede establecer su estado inicial
    switchView.setText("Texto prueba"); // Se puede cambiar el texto inicial mostrado mediante código
    switchView.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
            Toast.makeText(MainActivity.this, "Estado: "+b, LENGTH_SHORT).show();
            switchView.setText(b?"Activado":"Desactivado");
        }
    });
    return true;
}
```

Ejercicio 3:

Continúa 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 una 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.

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

3.4.1. Título y subtítulo 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 subtítulo con:

```
actionBar.setDisplayHomeAsUpEnabled(false);
```

3.4.2. Añadir un icono al ActionBar

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

```
actionBar.setDisplayHomeAsUpEnabled(true);
actionBar.setDisplayUseLogoEnabled(true);
actionBar.setLogo(android.R.drawable.ic_dialog_alert);
```



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);
```

Para gestionar la pulsación tenemos dos opciones:

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

```
<activity android:name=".Main2Activity"> <!--El botón se ha añadido a la actividad Main2Activity -->
    android:parentActivityName="com.example.javi.myapplication.MainActivity" >
    <!--Se añade para dar soporte a versión inferiores a Android 4.1 (API level 16) -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.javi.myapplication.MainActivity" />
</activity>
```

- 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);
}
```

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

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


3.4.4. Ocultar ActionBar

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

```
actionBar.hide(); // actionBar.show();
```

Otra opción es evitar que la *ActionBar* 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 una activity. El siguiente ejemplo muestra un tema de los predefinidos de *Android* sin *ActionBar*:

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

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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

Ejercicio 4:

Continúa 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`.
- Modifica el título de la `ActionBar` primaria y como subtítulo se debe mostrar el valor de la `SeekBar`. Este valor debe estar siempre actualizado.

3.5. Toolbar⁷

En las últimas versiones de *Android* se recomienda, por temas de compatibilidad, no usar la *ActionBar* 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 *ActionBar* que se añade por defecto indicando, en el archivo *AndroidManifest*, que se haga uso de un estilo sin *ActionBar* como hemos visto en el punto anterior. Existen dos posibilidades:

o Si solo alguna *Activity* va a usar *Toolbar* se puede indicar en la definición de la *Activity* en particular:

```
<activity android:name=".NombreActivity" android:theme="@style/AppTheme.NoActionBar">
```

o Se puede indicar en la definición de la aplicación si todas las *Activities* van a hacer uso de *Toolbar*.

```
<application android:theme="@style/AppTheme.NoActionBar">
```

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`.

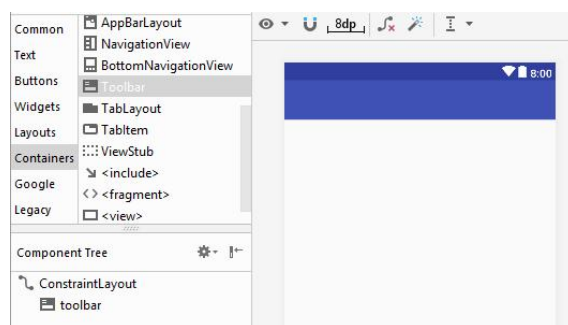
6. Para trabajar con la barra de herramientas se puede conseguir la *ActionBar* y usar los métodos y funciones indicados en puntos anteriores:

```
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 cláusula *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/>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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:

- Un *ListView* puede colocarse sobre cualquier *layout* como todos los demás componentes.

Añadiremos un botón y debajo un componente *ListView* (deberemos asegurarnos que tenga un id asignado).

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, en general, puede ser de cualquier tipo dato ya que en cada fila del *ListView* lo que se visualizara es la cadena que devuelve la aplicación de *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<String>(); // Fuente de datos
    ArrayAdapter<String> adapter; // Adaptador para el ListView

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView 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", "Windows10", "Solaris", "Linux", "OS/2", "Android", "iOS", "FreeBSD", "Haiku" };
        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 los datos.
        adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, lista);
        adapter.addAll("Windows 8", "Unix"); // se añaden elementos mediante código
        lv.setAdapter(adapter); // se asigna el adaptador al ListView

        // Se gestiona la pulsación de un elemento del ListView
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int posición, long id) {
                // Se obtiene los datos de la fila del ListView seleccionada de dos formas diferentes
                String cad="Posición: "+posición + "\n" + adapter.getItem(posición) + "\n" + lista.get(posición);
                Toast toast = Toast.makeText(getApplicationContext(), cad,Toast.LENGTH_SHORT);
                toast.setGravity(Gravity.CENTER|Gravity.CENTER, 0, 0);
                toast.show();
            }
        });
    }
}
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

- Cambiando la herencia de la activity principal para que herede de *ListActivity* y eliminado la llamada a *setContentView*, si no fallará (ya no se representa 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 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<String>(); // Fuente de datos
    ArrayAdapter<String> adapter;

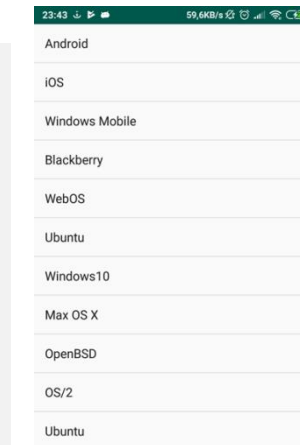
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView lv = findViewById(R.id.listView);

        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", "Windows10", "Solaris",
            "Linux", "OS/2", "Android", "iOS", "FreeBSD", "Haiku" };

        for (int i = 0; i < sistemas.length; ++i) {
            lista.add(sistemas[i]);
        }

        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, lista);
        adapter.addAll("Windows 8", "Unix"); // añadimos elementos mediante código
        setListAdapter(adapter);

        @Override
        protected void onListItemClick(ListView l, View v, int position, long id) {
            super.onListItemClick(l, v, position, id);
            String cad = "Posición: " + position + "\n" + adapter.getItem(position) + "\n" + lista.get(position);
            Toast toast = Toast.makeText(getApplicationContext(), cad, Toast.LENGTH_SHORT).show();
        }
    }
}
```



4.2. ListView con selección.

Ahora veremos cómo marcar los elementos que se seleccionan 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:

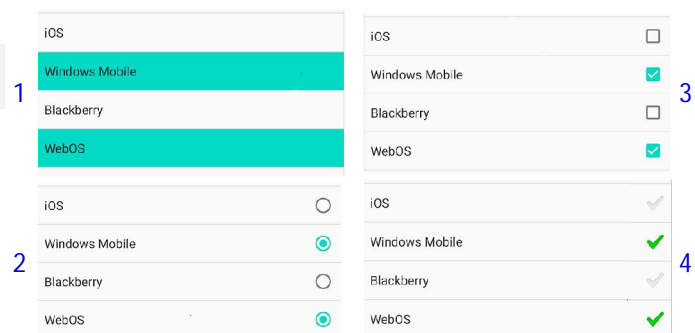
```
lv.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE); // El ListView está como componente
this.getListView().setChoiceMode(ListView.CHOICE_MODE_MULTIPLE); // La clase hereda de ListView
```


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*



	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Para eliminar los elementos seleccionados creamos un menú en el que a una opción se establece el atributo *onClick* a *clickEliminar*. Será en este método donde se realice el borrado de los elementos.

Su código es el siguiente:

```
public void clickEliminar(Menuitem item) {
    @SuppressWarnings("unchecked")
    ArrayAdapter<String> adapter=(ArrayAdapter<String>) lv.getAdapter(); // Obtenemos el adaptador

    // Recorremos la lista en orden inverso para borrar sin problema
    for (int i= lv.getCount() - 1; i >= 0; i--) {
        if (lv.isItemChecked(i)) lista.remove(i); // Si un elemento está marcado lo eliminamos
    }
    lv.getCheckedItemPositions().clear(); // Eliminamos las marcas de la lista
    adapter.notifyDataSetChanged(); // Informamos que ha habido cambios en los datos del adaptador
    // para actualizar la lista
}
```

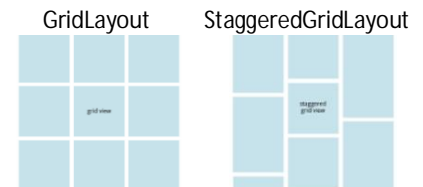
En el ejemplo se usa *lista.remove(i)* para eliminar un elemento, siendo *i* la posición de un elemento seleccionado.

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 la eficiencia. *RecyclerView* pretende substituir a *ListView*.

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 él.
- 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.
- LayoutManager¹¹: 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 cuadrícula con celdas del mismo tamaño en la misma fila o columna.
 - *StaggeredGridLayoutManager*¹⁴ (2D): muestra los datos en una cuadrícula con celdas de tamaño variable o incluso se puede crear una distribución propia.
- ItemDecoration¹⁵: permite definir aspectos visuales de *RecyclerView* como pueden los divisores. No es obligatorio.
- ItemAnimator¹⁶: 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 nuestra 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>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

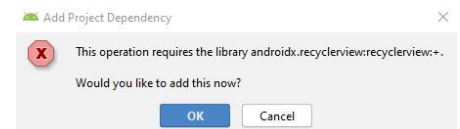
Android no crea todos los elementos que se van a visualizar en el *RecyclerView* sino que solo crea, de forma dinámica, los elementos que se están visualizando en el *RecyclerView* añadiendo alguno más para visualizar los próximos elementos que pueden aparecer, por ejemplo al realizar un *scroll*. Con 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 generada el siguiente código *XML*:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/mi_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_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. Debemos 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 al drawable correspondiente 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;
    }
}
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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

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

4. La visualización de cada posición del *RecyclerView* se lleva a cabo mediante un *Layout*. Este deberá poseer los componentes necesarios para mostrar la información que se desee. En nuestro ejemplo el *Layout* se llamará llamaremos *elemento_individual.xml* y tendrá un *ImageView* y dos *TextView*.


```
<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/elemento"
    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="20sp"
    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="#FF0000"
    android:textSize="14sp"
    app:layout_constraintBottom_toBottomOf="@+id/imageView"
    app:layout_constraintStart_toStartOf="@+id/imageView" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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. Tendrá como atributo una colección con los datos que se visualizarán en el *RecyclerView*. En nuestro ejemplo se usará un *ArrayList* de sistemas operativos.

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

- b) Implementar, normalmente dentro del adaptador, una clase que herede de *RecyclerView.ViewHolder* que representa cada uno de los elementos que se definen en cada celda del *RecyclerView*. En nuestro caso el nombre es *MyViewHolder* pero puede ser cualquiera.

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

    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
        // en el layout definido en 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);
        }
    }
}
```

- c) Para que la clase *MiAdaptador* creada 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()*: cuando se rehusa un *ViewHolder* se encarga de actualizar su contenido con los datos que le corresponde a su posición.
- ❖ *onItemCount*: muestra el tamaño de la colección de datos que va a mostrar *RecyclerView*.

```
public class MiAdaptador extends RecyclerView.Adapter<MiAdaptador.MyViewHolder> {
    ArrayList<SistemaOperativo> sistemasOperativos; // puede ser cualquier estructura de datos
    // Crea nuevos elementos expandiendo el layout definido en el fichero R.layout.elemento_individual
    // 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.elemento_individual, parent, false);
        MyViewHolder mvh = new MyViewHolder(elemento); // return new MyViewHolder(elemento);
        return mvh ;
    }

    // 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);
    }
}
```


	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

holder.nombre.setText(so.getNombre());
holder.year.setText(so.getAno());
holder.logo.setImageResource(so.getLogo());
}
// Número de elementos en la colección de datos. Será el número de elemento que se tendrá el RecyclerView
@Override
public int getItemCount() {
    return this.sistemasOperativos.size();
}

```

d) Al adaptador se le añade un constructor con el que suministrarle datos.

```

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

```

Por lo que el adaptador completo quedaría:

```

public class MiAdaptador extends RecyclerView.Adapter<MiAdaptador.MyViewHolder> {
    ArrayList<SistemaOperativo> sistemasOperativos; // puede ser cualquier estructura de datos
    // Constructor
    public MiAdaptador(ArrayList<SistemaOperativo> sistemasOperativos) {
        this.sistemasOperativos = sistemasOperativos;
    }

    public class MyViewHolder extends RecyclerView.ViewHolder {
        // Elementos que queremos mostrar en el 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 en el layout definido en 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);
        }
    }

    // Crea nuevos elementos expandiendo el layout definido en el fichero R.layout.elemento_individual
    // 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.elemento_individual, parent, false);
        MyViewHolder mvh = new MyViewHolder(elemento); // return new MyViewHolder(elemento);
        return mvh ;
    }

    // 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.nombre.setText(so.getNombre());
    }
}

```



	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

holder.year.setText(so.getAno());
holder.logo.setImageResource(so.getLogo());
}

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

```

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

```

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

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

        sistemas= new ArrayList<SistemaOperativo>();
        for (int i=1;i<=9;i++) {
            sistemas.add(new SistemaOperativo("Ubuntu 14.04 "+i, "2014", R.drawable.ubuntu14));
            sistemas.add(new SistemaOperativo("MacOS X Tiger "+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 OS "+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));
        }

        MiAdaptador 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 los datos en el *RecyclerView*. En un mismo *RecyclerView* cambiando el *LayoutManager* los elementos se distribuirán de forma totalmente distinta. Como se comentó 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;
```

- o *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);

```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

El primero creara un *LinearLayoutManager* 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.

- o *GridLayoutManager*: los elementos se distribuyen en forma de cuadrícula, **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); // es vertical
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 o columnas.
- ❖ 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.

- o *StaggeredGridLayoutManager*: los elementos se distribuyen en forma de cuadrícula 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 tipo de objeto tendremos unos métodos u otros. Por ejemplo en el caso de *GridLayoutManager* o del *StaggeredGridLayoutManager* tendríamos el método *setSpanCount* que mediante código permite cambiar el número de filas si el Layout tiene una orientación horizontal o el número de columnas si es vertical.

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);
```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 la orientación de *LayoutManager* y se establece el divisor correspondiente.

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

StaggeredGridLayoutManager se debe sustituir por el *LayoutManager* 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;
        if (parent.getChildAdapterPosition(view) == 0) // Add top margin only for the first item to avoid
            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 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);
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 añadirselo 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) {
    if (nuevaPos ==this.selectedPos){ // Si se pulsa sobre el elemento marcado
        notifyItemChanged(this.selectedPos); // Se avisa al adaptador para que desmarque esa posición
        this.selectedPos=RecyclerView.NO_POSITION; // Se establece que no hay una posición marcada
    } else { // El elemento pulsado no está marcado
        if (this.selectedPos >=0 ) notifyItemChanged(this.selectedPos); // Si ya hay otra posición marcada se desmarca
        this.selectedPos = nuevaPos; // Se actualiza la nueva posición a la posición pulsada
        notifyItemChanged(this.selectedPos); // Se marca la nueva posición
    }
}
```


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.elemento);
```

4. En constructor de la clase que representa cada elemento del *RecyclerView*, *MyViewHolder* en nuestro ejemplo, establecemos un listener para gestionar la pulsación de cada elemento (cada celda) del *RecyclerView*.

```
public MyViewHolder(View viewElemento, Context context) {
    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) {
            setSelectedPos(getAdapterPosition()); // getAdapterPosition devuelve la posición del view en el adaptador
            if (selectedPos>RecyclerView.NO_POSITION){ // Si hay una posición marcada muestro el Toast
                Toast.makeText(view.getContext(), sistemasOperativos.get(selectedPos).getNombre(),
                    Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

4.3.4. Cambios en el contenido del RecyclerView

Al igual que sucedía con *ListView* cuando se realizan modificaciones en el 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 válidos por lo que fuerza al *LayoutManager* a actualizar y recolocar todos estos elementos.

Una vez realizados los cambios existe la posibilidad de moverse a una posición determinada, indicada mediante el parámetro `posición`, mediante los 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 menú puede ser:



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();


    if (miAdaptador.getSelectedPos()>=0){
        switch (item.getItemId()) {

            case R.id.madd: // añadir: se añade un nuevo elemento con la imagen new_item
                sistemas.add(pos,new SistemaOperativo("prueba" + pos, String.valueOf(pos), R.drawable.new_item));
                miAdaptador.notifyItemInserted(pos);
                miAdaptador.notifyItemChanged(pos+1);
                break;

            case R.id.mdel: // borrar
                sistemas.remove(pos);
                miAdaptador.notifyItemRemoved(pos);
                miAdaptador.setSelectedPos(RecyclerView.NO_POSITION);
                break;

            case R.id.mmove: // mover: se mueve el elemento seleccionado a la posición 1 (la primera posición es 0)
                SistemaOperativo aux=sistemas.get(pos);
                int newPos=1;
                sistemas.remove(pos);
                sistemas.add(newPos, aux);
                miAdaptador.notifyItemChanged(pos);
                miAdaptador.notifyItemMoved(pos,newPos);
                break;

            case R.id.mchange: // modificar: se cambia el nombre del sistema operativo a modificado
                sistemas.get(pos).setNombre("Modificado");
                miAdaptador.notifyItemChanged(pos);
                break;
        }
    }
}
```


	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

        ab.setTitle("Tam: "+sistemas.size());
        ab.setSubtitle("Pos: "+pos);
        return true;
    }else {
        Toast.makeText(this, "Pulsa posición", Toast.LENGTH_SHORT).show();
        ab.setSubtitle("Pos: "+pos);
    }
}

return super.onOptionsItemSelected(item);
}

```

5. Spinner¹⁷

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*.

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

```

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

```

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 usará el siguiente *listener*: *setOnItemSelectedListener*.

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

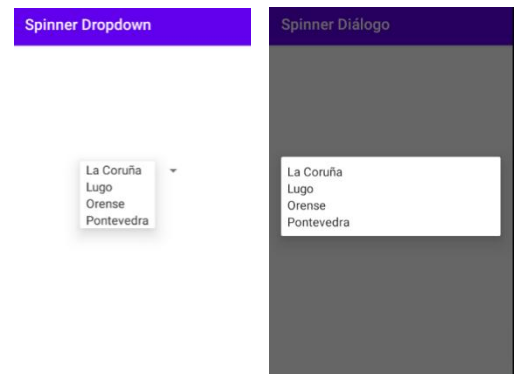
```

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;
        }
    });
}

```



¹⁷ <https://developer.android.com/guide/topics/ui/controls/spinner>

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

        else {
            Toast.makeText(getApplicationContext(), "Has seleccionado: " + adapterView.getItemAtPosition(i).
                                                                    toString(), Toast.LENGTH_SHORT).show();
            setTitle(adapterView.getItemAtPosition(i).toString());
        }
    }

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

```

6. Menú contextual¹⁸

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*.

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

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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.getContextMenuInfo();
    switch (item.getItemId()) {
        case R.id.editar:
            Toast.makeText(this, item.getTitle()+" -> "+info.position, Toast.LENGTH_SHORT).show();
            return true;
        case R.id.copiar:
            Toast.makeText(this, item.getTitle()+" -> "+info.position, Toast.LENGTH_SHORT).show();
            return true;
        case R.id.pegar:
            Toast.makeText(this, item.getTitle()+" -> "+info.position, 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 tenemos 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();
    if (v.getId()==btn.getId()){
        inflater.inflate(R.menu.mboton,menu);
        menu.setHeaderTitle("Menu botón");
    } else {
        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 context MenuInfo) {
        getMenuInflater().inflate(R.menu.conte,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.

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

7. Menús emergentes¹⁹

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.

Un ejemplo asociado con un botón sería:

```
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 {
```

¹⁹ <https://developer.android.com/guide/topics/ui/menus#PopupMenu>


	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 onOptionsItemSelected(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;
}
```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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 añadirselo 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;
public void setOnClickListener(View.OnClickListener listener) { // el nombre del método es indiferente
    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 *RecyclerView*. 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) {
    if (selectedPos==this.selectedPos){ // Si pulso sobre el elemento marcado
        notifyItemChanged(this.selectedPos); // Se avisa para que desmarque esa posición
        this.selectedPos=RecyclerView.NO_POSITION;
    } else { // El elemento pulsado no está marcado
        if (this.selectedPos >=0) notifyItemChanged(this.selectedPos); // Si hay otra posición marcada se desmarca
        this.selectedPos = selectedPos;
        notifyItemChanged(this.selectedPos); // Se marca la nueva posición
    }
}
```

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);
        if (miAdaptador.getSelectedPos()>=0){ // Si marco una posición
```


	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

        Toast.makeText(MainActivity.this, "Pos: " + pos + "\n" + sistemas.get(pos).getNombre() + "\n"
            + sistemas.get(pos).getAno(), Toast.LENGTH_SHORT).show();
    }
}
};
miAdaptador.setOnClickListener(listener); // Se establece el escuchado del RecyclerView

```

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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
    private int logo; // Int que representa al drawable correspondiente 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) {
        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"
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

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;
    }
}

```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```
// 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 = inflater.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.

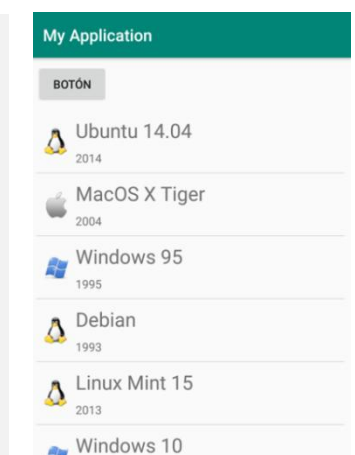
De esta forma en el *getView*, que es de donde se obtiene la fila con los datos, se comprueba que no haya ninguna fila disponible, en ese caso se llama al *inflater*. Si la hay se coge dicha fila. Finalmente se colocan los datos.


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));
    }
}
```



	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

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 onItemClick(AdapterView<?> adapterView, 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.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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:

- Título, 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_\(cine\)](http://es.wikipedia.org/wiki/Clasificación_por_edades_(cine))


- La *activity* principal constará de un *RecyclerView* en el que se mostrará el título 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 título de la *activity* el título 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: Título, 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.

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

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.