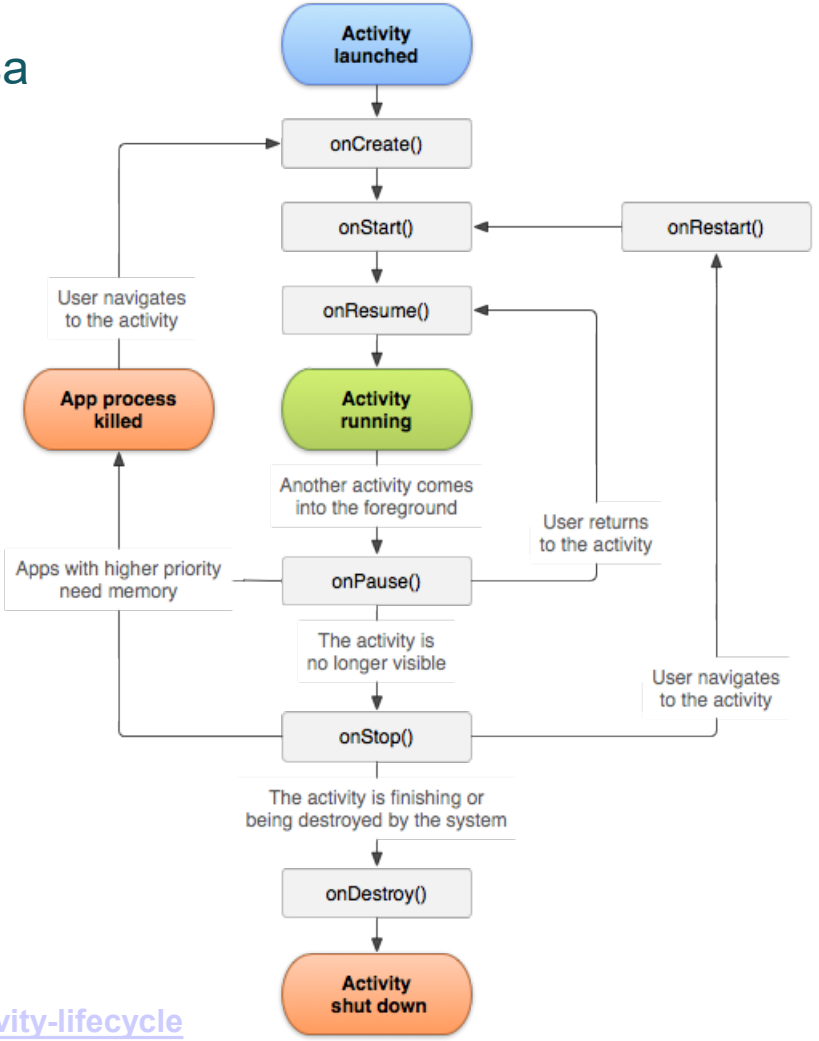
SESIÓN 3

- Ciclo de vida de una actividad
- Guardar y recuperar el estado
- Widgets básicos
- Cuadros de diálogo
- Scroll
- Visibilidad
- Widget Spinner

Ciclo de vida de una actividad

Una actividad tiene un ciclo de vida y pasa por varios estados:

- Created
- Started
- Resumed
- Paused
- Stopped
- Destroyed



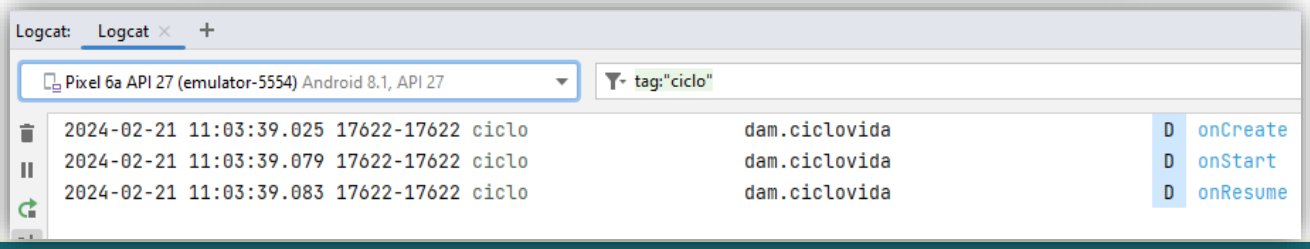
Más información:

https://developer.android.com/guide/components/activities/activity-lifecycle

Ejercicio 1 (1)

- Crear un proyecto nuevo, con las siguientes características:
  - Nombre: 3 Ciclo vida
  - Paquete: dam.ciclovida
  - Directorio: 3-CicloVida
- Sobrescribir todos los métodos que se invocan durante el ciclo de vida de la actividad para mostrar un mensaje con `Toast` en pantalla y otro con `Log.d` en la consola Logcat


```
@Override protected void onStart() {
    Toast.makeText( context: this,  text: "onStart", Toast.LENGTH_SHORT).show();
    Log.d ( tag: "ciclodevida", msg: "onStart");
    super.onStart();
}
```



Ejercicio 1 (2)

Haz estas pruebas para entender el ciclo de vida:

- Gira el terminal
- Sácala del foco abriendo la pantalla de aplicaciones recientes (botón cuadrado)
- Pasa a otra aplicación
- Muestra la pantalla de inicio y vuelve a la aplicación
- Pulsa el botón atrás y vuelve a la aplicación mediante la pantalla de aplicaciones recientes
- Pon la aplicación en modo multiventana y conmuta varias veces entre las aplicaciones

Guardar y recuperar el estado (I)

- Cuando el sistema destruye una actividad, guarda en un objeto `Bundle` el ***estado de instancia***: valores de algunas vistas que tienen identificador (atributo `id` con un valor), por ejemplo, el de los `EditText`, `CheckBox` `RadioButton`, `Spinner`, ... si el valor no se ha establecido por código
- Al ejecutar `onCreate()`, se recuperan los valores guardados en el `Bundle`
- Tras ejecutar `onPause()`, se ejecuta el método `onSaveInstanceState()`
- Tras ejecutar `onStart()`, si la actividad se reinició se ejecuta el método `onRestoreInstanceState()`
- Si queremos guardar otros valores (variables, atributos, etc.), hay que hacerlo sobrescribiendo estos métodos
- Una alternativa a sobrescribir `onRestoreInstanceState()` es recuperar los datos desde `onCreate()`, en caso de necesitarlos

https://developer.android.com/guide/components/activities/activity-lifecycle?#saras

Guardar y recuperar el estado (II)

- Para **guardar** el *estado de instancia*, sobrescribir `onSaveInstanceState(Bundle savedInstanceState)`:
  - Mediante métodos de la clase `Bundle`, guardar parejas clave/valor
  - Ejemplos:
    - `savedInstanceState.putString ("clave1", "valor");`
    - `savedInstanceState.putBoolean ("clave2", true);`
    - `savedInstanceState.putStringArrayList ("clave3", un_ArrayList_String);`
- Para **recuperar** el *estado de instancia*, sobrescribir `onRestoreInstanceState (Bundle savedInstanceState)`:
  - Mediante métodos de la clase `Bundle`, recuperar valores asociados a claves
  - Ejemplos:
    - `String string = savedInstanceState.getString ("clave1", "valor por defecto");`
    - `boolean un_boolean = savedInstanceState.getBoolean ("clave2", false);`
    - `ArrayList<String> un_ArrayList = savedInstanceState.getStringArrayList ("clave3");`
- Solo hay métodos de la clase `Bundle` para guardar datos primitivos u objetos simples. Para objeto complejos se usa `ViewModel` o persistencia

Guardar y recuperar el estado (III)

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
/* El siguiente método se ejecuta de forma automática cada vez que se destruye la actividad, por ejemplo
    al cambiar de orientación el terminal. Se ejecuta entre onPause() y onStop()    */
@Override
protected void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState); // Llamar a la superclase para que guarde el estado de otros objetos
    // Meter en el Bundle lo que queremos conservar
    savedInstanceState.putBoolean("k_boolean", true);
    savedInstanceState.putString("k_texto", "Un texto");
}
//  El siguiente método se invoca después del método onStart() si el Bundle no es nulo. Se ejecuta entre onStart() y onResume() *
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState){
    super.onRestoreInstanceState(savedInstanceState); // Llamar a la superclase para que recupere el estado de otros objetos
    un_boolean = savedInstanceState.getBoolean ("k_boolean", true);
    texto = savedInstanceState.getString("k_texto", "Un texto");
}
```

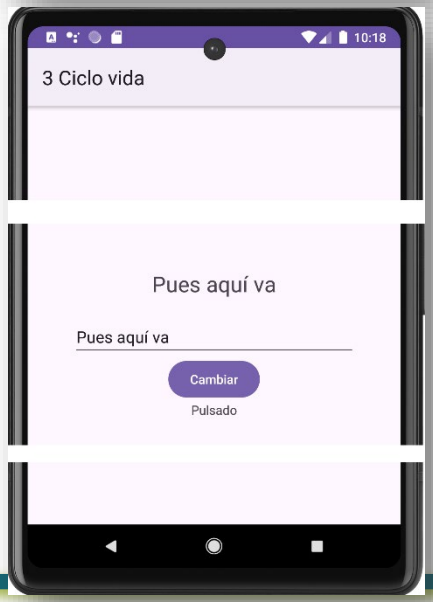
Guardar y recuperar el estado (IV)

Alternativa si se quiere usar valores del `Bundle` en `onCreate()`:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Recuperar los valores que se hayan guardado en onSavedInstanceState porque se necesitan ahora
    if (savedInstanceState != null) {   // Únicamente si la actividad ya existió y fue destruida
        un_boolean = savedInstanceState.getBoolean ("k_boolean", true);
        texto = savedInstanceState.getString("k_texto", "Un texto");

    } else {  // Es la primera vez que se ejecuta la actividad
        // Dar valores iniciales o por defecto a las variables/atributos que se necesite
        un_boolean = false;
        texto = "valor inicial";
    }
    setContentView(R.layout.activity_main);
}
@Override
protected void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    // Meter en el Bundle lo que queremos conservar
    savedInstanceState.putBoolean("k_boolean", true);
    savedInstanceState.putString("k_texto", "Un texto");
}
```

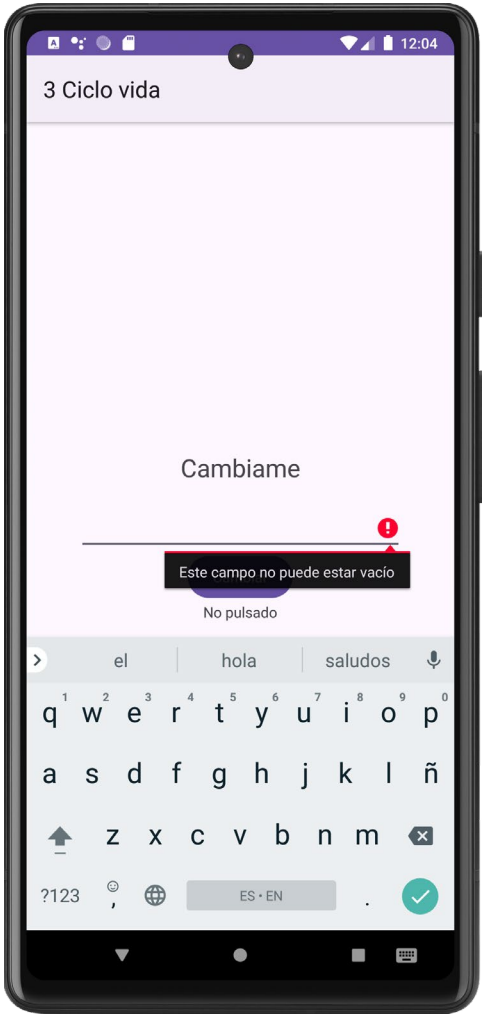
Ejercicio 2

- Modificar el proyecto anterior para añadir:
  - Un `EditText` donde poder escribir un texto
  - Un botón
  - Un `TextView` que indique si se ha pulsado el botón
- Al pulsar el botón, el texto escrito en el `EditText` se escribirá en el `TextView` superior. El `TextView` que indica si se ha pulsado el botón, mostrará que el botón ha sido pulsado
- Guardar y recuperar el estado de la actividad para que al girar el terminal se mantengan los valores de los dos `TextView`



Validación de EditText (1)

- Es común necesitar validar los campos de entrada de los formularios, como los `EditText`, por ejemplo, para asegurarse de que no estén vacíos antes de procesarlos.
- Para indicar al usuario los campos vacíos o con valores no válidos se usan los métodos `setError()` y `requestFocus()` de la clase `EditText`
- El método `showSoftInput()`, de la clase `InputMethodManager`, muestra el teclado

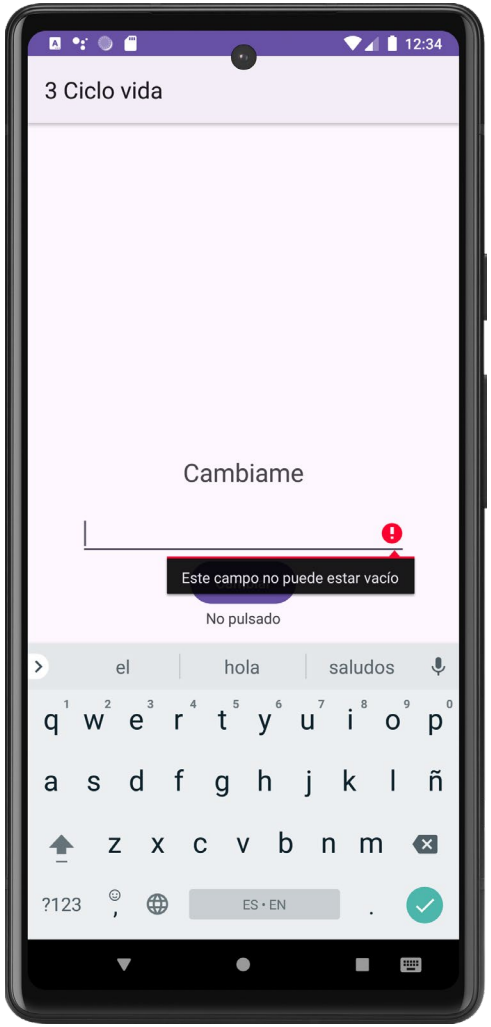


Validación de EditText (2)

```
EditText editText = findViewById(R.id.et_texto);
Button boton = findViewById(R.id.btn_boton);
InputMethodManager imm = (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);

boton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      String input = editText.getText().toString().trim();

      if(input.isEmpty()){
         editText.setError("Este campo no puede estar vacío");
         editText.requestFocus();


         // Abrir el teclado sobre el EditText que ya tiene el foco
         if (imm != null)
            imm.showSoftInput(editText, InputMethodManager.SHOW_IMPLICIT);


      } else {
         // Continuar con la lógica de la aplicación...
      }
   }
});
```

Ejercicio 3

- Modificar el proyecto anterior validar el `EditText` obligando a que no esté vacío.
- Si esto sucede debe mostrar un texto de error y el teclado, poniendo el foco en el `EditText`



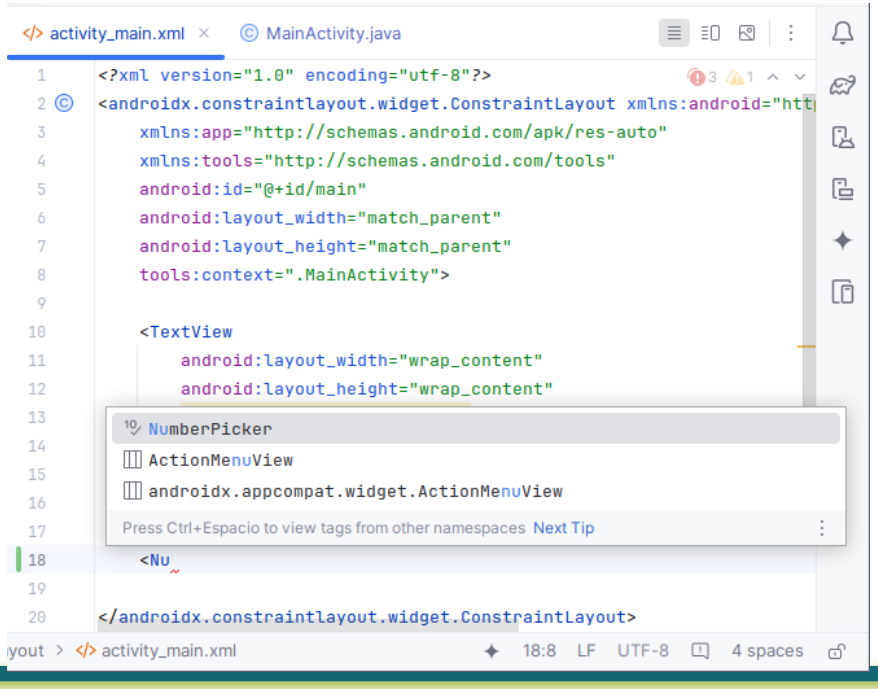
Widgets (I)

Además de los componentes gráficos básicos (**`TextView`**, **`EditText`** o **`Button`**), Android proporciona otros componentes más elaborados que pueden ser usados en las interfaces de usuario:

- **`TextClock`**: mostrar un reloj
- **`RatingBar`**: barra de valoración
- **`Switch`**: interruptor
- **`CheckBox`**: casillas de verificación
- **`RadioButton`**: botones de opción
- **`ImageView`**: mostrar imagen
- **`Spinner`**: lista desplegable
- …

Widgets (II)

- Algunos *Widges* no aparecen en la ventana *Pallete* del *Layout Editor*, en este caso hay que buscarlos
- Se puede usar la vista *Code* y escribir su nombre para usar la función de autocompletado:
- Ej con `NumberPicker`:



TextClock

- Declaración XML:


```
<TextClock
    android:id="@+id/hora"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:textColor="#F44336" />
```

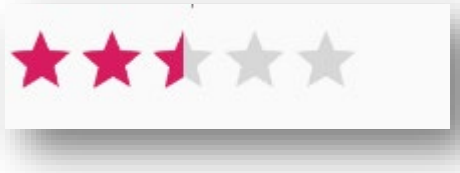
- Obtener la hora:

```
TextClock horaVw = findViewById(R.id.hora);
String hora = horaVw.getText()
```

https://developer.android.com/reference/android/widget/TextClock

RatingBar

- Declaración XML:



```
<RatingBar
    android:id="@+id/valoracionRatingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:scaleX="0.5"  <-- Opcional -->
    android:scaleY="0.5"  <-- Opcional -->
/>
```

- Obtener la valoración:

```
RatingBar valoracionVw = findViewById(R.id.valoracionRaingBar);
float valoracion = valoracionVw.getRating();
```

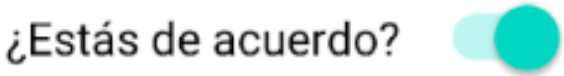
- Dar valor inicial:

```
valoracionVw.setRating(valorFloat);
```

https://developer.android.com/reference/android/widget/RatingBar

Switch

- Declaración XML:




```
<Switch
    android:id="@+id/acuerdo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="¿Estás de acuerdo?"/>
```

- Obtener el estado del `Switch`:

```
Switch acuerdoVw = findViewById(R.id.acuerdo);
boolean acuerdo = acuerdoVw.isChecked();
```

- Dar valor inicial:

```
acuerdoVw.setChecked(true/false);
```

https://developer.android.com/reference/android/widget/Switch

CheckBox

- Declaración XML:


```
<CheckBox
    android:id="@+id/cbProfesional"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Profesional"/>
```

- Obtener el valor de un `CheckBox`:

```
CheckBox checkProfesionalVw = findViewById(R.id.cbProfesional);
boolean marcado = acuerdoVw.isChecked();
String valor= checkProfesionalVw.getText().toString();
```

- Dar valor inicial:

```
checkProfesionalVw.setChecked(true/false);
```

https://developer.android.com/reference/android/widget/CheckBox

RadioGroup y RadioButton

Como los `RadioButton` son mutuamente excluyentes, deben agruparse en un `RadioGroup`

Declaración XML

```
<RadioGroup
 android:id="@+id/RG"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center"
  android:orientation="horizontal">
  <RadioButton
    android:id="@+id/rBtnSi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Si" />
  <RadioButton
    android:id="@+id/rBtnNo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="No" />
</RadioGroup>
```

Obtener el estado del `RadioButton` *rBtnSi*:

```
RadioButton botonSiVw = findViewById(R.id.rBtnSi);
boolean botonSi = botonSiVw.isChecked();
```

Dar valor inicial al `RadioButton` *rBtnSi*:

```
botonSiVw.setChecked(true/false);
```

Obtener el texto del `RadioButton` seleccionado:

```
RadioGroup rgVw = findViewById(R.id.RG);
int seleccionadoId = rgVw.getCheckedRadioButtonId();
RadioButton rbVw = findViewById(selectedId);
String texto = rbVw.getText().toString();
```

https://developer.android.com/reference/android/widget/RadioButton

ImageView

- Permite mostrar imágenes
- Sintaxis básica:

```
<ImageView
   android:id="@+id/miImagen"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   app:srcCompat="@drawable/mi_imagen" />
```

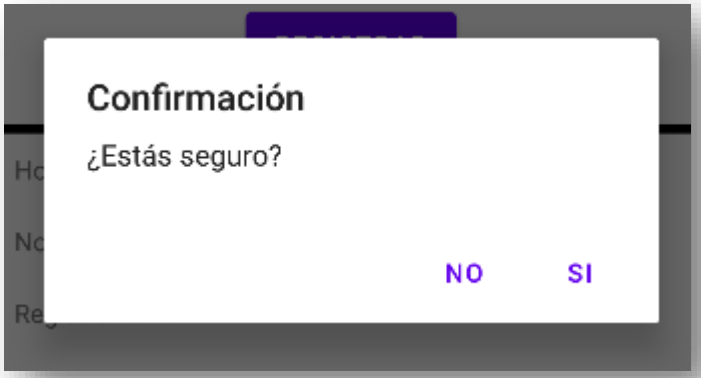
- Cambiar la imagen actual por otra:

```
ImageView imagen = findViewById(R.id.miImageView);
imagen.setImageResource(R.drawable.nueva_imagen);
```

- En estos ejemplos la imagen está en la carpeta `res/drawable/` dentro del proyecto

Cuadros de diálogo (1)

- Un diálogo es una ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional.

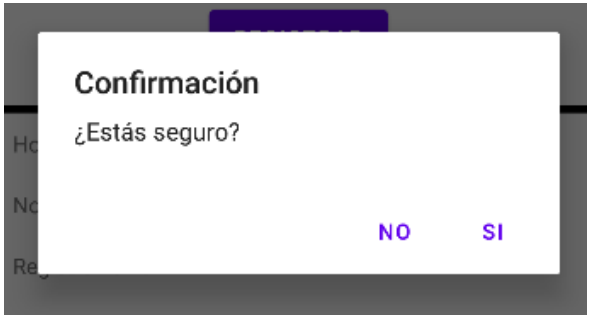


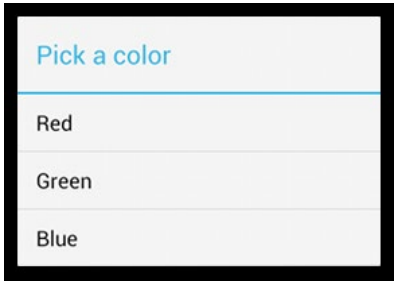
- Un diálogo no ocupa toda la pantalla y, generalmente, se usa para eventos modales que requieren que los usuarios realicen alguna acción para poder continuar.
- Las clases usadas para crear cuadros de diálogo heredan de la clase `Dialog`

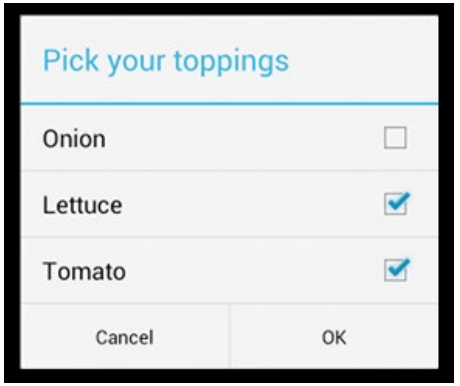
https://developer.android.com/guide/topics/ui/dialogs

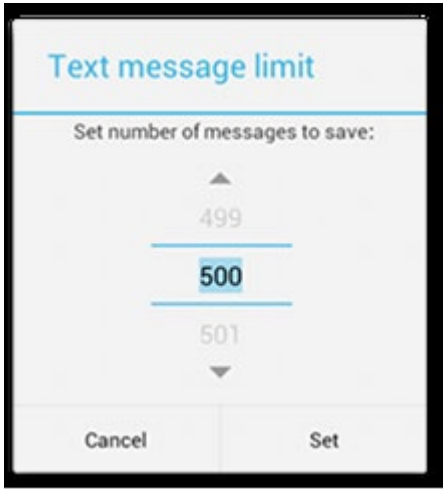
Cuadros de diálogo (2)

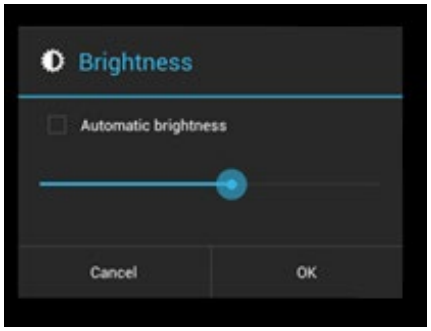
AlertDialog

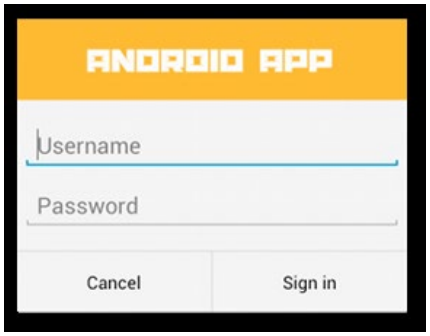






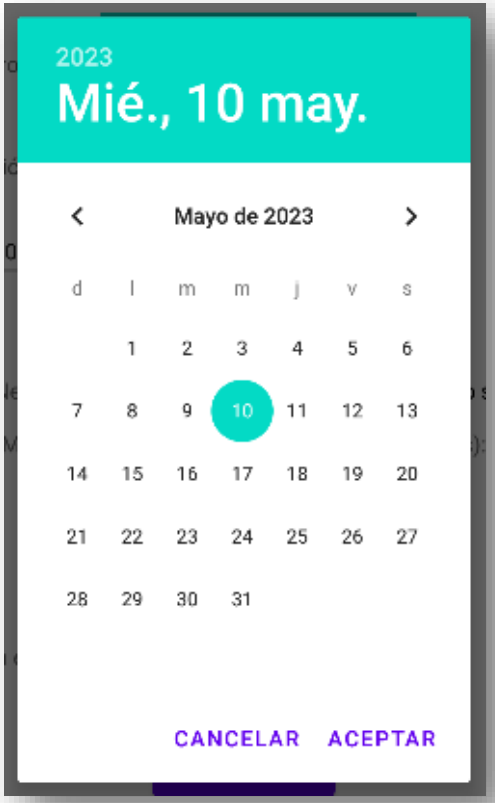




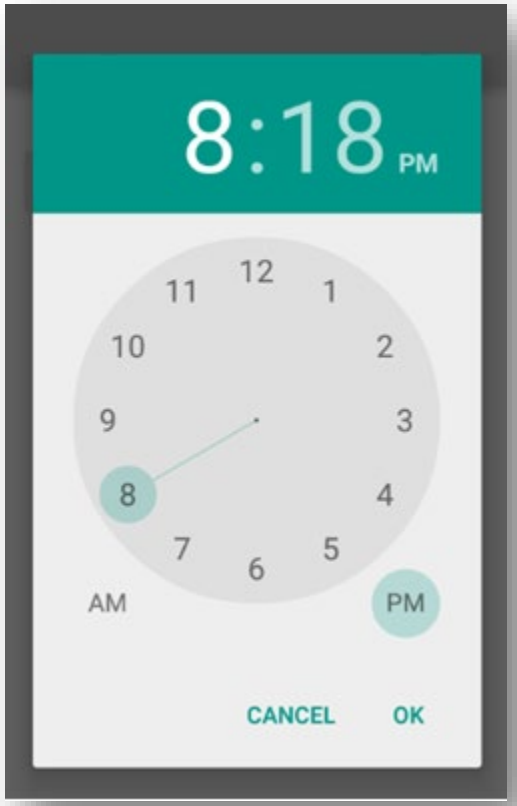


Cuadros de diálogo (3)

DatePickerDialog

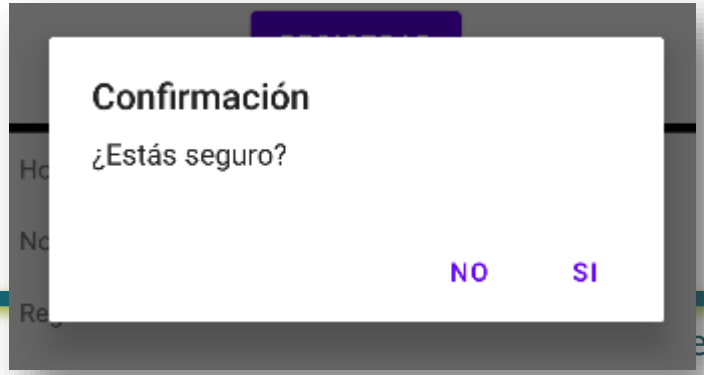


TimePickerDialog



AlertDialog básico (4)

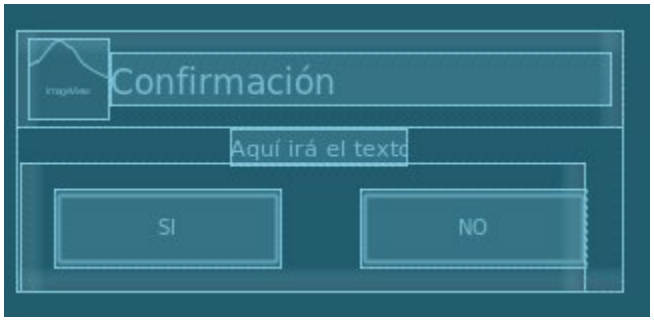
```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setCancelable(false);   // Evita que al pulsar fuera del cuadro de diálogo
                                // este desaparezca
builder.setTitle("Confirmación")
builder.setMessage("¿Estás seguro?");
builder.setPositiveButton("Si", new DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int whichButton) {
      // Código de Si

   }
});
builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int whichButton) {
      // Código de No

   }
});

builder.show();  // Mostrar el cuadro de diálogo
```

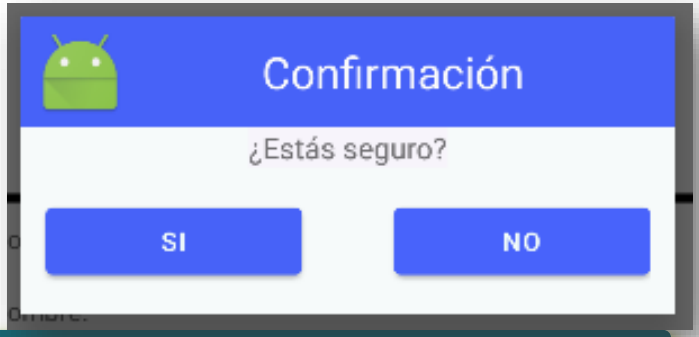


AlertDialog personalizado (5)

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
View vistaCuadroDialogo = View.inflate(this, R.layout.cuadro_dialogo_view, null);
builder.setView(vistaCuadroDialogo);
AlertDialog dialog = builder.create();
dialog.setCancelable(false);

TextView contenido = vistaCuadroDialogo.findViewById(R.id.tv_mensaje);
contenido.setText("¿Estás seguro?");

Button botonSi = vistaCuadroDialogo.findViewById(R.id.Si);
botonSi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Código del SI

        dialog.dismiss();
    }
});
// … idem para el botón del NO ….
dialog.show();  // Mostrar el cuadro de diálogo
```

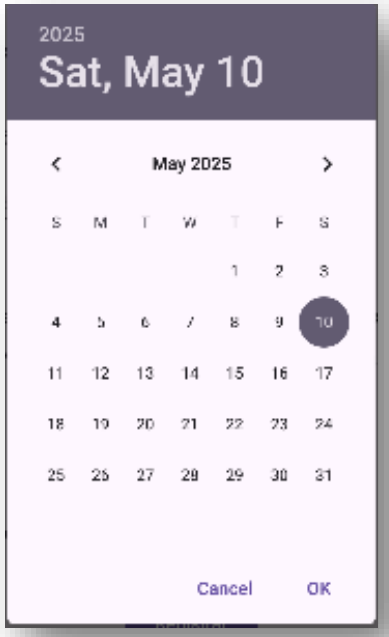
cuadro_dialogo_view.xml





DatePickerDialog

```
// Variables locales donde se volcará la fecha al retornar del cuadro de diálogo
// También sirven para dar el valor inicial
final int añoSeleccionado = 2025;
final int mesSeleccionado = 4;    // Los meses empiezan a numerarse desde el 0, así que el 4 es el mes de mayo
final int diaSeleccionado = 10;

// Crear el manejador del DatePickerDialog
DatePickerDialog.OnDateSetListener dateSetListener = new DatePickerDialog.OnDateSetListener() {
    @Override
    public void onDateSet(DatePicker view, int year,
                          int monthOfYear, int dayOfMonth) {
        // Código a ejecutar cuando se selecciona una fecha
        // En este ejemplo, escribirla en un EditText del layout de la actividad
        fechaSeleccionada.setText(dayOfMonth + "-" + monthOfYear  + "-" + year);
    }
};


// Crear el DatePickerDialog (tema por defecto):
DatePickerDialog datePickerDialog = new DatePickerDialog(this,
    dateSetListener, añoSeleccionado, mesSeleccionado, diaSeleccionado);


datePickerDialog.show();    // Y mostrarlo
```



Diseño con Scroll (1)

- Es posible que todos los elementos de una actividad no se puedan visualizar debido al tamaño de la pantalla.
- Para permitir el *scroll* de una GUI se necesita envolver el contenedor raíz dentro de un elemento `ScrollView`
- Un elemento `ScrollView` únicamente puede tener un elemento hijo, que debería ser un contendedor
- Funciona con las dos orientaciones: horizontal y vertical

https://developer.android.com/reference/android/widget/ScrollView

Diseño con Scroll (2)

Diseño sin scroll:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    ::::::::::::

</LinearLayout>
```

Diseño con scroll:

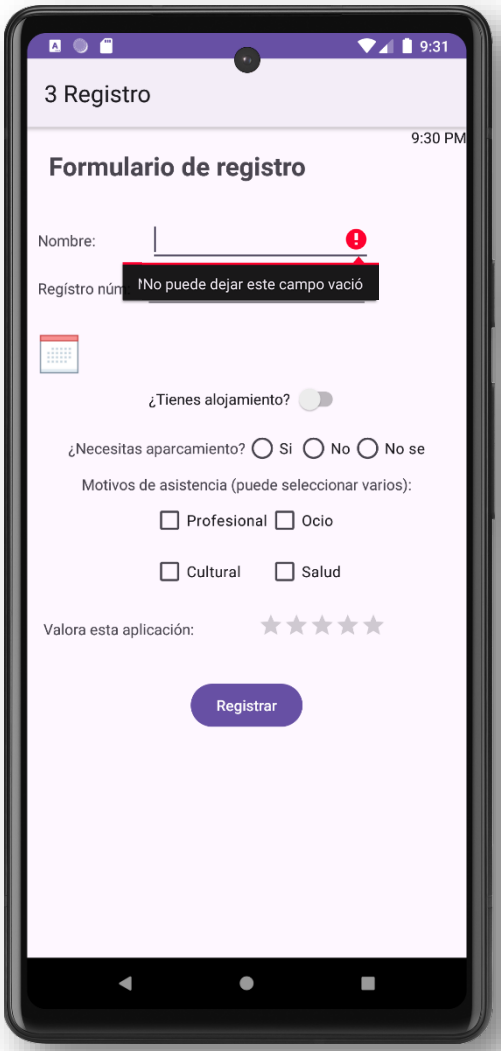
```
?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"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

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    ::::::::::::
    </LinearLayout>

</ScrollView>
```

Visibilidad de las vistas

- En XML, la propiedad `android:visibility` indica:
  - `VISIBLE`: es visible (por defecto)
  - `INVISIBLE`: es invisible, pero ocupa espacio
  - `GONE`: es invisible y no ocupa espacio
- Con el método `setVisibility(View.`*`visibildiad`*`)` de una vista se puede establecer la visibilidad de la vista.

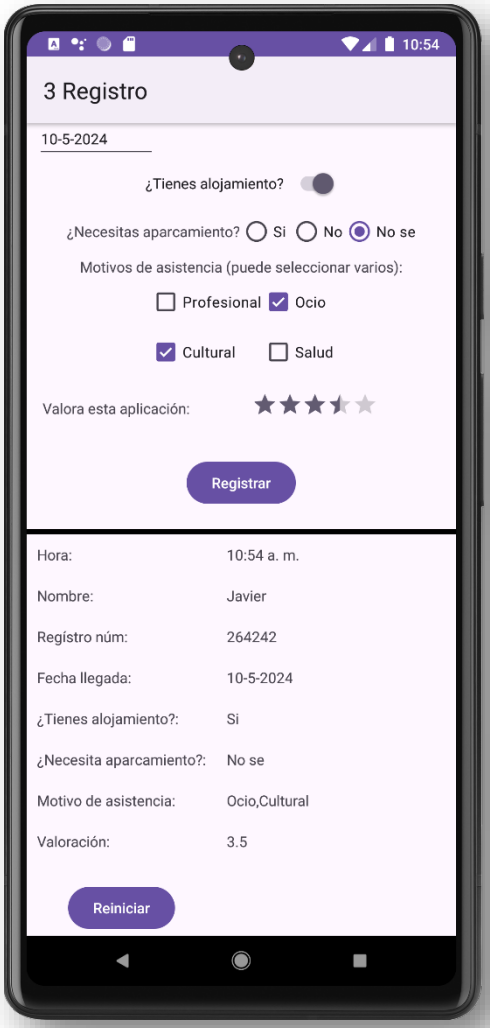
Ejercicio 4 (1)

- Crea un proyecto nuevo, con las siguientes características:
  - Nombre: 3 Registro
  - Paquete: dam.registroV1
  - Directorio: 3-Registro
- Pon un icono a la aplicación
- Construye un formulario de registro con los siguientes componentes:
  - Un reloj (`TextClock`) en la parte superior derecha
  - Dos entradas (`EditText`) una de texto y la otra numérica, ambas entradas obligatorias
  - Un icono con un calendario que, al hacer clic, abra un cuadro de diálogo para poner una fecha. Una vez establecida la fecha se mostrará en un `TextView` que, al hacer clic, abrirá el cuadro de diálogo para poner la fecha recuperando el valor establecido
  - Un interruptor (`Switch`)
  - Tres botones de opción (`RadioButton`)
  - Cuatro casillas de verificación (`CheckBox`)
  - Uns barra de valoración (`Ratingbar`)



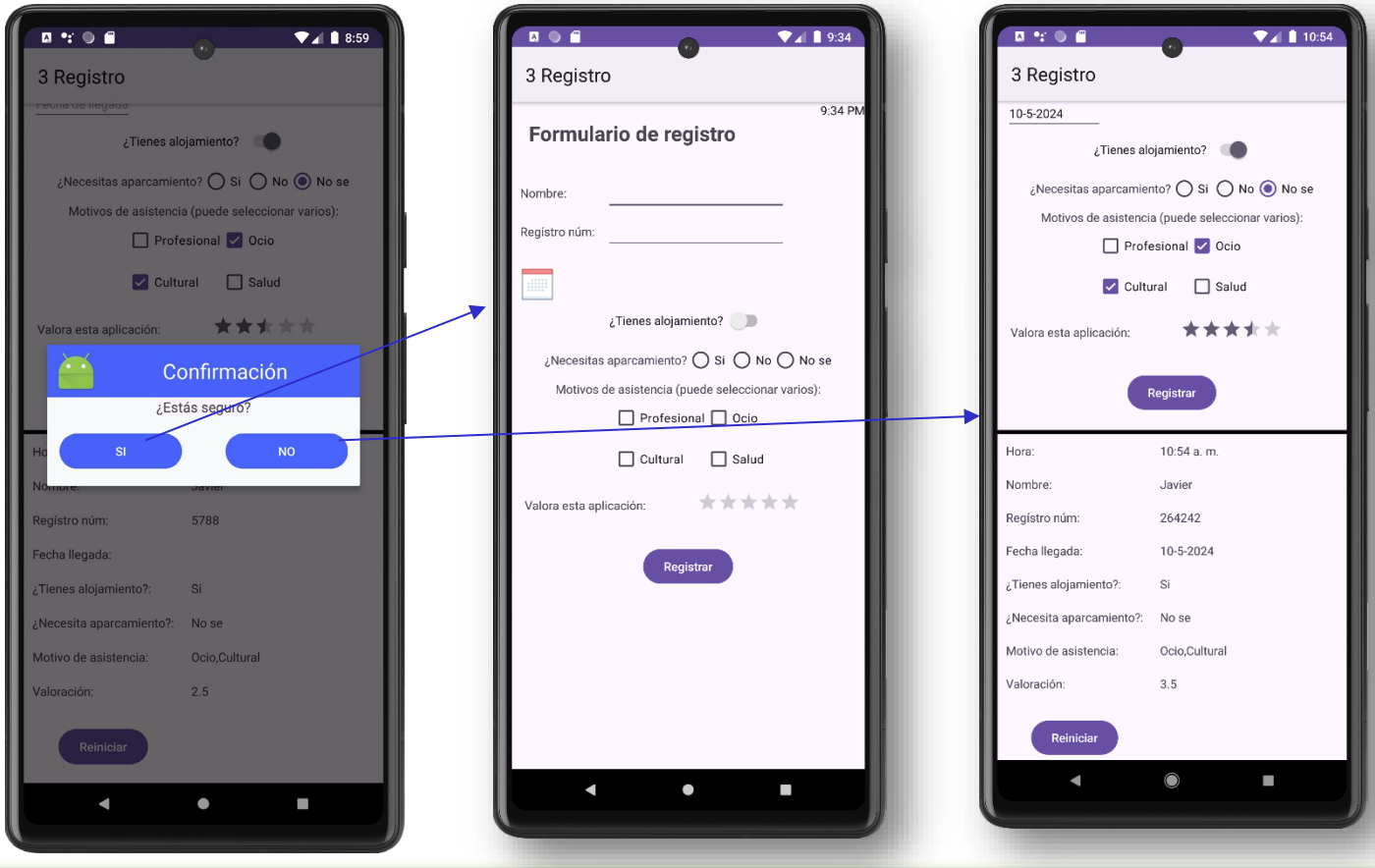
Ejercicio 4 (2)

- Al pulsar el botón de registrar, se mostrará en la parte inferior los valores presentes en los elementos del formulario. Antes de pulsar el botón esa parte estará oculta
- Se debe poder usar *scroll* para poder ver todos los datos
- El *layout* GridLayout puede ser útil para contener las etiquetas y elementos de entrada de datos y resultado, poniendo el atributo `columnCount` a 2.



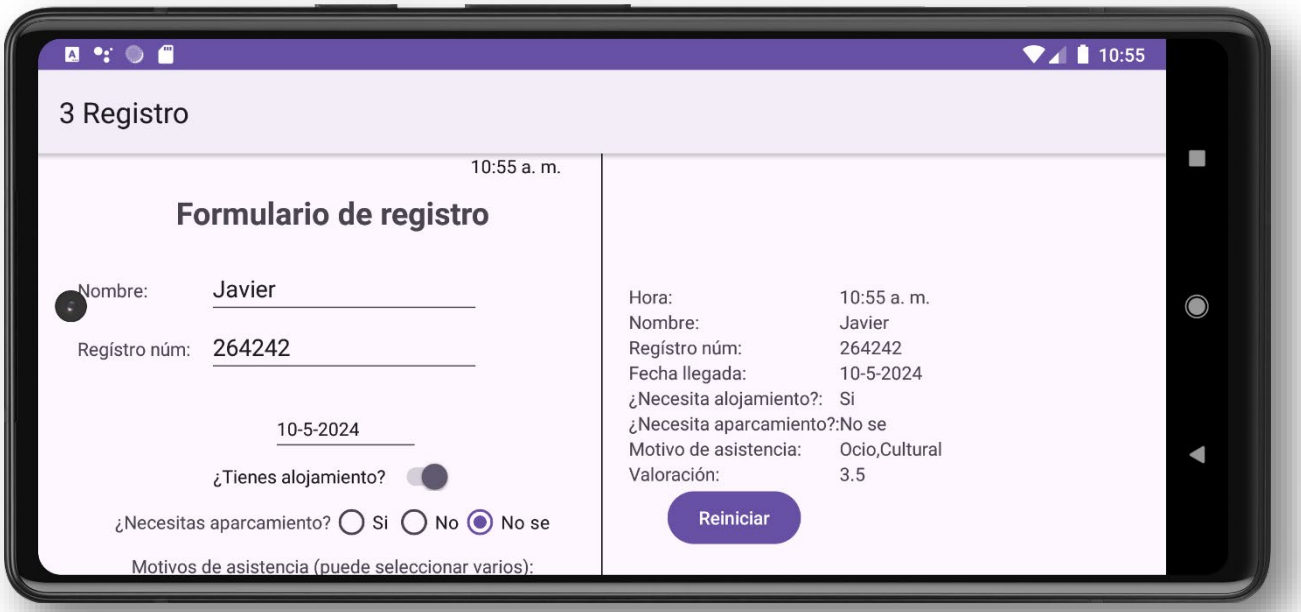
Ejercicio 4 (3)

- Al pulsar el botón de reiniciar, se mostrará un cuadro de diálogo de confirmación, se borrarán todos los valores del formulario y se ocultará la parte inferior

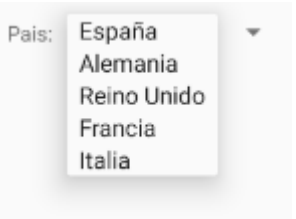


Ejercicio 4 (4)

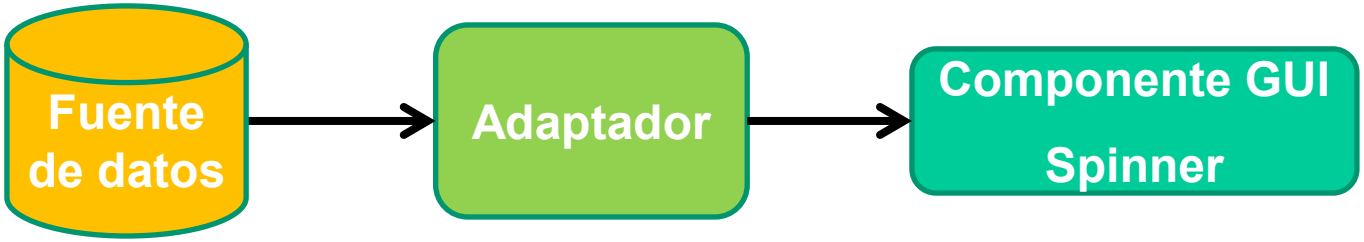
- Definir un *layout* para la orientación horizontal, donde se distribuirá la información en dos columnas, a la izquierda los elementos de entrada y a la derecha sus valores al pulsar actualizar
- Al cambiar de orientación, se debe guardar y recuperar el estado, de forma que, si ya se había pulsado el botón de registrar, se muestren los datos recogidos



Spinner (1)



- Un `Spinner` es un componente que permite escoger un elemento de una lista
- Los elementos de la lista se insertan en el `Spinner` mediante un adaptador (`Adapter`) que:
  - Obtiene los datos de una variable de Java, un XML o una base de datos
  - Convierte cada valor en una vista `TextView`



- La clase `ArrayAdapter<T>`:
  - Se usa cuando los datos están en un array de Java o en un elemento `string-array` en un fichero XML
  - Por defecto, `ArrayAdapter` usa `toString()` sobre cada elemento y sitúa cada uno en un `TextView` para formar la lista desplegable

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

Spinner (2)

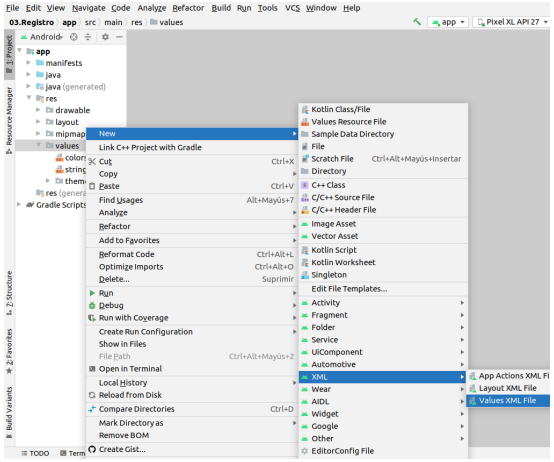
- Declaración del **`Spinner`** en el XML de diseño de la actividad:

```
<Spinner
    android:id="@+id/sp_paises"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:spinnerMode="dropdown" />
```

- Los valores de la lista desplegable se pueden obtener de:
  - Un array de `String` en Java

```
String [] paises_array={"España", "Alemania", "Reino Unido", "Francia", "Italia"};
```

  - Un fichero XML situado en `/res/values`



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="paises_array">
        <item>España</item>
        <item>Alemania</item>
        <item>Reino Unido</item>
        <item>Francia</item>
        <item>Italia</item>
    </string-array>
</resources>
```

Spinner (3)

- Obtener la referencia al componente definido en el *layout* XML

```
Spinner paises_Vw = findViewById(R.id.sp_paises);
```

- Instanciar un adaptador con datos. Dependiendo del origen de los datos:
  - A partir del array de String `paises_array` definido en Java:

```
ArrayAdapter adaptador= new ArrayAdapter( this,
                                          android.R.layout.simple_spinner_item,
                                          paises_array);
```

  - A partir del elemento **`string-array`** definido en `/res/values/paises.xml`

```
ArrayAdapter adaptador = ArrayAdapter.createFromResource( this,
                                                          R.array.paises_array,
                                                          android.R.layout.simple_spinner_item);
```

- Asociar el adaptador al `Spinner`:

```
paises_Vw.setAdapter(adaptador);
```

- Recuperar el valor seleccionado:

```
String seleccion = paises_Vw.getSelectedItem().toString()
```

- Dar valor inicial:

```
paises_Vw.setSelection(adaptador.getPosition("valor"))
```

Spinner (4)

Parámetros del constructor de `ArrayAdapter`:

- **Contexto de aplicación**. Típicamente es "this", la actividad actual.
- **La lista de datos a mostrar**. Puede ser un array de `String` definido en java o el identificador del recurso descrito en un fichero XML
- **El *layout* con el que se mostrará cada ítem de la lista**. Se puede usar un *layout* predefinido en Android, o bien diseñar un *layout* propio con una presentación diferente que debe contener un `TextView` con un identificador **"@android:id/text1"**

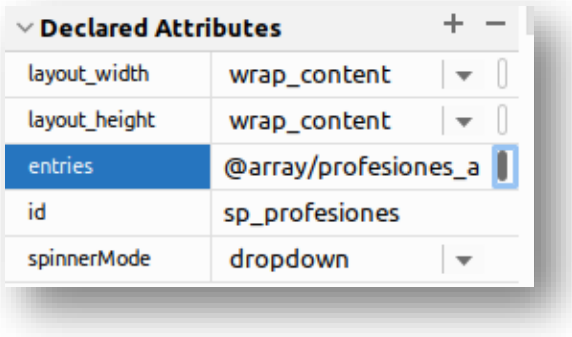
```
ArrayAdapter adaptador= new ArrayAdapter( this,
                                          android.R.layout.simple_spinner_item,
                                          paises_array);
```

```
ArrayAdapter adaptador = ArrayAdapter.createFromResource( this,
                                                          R.array.paises_array,
                                                          android.R.layout.simple_spinner_item);
```

Spinner (5)

- Una alternativa a usar un adaptador, cuando los datos están en un recurso XML, es usar, en la declaración del elemento Spinner del XML de diseño, el atributo **android:entries** e indicar el recurso XML con los valores
- En este caso, no hace falta crear un adaptador por código
- El *layout* por defecto que usará al desplegar la lista es `simple_spinner_dropdown_item`

```
<Spinner
   android:id="@+id/Profesiones"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:entries="@array/profesiones_array"
   android:spinnerMode="dropdown" />
```



Spinner (6)

Diseños predefinidos:

- https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/layout/
- `simple_spinner_item.xml`:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    style="?android:attr/spinnerItemStyle"
    android:singleLine="true"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ellipsize="marquee"
    android:textAlignment="inherit"/>
```

Spinner (7)

Personalizar el diseño:

- Crear en `res/layout` un fichero XML basado en `simple_spinner_item.xml`
- Ejemplo cambiando el color del texto:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    style="?android:attr/spinnerItemStyle"
    android:singleLine="true"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ellipsize="marquee"
    android:textAlignment="inherit"
    android:textColor="@android:color/holo_red_light"/>
```

- Usarlo al crear el adaptador:

```
ArrayAdapter adaptador = ArrayAdapter.createFromResource( this,
                                R.array.paises_array,
                                R.layout.personalizado_spinner_item);
```

Spinner (8)

Es posible querer manejar el evento de seleccionar un elemento:

```
public class Main extends Activity{
    ...
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      ...
      Spinner paises_Vw = findViewById(R.id.sp_paises);
      ... Instanciar un adaptador con datos y asociarlo al spinner
      //Asociar un manejador al spinner
      paises_Vw.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
      {

         @Override
         public void onItemSelected(AdapterView<?> parent, View view,
                                                    int position, long id) {
             String selection = parent.getItemAtPosition(position).toString();
             Toast.makeText(parent.getContext(),"Selección actual: "+
                   selection,Toast.LENGTH_SHORT).show();
         }

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

         }
   });
}
```

Ejercicio 4

- Modificar la aplicación del ejercicio 3 para añadir:
  - Un `Spinner` con una lista de países declarando los valores en un array
  - Un `Spinner` con una lista de profesiones declarando los valores en un fichero XML y con *layout* personalizado
- Establecer un manejador en cada `Spinner` que muestre un mensaje *toast* con cada cambio de selección