PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

TEMA 6:

Controles básicos.

UNION EUROPEA
Fondo Social Europeo
"Una manera de hacer Europa"

"Una manera de hacer Europa". Cofinanciación a cargo del Programa Operativo del FSE 2014-2020 para Extremadura gastos de Ciclos Formativos de Grados Medio y Superior.

ÍNDICE

PMDM

Controles básicos.

1. Tratamiento general de los controles

UNION EUROPEA
Fondo Social Europeo
"Una manera de hacer Europa"
"Una manera de hacer Europa". Cofinanciación a cargo del Programa Operativo del FSE 2014-2020 para Extremadura gastos de Ciclos Formativos de Grados Medio y Superior.

1.- Tratamiento general de los controles

- Los controles tienen un ID de número entero asociado.
- Estos identificadores se asignan en los archivos XML de diseño.
- Nos sirven para asociar en los ficheros KOTLIN los distintos elementos del XML.
- Por tanto, una vez que añadimos un elemento (textView, Button, …) a nuestro XML, si queremos trabajar con él en el fichero KOTLIN, debemos "asociarlo" (capturarlo).

1.- Tratamiento general de los controles

- Para obtener una referencia a los elementos de una Activity debemos capturarlos de la siguiente manera:

val nombreElemento: tipoElemento = findViewById<tipoElemento>(R.id.identificadorElemento)

siendo *tipoElemento* un TextView, Button, ..., etc.

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val btnEjemplo: Button = findViewById<Button>(R.id.btnEjemplo)
    }
}
```

1.- Tratamiento general de los controles

- Para capturar los elementos del Fragment necesitaremos hacerlo a través de la vista.
- Por eso, en el método *onCreateView* debemos obtener la vista y, a través de ella, capturar los componentes mediante su Id (*findViewById*).

1.- Tratamiento general de los controles

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    val view = inflater.inflate(R.layout.fragment_blank, container, attachToRoot: false)

    val btnEjemplo: Button = view.findViewById<Button>(R.id.btnEjemploFragment)
    btnEjemplo.setOnClickListener() { it: View!
        Toast.makeText(context, text: "Botón pulsado", Toast.LENGTH_LONG).show()
    }

    return view
}
```

PMDM

Controles básicos.

2. La clase TextView

2.- La clase TextView

- El TextView o etiqueta (label) se utiliza para escribir un texto en nuestra activity.
- Tiene varias propiedades para formatear el texto que contiene:
  - **android:typeface** Permite cambiar el tipo de fuente.
  - **android:textStyle** Permite cambiar el estilo de la letra (normal, cursiva, negrita).
  - **android:textColor** Permite seleccionar el color de la etiqueta en formato RGB hexadecimal.
  - **android:textSize** Permite especificar el tamaño de la fuente, por ejemplo 20sp (Scale-independent Pixels: ajusta tanto para la densidad de pantalla como a las preferencia del usuario).

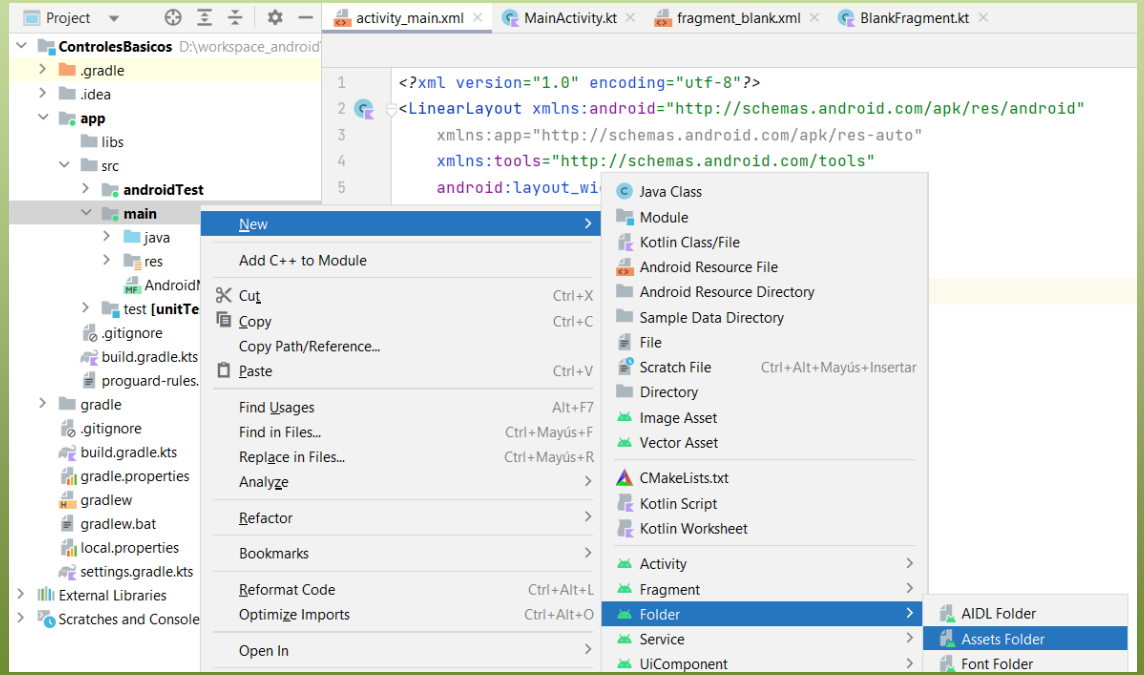
2.- La clase TextView

- Debemos tener en cuenta que, por defecto, los tipos de fuente incluidos en Android Studio son *sans*, *monospace* y *serif*.
- Por tanto, cualquier otra opción debe ser cargada.
- Por ejemplo, para poder usar una fuente Arial, debemos cargar el fichero *arial.ttf* en el directorio "assets" (herramientas) y cargarla desde código también:

UNION EUROPEA
Fondo Social Europeo
"Una manera de hacer Europa"
"Una manera de hacer Europa". Cofinanciación a cargo del Programa Operativo del FSE 2014-2020 para Extremadura gastos de Ciclos Formativos de Grados Medio y Superior.

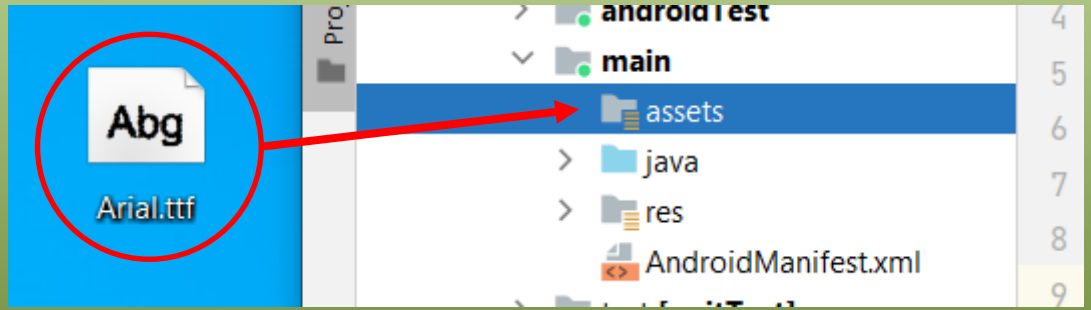
2.- La clase TextView

- Creamos el directorio *assets* dentro del directorio *main:*



2.- La clase TextView

- Arrastramos dentro del directorio *assets* la fuente o fuentes que deseemos utilizar en nuestra aplicación:



2.- La clase TextView

- Posteriormente, modificamos el archivo .kt correspondiente al xml donde queremos cambiar la fuente.
- En este ejemplo, modificamos el MainActivity.kt, ya que queremos que la letra sea distinta en el activity_main.xml:

2.- La clase TextView

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val textViewEjemplo: TextView = findViewById<TextView>(R.id.textViewEjemplo)
        val typeface: Typeface = Typeface.createFromAsset(assets,  path: "Arial.ttf")
        textViewEjemplo.setTypeface(typeface)
    }
}
```

EJERCICIOS

- **Ejercicio 01:** Crea un proyecto en Android Studio con una sola actividad. Mediante Constraint Layout, sitúa un textView en el centro de la activity.
- Carga una nueva fuente en Android Studio y utiliza éste nuevo tipo de letra en tu textView.

2.- La clase TextView

- Otra forma de mostrar información al usuario es a través de los **Toast**.
- Un Toast es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento.

2.- La clase TextView

- Su utilización se basa en la clase Toast.
- Esta clase dispone de un método estático *makeText()*, que tiene los siguientes parámetros:
  - **Contexto de la actividad.** Que obtenemos mediante *applicationContext.*
  - **Texto a mostrar.** No olvidar añadir el recurso en el fichero *strings.xml*.
  - **Duración del mensaje.** Con los valores *LENGTH_LONG* o *LENGTH_SHORT*.
- Sólo quedaría mostrarlo por pantalla a través del método *show()*.

Ejemplo:

Toast.makeText(applicationContext,"Error", Toast.LENGTH_SHORT).show()

PMDM

Controles básicos.

3. La clase Button

3.- La clase Button

- Button es una subclase de la clase View, por lo que las propiedades que vimos en el punto anterior también son válidas para la clase Button.
- **Tratamiento de un botón**. Para que un botón responda a un evento de tipo *Click*, debemos realizar lo siguiente:

3.- La clase Button

- Tratamiento de un botón:
  - **Capturamos el botón**
    - *val btnEjemplo: Button = findViewById<Button>(R.id.btnEjemplo);.*
  - **Implementamos setOnClickListener**

```
btnEjemplo.setOnClickListener() {
        //Código
}
```

Importante: Para poder realizar estos pasos en el fichero .kt, previamente debemos añadir un botón en el .xml.

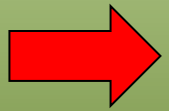
3.- La clase Button

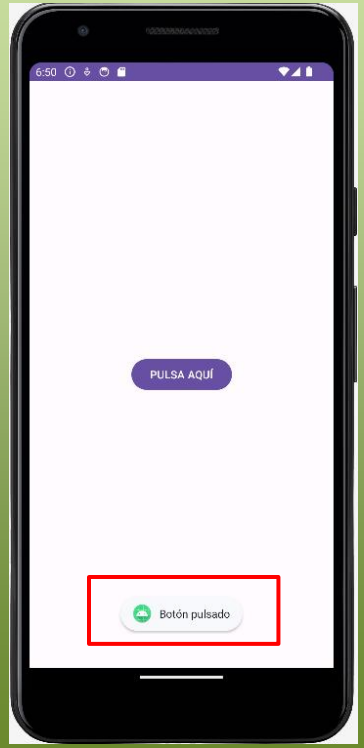
```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val btnEjemplo: Button = findViewById<Button>(R.id.btnEjemplo)

        btnEjemplo.setOnClickListener() { it: View!
            Toast.makeText(applicationContext, text: "Botón pulsado", Toast.LENGTH_SHORT).show()
        }
    }
}
```





3.- La clase Button

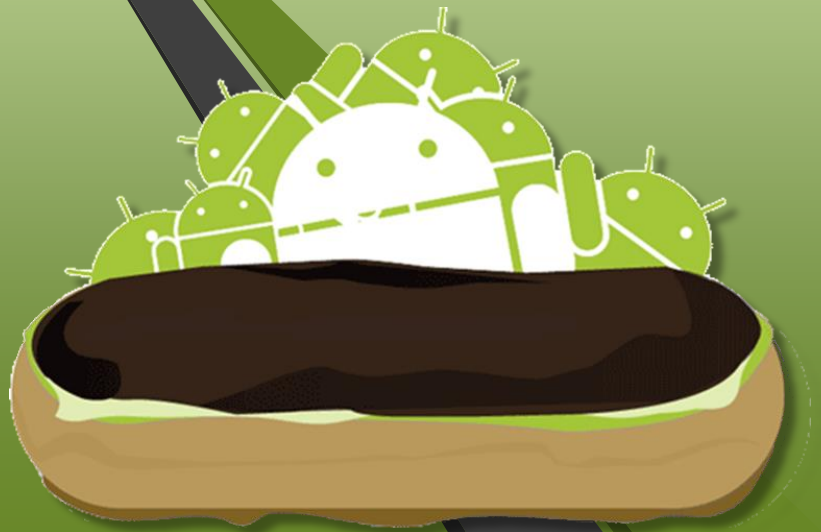
- Android Studio nos permite diseñar nuestros propios botones para darles la apariencia que más nos guste.
- De hecho, hay páginas web que te ayudan a diseñar botones de forma sencilla.
- Estas páginas nos permiten realizar el diseño de los botones y generan automáticamente el código fuente para poder copiarlo en nuestro proyecto.

EJERCICIOS

- **Ejercicio 02:** Crea un proyecto en Android Studio con una sola actividad. Mediante Constraint Layout, sitúa un botón en el centro de la activity.
- Captura el evento click del botón de manera que al pulsarlo muestre un Toast con el mensaje "Botón pulsado".
- Personaliza el diseño de tu botón utilizando alguna web que encuentres en internet.

PMDM

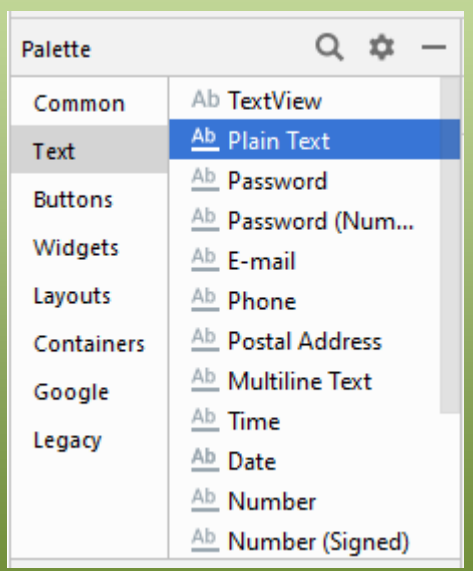
Controles básicos.



4. La clase EditText

4.- La clase EditText

- Es una subclase de EditText y es la manera más popular de que el usuario interaccione e introduzca datos en nuestras aplicaciones.
- Android Studio incorpora en la paleta de widgets un montón de campos de texto distintos para no tener que cambiar sus propiedades en el XML mediante código:



4.- La clase EditText

- Estas son algunas de las propiedades más interesantes que podemos agregar en el XML:
  - Primera letra mayúscula → android:inputType = "textCapSentences"
  - Mayúsculas todas las palabras → android:inputType = "textCapWords"
  - Todo en mayúsculas → android:inputType = "textCapCharacters"
  - Primera letra mayúscula y autocorreción de errores ortográficos → android:inputType="textCapSentences|textAutoCorrect"

PMDM

Controles básicos.

5. Las clases RadioGroup y RadioButton

5.- Las clases RadioGroup y RadioButton

- Permiten al usuario la selección de un conjunto de opciones excluyentes entre sí, es decir, que sólo puede seleccionar una de ellas a la vez.
- Para crearlo, en la pantalla de diseño, insertaremos un RadioGroup y luego arrastraremos dentro los RadioButton que queramos.
- El Click en un RadioGroup y en un RadioButton se pueden capturar de forma idéntica a un botón.

5.- Las clases RadioGroup y RadioButton

- Podemos saber qué RadioButton está pulsado en un RadioGroup a través del método *checkedRadioButtonId* perteneciente al RadioGroup.
- Para poder utilizar este método debemos capturar previamente el RadioGroup:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val radioGroup: RadioGroup = findViewById<RadioGroup>(R.id.radioGroup)
        val selectedRadioButtonId = radioGroup.checkedRadioButtonId

        if(selectedRadioButtonId == -1) {
            //No se ha seleccionado ningún radioButton dentro del radioGroup
        } else if(selectedRadioButtonId == R.id.radioOpc1) {
            //Se ha seleccionado el radioButton con id "radioOpc1"
        } else if(selectedRadioButtonId == R.id.radioOpc2) {
            //Se ha seleccionado el radioButton con id "radioOpc2"
        }
    }
}
```

EJERCICIOS

- **Ejercicio 03:** Crea un proyecto en Android Studio con un botón (en su interior tendrá el texto "¿Qué tengo seleccionado?" y dos radioButton (masculino y femenino).
- Cuando el usuario pulse el botón, mostraremos un Toast con el mensaje "Ha seleccionado masculino" o "Ha seleccionado femenino", dependiendo del radioButton elegido.

PMDM

Controles básicos.

6. La clase Checkbox

6.- La clase Checkbox

- Se trata de una casilla de verificación que tiene dos estados: marcado (cheked) o desmarcado (unchecked).
- Para incluir un checkbox en nuestra actividad bastará con arrastrarlo en la vista de diseño, ponerle un id y un texto.
- Al contrario de los RadioButton, son totalmente independientes unos de otros, por lo que varios de ellos pueden encontrarse seleccionados al mismo tiempo.

6.- La clase Checkbox

- Checkbox también es descendiente de TextView, por lo que podrás poner todas las características de texto que ya vimos.
- El método más importante para este tipo de view es *isChecked*, que devuelve un booleano indicando si el botón está seleccionado o no. Para saber si un Checkbox está pulsado, debemos capturarlo y utilizar el método isChecked.
- Mediante la propiedad *checkBox.isChecked* podemos modificar el estado del Checkbox, marcándolo (*checkBox.isChecked = true*) o desmarcándolo (*checkBox.isChecked = false*).

EJERCICIOS

- **Ejercicio 04:** Crea un proyecto en Android Studio que contenga una Actividad.
- Este proyecto consistirá en un ejemplo sencillo utilizando dos CheckBox:

PMDM

Controles básicos.

7. Las clases ToggleButton y Switch

7.- Las clases ToggleButton y Switch

- Son muy parecidos a los Checkbox y también tienen dos estados, encendido y apagado (on y off).

ToggleButton

Switch

- Al igual que los CheckBox, también disponen del método *isChecked* y de la propiedad *isChecked* (con la que podemos modificar el estado del componente).
- Para capturar el evento de encenderlo o apagarlo, disponemos del método *setOnCheckedChangeListener.*

7.- Las clases ToggleButton y Switch

- ToggleButton:

```
val toggleButton: ToggleButton = findViewById<ToggleButton>(R.id.toggleButton)
toggleButton.setOnCheckedChangeListener { buttonView, isChecked ->
    //¿ESTÁ ENCENDIDO?
    if (toggleButton.isChecked){
```

- Switch:

```
val switch: Switch = findViewById<Switch>(R.id.switch1)
switch.setOnCheckedChangeListener { buttonView, isChecked ->
    //¿ESTÁ ENCENDIDO?
    if (switch.isChecked){
```

EJERCICIOS

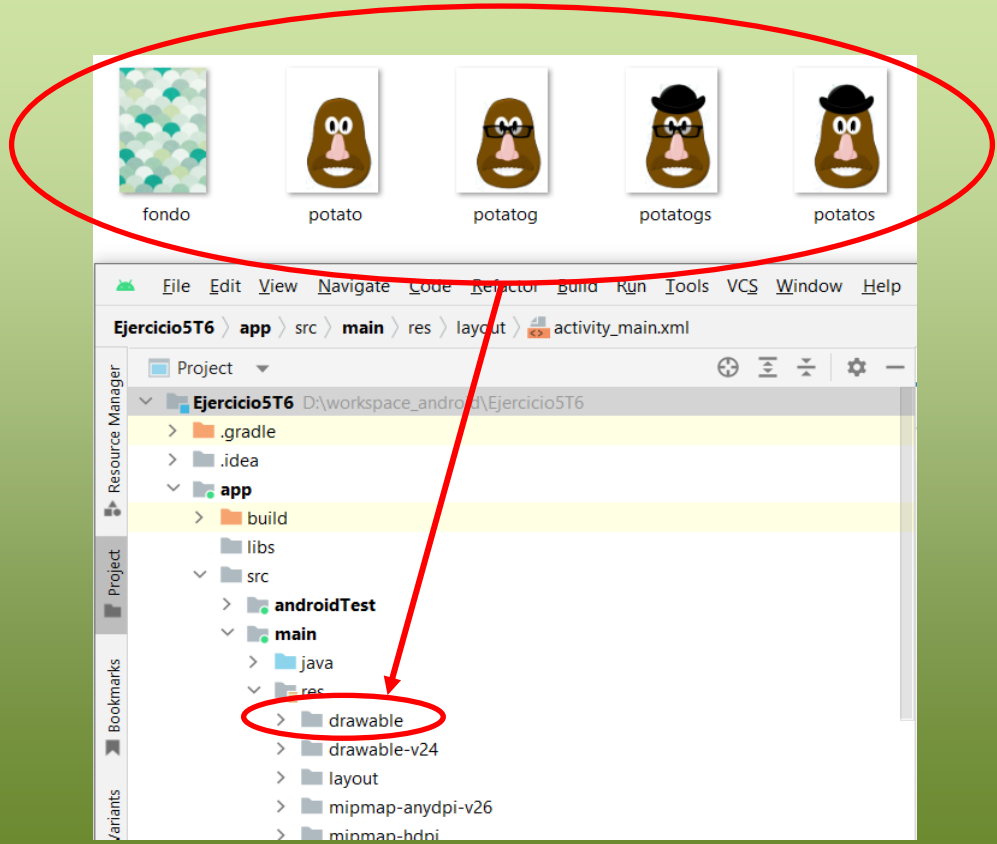
- **Ejercicio 05:** Crea un proyecto en Android Studio llamado "Mr. Potato" que contenga una Actividad.
- Este proyecto consistirá en crear una actividad que contiene un ToggleButton y un Switch:

EJERCICIOS

- Este será el resultado:

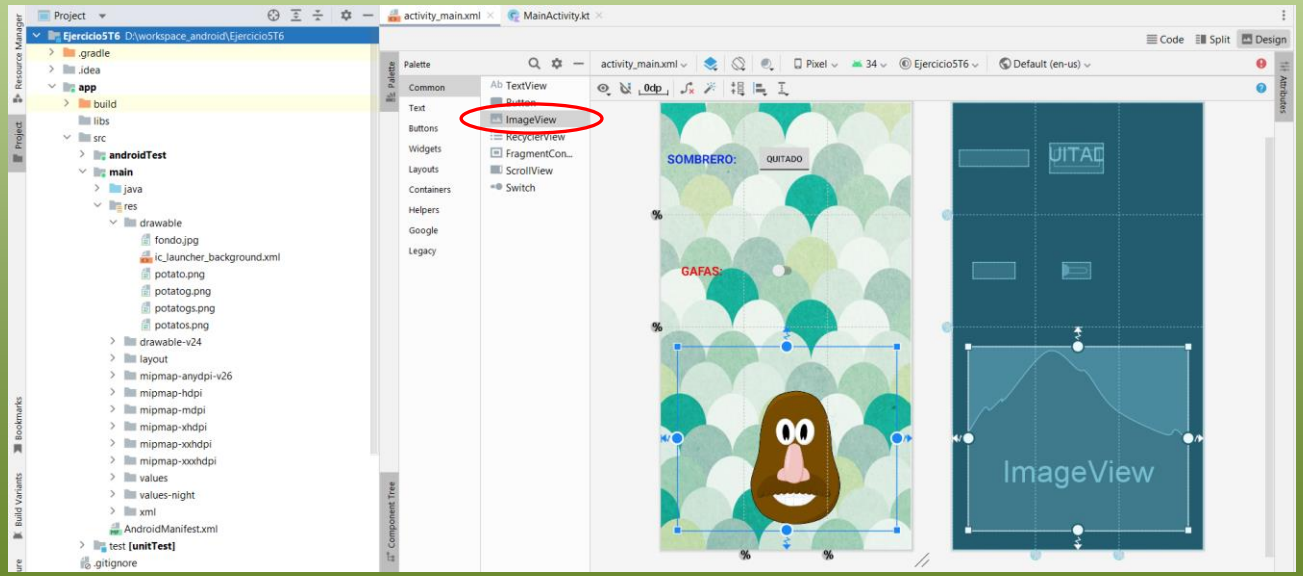
EJERCICIOS

- Mete las imágenes de "Potato" y fondo.jpg en la carpeta "drawable":
- IMPORTANTE: No poner mayúsculas en los nombres de las imágenes.



EJERCICIOS

- La imagen de "Potato" irá dentro de un **ImageView** con la propiedad android:src="@drawable/potato":



EJERCICIOS

- En el código bastará con implementar el método set*OnCheckedChangedListener* tanto para el Toggle Button como para el Switch:

```
//TOGGLE BUTTON SOMBRERO
toggleButtonSombrero.setOnCheckedChangeListener { buttonView, isChecked ->
    //SOMBRERO Y GAFAS
    if (toggleButtonSombrero.isChecked && switchGafas.isChecked) {
        imagen.setImageResource(R.drawable.potatogs)
    } else {
```

```
//SWITCH GAFAS
switchGafas.setOnCheckedChangeListener { buttonView, isChecked ->
    //SOMBRERO Y GAFAS
    if (toggleButtonSombrero.isChecked && switchGafas.isChecked) {
        imagen.setImageResource(R.drawable.potatogs)
    } else {
```

PMDM

Controles básicos.

8. Las clases ImageView e ImageButton

8.- Las clases ImageView e ImageButton

- Misma funcionalidad que TextView y Button pero con imágenes.
- **ImageView** muestra recursos de imágenes.
- **ImageButton** muestra un botón con una imagen (en lugar de texto) que el usuario puede presionar o hacer click en él.
- La imagen, tanto en el ImageView como en el ImageButton, vendrá definida por el atributo *android:src* en el XML o por el método *setImageResource(Recurso)* en el código Kotlin.

8.- Las clases ImageView e ImageButton

- Si queremos saber qué imagen tenemos cargada en un ImageView o ImageButton, lo ideal es utilizar el método *getDrawable* del componente*:*

```
val imageButton = findViewById<ImageButton>(R.id.imageButton)
// Se obtiene el recurso de la imagen actualmente asignada
val imagenActual = imageButton.drawable
// Se comprueba si la imagen actual es igual a una imagen específica
if (imagenActual.constantState == getResources().getDrawable(R.drawable.phone).constantState) {
    // La imagen actual coincide con la imagen especificada
} else {
    // La imagen actual es diferente de la imagen especificada
}
```

8.- Las clases ImageView e ImageButton

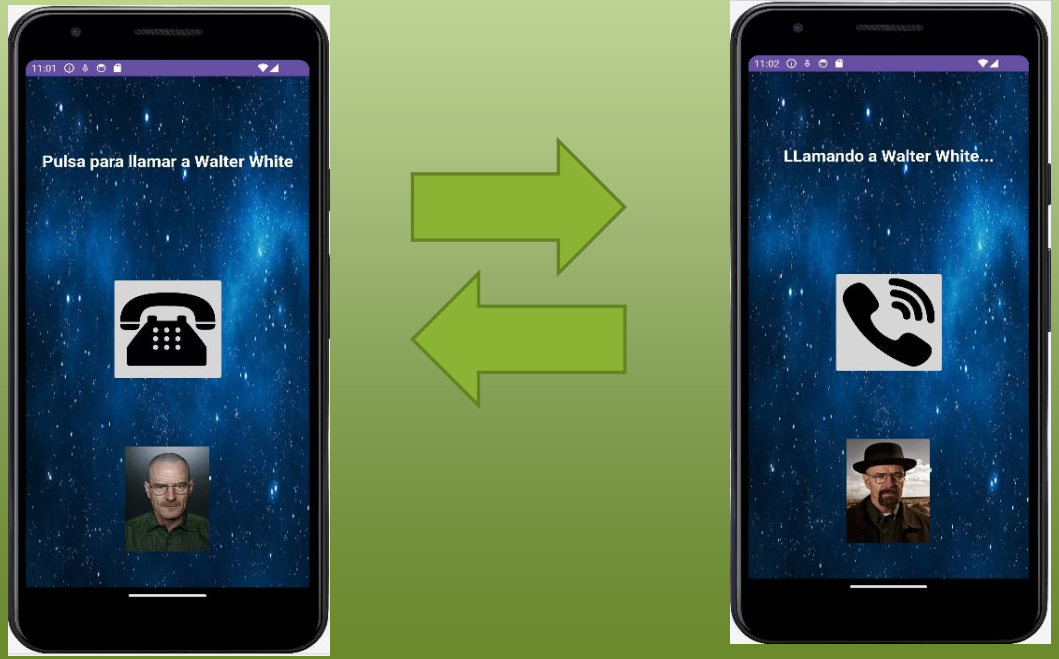
- Cuando diseñemos una aplicación profesional, debemos tener en cuenta que ésta se vea de igual forma en diferentes dispositivos. Para ello podrás crear diferentes carpetas drawable para las diferentes resoluciones, al igual que ocurre con los iconos y los layouts.

EJERCICIOS

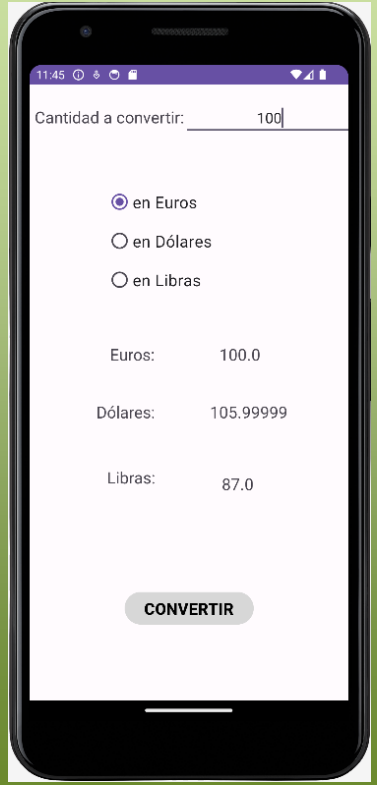
- **Ejercicio 06:** Crea un proyecto en Android Studio.
- Este proyecto consistirá en crear una actividad que muestre un texto, un botón con imagen y, debajo, otra imagen. Al pulsar el botón se cambiará el texto y las imágenes tanto del ImageView como del ImageButton. Al pulsarlo de nuevo, todo volverá a estar como al principio.

EJERCICIOS

- Quedaría de la siguiente manera:



EJERCICIOS



- **Ejercicio 07:** Crea un conversor en Android. Nuestra aplicación convertirá a euros, dólares y libras.
- Mediante radioButtons indicaremos a qué moneda pertenece la cantidad introducida.

EJERCICIOS

- Pista: El método *checkedRadioButtonId* me devuelve el botón pulsado en forma de entero. Luego, con un when, puedo programar las acciones deseadas en Kotlin:

```
val monedaSeleccionada: Int = radioGroupMonedas.checkedRadioButtonId
when(monedaSeleccionada) {
    -1 -> {
        //No se ha seleccionado ninguna moneda
    }
    R.id.radioButtonEuros -> {
        //Se ha seleccionado "Euros"
    }
    R.id.radioButtonDolar -> {
        //Se ha seleccionado "Dolar"
    }
```

PMDM

Controles básicos.

9. Ejercicios de consolidación

EJERCICIOS

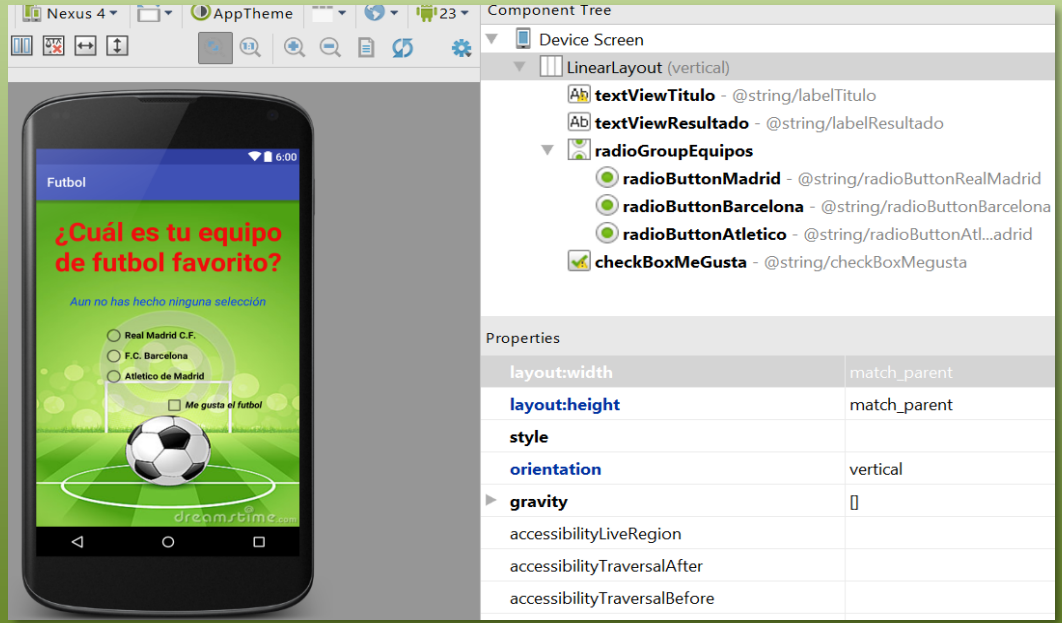
- **Ejercicio 08:** Crea un proyecto Android llamado "Preguntas Futbol" que contenga una única Actividad (Empty) llamada ActividadPrincipal.
- Este proyecto consistirá en realizar una aplicación similar a la anterior pero con la utilización de un RadioGroup con 3 RadioButtons, un Checkbox y un LinearLayout (Vertical)

EJERCICIOS

- El funcionamiento es muy sencillo. Consistirá en cambiar el contenido de un textView cuando pulsemos en algún radioButton o cuando pulsemos en el checkBox:

EJERCICIOS

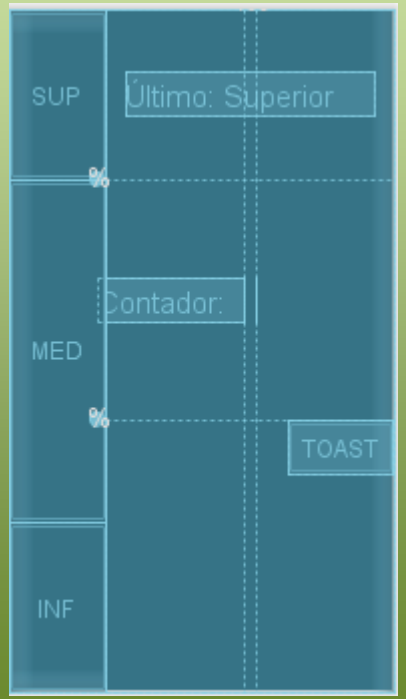
- Quedaría de la siguiente manera:



EJERCICIOS

- **Ejercicio 09:** Crea una app con una interfaz tal y como se muestra en la siguiente imagen:

EJERCICIOS



- La Activity principal estará dividida en dos partes (25% y 75%).
- El texto del título cambiará para indicar cuál de los botones de la izquierda ha sido el último en pulsarse.
- Botones "SUP", "MED" e "INF": Actualizarán el texto del título e incrementarán en 1 el contador.
- Botón TOAST: Al hacer click en él, se mostrará un Toast con el mensaje que desees.
- Helpers verticales: 48% y 52%.
- Helpers horizontales: 25% y 60%.

EJERCICIOS

- **Ejercicio 10:** Crea un proyecto en Android Studio en el que utilices la plantilla *Tabbed Activity*. Tendrá las pestañas *Personaje* y *Misión*.

EJERCICIOS

- El usuario elegirá uno de los personajes de los radioButtons.
- Si el usuario pulsa el ImageButton y no hay ningún nombre, se mostrará el Toast "*Por favor, debe indicar un nombre*".
- Si al pulsar el ImageButton existe un nombre, se mostrará un Toast con el mensaje "*Personaje [nombre] registrado como [tipo]*".

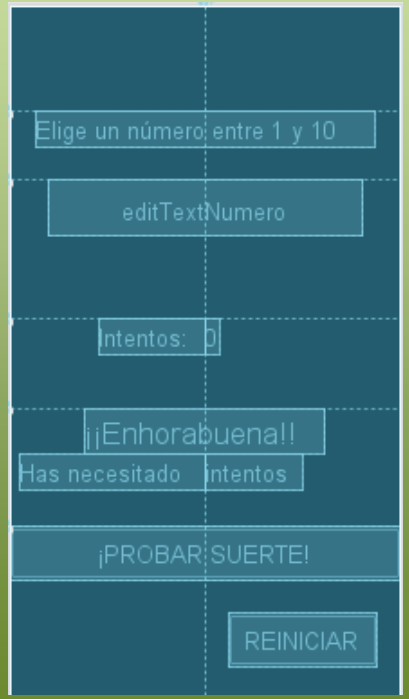
EJERCICIOS

- Si el usuario pulsa el ToggleButton se mostrará *Multijugador: Off* o *Multijugador: On*.
- Al pulsar el botón "Elegir" se mostrará un mensaje con las misiones seleccionadas: "*Elegida/s misión/es Francia, Noruega*".
- Si pulsamos "Elegir" sin que haya seleccionada ninguna misión, mostraremos "*Debe elegir al menos una misión*".

EJERCICIOS

- **Ejercicio 11:** Crea una app en Android Studio que juegue con el usuario a adivinar un nùmero.
- La aplicación generará un número aleatorio entre 1 y 10.

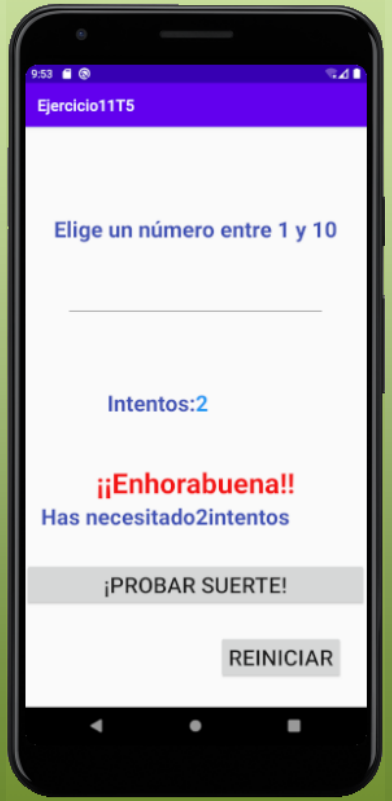
EJERCICIOS



- Las guías horizontales estarán al 15%, 25%, 45%, 58% y 75%.
- Cada vez que el usuario pulse el botón "Probar suerte", se actualizará el número de intentos y se mostrará un Toast con una pista:
  - *"Pista: Introduce un número menor"*
  - *"Pista: Introduce un número mayor"*

EJERCICIOS

- Cuando el usuario acierte, se le mostrará el mensaje de enhorabuena junto con el número de intentos totales que ha necesitado.

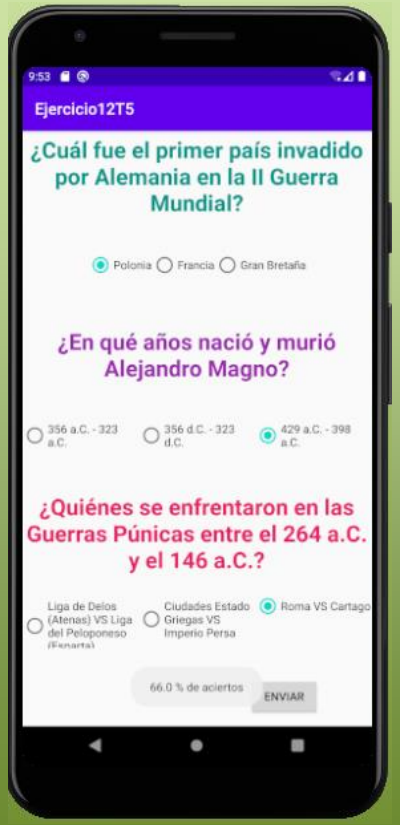


EJERCICIOS

- Cuando pulse "Reiniciar", se generará un nuevo número aleatorio y se inicializará el número de intentos.
- Además, los mensajes de "enhorabuena" y el número de intentos, desaparecerán.

EJERCICIOS

- **Ejercicio 12:** Crea un trivial con 3 preguntas.
- Al pulsar el botón "Enviar" se mostrará un Toast con la puntuación total, tal y como se muestra en la imagen.
- Busca en Internet cómo situar los radioButtons en horizontal.



EJERCICIOS

- Cada pregunta, junto con sus respuestas, se situarán dentro de un Constraint Layout.
- Los distintos layouts están remarcados en rojo y tienen un peso de 30, 30, 30 y 10.
- Dentro de cada Constraint Layout, dispondremos de una guía horizontal al 50%.