



cpi'fp

Los Enlaces

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

2º DESARROLLO DE APLICACIONES MULTIPLATAFORMA

TEMA 3. MENÚ Y NUEVA APLICACIÓN

0. MENÚ ENTRE APLICACIONES

Una vez creadas nuestras dos primeras aplicaciones dentro del mismo proyecto, podemos añadir un menú inicial para seleccionar una de ellas:

Empty Views Activity

Creates a new empty activity

Activity Name

☒ Generate a Layout File

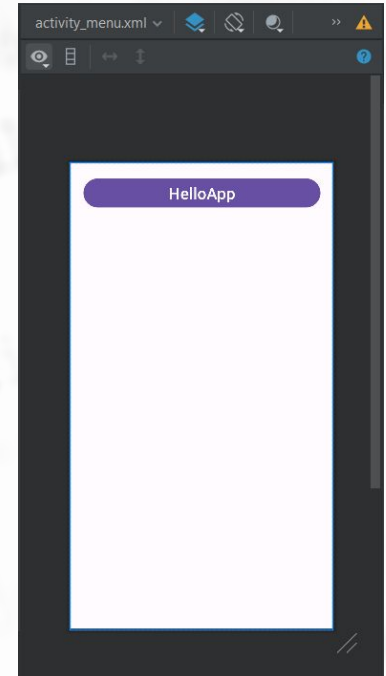
Layout Name

☒ Launcher Activity

0. MENÚ ENTRE APLICACIONES

Para ello, diseñamos un LinearLayout con un botón cuyo texto sea el nombre de la primera aplicación:

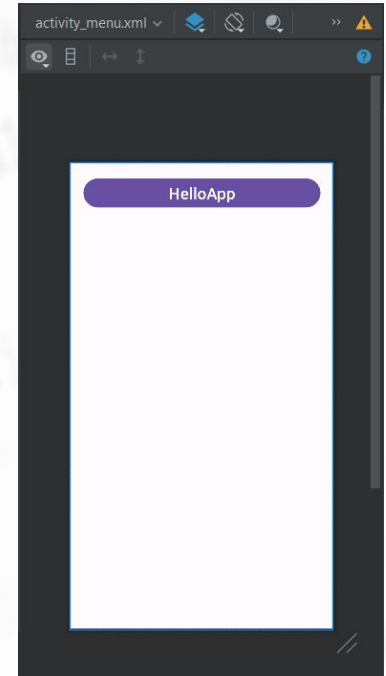
```
<Button
    android:id="@+id/btnHelloApp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="HelloApp"
    android:textSize="25sp"
    android:layout_marginHorizontal="20dp"
    android:layout_marginTop="20dp"/>
```



0. MENÚ ENTRE APLICACIONES

Y en la parte lógica creamos un Intent mediante una función externa que llame a la actividad principal de dicha aplicación:

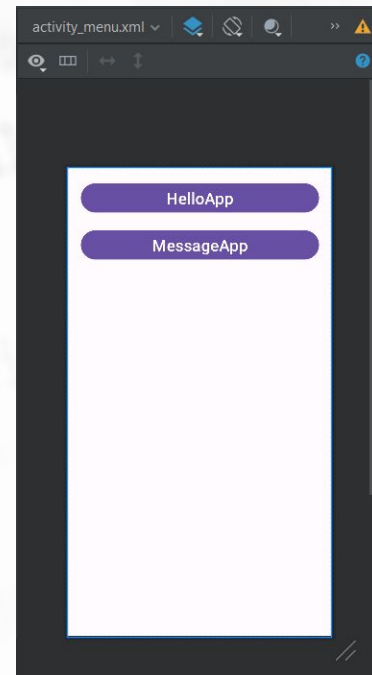
```
//Dentro de la función onCreate()  
    var btnHelloApp = findViewById<Button>(R.id.btnHelloApp)  
    btnHelloApp.setOnClickListener{ navigateToHelloApp() }  
}  
//Fuera de la función onCreate()  
private fun navigateToHelloApp(){  
    var intent = Intent(this,NameActivity::class.java)  
    startActivity(intent)  
}  
}
```



0. MENÚ ENTRE APLICACIONES

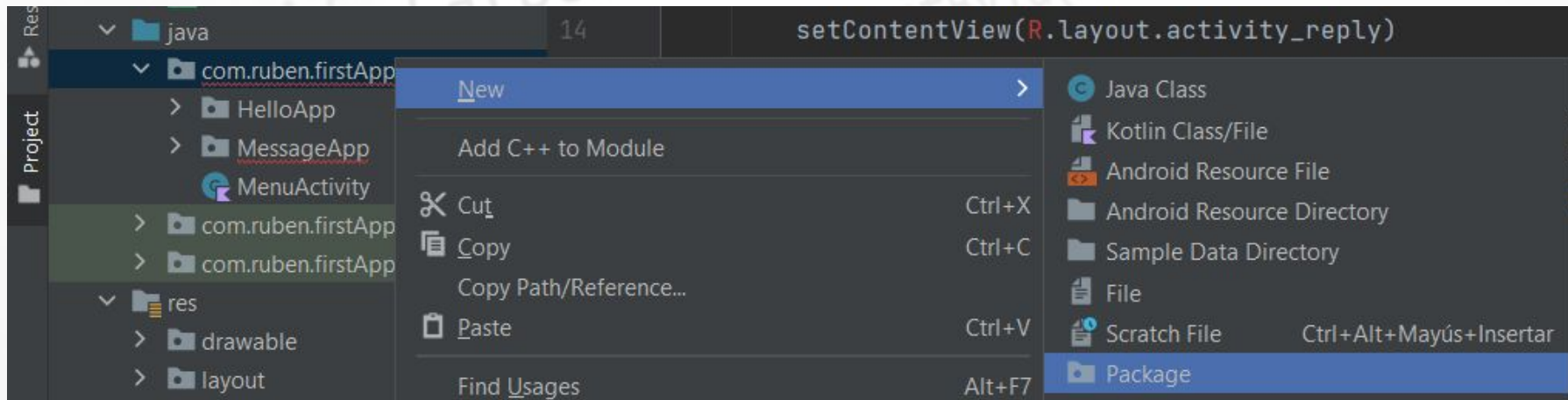
Repite el proceso para la aplicación MessageApp que has desarrollado en el tema anterior.

```
//Dentro de la función onCreate()  
var btnHelloApp = findViewById<Button>(R.id.btnHelloApp)  
var btnMessageApp = ...  
  
btnHelloApp.setOnClickListener{ navigateToHelloApp() }  
btnMessageApp.setOnClickListener{ ... }  
}  
}
```



1. ORGANIZACIÓN DE UN PROYECTO

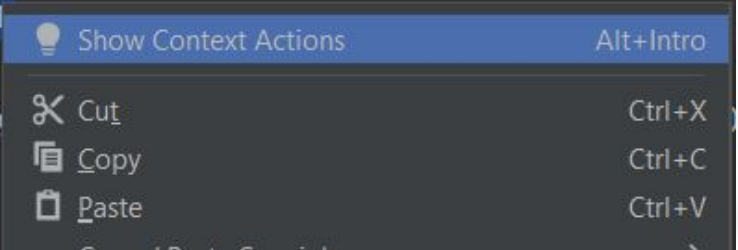
Lo ideal al crear una aplicación con varios apartados sería distribuir las actividades por directorios, así que vamos a crearlos con el menú desplegable de nuestro espacio de nombres:



1. ORGANIZACIÓN DE UN PROYECTO

Al arrastrar las actividades creadas a sus directorios correspondientes se producirá un error de detección de la clase R que habrá que importar en cada una de ellas (botón derecho → Show Context Actions).

```
12 | override fun onCreate(savedInstanceState: Bundle?) {  
13 |     super.onCreate(savedInstanceState)  
14 |     setContentView(R.layout.activity_reply)  
15 |  
16 |     var msgReceived = findViewById<TextView>(R.id.msg_received)  
17 |     var txtMessage = findViewById<TextView>(R.id.txt_message)  
18 |     var userMessage: String = intent.extras?.getString(R.id.user_message)  
19 |  
20 |     if (userMessage.isNotEmpty()) {
```



- Show Context Actions (Alt+Intro)
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)

2. APLICACIÓN IMC

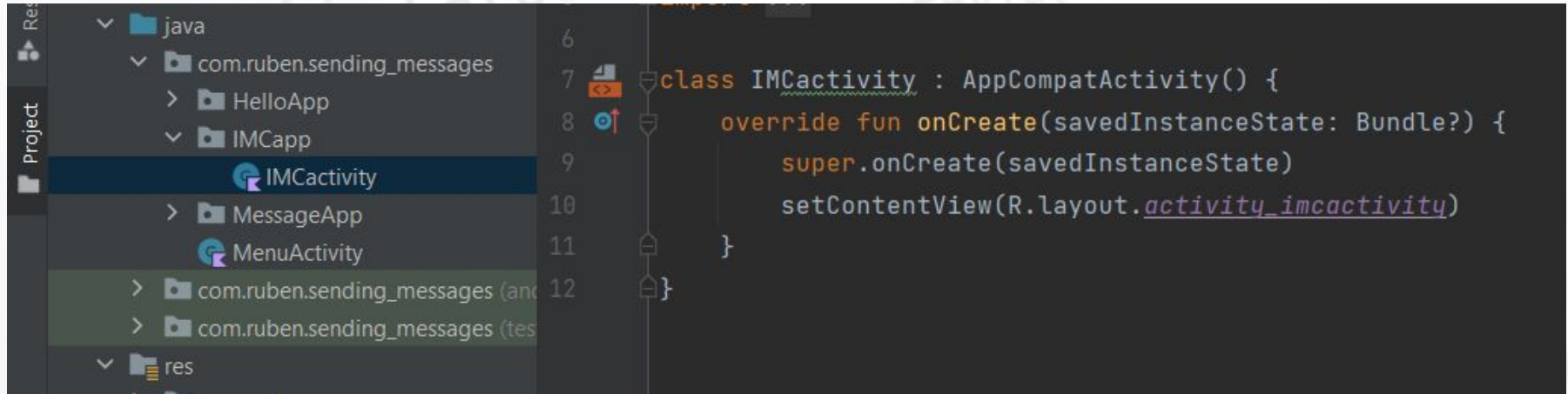
Una medida de la obesidad se determina mediante el índice de masa corporal (IMC), que se calcula dividiendo los kilogramos de peso entre el cuadrado de la estatura en metros:

$$\text{IMC} = \text{peso (kg)} / [\text{estatura (m)}]^2$$

Según el Instituto Nacional del Corazón, los Pulmones y la Sangre de los Estados Unidos (NHLBI), el sobrepeso se define como un IMC de más de 25. Se considera que una persona es obesa si su IMC es superior a 30 y que su peso es inferior al normal si está por debajo de 18'5.

2. LAYOUT APLICACIÓN IMC

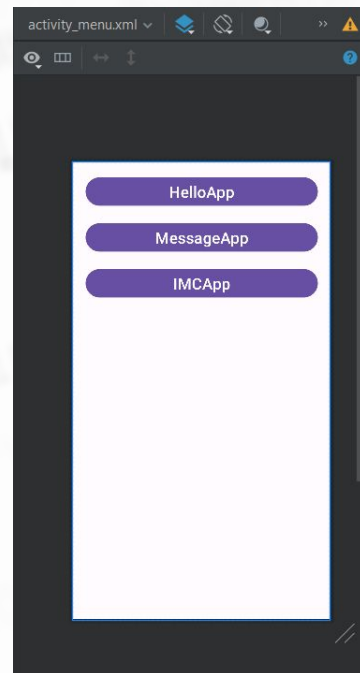
Una vez comprobado que el menú y las anteriores aplicaciones funcionan correctamente, creamos una nueva actividad para la aplicación de cálculo del índice de masa corporal:



2. APLICACIÓN IMC

Podemos incorporar ya a nuestro menú el botón de acceso a esta nueva aplicación, con su función *navigateToIMCApp()* correspondiente:

```
//Dentro de la función onCreate()  
var btnHelloApp = findViewById<Button>(R.id.btnHelloApp)  
var btnMessageApp = ...  
var btnImcApp = ...  
  
btnHelloApp.setOnClickListener{ navigateToHelloApp() }  
btnMessageApp.setOnClickListener{ ... }  
btnImcApp.setOnClickListener{ ... }  
  
}
```



3. LAYOUT APLICACIÓN IMC

En esta aplicación vamos a crear un diseño más vistoso con nuevos elementos e insertando algunas imágenes. En la carpeta correspondiente a este tema en *aeducar* encontrareis las imágenes necesarias.

Desde Android Studio podemos insertar imágenes vectoriales en el directorio */res/drawable* desde el menú desplegable New → Vector Asset.

Elegimos la opción *Local file* y seleccionamos la ruta a la imagen que queremos insertar.

De esta manera, tendremos disponibles en este directorio las imágenes que queramos usar en nuestro layout.

3. LAYOUT APLICACIÓN IMC

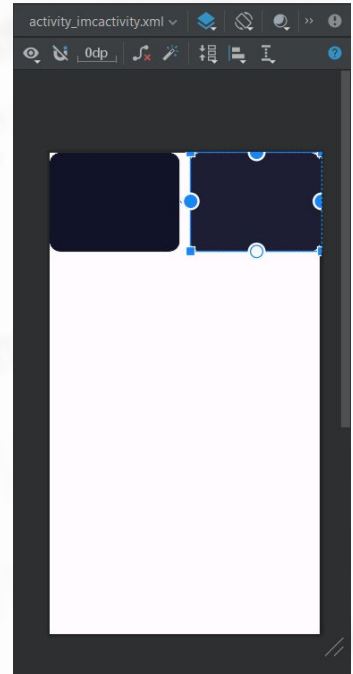
Podemos crear nuestra propia paleta de colores en el archivo `colors.xml` dentro del directorio `/res/values`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="background_component">#1D1E33</color>
    <color name="background_component_selected">#FF111328</color>
    <color name="background_app">#0E0B20</color>
    <color name="background_button">#EB1555</color>
    <color name="title_text">#FF8D8E98</color>
    <color name="background_fab">#FF6200EE</color>
</resources>
```

3. LAYOUT APLICACIÓN IMC

Vamos a empezar el layout creando dos elementos CardView que permiten introducir otros elementos dentro de ellos y tienen diversas propiedades:

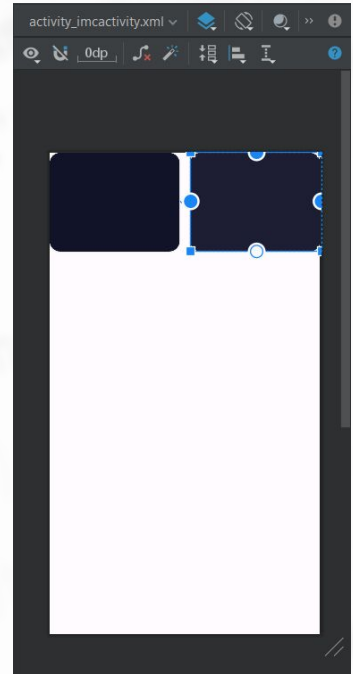
```
<androidx.cardview.widget.CardView
    android:id="@+id/viewMale"
    android:layout_width="0dp"
    android:layout_height="150dp"
    android:layout_marginEnd="16dp"
    app:cardBackgroundColor="@color/background_component_selected"
    app:cardCornerRadius="16dp"
    app:layout_constraintEnd_toStartOf="@+id/viewFemale"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">
```



3. LAYOUT APLICACIÓN IMC

El segundo elemento no tiene margen a la izquierda porque con los anclajes ya se reparten la anchura total entre los dos elementos y el margen aplicado al primero.

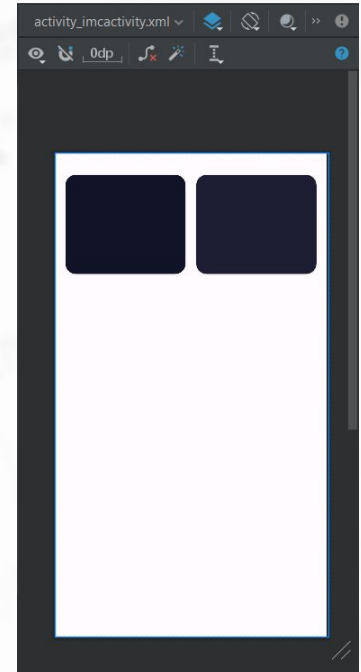
```
<androidx.cardview.widget.CardView
    android:id="@+id/viewFemale"
    android:layout_width="0dp"
    android:layout_height="150dp"
    app:cardBackgroundColor="@color/background_component"
    app:cardCornerRadius="16dp"
    app:layout_constraintStart_toEndOf="@+id/viewMale"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent">
```



3. LAYOUT APLICACIÓN IMC

Para que estos CardViews no queden pegados a los bordes de la pantalla, aplicamos un padding horizontal y otro vertical al ConstraintLayout:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingHorizontal="16dp"
    android:paddingVertical="32dp"
    tools:context=".IMCapp.IMCActivity">
```

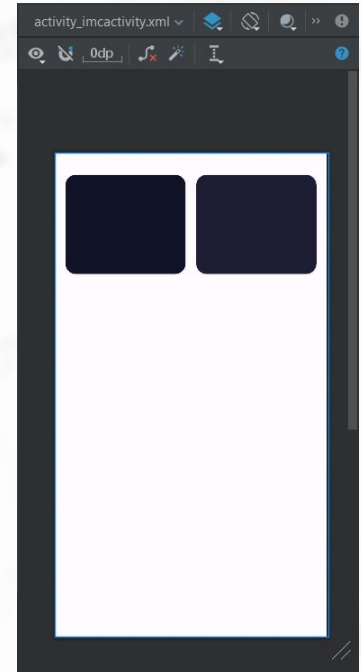


3. LAYOUT APLICACIÓN IMC

Lo interesante de los CardViews es que podemos insertar otros elementos dentro de ellos y maquetarlos con layouts anidados:

```
<androidx.cardview.widget.CardView
...>
<androidx.appcompat.widget.LinearLayoutCompat
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">

    </androidx.appcompat.widget.LinearLayoutCompat>
</androidx.cardview.widget.CardView>
```

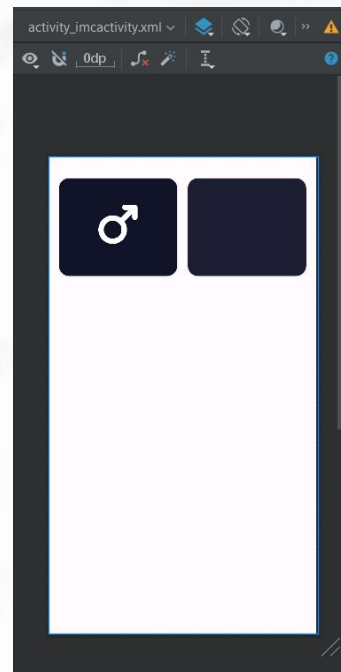


3. LAYOUT APLICACIÓN IMC

Para poder insertar una imagen debemos hacer uso del elemento `ImageView`, en cuyo atributo `src` se le indica la ruta al directorio `/res/drawable` que hemos comentado:

```
<LinearLayout
    ...>
    <ImageView
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:layout_marginBottom="8dp"
        android:src="@drawable/simbolo_masculino"
        app:tint="@color/white" />

</LinearLayout>
```

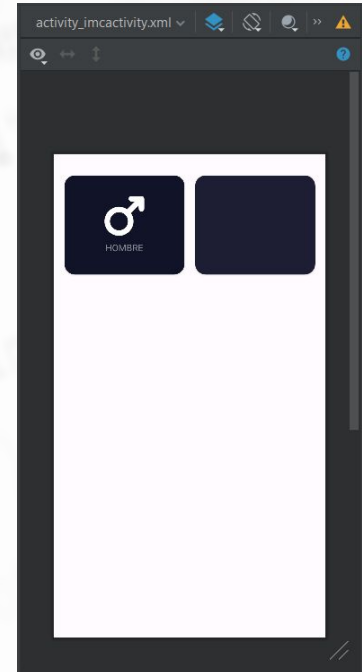


3. LAYOUT APLICACIÓN IMC

Ahora añadimos un texto debajo utilizando la opción *Extract string resource* en el atributo *text* a través de *Show Content Actions* (Alt + Intro).

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/male"
    android:textAllCaps="true"
    android:textColor="@color/title_text" />

</androidx.appcompat.widget.LinearLayoutCompat>
```

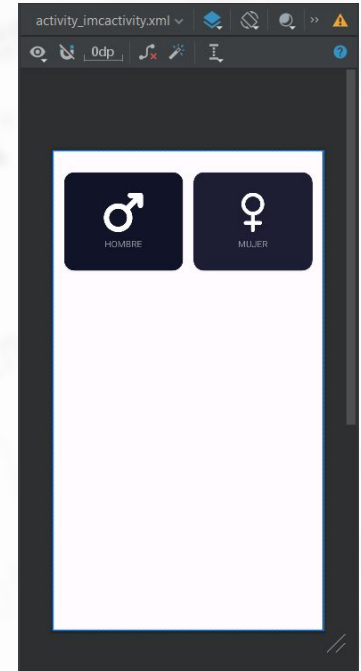


3. LAYOUT APLICACIÓN IMC

Repite los pasos realizados para esta CardView en la de la derecha utilizando la imagen *simbolo_femenino.xml* (mirar en */res/drawable*) y el texto *Mujer*.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/female"
    android:textAllCaps="true"
    android:textColor="@color/title_text" />

</androidx.appcompat.widget.LinearLayoutCompat>
```



3. LAYOUT APLICACIÓN IMC

Creamos ahora otra CardView por debajo de las anteriores que ocupe todo el ancho de la pantalla y cuya altura se adapte al contenido:

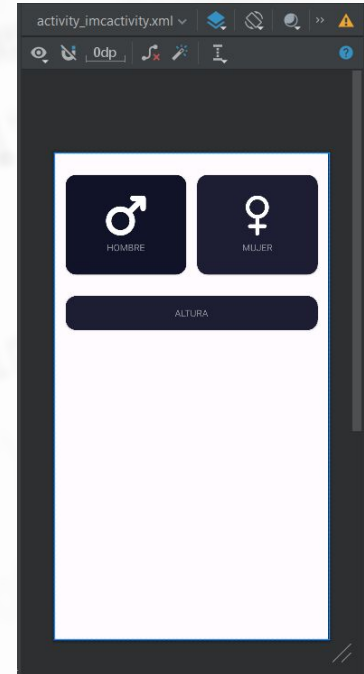
```
<androidx.cardview.widget.CardView
    android:id="@+id/viewHeight"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    app:cardBackgroundColor="@color/background_component"
    app:cardCornerRadius="16dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/viewMale">
```



3. LAYOUT APLICACIÓN IMC

Al insertar un `TextView` entre las etiquetas de apertura y cierre de la `CardView` con el texto *Altura* en su atributo `text` podremos visualizarla:

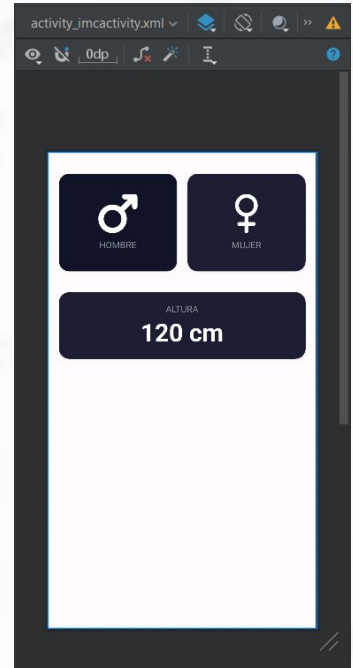
```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/height"  
    android:textAllCaps="true"  
    android:labelFor="@id/rsHeight"  
    android:textColor="@color/title_text" />
```



3. LAYOUT APLICACIÓN IMC

Añadimos además un TextView con un texto provisional de 120 cm que posteriormente podrá modificar el usuario con el siguiente elemento que veremos:

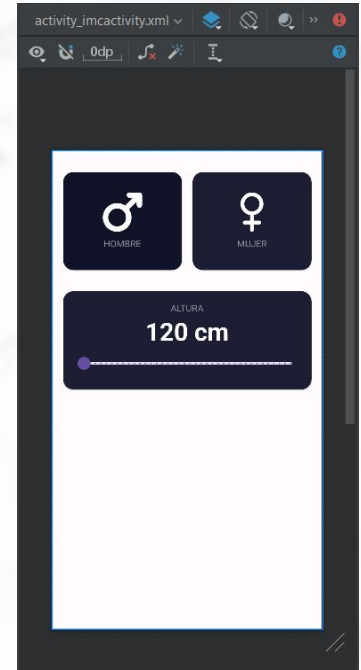
```
<TextView
    android:id="@+id/tvHeight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="120 cm"
    android:textColor="@color/white"
    android:textSize="38sp"
    android:textStyle="bold" />
```



3. LAYOUT APLICACIÓN IMC

El elemento `RangeSlider` inserta una barra con un puntero desplazable que nos permite cambiar un valor en un rango dado y con el intervalo que le indiquemos:

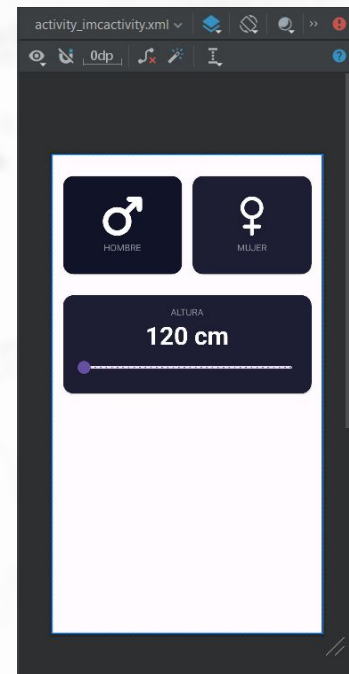
```
<com.google.android.material.slider.RangeSlider  
    android:id="@+id/rsHeight"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:stepSize="1"  
    android:valueFrom="120"  
    android:valueTo="200"  
    app:thumbColor="@color/background_fab" />
```



3. LAYOUT APLICACIÓN IMC

Si Android Studio no detecta automáticamente el elemento RangeSlider, sustituir en Gradle Scripts → build.gradle.kts (Module:app) → dependencies...

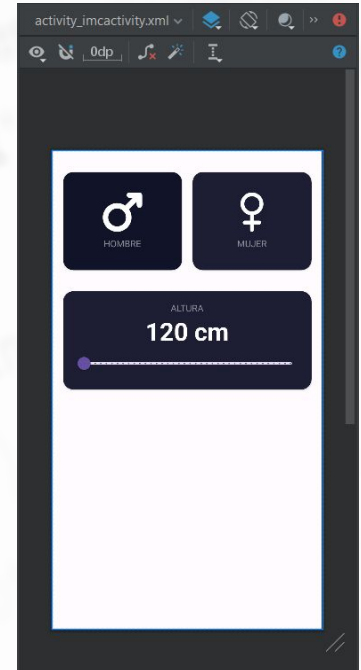
```
dependencies {  
    implementation("androidx.core:core-ktx:1.9.0")  
    implementation("androidx.appcompat:appcompat:1.6.1")  
    implementation("com.google.android.material:material:1.9.0")  
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")  
    testImplementation("junit:junit:4.13.2")  
    androidTestImplementation("androidx.test.ext:junit:1.1.5")  
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")  
}
```



3. LAYOUT APLICACIÓN IMC

Todos los elementos del layout que se hayan creado a partir de las librerías que tuviéramos cargadas habrá que sustituirlos por los incluidos en éstas.

```
dependencies {  
    implementation("androidx.core:core-ktx:1.9.0")  
    implementation("androidx.appcompat:appcompat:1.6.1")  
    implementation("com.google.android.material:material:1.9.0")  
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")  
    testImplementation("junit:junit:4.13.2")  
    androidTestImplementation("androidx.test.ext:junit:1.1.5")  
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")  
}
```

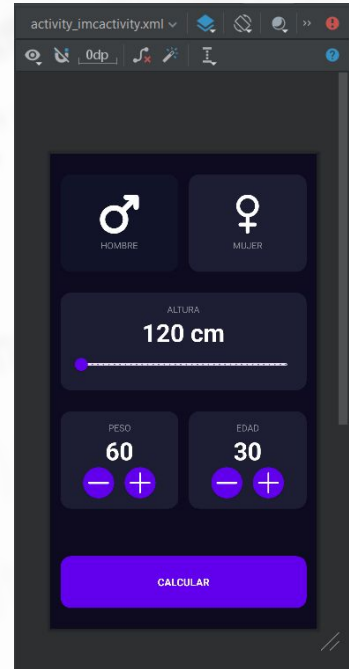


3. LAYOUT APLICACIÓN IMC

Termina el diseño de la aplicación guiándote con la imagen de la derecha. Todos los elementos son CardViews con LinearLayouts anidados y colores suministrados en la paleta de colores del proyecto.

Asigna una *id* a cada elemento y aplica márgenes adecuados para que el diseño quede uniforme.

Los botones con imágenes + y – tienen altura y anchura ajustadas al contenido y margen de 8dp en las ImageViews, de anchura y altura 32dp .



4. LÓGICA APLICACIÓN IMC

Una vez tenemos el diseño creado, vamos a crear los objetos de los dos primeros Views. Los vamos a declarar fuera de la función onCreate() para tratarlos como variables globales.

```
class IMCactivity : AppCompatActivity() {  
  
    private lateinit var viewMale: CardView  
    private lateinit var viewFemale: CardView  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_imcactivity)  
    }  
}
```

4. LÓGICA APLICACIÓN IMC

Así podremos utilizarlos en cualquier función que definamos para esta actividad. Los inicializaremos posteriormente dentro de la función `onCreate()` de la siguiente manera:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_imcactivity)  
    initComponents()  
    initListeners()  
}  
  
private fun initComponents() {  
    viewMale = findViewById(R.id.viewMale)  
    viewFemale = findViewById(R.id.viewFemale)  
}
```

4. LÓGICA APLICACIÓN IMC

Y haremos que dichos objetos se mantengan a la espera de que el usuario interactúe con ellos definiendo una nueva función `initListeners()`:

```
private fun initComponents() {  
    viewMale = findViewById(R.id.viewMale)  
    viewFemale = findViewById(R.id.viewFemale)  
}  
  
private fun initListeners() {  
    viewMale.setOnClickListener { }  
    viewFemale.setOnClickListener { }  
}
```

4. LÓGICA APLICACIÓN IMC

Ahora queremos que estas CardViews cambien de color según cual esté seleccionada. Declaramos entonces dos variables booleanas inicializadas según los colores que les hayamos asignado en el layout:

```
class IMCactivity : AppCompatActivity() {  
  
    private var isMaleSelected: Boolean = true  
    private var isFemaleSelected: Boolean = false  
  
    private lateinit var viewMale: CardView  
    private lateinit var viewFemale: CardView  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
    }  
}
```

4. LÓGICA APLICACIÓN IMC

Dentro del método `setOnClickListener()` definimos entonces una nueva función para el cambio de color `setComponentColor()`. Haremos una por cada View evaluando si el componente está seleccionado o no:

```
viewMale.setOnClickListener { setComponentColorMale() }  
  
private fun setComponentColorMale () {  
    if (!isMaleSelected) {  
        viewMale.setCardBackgroundColor(getColor(R.color.background_component_selected))  
        viewFemale.setCardBackgroundColor(getColor(R.color.background_component))  
        isMaleSelected = true  
        isFemaleSelected = false  
    }  
}
```

4. LÓGICA APLICACIÓN IMC

Para que el usuario pueda indicar su altura mediante el RangeSlider agregado al layout y que ésta se muestre en el TextView correspondiente debemos declarar primero estos objetos e inicializarlos:

```
private lateinit var tvHeight: TextView
private lateinit var rsHeight: RangeSlider
...

private fun initComponents() {
    viewMale = findViewById(R.id.viewMale)
    viewFemale = findViewById(R.id.viewFemale)
    tvHeight = findViewById(R.id.tvHeight)
    rsHeight = findViewById(R.id.rsHeight)
}
```


4. LÓGICA APLICACIÓN IMC

El método para interactuar con el RangeSlider es `addOnChangeListener()` y al empezar a escribirlo en AndroidStudio elegiremos la tercera opción, que incluye 3 variables con información del slider:

```
private fun initListeners() {  
    viewMale.setOnClickListener { setComponentColorMale() }  
    viewFemale.setOnClickListener { setComponentColorFemale() }  
    rsHeight.addOn|  
}  
  
private fun
```

- `addOnChangeListener(listener: RangeSlider.OnChange... Unit`
- `addOnChangeListener(listener: (RangeSlider, Float,... Unit`
- `addOnChangeListener { slider, value, fromUser -> ... } (listener`
- `addOnAttachStateChangeListener(listener: View.OnAtt Unit`

4. LÓGICA APLICACIÓN IMC

De estas 3 variables solo nos interesa la que almacena el valor indicado en el RangeSlider, las otras dos las omitiremos con barra baja, “_”. La variable *value* es tipo *float* y debemos transformarlo a *string*.

```
private fun initListeners() {  
    viewMale.setOnClickListener { setComponentColorMale() }  
    viewFemale.setOnClickListener { setComponentColorFemale() }  
    rsHeight.addOnChangeListener { _, value, _ ->  
        tvHeight.text = value.toString()  
    }  
}
```

4. LÓGICA APLICACIÓN IMC

Si lanzamos la aplicación veremos que el formato que obtenemos es un número `##` sin el texto *cm* a continuación. Para darle un formato más adecuado aplicamos la función `DecimalFormat()` y el método `format()`:

```
private fun initListeners() {  
    viewMale.setOnClickListener { setComponentColorMale() }  
    viewFemale.setOnClickListener { setComponentColorFemale() }  
    rsHeight.addOnChangeListener { _, value, _ ->  
        val df = DecimalFormat("#")  
        val result = df.format(value)  
        tvHeight.text = "$result cm"  
    }  
}
```

4. LÓGICA APLICACIÓN IMC

En el siguiente componente, el CardView donde el usuario indicará su peso, necesitamos crear 3 objetos: el TextView con el peso en kg y los botones - y + para variar el peso.

```
private lateinit var viewMale: CardView
private lateinit var viewFemale: CardView
private lateinit var tvHeight: TextView
private lateinit var rsHeight: RangeSlider
private lateinit var tvWeight: TextView
private lateinit var btnSubtractWeight: CardView
private lateinit var btnAddWeight: CardView
```

4. LÓGICA APLICACIÓN IMC

Podemos crear una variable con el valor inicial del peso:

```
private var currentWeight: Int = 70
```

E inicializamos los objetos definidos:

```
private fun initComponents() {  
    viewMale = findViewById(R.id.viewMale)  
    viewFemale = findViewById(R.id.viewFemale)  
    tvHeight = findViewById(R.id.tvHeight)  
    rsHeight = findViewById(R.id.rsHeight)  
    tvWeight = findViewById(R.id.tvWeight)  
    btnSubstractWeight = findViewById(R.id.btnSubstractWeight)  
    btnAddWeight = findViewById(R.id.btnAddWeight)  
}
```

4. LÓGICA APLICACIÓN IMC

Ahora queda inicializar los Listeners, cuya función será modificar la variable *currentWeight* según pulsemos un botón u otro. Creamos a su vez una función *setWeight()* que muestre el peso en el TextView.

```
btnSubstractWeight.setOnClickListener {  
    currentWeight -= 1  
    setWeight()  
}  
btnAddWeight.setOnClickListener {  
    currentWeight += 1  
    setWeight()  
}  
}  
private fun setWeight() { tvWeight.text = currentWeight.toString() }
```

4. LÓGICA APLICACIÓN IMC

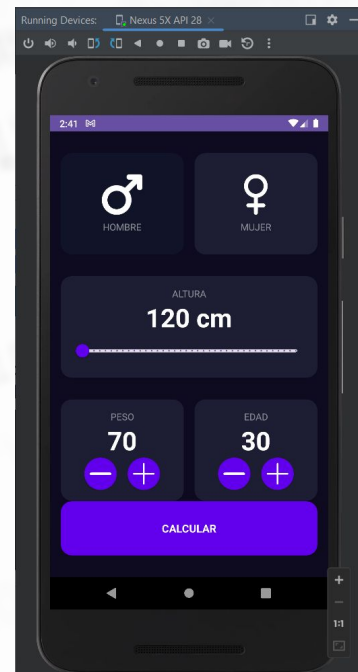
Si lanzamos la aplicación comprobaremos que inicialmente no se muestra ningún peso hasta que no pulsamos alguno de los botones. Para que se muestre basta con insertar `setWeight()` en la función `onCreate()`.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_imcactivity)  
  
    initComponents()  
    initListeners()  
    setWeight()  
}
```

4. LÓGICA APLICACIÓN IMC

Repite este último proceso para el CardView que indica la edad del usuario con una variable *currentAge* inicializada a 30.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_imcactivity)  
  
    initComponents()  
    initListeners()  
    setWeight()  
    setAge()  
}
```



4. LÓGICA APLICACIÓN IMC

Ya solo nos queda configurar la función del botón *calculate* de nuestro layout. Creamos su variable correspondiente e inicializamos el objeto. Ahora el método `setOnClickListener()` incluirá las siguientes funciones:

```
btnCalculate.setOnClickListener {  
    val result = calculateIMC()  
    navigateToResult(result)  
}
```

Para calcular el IMC nos va a hacer falta una variable *currentHeight* que almacene la altura indicada por el usuario:

```
private var currentHeight: Int = 120
```

4. LÓGICA APLICACIÓN IMC

Esta última variable deberemos usarla dentro del método `addOnChangeListener()` asociado al `RangeSlider` para que vaya almacenando el valor de la altura. Hay que convertir a *int* el formato.

```
private fun initListeners() {  
    viewMale.setOnClickListener { setComponentColorMale() }  
    viewFemale.setOnClickListener { setComponentColorFemale() }  
    rsHeight.addOnChangeListener { _, value, _ ->  
        val df = DecimalFormat("#")  
        currentHeight = df.format(value).toInt()  
        tvHeight.text = "$currentHeight cm"  
    }  
}
```

4. LÓGICA APLICACIÓN IMC

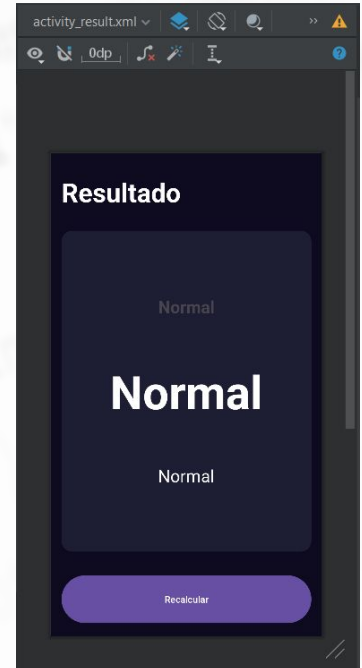
Con el peso y la altura almacenados en sus variables correspondientes ya podemos definir la función `calculateIMC()` que devolverá el valor del cálculo realizado.

```
private fun calculateIMC():Double {  
    val df = DecimalFormat("#.##")  
    val imc = currentWeight / pow(currentHeight.toDouble() /100, 2.0)  
  
    return df.format(imc).toDouble()  
}  
  
// Log.i("IMC", "El IMC es $result") en el método  
// setOnClickListener para mostrarlo por consola
```

5. LAYOUT RESULT ACTIVITY

Una vez tenemos definida la actividad que calcula el IMC, nos toca preparar la pantalla que mostrará este resultado. Creamos entonces la actividad ResultActivity.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/background_app"
    android:paddingHorizontal="16dp"
    android:paddingVertical="32dp"
    tools:context=".IMCapp.ResultActivity">
```



5. LAYOUT RESULT ACTIVITY

El primer elemento que insertaremos en el layout será un TextView con el texto “Resultado” alineado arriba a la izquierda:

```
<TextView
    android:id="@+id/tvTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/result"
    android:textColor="@color/white"
    android:textSize="40sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



5. LAYOUT RESULT ACTIVITY

A la seguida un CardView que irá anclado por arriba al TextView que hace de título y por abajo a un botón “Recalcular” que insertaremos a la seguida.

```
<androidx.cardview.widget.CardView
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:cardBackgroundColor="@color/background_component"
    app:cardCornerRadius="16dp"
    android:layout_marginVertical="32dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/btnRecalculate"
    app:layout_constraintTop_toBottomOf="@+id/tvTitle">
```



5. LAYOUT RESULT ACTIVITY

Vemos como el CardView anterior se ajusta ahora a los anclajes asignados y solo nos queda modificar el TextView del botón para cambiar su texto.

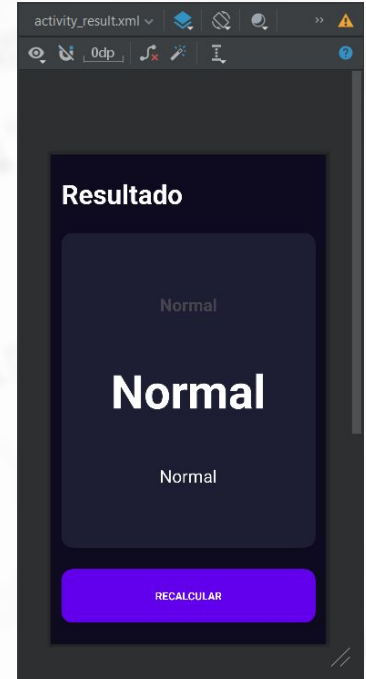
```
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:text="@string/recalculate"
    android:textAllCaps="true"
    android:textSize="16sp"
    android:textStyle="bold"
    android:textColor="@color/white"/>
```



5. LAYOUT RESULT ACTIVITY

Dentro del CardView intermedio vamos a introducir tres textos para mostrar los resultados del cálculo del IMC con un LinearLayout vertical.

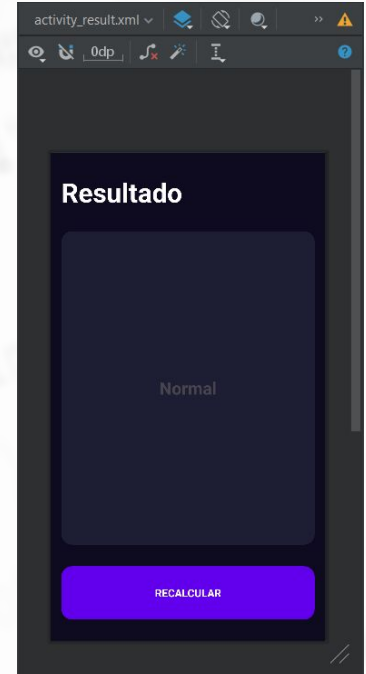
```
<androidx.appcompat.widget.LinearLayoutCompat  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:layout_margin="16dp"  
    android:gravity="center"  
    android:orientation="vertical">
```



5. LAYOUT RESULT ACTIVITY

El primero de ellos le indicará al usuario si su peso es normal o si está por encima o por debajo del IMC recomendado para su altura.

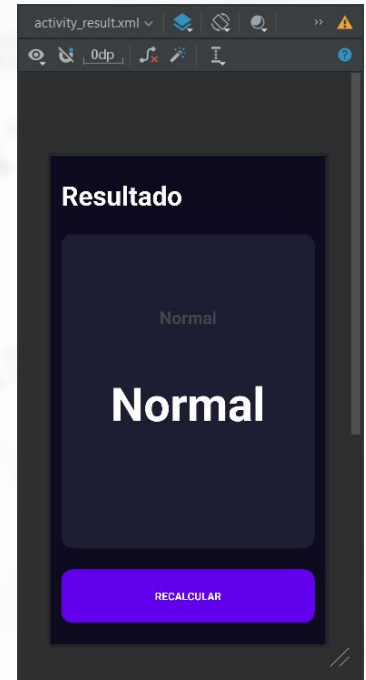
```
<TextView
    android:id="@+id/tvResult"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="26sp"
    android:textStyle="bold"
    tools:text="Normal" />
```



5. LAYOUT RESULT ACTIVITY

El segundo mostrará el cálculo del IMC realizado por la aplicación, es decir, la valoración numérica.

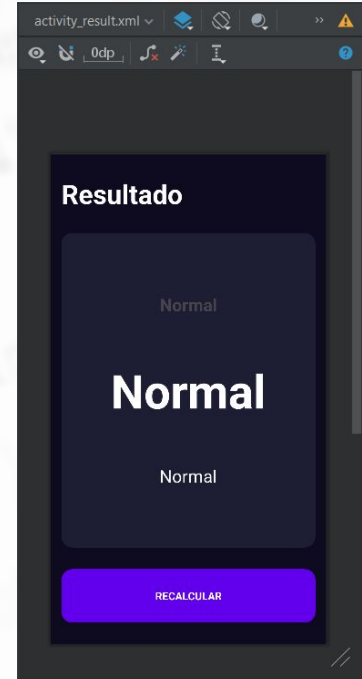
```
<TextView
    android:id="@+id/tvIMC"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginVertical="64dp"
    android:textColor="@color/white"
    android:textSize="70sp"
    android:textStyle="bold"
    tools:text="Normal" />
```



5. LAYOUT RESULT ACTIVITY

Y el tercero una pequeña descripción del intervalo en el que se encuentra el usuario según su IMC.

```
<TextView
    android:id="@+id/tvDescription"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/white"
    android:textSize="26sp"
    tools:text="Normal" />
```



6. LÓGICA RESULT ACTIVITY

En esta actividad debemos configurar la lógica de los 3 últimos TextViews creados para que muestren los resultados obtenidos del cálculo realizado en la primera actividad y del botón “Recalcular” para que nos devuelva a dicha actividad.

Crea las variables necesarias para interactuar con los distintos Views y las funciones initComponents() e initListeners() que hemos desarrollado en la primera actividad.

Incluye también la llamada a estas funciones desde la función onCreate().

6. LÓGICA RESULT ACTIVITY

Ahora, desde la primera actividad, crea la función `navigateToResult()` a la cual se le habrá de pasar un parámetro “result” con el resultado del cálculo realizado por `calculateIMC()`. Esta función iniciará la segunda actividad y le pasará dicho valor.

En la segunda actividad, crea la variable *result:Double* que almacene dicho dato, comprobando su nulabilidad y almacenando -1.0 si la variable es NULL.

Esto último se puede hacer con el operador Elvis `? : -1.0` en vez de con la función utilizada para strings en el tema anterior `.orEmpty()`.

6. LÓGICA RESULT ACTIVITY

Vamos a crear ahora una función para inicializar la interfaz de usuario, `initUI()`, a la que le pasaremos la variable *result* y mostrará por pantalla los resultados obtenidos.

```
private fun initUI(result: Double) {  
    tvIMC.text = result.toString()  
    when(result){  
        in 0.00..18.50 -> { //Bajo peso  
            tvResult.text = getString(R.string.title_bajo_peso)  
            tvResult.setTextColor(getColor(R.color.peso_bajo))  
            tvDescription.text = getString(R.string.description_bajo_peso)  
        }  
    }  
}
```

6. LÓGICA RESULT ACTIVITY

Antes habrá que definir los textos y colores que queremos que muestre nuestra aplicación para cada uno de los intervalos indicados por el NHLBI respecto al IMC. Añadiremos los colores:

```
<color name="peso_bajo">#FFEB3B</color>  
<color name="peso_normal">#4CAF50</color>  
<color name="sobrepeso">#FF5722</color>  
<color name="obesidad">#FF1100</color>
```

6. LÓGICA RESULT ACTIVITY

Y los textos:

```
<string name="error">Error</string>
<string name="title_bajo_peso">Bajo peso</string>
<string name="description_bajo_peso">Estás por debajo de lo óptimo para tu peso y
altura.</string>
<string name="title_peso_normal">Normal</string>
<string name="description_peso_normal">Estás en lo óptimo para tu peso y
altura.</string>
<string name="title_sobrepeso">Sobrepeso</string>
<string name="description_sobrepeso">Estás por encima de lo óptimo para tu peso y
altura.</string>
<string name="title_obesidad">Obesidad</string>
<string name="description_obesidad">Estás MUY por encima de lo óptimo para tu peso y
altura.</string>
```


6. LÓGICA RESULT ACTIVITY

Podemos entonces terminar de añadir los resultados para el resto de intervalos:

```
in 18.51..24.99 -> { //Peso normal
    tvResult.text = getString(R.string.title_peso_normal)
    tvResult.setTextColor(getColor(R.color.peso_normal))
    tvDescription.text = getString(R.string.description_peso_normal)
}
in 25.00..29.99 -> { //Sobrepeso
    tvResult.text = getString(R.string.title_sobrepeso)
    tvResult.setTextColor(getColor(R.color.sobrepeso))
    tvDescription.text = getString(R.string.description_sobrepeso)
}
```

6. LÓGICA RESULT ACTIVITY

Y en el caso de que el cálculo falle, mostramos en todos los textos un error en color rojo...

```
in 30.00..99.00 -> { //Obesidad
    tvResult.text = getString(R.string.title_obesidad)
    tvResult.setTextColor(getColor(R.color.obesidad))
    tvDescription.text = getString(R.string.description_obesidad)
}
else -> { //Error
    tvIMC.text = getString(R.string.error)
    tvResult.text = getString(R.string.error)
    tvResult.setTextColor(getColor(R.color.obesidad))
    tvDescription.text = getString(R.string.error)
}
```

6. LÓGICA RESULT ACTIVITY

Por último, vamos a ver cómo volver a la primera actividad al pulsar el botón “Recalcular” sin necesidad de crear un intent. Dentro de la función `initListeners()` utilizaremos el comando `onBackPressed()`.

```
private fun initListeners() {  
    btnRecalculate.setOnClickListener { onBackPressed() }  
}
```

Y con esto ya tenemos nuestra aplicación terminada.