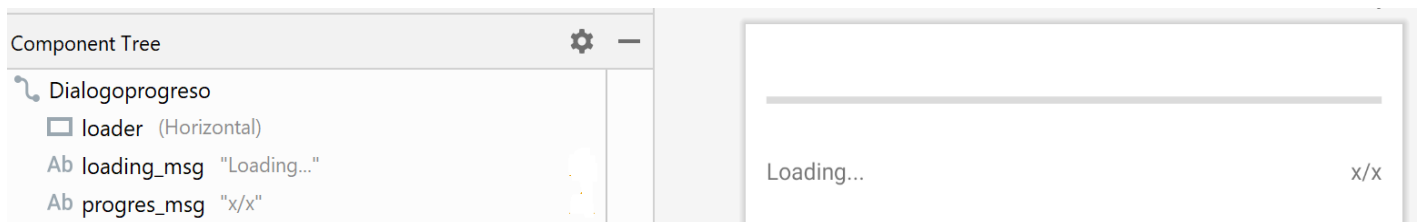


Bloque 10 . Ejercicio Resuelto Barra Progreso

[Descargar este ejercicio](#)

Vamos a desarrollar una aplicación que simule la carga de un conjunto de archivos, simularemos la carga mediante **CorrutinA** donde especificaremos un retardo por cada paso de la tarea, y utilizaremos una **LinearProgressIndicator** para ver la evolución del estado de la carga.

El diseño de nuestro **LinearProgressIndicator** va a ser personalizado, incluyendo unos campos de texto tal y como se ve en la siguiente imagen:



El archivo xml correspondiente a esta interfaz sería:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:padding="13dp"
7      android:id="@+id/Dialogoprogreso"
8      android:layout_centerHorizontal="true"
9      android:layout_width="match_parent"
10     android:layout_height="wrap_content">
11
12     <com.google.android.material.progressindicator.LinearProgressIndicator
13         android:id="@+id/loader"
14         style="@style/Widget.MaterialComponents.LinearProgressIndicator"
15         android:layout_width="match_parent"
16         android:layout_height="65dp"
17         app:layout_constraintStart_toStartOf="parent"
18         app:layout_constraintTop_toTopOf="parent"/>
19
20     <TextView
21         android:id="@+id/loading_msg"
22         android:layout_width="wrap_content"
23         android:layout_height="wrap_content"
24         android:layout_alignParentLeft="true"
25         android:text="Loading..."
26         android:textAppearance="?android:textAppearanceSmall"
27         app:layout_constraintStart_toStartOf="parent"
28         app:layout_constraintTop_toBottomOf="@id/loader"/>
29
30     <TextView
31         android:id="@+id/progres_msg"
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content"
34         android:text="x/x"
35         android:textAppearance="?android:textAppearanceSmall"
36         app:layout_constraintEnd_toEndOf="parent"
37         app:layout_constraintTop_toBottomOf="@+id/loader" />
38 </androidx.constraintlayout.widget.ConstraintLayout>

```

Aclaraciones:

- **Líneas 12-18:** definición del **LinearProgressIndicator** . En la línea 16 especificamos el texto que se visualizará cuando el **FAB** esté pulsado. El resto de propiedades ya están explicadas.
- **Líneas 20-28:** aquí definimos el primer **Text** que indicará lo que hace nuestra **LinearProgressIndicator**
- **Líneas 30-37:** aquí definimos el texto asociado al progreso de la acción (1/10, 2/10...)

Todo el proceso comenzará cuando pulsemos el botón de **Load** en la pantalla principal. Añadiremos un botón para iniciar el funcionamiento de la barra de progreso.

main_activity.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <Button
11         android:id="@+id/buttonProgress"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="Load"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintLeft_toLeftOf="parent"
17         app:layout_constraintRight_toRightOf="parent"
18         app:layout_constraintTop_toTopOf="parent" />
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

Deberemos inflar el dialogo que contiene la barra de progreso y mostrarlo cuando pulsemos el botón. Para ello se puede crear un método que se encargue de este trabajo y que puede quedar de la siguiente manera:

```
1 private fun setDialog() {
2     val builder = MaterialAlertDialogBuilder(this)
3     val inflater = this@MainActivity.layoutInflater
4     val v = inflater.inflate(R.layout.dialogo_progress, null)
5     progressBar = v.findViewById(R.id.loader)
6     loadingMessage = v.findViewById(R.id.loading_msg)
7     progressMessage = v.findViewById(R.id.progres_msg)
8     builder.setView(v)
9     dialog = builder.create()
10    dialog?.setCancelable(false)
11    dialog?.show()
12 }
```

📌 como se puede ver en este código, tenemos tres variables que serán infladas dentro del método pero son propiedades de la clase, ya que tendrán que ser accedidas desde fuera de este para modificar el aspecto de la progressBar. Lo mismo ocurre con el objeto dialog.

El código principal que permite crear la tarea en segundo plano lo podemos realizar de la siguiente manera:

```
1 fun barraProgreso()
2 {
3     setDialog()
4     job = CoroutineScope(Dispatchers.Default).launch {
5         val inicio = progreso
6         for (i in inicio..100) {
7             Thread.sleep(100)
8             withContext(Dispatchers.Main) {
9                 progressBar!!.progress = i
10                progressMessage.text = i.toString() + "/100"
11                progreso = i
12            }
13        }
14        progreso = 0
15        dialog?.dismiss()
16        dialog = null
17    }
18 }
```

📌 lanzamos una corrutina mediante **launch** con un alcance definido en **CoroutineScope** fuera del hilo principal y para trabajos costosos en el tiempo **Dispatchers.Default** **Línea 4**, (el trabajo costoso se supone que es el retardo que producimos con el bucle y el sleep **Líneas 6**

y 7). Para poder cambiar las vistas relacionadas con el hilo principal, lanzamos la función de suspensión `withContext` con `Dispatchers.Main` **Líneas 8-12**.

Las **líneas de 14 a 16** se encargan de reiniciar los elementos que se han usado. El dialogo lo cerramos e iniciamos la variable a null, para evitar problemas si se produce un giro. Usamos la propiedad progreso para guardar el progreso de la barra y poder recuperarlo posteriormente al reiniciar después de un giro **Línea 5 y 14**.

Si queremos controlar los giros de pantalla, y que la barra de progreso siga funcionando por el lugar en que se había quedado al producirse el giro, tendremos que guardar el estado de esta. Eso lo hacemos con la propiedad de tipo entero `progreso` que hemos comentando anteriormente. Para guardar y recuperar información de una actividad podemos utilizar los método

`onSaveInstanceState` y `onRestoreInstanceState` estos métodos serán llamados automáticamente al destruir la aplicación y al volverse a crear (siempre y cuando no sea por un cierre realizado de forma correcta por el usuario).

```
1  override fun onSaveInstanceState(outState: Bundle)
2  {
3      super.onSaveInstanceState(outState)
4      outState.putInt("PROGRESO", progreso)
5      if (dialog != null) {
6          dialog?.dismiss()
7          dialog = null
8      }
9  }
10 override fun onRestoreInstanceState(savedInstanceState: Bundle)
11 {
12     super.onRestoreInstanceState(savedInstanceState)
13     progreso = savedInstanceState.getInt("PROGRESO")
14 }
```

📌 método `onSaveInstanceState` utilizamos el bundle que tiene como parámetro para guardar el progreso y anulamos el dialogo siempre y cuando esté creado. Método

`onRestoreInstanceState` recuperamos la información del progreso que nos llegará mediante el bundle de parámetro. Éstos métodos guardan la información en la memoria no volátil mediante serialización, por lo que el proceso es un poco más lento que si se usara ViewModel y solo se debe utilizar cuando se quiere guardar información simple.

Para acabar queda el código de la **MainActivity**, donde llamaremos al método barraProgreso directamente si hay una posición guardada de antemano (significa que se ha recuperado el estado después de un reinicio), o habrá que pulsar en el botón para que se realice la llamada y se lance la tarea.

```

1  class MainActivity : AppCompatActivity() {
2      private lateinit var binding: ActivityMainBinding
3      var dialog: AlertDialog?=null
4      var progressBar: ProgressBar? = null
5      var progreso = 0
6      lateinit var loadingMessage: TextView
7      lateinit var progressMessage: TextView
8
9
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         if(savedInstanceState!=null) progreso=savedInstanceState.getInt("PROGRESO")
13         binding = ActivityMainBinding.inflate(layoutInflater)
14         val view = binding.root
15         setContentView(view)
16         if(progreso>0) barraProgreso()
17         binding.buttonProgress.setOnClickListener {barraProgreso()}
18     }
19     ...
20 }

```