Bloque 10 . Ejercicio Resuelto Barra Progreso

Descargar este ejercicio

Vamos a partir del ejercicio resuelto **Barra de Progreso** del **Bloque 6**. Como se explica en el ejercicio anterior, la manera de resolverlo con AsynTask ha sido deprecada en versiones recientes, por lo que ahora aplicaremos los conocimientos de `corrutinas` para resolverlo. Además tendremos en cuenta los reinicios de la aplicación por giros y demás.

main_activity.xml

Añadiremos un botón para iniciar el funcionamiento de la barra de progreso.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
    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">

    <Button
        android:id="@+id/buttonProgress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Load"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Además se deberá de tener un `layout` para la vista del dialogo personalizado. Podría ser un archivo como el siguiente.

dialogo_progress.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:padding="13dp"
    android:id="@+id/Dialogoprogreso"
    android:layout_centerHorizontal="true"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <com.google.android.material.progressindicator.LinearProgressIndicator
        android:id="@+id/loader"
        style="@style/Widget.MaterialComponents.LinearProgressIndicator"
        android:layout_width="match_parent"
        android:layout_height="65dp"
        android:max="100"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/loading_msg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:text="Loading..."
        android:textAppearance="?android:textAppearanceSmall"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/loader"/>

    <TextView
        android:id="@+id/progres_msg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="x/x"
        android:textAppearance="?android:textAppearanceSmall"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/loader" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Como ya hemos hecho en el Bloque 6, 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:

```
private fun setDialog() {
        val builder = MaterialAlertDialogBuilder(this)
        val inflater = this@MainActivity.layoutInflater
        val v = inflater.inflate(R.layout.dialogo_progress, null)
        progressBar = v.findViewById(R.id.loader)
        loadingMessage = v.findViewById(R.id.loading_msg)
        progressMessage = v.findViewById(R.id.progres_msg)
        builder.setView(v)
        dialog = builder.create()
        dialog?.setCancelable(false)
        dialog?.show()
    }
```

📌 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:

```
    fun barraProgreso()
    {
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) {
                        progressBar!!.progress = i
                        progressMessage.text = i.toString() + "/100"
                        progreso = i
12              }
            }
14          progreso = 0
            dialog?.dismiss()
16          dialog = null
        }
    }
```

📌 lanzamos una corrutina mediante `launch` con un alcance definido en CoroutineScope fuera del hilo principal y para trabajos costosos en el tiempo `Dispachers.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 `Dispachers.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).

```
override fun onSaveInstanceState(outState: Bundle)
{
    super.onSaveInstanceState(outState)
    outState.putInt("PROGRESO", progreso)
    if (dialog != null) {
            dialog?.dismiss()
            dialog = null
    }
}
override fun onRestoreInstanceState(savedInstanceState: Bundle)
{
    super.onRestoreInstanceState(savedInstanceState)
    progreso = savedInstanceState.getInt("PROGRESO")
}
```

> 📌 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.

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    var dialog: AlertDialog?=null
    var progressBar: ProgressBar? = null
    var progreso = 0
    lateinit var loadingMessage: TextView
    lateinit var progressMessage: TextView


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        if(savedInstanceState!=null) progreso=savedInstanceState.getInt("PROGRESO")
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
        if(progreso>0)  barraProgreso()
        binding.buttonProgress.setOnClickListener {barraProgreso()}
    }
    ...
}
```