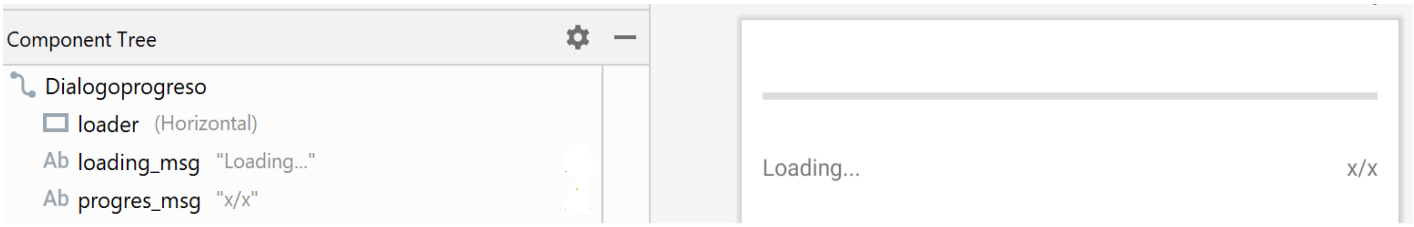
Ejercicio resuelto Progress indicators

Descargar estos apuntes

Vamos a desarrollar una aplicaión que simule la carga de un conjunto de archivos, simularemos la carga mediante un retardo y utilizaremos una `LinearProgressIndicator` para ver la evolución del estado de la carga. Todo el proceso comenzará cuando pulsemos el botón de `Load` en la pantalla principal.

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:

```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"
        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>
```

📌 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 indicaará lo que hace nuestra `LinearProgressIndicator`
- **Líneas 30-37**: aquí definimos el texto asociado al progreso de la acción (1/10, 2/10...)

Solamente nos falta analizar el código de nuestra aplicación, donde iremos visualizando los elementos de la vista y las acciones de cada uno de los botones.

```
class MainActivity : AppCompatActivity() {
    var isAllFabsVisible: Boolean? = null
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
        binding.imageFab.visibility= View.GONE
        binding.photocameraFab.visibility= View.GONE
        binding.videocameraFab.visibility= View.GONE
        binding.imageText.visibility= View.GONE
        binding.photocameraText.visibility= View.GONE
        binding.videocameraText.visibility= View.GONE
        isAllFabsVisible = false

        binding.addFab.shrink()

        binding.addFab.setOnClickListener(){
            if (isAllFabsVisible==false){
                isAllFabsVisible=true
                binding.imageFab.show()
                binding.photocameraFab.show()
                binding.videocameraFab.show()
                binding.imageText.visibility= View.VISIBLE
                binding.photocameraText.visibility= View.VISIBLE
                binding.videocameraText.visibility= View.VISIBLE
                binding.addFab.extend()
            }
            else{
                isAllFabsVisible=false
                binding.imageFab.hide()
                binding.photocameraFab.hide()
                binding.videocameraFab.hide()
                binding.imageText.visibility= View.GONE
                binding.photocameraText.visibility= View.GONE
                binding.videocameraText.visibility= View.GONE
                binding.addFab.shrink()
            }
        }

        binding.imageFab.setOnClickListener(){
            Toast.makeText(applicationContext,"Pulsaste el botón IMAGEN",Toast.LENGTH_LON
        }

        binding.photocameraFab.setOnClickListener(){
            Toast.makeText(applicationContext,"Pulsaste el botón CAMARA",Toast.LENGTH_LON
        }

        binding.videocameraFab.setOnClickListener(){
```

```
                Toast.makeText(applicationContext,"Pulsaste el botón VIDEO",Toast.LENGTH_LONG
            }
        }
    }
```

📌 Aclaraciones:

- **Línea 2**: utilizamos la variable `isAllFabsVisible` para saber si está extendido o mno el `FAB` principal, en principio (línea 16) se inicia a `false` indicando que está el menú contraído.
- **Líneas 10-15**: las vistas asociadas al menú de acciones están no visibles, para ello utilizamos el atributo `visibility` de cada una de ellas y le asociamos el valor `View.GONE` que hace que la vista sea invisible sin que la vista ocupe espacio en el diseño. `View.INVISIBLE` hace que la vista sea invisible pero ocupando espacio.
- **Líneas 18** con el método `shrink()` se desactiva el mode extendiso del `FAB` ocultando el texto que tiene asociado cuando se expanda.
- **Líneas 20-41**: aquí gestionamos el `click` sobre el botón principal. Si el menú de acciones estaba invisible, entonces hacemos visibles las vistas con `View.VISIBLE` y hacemos que aparezca el texto asociado al `FAB` inicial con el método `extend()` . Por el contrario si el `FAB` principal estaba pulsado, aplicamos la lógica inversa a lo explicado anteriormente. El método `hide()` oculta la vista con animación. Similar a `show()` que la muestra con animación.
- **Líneas 43-45** y siguientes, donde se implementa el `click` de cada una de las acciones del menú. Simplemente visualizamos un `Toast` indicando la acción pulsada.