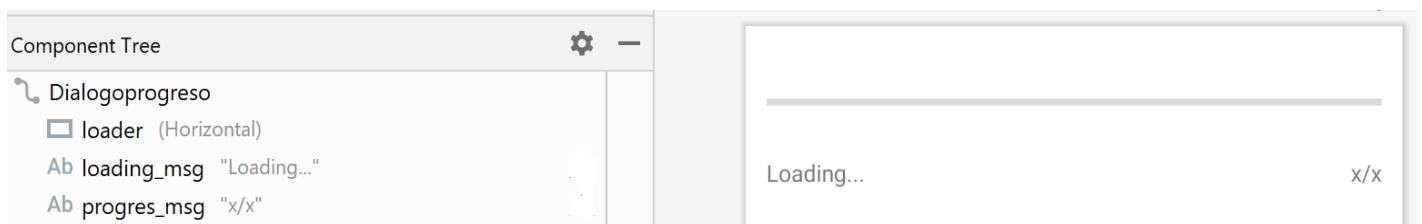


# Ejercicio resuelto Progress indicators

[Descargar estos apuntes](#)

Vamos a desarrollar una aplicación que simule la carga de un conjunto de archivos, simularemos la carga mediante **AsyncTask** donde especificaremos un retardo por cada paso de la tarea, 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 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">

    12 <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"
    18 app:layout_constraintTop_toTopOf="parent"/>

    20 <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"
    28 app:layout_constraintTop_toBottomOf="@id/loader"/>

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

## Tareas asíncronas

Un hilo es una unidad de ejecución asociada a una aplicación. Es la estructura de programación concurrente que tiene como objetivo dar la percepción al usuario que el sistema ejecuta múltiples tareas a la vez.

Desde **Android** podemos ejecutar tareas en segundo plano utilizando **thread** (hilos), con **AsyncTask** y con corrutinas **Kotlin**.

El problema con los hilos es que no podemos acceder a ningún elemento de la interfaz gráfica desde dentro del mismo.

La finalidad de **AsyncTask** es unificar la actualización de la vista asociada a la tarea en segundo plano con la ejecución de la misma. Está marcada como **deprecated** desde la versión 11 de **Android (API 30)**. A partir de esta versión se recomienda el uso de corrutinas.

**AsyncTask** es una interfaz que nos va a permitir crear un hilo secundario en el que realizar tareas en background.

La definición genérica de una tarea asíncrona sigue el siguiente esquema:

```
class MiTarea:AsyncTask<T1,T2,T3>(){
    override fun onPreExecute(){
        //...
    }

    override fun doInBackground(vararg params:T1):T3{
        //...
        return T3
    }

    override fun onProgressUpdate(vararg params:T2){
        //...
    }

    override fun onPostExecute(result:T3){
        //...
    }

    override fun onCancelled(){
        //...
    }
}
```

El método encargado de ejecutar el hilo secundario es **doInBackground()**, el resto se ejecutan en el hilo principal.

El método `onPreExecuted()` se ejecuta antes de empezar la tarea en segundo plano.

En la definición se especifican tres tipos de datos:


- El primero es el que recibe el método `doInBackground()`. La notación `vararg params:T1` indica que se pueden recibir un número indeterminado de parámetros indeterminado del tipo `T1`.
- El segundo es el que permite comunicar el avance de la tarea. La comunicación con el hilo principal se realizará a través de la invocación del método `publishProgress()` que hará que se ejecute el método `onProgressUpdate()` que se encargará de la actualización de la interfaz.
- El tercero es el tipo de datos que se devolverá al finalizar el hilo secundario `doInBackground()` y que será recibido en `onPostExecute()`.

El método `onCancelled()` se ejecuta si la tarea es cancelada, y hace que no se ejecute el `onPostExecute()`.

Vamos a integrar una tarea asíncrona para simular la descarga de archivos en nuestro ejemplo.

#### Aclaraciones:

- **Línea 2:** utilizamos la variable `isAllFabsVisible` para saber si está extendido o no el **FAB** principal, en principio (línea 16) se inicia a `false` indicando que está el menú contraído.

 **Ejercicio propuesto:** Implementar este ejemplo con corrutinas `Kotlin`. Es un trabajo de investigación y desarrollo.