**Ejemplo de Tareas Asíncronas:**

1. Vamos a empezar por crear una aplicación de ejemplo en cuya actividad colocaremos un control ProgressBar (en el ejemplo lo llamamos miprogressbar) y un botón (start) que ejecute una tarea de larga duración.

   **XML:**

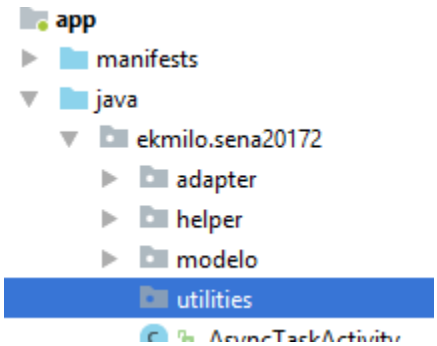
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="ekmilo.sena20172.AsyncTaskActivity">

    <ProgressBar
        android:id="@+id/miprogressbar"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:text="iniciar"
        android:id="@+id/start"
        android:layout_gravity="center_horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

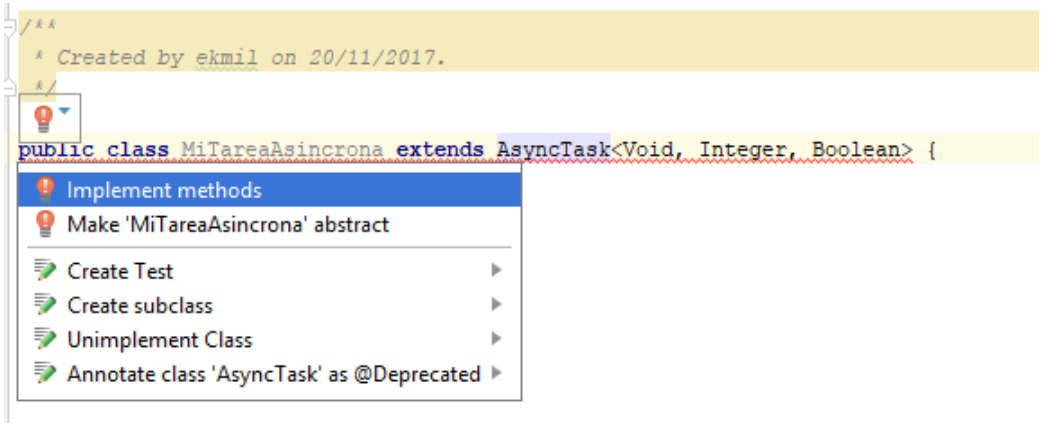
   **Actividad:**

```java
public class AsyncTaskActivity extends AppCompatActivity {

    private Button start;
    private ProgressBar miprogressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_async_task);
        start = findViewById(R.id.start);
        miprogressBar=findViewById(R.id.miprogressbar);
    }
}
```

2. Creamos la clase MiTareaAsincrona en el paquete utilities, extendiendo a AsyncTask, fíjese en los tres parámetros que se indican cuando se hereda de la clase en este ejemplo son Void, Integer y Boolean, esto lo que nos indica es que **doInBackground**() no recibirá ningún parámetro de entrada (Void),

**publishProgress**() y **onProgressUpdate**() recibirán como parámetros datos de tipo entero (Integer) y **doInBackground**() devolverá como retorno un dato de tipo booleano y **onPostExecute**() también recibirá como parámetro un dato del dicho tipo (Boolean).




**3.** Para simular una operación de larga duración vamos a crear un método auxiliar que lo único que haga sea esperar 1 segundo, mediante una llamada a Thread.sleep().

```
private void tareaLarga()
{
    try {
        Thread.sleep( millis: 1000);
    } catch(InterruptedException e) {}
}
```

```
public class MiTareaAsincrona extends AsyncTask<Void, Integer, Boolean> {

    private void tareaLarga()
    {
        try {
            Thread.sleep( millis: 1000);
        } catch(InterruptedException e) {}
    }

    @Override
    protected Boolean doInBackground(Void... voids) {
        return null;
    }
}
```

4. Creamos un constructor que reciba los elementos necesarios de la actividad para usarlos en los métodos de la tarea asíncrona

```
private ProgressBar miprogressbar;
private Context context;

public MiTareaAsincrona(ProgressBar miprogressbar, Context context){
    this.miprogressbar=miprogressbar;
    this.context=context;
}
```

5. Agregamos la lógica del método **doInBackground**, dentro de la ejecución de nuestra tarea en doInBackground() tendremos además que consultar periódicamente el resultado del método isCancelled() que nos dirá si el usuario ha cancelado la tarea (es decir, si se ha llamado al método cancel()), en cuyo caso deberemos de terminar la ejecución lo antes posible, en nuestro caso de ejemplo simplemente saldremos del bucle con la instrucción break.

```
@Override
protected Boolean doInBackground(Void... voids) {
    for(int i=1; i<=10; i++) {
        tareaLarga();

        publishProgress( ...values: i*10);

        if(isCancelled())
            break;
    }

    return true;
}
```

6. Agregamos la lógica del método onProgressUpdate

```
@Override
protected void onProgressUpdate(Integer... values) {
    int progreso = values[0].intValue();
    miprogressbar.setProgress(progreso);
}
```

7. Agregamos la lógica del método onPreExecute

```
@Override
protected void onPreExecute() {
    miprogressbar.setMax(100);
    miprogressbar.setProgress(0);
}
```

8. Agregamos la lógica del método onPostExecute

```
@Override
protected void onPostExecute(Boolean result) {
    if(result)
        Toast.makeText(context, text: "Tarea finalizada!",Toast.LENGTH_SHORT).show();
}
```

9. Agregamos la lógica del método onCancelled, en los casos que se cancela la tarea, tras el método doInBackground() no se llamará a onPostExecute() sino al método onCancelled(), dentro del cual podremos realizar cualquier acción para confirma la cancelación de la tarea. En nuestro caso mostraremos un mensaje Toast informando de ello.

```
@Override
protected void onCancelled() {
    Toast.makeText(context, text: "Tarea cancelada!", Toast.LENGTH_SHORT).show();
}
```

10. Finalmente en la actividad, creamos el Listener en el botón de nuestra actividad y creamos e iniciamos la tarea asíncrona

```
start.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        MiTareaAsincrona tarea = new MiTareaAsincrona(miprogressBar, context: AsyncTaskActivity.this);
        tarea.execute();
    }
});
```

**Generando**: