AsyncTask

Las **AsyncTasks** le permiten ejecutar fácilmente un tratamiento en tareas de segundo plano y recuperar el resultado en la interfaz del usuario sin bloquear el UI Thread.

Para utilizar las **AsyncTasks** hay que crear una nueva clase que extienda la clase **AsyncTask**.

```
public class myDownloadTask extends AsyncTask<Params, Progress,
Result> {
```

La clase **AsyncTask** se parametriza con tres tipos de datos:

- El tipo de dato que se pasa como parámetro a la clase, en concreto al método **doInBackground**.
- El tipo de datos utilizado para publicar el avance de la tarea en ejecución. Se utiliza en el método **onProgressUpdate** (en una barra de progreso horizontal, por ejemplo).
- El tipo de datos utilizado para publicar el resultado a la interfaz, se transmitirá al método **onPostExecute** a través del método **doInBackground**.

La clase **AsyncTask** le permite sobrecargar los siguientes métodos:

- **onPreExecute**: este método le permite actualizar la interfaz de su aplicación antes de empezar a ejecutar la tarea en segundo plano. Este método se ejecuta en el UI Thread.
- **doInBackground**: este método se ejecuta en un thread separado, lo que le permite ejecutar un tratamiento pesado en una tarea de segundo plano.
- **onProgressUpdate**: este método le permite actualizar el progreso de la tarea en ejecución. Se invoca gracias a la función**publishProgress**.
- **onPostExecute**: este método permite actualizar la interfaz con el resultado obtenido al final del tratamiento ejecutado en el método **doInBackground**.

Para comprender mejor este componente, cree un proyecto Android que tenga un botón. Un clic en este botón provocará la simulación de un tratamiento en una **AsyncTask**.

El progreso del tratamiento se indica mediante una barra de progreso horizontal.

La interfaz se compondrá de dos elementos:

- Un botón.
- Una barra de progreso: tiene un estilo horizontal y está oculta por defecto (atributo **visibility**), el objetivo es mostrarla durante el tratamiento.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">

      <Button android:text="@string/launch_async"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:id="@+id/launch_async"
          android:layout_gravity="center_horizontal"
          android:layout_margin="10sp"/>

      <ProgressBar
style="@android:style/Widget.ProgressBar.Horizontal"
        android:id="@+id/progress"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:visibility="gone"
        android:layout_margin="10sp"/>
```

```
</FrameLayout>
```

Ahora, cree una clase que herede de la clase **AsyncTask**.

```
public class myDownloadTask extends AsyncTask<String, Integer,
String> {
}
```

Puede utilizar **Void** como tipo de datos si no desea utilizar datos en el parámetro (en alguno de los tres tipos definidos durante la creación de una **AsyncTask**).

Seguidamente, implemente el método **onPreExecute** para ocultar el botón y mostrar la barra de progreso.

```
@Override
protected void onPreExecute() {
  super.onPreExecute();
  launchAsync.setVisibility(View.GONE);
  progress.setVisibility(View.VISIBLE);
}
```

A continuación, implemente el método **doInBackground**:

```
@Override
protected String doInBackground(String... params) {
    String uri = params[0];
    String result = "";
    for (int i = 1; i <= 10; ++i) {
       try {
         Thread.sleep(1000L);
      } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        publishProgress(i * 10);
        result += i;
    }
    return result;
}
```

- Este método recibe como parámetro una tabla sin acotar de cadenas de caracteres (tipo elegido en la sobrecarga de la clase**AsyncTask**).
- Se realiza la simulación de un tratamiento pesado mediante un bucle que recorre del 1 al 10. Con cada iteración se detiene el thread durante 1 segundo y, a continuación, se publica el grado de avance del tratamiento (se aumenta de 10 en 10 en cada iteración).
- Para finalizar, hay que devolver el resultado al método **onPostExecute**.

La llamada al método **publishProgress** provoca la ejecución del método **onProgressUpdate**.

```
@Override
protected void onProgressUpdate(Integer... progress) {
    super.onProgressUpdate(progress);
progress.setProgress(progress[0]);
}
```

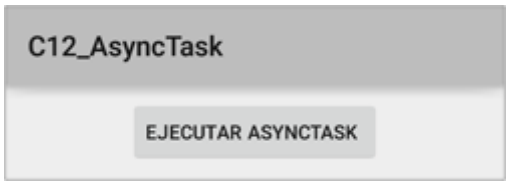
Este método permite definir el grado de progreso actualizando la barra de progreso.

Ahora, implemente el método **onPostExecute** que, en este ejemplo, sirve para mostrar un Toast indicando la finalización de la tarea en segundo plano así como para ocultar la barra de progreso y mostrar el botón.

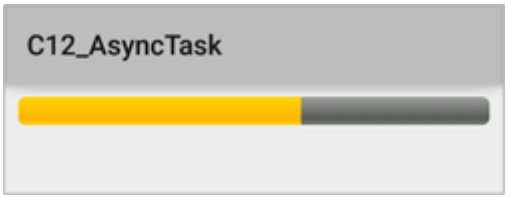
```
@Override
```

```
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    Toast.makeText(AsyncTaskActivity.this, "Finalización
del tratamiento en segundo plano", Toast.LENGTH_LONG).show();
    launchAsync.setVisibility(View.VISIBLE);
    progress.setVisibility(View.GONE);
}
```

Con lo que se obtendrá:



Antes de la ejecución del tratamiento



Durante la ejecución del tratamiento

Después de la ejecución del tratamiento

En lo concerniente a las **AsyncTasks**, debe recordar los siguientes puntos:

- Sólo el método **doInBackground** es el que no se ejecuta en el UI Thread.
- Las **AsyncTasks** no persisten si se mata la actividad.
- Su uso es ideal para tratamientos cortos.
- Una **AsyncTask** no puede ejecutarse hasta que la ejecución anterior finalice.
- El tipo **Void** evita el uso de uno de los tipos de datos definidos en la sobrecarga de una **AsyncTask**.
- La ejecución de la **AsyncTask** se realiza mediante el método **execute**:

```
downloadTask = new myDownloadTask();

downloadTask.execute(uri);
```