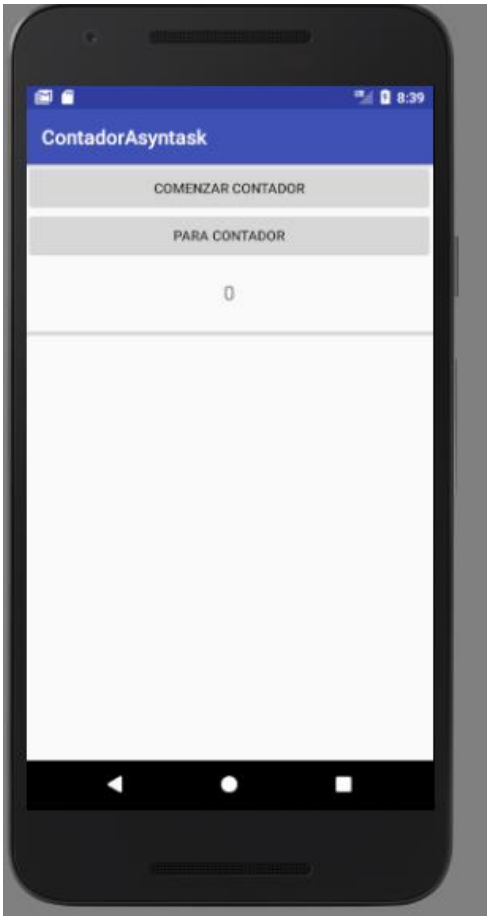
Contador con AsyncTask

Para este proyecto vamos a realizar un contado creando hilos para su ejecución. Para ello utilizamos la clase AsyncTask que es la clase que proporciona Android para la realización de hilos.

Lo primero que haremos será crear la vista que tendrá la aplicación. Una interfaz sencilla con dos botones (para comenzar el hilo y para parar el hilo), un textView para mostrar el proceso del contador y una barra de progreso.



Una vez realizada la vista pasamos al código. Tendremos una clase principal (MainActivity.class) y dentro de ella una clase interna que será Contador que heredará se AsyncTask. Cargaremos los componentes de la vista en código java.

```java
public class MainActivity extends AppCompatActivity {

    private TextView cont;
    private Button start, stop;
    private ProgressBar bar;
    private Contador contador = new Contador();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Cargamos los recursos
        cont = (TextView) findViewById(R.id.textView_cont);
        start = (Button) findViewById(R.id.button_comenzar);
        stop = (Button) findViewById(R.id.button_parar);
        bar = (ProgressBar) findViewById(R.id.progressBar);

        //Configuramos la barra de progreso, que tendrá un máximo de 10
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            bar.setMin(0);
        }
        bar.setMax(10);
    }
```

Cada botón responderá a un método que le hemos asignado en la propiedad onClick. Como es lógico el método comenzarCont, comenzará el contador si no existe ningún hilo en ejecución, en caso de que haya algún hilo en ejecución no inicia uno nuevo. El método cancelarCont, parará el hilo si existe uno en ejecución. Utilizamos getStatus() para comprobar en qué fase se encuentra el hilo. Aquí vemos el código.

```java
/**
 * Método que hace comenzar el contador. Si ya hay un proceso activo no comenzará uno nuevo
 * @param v El botón de la vista "comenzar contador"
 */
public void comenzarCont(View v) {
    if(!contador.getStatus().equals(AsyncTask.Status.RUNNING)) { //Si hay algún contador activo no inicia el hilo
        contador = new Contador();
        contador.execute();
    }
}

/**
 * Método que cancela el proceso. Si no hay ningún proceso activo no hace nada
 * @param v
 */
public void cancelarCont(View v){
    //Si contador está ejecitandose podremos para el hilo, si no, no podemos parar el contador ya que no
    // hay ningún proceso de contador activo
    if(contador.getStatus().equals(AsyncTask.Status.RUNNING)) {
        contador.cancel( mayInterruptIfRunning: true);
        Toast.makeText( context: this, text: "Contador parado", Toast.LENGTH_LONG).show();
    }
}
```

Ahora pasaremos a la clase Contador. Extiende de la clase AsyncTack y no tendremos que pasarle parámetros por lo que indicamos Void en la primera posición. Durante el proceso necesitaremos un entero para actualizar la vista del contador y la barra de progreso por lo que indicamos Integer en la segunda posición. Por último, el resultado que nos devuelve el proceso realizado, que en este caso no va a devolver nada por lo que indicamos Void.

La clase AsyncTask es abstracta y contiene 7 métodos:

- **doInBackground(Params... params)** – Donde se ejecuta el hilo.
- **onCancelled()** – Método para cuando se cancela el hilo.
- **onCancelled(Result result)** – Cuando el hilo se cancela pero el hilo principal devuelve un resultado
- **onPostExecute(Result result)** – Una vez terminado el hilo, se ejecuta procesando los datos que vuelva.
- **onPreExecute()** – Instrucciones que ejecuta antes de empezar el proceso.
- **onProgressUpdate(Progress... values)** – Este método se carga en el hilo principal. Ideal para actualizar la interfaz de usuario mientras el hilo secundario se ejecuta.
- **publishProgress(Progress... values)** – Es el método llama a onProgressUpdate y ejecutará lo que contenga este método.

Para esta práctica he utilizado 3 métodos de la clase: doInBackground() (este método es abstracto, siempre se ha de sobrescribir), onPreExecute() y onProgressUpdate()

En onPreExecute() ponemos el textView del contador y la barra de progreso a 0.

```
class Contador extends AsyncTask<Void, Integer, Void> {

    @Override
    protected void onPreExecute() {
        //Antes de la ejecución del proceso. Inicializamos el contador y la barra de proceso a 0
        cont.setText("0");
        bar.setProgress(0);
    }
```

Comienza el hilo y tendremos una iteración de un for. En él incrementamos hasta 10 o cuando se cancele el hilo (con el método isCancelled() podremos saber si ha sido cancelado). Dentro del for llamamos al método publishProgress() que le pasamos la variable del bucle y paramos el hilo durante un segundo.

```
@Override
protected Void doInBackground(Void... n) {
    //Hacemos las iteraciones del for hasta 10 o se cancele el proceso
    for (int i = 0; i <= 10 && !isCancelled(); i++) {
        //Actualizamos la barra de proceso. Este método le envia la información al método onProgressUpdate
        publishProgress(i);
        //Dejamos el proceso dormido 1 segundo
        SystemClock.sleep( ms: 1000);
    }
    return null;
}
```

Y llegamos a la actualización de las vistas. Este método se ejecuta sobre el hilo principal por lo que no tendremos que llamar a métodos que hagan que las vistas se actualicen sobre la principal. Recibimos un Integer, que tendrá solo una posición.

```
@Override
protected void onProgressUpdate(final Integer... porc) {
    //Aquí se actualizará la barra de poceso y la vista que muestra los números del contador
    bar.setProgress(porc[0]);
    cont.setText(porc[0].toString());

}
```

No ha sido necesaria en este caso la implantación del onPostExecute ni de onCancelled() ya que no vamos a realizar nada cuando se termine de ejecutar ni cuando se cancele. Si se hiciese la comprobación del estado del hilo mediante un booleano, si sería conveniente implementarlos para cambiar los estados del booleano, pero el utilizar el método getStatus() de la clase contador no es necesario