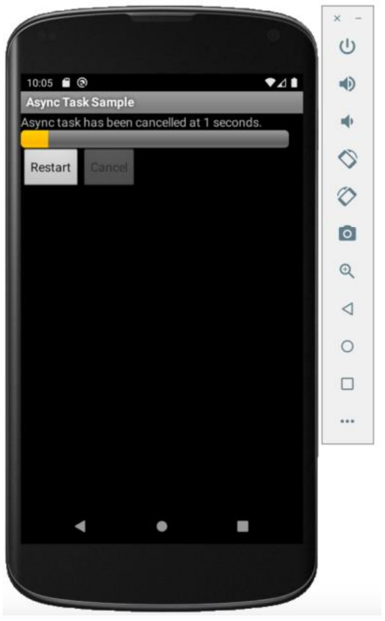
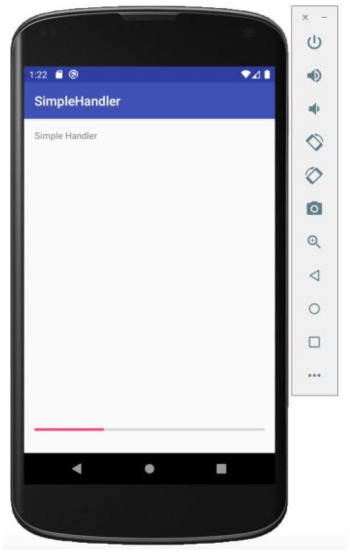
```java
package com.asynctask;

import java.text.MessageFormat;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends Activity {
    private Button btnRestart;
      private Button btnCancel = null;
      private TextView txtMessage =  null;
      private ProgressBar mProgressBar =  null;
      private HugeWork task = null;
    private static final int MAX_PROGRESS = 10;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

            btnRestart = (Button) findViewById(R.id.btnRestart);
            btnCancel = (Button) findViewById(R.id.btnCancel);
            txtMessage = (TextView) findViewById(R.id.txtMessage);
            mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

            // set an arbitrary max value for the progress bar
            mProgressBar.setMax(MAX_PROGRESS);
            // start the async task
            start();
    }

    // Cancel the async task and handle buttons enablement. Note that
    // the Cancel button is disabled because the task is finished and the
    // restart button is enabled so one can execute the process again.
    //
    // this is the listener for the Cancel Button.
    public void cancelOnclick(View v) {
      task.cancel(true);
      btnCancel.setEnabled(false);
      btnRestart.setEnabled(true);
    }

    // Restart the process execution. This is the listener to the Restart button.
    public void restartOnclick(View v) {
      start();
    }

    // Here we start the big task. For that, we reset the progress bar, set the
    // cancel button to be enable so one can stop the operation at any time and
    // finally we disable the restart button because the task is on-going.
    private void start() {
      // instantiate a new async task
      task = new HugeWork();
      // start async task setting the progress to zero
            task.execute(0);
            // reset progress
            mProgressBar.setProgress(0);
            // handle buttons
            btnCancel.setEnabled(true);
            btnRestart.setEnabled(false);
    }
```



Async task has been cancelled at 1 seconds.

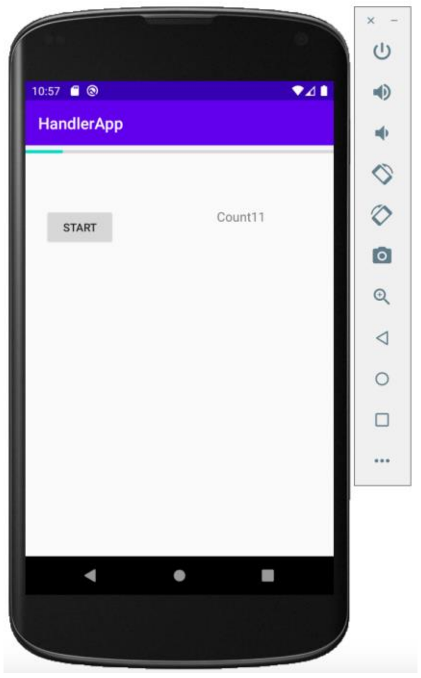
```
  // execute the hard will which will take a lot of time. For our example,
  // 1 second.
  private void executeHardWork() {
        try {
              Thread.sleep(1000);
        }
         catch (InterruptedException e) {
              e.printStackTrace();
        }
  }

// This class implements the methods for an async task to be executed
// The only required method is the doInBackground(Params... params). This
// method execute the big job in background. The other methods are not
// required but they are implemented here so you can better understand how
// tye work.
  //
  // Note that this class has three generic types assigned to Integer. These
  // types represents the arguments of the implemented methods.
  //
  // The first one, is passed when the async task is executed. It is an array
  // of necessary elements to be passed for the async task to be executed, in
  // case there is a need to do so. This parameter is used in the method doInBackground(...).
  //
  // The second parameter is eh type used for progress. Thus, when onProgressUpdate(...) is called,
  // the parameters for this methods are of the type of this second parameter.
  //
  // The third parameter is used for when the task is complete. Note that this parameter is the
  // return type of the method doInBackground(...) and the parameter of the methods onPostExecute(...)
  // and onCancelled(...).
class HugeWork extends AsyncTask<Integer, Integer, Integer> {

  // Method executed before the async task start. All things needed to be
  // setup before the async task must be done here. In this example we
  // simply display a message.
  @Override
  protected void onPreExecute() {
        txtMessage.setText("Executing async task...");
        super.onPreExecute();
  }

  // Here is where all the hard work is done. We simulate it by executing
  // a sleep for 1 second, 10 times. Each time the sleep is performed, we update
  // our progress in the method publishProgress(...). This method executes the
  // overridden method onProgressUpdate(...) which updates the progress.
        @Override
        protected Integer doInBackground(Integer... params) {

              // get the initial parameters. For us, this is the initial bar progress = 0
              int progress = ((Integer[])params)[0];

              do {

                    // only keep going in case the task was not cancelled
                    if (!this.isCancelled()) {
                          // execute hard work - sleep
                          executeHardWork();
                    }
                    else {
                          // in case the task was cancelled, break the loop
                          // and finish this task
                          break;
                    }

                    // upgrade progress
                    progress++;
                    publishProgress(progress);
              } while (progress <= MAX_PROGRESS);

              return progress;
        }
```

```
            // Every time the progress is informed, we update the progress bar
            @Override
            protected void onProgressUpdate(Integer... values) {
                  int progress = ((Integer[])values)[0];
                  mProgressBar.setProgress(progress);
                  super.onProgressUpdate(values);
            }

            // If the cancellation occurs, set the message informing so
            @Override
            protected void onCancelled(Integer result) {
                  txtMessage.setText(MessageFormat.format("Async task has been cancelled at {0} seconds.",
result - 1));
                  super.onCancelled(result);
            }

            // Method executed after the task is finished. If the task is cancelled this method is not
            // called. Here we display a finishing message and arrange the buttons.
            @Override
            protected void onPostExecute(Integer result) {
                  txtMessage.setText(MessageFormat.format("Async task execution finished in {0} seconds.",
result - 1));
                  btnCancel.setEnabled(false);
                  btnRestart.setEnabled(true);
                  super.onPostExecute(result);
            }
    }


}
```



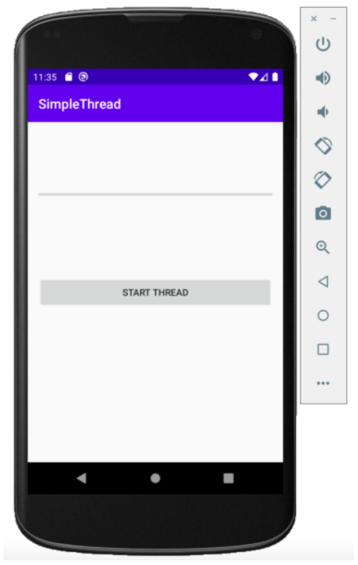
```java
package com.example.student.simplehandler;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.ProgressBar;
import android.widget.TextView;

import java.util.concurrent.atomic.AtomicBoolean;

public class MainActivity extends AppCompatActivity {
    ProgressBar bar;
    Handler handler=new Handler() {
        @Override
        public void handleMessage(Message msg) {
            bar.incrementProgressBy(msg.arg1);
            if(msg.arg1 == 345){
                TextView tv = (TextView)findViewById(R.id.mytextview);
                tv.setText("..that's all !");
            }
        }
    };
    AtomicBoolean isRunning=new AtomicBoolean(false);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        bar=(ProgressBar)findViewById(R.id.progressBar);
    }

    @Override
    public void onResume() {
        super.onResume();
        bar.setProgress(0);

        Thread background=new Thread(new Runnable() {
            public void run() {
                try {
                    for (int i=0;i<10 && isRunning.get();i++) {
                        Thread.sleep(5000);

                        Message msg=handler.obtainMessage();

                        msg.arg1=5;
                        msg.sendToTarget();
                    }
                }
                catch (Throwable t) {
                    // just end the background thread
                }

                // ready
                Message msg=handler.obtainMessage();
                msg.arg1=345;
                msg.sendToTarget();
            }
        });

        isRunning.set(true);
        background.start();
    }

    @Override
    public void onPause() {
        isRunning.set(false);
        super.onPause();
    }
}
```

```java
package ro.cunbm.handlerapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;


public class MainActivity extends AppCompatActivity {

    ProgressBar progressBar;
    Button startProgress, stopProgress;
    static TextView textView;
    int MAX = 100;
    static Handler mHandlerThread;
    private static final int START_PROGRESS = 100;
    private static final int UPDATE_COUNT = 101;
    static Thread thread1;


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

        progressBar = (ProgressBar) findViewById(R.id.progressBar);
        startProgress = (Button) findViewById(R.id.start_progress);
        textView = (TextView) findViewById(R.id.textView);
        progressBar.setMax(MAX);

        thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 100; i++) {
                    Log.d("I", ":" + i);
                    progressBar.setProgress(i);
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ex) {
                        ex.printStackTrace();
                    }
                    Message message = new Message();
                    message.what = UPDATE_COUNT;
                    message.arg1 = i;
                    mHandlerThread.sendMessage(message);
                }
            }
        });


        startProgress.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                int currentProgess = progressBar.getProgress();
                /*Message message = new Message();
                message.what = START_PROGRESS;*/
                mHandlerThread.sendEmptyMessage(START_PROGRESS);
            }
        });
    }
```



```
    private static class MyVeryOwnHandler extends Handler {
        // do cool stuff
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            if (msg.what == START_PROGRESS) {
                thread1.start();
            } else if (msg.what == UPDATE_COUNT) {
                textView.setText("Count" + msg.arg1);
            }
        }
    }

    @Override
    protected void onResume() {
        super.onResume();
        mHandlerThread = new MyVeryOwnHandler();
    }

    ;
}
```

```java
package ro.cunbm.simplethread;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;

public class SimpleThreadActivity extends AppCompatActivity {
    private ProgressBar mProgressBar;
    private Button mStartBtn;
    private Thread mThread;

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

        mProgressBar = (ProgressBar) findViewById(  R.id.progressBarThread);
        mStartBtn = (Button) findViewById(R.id.startBtnThread);

        mStartBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startProgress(view);
            }
        });
    }

    public void startProgress(View view) {
        mProgressBar.setProgress(0);
        mThread =  new Thread(){
            @Override
            public void run(){
                // Perform thread commands...
                for (int i = 0; i <= 10; i++) {
                    final int value = i;
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    mProgressBar.setProgress(value);
                }
                // Call the stopThread() method.
                stopThread(this);
            }
        };
        // Start the thread.
        mThread.start();
    }

    private synchronized void stopThread(Thread theThread)
    {
        if (theThread != null)
        {
            theThread = null;
        }
    }
}
```


11:35
SimpleThread
START THREAD