```
package es.uma.muii.apdm.parallelbasic;

import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.TextView;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class MainActivity extends AppCompatActivity {

    TextView stats;
    TextView incval;
    CheckBox asyncTaskBox;
    RadioButton threadsButton;
    RadioButton executorButton;
    Button doWorkButton;
    Button incrementButton;
    TextView correctText;
    long t0, t1;

    int work = 5000000;
    int nThreads = 4;
    long[] acc;

    // listener del boton de incrementar
    public void onClickIncrement(View v) {
        TextView tv = (TextView) findViewById(R.id.incrementval);
        tv.setText(String.valueOf(Integer.parseInt(tv.getText().toString()) + 1));
    }

    // listener del boton para hacer trabajo
    public void onClickDoWork(View v) {
        // se borran estadisticas e informacion de calculo correcto
        stats.setText(getText(R.string.stat).toString());
        correctText.setText("");
        if (!asyncTaskBox.isChecked()) {
            // No lanzar tarea asincrona para hacer los calculos
            doWorkButton.setEnabled(false);
            doAllWork();    // hacer el trabajo
            doWorkButton.setEnabled(true);
            stats.setText(getText(R.string.stat).toString() + " " + ((t1 - t0) / 10000001) +
 " ms");
            if (isCorrect())
                correctText.setText("Correct");
            else
                correctText.setText("Not correct");
        } else {
            // lanzar tarea asincrona para hacer el trabajo
            doWorkButton.setEnabled(false);
            // crear la tarea asincrona y ejecutarla
            new myAsyncTask().execute();
        }
    }

    // Tarea asincrona
    private class myAsyncTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... v) {
            // se hace el trabajo en el thread de background
            doAllWork();
            return null;
        }
        @Override
        protected void onPostExecute(Void voids) {
            // al terminar, en el thread UI se actualizan los elementos de la pantalla
            doWorkButton.setEnabled(true);
            stats.setText(getText(R.string.stat).toString() + " " + ((t1 - t0) / 10000001) +
 " ms");
            if (isCorrect())
                correctText.setText("Correct");
            else
                correctText.setText("Not correct");
        }
    }

    // metodo que realiza el trabajo
    public void doAllWork() {
        if (!threadsButton.isChecked() && !executorButton.isChecked()) {
            // Hacer trabajo usando un solo thread (serie)
            t0 = System.nanoTime();
            doSerialWork(); // hacer el trabajo en serie
            t1 = System.nanoTime();
        } else if (threadsButton.isChecked()) {
            // hacer el trabajo usando Java Threads
            t0 = System.nanoTime();
            doThreadWork();
            t1 = System.nanoTime();
        } else if (executorButton.isChecked()) {
            // hacer el trabajo usando un Executor
            t0 = System.nanoTime();
            doExecutorWork();
            t1 = System.nanoTime();
        }
    }

    // metodo para hacer el trabajo usando Executor (FixedThreadPool)
    public void doExecutorWork() {
        // se lee el numero de tareas especificado por el usuario
        int numtasks = Integer.parseInt(((EditText)findViewById(R.id.editTextNunThreads)).ge
tText().toString());
        if (numtasks > work) numtasks = work;

        // se crea un executor de typo FixedThreadPool con tantos threads como numero de pro
cesadores
        ExecutorService executor = Executors.newFixedThreadPool(nThreads);
        // se crean todas las tasks indicadas y se mandan a ejecutar en el executor
        for (int i=0; i < numtasks; i++) {
            executor.execute(new myTask(i, numtasks));
        }
        // se para el Executor para que no acepte mas tareas y termine las lanzadas
        executor.shutdown();
        try {
            // se espera a que terminen todas las tareas pendientes en el Executor
            executor.awaitTermination(10, TimeUnit.MINUTES);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
    // metodo para hacer el trabajo con Java Threads directamente
    public void doThreadWork() {
        // se lee el numero de tareas especificadas por el usuario (cada tarea se ejecutara
en un thread)
        int numtasks = Integer.parseInt(((EditText)findViewById(R.id.editTextNunThreads)).ge
tText().toString());
        if (numtasks > work) numtasks = work;

        Thread[] threads = new Thread[nThreads];
        int count = 0;
        while (count < numtasks) {
            // Se van ha crear tantos threads como tareas especificadas pero en tandas de nT
hreads cada vez
            int nth = 0;
            while (nth < nThreads && count < numtasks) {
                // se crean nThreads y se ponen a trabajar (start)
                threads[nth] = new Thread(new myTask(count, numtasks));
                threads[nth].start();
                nth++;
                count++;
            }
            for (int i = 0; i < nth; i++) {
                // se espera a que terminen los nThreads que estan trabajando antes de crear
 otros
                try {
                    threads[i].join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    // clase que representa las tareas, su metodo "run" es el que realiza el trabajo efectiv
o
    public class myTask implements Runnable {
        private int myId;
        private int nTasks;
        myTask(int _myId, int _nTasks) { myId = _myId; nTasks = _nTasks; }
        @Override
        public void run () { doWork(myId, nTasks); }  // metodo para hacer el trabajo de la
tarea
    }

    // metodo para hacer el trabajo completo en serie
    public void doSerialWork() {
        doWork(0 /* id del trozo */, 1 /* numero de trozos */);
    }

    // metodo para hacer un trozo del total del trabajo indicando el identificador del trozo
 y el numero total de trozos
    public void doWork(int taskId, int nTasks) {
        int ini = (int)((long)taskId*work/nTasks);
        int fin = (int)((long)(taskId+1)*work/nTasks);
        long aux=0;
        for (long i = ini; i < fin; i++) {
            for (long j = 0; j < 1000; j++)
                aux = i * i * i * i - i * i * i + i * i;
            acc[(int)i] = aux;
        }
    }

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

        incval = (TextView) findViewById(R.id.incrementval);
        incval.setText("0");
        stats = (TextView) findViewById(R.id.stats);
        correctText = (TextView) findViewById(R.id.correctTextView);
        asyncTaskBox = (CheckBox) findViewById(R.id.asynctaskcheckbox);
        threadsButton = (RadioButton) findViewById(R.id.javathreadradiobutton);
        executorButton = (RadioButton) findViewById(R.id.executorradiobutton);
        doWorkButton = (Button) findViewById(R.id.doworkbutton);
        doWorkButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onClickDoWork(v);
            }
        });

        incrementButton = (Button) findViewById(R.id.incrementbutton);
        incrementButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onClickIncrement(v);
            }
        });

        acc = new long[work];

        stats.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Log.i("APDM_", "stats pulsado");
                threadsButton.setChecked(false);
                executorButton.setChecked(false);
            }
        });
    }

    // metodo para comprobar que el resultado final hecho en paralelo es correcto
    private boolean isCorrect() {
        for (long i = 0; i < work; i++)
            if (acc[(int)i] != i * i * i * i  - i * i * i + i * i) {
                Log.i("APDM_", "Fallo en i "+i);
                return false;
            } else { acc[(int)i] = 0; }
        return true;
    }
}
```