```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package lab8.solution;

/**
 *
 * @author vanting
 */
public class CPUCores {

    public static void main(String[] args) {

        int cores = Runtime.getRuntime().availableProcessors();
        System.out.println("Number of logical cores: " + cores);

    }
}

```



```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package lab8.solution;

import java.util.Random;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

/**
 *
 * @author vanting
 */
public class ForkJoinPiEstimator {

    private static final ForkJoinPool POOL = new ForkJoinPool();
    private static final int DARTS = 100_000_000;
    static final int THRESHOLD = 10000;

    public static void main(String[] args) {

        System.out.println("This multithreaded program approximates PI using the Monte Carlo method.");

        long start = System.currentTimeMillis();
        double PI = computePI(DARTS);
        long time = System.currentTimeMillis() - start;

        System.out.println("Computed PI = " + PI + ", Difference = " + Math.abs(PI - Math.PI));
        System.out.println("Running Time (ms): " + time);
        System.out.println("Threshold: " + THRESHOLD);
    }

    public static double computePI(int numThrows) {
        int hits = POOL.invoke(new RecursiveThrowDartTask(numThrows));
        return 4.0 * hits / numThrows;
    }
}

/**
 * Note that a small threshold may result in lower accuracy in the estimation
 * due to the loss of samples in task division
 *
 * @author cwting
 */
class RecursiveThrowDartTask extends RecursiveTask<Integer> {

    private static final int THRESHOLD = ForkJoinPiEstimator.THRESHOLD;
    private final int numThrows;
    private final Random randomer = new Random();

    public RecursiveThrowDartTask(int numThrows) {
        this.numThrows = numThrows;
    }

    @Override
```

```
    protected Integer compute() {
        int hits = 0;

        if (numThrows < THRESHOLD) {
            for (int i = 0; i < numThrows; i++) {
                double x = randomer.nextDouble();
                double y = randomer.nextDouble();
                if (x * x + y * y < 1) {
                    hits++;
                }
            }
        } else {
            int sizeA = (int) Math.ceil(numThrows / 2.0);
            int sizeB = numThrows / 2;

            RecursiveThrowDartTask task1 = new RecursiveThrowDartTask(sizeA);
            RecursiveThrowDartTask task2 = new RecursiveThrowDartTask(sizeB);
            task1.fork();
            task2.fork();
            hits = task1.join() + task2.join();
        }

        return hits;
    }

}

```



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package lab8.solution;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

/**
 *
 * @author vanting
 */
public class MultiThreadPiEstimator {

    private static final int DARTS = 100000000;
    private static final int TASKS = 100;
    // private static final ExecutorService POOL = Executors.newCachedThreadPool();
    private static final ExecutorService POOL = Executors.newFixedThreadPool(TASKS);

    public static void main(String[] args) throws InterruptedException,
ExecutionException {

        System.out.println("This multithreaded program approximates PI using the
Monte Carlo method.");

        long start = System.currentTimeMillis();
        double PI = computePI(DARTS, TASKS);
        long time = System.currentTimeMillis() - start;

        POOL.shutdown(); // Terminate the threads
        System.out.println("Computed PI = " + PI + ", Difference = " + Math.abs(PI -
Math.PI));
        System.out.println("Running Time (ms): " + time);
        System.out.println("Level of paralleism: " + TASKS);
    }

    // public static double computePI(int numThrows) throws InterruptedException,
    // ExecutionException {
    // ThrowDartTask t1 = new ThrowDartTask(numThrows/2);
    // ThrowDartTask t2 = new ThrowDartTask(numThrows/2);
    // Future<Integer> f1 = POOL.submit(t1);
    // Future<Integer> f2 = POOL.submit(t2);
    // int hits = f1.get() + f2.get();
    // return 4.0 * hits / numThrows;
    // }

    public static double computePI(int numThrows, int numTasks) throws
InterruptedException, ExecutionException {
        List<ThrowDartTask> tasks = new ArrayList<>();
        for (int i = 0; i < numTasks; i++)
            tasks.add(new ThrowDartTask(numThrows / numTasks));
```

```
56
57          int hits = 0;
58          for (Future<Integer> f : POOL.invokeAll(tasks)) {
59              hits += f.get();
60          }
61          return 4.0 * hits / numThrows;
62      }
63  }
64
65  class ThrowDartTask implements Callable<Integer> {
66
67      private final int numThrows;
68      private final Random randomer = new Random();
69
70      public ThrowDartTask(int numThrows) {
71          this.numThrows = numThrows;
72      }
73
74      @Override
75      public Integer call() {
76          int hits = 0;
77
78          for (int i = 0; i < numThrows; i++) {
79
80              // Math.random is a synchronized method
81              // double x = Math.random();
82              // double y = Math.random();
83
84              // reduce contention for each thread to have its own
85              // pseudorandom-number generator
86              double x = randomer.nextDouble();
87              double y = randomer.nextDouble();
88
89              if (x * x + y * y < 1)
90                  hits++;
91          }
92
93          return hits;
94      }
95
96  }
97
```