REPORT

#problem1_the parallel performance of my code

과목명	멀티코어컴퓨팅 01분반
교수명	손봉수
학번	20206319
학과	소프트웨어학부 소프트웨어전공
이름	김가연

(a) What environment (e.g. CPU type, number of cores, memory size, OS type) the experimentation was performed.

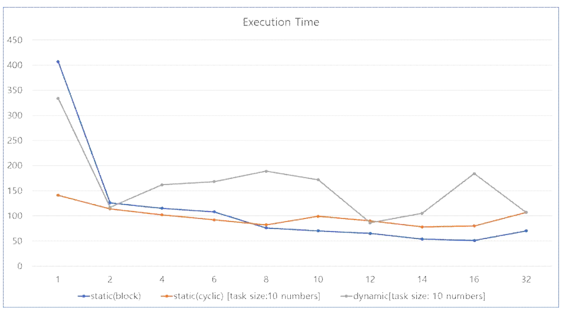
The CPU type of this machine is Intel Core i5-5200U.
It has two cores, four logical processors(when hyperthreading on), and clock speed is 2.20 GHz.
The memory size is 8.0 GB and is DDR3. The speed of the memory is 1600 MHz.
The OS type is 64-bit operating system, x64 based processor.
Experiments were conducted in the same environment as above. Unfortunately, it did not perform on PCs with more than the recommended 4 cores.

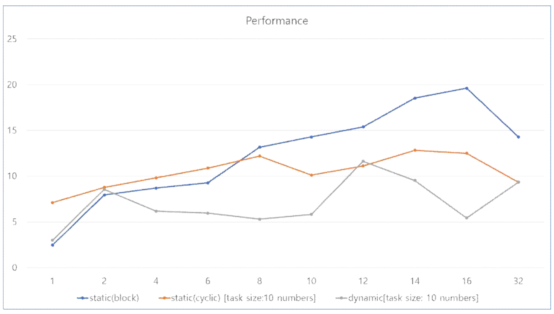
(b) Tables and graphs that show the execution time (unit: ms) for the number of entire threads = {1,2,4,6,8,10,12,14,16,32}.

*The unit of execution time is million seconds.
*The performance is expressed up to 3 decimal places.

exec time	1	2	4	6	8	10	12	14	16	32
static(block)	407	126	115	108	76	70	65	54	51	70
static (cyclic) [task size : 10 numbers]	141	114	102	92	82	99	90	78	80	107
dynamic [task size : 10 numbers]	334	117	162	168	189	172	86	105	184	107

performace (1/exec time)	1	2	4	6	8	10	12	14	16	32
static(block)	2.457	7.937	8.696	9.259	13.158	14.286	15.385	18.519	19.608	14.286
static (cyclic) [task size : 10 numbers]	7.092	8.772	9.804	10.870	12.195	10.101	11.111	12.821	12.5	9.346
dynamic [task size : 10 numbers]	2.994	8.547	6.173	5.952	5.291	5.814	11.628	9.524	5.435	9.346





(c) Explanation/analysis on the results and why such results are obtained with sufficient details.

All load balancing shows sudden time reduction when using two threads in a single thread. The above example of determining a prime number among a large number is fast by deploying multiple threads according to a given interval. Multi-threading is more efficient than single-threading because it allows operations to be completed, reducing execution time. It shows an increase in performance.

Static Load Balancing shows a tendency for execution time to decrease up to 16 threads in general at 32 threads it rises again. In the case of Dynamic, the shape of the graph is jagged. This was analyzed according to each case as follows.

In the case of Static Load Balancing with Block Decomposition, the execution time is reduced to 16 threads. Performance shows improvement. However, when using 32 threads, execution time increases and performance decreases. Similarly, the case of Static Load Balancing with Cyclic Decomposition shows a similar pattern.

The reasons for this were analyzed in each case.

For Static Load Balancing with Block Decomposition:

1. 32 threads may not be optimal depending on the execution environment. In addition, if the number of threads is set too high, performance may deteriorate.

2. In my current code, I split the work by the number of threads. However, this split has the potential for an unbalanced distribution of work. So I might consider breaking my work into smaller chunks or dynamically distributing my work.

3. Current code is a CPU bound task, the thread is CPU intensive. Therefore, performance may decrease if other CPU-intensive tasks are being performed. In this case, performance degradation can be prevented by changing the task to an I/O bound task or by considering other methods to reduce CPU usage.

4. In the current code, each thread uses a common variable, counter. In this case, synchronization problems may occur, which can cause performance degradation, so it is recommended to design the resources not to be shared.

For Static Load Balancing with Cyclic Decomposition:

1. The current code partitions each thread to do a fixed-size task. This prevents threads from doing more work even when they can do it faster. Also, if the size of the task is small, the load balancing between threads can become unbalanced.

2. The current code allows each thread to have read and write access to its own worklist. However, resource contention can occur when multiple threads try to access the same work list at the same time. This resource contention can lead to poor performance and longer execution times between threads.

3. In my current code, the number of threads is fixed. However, in other configurations of the computer more threads are available. Therefore, it is

recommended to dynamically adjust the number of threads to improve performance.

4. It is possible to improve performance by performing optimizations on the code. These optimizations can be code improvements or algorithm changes. I can also improve performance by doing things like tuning the JVM configuration and garbage collection parameters.

For Dynamic Load Balancing:

1. If the thread value is too large or too small, performance may be affected. If the Threads value is too small, processing time can be long because one thread must handle all the work. If the Threads value is too large, contention between threads may occur or overhead for thread creation and management may occur.

2. There may be overhead for creating and managing threads. If the Threads value is too large, the overhead of creating and managing threads can result in performance degradation.

3. The synchronized keyword is used to synchronize shared resources. Synchronization of shared resources is used to reduce contention between threads. However, using the synchronized keyword can cause competition between threads, which can reduce performance. Also, the cost of using the synchronized keyword increases with more shared resources, which can lead to performance degradation.

Therefore, properly throttling the thread value and minimizing synchronization processing on shared resources are important to improve performance. I can also use a thread pool to reduce the overhead of creating and managing threads.

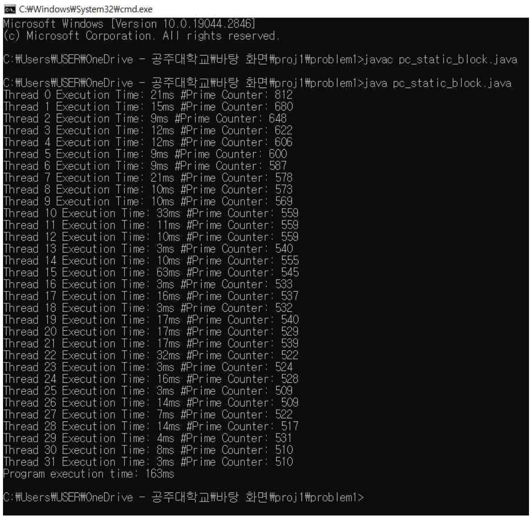
(d) Entire JAVA source code and explain my code, screen capture image of program execution and output, briefly explain how to compile and execute the source code I submit.

The java version of the created class is 20. The Java version of the running PC must be 20 so that the error window does not appear. The folder contains both java code and jar files. The following proposes a way to run both java code and jar files.

First, how to execute java code. In the Command Prompt window, go to the path of the folder containing the written code. Compilation is completed by entering the javac java_filename.java command. Then, if you enter the java java_file name.java command, it will be executed.

Here's how to run a jar file. After moving to the path of the folder in the command prompt window, enter the command java -jar jar_filename.jar to execute it.

The pictures below are example screens. The left side is a java file, and the right side is a screen executed with a jar file.

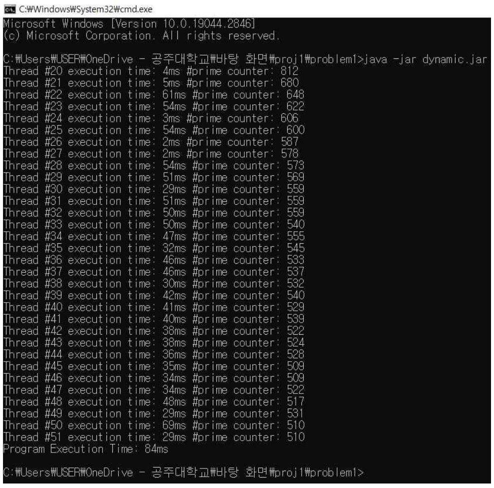


```
C:￦Windows￦System32￦cmd.exe
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:￦Users￦USER￦OneDrive - 공주대학교￦바탕 화면￦proj1￦problem1>javac pc_static_block.java

C:￦Users￦USER￦OneDrive - 공주대학교￦바탕 화면￦proj1￦problem1>java pc_static_block.java
Thread 0 Execution Time: 21ms #Prime Counter: 812
Thread 1 Execution Time: 15ms #Prime Counter: 680
Thread 2 Execution Time: 9ms #Prime Counter: 648
Thread 3 Execution Time: 12ms #Prime Counter: 622
Thread 4 Execution Time: 12ms #Prime Counter: 606
Thread 5 Execution Time: 9ms #Prime Counter: 600
Thread 6 Execution Time: 9ms #Prime Counter: 587
Thread 7 Execution Time: 21ms #Prime Counter: 578
Thread 8 Execution Time: 10ms #Prime Counter: 573
Thread 9 Execution Time: 10ms #Prime Counter: 569
Thread 10 Execution Time: 33ms #Prime Counter: 559
Thread 11 Execution Time: 11ms #Prime Counter: 559
Thread 12 Execution Time: 10ms #Prime Counter: 559
Thread 13 Execution Time: 3ms #Prime Counter: 540
Thread 14 Execution Time: 10ms #Prime Counter: 555
Thread 15 Execution Time: 63ms #Prime Counter: 545
Thread 16 Execution Time: 3ms #Prime Counter: 533
Thread 17 Execution Time: 16ms #Prime Counter: 537
Thread 18 Execution Time: 3ms #Prime Counter: 532
Thread 19 Execution Time: 17ms #Prime Counter: 540
Thread 20 Execution Time: 17ms #Prime Counter: 529
Thread 21 Execution Time: 17ms #Prime Counter: 539
Thread 22 Execution Time: 32ms #Prime Counter: 522
Thread 23 Execution Time: 3ms #Prime Counter: 524
Thread 24 Execution Time: 16ms #Prime Counter: 528
Thread 25 Execution Time: 3ms #Prime Counter: 509
Thread 26 Execution Time: 14ms #Prime Counter: 509
Thread 27 Execution Time: 7ms #Prime Counter: 522
Thread 28 Execution Time: 14ms #Prime Counter: 517
Thread 29 Execution Time: 4ms #Prime Counter: 531
Thread 30 Execution Time: 8ms #Prime Counter: 510
Thread 31 Execution Time: 3ms #Prime Counter: 510
Program execution time: 163ms

C:￦Users￦USER￦OneDrive - 공주대학교￦바탕 화면￦proj1￦problem1>
```



```
C:￦Windows￦System32￦cmd.exe
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:￦Users￦USER￦OneDrive - 공주대학교￦바탕 화면￦proj1￦problem1>java -jar dynamic.jar
Thread #20 execution time: 4ms #prime counter: 812
Thread #21 execution time: 5ms #prime counter: 680
Thread #22 execution time: 61ms #prime counter: 648
Thread #23 execution time: 54ms #prime counter: 622
Thread #24 execution time: 3ms #prime counter: 606
Thread #25 execution time: 54ms #prime counter: 600
Thread #26 execution time: 2ms #prime counter: 587
Thread #27 execution time: 2ms #prime counter: 578
Thread #28 execution time: 54ms #prime counter: 573
Thread #29 execution time: 51ms #prime counter: 569
Thread #30 execution time: 29ms #prime counter: 559
Thread #31 execution time: 51ms #prime counter: 559
Thread #32 execution time: 50ms #prime counter: 559
Thread #33 execution time: 50ms #prime counter: 540
Thread #34 execution time: 47ms #prime counter: 555
Thread #35 execution time: 32ms #prime counter: 545
Thread #36 execution time: 46ms #prime counter: 533
Thread #37 execution time: 46ms #prime counter: 537
Thread #38 execution time: 30ms #prime counter: 532
Thread #39 execution time: 42ms #prime counter: 540
Thread #40 execution time: 41ms #prime counter: 529
Thread #41 execution time: 40ms #prime counter: 539
Thread #42 execution time: 38ms #prime counter: 522
Thread #43 execution time: 38ms #prime counter: 524
Thread #44 execution time: 36ms #prime counter: 528
Thread #45 execution time: 35ms #prime counter: 509
Thread #46 execution time: 34ms #prime counter: 509
Thread #47 execution time: 34ms #prime counter: 522
Thread #48 execution time: 48ms #prime counter: 517
Thread #49 execution time: 29ms #prime counter: 531
Thread #50 execution time: 69ms #prime counter: 510
Thread #51 execution time: 29ms #prime counter: 510
Program Execution Time: 84ms

C:￦Users￦USER￦OneDrive - 공주대학교￦바탕 화면￦proj1￦problem1>
```

#pc_static_block.java source code

```java
package problem1;

import java.util.*;

public class pc_static_block {
private static int NUM_END = 200000;
private static int NUM_THREAD = 4;

public static void main(String[] args) throws InterruptedException {
if (args.length == 2) {
NUM_THREAD = Integer.parseInt(args[0]);
NUM_END = Integer.parseInt(args[1]);
}

long startTime = System.currentTimeMillis();
int num = NUM_END / NUM_THREAD, counter = 0;
List<MultiThread> threads = new ArrayList<>();

for (int i = 0; i < NUM_THREAD; i++) {
MultiThread thread = new MultiThread(i * num, (i + 1) * num);
threads.add(thread);
thread.start();
}

for (MultiThread thread : threads) {
thread.join();
counter += thread.getCounter();
}

long endTime = System.currentTimeMillis();
long timeDiff = endTime - startTime;

for (int i = 0; i < threads.size(); i++) {
MultiThread thread = threads.get(i);
System.out.println("Thread " + i + " Execution Time: " + thread.getThread_ExecutionTime() +
"ms #Prime Counter: " + thread.getCounter());
}
System.out.println("Program execution time: " + timeDiff + "ms");
}

private static boolean isPrime(int x) {
if (x <= 1) return false;

for (int i = 2; i <= Math.sqrt(x); i++) {
if (x % i == 0) return false;
}
return true;
}
```

```java
private static class MultiThread extends Thread {
private final int start, end;
private int counter = 0;
private long thread_executionTime;

public MultiThread(int start, int end) {
this.start = start;
this.end = end;
}

public int getCounter() { return counter; }
public long getThread_ExecutionTime() { return thread_executionTime; }

@Override
public void run() {
long startTime = System.currentTimeMillis();
for (int i = start; i < end; i++) {
if (isPrime(i)) counter++;
}
long endTime = System.currentTimeMillis();
thread_executionTime = endTime - startTime;
}
}
}
```

1. Main method

- Determines the number of threads running (NUM_THREAD) and the number of tasks(NUM_END) based on user input.
- Creates NUM_THREAD threads and assigns start and end points to each thread.
- Starts the created threads, and waits until all threads are finished.
- Adds the number of prime numbers each thread finds, and prints the running time of each thread.
- Outputs the total execution time of the program.

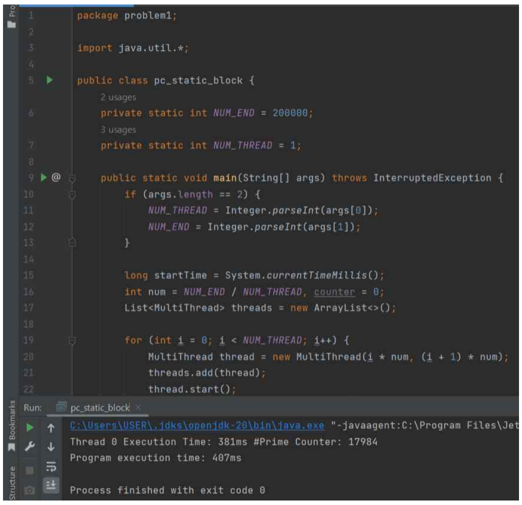
2. isPrime method

- Determines whether an input number is prime or not.
- Determines whether a number is prime by checking if it is divisible by a number from 2 to the square root of that number. Returns true if prime, otherwise returns false.

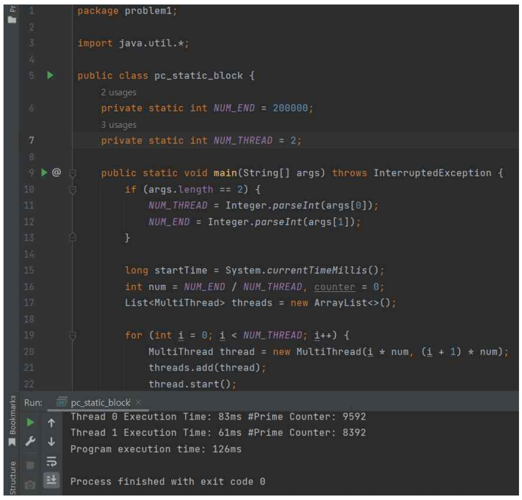
3. MultiThread class

- Creates and executes threads.
- Takes a start and an end, performs a prime number check, stores the number of primes checked and the execution time, and returns the result when called from the main thread.
- There is a method to output the execution time of each thread (getCounter) and a method to output the number of prime numbers (getThread_ExecutionTime).

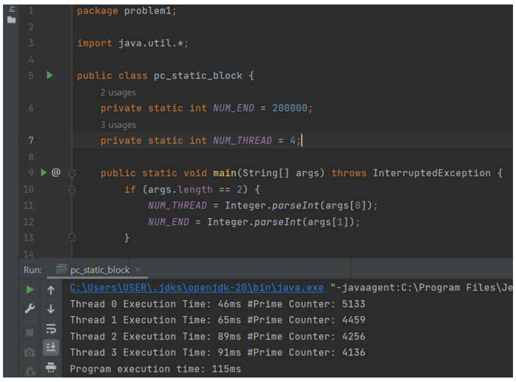
case 1)1 thread



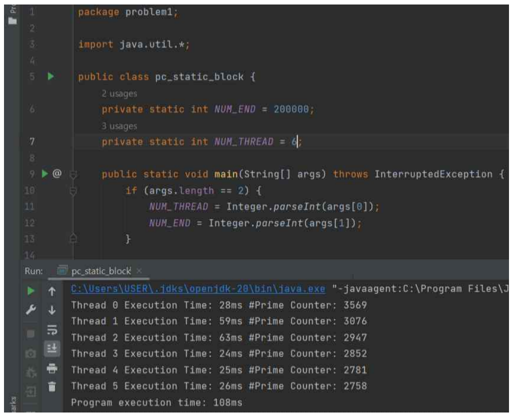
case 2)2 threads



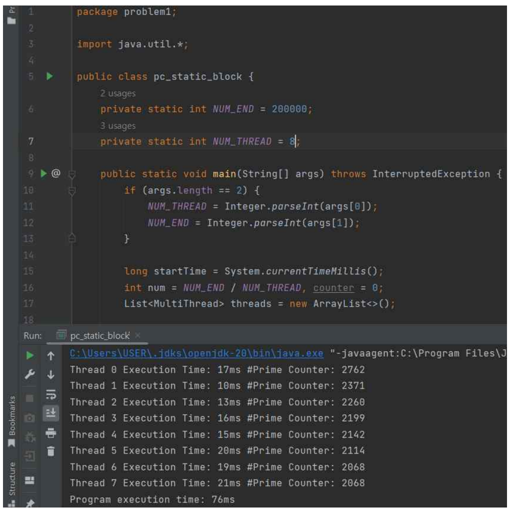
case 3)4 threads



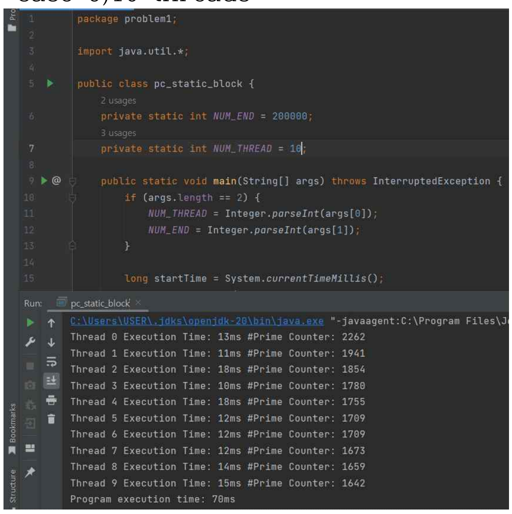
case 4)6 threads



case 5)8 threads



case 6)10 threads



case 7)12 threads

```
package problem1;

import java.util.*;

public class pc_static_block {
    2 usages
    private static int NUM_END = 200000;
    3 usages
    private static int NUM_THREAD = 12;

    public static void main(String[] args) throws InterruptedException {
        if (args.length == 2) {
            NUM_THREAD = Integer.parseInt(args[0]);
            NUM_END = Integer.parseInt(args[1]);
```

```
Run: pc_static_block
C:\Users\USER\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\J
Thread 0 Execution Time: 12ms #Prime Counter: 1928
Thread 1 Execution Time: 6ms #Prime Counter: 1641
Thread 2 Execution Time: 8ms #Prime Counter: 1563
Thread 3 Execution Time: 8ms #Prime Counter: 1513
Thread 4 Execution Time: 10ms #Prime Counter: 1489
Thread 5 Execution Time: 13ms #Prime Counter: 1458
Thread 6 Execution Time: 10ms #Prime Counter: 1422
Thread 7 Execution Time: 10ms #Prime Counter: 1430
Thread 8 Execution Time: 10ms #Prime Counter: 1404
Thread 9 Execution Time: 10ms #Prime Counter: 1377
Thread 10 Execution Time: 11ms #Prime Counter: 1381
Thread 11 Execution Time: 11ms #Prime Counter: 1377
Program execution time: 65ms
```

case 8)14 threads

```
package problem1;

import java.util.*;

public class pc_static_block {
    2 usages
    private static int NUM_END = 200000;
    3 usages
    private static int NUM_THREAD = 14;

    public static void main(String[] args) throws InterruptedException {
        if (args.length == 2) {
            NUM_THREAD = Integer.parseInt(args[0]);
            NUM_END = Integer.parseInt(args[1]);
```

```
Run: pc_static_block
C:\Users\USER\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\J
Thread 0 Execution Time: 8ms #Prime Counter: 1676
Thread 1 Execution Time: 13ms #Prime Counter: 1431
Thread 2 Execution Time: 7ms #Prime Counter: 1374
Thread 3 Execution Time: 8ms #Prime Counter: 1313
Thread 4 Execution Time: 8ms #Prime Counter: 1280
Thread 5 Execution Time: 10ms #Prime Counter: 1267
Thread 6 Execution Time: 11ms #Prime Counter: 1251
Thread 7 Execution Time: 7ms #Prime Counter: 1222
Thread 8 Execution Time: 8ms #Prime Counter: 1224
Thread 9 Execution Time: 8ms #Prime Counter: 1214
Thread 10 Execution Time: 8ms #Prime Counter: 1194
Thread 11 Execution Time: 10ms #Prime Counter: 1169
Thread 12 Execution Time: 9ms #Prime Counter: 1191
Thread 13 Execution Time: 10ms #Prime Counter: 1177
Program execution time: 54ms
```

case 9)16 threads

```
package problem1;

import java.util.*;

public class pc_static_block {
    2 usages
    private static int NUM_END = 200000;
    3 usages
    private static int NUM_THREAD = 16;

    public static void main(String[] args) throws InterruptedException {
        if (args.length == 2) {
            NUM_THREAD = Integer.parseInt(args[0]);
```

```
Run: pc_static_block
Thread 0 Execution Time: 10ms #Prime Counter: 1492
Thread 1 Execution Time: 9ms #Prime Counter: 1270
Thread 2 Execution Time: 6ms #Prime Counter: 1206
Thread 3 Execution Time: 6ms #Prime Counter: 1165
Thread 4 Execution Time: 8ms #Prime Counter: 1142
Thread 5 Execution Time: 7ms #Prime Counter: 1118
Thread 6 Execution Time: 7ms #Prime Counter: 1099
Thread 7 Execution Time: 8ms #Prime Counter: 1100
Thread 8 Execution Time: 7ms #Prime Counter: 1070
Thread 9 Execution Time: 6ms #Prime Counter: 1072
Thread 10 Execution Time: 9ms #Prime Counter: 1068
Thread 11 Execution Time: 7ms #Prime Counter: 1046
Thread 12 Execution Time: 9ms #Prime Counter: 1037
Thread 13 Execution Time: 9ms #Prime Counter: 1031
Thread 14 Execution Time: 10ms #Prime Counter: 1048
Thread 15 Execution Time: 8ms #Prime Counter: 1020
Program execution time: 51ms
```

case 10)32 threads

```
package problem1;

import java.util.*;

public class pc_static_block {
    2 usages
    private static int NUM_END = 200000;
    3 usages
    private static int NUM_THREAD = 32;

    public static void main(String[] args) throws InterruptedException {
        if (args.length == 2) {
            NUM_THREAD = Integer.parseInt(args[0]);
```

```
Run: pc_static_block
Thread 0 Execution Time: 8ms #Prime Counter: 812
Thread 1 Execution Time: 2ms #Prime Counter: 680
Thread 2 Execution Time: 3ms #Prime Counter: 648
Thread 3 Execution Time: 4ms #Prime Counter: 622
Thread 4 Execution Time: 3ms #Prime Counter: 606
Thread 5 Execution Time: 4ms #Prime Counter: 600
Thread 6 Execution Time: 3ms #Prime Counter: 587
Thread 7 Execution Time: 3ms #Prime Counter: 578
Thread 8 Execution Time: 5ms #Prime Counter: 573
Thread 9 Execution Time: 3ms #Prime Counter: 569
Thread 10 Execution Time: 3ms #Prime Counter: 559
Thread 11 Execution Time: 4ms #Prime Counter: 559
Thread 12 Execution Time: 5ms #Prime Counter: 559
Thread 13 Execution Time: 3ms #Prime Counter: 540
Thread 14 Execution Time: 5ms #Prime Counter: 555
Thread 15 Execution Time: 5ms #Prime Counter: 545
Thread 16 Execution Time: 3ms #Prime Counter: 533
```

```
Run: pc_static_block
Thread 17 Execution Time: 4ms #Prime Counter: 537
Thread 18 Execution Time: 5ms #Prime Counter: 532
Thread 19 Execution Time: 4ms #Prime Counter: 540
Thread 20 Execution Time: 5ms #Prime Counter: 529
Thread 21 Execution Time: 4ms #Prime Counter: 539
Thread 22 Execution Time: 4ms #Prime Counter: 522
Thread 23 Execution Time: 4ms #Prime Counter: 524
Thread 24 Execution Time: 7ms #Prime Counter: 528
Thread 25 Execution Time: 4ms #Prime Counter: 509
Thread 26 Execution Time: 5ms #Prime Counter: 509
Thread 27 Execution Time: 3ms #Prime Counter: 522
Thread 28 Execution Time: 5ms #Prime Counter: 517
Thread 29 Execution Time: 3ms #Prime Counter: 531
Thread 30 Execution Time: 4ms #Prime Counter: 510
Thread 31 Execution Time: 4ms #Prime Counter: 510
Program execution time: 70ms
```

#pc_static_cyclic.java source code

```
package problem1;

import java.util.*;

public class pc_static_cyclic {
private static int NUM_END = 200000;
private static int NUM_THREAD = 4;
private static int TASK_SIZE = 10;
private static int TASKS_PER_THREAD = NUM_END / (NUM_THREAD * TASK_SIZE);
private static List<List<Integer>> taskList = new ArrayList<>();

public static void main(String[] args) throws InterruptedException {
if (args.length == 2) {
NUM_THREAD = Integer.parseInt(args[0]);
NUM_END = Integer.parseInt(args[1]);
TASKS_PER_THREAD = NUM_END / (NUM_THREAD * TASK_SIZE);
}

long startTime = System.currentTimeMillis();
int counter = 0;
createTasks();
List<MultiThread> threads = new ArrayList<>();

for (int i = 0; i < NUM_THREAD; i++) {
MultiThread thread = new MultiThread(i, taskList.get(i));
threads.add(thread);
thread.start();
}

long endTime = 0;

for (MultiThread thread : threads) {
thread.join();
counter += thread.getCounter();
endTime = Math.max(endTime, thread.getEndTime());
}

long timeDiff = endTime - startTime;

for (int i = 0; i < threads.size(); i++) {
MultiThread thread = threads.get(i);
System.out.println("Thread " + i + " Execution Time: " + thread.getExecutionTime() + "ms
#Prime Counter: " + thread.getCounter());
}
System.out.println("Program Execution Time: " + timeDiff + "ms");
}

private static boolean isPrime(int x) {
if (x <= 1) return false;

for (int i = 2; i <= Math.sqrt(x); i++) {
if (x % i == 0) return false;
}
return true;
}

private static void createTasks() {
for (int i = 0; i < NUM_THREAD; i++) {
List<Integer> task = new ArrayList<>();
for (int j = 0; j < TASKS_PER_THREAD; j++) {
int start = i * TASK_SIZE * TASKS_PER_THREAD + j * TASK_SIZE + 1;
int end = start + TASK_SIZE - 1;
if ( (i == NUM_THREAD - 1) && (j == TASKS_PER_THREAD - 1)) end = NUM_END;
task.add(start);
task.add(end);
}
taskList.add(task);
}
}

private static class MultiThread extends Thread {
```

```
private final int num;
private final List<Integer> task;
private int counter = 0;
private long thread_executionTime, endTime;

public MultiThread(int num, List<Integer> task) {
this.num = num;
this.task = task;
}

public int getCounter() { return counter; }
public long getExecutionTime() { return thread_executionTime; }
public long getEndTime() { return endTime; }

@Override
public void run() {
long startTime = System.currentTimeMillis();
for (int i = 0; i < task.size(); i += 2) {
int start = task.get(i);
int end = task.get(i + 1);
for (int j = start; j <= end; j++) {
if (isPrime(j)) counter++;
}
}
long currentTime = System.currentTimeMillis();
thread_executionTime = currentTime - startTime;
endTime = currentTime;
}
}
}
```

1. Main method

- Initialize the number of threads (NUM_THREAD), the number of tasks(NUM_END), the number of tasks allocated to each thread (TASKS_PER_THREAD), and the task list (taskList), and call the createTasks() function to create taskList.

- Create and start a MultiThread instance using the created task list.

- When each MultiThread instance finishes execution and returns a result, the execution time of each thread and the number of decimals are output.

- Outputs the entire program time (timeDiff).

2. isPrime method

- Determines whether an input number is prime or not.

- Determines whether a number is prime by checking if it is divisible by a number from 2 to the square root of that number. Returns true if prime, otherwise returns false.

3. createTasks method

- This function creates a taskList with a total number of NUM_THREADs.

- Each task list is allocated by dividing the range of numbers up to NUM_END by the size of TASK_SIZE.

- The last working list adjusts the range to handle the rest of the numbers.

4. MultiThread class

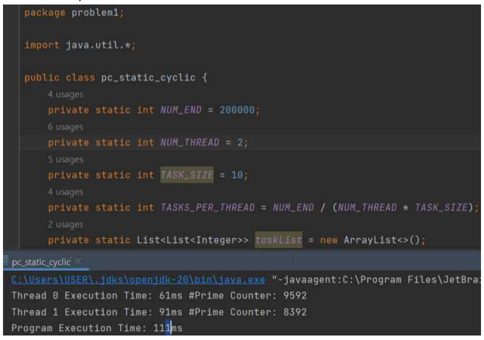
- When creating a MultiThread instance, pass the task list (task) and thread number (num).

- In the run() function, the isPrime function is used to determine whether the numbers within the range allocated to the work list are prime numbers, and if they are prime numbers, the counter variable is increased.

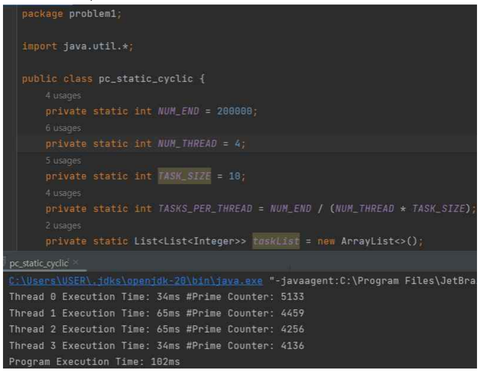
- There is a method to output the execution time of each thread (getCounter) and a method to output the number of prime numbers (getThread_ExecutionTime).

case 1)1 thread

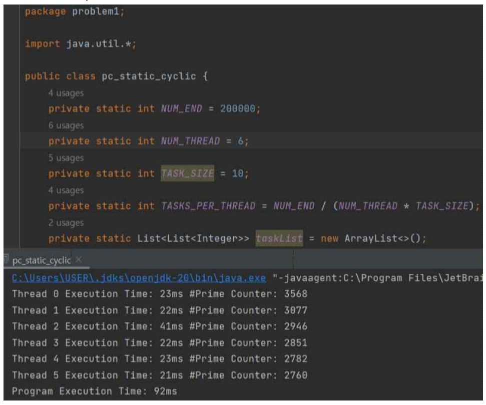
case 2)2 threads



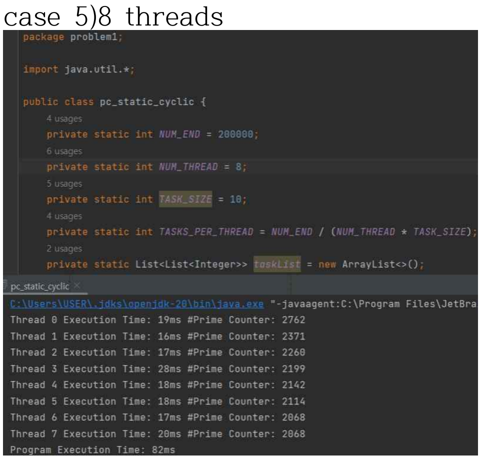
case 3)4 threads



case 4)6 threads



case 5)8 threads



case 6)10 threads

case 7)12 threads

```
package problem1;

import java.util.*;

public class pc_static_cyclic {
    4 usages
    private static int NUM_END = 200000;
    6 usages
    private static int NUM_THREAD = 12;
    5 usages
    private static int TASK_SIZE = 10;
    4 usages
    private static int TASKS_PER_THREAD = NUM_END / (NUM_THREAD * TASK_SIZE);
    2 usages
    private static List<List<Integer>> taskList = new ArrayList<>();
```

```
pc_static_cyclic
C:\Users\USER\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrai
Thread 0 Execution Time: 17ms #Prime Counter: 1927
Thread 1 Execution Time: 11ms #Prime Counter: 1640
Thread 2 Execution Time: 10ms #Prime Counter: 1563
Thread 3 Execution Time: 9ms #Prime Counter: 1513
Thread 4 Execution Time: 12ms #Prime Counter: 1490
Thread 5 Execution Time: 9ms #Prime Counter: 1455
Thread 6 Execution Time: 16ms #Prime Counter: 1424
Thread 7 Execution Time: 11ms #Prime Counter: 1427
Thread 8 Execution Time: 12ms #Prime Counter: 1405
Thread 9 Execution Time: 12ms #Prime Counter: 1372
Thread 10 Execution Time: 13ms #Prime Counter: 1382
Thread 11 Execution Time: 14ms #Prime Counter: 1386
Program Execution Time: 90ms
```

case 8)14 threads

```
package problem1;

import java.util.*;

public class pc_static_cyclic {
    4 usages
    private static int NUM_END = 200000;
    6 usages
    private static int NUM_THREAD = 14;
    5 usages
    private static int TASK_SIZE = 10;
    4 usages
    private static int TASKS_PER_THREAD = NUM_END / (NUM_THREAD * TASK_SIZE);
    2 usages
    private static List<List<Integer>> taskList = new ArrayList<>();
```

```
pc_static_cyclic
Thread 0 Execution Time: 8ms #Prime Counter: 1675
Thread 1 Execution Time: 11ms #Prime Counter: 1432
Thread 2 Execution Time: 11ms #Prime Counter: 1372
Thread 3 Execution Time: 15ms #Prime Counter: 1313
Thread 4 Execution Time: 11ms #Prime Counter: 1279
Thread 5 Execution Time: 9ms #Prime Counter: 1268
Thread 6 Execution Time: 10ms #Prime Counter: 1249
Thread 7 Execution Time: 11ms #Prime Counter: 1223
Thread 8 Execution Time: 9ms #Prime Counter: 1223
Thread 9 Execution Time: 9ms #Prime Counter: 1215
Thread 10 Execution Time: 10ms #Prime Counter: 1192
Thread 11 Execution Time: 11ms #Prime Counter: 1171
Thread 12 Execution Time: 9ms #Prime Counter: 1187
Thread 13 Execution Time: 11ms #Prime Counter: 1185
Program Execution Time: 78ms
```

case 9)16 threads

```
package problem1;

import java.util.*;

public class pc_static_cyclic {
    4 usages
    private static int NUM_END = 200000;
    6 usages
    private static int NUM_THREAD = 16;
    5 usages
    private static int TASK_SIZE = 10;
    4 usages
```

```
pc_static_cyclic
Thread 0 Execution Time: 11ms #Prime Counter: 1492
Thread 1 Execution Time: 13ms #Prime Counter: 1270
Thread 2 Execution Time: 8ms #Prime Counter: 1206
Thread 3 Execution Time: 8ms #Prime Counter: 1165
Thread 4 Execution Time: 8ms #Prime Counter: 1142
Thread 5 Execution Time: 7ms #Prime Counter: 1118
Thread 6 Execution Time: 7ms #Prime Counter: 1099
Thread 7 Execution Time: 8ms #Prime Counter: 1100
Thread 8 Execution Time: 8ms #Prime Counter: 1070
Thread 9 Execution Time: 9ms #Prime Counter: 1072
Thread 10 Execution Time: 9ms #Prime Counter: 1068
Thread 11 Execution Time: 9ms #Prime Counter: 1046
Thread 12 Execution Time: 9ms #Prime Counter: 1037
Thread 13 Execution Time: 10ms #Prime Counter: 1031
Thread 14 Execution Time: 9ms #Prime Counter: 1048
Thread 15 Execution Time: 11ms #Prime Counter: 1020
Program Execution Time: 80ms
```

case 10)32 threads

```
package problem1;

import java.util.*;

public class pc_static_cyclic {
    4 usages
    private static int NUM_END = 200000;
    6 usages
    private static int NUM_THREAD = 32;
    5 usages
    private static int TASK_SIZE = 10;
    4 usages
```

```
pc_static_cyclic
Thread 0 Execution Time: 15ms #Prime Counter: 812
Thread 1 Execution Time: 7ms #Prime Counter: 680
Thread 2 Execution Time: 4ms #Prime Counter: 648
Thread 3 Execution Time: 3ms #Prime Counter: 622
Thread 4 Execution Time: 4ms #Prime Counter: 606
Thread 5 Execution Time: 3ms #Prime Counter: 600
Thread 6 Execution Time: 4ms #Prime Counter: 587
Thread 7 Execution Time: 3ms #Prime Counter: 578
Thread 8 Execution Time: 6ms #Prime Counter: 573
Thread 9 Execution Time: 3ms #Prime Counter: 569
Thread 10 Execution Time: 3ms #Prime Counter: 559
Thread 11 Execution Time: 4ms #Prime Counter: 559
Thread 12 Execution Time: 4ms #Prime Counter: 559
Thread 13 Execution Time: 4ms #Prime Counter: 540
Thread 14 Execution Time: 3ms #Prime Counter: 555
Thread 15 Execution Time: 4ms #Prime Counter: 545
Thread 16 Execution Time: 4ms #Prime Counter: 533
Thread 17 Execution Time: 5ms #Prime Counter: 537
```

```
pc_static_cyclic
Thread 18 Execution Time: 6ms #Prime Counter: 532
Thread 19 Execution Time: 4ms #Prime Counter: 540
Thread 20 Execution Time: 4ms #Prime Counter: 529
Thread 21 Execution Time: 5ms #Prime Counter: 539
Thread 22 Execution Time: 4ms #Prime Counter: 522
Thread 23 Execution Time: 5ms #Prime Counter: 524
Thread 24 Execution Time: 5ms #Prime Counter: 528
Thread 25 Execution Time: 5ms #Prime Counter: 509
Thread 26 Execution Time: 5ms #Prime Counter: 509
Thread 27 Execution Time: 4ms #Prime Counter: 522
Thread 28 Execution Time: 5ms #Prime Counter: 517
Thread 29 Execution Time: 5ms #Prime Counter: 531
Thread 30 Execution Time: 5ms #Prime Counter: 510
Thread 31 Execution Time: 5ms #Prime Counter: 510
Program Execution Time: 107ms
```

#pc_dynamic.java source code

```
package problem1;

import java.util.*;

public class pc_dynamic {
private static int NUM_END = 200000;
private static int NUM_THREAD = 4;
private static final int TASK_SIZE = 10;
private static final Object lock = new Object();

public static void main(String[] args) throws InterruptedException {
if (args.length == 2) {
NUM_THREAD = Integer.parseInt(args[0]);
NUM_END = Integer.parseInt(args[1]);
}

long startTime = System.currentTimeMillis();
int range = NUM_END / (NUM_THREAD * TASK_SIZE), counter = 0;
List<MultiThread> threads = new ArrayList<>();
```

```java
for (int i = 0; i < NUM_THREAD; i++) {
int start = i * range * TASK_SIZE;
int end = (i == NUM_THREAD - 1) ? NUM_END : start + range * TASK_SIZE;
threads.add(new MultiThread(start, end));
}

for (MultiThread thread : threads) {
thread.start();
}

for (MultiThread thread : threads) {
thread.join();
synchronized (lock) {
counter += thread.getCounter();
System.out.println(thread.getName() + " execution time: " + thread.getExecutionTime() + "ms
#prime counter: " + thread.getCounter());
}
}
long endTime = System.currentTimeMillis();
long timeDiff = endTime - startTime;
System.out.println("Program Execution Time: " + timeDiff + "ms");
}

private static boolean isPrime(int x) {
if (x <= 1) return false;

for (int i = 2; i <= Math.sqrt(x); i++) {
if (x % i == 0) return false;
}
return true;
}

static class MultiThread extends Thread {
private final int start, end;
private int counter;
private long executionTime;

public MultiThread(int start, int end) {
this.start = start;
this.end = end;
}

public int getCounter() { return counter; }
public long getExecutionTime() { return executionTime; }

@Override
public void run() {
long startTime = System.currentTimeMillis();
for (int i = start; i < end; i++) {
if (isPrime(i)) {
synchronized (lock) {
counter++;
}
}
}
long endTime = System.currentTimeMillis();
executionTime = endTime - startTime;
}
}
}
```

1. Main method

- Initializes the number of threads (NUM_THREAD) and number of tasks (NUM_END) variables, and calculates the range variable of the number to be checked by each thread.
- Create NUM_THREAD MultiThread objects and start each thread.
- Starts the created threads, and waits until all threads are finished.

- Adds the number of prime numbers each thread finds, and prints the running time of each thread.
- Outputs the total execution time of the program.

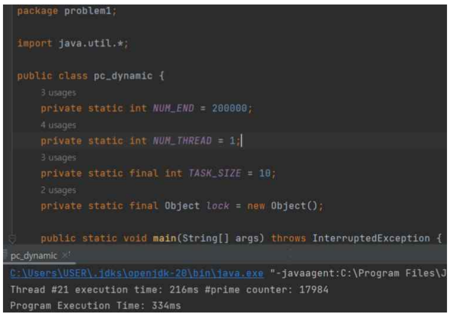
2. isPrime method

- Determines whether an input number is prime or not.
- Determines whether a number is prime by checking if it is divisible by a number from 2 to the square root of that number. Returns true if prime, otherwise returns false.

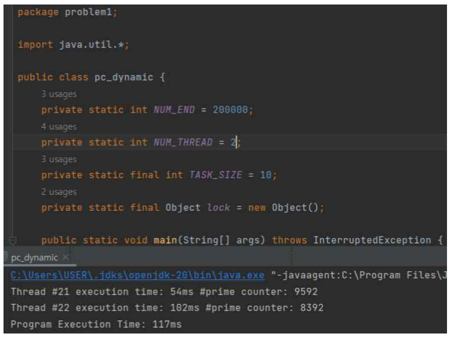
3. MultiThread class

- Each thread calculates a prime number within the allocated range (start, end). When it finds a prime number, it increments the counter variable shared inside the thread. At this time, a synchronized block is used to prevent multiple threads from accessing the counter variable at the same time, and the value is safely increased.
- There is a method to output the execution time of each thread (getCounter) and a method to output the number of prime numbers (getThread_ExecutionTime).

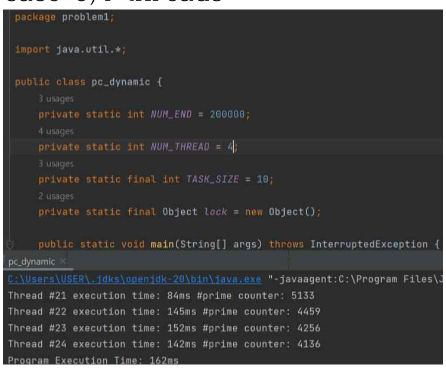
case 1)1 thread



case 2)2 threads



case 3)4 threads



case 4)6 threads

case 5)8 threads

```
package problem1;

import java.util.*;

public class pc_dynamic {
    3 usages
    private static int NUM_END = 200000;
    4 usages
    private static int NUM_THREAD = 8;
    3 usages
    private static final int TASK_SIZE = 10;
    2 usages
    private static final Object lock = new Object();

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

```
pc_dynamic
C:\Users\USER\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\J
Thread #21 execution time: 65ms #prime counter: 2762
Thread #22 execution time: 29ms #prime counter: 2371
Thread #23 execution time: 30ms #prime counter: 2260
Thread #24 execution time: 22ms #prime counter: 2199
Thread #25 execution time: 38ms #prime counter: 2142
Thread #26 execution time: 19ms #prime counter: 2114
Thread #27 execution time: 30ms #prime counter: 2068
Thread #28 execution time: 104ms #prime counter: 2068
Program Execution Time: 189ms
```

case 6)10 threads

```
package problem1;

import java.util.*;

public class pc_dynamic {
    3 usages
    private static int NUM_END = 200000;
    4 usages
    private static int NUM_THREAD = 10;
    3 usages
    private static final int TASK_SIZE = 10;
    2 usages
    private static final Object lock = new Object();

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

```
pc_dynamic
C:\Users\USER\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\J
Thread #21 execution time: 18ms #prime counter: 2262
Thread #22 execution time: 133ms #prime counter: 1941
Thread #23 execution time: 123ms #prime counter: 1854
Thread #24 execution time: 119ms #prime counter: 1780
Thread #25 execution time: 111ms #prime counter: 1755
Thread #26 execution time: 108ms #prime counter: 1709
Thread #27 execution time: 115ms #prime counter: 1709
Thread #28 execution time: 97ms #prime counter: 1673
Thread #29 execution time: 92ms #prime counter: 1659
Thread #30 execution time: 91ms #prime counter: 1642
Program Execution Time: 172ms
```

case 7)12 threads

```
package problem1;

import java.util.*;

public class pc_dynamic {
    3 usages
    private static int NUM_END = 200000;
    4 usages
    private static int NUM_THREAD = 12;
    3 usages
    private static final int TASK_SIZE = 10;
    2 usages
    private static final Object lock = new Object();

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

```
pc_dynamic
C:\Users\USER\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\J
Thread #21 execution time: 12ms #prime counter: 1927
Thread #22 execution time: 72ms #prime counter: 1640
Thread #23 execution time: 68ms #prime counter: 1563
Thread #24 execution time: 68ms #prime counter: 1513
Thread #25 execution time: 60ms #prime counter: 1490
Thread #26 execution time: 66ms #prime counter: 1455
Thread #27 execution time: 62ms #prime counter: 1424
Thread #28 execution time: 57ms #prime counter: 1427
Thread #29 execution time: 51ms #prime counter: 1405
Thread #30 execution time: 45ms #prime counter: 1372
Thread #31 execution time: 47ms #prime counter: 1382
Thread #32 execution time: 45ms #prime counter: 1386
Program Execution Time: 86ms
```

case 8)14 threads

```
package problem1;

import java.util.*;

public class pc_dynamic {
    3 usages
    private static int NUM_END = 200000;
    4 usages
    private static int NUM_THREAD = 14;
    3 usages
    private static final int TASK_SIZE = 10;
    2 usages
    private static final Object lock = new Object();

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

```
pc_dynamic
Thread #21 execution time: 12ms #prime counter: 1675
Thread #22 execution time: 87ms #prime counter: 1432
Thread #23 execution time: 85ms #prime counter: 1372
Thread #24 execution time: 78ms #prime counter: 1313
Thread #25 execution time: 78ms #prime counter: 1279
Thread #26 execution time: 76ms #prime counter: 1268
Thread #27 execution time: 71ms #prime counter: 1249
Thread #28 execution time: 70ms #prime counter: 1223
Thread #29 execution time: 69ms #prime counter: 1223
Thread #30 execution time: 60ms #prime counter: 1215
Thread #31 execution time: 60ms #prime counter: 1192
Thread #32 execution time: 49ms #prime counter: 1171
Thread #33 execution time: 49ms #prime counter: 1187
Thread #34 execution time: 49ms #prime counter: 1185
Program Execution Time: 105ms
```

case 9)16 threads

```
package problem1;

import java.util.*;

public class pc_dynamic {
    3 usages
    private static int NUM_END = 200000;
    4 usages
    private static int NUM_THREAD = 16;
    3 usages
    private static final int TASK_SIZE = 10;
    2 usages
```

```
pc_dynamic
Thread #21 execution time: 13ms #prime counter: 1492
Thread #22 execution time: 103ms #prime counter: 1270
Thread #23 execution time: 97ms #prime counter: 1206
Thread #24 execution time: 94ms #prime counter: 1165
Thread #25 execution time: 88ms #prime counter: 1142
Thread #26 execution time: 88ms #prime counter: 1118
Thread #27 execution time: 86ms #prime counter: 1099
Thread #28 execution time: 78ms #prime counter: 1100
Thread #29 execution time: 78ms #prime counter: 1070
Thread #30 execution time: 63ms #prime counter: 1072
Thread #31 execution time: 56ms #prime counter: 1068
Thread #32 execution time: 53ms #prime counter: 1046
Thread #33 execution time: 54ms #prime counter: 1037
Thread #34 execution time: 43ms #prime counter: 1031
Thread #35 execution time: 43ms #prime counter: 1048
Thread #36 execution time: 41ms #prime counter: 1020
Program Execution Time: 184ms
```

case 10)32 threads

```
import java.util.*;

public class pc_dynamic {
    3 usages
    private static int NUM_END = 200000;
    4 usages
    private static int NUM_THREAD = 32;
    3 usages
    private static final int TASK_SIZE = 10;
    2 usages
    private static final Object lock = new Object();
```

```
pc_dynamic
Thread #21 execution time: 6ms #prime counter: 812
Thread #22 execution time: 4ms #prime counter: 680
Thread #23 execution time: 3ms #prime counter: 648
Thread #24 execution time: 76ms #prime counter: 622
Thread #25 execution time: 77ms #prime counter: 606
Thread #26 execution time: 71ms #prime counter: 600
Thread #27 execution time: 71ms #prime counter: 587
Thread #28 execution time: 71ms #prime counter: 578
Thread #29 execution time: 67ms #prime counter: 573
Thread #30 execution time: 62ms #prime counter: 569
Thread #31 execution time: 62ms #prime counter: 559
Thread #32 execution time: 57ms #prime counter: 559
Thread #33 execution time: 56ms #prime counter: 559
Thread #34 execution time: 57ms #prime counter: 540
Thread #35 execution time: 50ms #prime counter: 555
Thread #36 execution time: 51ms #prime counter: 545
Thread #37 execution time: 52ms #prime counter: 533
Thread #38 execution time: 46ms #prime counter: 537
```

```
pc_dynamic
Thread #39 execution time: 45ms #prime counter: 532
Thread #40 execution time: 43ms #prime counter: 540
Thread #41 execution time: 42ms #prime counter: 529
Thread #42 execution time: 37ms #prime counter: 539
Thread #43 execution time: 39ms #prime counter: 522
Thread #44 execution time: 36ms #prime counter: 524
Thread #45 execution time: 33ms #prime counter: 528
Thread #46 execution time: 32ms #prime counter: 509
Thread #47 execution time: 31ms #prime counter: 509
Thread #48 execution time: 26ms #prime counter: 522
Thread #49 execution time: 27ms #prime counter: 517
Thread #50 execution time: 26ms #prime counter: 531
Thread #51 execution time: 68ms #prime counter: 510
Thread #52 execution time: 70ms #prime counter: 510
Program Execution Time: 107ms
```