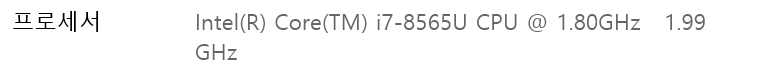
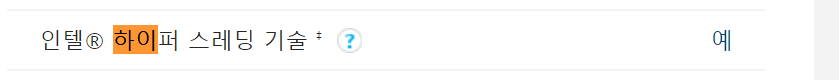
1. Environment

Intel® Core™ i7-8565U CPU @ 1.80GHz 1.99Ghz

RAM:16GB

Quadcore



Tables and graphs

Below graphs show comparison between static load balancing and dynamic load balancing.

Below graphs show comparison between static load balancing and dynamic load balancing.

Below tables show comparison between static load balancing and dynamic load balancing

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exec time | 1Thread | 2Thread | 4Thread | 6Thread | 8Thread | 10Thread | 12Thread | 14Thread | 16Thread |
| static | 3874 | 3044 | 2137 | 1622 | 1420 | 1186 | 1045 | 1048 | 1139 |
| dynamic | 4009 | 2091 | 1315 | 1109 | 1046 | 1033 | 1015 | 1000 | 1015 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Performance (1/exec time) | 1Thread | 2Thread | 4Thread | 6Thread | 8Thread | 10Thread | 12Thread | 14Thread | 16Thread |
| static | 0.00026 | 0.00033 | 0.00047 | 0.00062 | 0.00070 | 0.00084 | 0.00096 | 0.00096 | 0.00088 |
| dynamic | 0.000249 | 0.000478 | 0.00076 | 0.000902 | 0.000956 | 0.000968 | 0.000985 | 0.001 | 0.000985 |

1. Explanation of the results

Static and dynamic load balancing have similarity that as more threads we have, the better.

However, overhead of thread creating and thread managing was increased if too many threads were created. So, it has the best performance with 14 threads both static and dynamic load balancing and if with 16 threads, the thread management overhead became large, so the speed became slow and performance became bad.

Another no increase reason is due to Amdahl’s law. Even if the more threads we have, important factor of increasing performance is how much we can parallelizable. The reason I got similar performance between 8 and 16 threads is limitation of parallelizable part in computation code.

1. JAVA source code and screen capture image

(d-1) pc\_dynamic.java code

**public** **class** pc\_dynamic {

**private** **static** **final** **int** ***NUM\_END*** = 200000;

**private** **static** **final** **int** ***NUM\_THREAD*** = 16;

**public** **static** **void** main(String[] args) {

**int** counter=0;

**int** i=0;

**int** cnt = 0;

Num n = **new** Num();

IntThread[] ts = **new** IntThread[***NUM\_THREAD***];

**long** startTime = System.*currentTimeMillis*();

**for** (i=0;i<***NUM\_THREAD***;i++) {

ts[i] = **new** IntThread(i, cnt, n);

ts[i].start();

}

**try** {

**for**(i=0;i<***NUM\_THREAD***;i++) {

ts[i].join();

counter += ts[i].getSum();

}

}**catch**(InterruptedException e) {}

**long** endTime = System.*currentTimeMillis*();

**long** timeDiff = endTime - startTime;

System.***out***.println("\nTotal Execution Time : "+timeDiff+"ms");

System.***out***.println("1..."+(***NUM\_END***-1)+" prime# counter=" + counter +"\n");

}

}

// shared Num class

**class** Num{

**private** **static** **final** **int** ***SEQUENTIAL\_CUTOFF*** = 5000;

**int** i = -1;

**synchronized** **int** ranges() {

i++;

**return** i\****SEQUENTIAL\_CUTOFF***;

}

**public** **int** getCutoff() { **return** ***SEQUENTIAL\_CUTOFF***;}

}

**class** IntThread **extends** Thread {

**int** my\_id;

**int** num\_start;

**int** num\_end;

**int** counter=0;

**int** c\_thread;

Num n = **new** Num();

IntThread(**int** id, **int** cnt, Num n) {

my\_id = id; c\_thread = cnt;

**this**.n =n;

}

**public** **void** run() {

**long** startTime = System.*currentTimeMillis*();

**int** i;

**while**((num\_start = n.ranges()) < 200000 ) {

num\_end = num\_start + n.getCutoff();

**for** (i=num\_start;i<num\_end;i++) {

**if** (*isPrime*(i)) counter++;

}

}

**long** endTime = System.*currentTimeMillis*();

**long** timeDiff = endTime - startTime;

System.***out***.println("Thread"+my\_id+"'s Execution Time : "+timeDiff+"ms");

}

**private** **static** **boolean** isPrime(**int** x) {

**int** i;

**if** (x<=1) **return** **false**;

**for** (i=2;i<x;i++) {

**if** ((x%i == 0) && (i!=x)) **return** **false**;

}

**return** **true**;

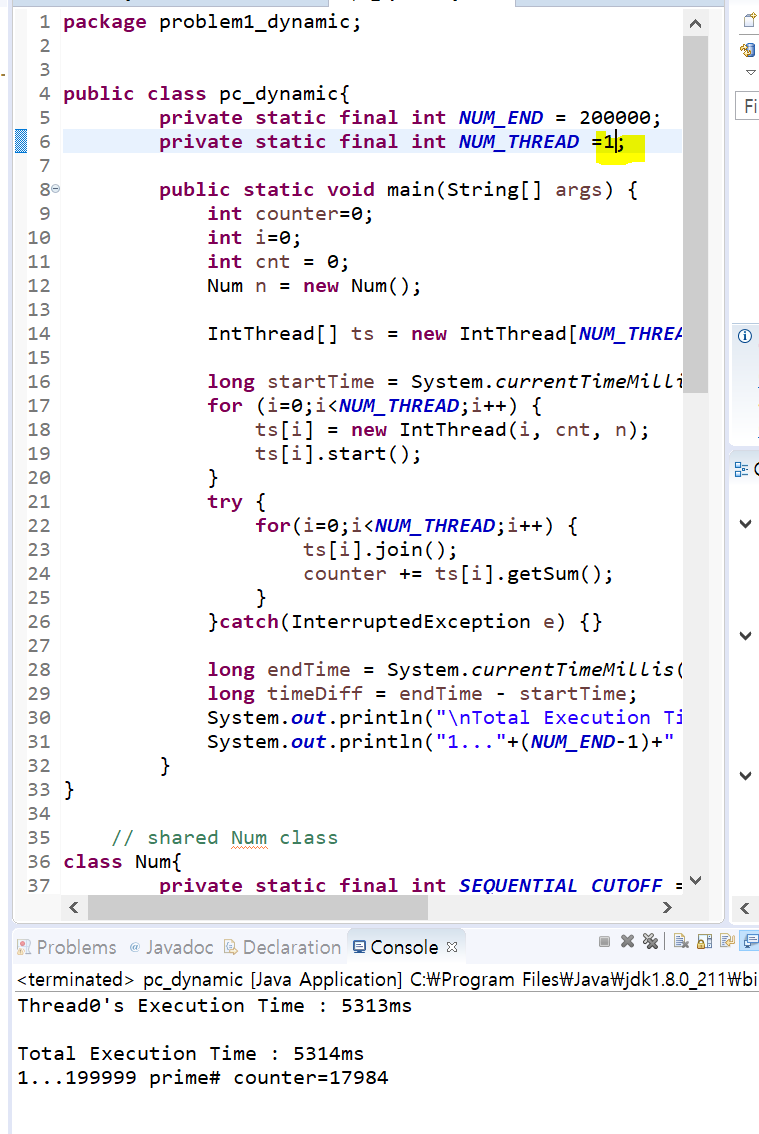
}

**public** **int** getSum() {**return** counter;}

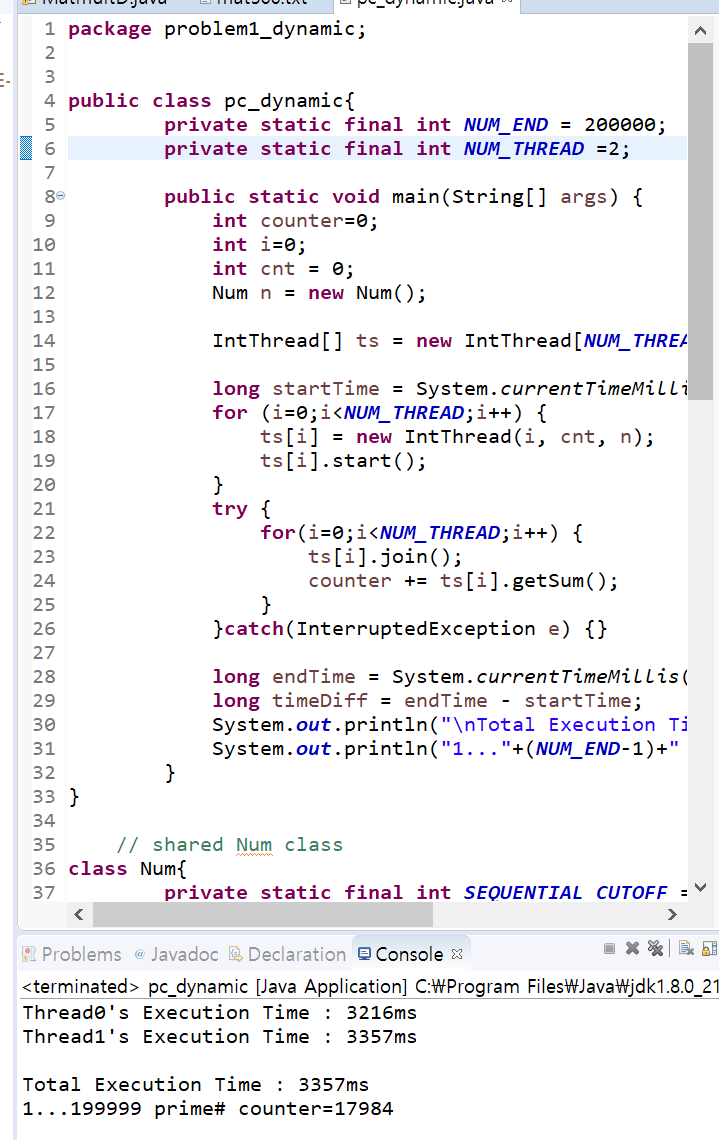
}

(d-2) screen capture image of dynamic load balancing

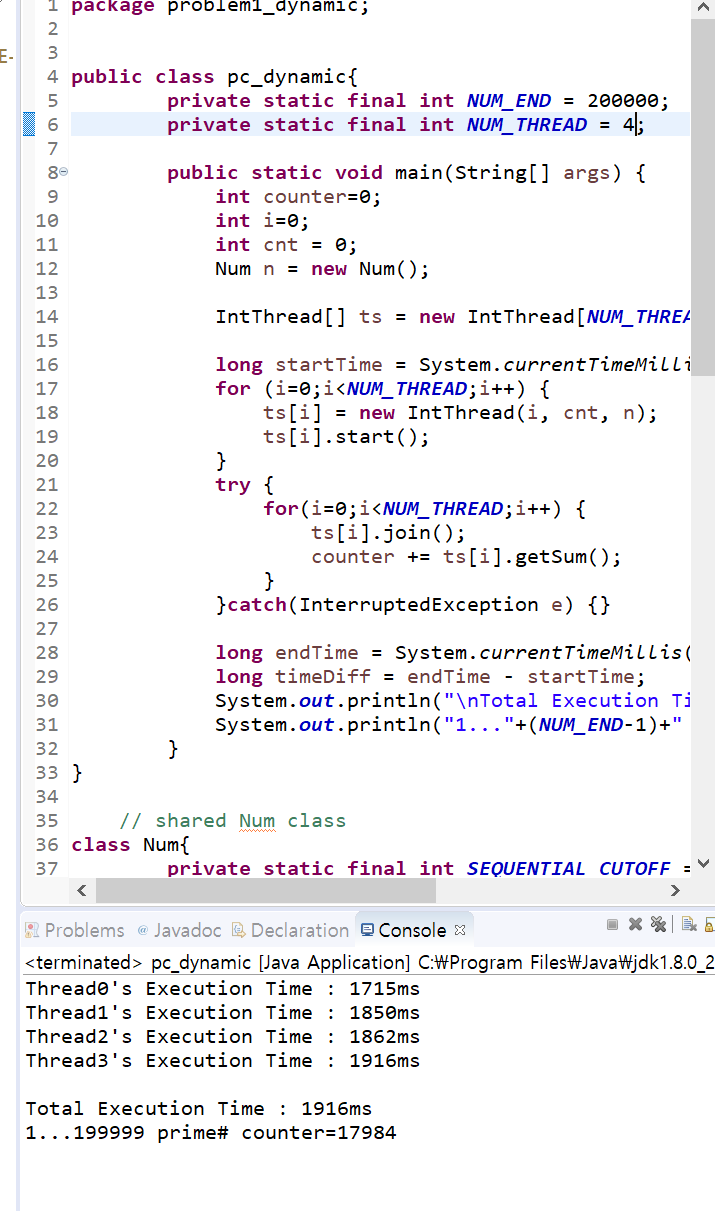
1 thread



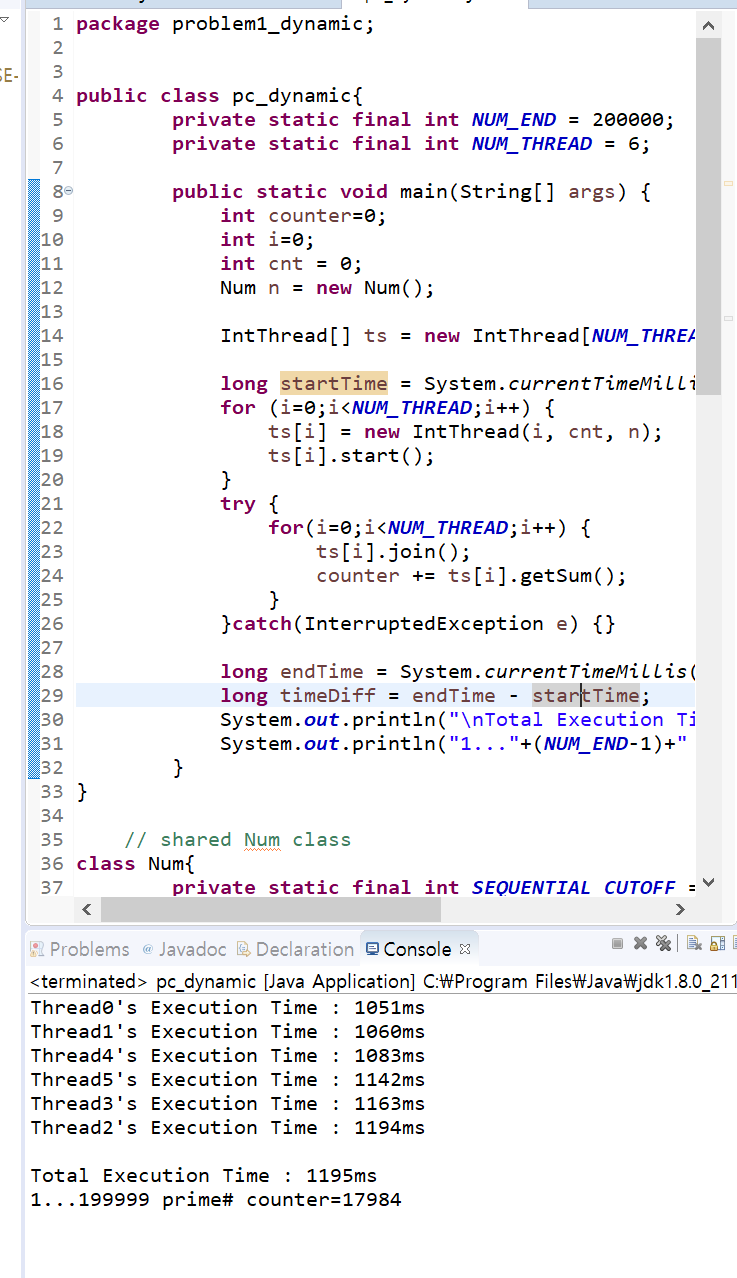
2 threads



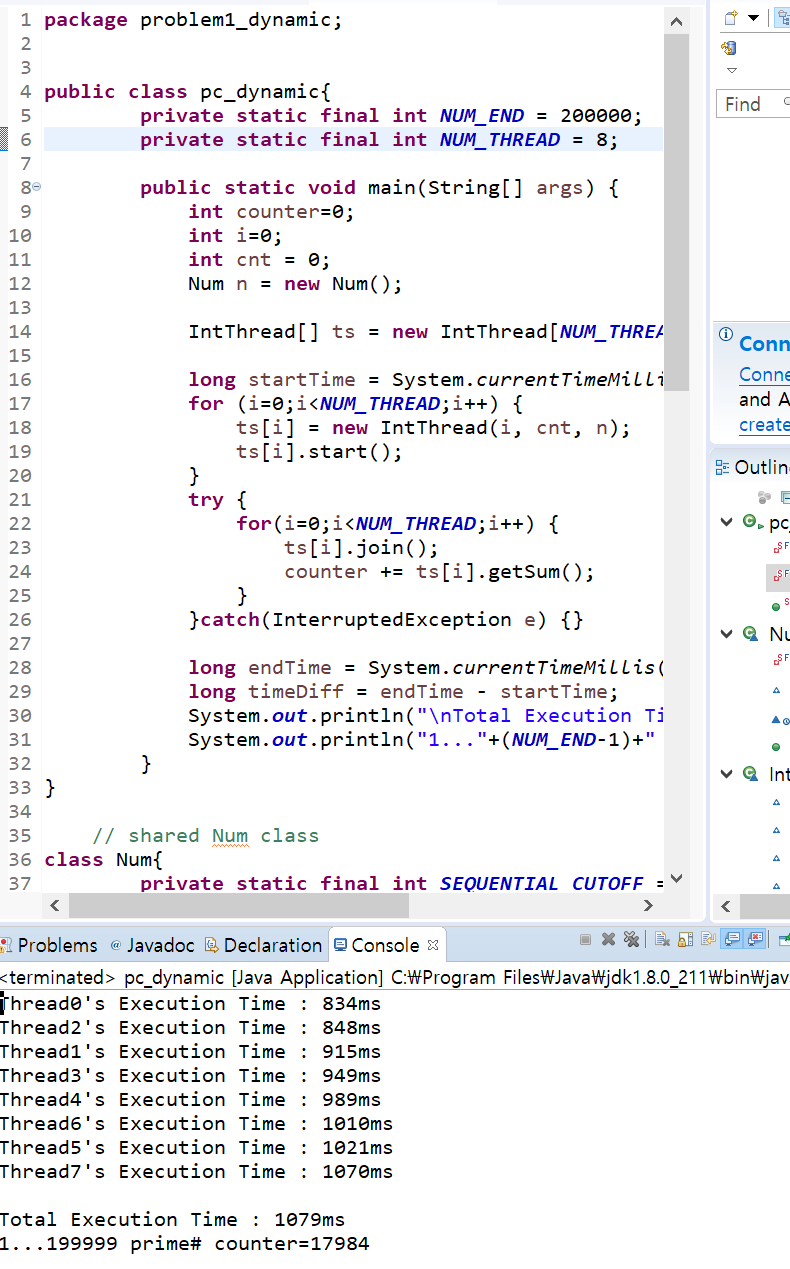
4 threads



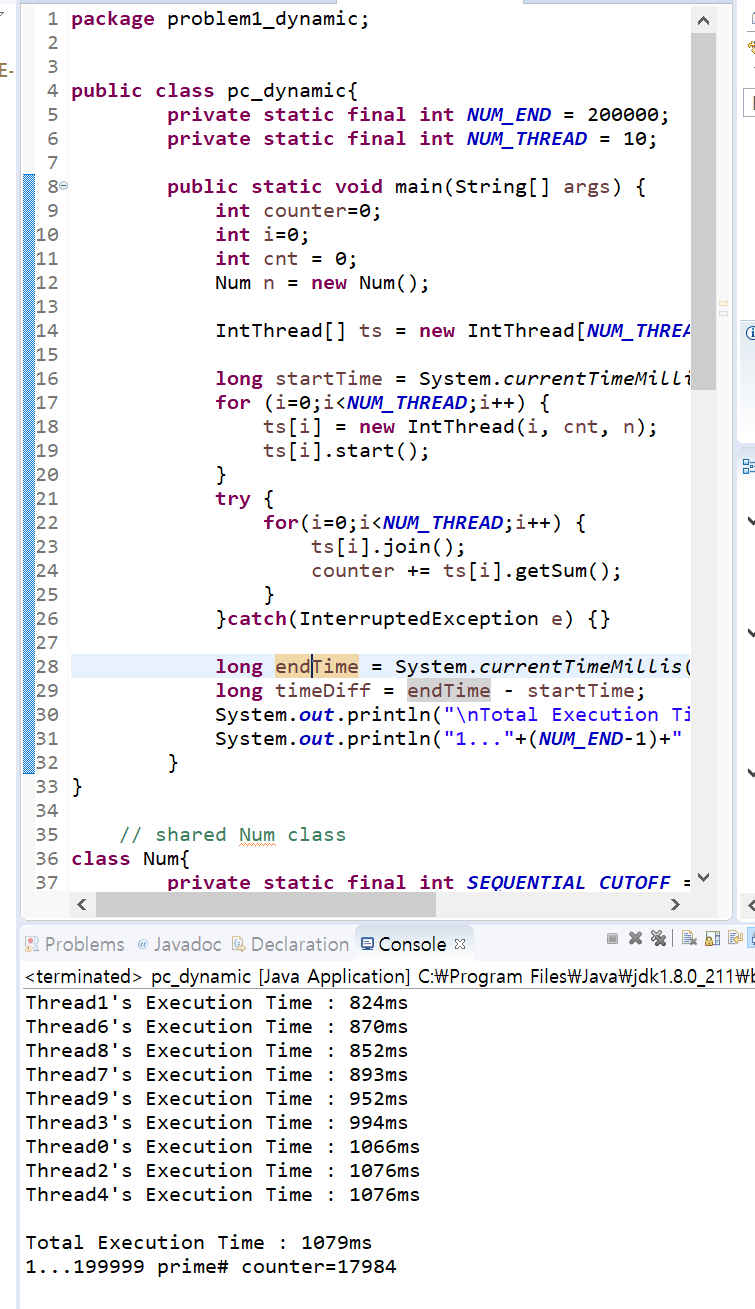
6 threads



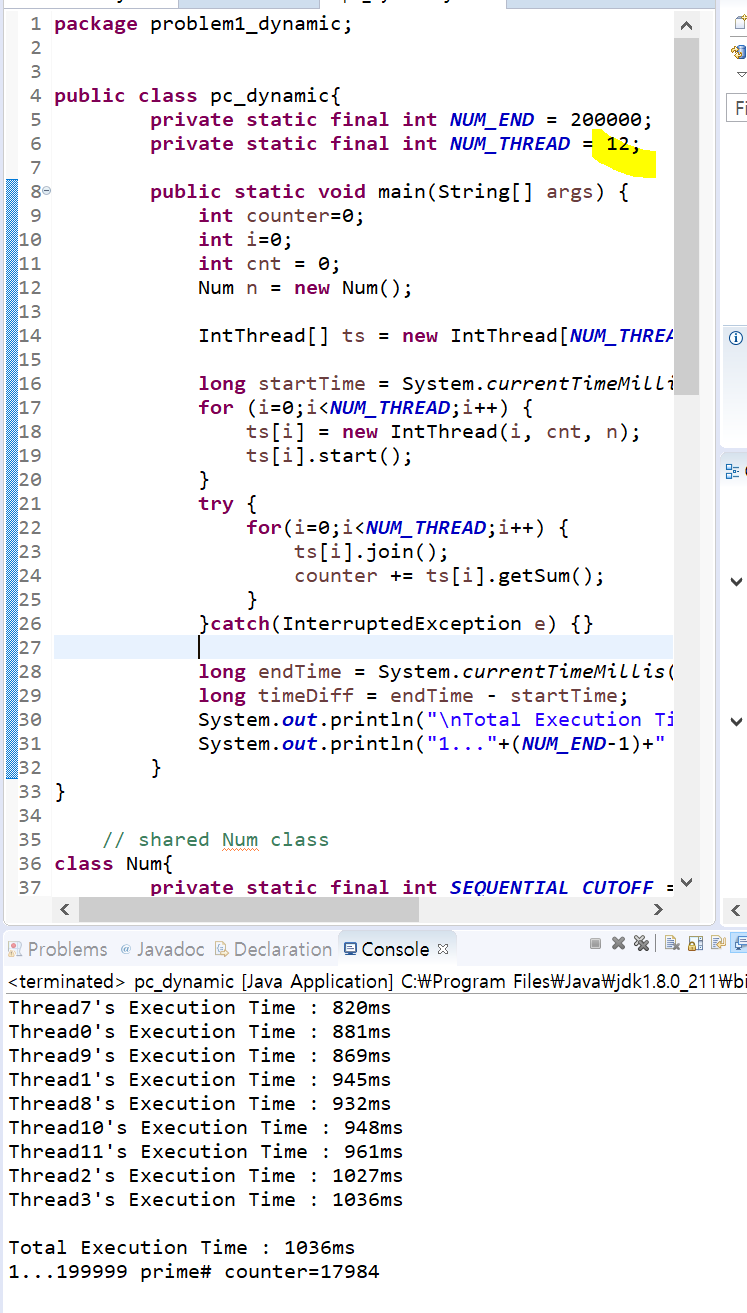
8 threads



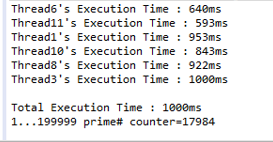
10threads



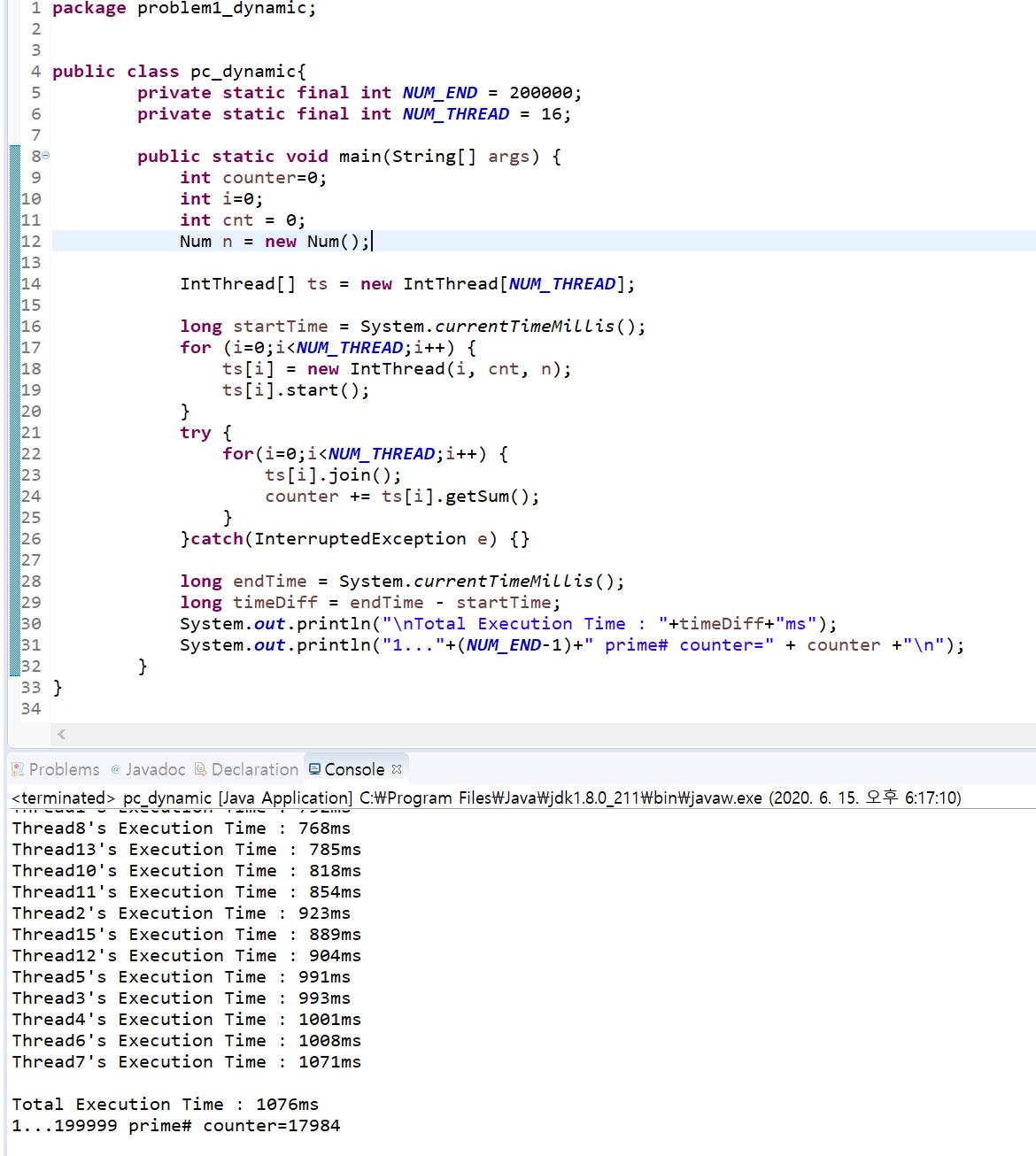
12 threads



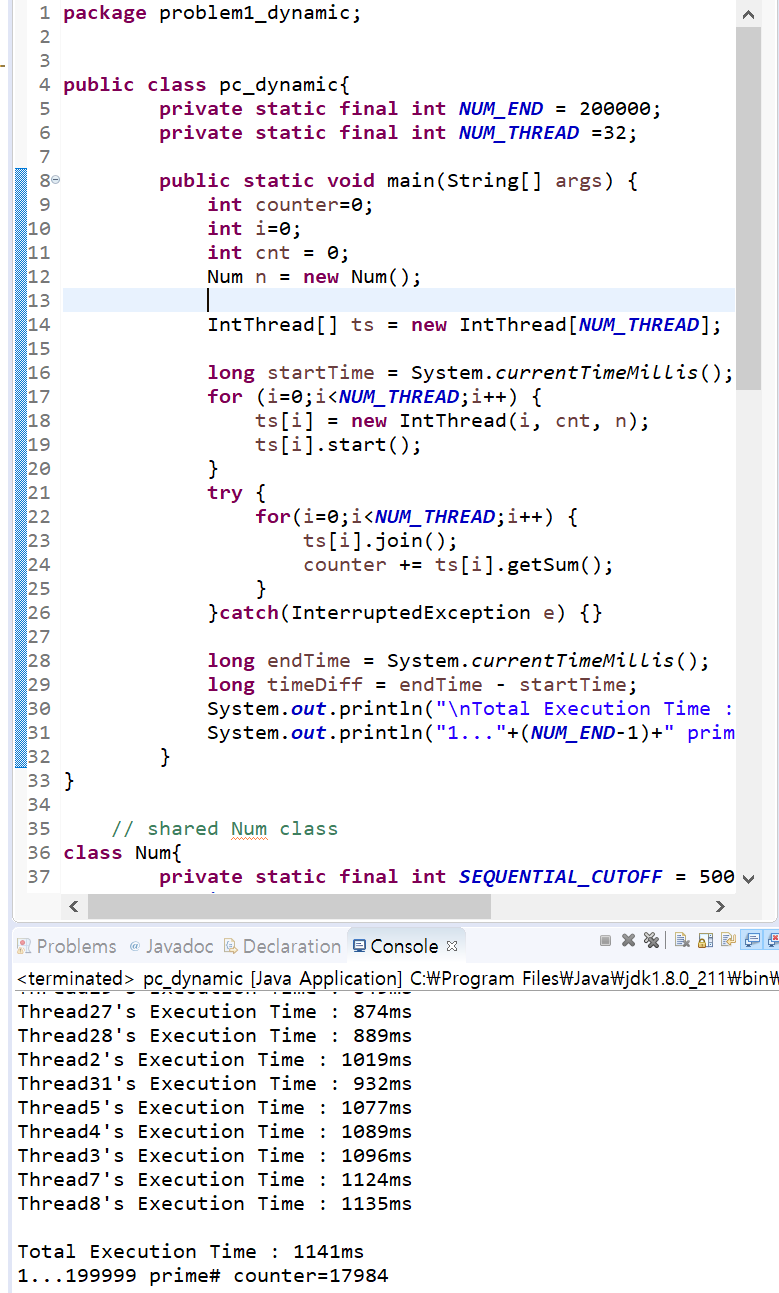
14 threads



16 threads



32 threads



(d-3) pc\_static.java

**package** problem1\_static;

**public** **class** pc\_static {

**private** **static** **final** **int** ***NUM\_END*** = 200000;

**private** **static** **final** **int** ***NUM\_THREAD*** = 8;

**public** **static** **void** main(String[] args) {

**int** counter = 0;

**int** i;

IntThread[] ts = **new** IntThread[***NUM\_THREAD***];

**long** startTime = System.*currentTimeMillis*();

**for** (i = 0; i < ***NUM\_THREAD***; i++) {

ts[i] = **new** IntThread(i, ***NUM\_END***, ***NUM\_THREAD***);

ts[i].start();

}

**try** {

**for** (i = 0; i < ***NUM\_THREAD***; i++) {

ts[i].join();

counter += ts[i].getSum();

}

} **catch** (InterruptedException e) {

}

**long** endTime = System.*currentTimeMillis*();

**long** timeDiff = endTime - startTime;

System.***out***.println("Execution Time : " + timeDiff + "ms");

System.***out***.println("1..." + (***NUM\_END*** - 1) + " prime# counter=" + counter + "\n");

}

}

**class** IntThread **extends** Thread {

**int** my\_id; // fields for communicating inputs

**int** num\_steps;

**int** num\_threads;

**int** counter = 0;

IntThread(**int** id, **int** numSteps, **int** numThreads) {

my\_id = id;

num\_steps = numSteps;

num\_threads = numThreads;

}

**public** **void** run() {

**long** startTime = System.*currentTimeMillis*();

**int** i;

**int** i\_start = my\_id \* (**int**) (num\_steps / num\_threads);

**int** i\_end = i\_start + (**int**) (num\_steps / num\_threads);

**if** (my\_id != num\_threads - 1) {

**for** (i = i\_start; i < i\_end; i++) {

**if** (*isPrime*(i))

counter++;

}

} **else** {

**for** (i = i\_start; i < num\_steps; i++) {

**if** (*isPrime*(i))

counter++;

}

}

**long** endTime = System.*currentTimeMillis*();

**long** timeDiff = endTime - startTime;

System.***out***.println("Thread" + my\_id + "'s Execution Time : " + timeDiff + "ms");

}

**private** **static** **boolean** isPrime(**int** x) {

**int** i;

**if** (x <= 1)

**return** **false**;

**for** (i = 2; i < x; i++) {

**if** ((x % i == 0) && (i != x))

**return** **false**;

}

**return** **true**;

}

**public** **int** getSum() {

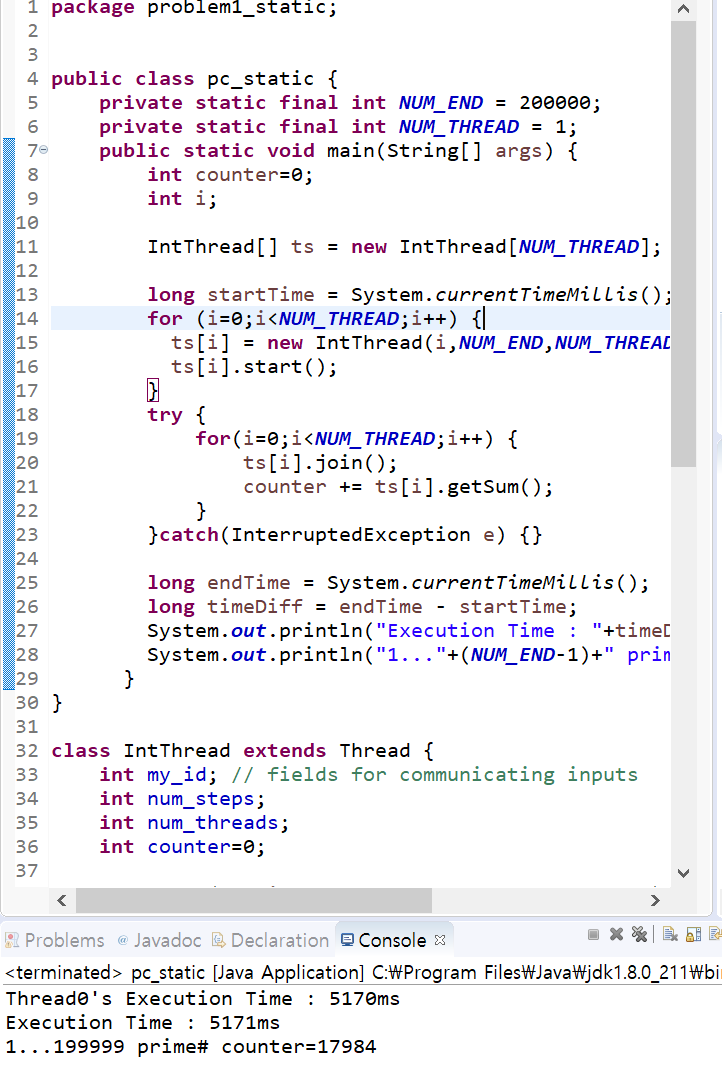
**return** counter;

}

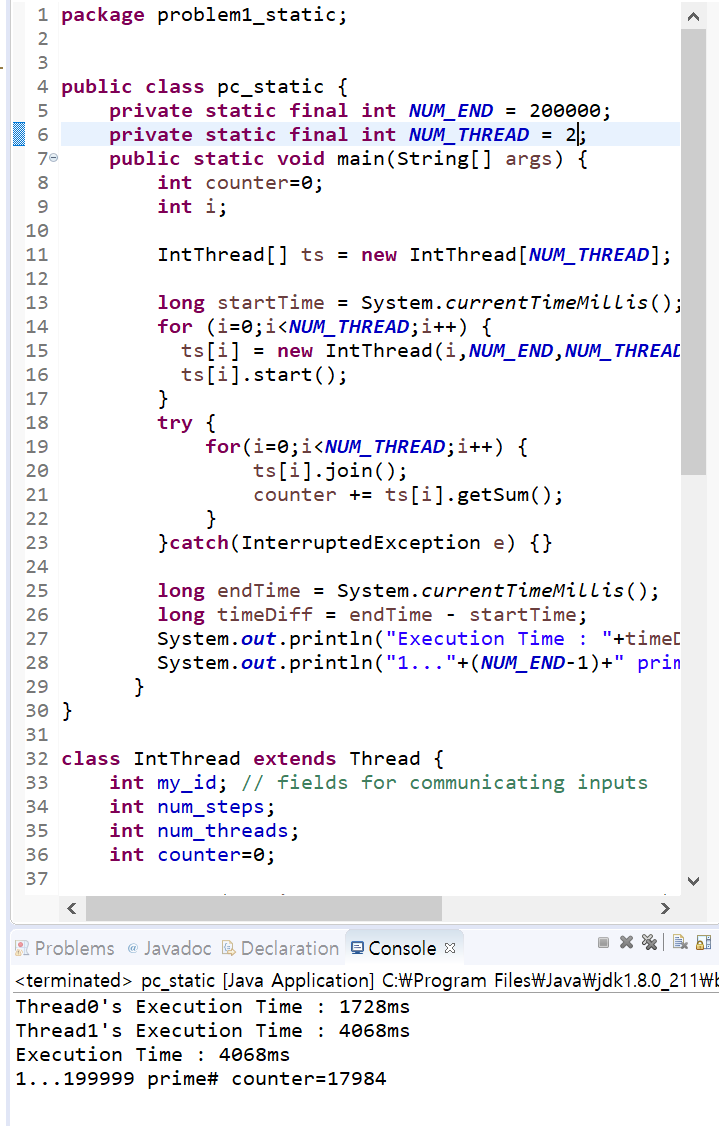
}

(d-4) screen capture image of static load balancing

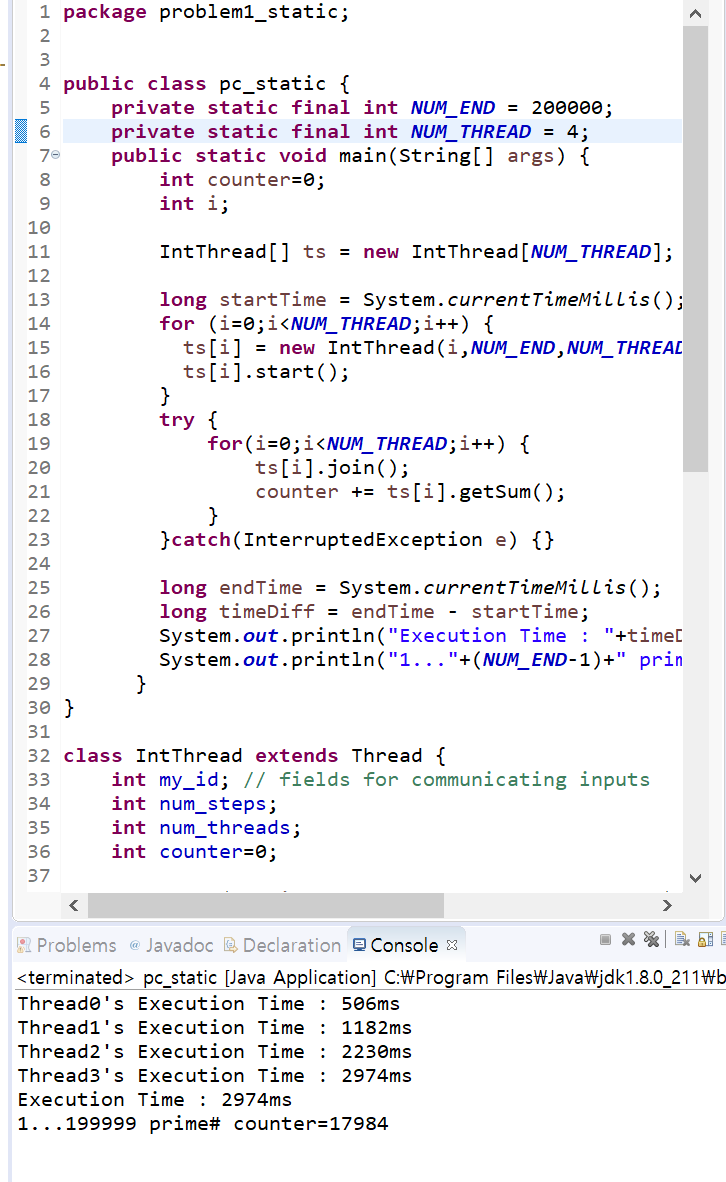
1 thread



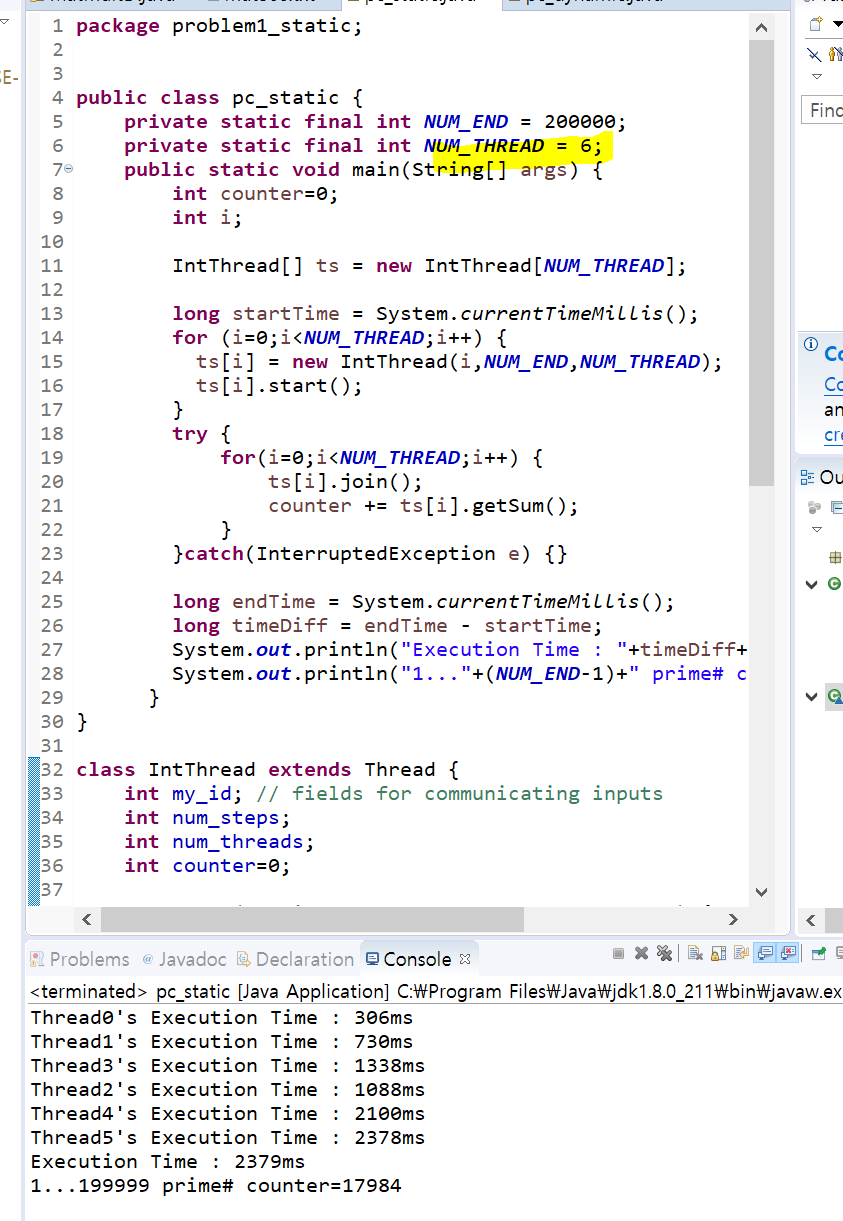
2 threads



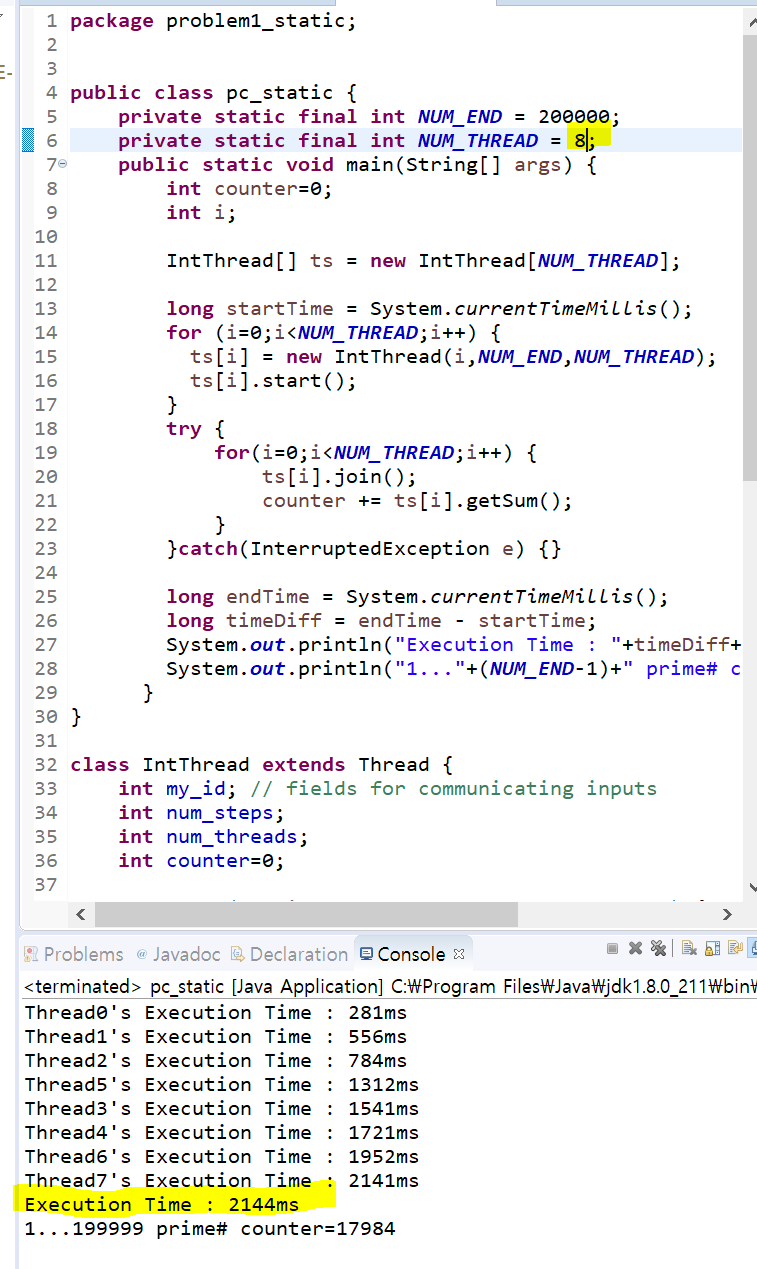
4 threads



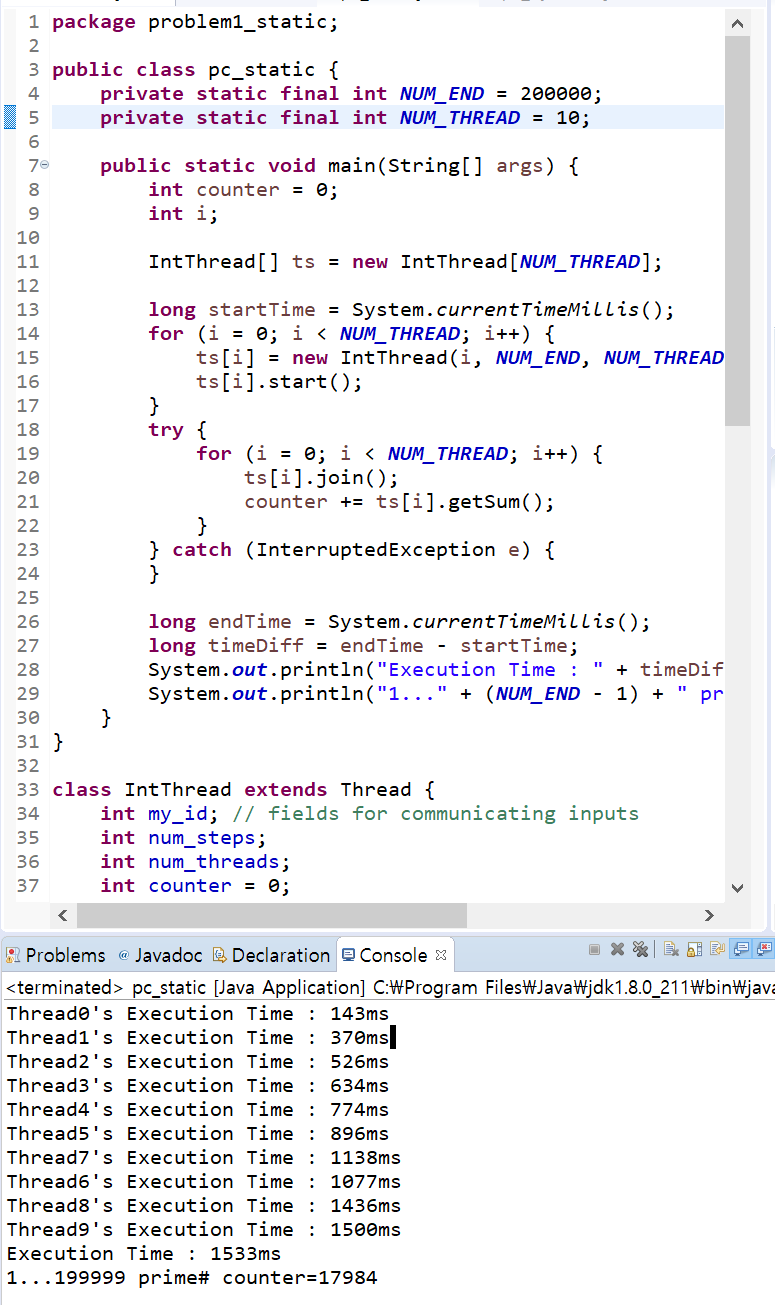
6 threads



8 threads



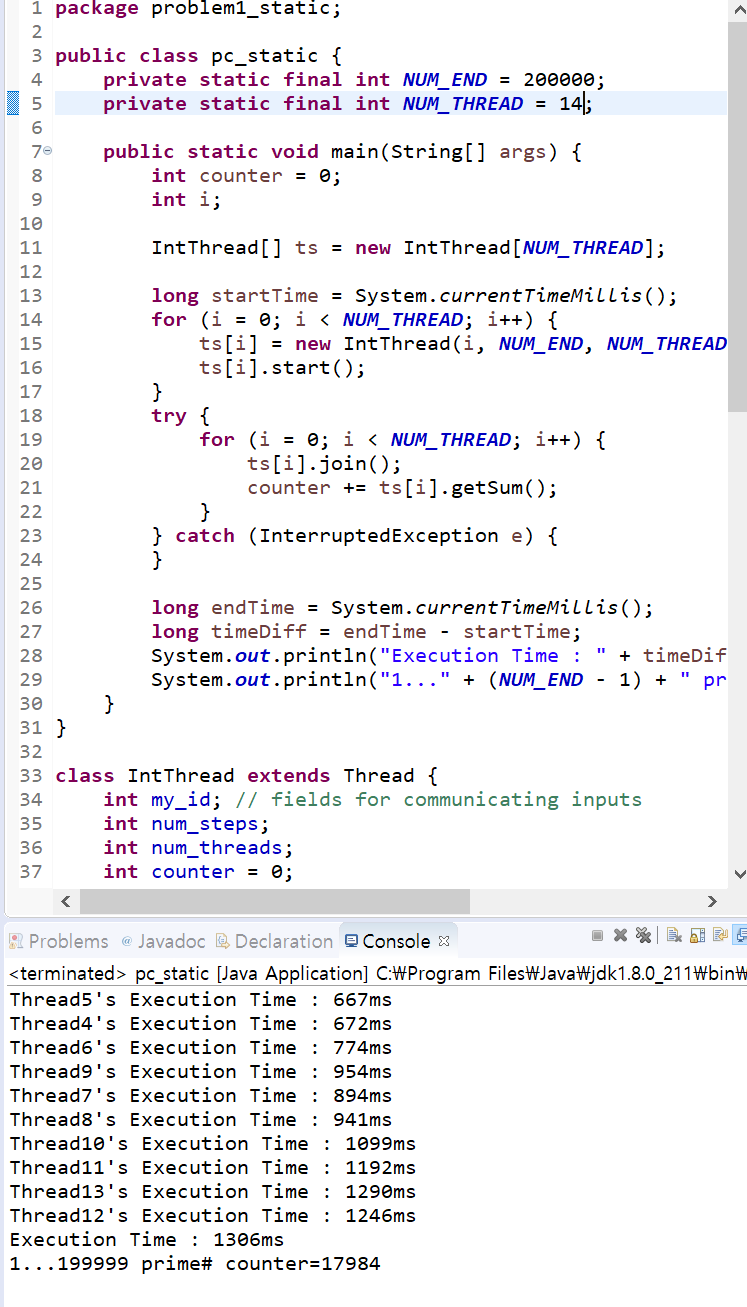
10 threads



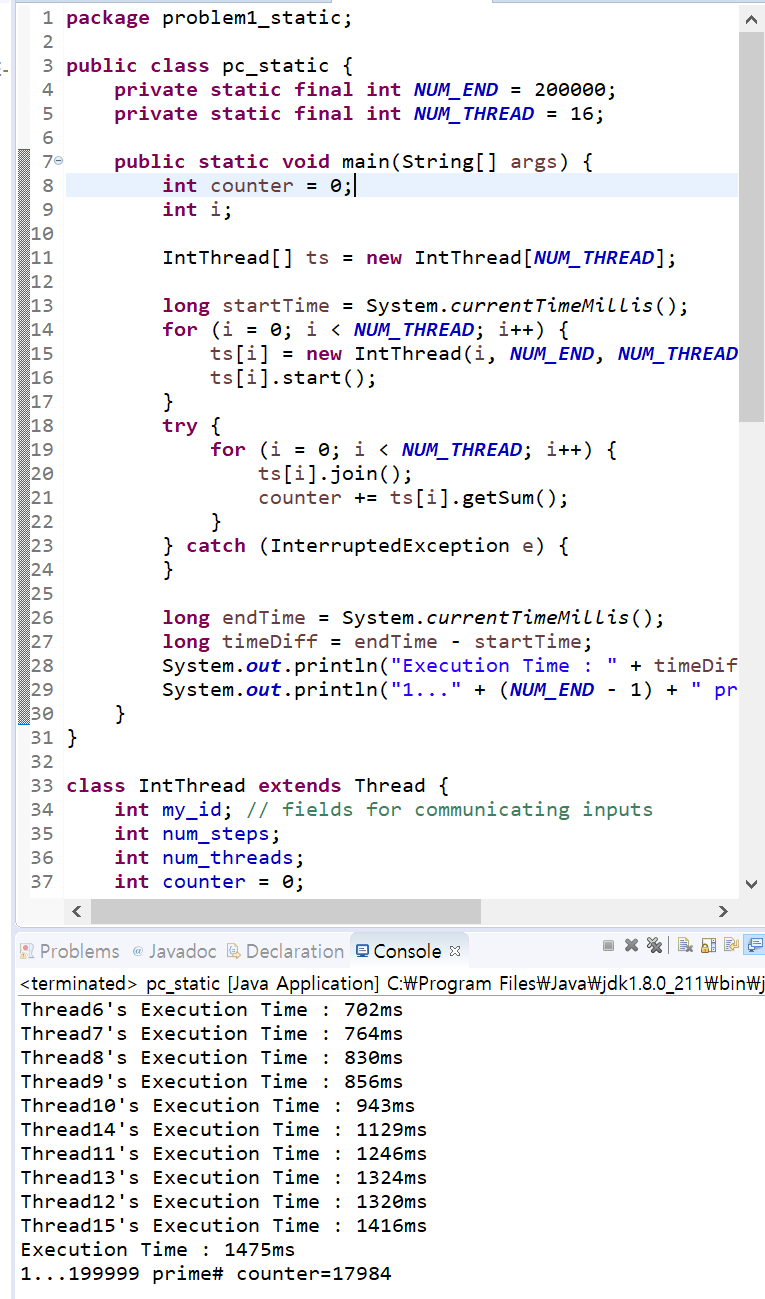
12 threads



14 threads



16 threads



32 threads