

과제 1

프로그램 진행시간 측정

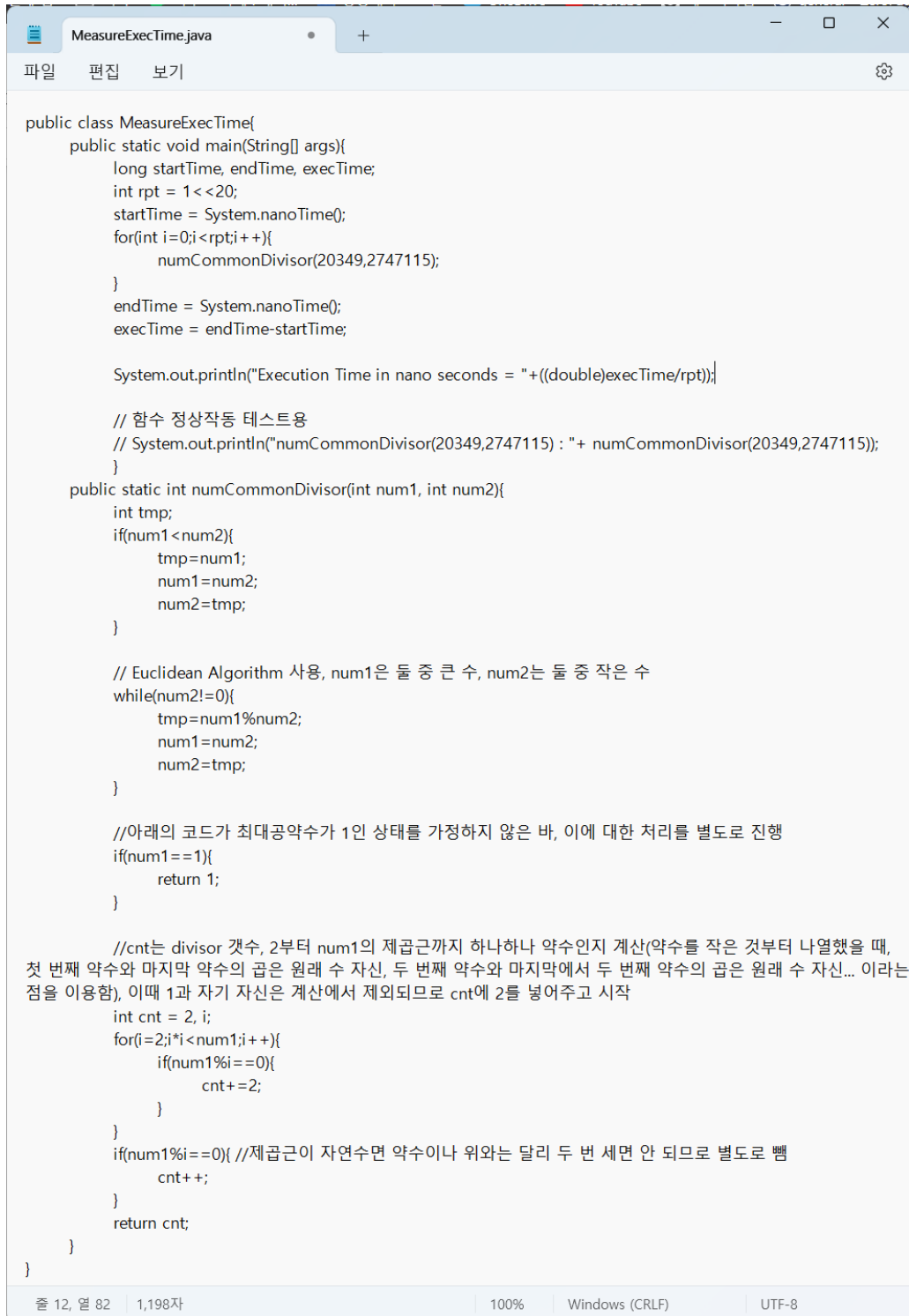
20232907 정현승

목차

1. 과제(메모장) 수행 방법	3
2. 과제(메모장) 수행 과정 및 결과	5
A. 메모장에 코드 작성	5
B. 코드 컴파일 및 실행	6
3. 과제(Eclipse) 수행 방법	9
4. 과제(Eclipse) 수행 증명 및 결과	10
A. 코드 작성 및 확인 과정	10
B. 실행 결과 확인	13
5. 소스코드	14
붙임. 자체평가표	16

1. 과제(메모장) 수행 방법

우선 메모장을 키고 코드를 작성하였다. 작성한 코드는 아래 5문단에도 기재되어 있다.



```
public class MeasureExecTime{
    public static void main(String[] args){
        long startTime, endTime, execTime;
        int rpt = 1<<20;
        startTime = System.nanoTime();
        for(int i=0;i<rpt;i++){
            numCommonDivisor(20349,2747115);
        }
        endTime = System.nanoTime();
        execTime = endTime-startTime;

        System.out.println("Execution Time in nano seconds = "+((double)execTime/rpt));

        // 함수 정상작동 테스트용
        // System.out.println("numCommonDivisor(20349,2747115) : "+ numCommonDivisor(20349,2747115));
    }
    public static int numCommonDivisor(int num1, int num2){
        int tmp;
        if(num1<num2){
            tmp=num1;
            num1=num2;
            num2=tmp;
        }

        // Euclidean Algorithm 사용, num1은 둘 중 큰 수, num2는 둘 중 작은 수
        while(num2!=0){
            tmp=num1%num2;
            num1=num2;
            num2=tmp;
        }

        //아래의 코드가 최대공약수가 1인 상태를 가정하지 않은 바, 이에 대한 처리를 별도로 진행
        if(num1==1){
            return 1;
        }

        //cnt는 divisor 갯수, 2부터 num1의 제곱근까지 하나하나 약수인지 계산(약수를 작은 것부터 나열했을 때,
        첫 번째 약수와 마지막 약수의 곱은 원래 수 자신, 두 번째 약수와 마지막에서 두 번째 약수의 곱은 원래 수 자신... 이라는
        점을 이용함), 이때 1과 자기 자신은 계산에서 제외되므로 cnt에 2를 넣어주고 시작
        int cnt = 2, i;
        for(i=2;i*i<num1;i++){
            if(num1%i==0){
                cnt+=2;
            }
        }
        if(num1%i==0){ //제곱근이 자연수면 약수이나 위와는 달리 두 번 세면 안 되므로 별도로 뺌
            cnt++;
        }
        return cnt;
    }
}
```

그림 1

위와 같이 코드를 작성했다. “함수 정상작동 테스트용”이라 적힌 부분은 테스트 시 추가하였다.

시간 측정을 위한 반복 횟수는 $1 < 20$ (1048576) 번으로 하였다.

공약수의 개수를 구하는 것은 **최대공약수를 구한 후, 그 수의 약수 개수를 구하는 방법**으로 진행하였다. 1부터 하나씩 공약수인지 찾는 방법도 가능하나, 그렇게 하면 **시간이 너무 오래 걸릴 것으로 생각하여, 지금까지 배웠던 것을 활용**했다. 최대공약수를 구하는 과정은 이산수학 시간에 배웠던 **Euclidean Algorithm**을 활용했다.



Euclidean Algorithm

- Need more efficient prime factorization algorithm to find GCD.
- Example: find $\gcd(91, 287)$
 - $287 = 91 \cdot 3 + 14$
 - Any divisor of 287 and 91 must be a divisor of $287 - 91 \cdot 3 = 14$.
 - Any divisor of 91 and 14 must also be a divisor of $287 = 91 \cdot 3 + 14$.
 - Hence, $\gcd(91, 287) = \gcd(91, 14)$.
 - Next, $91 = 14 \cdot 6 + 7$
 - Any divisor of 91 and 14 also divides $91 - 14 \cdot 6 = 7$.
 - Any divisor of 14 and 7 divides 91, i.e., $\gcd(91, 14) = \gcd(14, 7)$.
 - $14 = 7 \cdot 2$
 - $\gcd(14, 7) = 7$.
 - Thus, $\gcd(287, 91) = 7$.

Prof. Eunwoo Kim (Chung-Ang University)

55

그림 2

위 그림은 Euclidean Algorithm을 설명하는 그림으로, 그 방법을 간략히 설명하면 다음과 같다. 큰 수를 작은 수로 나눈 나머지를 구한다. 이후 작은 수와 나머지로 같은 연산을 계속 반복한다. 반복하다 나머지가 0이 나왔을 때의 작은 수가 두 수의 최대공약수이다.¹

이 최대공약수의 약수 개수는 약수를 작은 것부터 나열했을 때, 첫 번째 약수와 마지막 약수의 곱은 원래 수 자신, 두 번째 약수와 마지막에서 두 번째 약수의 곱은 원래 수 자신이고, 이게 세 번째 이상으로 계속 가도 두 수의 곱이 원래 수 자신이라는 건 변하지 않는다는 점을 이용하였다. 예를 들어 24의 약수는 1, 2, 3, 4, 6, 8, 12, 24인데, 첫 번째 약수 1과 마지막 약수 24의 곱은 24이고, 두 번째 약수 2와 마지막에서 두 번째 약수 12의 곱도 24이다. 같은 방법으로 3과 8의 곱도, 4와 6의 곱도 24이다. 이 점을 이용해 **1부터 그 숫자의 제곱근까지만 약수인지 확인**하면 전체 약수의 개수를 구할 수 있다. 예를 들어 24의 제곱근은 4보다 크고 5보다 작다. 1부터 4까지의 숫자 중 24의 약수는 4개인데, 각 약수에 자기 자신과 곱하여 24가 되는 약수(즉 짝이 되는 약수로, 1은 24, 2는 12, 3은 8, 4는 6)가 각 1개씩 총 4개가 더 존재한다는 것을 알 수 있다. 따라서 어떤

¹ 김은우, Discrete_Math_w9_alg_number_theory, 이산수학(영어A강의) 03분반, 2023, p.55

수의 약수 개수는 1부터 그 수의 제곱근까지만 약수인지 확인해도 그 수의 약수 개수를 알 수 있다. 여기에 1을 제외한 모든 수는 1과 자기 자신을 약수로 가지므로 **1은 건너뛰고 2부터 약수인지 확인**하였으며, 최대공약수가 1이면 이 확인 과정을 모두 건너뛰고 1개를 반환하도록 했다. 또 어떤 숫자의 제곱근이 정수이면, 그 수도 약수이긴 하나 짝을 이루는 수가 자기 자신이므로, 약수 개수를 2개가 아닌 1개로 세도록 했다. (예를 들어 16의 약수는 1, 2, 4, 8, 16이다. 1과 16, 2와 8은 짝이 되어 서로 곱하면 원래의 수인 16이 되지만 4는 4 자기 자신과 곱해야 16이 된다. 1이나 2를 셀 때는 16과 8을 별도로 세지 않고 한 번 셀 때마다 약수가 2개씩 있는 것으로 본다. 그러나 4가 약수인지 확인하는 과정에서 별도의 처리가 없는 경우, 짝이 존재하는 것으로 취급되어 약수 개수를 하나만 세야 하는 것을 2개로 세게 된다. 따라서 4가 약수인지 아닌지 확인할 때는 약수 개수를 1개만 세는 것이다.)

소인수분해를 이용해 약수를 세었다면 계산 속도가 더 빨랐겠지만, 코드 완성 및 정상동작 확인 이후에 이 아이디어가 떠올랐고, 무엇보다 이 코드의 성능이 중요하지 않기 때문에 그대로 두었다.

2. 과제(메모장) 수행 과정 및 결과

A. 메모장에 코드 작성

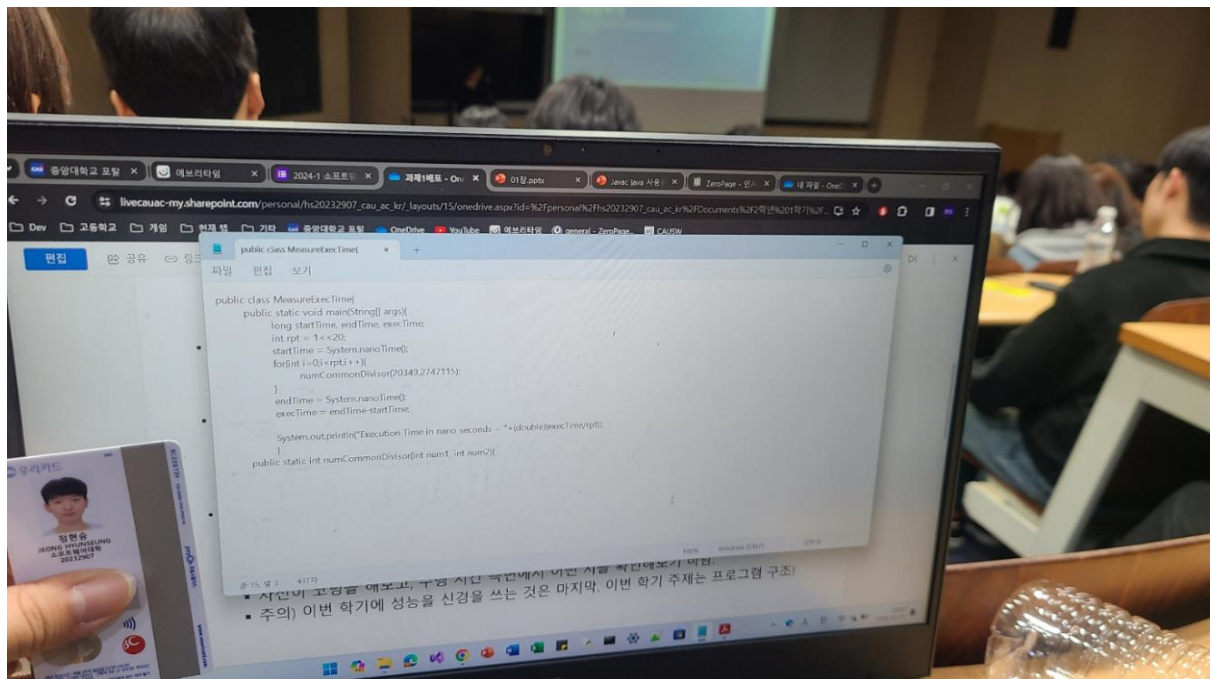


그림 3

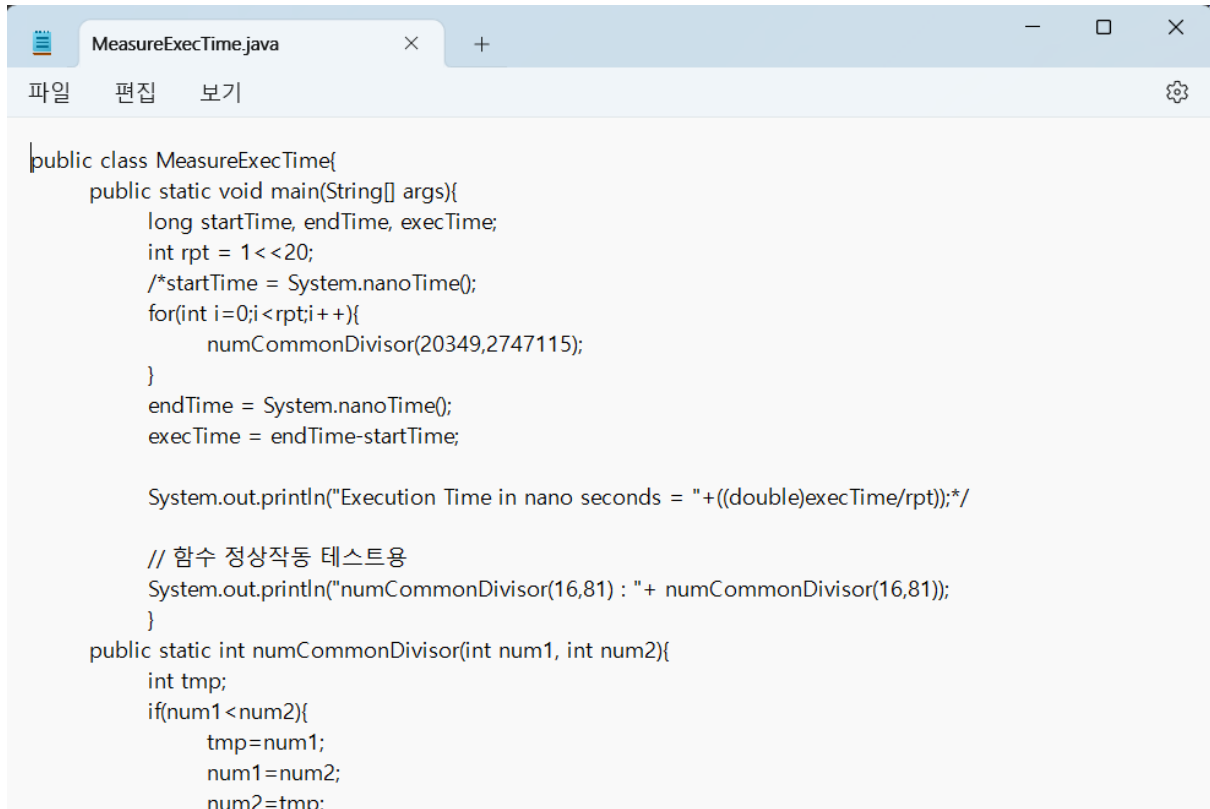
위 사진은 코드 작성 중임을 증명하기 위해 코드 작성 중 찍은 사진이다.

코드를 다 작성한 이후 이 파일이 **JAVA 소스코드**라는 의미로, "MeasureExecTime.java"라는 이름으로, 즉 확장자를 "java"라고 하여 **코드를 저장**하였다.

그림 4

코드가 저장된 모습이다. 확장자가 ".java"라 텍스트 파일로 표시되지 않고 java 로고가 표시된다.

이후 컴파일하기 전 먼저 공약수의 개수를 구하는 부분이 정상 동작하는지 확인하기 위해 코드를 다음과 같이 수정했다. 아래 그림 5는 첫 번째 테스트 당시 캡처한 것이다.



```
public class MeasureExecTime{
    public static void main(String[] args){
        long startTime, endTime, execTime;
        int rpt = 1<<20;
        /*startTime = System.nanoTime();
        for(int i=0;i<rpt;i++){
            numCommonDivisor(20349,2747115);
        }
        endTime = System.nanoTime();
        execTime = endTime-startTime;

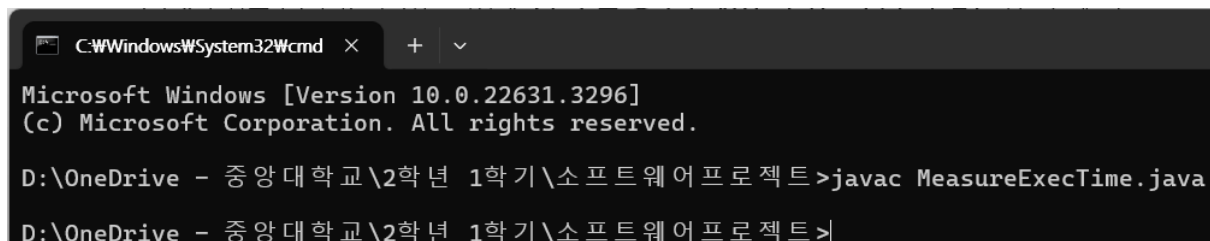
        System.out.println("Execution Time in nano seconds = "+((double)execTime/rpt));*/

        // 함수 정상작동 테스트용
        System.out.println("numCommonDivisor(16,81) : "+ numCommonDivisor(16,81));
    }
    public static int numCommonDivisor(int num1, int num2){
        int tmp;
        if(num1<num2){
            tmp=num1;
            num1=num2;
            num2=tmp;
        }
    }
}
```

그림 5

B. 코드 컴파일 및 실행

이후 이 코드를 실행하기 위해 아래 그림 6과 같이 **컴파일을 진행**하였다.



```
C:\Windows\System32\cmd
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>javac MeasureExecTime.java
D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>
```

그림 6

컴파일의 결과로 그림 7의 **파일이 생성**되었다.

그림 7

이 파일을 메모장으로 열어본 결과는 다음과 같다.

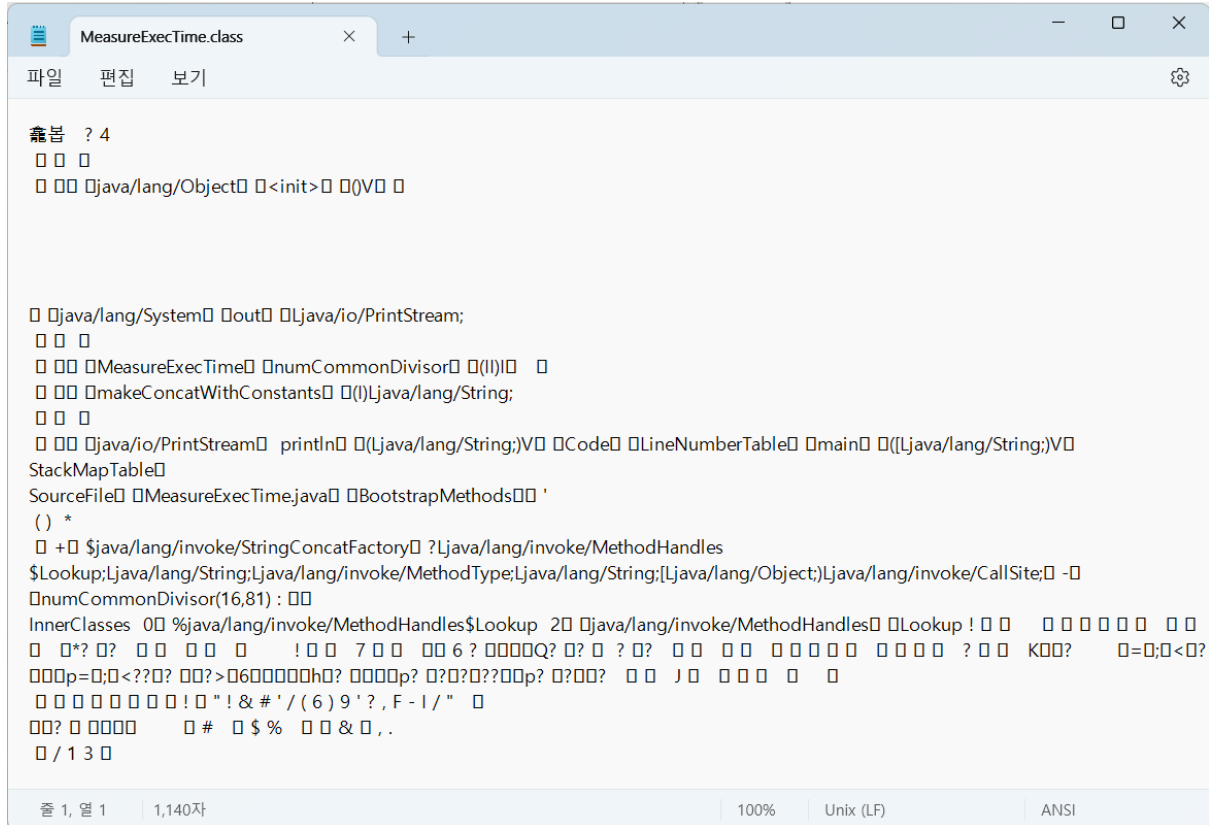


그림 8

글자가 많이 깨져있지만 중간중간 파일명, 메서드명, 입력한 문자열과 System, String 등 클래스 이름이 적혀있는 것을 확인할 수 있다. 애초에 이 컴파일의 결과물인 ".class" 파일은 사람의 이해를 위한 것이 아니라 컴퓨터의 이해를 위해 만들어진 파일이므로, 텍스트 형식으로 이해하지 못 하더라도 실행하여 결과를 확인하면 된다.

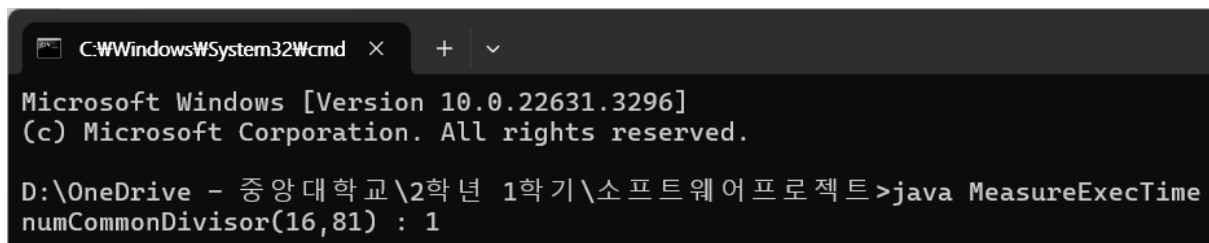


그림 9

위 그림 9는 컴파일된 파일을 실행한 결과이다. 16과 81은 서로소이므로 공약수의 개수는 1이며, 이것이 정확하게 계산되었음을 통해 코드가 정상 동작함을 확인할 수 있다.

테스트를 위해 숫자를 바꿔 컴파일 후 실행하는 과정을 반복했다.

```
C:\Windows\System32\cmd
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>javac MeasureExecTime.java

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>java MeasureExecTime
numCommonDivisor(16,24) : 4

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>javac MeasureExecTime.java

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>java MeasureExecTime
numCommonDivisor(7,49) : 7

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>javac MeasureExecTime.java

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>java MeasureExecTime
numCommonDivisor(20349,2747115) : 24

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>
```

그림 10

계속 테스트했더니 콘솔 출력과 계산 모두 정상으로 동작하는 것을 알 수 있다. 20349와 2747115의 공약수 개수는 아래 그림 11에서 확인할 수 있다.

FROM THE MAKERS OF WOLFRAM LANGUAGE AND MATHEMATICA



NATURAL LANGUAGE MATH INPUT EXTENDED KEYBOARD EXAMPLES UPLOAD RANDOM

Input

gcd(20349, 2747115)

$\text{gcd}(n_1, n_2, \dots)$ is the greatest common divisor of the n_i

Result Step-by-step solution

20349

Prime factorizations

$20349 = 3^2 \times 7 \times 17 \times 19$ (5 prime factors, 4 distinct)

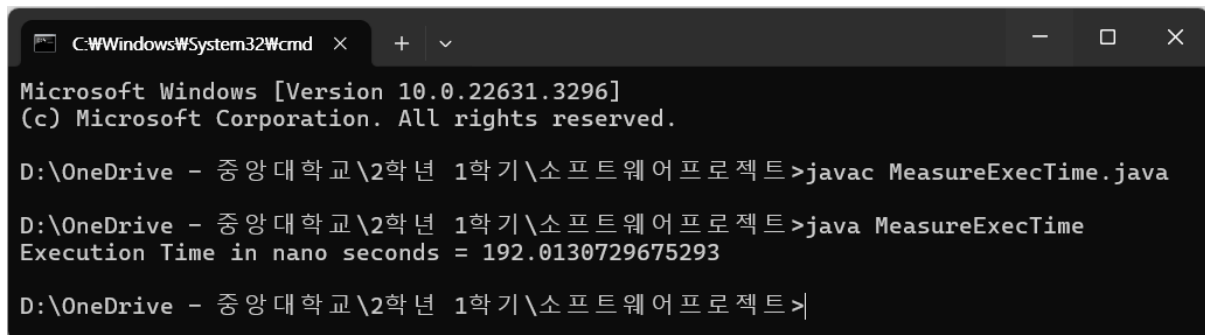
$2747115 = 3^5 \times 5 \times 7 \times 17 \times 19$ (9 prime factors, 5 distinct)

[Download Page](#) POWERED BY THE WOLFRAM LANGUAGE

그림 11

numCommonDivisor(20349,2747115)의 값이 맞게 나왔는지를 확인하기 위해 다른 계산기를 돌려본 결과이다. 계산기로는 WolframAlpha를 사용했으며, 둘의 최대공약수는 20349이고 $20349 = 3^2 \times 7 \times 17 \times 19$ 이므로 소인수분해를 이용해 약수를 세면(소인수분해 후 각 소인수의 차수에 1을 더한 값을 모두 곱하면 약수의 개수이다), numCommonDivisor(20349,2747115)는 $3 \times 2 \times 2 \times 2 = 24$ 가 맞다는 점을 알 수 있다.

공약수의 개수를 구하는 메소드의 정상 동작을 확인했으므로 그림 1처럼 코드를 원래대로 되돌려 놓고 다시 **컴파일 후 실행**하였다.



```
C:\Windows\System32\cmd
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>javac MeasureExecTime.java

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>java MeasureExecTime
Execution Time in nano seconds = 192.0130729675293

D:\OneDrive - 중앙대학교\2학년 1학기\소프트웨어프로젝트>
```

그림 12

이후 다음과 같이 시간을 측정할 수 있었다.

3. 과제(Eclipse) 수행 방법

Eclipse를 설치하고 프로젝트와 클래스를 만든 다음, 메모장에 작성했던 파일을 그대로 복사하고 실행했다. 코드를 메모장에서 그대로 복사한 이유는, 이미 만든 함수를 버리고 새로 만들 이유가 전혀 없다고 판단했기 때문이다.

4. 과제(Eclipse) 수행 증명 및 결과

A. 코드 작성 및 확인 과정

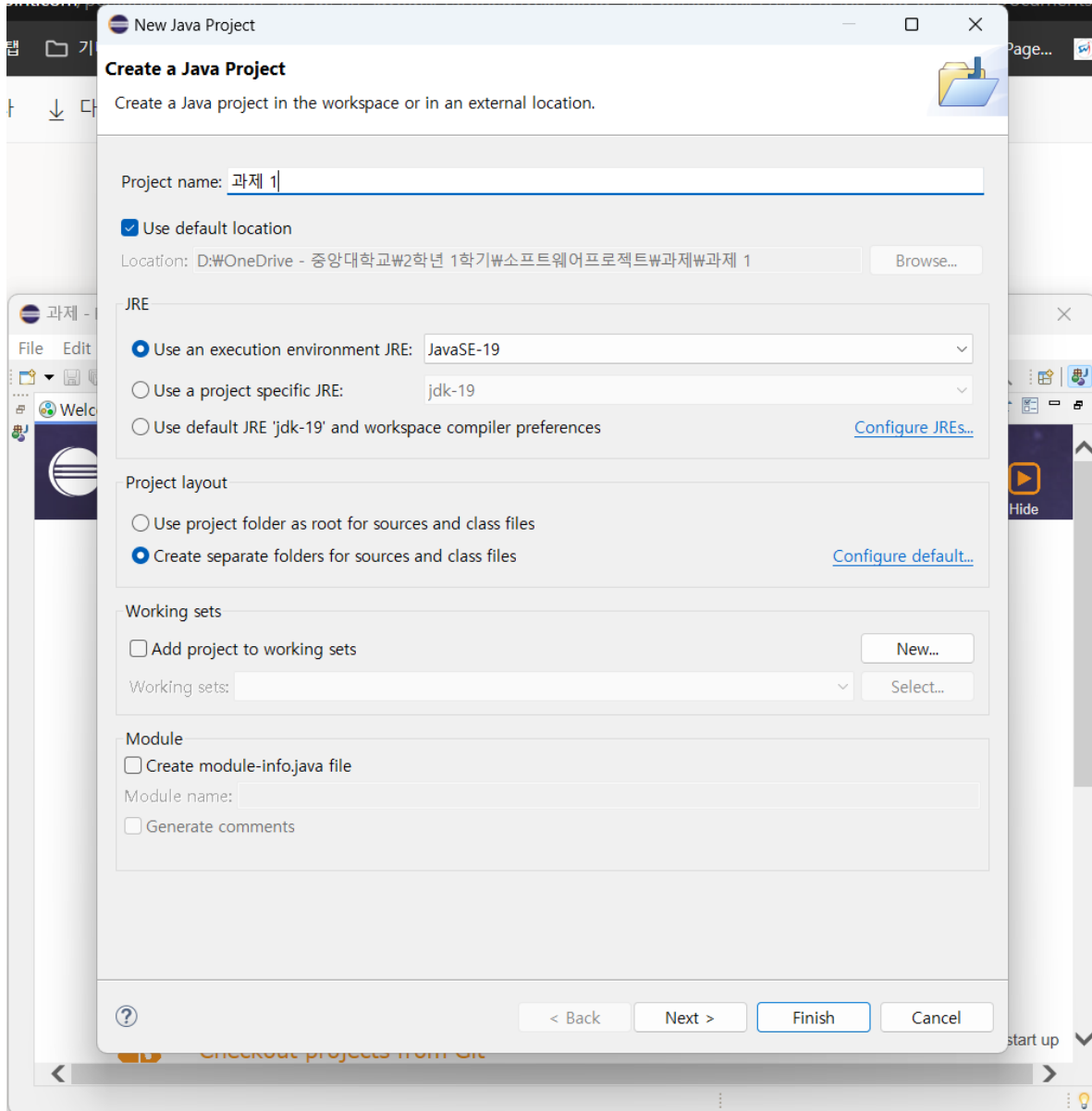


그림 13

위 그림은 프로젝트를 만드는 과정에서 캡처한 것이다.

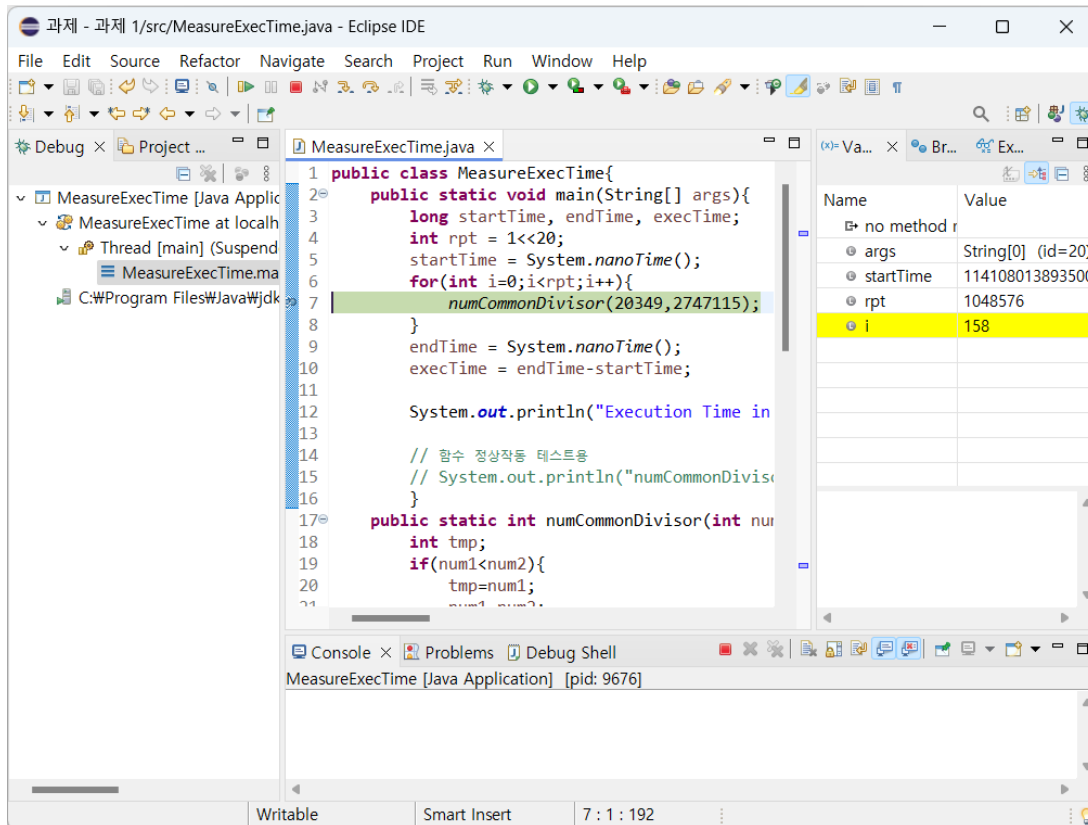


그림 14

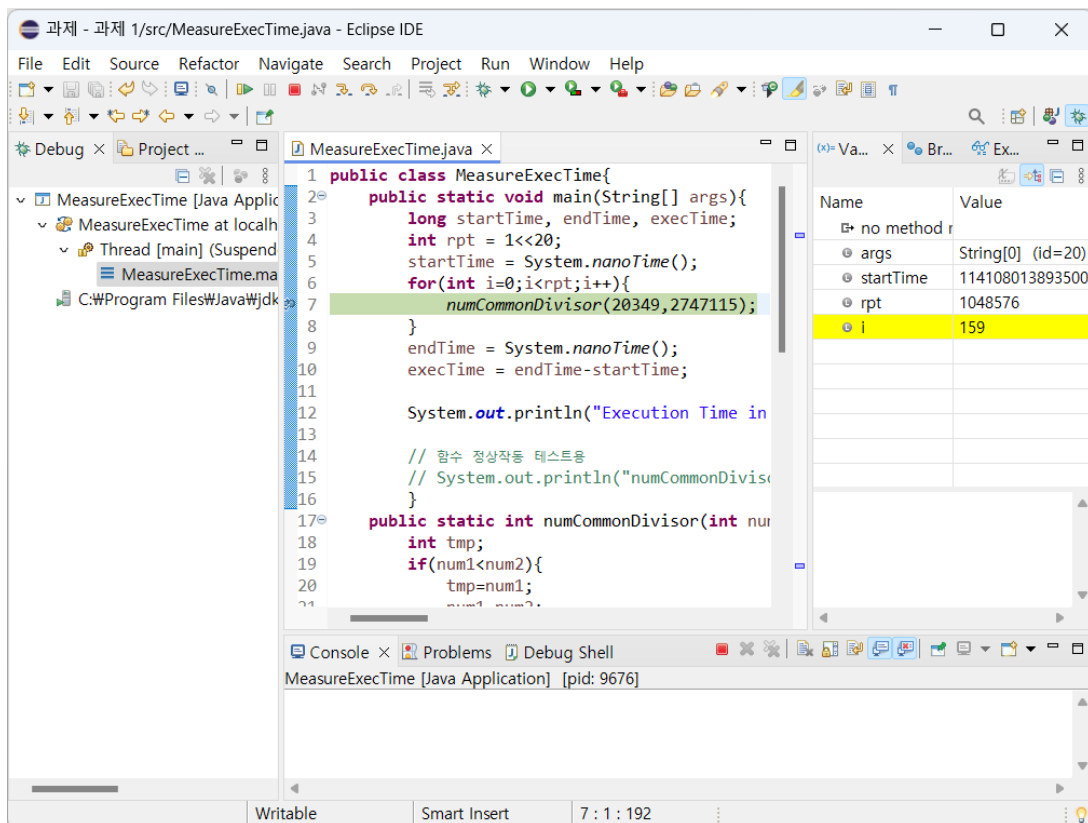


그림 15

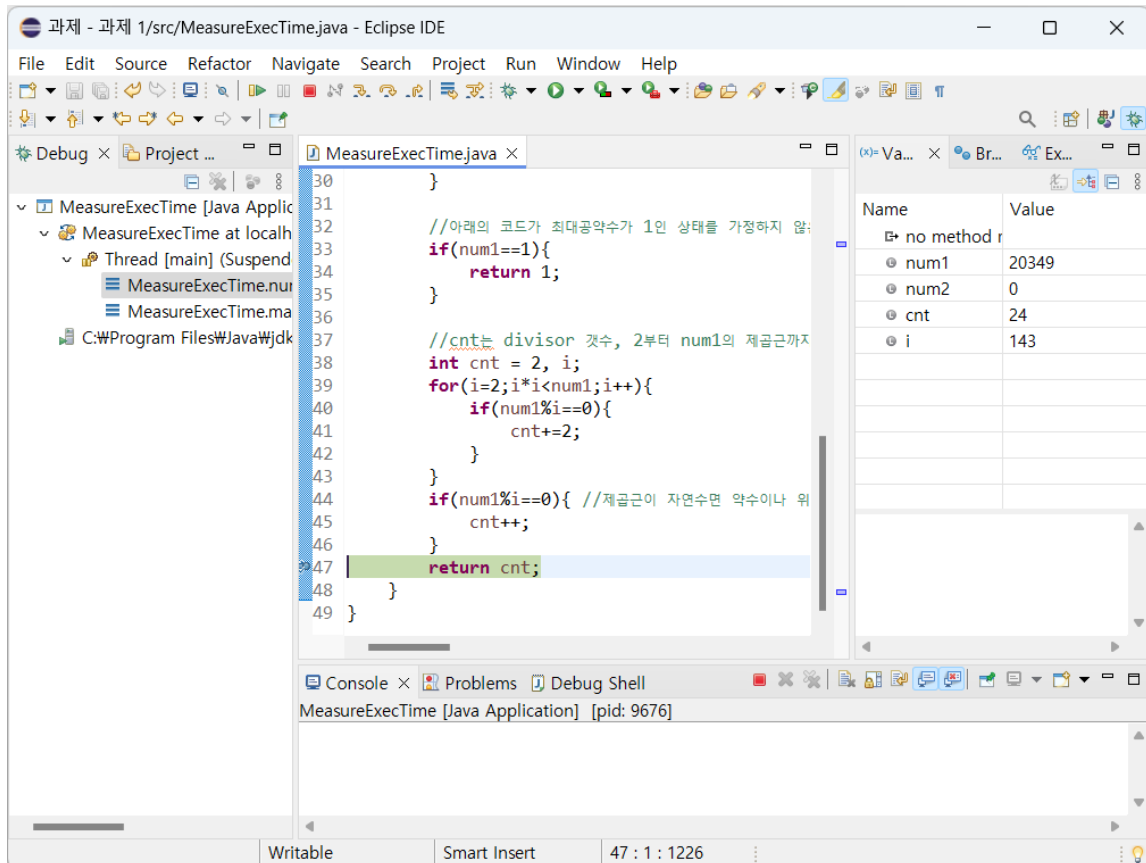


그림 16

위 그림 14, 그림 15, 그림 16은 코드를 붙여 넣은 뒤 디버그 모드로 실행하여 코드의 정상 작동을 다시 한번 확인한 것이다. 그림 14와 그림 15에서 시작 시각과 반복 횟수 1048576회 등 모든 변수가 정확히 입력되었고, 두 그림의 차이에서 한 번 공약수의 개수를 구하는 메서드가 실행될 때마다 `i` 값이 1씩 올라간다는 점을 알 수 있다. 그리고 그림 16에서는 약수를 세는 메서드에서 약수의 개수를 정확히 24개를 세었다는 점을 확인할 수 있다.

B. 실행 결과 확인

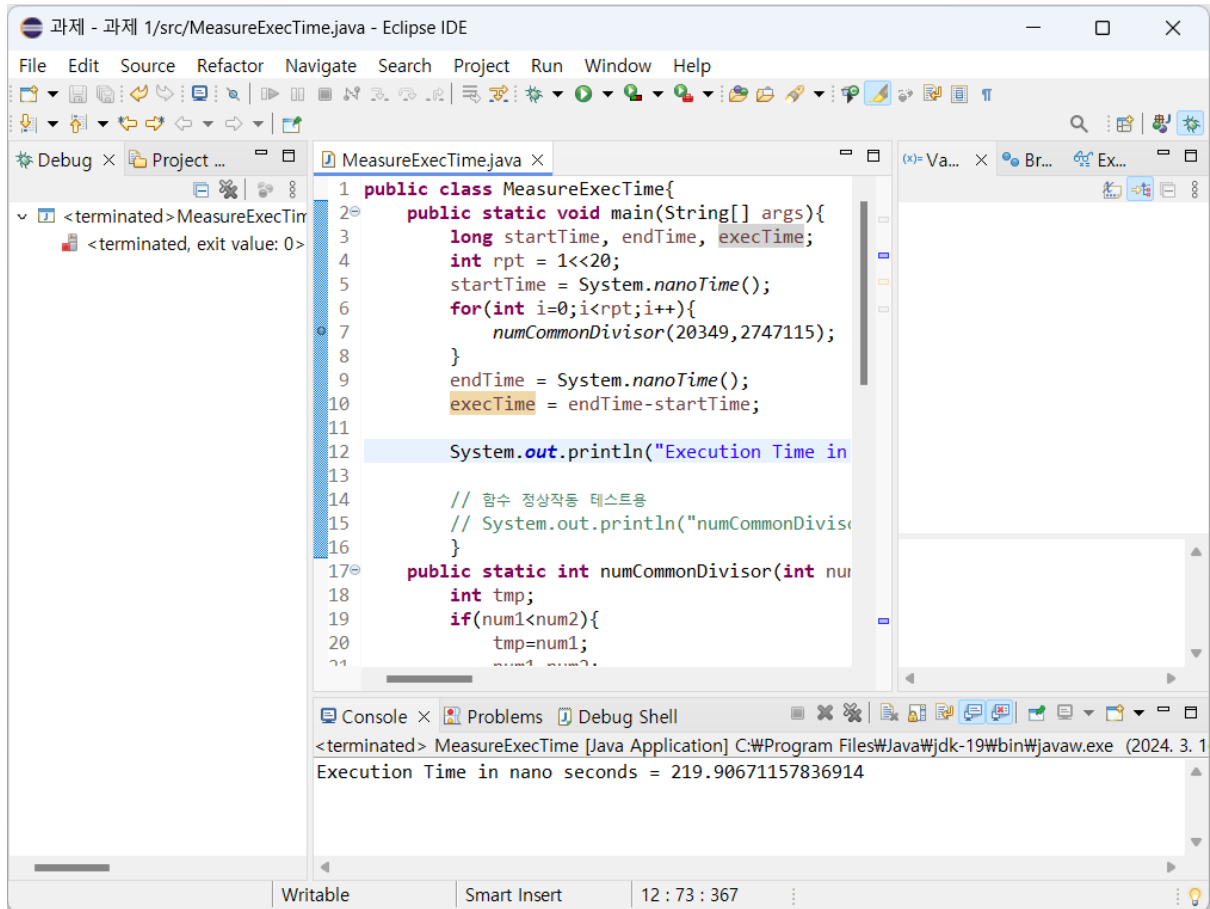


그림 17

위 그림 17은 디버그 모드를 중단하고 코드를 실행한 결과이다. 걸린 시간은 메모장에 작성했던 것보다는 길지만 어쨌든 정상 작동한다는 점을 알 수 있다.

5. 소스코드

```
public class MeasureExecTime{
    public static void main(String[] args){
        long startTime, endTime, execTime;
        int rpt = 1<<20;
        startTime = System.nanoTime();
        for(int i=0;i<rpt;i++){
            numCommonDivisor(20349,2747115);
        }
        endTime = System.nanoTime();
        execTime = endTime-startTime;

        System.out.println("Execution Time in nano seconds = "+((double)execTime/rpt));

        // 함수 정상작동 테스트용
        // System.out.println("numCommonDivisor(20349,2747115) : "+
numCommonDivisor(20349,2747115));
    }
    public static int numCommonDivisor(int num1, int num2){
        int tmp;
        if(num1<num2){
            tmp=num1;
            num1=num2;
            num2=tmp;
        }

        // Euclidean Algorithm 사용, num1은 둘 중 큰 수, num2는 둘 중 작은 수
        while(num2!=0){
            tmp=num1%num2;
            num1=num2;
            num2=tmp;
        }

        //아래의 코드가 최대공약수가 1인 상태를 가정하지 않은 바, 이에 대한 처리를
        별도로 진행
        if(num1==1){
            return 1;
        }
    }
}
```

```
}
```

//cnt는 divisor 갯수, 2부터 num1의 제곱근까지 하나하나 약수인지 계산(약수를 작은 것부터 나열했을 때, 첫 번째 약수와 마지막 약수의 곱은 원래 수 자신, 두 번째 약수와 마지막에서 두 번째 약수의 곱은 원래 수 자신... 이라는 점을 이용함), 이때 1과 자기 자신은 계산에서 제외되므로 cnt에 2를 넣어주고 시작

```
int cnt = 2, i;
```

```
for(i=2;i*i<num1;i++){
```

```
    if(num1%i==0){
```

```
        cnt+=2;
```

```
    }
```

```
}
```

if(num1%i==0){ //제곱근이 자연수면 약수이나 위와는 달리 두 번 세면 안 되므로 별도로 뺌

```
    cnt++;
```

```
}
```

```
return cnt;
```

```
}
```

```
}
```

평 가 표

평가 항목	학생 자체 평가 (리포트 해당 부분 표시 및 간단한 의견)	평가 (빈칸)	기타
- 실행 시간 동작? - numCommonDivisor() 정확히 동작? * 여러 테스트 * numCommonDivisor(20349, 2747115)의 실행 시간	실행 시간 동작 확인 numCommonDivisor() 동작 확인(4회) (16, 81), (16, 24), (7, 49), (20349, 2747115) (WolframAlpha로 계산하여 결과 확인) 실행 시간 측정 성공		
Eclipse를 이용한 실행	Eclipse를 활용한 실행 완료		
에디터와 명령어를 이용한 실행 - 각 단계별 결과물 과 그 의미 는?	".java" 파일 생성 후 javac로 시작하는 명령어 입력 시 컴파일 후 확장자가 ".class"인 파일의 생성을 확인함 결과물의 의미: Chapter 2 Java + 클래스 이름 입력 시 코드 동작 확인		
리포트 작성 - 평가 항목에 맞게 리포트 작 성? - <u>모든 파일을 하나의 pdf 문서</u> <u>로</u>	동작 여부를 알아볼 수 있는 결과: 그림 10, 그림 15 각 방법별로 구분: Chapter 1~2와 3~4로 나눔 평가자가 보기 좋게 작성: 목차로 나눔 자체평가표 첨부: 완료 수행 증명: 그림들 JAVA 실행 모델을 중간결과물과 함께 확인: Chapter 2 과정 및 이유 설명: Chapter 1~4 하나의 pdf 문서로: 완료		
기타 추가 설명 (필요한 경우)	방법 설명(코드의 원리를 이해 못 하는 때 대비): Chapter 1		
총평/계	평가자 입장에서 자신의 리포트를 살펴보기가 목적 즉, 평가자가 체크하고자하는 사항을 쉽게 찾아볼 수 있도록 리포트가 기술되어있는지 점검		

* 학생 자체 평가는 점수에 반영되지 않음.

* 학생 스스로 자신의 보고서를 평가하면서, 요구 사항을 만족하면서 체계적으로 프로젝트를 마무리하도록 유도
 하는 것이 목적임.