**12161581 박진성**

**<1> 소스코드 요구사항 및 설계 주안점**

우선 구현해야 하는 메소드는 총 4개였다.

check\_prime 메소드)

기존 소수 구하는 메소드에서, 효율성을 위해 두가지를 수정하였다

For 루프에서 I <= k 까지 반복하는게 아닌, i\*I <=k 까지 반복하게 하여,

반복횟수를 줄였다.

또한, for 루프 내부에서, is\_prime[i] 이 ture 일때만 반복을 실행하도록 하여, 조건 하나를 더 추가시켰다

printPrimeNumber)

소수가 판별되었다면, is\_prime 배열에 해당 숫자를 인덱스로 갖는 부분이 true 로 바뀐다.

따라서, 소수가 몇 개인지 count 하기 위해, is\_prime 에서 true 의 개수를 구하면 된다

printLegendre)

르장드르 수를 구하기 위해서는, 자연로그를 사용해야한다.

Math.log() 를 이용하여, 자연로그를 구하였고, 해당 자료형이 double 이기 때문에,

Math.round() 를 이용하여 반올림을 해주었다.

printTwinprime)

쌍둥이 소수는 (3,5) 한가지 예외를 제외하고,

모두 (6k+5, 6k+1) 꼴이 된다. 따라서 1부터 count 를 해주었으며,

Check\_prime으로 또다시 소수판별을 할 필요 없이,

이미 완성되어있던, is\_prime 배열에서 바로 소수판별이 가능하게끔 하였다.

**<2> 소스코드 전체**

**package** Main;

**import** java.util.Scanner;

**import** java.util.Arrays;

**public** **class** Main{

**public** **static** **boolean**[] *is\_prime*;

**public** **static** **void** check\_prime(**int** k) {

**int** i;

**for**(i = 2; i\*i <= k; i++) {

**if**(*is\_prime*[i] == **true**) {

**if**(k % i == 0) {

*is\_prime*[k] = **false**;

**return**;

}

}

}

**return**;

}

**public** **static** **void** printPrimeNumber() {

**int** count = 0;

**for**(**int** i = 2; i < *is\_prime*.length; i++) {

**if**(*is\_prime*[i] == **true**) {

count++;

}

}

System.***out***.print("Number of Primes:" + count + ", ");

}

**public** **static** **void** printLegendre(**int** L) {

**long** temp = Math.*round*(L / (Math.*log*(L) - 1));

// 르장드르 수 실수로 나오므로, 반올림처리

System.***out***.print("Legendre:" + temp + ", ");

}

**public** **static** **void** printTwinprime(**int** T) {

**int** k = 1;

**int** count = 1;

// 유일한 반례 (3,5) 존재하므로, count 1 부터 시작

**while**((6\*k+1) <= T) {

**if**(*is\_prime*[6\*(k-1) + 5] == **true** && *is\_prime*[6\*k + 1] == **true**) {

count++;

}

k++;

}

System.***out***.print("Nums of Twin Primes: " + count);

}

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

Scanner scanner = **new** Scanner(System.***in***);

**int** n;

System.***out***.println("학번: 12161581 이름:박진성");

**while**(**true**) {

System.***out***.print("탐색 범위 선택(0입력시 종료):");

n = scanner.nextInt();

**if**(n == 0){

**break**;

}

*is\_prime* = **new** **boolean**[n+1];

Arrays.*fill*(*is\_prime*, **true**);

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

**for**(**int** i = 2; i <= n; i++) {

*check\_prime*(i);

}

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

*printPrimeNumber*();

*printLegendre*(n);

*printTwinprime*(n);

System.***out***.println("(" + (ex - sx) + "ms)");

}

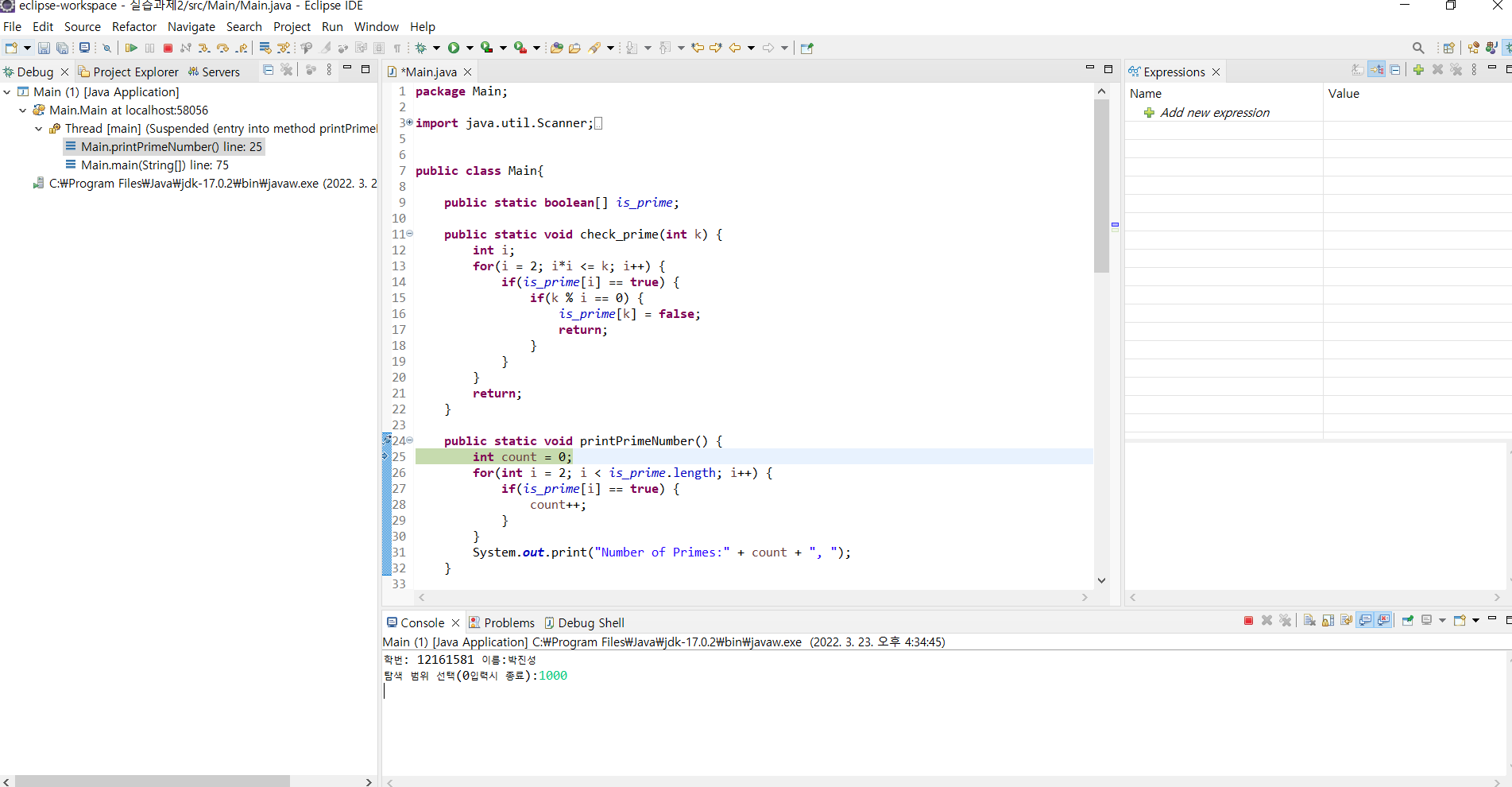
scanner.close();

**return**;

}

}

**<3> 실행 결과**

****

(디버깅화면)

**텍스트이(가) 표시된 사진

자동 생성된 설명**

(효율적인 방법으로 하였을 때 실행화면)

**텍스트이(가) 표시된 사진

자동 생성된 설명**

(브루트포스 한 방법으로 하였을 때 실행화면. 속도차이가 어마어마하다)

**<4> 결론 및 배운점**

소수, 르장드르수, 쌍둥이 소수 세가지를 모두 구해보는 코드를 작성 해 보았다.

과제의 요구사항대로 브루트포스한 방법과 개선된방법 두가지 모두 실행 해 보았으며,

생각보다 어마어마한 속도차이가 남에 놀랐다.

이를통해, 알고리즘을 작성할때는 최소한의 반복횟수를 가져가게끔 생각하는 것이 매우 중요하다고 느꼈다.