Goal: Implement simple programs with and without control flow (ie, branch and loop) statements.

Problem 1. (*Greet Three*) Write a program called `GreetThree.java` that receives $name_1$ (String), $name_2$ (String), and $name_3$ (String) as command-line inputs, and writes the string "Hi $name_3$, $name_2$, and $name_1$." as standard output.

```
× ~/workspace/simple_programs
$ javac -d out src/GreetThree.java
$ java GreetThree Alice Bob Carol
Hi Carol, Bob, and Alice.
$ java GreetThree Dan Eve Fred
Hi Fred, Eve, and Dan.
$ _
```

Problem 2. (*Three Sort*) Write a program called `ThreeSort.java` that receives x (int), y (int), and z (int) as command-line inputs, and writes them as standard output in ascending order, separated by a space. Your solution must only use `Math.min()`, `Math.max()`, and basic arithmetic operations to figure out the ordering.

```
× ~/workspace/simple_programs
$ javac -d out src/ThreeSort.java
$ java ThreeSort 1 3 2
1 2 3
$ java ThreeSort 3 2 1
1 2 3
$ _
```

Problem 3. (*Great Circle Distance*) Write a program called `GreatCircle.java` that receives x_1 (double), y_1 (double), x_2 (double), and y_2 (double) as command-line inputs, representing the latitude and longitude in degrees of two points on Earth, and writes as standard output the great-circle distance d (in km) between them, computed as

$$d = 6359.83 \arccos(\sin(x_1)\sin(x_2) + \cos(x_1)\cos(x_2)\cos(y_1 - y_2)).$$

```
× ~/workspace/simple_programs
$ javac -d out src/GreatCircle.java
$ java GreatCircle 48.87 -2.33 37.8 -122.4
8701.387455462233
$ java GreatCircle 46.36 -71.06 39.90 116.41
10376.503884802196
$ _
```

Problem 4. (*Uniform Random Numbers*) Write a program called `Stats.java` that receives a (int) and b (int) as command-line inputs, generates three random doubles (x_1, x_2, and x_3), each from the interval $[a, b)$, computes their mean $\mu = (x_1 + x_2 + x_3)/3$, variance $\text{var} = ((x_1 - \mu)^2 + (x_2 - \mu)^2 + (x_3 - \mu)^2)/3$, and standard deviation $\sigma = \sqrt{\text{var}}$, and writes those values as standard output, separated by a space.

```
× ~/workspace/simple_programs
$ javac -d out src/Stats.java
$  java Stats 0 1
0.13146913917517933 0.011467803615287939 0.1070878313128431
$ java Stats 50 100
```

```
55.36812680970314 7.156153169158455 2.675098721385522
$ _
```

Problem 5. (*Triangle Inequality*) Write a program called `Triangle.java` that receives x (int), y (int), and z (int) as command-line inputs, and writes `true` as standard output if each one of them is less than or equal to the sum of the other two, and `false` otherwise.

```
× ~/workspace/simple_programs
$ javac -d out src/Triangle.java
$ java Triangle 3 3 3
true
$ java Triangle 2 4 7
false
$ _
```

Problem 6. (*Quadratic Equation*) Write a program called `Quadratic.java` (a variant of the one we dicussed in class) that receives a (double), b (double), and c (double) as command-line inputs, and writes as standard output the roots of the quadratic equation $ax^2 + bx + c = 0$. Your program should report the message "Value of a must not be 0" if $a = 0$, and the message "Value of discriminant must not be negative" if $b^2 - 4ac < 0$.

```
× ~/workspace/simple_programs
$ javac -d out src/Quadratic.java
$ java Quadratic 0 1 -3
Value of a must not be 0
$ java Quadratic 1 1 1
Value of discriminant must not be negative
$ java Quadratic 1 -5 6
3.0 2.0
$ _
```

Problem 7. (*Six-sided Die*) Write a program called `Die.java` that simulates the roll of a six-sided die, and writes as standard output the pattern on the top face.

```
× ~/workspace/simple_programs
$ javac -d out src/Die.java
$ java Die
*   *
  *
*   *
$ java Die
*


    *
$ _
```

Problem 8. (*Playing Card*) Write a program called `Card.java` that simulates the selection of a random card from a standard deck of 52 playing cards, and writes it as standard output.

```
× ~/workspace/simple_programs
$ javac -d out src/Card.java
$ java Card
3 of Clubs
$ java Card
Ace of Spades
$ _
```

Problem 9. (*Greatest Common Divisor*) Write a program called `GCD.java` that receives p (int) and q (int) as command-line inputs, and writes as standard output the greatest common divisor (GCD) of p and q.

```
× ~/workspace/simple_programs
$ javac -d out src/GCD.java
$ java GCD 408 1440
24
$ java GCD 21 22
1
$ _
```

Problem 10. (*Factorial Function*) Write a program called `Factorial.java` that receives n (int) as command-line input, and writes as standard output the value of $n!$, which is defined as $n! = 1 \times 2 \times \ldots (n-1) \times n$. Note that $0! = 1$.

```
× ~/workspace/simple_programs
$ javac -d out src/Factorial.java
$ java Factorial 0
1
$ java Factorial 5
120
$ _
```

Problem 11. (*Fibonacci Function*) Write a program called `Fibonacci.java` that receives n (int) as command-line input, and writes as standard output the nth number from the Fibonacci sequence $(0, 1, 1, 2, 3, 5, 8, 13, \ldots)$.

```
× ~/workspace/simple_programs
$ javac -d out src/Fibonacci.java
$ java Fibonacci 10
55
$ java Fibonacci 15
610
$ _
```

Problem 12. (*Primality Test*) Write a program called `PrimalityTest.java` that receives n (int) as command-line input, and writes as standard output if n is a prime number or not.

```
× ~/workspace/simple_programs
$ javac -d out src/PrimalityTest.java
$ java PrimalityTest 31
true
```

```
$ java PrimalityTest 42
false
$ _
```

Problem 13. (*Counting Primes*) Write a program called `PrimeCounter.java` that receives n (int) as command-line input, and writes as standard output the number of primes less than or equal to n.

```
× ~/workspace/simple_programs
$ javac -d out src/PrimeCounter.java
$ java PrimeCounter 10
4
$ java PrimeCounter 100
25
$ java PrimeCounter 1000
168
$ _
```

Problem 14. (*Perfect Numbers*) A perfect number is a positive integer whose proper divisors add up to the number. For example, 6 is a perfect number since its proper divisors 1, 2, and 3 add up to 6. Write a program called `PerfectNumbers.java` that receives n (int) as command-line input, and writes as standard output the perfect numbers that are less than or equal to n.

```
× ~/workspace/simple_programs
$ javac -d out src/PerfectNumbers.java
$ java PerfectNumbers 10
6
$ java PerfectNumbers 1000
6
28
496
$ _
```

Problem 15. (*Ramanujan Numbers*) Srinivasa Ramanujan was an Indian mathematician who became famous for his intuition for numbers. When the English mathematician G. H. Hardy came to visit him one day, Hardy remarked that the number of his taxi was 1729, a rather dull number. Ramanujan replied, "No, Hardy! It is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways." Verify this claim by writing a program `RamanujanNumbers.java` that receives n (int) as command-line input, and writes as standard output all integers less than or equal to n that can be expressed as the sum of two cubes in two different ways. In other words, find distinct positive integers a, b, c, and d such that $a^3 + b^3 = c^3 + d^3 \leq n$.

```
× ~/workspace/simple_programs
$ javac -d out src/RamanujanNumbers.java
$ java RamanujanNumbers 10000
1729 = 1^3 + 12^3 = 9^3 + 10^3
4104 = 2^3 + 16^3 = 9^3 + 15^3
$ java RamanujanNumbers 40000
1729 = 1^3 + 12^3 = 9^3 + 10^3
4104 = 2^3 + 16^3 = 9^3 + 15^3
13832 = 2^3 + 24^3 = 18^3 + 20^3
39312 = 2^3 + 34^3 = 15^3 + 33^3
```

```
32832 = 4^3 + 32^3 = 18^3 + 30^3
20683 = 10^3 + 27^3 = 19^3 + 24^3
$ _
```

Files to Submit:

1. `GreetThree.java`
2. `ThreeSort.java`
3. `GreatCircle.java`
4. `Stats.java`
5. `Triangle.java`
6. `Quadratic.java`
7. `Die.java`
8. `Card.java`
9. `GCD.java`
10. `Factorial.java`
11. `Fibonacci.java`
12. `PrimalityTest.java`
13. `PrimeCounter.java`
14. `PerfectNumbers.java`
15. `RamanujanNumbers.java`
16. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond *Control Flow*.
- Your code is clean, well-organized, uses meaningful variable names, includes useful comments, and is efficient.
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. In section `#1`, for each problem, state its goal in your own words and describe your approach to solve the problem along with any issues you encountered and if/how you managed to solve those issues.