Joshua Pollock, Jamie Barton

February 28th

CS136 Lab Section 1

**Lab 04: RSA Encryption**

**Problem Statement**

The problem given to us by this lab is to create a Java program that will be able to perform RSA encryption. RSA encryption is a type of encryption that can encrypt a message/integer using a public key, and then only be decrypted using a private key that is mathematically based off the public key.

**Constraints:**

1. The program will implement the BigInteger class in java (Specifically the .probablePrime() method
2. The numbers used to create the totient and private keys must be prime numbers
3. The prime numbers used should have a bitlength of 256
4. The starting number, encrypted number, and decrypted numbers should be printed out.

**Assumptions:**

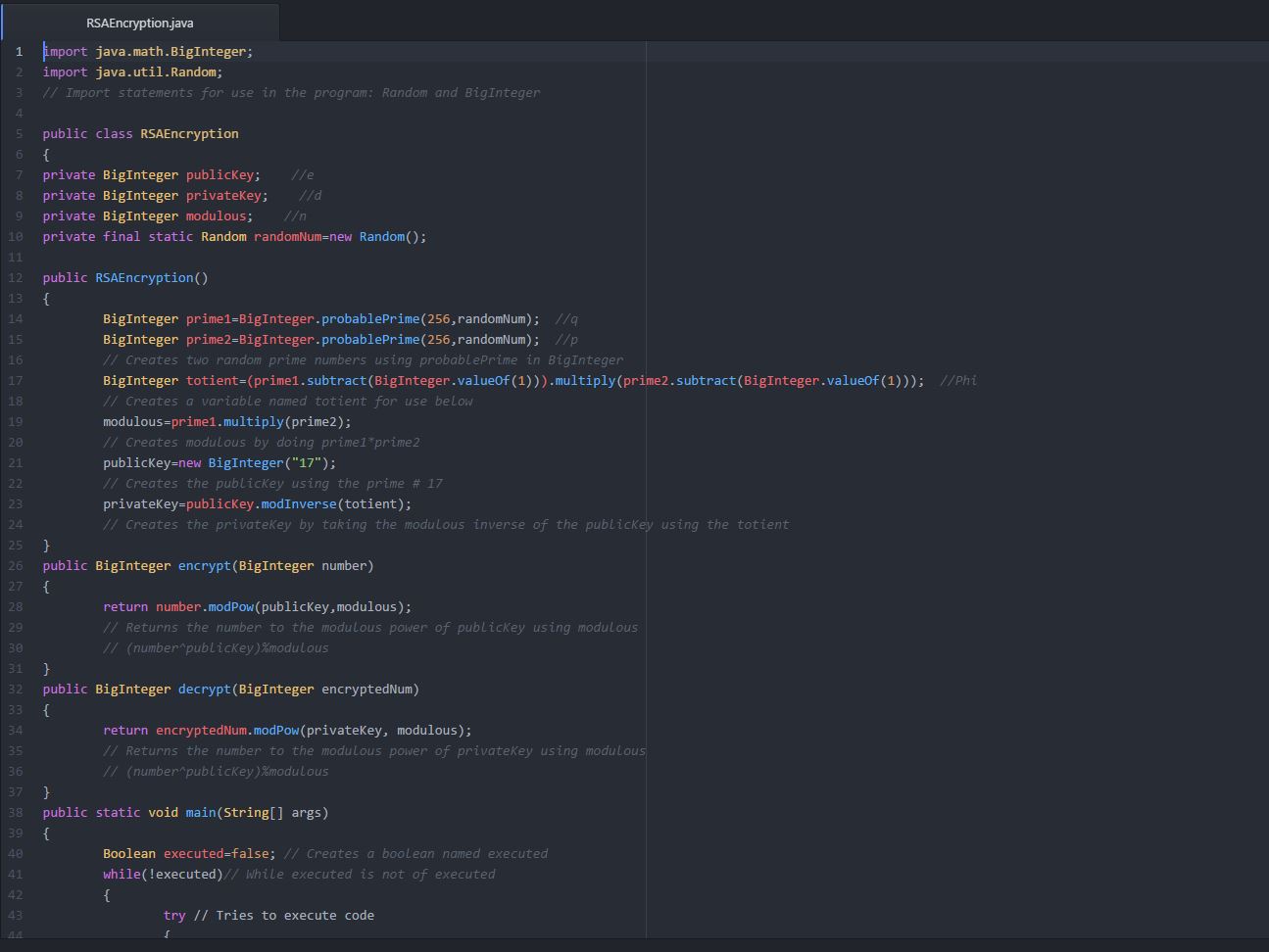
1. A try and catch block is acceptable to catch non-primes being put into the RSAEncryption class
2. The private key is created using the modulus inverse of the public key and the totient

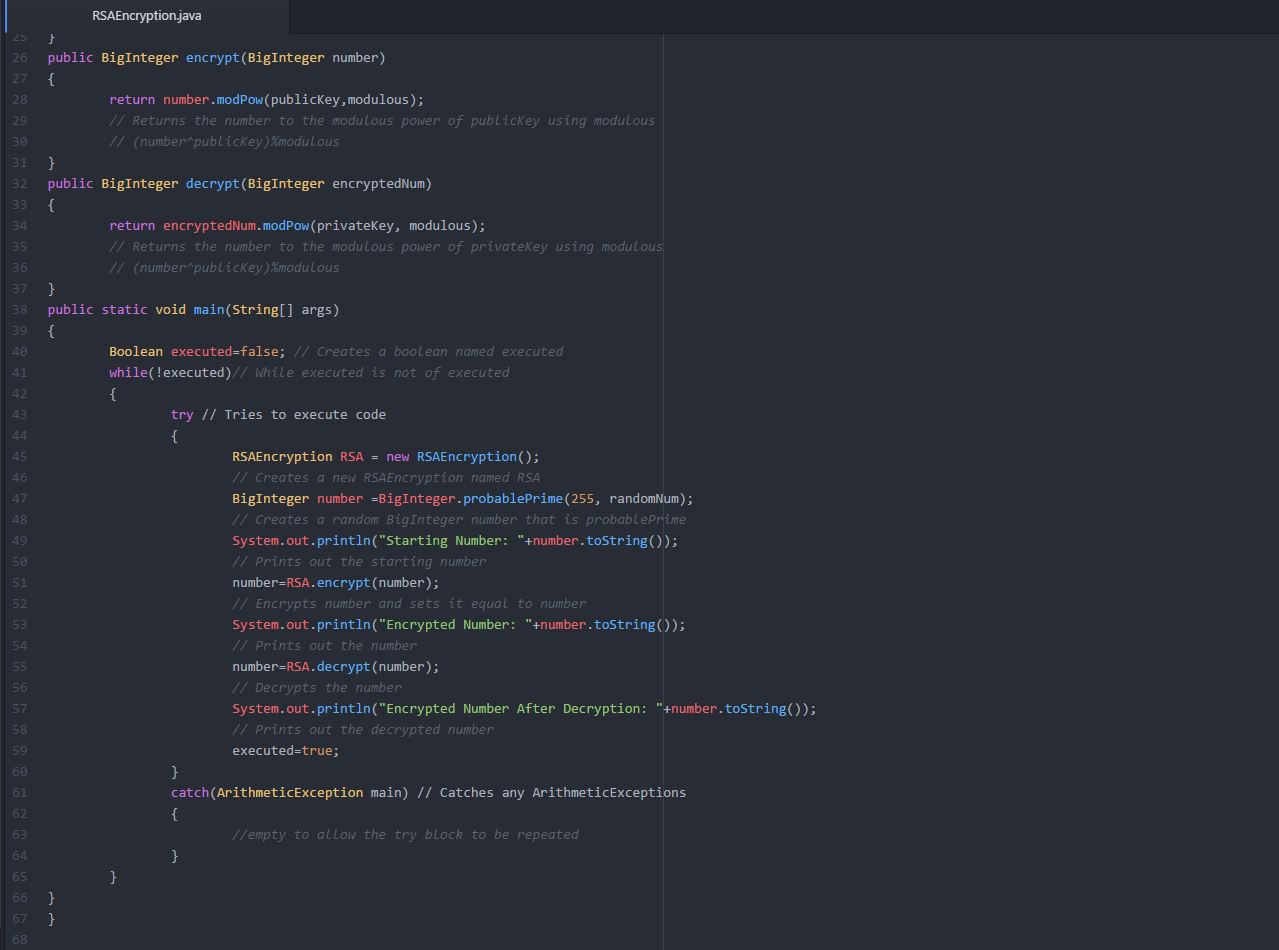
**Features:**

1. RSAEncryption()
2. encrypt(Biginteger number)
3. decrypt(Biginteger encryptedNum)

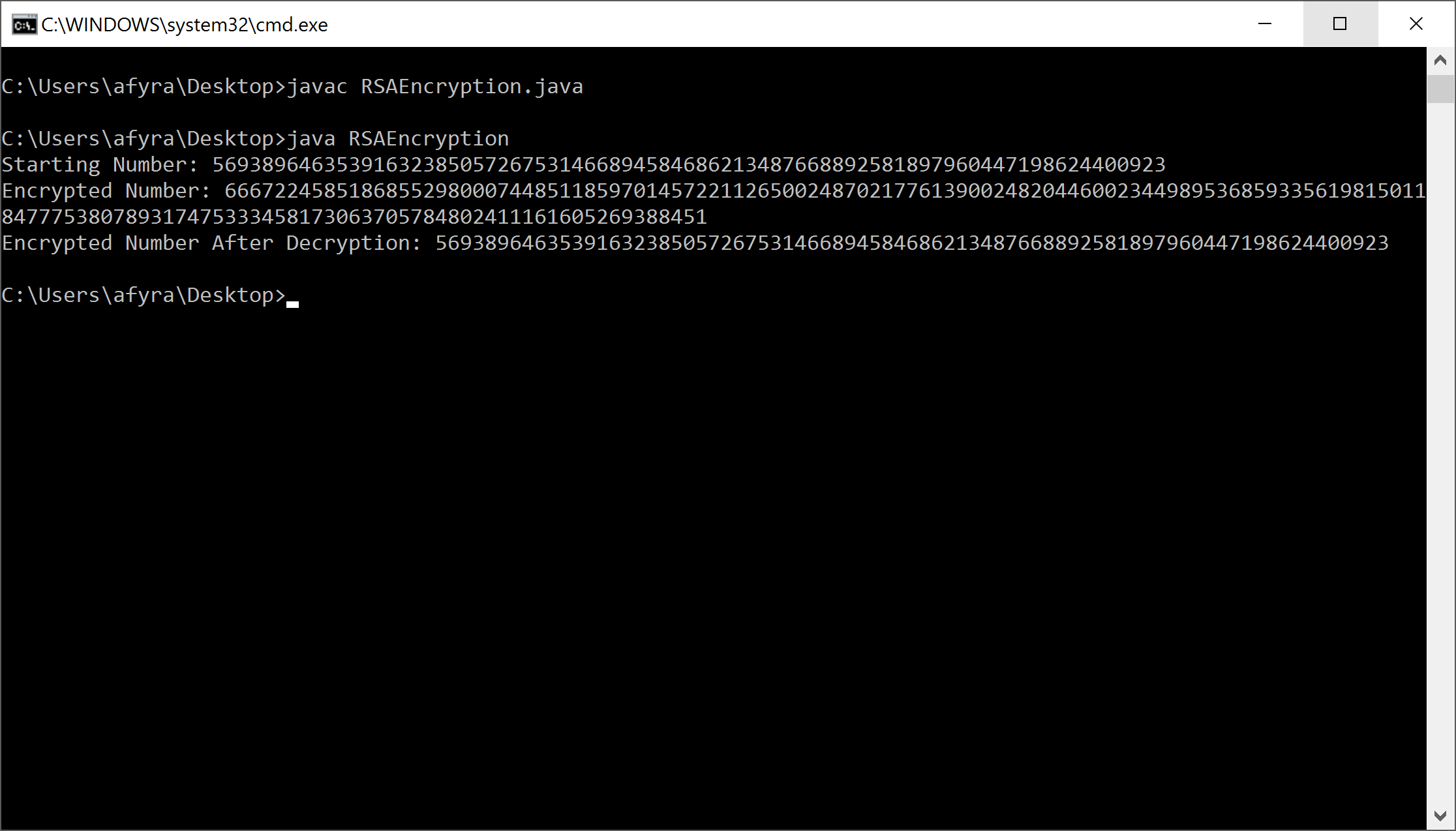
**Planning**

After reading the wikipedia page we decided that sorting out our variables would be a good way to start. After assigning all the variables we started brainstorming what we would need for everything in order to get everything working correctly, this involved looking up the different relationships between the Biginteger class and modulus power/inverse, as well as the .probablePrime() method, among others. After doing this, we began to work together to start the coding process.

**Code**



**Running Example:**



**Reflection**

This lab was complex for several reasons. One of these reasons being that RSA encryption is rather complex to someone who has no idea of what it is or what it does, along with trying to implement this into the Java code made it quite the task. We managed to get everything written in a Java file, but then we encountered a very strange compiling issues that we did not know how to fix. After discussion with the lab instructors, we discovered that we actually did not have a prime number as the publicKey, which is what caused the error. After the initial fix to the error we discovered that occasionally we would still get the same error, so we implemented a try and catch block to try to stop this ArithmeticException error from occurring in the main function block. This implementation of the block helped to solve the problem. This error is most likely due to a prime number not being generated in the RSAEncryption block due to random numbers being used, and could possibly be fixed. We found, however, that the try and catch block did an excellent job at keeping the error from stopping the code’s execution and filled the lab parameters. Overall, this lab helped us to understand the Biginteger class in Java and how RSA encryption works. We also do not see for crucial edits to our final code as it completes all requirements given by the lab.