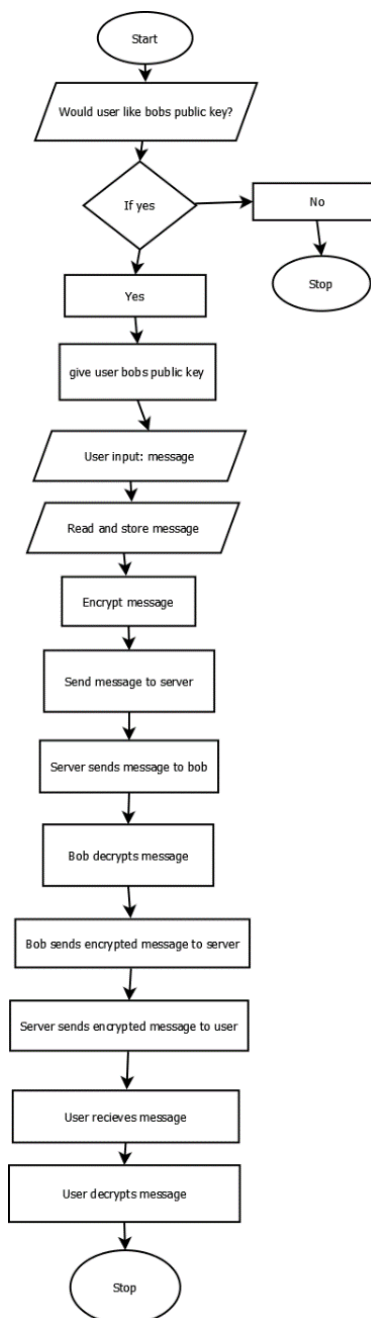
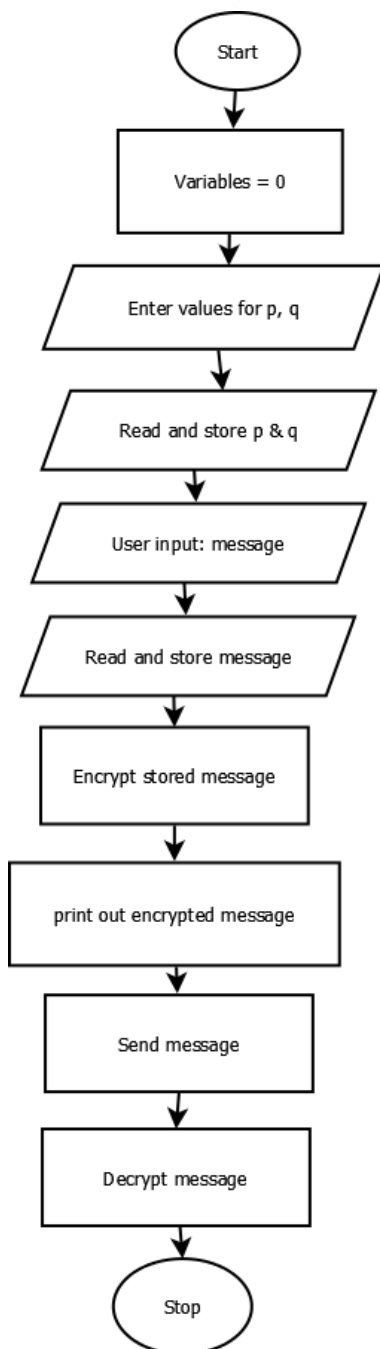


My Report:

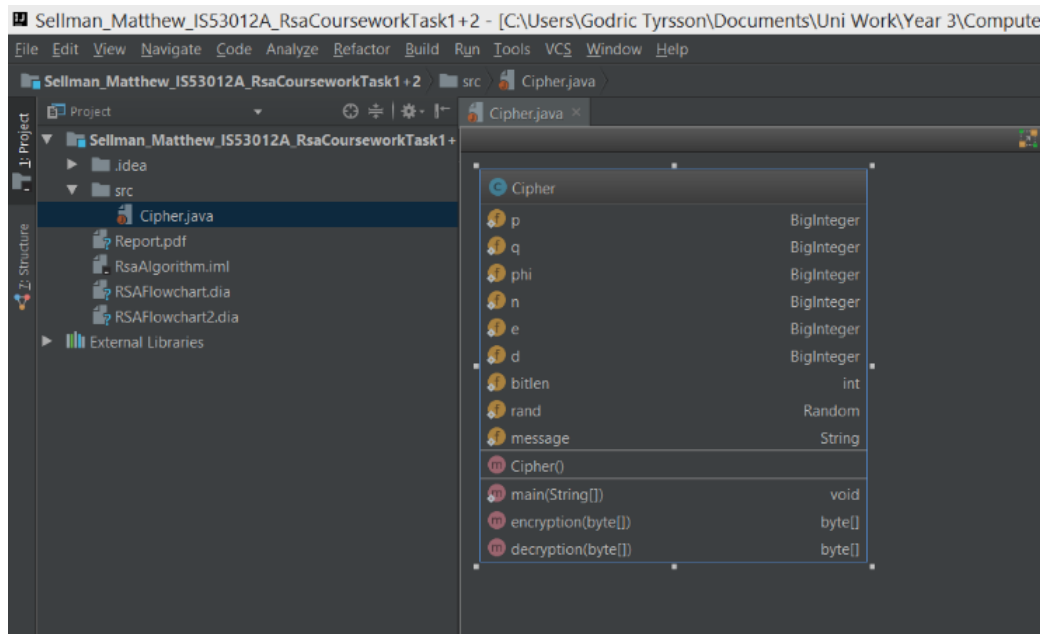
(a) Algorithms (in flow-chart):

In my code I used the RSA algorithm, I did this by first creating a key pair, namely, e and n . To make the values needed for the encrypt I needed a few numbers, the first of which was n . Which was made by multiplying p and q together. Next I needed to make ϕ , or ϕ by $(\phi = \phi(n) = (p-1) \cdot (q-1))$. While e was made by being the co-prime of ϕ and being less than n .

These are then used to encrypt a message by turning that message into bytecode, then multiplying the bytecode by the key pair. This would then be sent to the other user, who would decrypt it using their private key, which would be made from d and n . I made d by using the inverse of e modulo ϕ : $e \cdot d \mod \phi = 1$. So the flowchart will look more like:



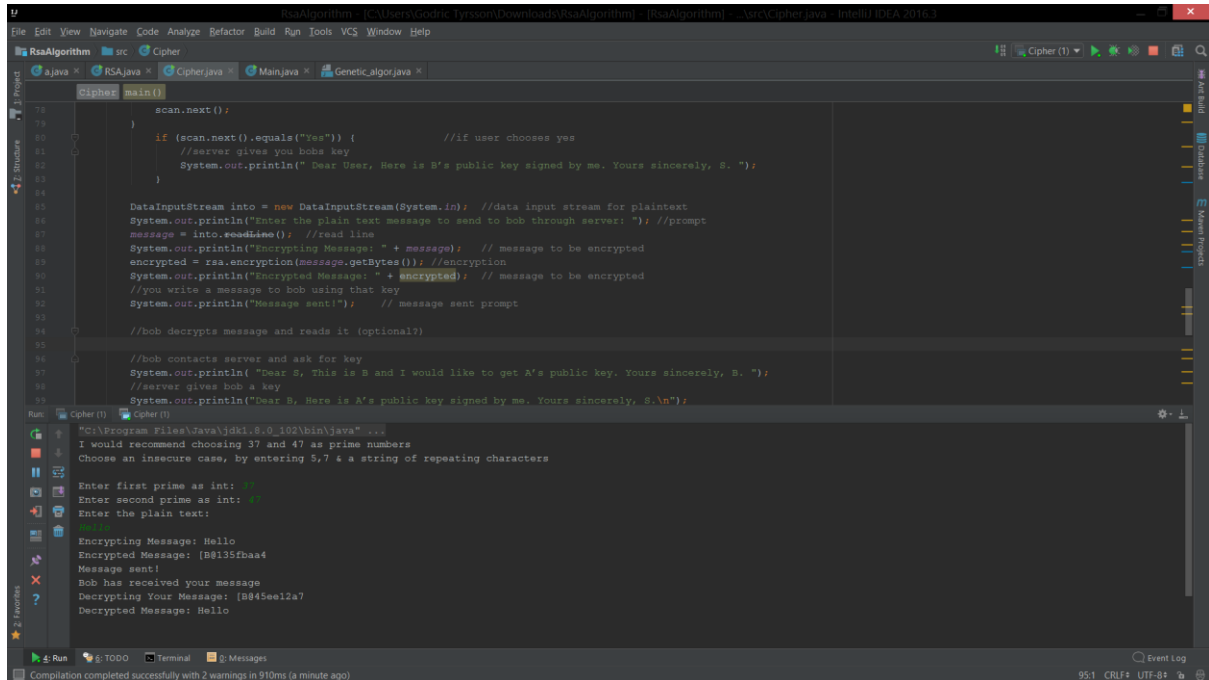
(b) Design (in block diagram or class-diagram in UML)



The main class is the cipher class, with encryption and decryption being two separate objects that interact with the main method. There are also several big integers, one int, one random variable and one string.

(c) Demonstration (in 5 best screen-shots)

Code functioning normally (task 1)



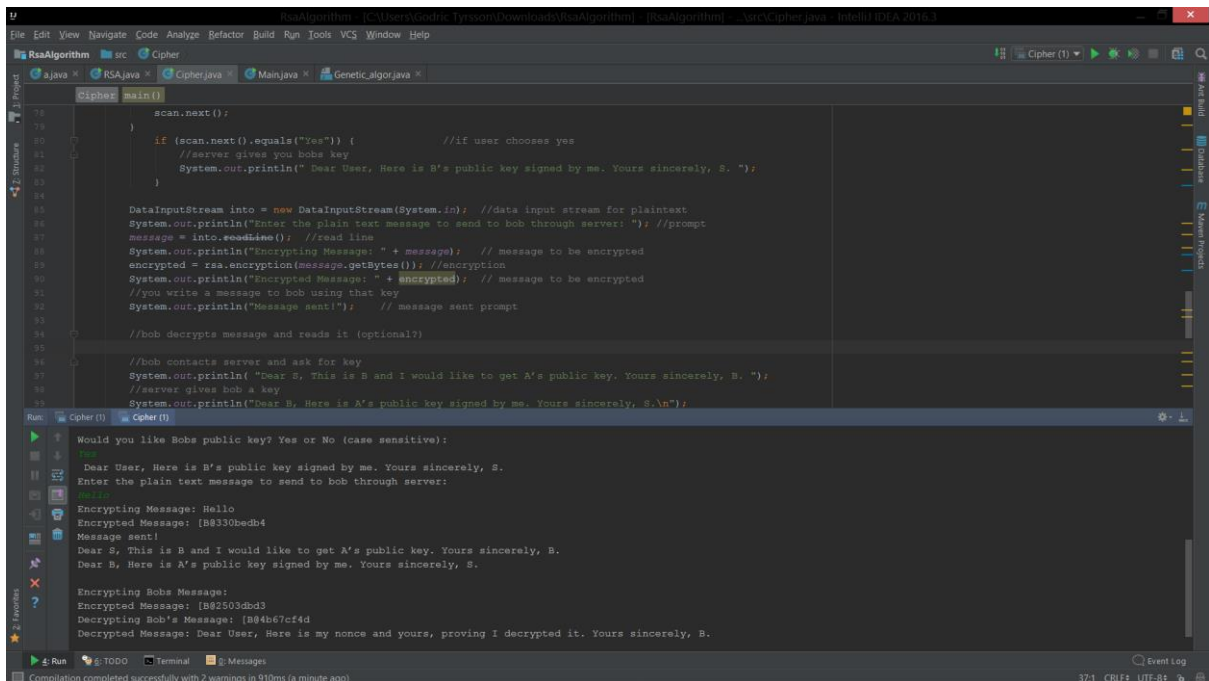
```
78 scan.next();
79 }
80 if (scan.next().equals("yes")) { //if user chooses yes
81     //server gives you bobs key
82     System.out.println(" Dear User, Here is B's public key signed by me. Yours sincerely, S. ");
83 }
84
85 DataInputStream into = new DataInputStream(System.in); //data input stream for plaintext
86 System.out.println("Enter the plain text message to send to bob through server: "); //prompt
87 message = into.readLine(); //read line
88 System.out.println("Encrypting Message: " + message); // message to be encrypted
89 encrypted = rsa.encryption(message.getBytes()); //encryption
90 System.out.println("Encrypted Message: " + encrypted); // message to be encrypted
91 //you write a message to bob using that key
92 System.out.println("Message sent!"); // message sent prompt
93
94 //Bob decrypts message and reads it (optional?)
95
96 //Bob contacts server and ask for key
97 System.out.println(" Dear S, This is B and I would like to get A's public key. Yours sincerely, B. ");
98 //server gives bob a key
99 System.out.println("Dear B, Here is A's public key signed by me. Yours sincerely, S.\n");
```

Run: Cipher (1) Cipher (1)

"C:\Program Files\Java\jdk1.8.0_102\bin\java"....
I would recommend choosing 37 and 47 as prime numbers
Choose an insecure case, by entering 5,7 & a string of repeating characters
Enter first prime as int: 5
Enter second prime as int: 7
Enter the plain text: Hello
Encrypting Message: Hello
Encrypted Message: B0135fbaa4
Message sent!
Bob has received your message
Decrypting Your Message: B045ee12a7
Decrypted Message: Hello

Compilation completed successfully with 2 warnings (a minute ago)

Code functioning normally (task 2)



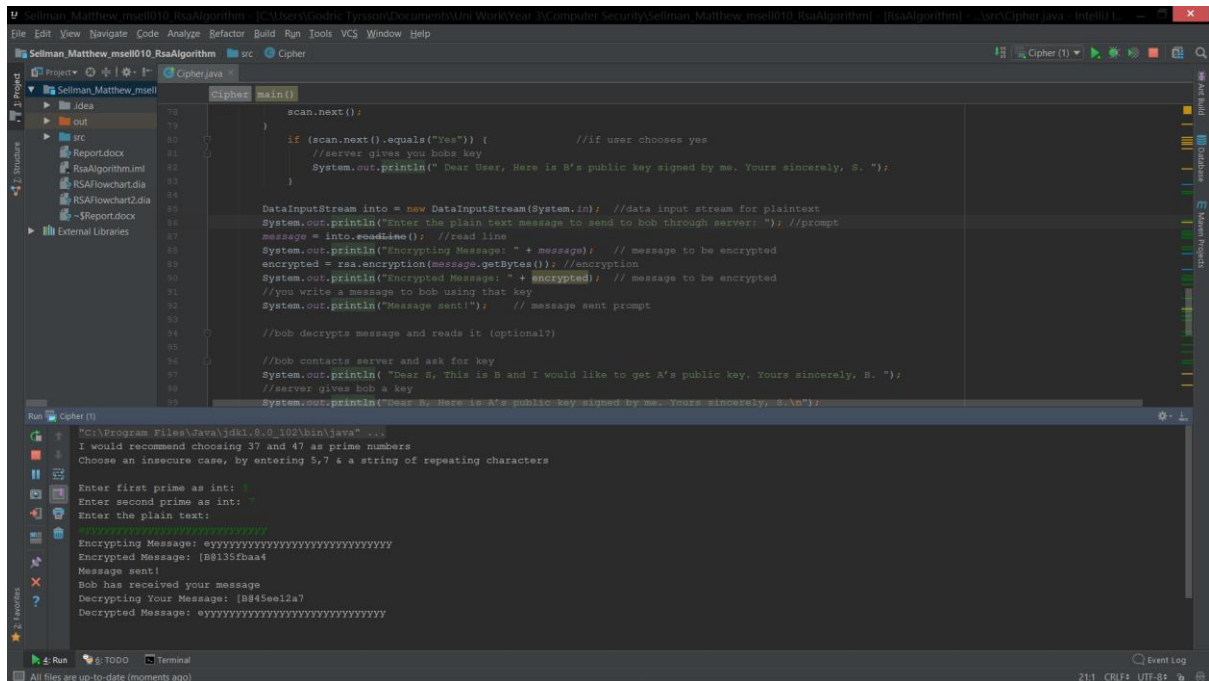
```
78 scan.next();
79 }
80 if (scan.next().equals("yes")) { //if user chooses yes
81     //server gives you bobs key
82     System.out.println(" Dear User, Here is B's public key signed by me. Yours sincerely, S. ");
83 }
84
85 DataInputStream into = new DataInputStream(System.in); //data input stream for plaintext
86 System.out.println("Enter the plain text message to send to bob through server: "); //prompt
87 message = into.readLine(); //read line
88 System.out.println("Encrypting Message: " + message); // message to be encrypted
89 encrypted = rsa.encryption(message.getBytes()); //encryption
90 System.out.println("Encrypted Message: " + encrypted); // message to be encrypted
91 //you write a message to bob using that key
92 System.out.println("Message sent!"); // message sent prompt
93
94 //Bob decrypts message and reads it (optional?)
95
96 //Bob contacts server and ask for key
97 System.out.println(" Dear S, This is B and I would like to get A's public key. Yours sincerely, B. ");
98 //server gives bob a key
99 System.out.println("Dear B, Here is A's public key signed by me. Yours sincerely, S.\n");
```

Run: Cipher (1) Cipher (1)

Would you like Bobs public key? Yes or No (case sensitive): yes
Dear User, Here is B's public key signed by me. Yours sincerely, S.
Enter the plain text message to send to bob through server: Hello
Encrypting Message: Hello
Encrypted Message: B0330bedb4
Message sent!
Dear S, This is B and I would like to get A's public key. Yours sincerely, B.
Dear B, Here is A's public key signed by me. Yours sincerely, S.
Encrypting Bobs Message:
Encrypted Message: B02503dbd3
Decrypting Bob's Message: B04b67cf4d
Decrypted Message: Dear User, Here is my nonce and yours, proving I decrypted it. Yours sincerely, B.

Compilation completed successfully with 2 warnings (a minute ago)

An insecure case (task 1) repeating characters mean that the bytecode and easily be interpreted and the small prime numbers mean that the keys are not large numbers. This means that their origin would only be a small range of numbers and therefore the private and public keys could be computed. (task 1)

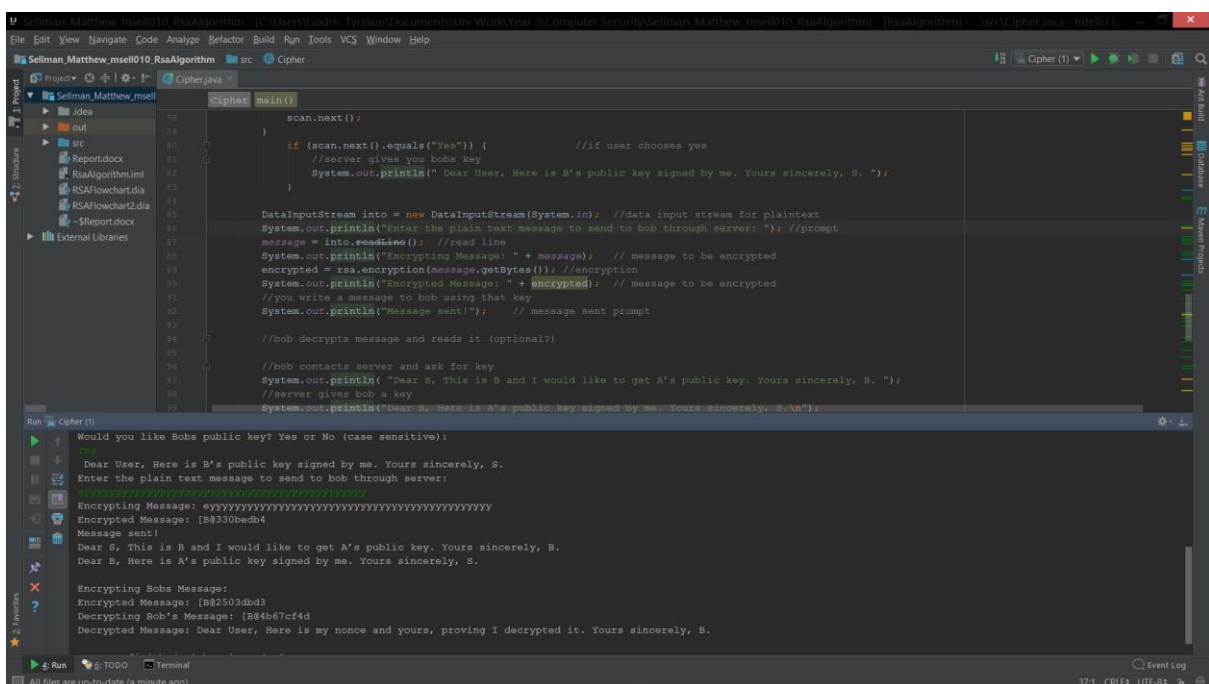


```
70      scan.next();
71    }
72    if (scan.next().equals("Yes")) {
73        //server gives you bobs key
74        System.out.println(" Dear User, Here is B's public key signed by me. Yours sincerely, S. ");
75    }
76
77    DataInputStream into = new DataInputStream(System.in); //data input stream for plaintext
78    System.out.println("Enter the plain text message to send to bob through server: "); //prompt
79    message = into.readLine(); //read line
80    System.out.println("Encrypting Message: " + message); // message to be encrypted
81    encrypted = rsa.encryption(message.getBytes()); //encryption
82    System.out.println("Encrypted Message: " + encrypted); // message to be encrypted
83    //you write a message to bob using that key
84    System.out.println("Message sent!"); // message sent prompt
85
86    //bob decrypts message and reads it (optional?)
87
88    //bob contacts server and ask for key
89    System.out.println("Dear S, This is B and I would like to get A's public key. Yours sincerely, B. ");
90    //server gives bob a key
91    System.out.println("Dear B, Here is A's public key signed by me. Yours sincerely, S.\n");
```

Run Cipher (1)

PC:\Program Files\Java\jdk1.8.0_102\bin\java" ...
I would recommend choosing 37 and 47 as prime numbers
Choose an insecure case, by entering 5,7 & a string of repeating characters
Enter first prime as int: 37
Enter second prime as int: 47
Enter the plain text:
I would recommend choosing 37 and 47 as prime numbers
Encrypting Message: eyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
Encrypted Message: [B0135fbaa4
Message sent!
Bob has received your message
Decrypting Your Message: [B045ee12a7
Decrypted Message: eyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

Below is the insecure case for task 2, the same as above applies, but this time, it is the server that is also in danger, since the composition of a public and private key pair could be used to forge messages that could ask for the public keys and therefore the private keys of individuals.

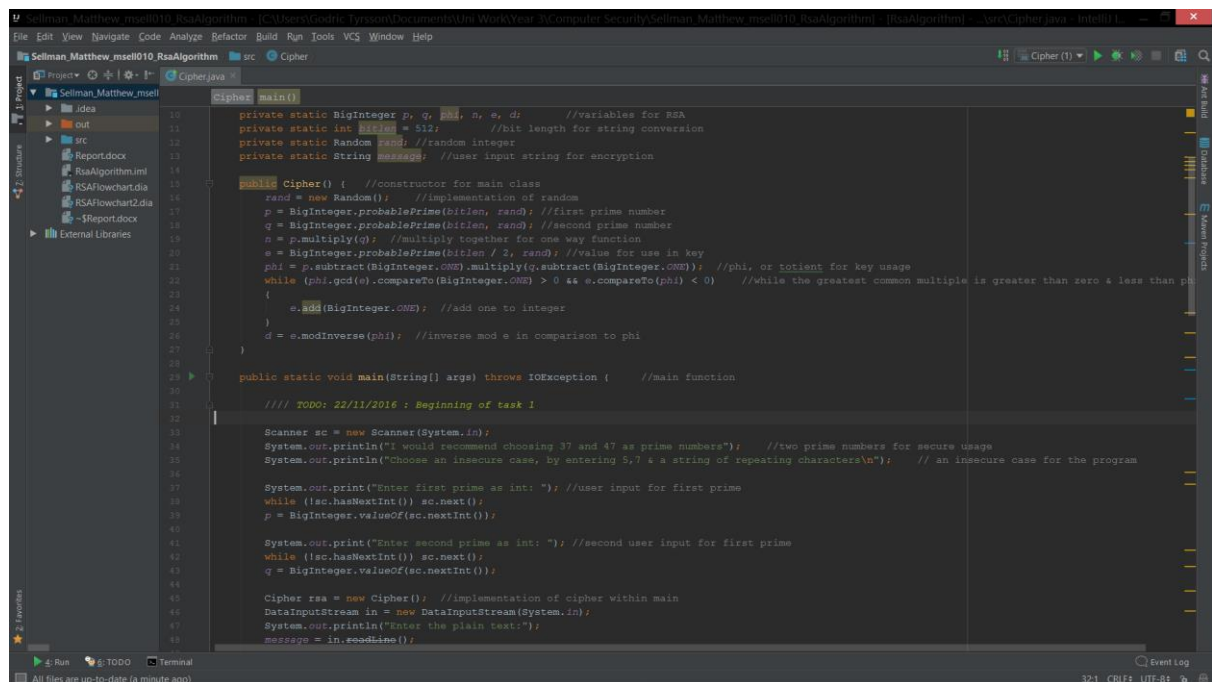


```
70      scan.next();
71    }
72    if (scan.next().equals("Yes")) {
73        //server gives you bobs key
74        System.out.println(" Dear User, Here is B's public key signed by me. Yours sincerely, S. ");
75    }
76
77    DataInputStream into = new DataInputStream(System.in); //data input stream for plaintext
78    System.out.println("Enter the plain text message to send to bob through server: "); //prompt
79    message = into.readLine(); //read line
80    System.out.println("Encrypting Message: " + message); // message to be encrypted
81    encrypted = rsa.encryption(message.getBytes()); //encryption
82    System.out.println("Encrypted Message: " + encrypted); // message to be encrypted
83    //you write a message to bob using that key
84    System.out.println("Message sent!"); // message sent prompt
85
86    //bob decrypts message and reads it (optional?)
87
88    //bob contacts server and ask for key
89    System.out.println("Dear S, This is B and I would like to get A's public key. Yours sincerely, B. ");
90    //server gives bob a key
91    System.out.println("Dear B, Here is A's public key signed by me. Yours sincerely, S.\n");
```

Run Cipher (1)

Would you like Bobs public key? Yes or No (case sensitive):
Dear User, Here is B's public key signed by me. Yours sincerely, S.
Enter the plain text message to send to bob through server:
Dear User, Here is B's public key signed by me. Yours sincerely, S.
Encrypting Message: eyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
Encrypted Message: [B0330bedb4
Message sent!
Dear S, This is B and I would like to get A's public key. Yours sincerely, B.
Dear B, Here is A's public key signed by me. Yours sincerely, S.
Encrypting Bobs Message:
Encrypted Message: [B02503dbd3
Decrypting Bob's Message: [B04b67cf4d
Decrypted Message: Dear User, Here is my nonce and yours, proving I decrypted it. Yours sincerely, B.

Below is a sample of the code I have made, without any processes running.

A screenshot of an IDE window showing a Java file named 'Cipher.java'. The code implements an RSA encryption algorithm. It includes a 'Cipher' class with a constructor that initializes variables for RSA (p, q, n, e, d) and a 'main' method that prompts the user for prime numbers and a message to encrypt. The code uses 'BigInteger' for large numbers and 'Random' for generating primes. Comments are present throughout the code, including a TODO note at the beginning of the main function.

```
10 private static BigInteger p, q, n, e, d; //variables for RSA
11 private static int bitlen = 512; //bit length for string conversion
12 private static Random rand; //random integer
13 private static String message; //user input string for encryption
14
15 public Cipher() { //constructor for main class
16     rand = new Random(); //implementation of random
17     p = BigInteger.probablePrime(bitlen, rand); //first prime number
18     q = BigInteger.probablePrime(bitlen, rand); //second prime number
19     n = p.multiply(q); //multiply together for one way function
20     e = BigInteger.probablePrime(bitlen / 2, rand); //value for use in key
21     phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE)); //phi, or totient for key usage
22     while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0) //while the greatest common multiple is greater than zero & less than phi
23     {
24         e = e.add(BigInteger.ONE); //add one to integer
25     }
26     d = e.modInverse(phi); //inverse mod e in comparison to phi
27 }
28
29 public static void main(String[] args) throws IOException { //main function
30
31     /// TODO: 22/11/2016 : Beginning of task 1
32
33     Scanner sc = new Scanner(System.in);
34     System.out.println("I would recommend choosing 37 and 47 as prime numbers"); //two prime numbers for secure usage
35     System.out.println("Choose an insecure case, by entering 5,7 & a string of repeating characters\n"); // an insecure case for the program
36
37     System.out.print("Enter first prime as int: "); //user input for first prime
38     while (!sc.hasNextInt()) sc.next();
39     p = BigInteger.valueOf(sc.nextInt());
40
41     System.out.print("Enter second prime as int: "); //second user input for first prime
42     while (!sc.hasNextInt()) sc.next();
43     q = BigInteger.valueOf(sc.nextInt());
44
45     Cipher rsa = new Cipher(); //implementation of cipher within main
46     DataInputStream in = new DataInputStream(System.in);
47     System.out.println("Enter the plain text:");
48     message = in.readLine();
49 }
```

(d) Discussion (including answers to any questions/problems in the Coursework assignment, your experience in attempt of the coursework, and full bibliography)

My main problem in the coursework was deciding how I would encrypt the message string given by the user. First I tried to convert the string into the respective ascii codes and try to encrypt those numbers. However, to do this I needed to convert the string to chars and then to numbers and then encrypt. This was too slow and clunky for what I needed. Therefore I decided to convert the string straight into bytecode and from there encrypt the numbers. This proved to be far more successful and much easier to integrate into my program.

Another issue that I had was that the encryption and decryption were the same no matter what input I put into the program. I fixed this by making sure the message variable being encrypted was the user input, rather than a different variable put in at an earlier time. This made the encryption different each time and the bug was removed from the program.

Another interesting problem I came across was the mathematics itself, the problem stemmed from the complication of the mathematical integers used. When I was using ints, longs and doubles, I had to try to add, subtract and multiply values that did not correspond to one another, in order to bypass this I had to turn the value into Bigintegers, this allowed me much more flexibility when it came to computing the individual values given to be computed.

Total hours spent:

Total Number of Hours Spent	60 hours
Hours Spent for Algorithm Design	5 hours
Hours Spent for Programming	20 hours
Hours Spent for Writing Report	15 hours
Hours Spent for Testing	20 hours
Note for the examiner (if any):	My optimal mark would be 75% or above

Bibliography: -

Code done with help from <http://www.sanfoundry.com/java-program-implement-rsa-algorithm/>

Everything else if my own work, as is most of the code.