My program has 4 functions:

- -getRandomBigInt()
  - -Generates/returns a random large integer (100 digit integer)
- -makeOdd(BigInteger input)
  - -Takes a large integer input and returns it as an odd number if it was originally an even number.
- -checkPrime(BigInteger input)
  - -Takes a large integer input and checks if it's prime through the Miller-Rabin primality test. Returns boolean value true for probably prime, false for not prime.
- -badModPow(BigInteger base, BigInteger exponent, BigInteger modulo)
  - -My homemade modPow function. It utilizes the bit shifting hack with exponents to make it fast! Returns the BigInteger value for base^exponent (mod modulo)
- -modInverse(BigInteger a, BigInteger b)
  - -Finds the inverse modulo of a mod b. Does it by going through the extended Euclid algorithm and saving each step through separate additional variables.

My main program uses these functions to generate two large prime numbers p and q (called primeNumber1 and primeNumber2), to help generate the public and private key.

More specifically: It runs a while loop to continuously check randomly generated odd 100 digit values for primality until it gets two different prime numbers.

I verified that my code worked by checking the large generated numbers with an online primality test (https://www.numberempire.com/primenumbers.php), and an online inverse mod calculator (https://www.boxentriq.com/code-breaking/modular-multiplicative-inverse).

List of problems/solutions:

-I'm pretty unfamiliar with Java so implementing BigInteger values so I could generate large integers took me so long to figure out. But I read the documentation and learned it.

-My initial code to check whether a large number was prime was way too inefficient and never ended up being able to load an answer, so I had to smarten up my algorithms and use the Miller-Rabin method (which I was initially less familiar with) to hasten the computation.

-I forgot how to do inverse modulo arithmetic so I had to relearn it in order to translate it into code for the computer to run. Also, I had to figure out a way to store the equations efficiently so it could hold all the coefficients so it could make the right inverse.

To compile (in java):

```
> javac Assignment6.java
```

To execute (in java):

```
> java Assignment6
```