Prgm 11 rsa algorithm

```
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
 public class RSA { private BigInteger
p;
 private BigInteger q;
 private BigInteger N;
 private BigInteger phi;
 private BigInteger e;
private BigInteger d;
 private int bitlength = 1024;
private int blocksize = 256;
 private Random r; public RSA() {
r = new Random();
 p =
BigInteger.probablePrime(bitlength,
r);
 q =
BigInteger.probablePrime(bitlength,
r);
 N = p.multiply(q);
 phi =
p.subtract(BigInteger.ONE).multiply(
q.subtract(BigInteger.ONE));
 e =
BigInteger.probablePrime(bitlength/
2, r);
 while
(phi.gcd(e).compareTo(BigInteger.O
NE) > 0 && e.compareTo(phi) < 0 )
 { e.add(BigInteger.ONE);
}
 d = e.modInverse(phi);
 }
public RSA(BigInteger e, BigInteger
d, BigInteger N)
{
 this.e = e;
this.d = d;
 this.N = N;
 }
public static void main (String[] args)
throws IOException {
 RSA rsa = new RSA();
DataInputStream in=new
DataInputStream(System.in);
 String teststring ;
 System.out.println("Enter the plain
text:");
teststring=in.readLine();
 System.out.println("Encrypting
String: " + teststring);
System.out.println("String in Bytes: "
+
bytesToString(teststring.getBytes()));
byte[] encrypted =
rsa.encrypt(teststring.getBytes());
 System.out.println("Encrypted
String in Bytes: " +
bytesToString(encrypted));
byte[] decrypted =
rsa.decrypt(encrypted);
System.out.println("Decrypted String
in Bytes: " +
bytesToString(decrypted));
System.out.println("Decrypted
String: " + new String(decrypted));
}
 private static String
bytesToString(byte[] encrypted) {
 String test = "";
 for (byte b : encrypted) {
 test += Byte.toString(b);
}
return test;
 }
public byte[] encrypt(byte[]
message) {
return (new
BigInteger(message)).modPow(e,
N).toByteArray();
}
public byte[] decrypt(byte[]
message) {
return (new
BigInteger(message)).modPow(d,
N).toByteArray();
 } }
```