// RSA.java

// Performs RSA Encryption

// Coded by LamyaAlShammary, Supervisor Ms.Saba

**import** java.math.\*;

**import** java.lang.\*;

**import** java.util.\*;

**import** java.security.\*;

**class** rsa

{

 // Variables - p and q (two prime numbers)

 **static** BigInteger *p*;

 **static** BigInteger *q*;

 **static** BigInteger *squared*;

 // generatePrime

 // -------------------------

 // Creates a (probably) prime 10-bit number

 **public** **static** BigInteger generatePrime()

 {

 // Random num generator

 Random generator = **new** SecureRandom();

 // Create a 'random' prime number

 BigInteger p = BigInteger.*probablePrime*(10, generator);

 **return** p;

 }

 // RSA

 // ------------------------

 // Takes in a message and two booleans (to determine whether or not to use

 // optimizing functions). Runs RSA to encrypt and decrypt the message.

 **public** **static** **void** RSA(String message, **boolean** square, **boolean** china)

 {

 // Step 1) Create two prime numbers, p and q

 *p* = *generatePrime*();

 *q* = *generatePrime*();

 // Step 2) n = pq

 BigInteger n = *p*.multiply(*q*);

 // Step 3) Find phi(n), quantity of numbers relatively prime to n

 BigInteger phi = *p*.subtract(BigInteger.*ONE*).multiply(*q*.subtract(BigInteger.*ONE*));

 // Step 4) Find e, the public key

 // find random number between 1 and phi

 Random generator = **new** Random();

 // make sure the int value is positive

 **int** phiint = phi.intValue();

 **if**(phiint < 0)

 {

 phiint \*= -1;

 }

 **int** rand = generator.nextInt(phiint) + 1;

 // make the random number odd

 **if**(rand % 2 == 0)

 {

 rand++;

 }

 // convert random number into a big integer

 String rands = Integer.*toString*(rand);

 BigInteger e = **new** BigInteger(rands);

 **while**( e.gcd(phi).intValue() > 1)

 {

 e = e.subtract( BigInteger.*ONE* );

 e = e.subtract( BigInteger.*ONE* );

 }

 // Step 5) Compute the secret key, d

 BigInteger d = e.modInverse(phi);

 // Chinese Remainder values

 BigInteger dp = d.mod(*p*.subtract(BigInteger.*ONE*));

 BigInteger dq = d.mod(*q*.subtract(BigInteger.*ONE*));

 BigInteger qinv = *q*.modInverse(*p*);

 System.*out*.println(" p = " + *p* + "\n q = " + *q*);

 System.*out*.println("Public keys:\n n = " + n + "\n e = " + e + "\nPrivate key, d = " + d);

 // Step 6) Encryption

 BigInteger encryption;

 **if**(square)

 {

 encryption = *encryptSquare*(message,e,n);

 System.*out*.println("Encrypted with Exponentiation by squaring: " + encryption);

 }

 **else**

 {

 encryption = *encrypt*(message,e,n);

 System.*out*.println("Encrypted: " + encryption);

 }

 // Step 7) Decryption

 **if**(china)

 System.*out*.println("Decrypted with Chinese Remainder: " + *chineseRemainder*(encryption, dp, dq, qinv));

 **else**

 System.*out*.println("Decrypted: " + *decrypt*(encryption, d, n));

 }

 // stringToNumber

 // ---------------------------

 // Converts a string to an BigInteger

 **public** **static** BigInteger stringToNumber(String str)

 {

 String number = "";

 **for**(**int** i = 0; i < str.length(); i++)

 {

 **char** c = str.charAt(i);

 **int** j = (**int**) c;

 number += j;

 }

 BigInteger b = **new** BigInteger(number);

 **return** b;

 }

 // numberToString

 // ---------------------------

 // Converts a BigInteger to a string

 **public** **static** String numberToString(BigInteger b)

 {

 String number = b.toString();

 String word = "";

 **for**(**int** i = 0; i < number.length(); i+= 2)

 {

 **char** c1 = number.charAt(i);

 **char** c2 = number.charAt(i + 1);

 // ASCII value as a string

 String letter = "";

 letter += c1;

 letter += c2;

 // ASCII value as a integer

 **int** ascii = Integer.*parseInt*(letter);

 // convert ascii value into a letter

 **char** c = (**char**) ascii;

 // Add the letter into the word

 word += c;

 }

 **return** word;

 }

 // encrypt

 // -------------------------

 // encrypts a string of text

 **public** **static** BigInteger encrypt(String message, BigInteger e, BigInteger n)

 {

 // Convert message to a number

 BigInteger number = *stringToNumber*(message);

 // compute the encrypted number

 BigInteger c = (number.pow(e.intValue())).mod(n);

 **return** c;

 }

 // exponentSquare

 // --------------------

 // calculates a power faster through exponentiation by squares

 **public** **static** BigInteger exponentSquare(BigInteger x, BigInteger n)

 {

 BigInteger TWO = BigInteger.*ONE*.add(BigInteger.*ONE*);

 // If exponent is zero, result is one

 **if**(n.compareTo(BigInteger.*ZERO*) == 0)

 {

 *squared* = BigInteger.*ONE*;

 }

 // n < 0

 **else** **if** (n.compareTo(BigInteger.*ZERO*) == -1)

 {

 *squared* = BigInteger.*ONE*.divide(*exponentSquare* (x, n.negate()));

 }

 // n is even

 **else** **if** (n.mod(TWO).compareTo(BigInteger.*ZERO*) == 0)

 {

 *squared* = *exponentSquare*(x, n.divide(TWO)).pow(2);

 }

 // n is odd

 **else**

 {

 *squared* = x.multiply(*exponentSquare*(x, n.subtract(BigInteger.*ONE*).divide(TWO)).pow(2));

 }

 **return** *squared*;

 }

// encryptSquare

// -------------------------

// encrypts the message faster using exponentiation by squares

// now should run in log2(n)

 **public** **static** BigInteger encryptSquare(String message, BigInteger e, BigInteger n)

 {

 // Convert message to a number

 BigInteger number = *stringToNumber*(message);

 // compute the encrypted number

 BigInteger c = *exponentSquare*(number,e).mod(n);

 **return** c;

 }

 // decrypt

 // -------------------------

 // decrypts an encrypted message

 **public** **static** String decrypt(BigInteger cypher, BigInteger d, BigInteger n)

 {

 // Unencrypted BigInteger representation

 BigInteger uncode = (cypher.pow(d.intValue())).mod(n);

 // Unencrypted string

 String done = *numberToString*(uncode);

 **return** done;

 }

 // chineseRemainder

 // ---------------------------

 // makes encryption and decryption more efficient

 **public** **static** String chineseRemainder(BigInteger cypher, BigInteger dp, BigInteger dq, BigInteger qinv)

 {

 BigInteger m1 = (cypher.pow(dp.intValue())).mod(*p*);

 BigInteger m2 = (cypher.pow(dq.intValue())).mod(*q*);

 BigInteger h;

 **if**(m1.compareTo(m2) == 0 || m1.compareTo(m2) == 1)

 {

 h = (qinv.multiply(m1.subtract(m2))).mod(*p*);

 }

 **else**

 {

 h = (qinv.multiply(m1.add(*p*).subtract(m2))).mod(*p*);

 }

 BigInteger m = m2.add(h.multiply(*q*));

 // Convert back into a string

 String message = *numberToString*(m);

 **return** message;

 }

 // Main method

 **public** **static** **void** main(String args[])

 {

 System.*out*.println("Test 1, Input Message: HI\n~~~");

 *RSA*("HI",**true**,**true**);

 System.*out*.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~");

 System.*out*.println("Test 2, Input Message: NO\n~~~");

 *RSA*("NO",**false**,**false**);

 }

}