Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической теории упругости и биомеханики

Отчет по дисциплине
«Математические основы информационной безопасности»

студенток 4 курса 451 группы

направления 38.03.05 - Бизнес-информатика

механико-математического факультета

Слаповской Ирины Романовны, Зима Кристины Витальевны,
Перекрестовой Анны Аркадьевны

ассистент Дмитриев П.О.

Саратов 2018

1 Описание практической работы

1.1 Постановка задачи

Вариант 12.

Необходимо реализовать алгоритм RSA. При реализации использовать стороннюю библиотеку для работы с большими числами.

1.2 Описание алгоритма RSA

В основе RSA лежит задача факторизации произведения двух простых больших чисел. Для шифрования используется простая операция возведения в степень по модулю N. Для дешифрования же необходимо вычислить функцию Эйлера от числа N. для этого необходимо знать разложение числа n на простые множители (В этом состоит задача факторизации).

В RSA открытый и закрытый ключ состоит из пары целых чисел. Закрытый ключ хранится в секрете, а открытый ключ сообщается другому участнику, либо где-то публикуется.

1.3 Пошаговое описание алгоритма RSA

I. Генерация ключей RSA

Всё начинается с генерации ключевой пары (открытый, закрытый ключ). Генерация ключей в RSA осуществляется следующим образом:

1. Выбираются два простых числа p и q (такие, что p ≠ q).

2. Вычисляется модуль $n = p * q$.

3. Вычисляется значение функции Эйлера модуля n:

$$\varphi(n) = (p - 1)(q - 1).$$

4. Выбирается число e, называемое открытой экспонентой, число e должно лежать в интервале 1<e< φ (n), а так же быть взаимно простым со значением функции φ (n).

5. Вычисляется число d, называемое секретной экспонентой, такое, что $d * e = 1(mod\ \varphi\ (n))$, то есть является мультипликативно обратное к числу e по модулю $\varphi\ (n)$.

Итак, мы получили пару ключей:

Пара (e, n) - открытый ключ (public). Пара (d, n) - закрытый ключ (private).

II. Шифрование и дешифрование в RSA

$\boldsymbol{M}$ – исходное сообщение;

$\boldsymbol{M'}$- зашифрованное сообщение.

Шифрование: $M' = M^e \text{mod n}$

Дешифрование: $M = M'^d \ mod \ n$

2 Ход работы

2.1 Описание программы

Исходный код программы представлен в Приложении. Программа написана на языке Java 8 SE. Программа предназначена для генерации открытого и закрытого ключей по алгоритму RSA, шифрования при помощи открытого ключа, расшифрования при помощи закрытого ключа. Модули генерации ключей, шифрования и дешифрования могут использоваться как самостоятельные приложения.

UML диаграмма классов программы, построенная при помощи PlantUML Web server (http://www.plantuml.com/plantuml/uml/), изображена на рисунке 2.1 и рисунке 2.2. Код, генерирующий диаграмму, представлен в Приложении.

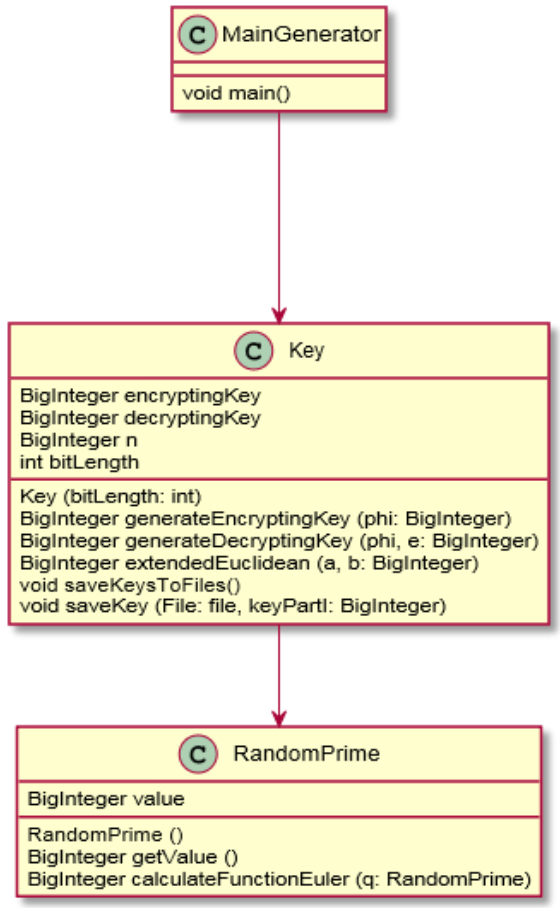

Рисунок 2.1 - UML диаграмма классов модуля генерации ключей

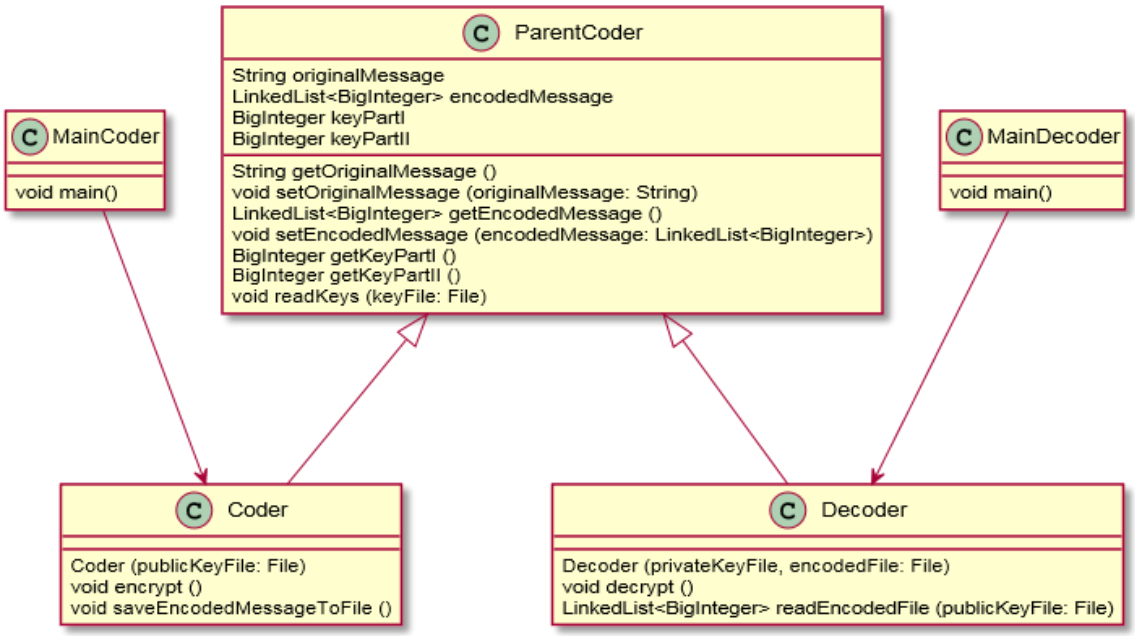

Рисунок 2.2 - UML диаграмма классов модулей шифрования и дешифрования

2.2 Пример работы приложения

Первым шагом работы с программой запускаем генерацию ключей RSA, которая создаёт файлы, содержащие открытый и закрытый ключи. Данный шаг изображен на рисунке 2.3 и рисунке 2.4. Скриншоты процесса шифрования и дешифрования представлены на рисунках 2.5-2.7.

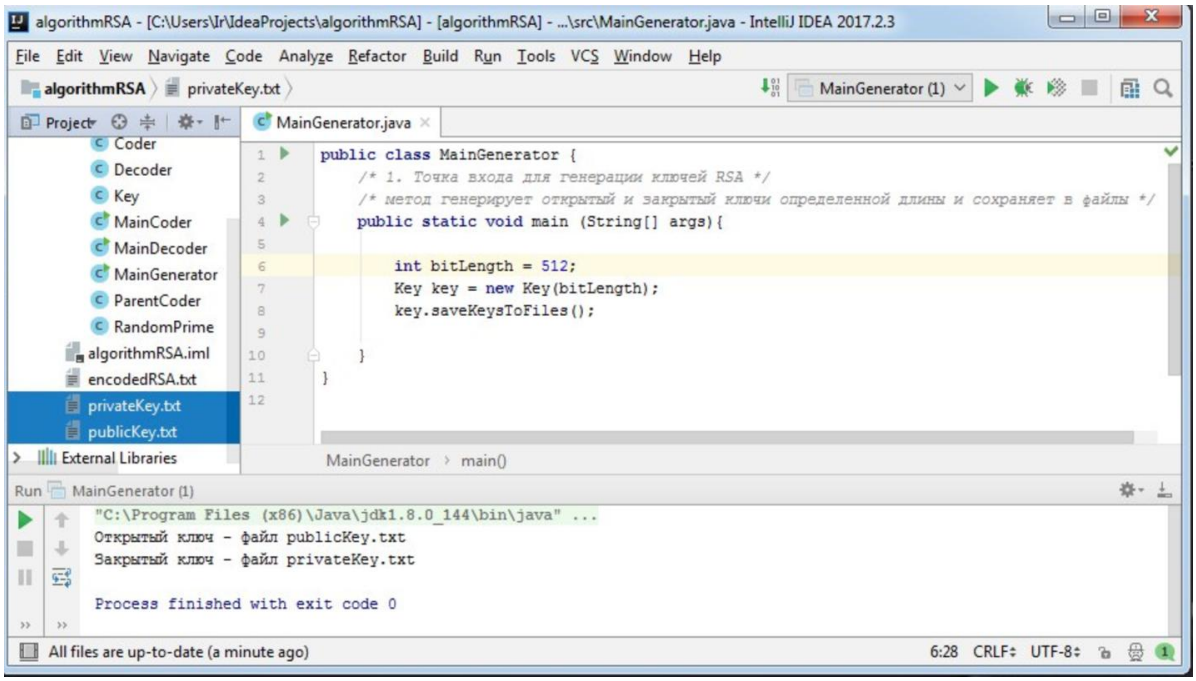

Рисунок 2.3 - Генерация ключей

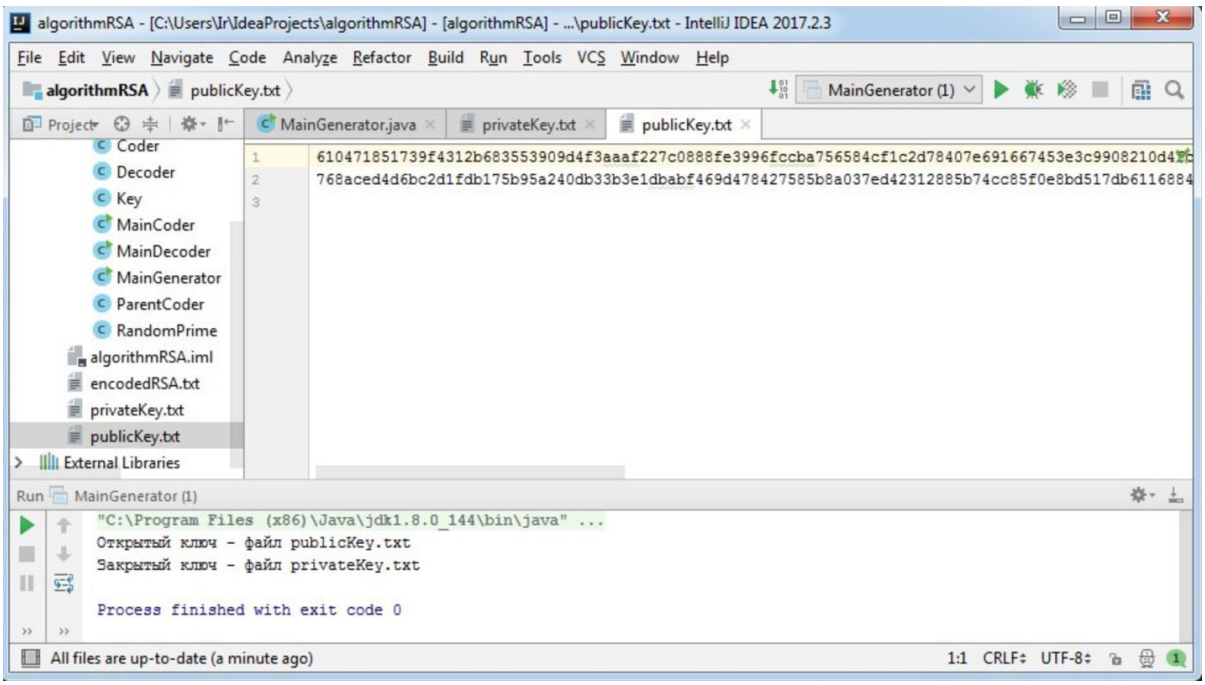

Рисунок 2.4 - Содержание файла с открытым ключом

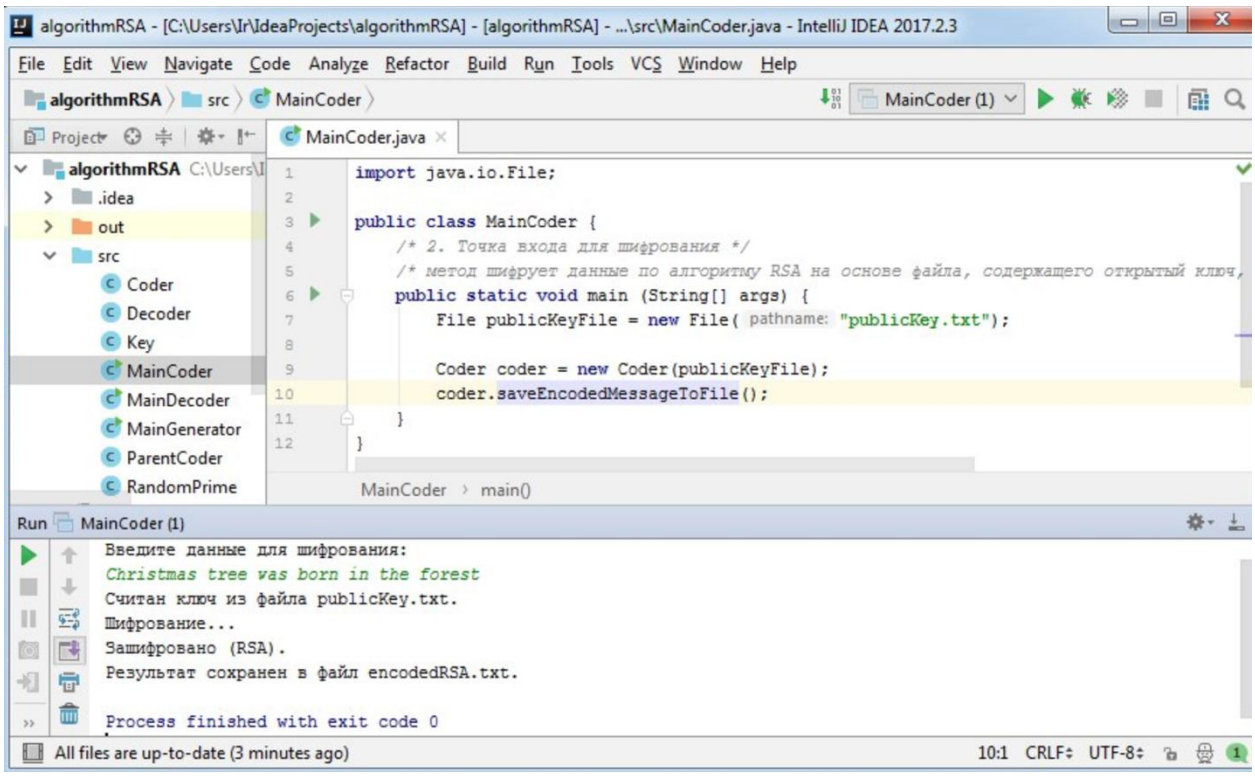

Рисунок 2.5 - Процесс шифрования

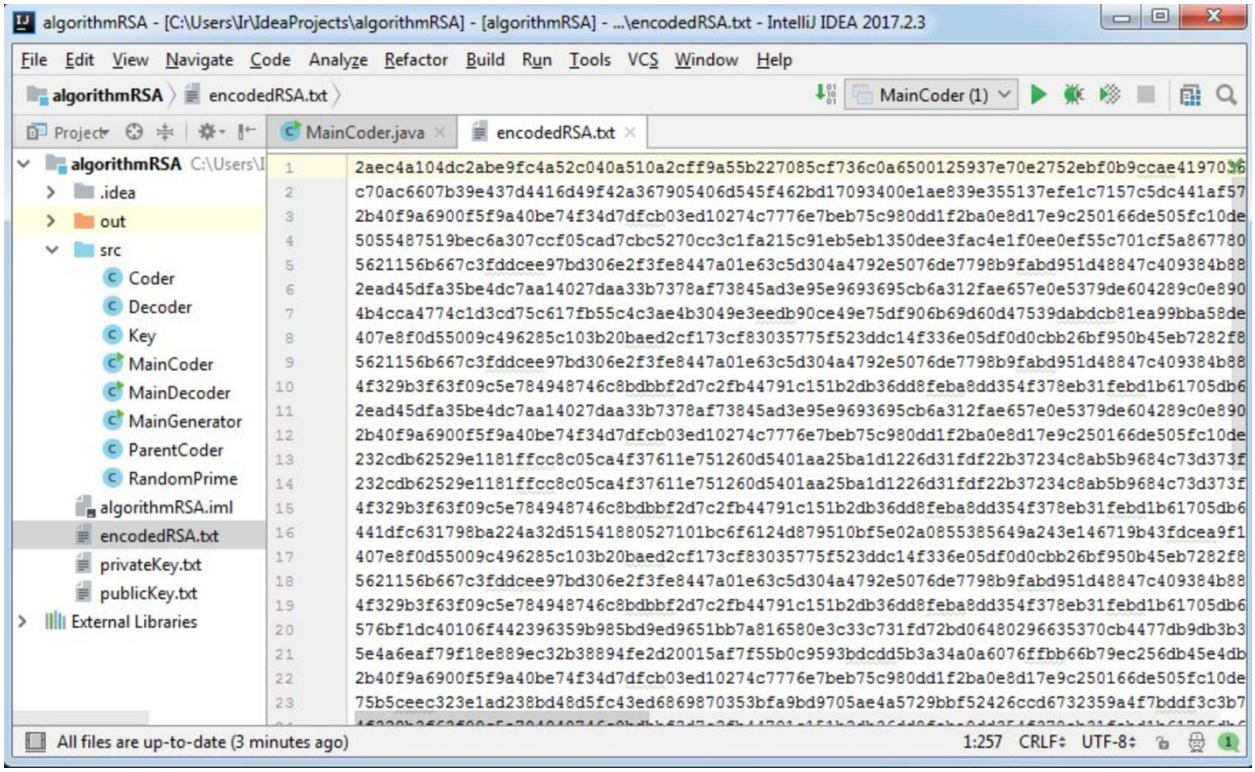

Рисунок 2.6 - Содержание зашифрованного файла

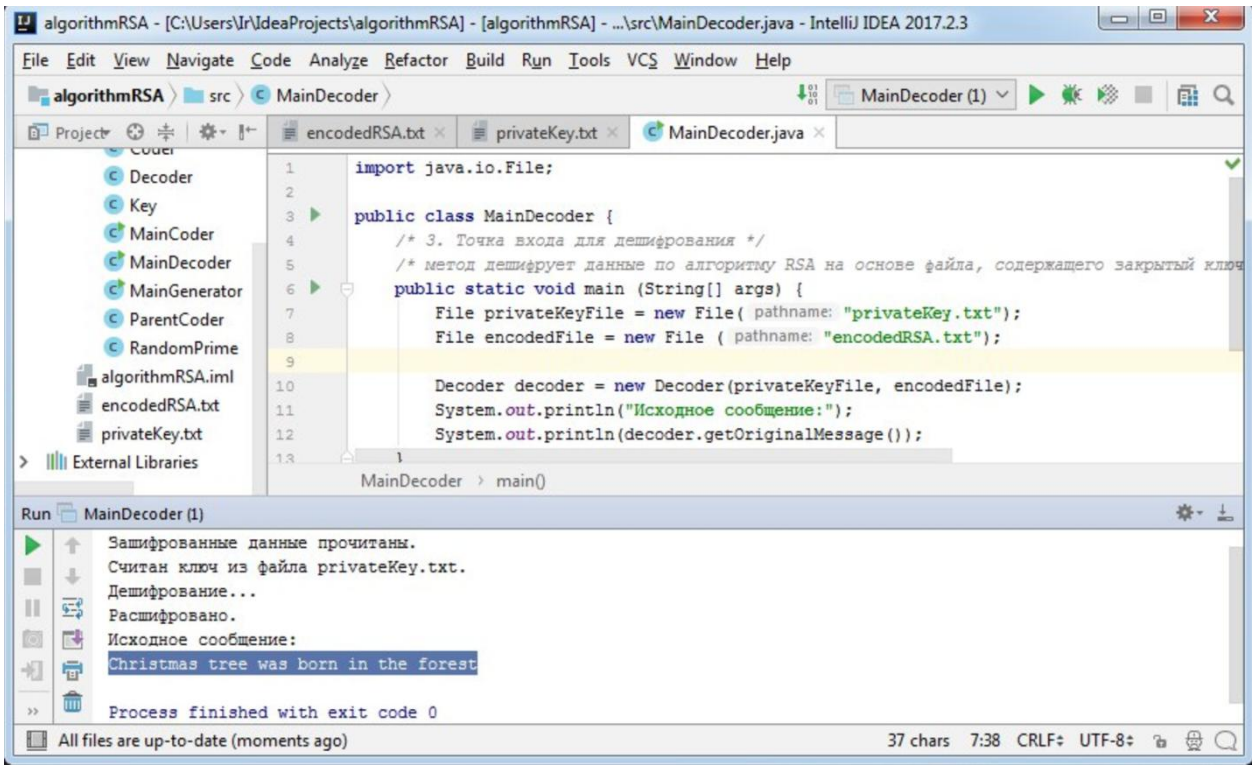

Рисунок 2.7 - Дешифрование

ПРИЛОЖЕНИЕ А

Код построения UML диаграммы классов программы

```
@startuml
class MainGenerator{
  void main()
}
class MainCoder{
  void main()
}
class MainDecoder{
  void main()
}
class Key{
  BigInteger encryptingKey
  BigInteger decryptingKey
  BigInteger n
  int bitLength
  Key (bitLength: int)
  BigInteger generateEncryptingKey (phi: BigInteger)
  BigInteger generateDecryptingKey (phi, e: BigInteger)
  BigInteger extendedEuclidean (a, b: BigInteger)
  void saveKeysToFiles()
  void saveKey (File: file, keyPartI: BigInteger)
}
class RandomPrime{
  BigInteger value
  RandomPrime ()
  BigInteger getValue ()
  BigInteger calculateFunctionEuler (q: RandomPrime)
}
class ParentCoder {
  String originalMessage
  LinkedList<BigInteger> encodedMessage
  BigInteger keyPartI
  BigInteger keyPartII
String getOriginalMessage ()
void setOriginalMessage (originalMessage: String)
LinkedList<BigInteger> getEncodedMessage ()
```

```
void setEncodedMessage (encodedMessage: LinkedList<BigInteger>)
BigInteger getKeyPartI ()
BigInteger getKeyPartII ()
void readKeys (keyFile: File)
}
class Coder {
  Coder (publicKeyFile: File)
  void encrypt ()
  void saveEncodedMessageToFile ()
}
class Decoder {
  Decoder (privateKeyFile, encodedFile: File)
  void decrypt ()
  LinkedList<BigInteger> readEncodedFile (publicKeyFile: File)
}
ParentCoder <|-- Coder
ParentCoder <|-- Decoder
MainCoder --> Coder
MainDecoder --> Decoder
MainGenerator --> Key
Key --> RandomPrime
@enduml
```

ПРИЛОЖЕНИЕ Б

Исходный код программы

Файл «MainGenerator.java» - генерация ключей

```
public class MainGenerator {
   /* 1. Точка входа для генерации ключей RSA */
   /* метод генерирует открытый и закрытый ключи определенной длины и сохраняет в файлы */
   public static void main (String[] args){

      int bitLength = 512;
      Key key = new Key(bitLength);
      key.saveKeysToFiles();

   }
}
```

Файл «Key.java»

```
import java.io.*;
import java.util.Random;
import java.math.BigDecimal;
import java.math.BigInteger;

public class Key {
   private BigInteger encryptingKey;
   private BigInteger decryptingKey;
   private BigInteger n;
   private int bitLength;

   /* конструктор экземпляра класса Key, генерирующий открытый и закрытый ключи */
   public Key(int bitLength){
      this.bitLength = bitLength;
      RandomPrime p = new RandomPrime(bitLength);
      RandomPrime q = new RandomPrime(bitLength);
      BigInteger phi = p.calculateFunctionEuler(q);
      this.n = p.getValue().multiply(q.getValue());
      this.encryptingKey = generateEncryptingKey(phi);
      this.decryptingKey = generateDecryptingKey(phi, encryptingKey);

   }

   /* метод-генератор константы e */
   private BigInteger generateEncryptingKey (BigInteger phi) {
```

```
        BigInteger e;
        do {
            Random random = new Random();
            e = new BigInteger(bitLength + bitLength, random);
        }
        while(e.compareTo(BigInteger.ONE)<0 || e.compareTo(phi)>0 || !e.gcd(phi).equals(BigInteger.ONE));
        return e;
    }

    /* метод-генератор константы d */
    private BigInteger generateDecryptingKey (BigInteger phi, BigInteger e) {
        return extendedEuclidean(phi,e);
    }

    /* метод, реализующий расширенный алгоритм Евклида, с целью вычисления секретной экспоненты d */
    private BigInteger extendedEuclidean (BigInteger a, BigInteger b) {

        BigDecimal a1 = new BigDecimal(a);
        BigDecimal a0 = new BigDecimal(a);
        BigDecimal b0 = new BigDecimal(b);
        BigDecimal t0 = BigDecimal.ZERO;
        BigDecimal t = BigDecimal.ONE;
        BigDecimal div = a0.divide(b0, BigDecimal.ROUND_FLOOR);
        BigDecimal r = a0.subtract(div.multiply(b0));

        while(r.compareTo(BigDecimal.ZERO)>0){
            BigDecimal temp = t0.subtract(div.multiply(t)).remainder(a1);
            t0 = t;
            t = temp;
            a0 = b0;
            b0 = r;
            div = a0.divide(b0, BigDecimal.ROUND_FLOOR);
            r = a0.subtract(div.multiply(b0));
        }
        return t.toBigInteger();
    }

    /* метод, создающий файлы, содержащие открытый и закрытый ключи*/
    public void saveKeysToFiles(){
        File publicKeyFile = new File("publicKey.txt");
        File privateKeyFile = new File("privateKey.txt");
        saveKey(publicKeyFile, encryptingKey);
        saveKey(privateKeyFile, decryptingKey);
```

```
            System.out.println("Открытый ключ - файл publicKey.txt");
            System.out.println("Закрытый ключ - файл privateKey.txt");
        }

        /* метод, записывающий значение ключа в файл */
        private void saveKey (File file, BigInteger keyPartI) {
            try (FileOutputStream fos = new FileOutputStream(file);
                 PrintStream ps = new PrintStream(fos)) {
                ps.println(keyPartI.toString(16));
                ps.println(this.n.toString(16));
            }
            catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
```

Файл «RandomPrime.java»

```
    import java.math.BigInteger;
    import java.util.Random;

    public class RandomPrime{
        private BigInteger value;

        public BigInteger getValue() {
            return value;
        }

        /* конструктор экземпляра класса и присвоение полю значения рандомного простого числа  */
        public RandomPrime(int bitLength) {
            Random random = new Random();
            value = new BigInteger(bitLength,50, random);
                // встроенный тест Миллера-Рабина на простоту
        }

        /* метод, вычисляющий функцию Эйлера от двух простых чисел */
        public BigInteger calculateFunctionEuler(RandomPrime q){
            BigInteger firstMultiplier = this.value.subtract(BigInteger.ONE);
            BigInteger secondMultiplier = q.value.subtract(BigInteger.ONE);
            return firstMultiplier.multiply(secondMultiplier);
        }
    }
```

Файл «MainCoder» – шифрование

```
import java.io.File;

public class MainCoder {
   /* 2. Точка входа для шифрования */
   /* метод шифрует данные по алгоритму RSA на основе файла, содержащего открытый ключ, и сохраняет результат в файл */
   public static void main (String[] args) {
      File publicKeyFile = new File("publicKey.txt");

      Coder coder = new Coder(publicKeyFile);
      coder.saveEncodedMessageToFile();
   }
}
```

Файл «ParentCoder.java»

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
import java.math.BigInteger;


public class ParentCoder {
   private String originalMessage;
   private LinkedList<BigInteger> encodedMessage;

   private BigInteger keyPartI;
   private BigInteger keyPartII;

   public String getOriginalMessage() {
      return originalMessage;
   }

   public void setOriginalMessage(String originalMessage) {
      this.originalMessage = originalMessage;
   }

   public LinkedList<BigInteger> getEncodedMessage() {
      return encodedMessage;
   }

   public void setEncodedMessage(LinkedList<BigInteger> encodedMessage) {
      this.encodedMessage = encodedMessage;
```

```
    }

    public BigInteger getKeyPartI() {
        return keyPartI;
    }

    public BigInteger getKeyPartII() {
        return keyPartII;
    }

    /* метод устанавливает значение ключа, считав его из файла */
    void readKeys (File keyFile) {
        List<BigInteger> publicKey = new ArrayList<>();
        Scanner scanner;
        try {
            int i = 0;
            scanner = new Scanner(keyFile);
            while (i < 2 && scanner.hasNextLine()) {
                String line = scanner.nextLine();
                BigInteger value = new BigInteger(line, 16);
                publicKey.add(value);
                i++;
            }
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        this.keyPartI = publicKey.get(0);
        this.keyPartII = publicKey.get(1);
        System.out.println("Считан ключ из файла " + keyFile + ".");
    }
}
```

Файл «Coder.java»

```
import java.io.*;
import java.util.LinkedList;
import java.util.Scanner;
import java.math.BigInteger;

public class Coder extends ParentCoder {

    /* конструктор экземпляра класса Coder */
    public Coder (File publicKeyFile){
        Scanner in = new Scanner(System.in);
        System.out.println("Введите данные для шифрования:");
```

```
            setOriginalMessage(in.nextLine());
            readKeys(publicKeyFile);
            encrypt();
        }

        /* метод, осуществляющий шифрование по алгоритму RSA */
        private void encrypt () {
            System.out.println("Шифрование...");
            char[] symbolsOriginalMessage = getOriginalMessage().toCharArray();
            LinkedList<BigInteger> symbolsResultData = new LinkedList<>();
            int i;
            int n = symbolsOriginalMessage.length;
            for (i = 0; i < n; i++) {
                int code = (int) symbolsOriginalMessage[i];
                BigInteger newSymbol =
BigInteger.valueOf(code).modPow(getKeyPartI(),getKeyPartII());
                symbolsResultData.add(newSymbol);
            }
            setEncodedMessage(symbolsResultData);
            System.out.println("Зашифровано (RSA).");
        }

        /* метод сохраняет зашифрованные данные в файл */
        public void saveEncodedMessageToFile (){
            try (FileOutputStream fos = new FileOutputStream("encodedRSA.txt");
                 PrintStream printStream = new PrintStream(fos)) {
                for (BigInteger hex : getEncodedMessage()) {
                    printStream.println(hex.toString(16));
                    //!(radix:16)
                }
                System.out.println("Результат сохранен в файл encodedRSA.txt.");
            }
            catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
```

Файл «MainDecoder» - дешифрование

```
import java.io.File;

public class MainDecoder {
    /* 3. Точка входа для дешифрования */
```

```
    /* метод дешифрует данные по алгоритму RSA на основе файла, содержащего закрытый ключ, и выводит результат в консоль */
    public static void main (String[] args) {
        File privateKeyFile = new File("privateKey.txt");
        File encodedFile = new File ("encodedRSA.txt");

        Decoder decoder = new Decoder(privateKeyFile, encodedFile);
        System.out.println("Исходное сообщение:");
        System.out.println(decoder.getOriginalMessage());
    }
}
```

Файл «Decoder.java»

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.LinkedList;
import java.util.Scanner;
import java.math.BigInteger;

public class Decoder extends ParentCoder {

    /* конструктор экземпляра класса Decoder */
    public Decoder(File privateKeyFile, File encodedFile){
        setEncodedMessage(readEncodedFile(encodedFile));
        readKeys(privateKeyFile);
        decrypt();
    }

    /* метод, осуществляющий дешифрование по алгоритму RSA */
    private void decrypt () {
        System.out.println("Дешифрование...");
        StringBuilder stringBuilder = new StringBuilder();
        for (BigInteger bigNumber : getEncodedMessage()) {
            BigInteger bigIntegerCodeSymbol = bigNumber.modPow(getKeyPartI(),getKeyPartII());
            int codeSymbol = bigIntegerCodeSymbol.intValueExact();
            char symbol = (char) codeSymbol;
            stringBuilder.append(symbol);
        }
        setOriginalMessage(stringBuilder.toString());
        System.out.println("Расшифровано.");
    }

    /* метод считывает зашифрованные данные из файла */
```

```
    private LinkedList<BigInteger> readEncodedFile (File publicKeyFile) {
        LinkedList<BigInteger> publicKey = new LinkedList<>();
        Scanner scanner;
        try {
            scanner = new Scanner(publicKeyFile);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                BigInteger value = new BigInteger(line, 16);
                publicKey.add(value);
            }
            System.out.println("Зашифрованные данные прочитаны.");
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return publicKey;
    }
}
```