Lab №02

Alyssandra Cordero, Maria Pedroza 02/15/2019

Listing 1: Lab 02 - TheVigenereCipher

```
1  /*************************************************************
2    * TheVigenereCipher class.
3    * Encrypts a desired message based on a desired key and table.
4    * Includes a method that decrypts a desired message.
5    *************************************************************/
6  public class TheVigenereCipher{
7   public static void main(String[]args){
8
9    final int n = 26;//Initialization of the array size.
10   String original, key = "";//original and key variable declaration.
11   char [][] twoD = new char [n][n];//char [][] twoD declaration.
12
13   try{
14    original = "THISISAMESSAGE";//original and key variable initialization.
15    key = "HELLO";
16
17    populateLookUpTable(twoD);//[][] twoD initialization.
18    printArrayValues(twoD);//print the content of [][]twoD.
19
20    char[][] buffer = filloutBuffer(original, key);// char [][] buffer ↩
        declaration & initialization.
21    System.out.print("\n");
22    printArrayValues(buffer);//print the content of [][]buffer.
23
24    System.out.print("\n");
25    String encrypted = encrypt(original, key, twoD);//String encrypted ↩
        declaration & initialization.
26    System.out.println(encrypted);
27
28    System.out.print("\n");
29    System.out.println("Decrypted Message: "+decrypt(encrypted, key, ↩
        twoD));//Decypt method call.
30
31   }
32   catch(LargerKeyException e){
33    System.out.println("\n" + e);
34   }
35  }
36  /*************************************************************************************
```

```
37    * Method that populates the look up table with the abcs.
38    * @param char[][]v The array of chars that holds the dimention of the ↩
          table to populate.
39    ************************************************************************
40   public static void populateLookUpTable(char[][]v){
41    int letter = 65; //letter declaration. letter holds the first letter ↩
          of the alphabet.
42    int letterNum = 0;// letterNum initialization.
43
44    for(int cols = 0; cols < v.length; cols++){//for loop that goes trough ↩
          every columnn of the array.
45     for(int rows = 0; rows < v[cols].length; rows++){//for loop that ↩
           goes trough every row of the array.
46
47      int counter = (rows+cols);//int counter that holds the sum of the ↩
            rows and the columms of the array.
48      letterNum = (65) + (counter);//letterNum declaration. letterNum is ↩
            used like a counter to update the.
49
50      if(letterNum <= 90){
51       letter = letterNum;
52       v[rows][cols] = (char)letter; ///*
53      }else{
54       counter = counter-26;//counter ''update''(to repeat the ↩
             alphabet.26*26).
55       letterNum = (65) + (counter);//
56       letter = letterNum;
57       counter++;
58      }
59      v[rows][cols] = (char)letter; ///*
60     }
61    }
62    letter++;//Updates the letter number.
63   }
64   /************************************************************
65    * Prints the values stored in a char array.
66    * @param char[][]t A char array.
67    ************************************************************/
68   public static void printArrayValues(char [][] t){
69    for(int rows = 0; rows < t.length; rows++){//for loop that goes trough ↩
          each row of the 2D-array.
70     System.out.print ("\n");
71     for(int cols = 0; cols < t[rows].length; cols++){//for loop that ↩
           goes trough each column of the 2D-array.
72      System.out.print (t[rows][cols]+" ");
73     }
74    }
```

```
75  }
76  /*****************************************************************************
77    * Stores a message and a key in a separate 2D-array.
78    * @param original .- The value to store in the first row of the ↩
        2D-array.
79    *@param key .- The value to store in the second row of the 2D- array.
80    * @return char[][] .- a two dimensional array of characters.
81    * @throws LargerKeyException .- an exception that is thrown if the key ↩
        is larger than the message.
82    *****************************************************************************
83  public static char[][] filloutBuffer(String original, String key)throws ↩
      LargerKeyException{
84
85    if(key.length() > original.length())//if statement that throws the ↩
        exception.
86      throw new LargerKeyException("THE KEY IS LARGER THAN THE MESSAGE...");
87
88    final int rowsLength = 2;
89    final int colsLength = original.length();
90    int a = 0; //int a that is used as a counter.
91    char [] [] buffer = new char[rowsLength][colsLength];
92
93    for (int rows=0; rows< buffer.length;rows++){
94      a=0;//:)
95      for (int cols = 0; cols < buffer[rows].length;cols++){
96
97        buffer[0][cols]= original.charAt(cols);//:)
98
99
100       if(a < key.length()){
101         buffer[1][cols] = key.charAt(a); //*
102       }else{
103         a=0;
104         buffer[1][cols] = key.charAt(a); //*
105       }
106       a++;//:)
107     }
108   }
109   return buffer;
110 }
111 /************************************************************
112   * The mapCharacters method is to find the corresponding character
113   * on the table of alphabets based on two characters.
114   * @param c1, A character representing the first character that will be
115   * searched on the table.
116   * @param c2, A character representing the second character that will be
117   * searched on the table.
```

```
118     * @param table, A 2D array of characters that will be used to ↩
            determine the original message.
119     ***********************************************************/
120   public static char mapCharacters( char c1, char c2, char [][]table){
121     char returned = ' ';
122     int c=0;
123     int r=0;
124     for(int cols = 0; cols < table.length; cols++){//for loop that goes ↩
            trough every columnn of the array.
125       if( table[0][cols] == c2){
126         c = cols;
127       }
128       for(int rows = 0; rows < table[cols].length; rows++){//for loop that ↩
              goes through each row of the specific column.
129         if (table[rows][0] == c1){
130           r = rows;
131         }
132         returned = table[r][c];//returned = char found
133       }
134     }
135
136     return returned;
137   }
138   /************************************************************
139     * The encrypt method will take the original message, the desired key, ↩
            and the
140     * alphabet table that was made in the populateLookUpTable method and ↩
            return the encrypted
141     * message.
142     * @param o, A String that represents the original message that will be ↩
            encrypted.
143     * @param k, A String that represents the key that will be use to ↩
            encrypt and decrypt
144     *  the original message.
145     * @param table, A 2D array of characters that will be used to ↩
            determine the original message.
146     * @return r, A String that represents the encrypted message.
147     * @throws LargerKeyException, it prints a message to the user if the ↩
            key is
148     * larger thatn the message.
149     ***********************************************************/
150   public static String encrypt(String o, String k, char[][]table)throws ↩
          LargerKeyException{
151     // ** c1 and c2 variables declaration**
152     char c1,c2 = ' ';
153     String r = "";
154     char[][] buffer = filloutBuffer(o,k);
```

```
155
156     // ** c1 and c2 variables initialization**
157     System.out.print("\n");
158     for(int i = 0; i < o.length(); i++){
159       c1 = (o.toUpperCase()).charAt(i);
160       c2 = buffer[1][i];
161       r +=(mapCharacters(c1,c2,table));//concatinating the char to the r ↩
            String.
162     }
163     return r;
164   }
165   /************************************************************
166     * The decrypt method will use the encrypted message, the key, and the ↩
          alpha-
167     * bet table to decrypt the message and return the original message.
168     * @param encrypted, a String of the encrypted message that was given ↩
          in the
169     * encrypt method.
170     * @param key, a String of the desired key that was originally chosen to
171     * encrypt the original message.
172     * @param table, A 2D array of characters that will be used to ↩
          determine the original message.
173     * @throws LargerKeyException, it prints a message to the user if the ↩
          key is
174     * larger thatn the message.
175     * @return decrypted, a String containing the original message.
176     ************************************************************/
177   public static String decrypt(String encrypted, String key, ↩
        char[][]table)throws LargerKeyException{
178     char[][] eBuffer = filloutBuffer(encrypted,key);//create a new buffer ↩
          for the encrypted message.
179
180     //printArrayValues(eBuffer);//dbug
181     //2.- row- the row number where the enrypted message is at.col - ↩
          counter's col number.
182     int row = 0, col = 0;
183
184     char encryptedChar = '\u0000', keyChar= '\u0000';
185     //System.out.println("\n");
186     //System.out.println("keyChar: "+keyChar);
187     //System.out.println("encryptedChar: "+encryptedChar);
188     String decrypted = "";
189
190     if(key.length() > encrypted.length())// validate that the key String ↩
          is smaller than the encrypted message.
191       throw new LargerKeyException("THE KEY IS LARGER THAN THE MESSAGE...");
192
```

```
193     for(int iterC = 0 ;iterC < encrypted.length();iterC++){ //:)
194       encryptedChar = eBuffer[0][iterC]; //will go through the whole ↩
              encrypted message
195       keyChar = eBuffer[1][iterC];
196       for(int tableRows = 0; tableRows < table.length; tableRows++){
197         if(keyChar == table[tableRows][0]){//searching for the character of ↩
              the key to be found on the rows of the table.
198           row = tableRows;
199           //System.out.println("row:"+row);
200         }
201       }
202       for(int tableCols = 0; tableCols < table[row].length;tableCols++){
203         //searching for the character of the encrypted message to be found ↩
              on the columns of the table.
204         if(encryptedChar == table[row][tableCols]){
205           col = tableCols;
206           //System.out.println("col:"+col);
207         }
208       }
209       decrypted += table[0][col]; //concatinating the chars to decrypted.
210     }
211     return decrypted;
212   }
213   /*****************************************************************
214     * TheVigenereCipher class.
215     * Encrypts a desired message based on a desired key and table.
216     * Includes a method that decrypts a desired message.
217     *****************************************************************/
218 public class TheVigenereCipher{
219   public static void main(String[]args){
220 
221     final int n = 26;//Initialization of the array size.
222     String original, key = "";//original and key variable declaration.
223     char [][] twoD = new char [n][n];//char [][] twoD declaration.
224 
225     try{
226       original = "THISISAMESSAGE";//original and key variable initialization.
227       key = "HELLO";
228 
229       populateLookUpTable(twoD);//[][] twoD initialization.
230       printArrayValues(twoD);//print the content of [][]twoD.
231 
232       char[][] buffer = filloutBuffer(original, key);// char [][] buffer ↩
              declaration & initialization.
233       System.out.print("\n");
234       printArrayValues(buffer);//print the content of [][]buffer.
235 
```

```
236       System.out.print("\n");
237       String encrypted = encrypt(original, key, twoD);//String encrypted ↩
              declaration & initialization.
238       System.out.println(encrypted);
239
240       System.out.print("\n");
241       System.out.println("Decrypted Message: "+decrypt(encrypted, key, ↩
              twoD));//Decypt method call.
242
243     }
244     catch(LargerKeyException e){
245       System.out.println("\n" + e);
246     }
247   }
248   /*****************************************************************************
249     * Method that populates the look up table with the abcs.
250     * @param char[][]v The array of chars that holds the dimention of the ↩
            table to populate.
251     *****************************************************************************
252   public static void populateLookUpTable(char[][]v){
253     int letter = 65; //letter declaration. letter holds the first letter ↩
          of the alphabet.
254     int letterNum = 0;// letterNum initialization.
255
256     for(int cols = 0; cols < v.length; cols++){//for loop that goes trough ↩
          every columnn of the array.
257       for(int rows = 0; rows < v[cols].length; rows++){//for loop that ↩
            goes trough every row of the array.
258
259         int counter = (rows+cols);//int counter that holds the sum of the ↩
              rows and the columms of the array.
260         letterNum = (65) + (counter);//letterNum declaration. letterNum is ↩
              used like a counter to update the.
261
262         if(letterNum <= 90){
263           letter = letterNum;
264           v[rows][cols] = (char)letter; ///*
265         }else{
266           counter = counter-26;//counter ''update''(to repeat the ↩
                alphabet.26*26).
267           letterNum = (65) + (counter);//
268           letter = letterNum;
269           counter++;
270         }
271         v[rows][cols] = (char)letter; ///*
272       }
273     }
```

```
274     letter++;//Updates the letter number.
275   }
276   /*************************************************************
277     * Prints the values stored in a char array.
278     * @param char[][]t A char array.
279     *************************************************************/
280   public static void printArrayValues(char [][] t){
281     for(int rows = 0; rows < t.length; rows++){//for loop that goes trough ↩
          each row of the 2D-array.
282       System.out.print ("\n");
283       for(int cols = 0; cols < t[rows].length; cols++){//for loop that ↩
            goes trough each column of the 2D-array.
284         System.out.print (t[rows][cols]+" ");
285       }
286     }
287   }
288   /*****************************************************************************
289     * Stores a message and a key in a separate 2D-array.
290     * @param original .- The value to store in the first row of the ↩
          2D-array.
291     *@param key .- The value to store in the second row of the 2D- array.
292     * @return char[][] .- a two dimensional array of characters.
293     * @throws LargerKeyException .- an exception that is thrown if the key ↩
          is larger than the message.
294     *****************************************************************************
295   public static char[][] filloutBuffer(String original, String key)throws ↩
        LargerKeyException{
296
297     if(key.length() > original.length())//if statement that throws the ↩
          exception.
298       throw new LargerKeyException("THE KEY IS LARGER THAN THE MESSAGE...");
299
300     final int rowsLength = 2;
301     final int colsLength = original.length();
302     int a = 0; //int a that is used as a counter.
303     char [] [] buffer = new char[rowsLength][colsLength];
304
305     for (int rows=0; rows< buffer.length;rows++){
306       a=0;//:)
307       for (int cols = 0; cols < buffer[rows].length;cols++){
308
309         buffer[0][cols]= original.charAt(cols);//:)
310
311
312         if(a < key.length()){
313           buffer[1][cols] = key.charAt(a); //*
314         }else{
```

```
315         a=0;
316         buffer[1][cols] = key.charAt(a); //*
317        }
318        a++;//:)
319      }
320    }
321    return buffer;
322  }
323  /************************************************************
324    * The mapCharacters method is to find the corresponding character
325    * on the table of alphabets based on two characters.
326    * @param c1, A character representing the first character that will be
327    * searched on the table.
328    * @param c2, A character representing the second character that will be
329    * searched on the table.
330    * @param table, A 2D array of characters that will be used to ↩
         determine the original message.
331    ************************************************************/
332  public static char mapCharacters( char c1, char c2, char [][]table){
333    char returned = ' ';
334    int c=0;
335    int r=0;
336    for(int cols = 0; cols < table.length; cols++){//for loop that goes ↩
         trough every columnn of the array.
337      if( table[0][cols] == c2){
338        c = cols;
339      }
340      for(int rows = 0; rows < table[cols].length; rows++){//for loop that ↩
           goes through each row of the specific column.
341        if (table[rows][0] == c1){
342          r = rows;
343        }
344        returned = table[r][c];//returned = char found
345      }
346    }
347
348    return returned;
349  }
350  /************************************************************
351    * The encrypt method will take the original message, the desired key, ↩
         and the
352    * alphabet table that was made in the populateLookUpTable method and ↩
         return the encrypted
353    * message.
354    * @param o, A String that represents the original message that will be ↩
         encrypted.
355    * @param k, A String that represents the key that will be use to ↩
```

```
        encrypt and decrypt
356     * the original message.
357     * @param table, A 2D array of characters that will be used to ↩
            determine the original message.
358     * @return r, A String that represents the encrypted message.
359     * @throws LargerKeyException, it prints a message to the user if the ↩
            key is
360     * larger thatn the message.
361     ************************************************************/
362    public static String encrypt(String o, String k, char[][]table)throws ↩
        LargerKeyException{
363     // ** c1 and c2 variables declaration**
364     char c1,c2 = ' ';
365     String r = "";
366     char[][] buffer = filloutBuffer(o,k);
367
368     // ** c1 and c2 variables initialization**
369     System.out.print("\n");
370     for(int i = 0; i < o.length(); i++){
371       c1 = (o.toUpperCase()).charAt(i);
372       c2 = buffer[1][i];
373       r +=(mapCharacters(c1,c2,table));//concatinating the char to the r ↩
              String.
374     }
375     return r;
376    }
377    /************************************************************
378     * The decrypt method will use the encrypted message, the key, and the ↩
            alpha-
379     * bet table to decrypt the message and return the original message.
380     * @param encrypted, a String of the encrypted message that was given ↩
            in the
381     * encrypt method.
382     * @param key, a String of the desired key that was originally chosen to
383     * encrypt the original message.
384     * @param table, A 2D array of characters that will be used to ↩
            determine the original message.
385     * @throws LargerKeyException, it prints a message to the user if the ↩
            key is
386     * larger thatn the message.
387     * @return decrypted, a String containing the original message.
388     ************************************************************/
389    public static String decrypt(String encrypted, String key, ↩
        char[][]table)throws LargerKeyException{
390     char[][] eBuffer = filloutBuffer(encrypted,key);//create a new buffer ↩
            for the encrypted message.
391
```

```
392     //printArrayValues(eBuffer);//dbug
393     //2.- row- the row number where the enrypted message is at.col - ↩
            counter's col number.
394     int row = 0, col = 0;
395
396     char encryptedChar = '\u0000', keyChar= '\u0000';
397     //System.out.println("\n");
398     //System.out.println("keyChar: "+keyChar);
399     //System.out.println("encryptedChar: "+encryptedChar);
400     String decrypted = "";
401
402     if(key.length() > encrypted.length())// validate that the key String ↩
            is smaller than the encrypted message.
403       throw new LargerKeyException("THE KEY IS LARGER THAN THE MESSAGE...");
404
405     for(int iterC = 0 ;iterC < encrypted.length();iterC++){ //:)
406       encryptedChar = eBuffer[0][iterC]; //will go through the whole ↩
              encrypted message
407       keyChar = eBuffer[1][iterC];
408       for(int tableRows = 0; tableRows < table.length; tableRows++){
409         if(keyChar == table[tableRows][0]){//searching for the character of ↩
                the key to be found on the rows of the table.
410           row = tableRows;
411           //System.out.println("row:"+row);
412         }
413       }
414       for(int tableCols = 0; tableCols < table[row].length;tableCols++){
415         //searching for the character of the encrypted message to be found ↩
                on the columns of the table.
416         if(encryptedChar == table[row][tableCols]){
417           col = tableCols;
418           //System.out.println("col:"+col);
419         }
420       }
421       decrypted += table[0][col]; //concatinating the chars to decrypted.
422     }
423     return decrypted;
424   }
425 }
```

Listing 2: Lab 02 - LargerKeyException

```
1   /*************************************************************
2     * LargerKeyException class.
3     * Extends the Exception class.
4     * Creates a new exception called LargerKeyException .
5     *************************************************************/
```

```
6  public class LargerKeyException extends Exception{
7    public LargerKeyException(String s){
8      super(s);
9    }
10 }
```