GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

H.Dip. in Science (Software Development) –Object-Oriented Software Development (2020)

ASSIGNMENT DESCRIPTION & SCHEDULE

A Rail Fence Cypher

Note: This assignment will constitute 50% of the total marks for this module.

Overview

You are required to develop a command line menu-driven Java application that is capable of encrypting and decrypting a short message or file using a *Rail Fence* cypher. The *Rail Fence* is a transposition cypher, where the letters of a plaintext are rearranged using a pattern to encrypt a message. The *Rail Fence* cypher has roots in antiquity, where the ancient Greeks used a device called a *scytale* (pronounced skit-ally) to encrypt military communications. It was also used by both the Union and Confederate forces during the US Civil War (1861-65) to send Morse-encoded messages by telegraph.

Requirements

You are required to develop a Java application that can parse, encrypt and decrypt the contents of a text file or URL using the *Rail Fence* cipher.

```
$ java ie.gmit.dip.Runner

1. Select File or URL
2. Enter Rail Fence Key
3. Encrypt
4. Decrypt
5. Display Rail Fence
6. Quit

Type Option [1-6]>
```

- The application should contain a ***command-line menu*** (as shown above) that prompts the user to enter an encryption key and a file or URL. It should then parse the given resource line-by-line, encrypt / decrypt the text and then append to an output file.
- You ***must use a 2D array*** structure to implement the cipher. Do not use any of the data structures in the *Java Collections Framework*. The knowledge you will gain from mastering array structures will prove invaluable for the programming modules in the next semester.

- Your implementation of the *Rail Fence* cipher should allow the user to change the starting row for encryption / decryption. This implies that there are two parts to the encryption key 1) an integer representing the number of rows in the encryption matrix and 2) an integer representing the row to start the encryption from (the offset). The examples shown below represent a key of (5, 0) for the rows and offset respectively.

Encrypting a Message with a Rail Fence

The Rail Fence cypher encrypts plaintext by creating an *n* x *k* matrix of characters, where *n* is the length of the plaintext and *k* is the key. The plain-text characters are written into the matrix in a zigzag pattern and resultant cypher-text is created by appending the set of characters in each row to a string. Consider the following matrix for encrypting the text STOPTHEMATTHECASTLEGATES with a key of 5.

S								A							T							
	T						M		T					S		L						S
		O				E				T			A				E				E	
			P		H						H		C						G		T		
				T								E								A			

The encrypted message will be SATTMTSLSOETAEEPHHCGTTEA.

Decrypting a Message with a Rail Fence

The decryption of a message requires that the columnar transposition above is undone, i.e. that the original encryption matrix is re-created. This can be performed by reading each character into a *n* x *k* matrix using a zigzag pattern and filling each row in turn. Consider the following states of the matrix to decrypt the cypher-text SATTMTSLSOETAEEPHHCGTTEA with a key of 5:

Zigzag and fill row 1: Adds the first three characters (SAT) to the matrix.

S								A							T							
	?						?		?					?		?						?
		?				?				?				?				?				?	
			?		?						?		?						?		?		
				?								?								?			

Zigzag and fill row 2: Adds the next six characters (TMTSLS) to the matrix.

S								A							T							
	T						M		T					S		L						S
		?				?				?				?				?				?	
			?		?						?		?						?		?		
				?								?								?			

Zigzag and fill row 3: Adds the next six characters (OETAEE) to the matrix.

S								A							T							
	T						M		T					S		L						S
		O				E				T			A				E				E	
			?		?						?		?						?		?		
				?								?								?			

Zigzag and fill row 4: Adds the next six characters (PHHCGT) to the matrix.

S								A							T							
	T						M		T					S		L						S
		O				E				T			A				E				E	
			P		H						H		C						G		T		
				?								?								?			

Zigzag and fill row 4: Adds the last three characters (TEA) to the matrix.

S								A							T							

	T						M		T						S		L					S
		O				E				T				A				E			E	
			P		H						H		C						G		T		
				T								E								A			

At this stage, the original encryption matrix has been created. The plain-text can now be created by zigzagging through the matrix and appending each character to a string. Note that the distance between the characters on each row can be computed as ***d = 2 * (k – row) – 2***, where *k* is the size of the key and *row* is the array index of a row in the range [0…k – 1]. It is possible to implement the Rail Fence cipher without a 2D array using this equivalence.

The objective of the assignment is to reinforce the programming concepts that we have covered throughout the module. In addition to the methods above, you are free to add any additional functionality you want. The application of good programming practice and object-oriented techniques will be rewarded.

Deployment and Submission

- ***The project must be submitted by midnight on Monday 31st August 2020.*** The project must be submitted as a **Zip** archive ***(not a 7z, Rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*A Rail Fence Cypher (50%) Assignment Upload"* heading in the assignment section of Moodle.
- Use the package name ***ie.gmit.dip*** for your classes.
- The name of the Zip archive should be ***<id>.zip*** where <*id*> is your GMIT student number. The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Component	Category
src	A directory that contains the packaged source code for your application. Your application should be launched using the following syntax from a command prompt: **java ie.gmit.dip.Runner** **Note:** this requires that the ***main()*** method is defined inside a class called *Runner.java*.
README.txt	A text file that lists the main features of your application. Any extra functionality should be clearly flagged in the README.

Scoring Rubric

Marks for the project will be applied using the following criteria:

Marks	Category
(10%)	Submission + Structure (includes class, method & variable names)
(10%)	Compilation (only if basic functionality implemented)
(10%)	Comments + Indentation
(10%)	Command Line Menu / Runner
(20%)	Encryption (including Array and Array Methods)
(20%)	Decryption (including Array and Array Methods)
(20%)	Relevant Extras (must be clearly documented in README.txt)

Each of the categories above will be scored using the following criteria:

- 0–39%: Not delivering on basic expectation
- 40-59%: Meeting basic expectation
- 60–79%: Tending to exceed expectation
- 80-90%: Exceeding expectations
- 90+%: Exemplary