Department of Computer Science & Applied Physics

H.Dip. in Science (Software Development) - Object-Oriented Software Development (2023)

File Encryption using 2D Arrays

ASSIGNMENT DESCRIPTION & SCHEDULE

***Note*: This assignment is worth 50% of the total marks for this module.**

1. Overview

You are required to develop a Java application that is capable of encrypting and decrypting a text using an ADFGVX cypher. The ADFGVX cypher was used by the German Army in WW1 from March 1918 to encrypt field communications during the Ludendorff Offensive (Kaiserschlacht). The cypher is so named because all messages are encrypted into codes from the small alphabet of ADFGVX to reduce operator error when sending Morse Code signals. Although an improvement on the ADFGX cypher used by the Germans up until 1918, the new cypher was broken by the French cryptanalyst Georges Painvin and proved decisive in repulsing the attack at Compiègne on June 11th 1918.

The ADFGVX cypher uses a Polybius square to encode each letter / number as two symbols in the alphabet {ADFGVX}. This is not unlike a Vigenère / polyalphabetic cipher.

	A	**D**	**F**	**G**	**V**	**X**
A	P	H	0	Q	G	6
D	4	M	E	A	1	Y
F	L	2	N	O	F	D
G	X	K	R	3	C	V
V	S	5	Z	W	7	B
X	J	9	U	T	I	8

Fig 1. Polybius Square

In addition to the Polybius square, the ADFGVX cypher also requires a code word to be used to diffuse and fractionate the codes derived from the square.

2. Minimum Requirements

You are required to develop a **menu-driven** Java application that can **1) Parse** all the text files in a directory, **2) Encrypt / decrypt** the text in each file using a symmetric key and then **3)**

Output the processed text to a file or files in a specified output directory. An overview of the process is shown below:

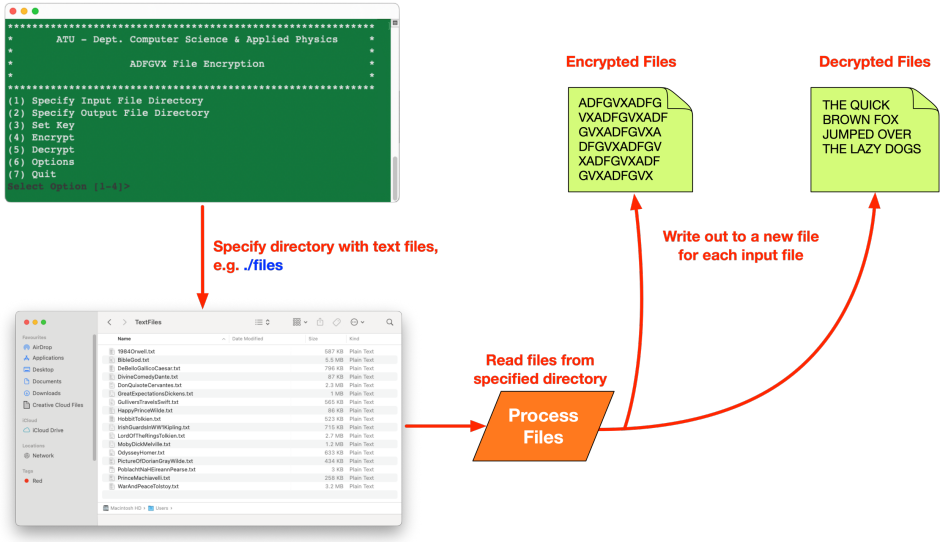

Fig 2.

As illustrated in Fig. 2, the application should:

- **Use the package name *ie.atu.sw*** and place the *main()* method in a class called *Runner*. A set of stubs is available on Moodle to get you started.

- **Provide a simple command-line user interface** that enables a user to specify the following:
    1. A path and name for the directory of text files to process.
    2. An option to set a cipher key.
    3. Options to encrypt or decrypt.
    4. A path and name for file(s) to output.

  You can include as many other menu items as you wish and will be given marks for relevant functionality added.

- **Process** each text file line-by-line and convert the text into its correct plain-text or cipher-text. You can strip out unwanted characters from a string with a regular expression. Consider the following:

```
String s = "H£@@a+++p p...'y D!@£$%^&*()ays!";
s = s.trim().replaceAll("[^a-zA-Z]", "").toUpperCase();
System.out.println(s);
```

  The output is **happydays**. The original String s can be broken up into a String[] of the words {"Happy", "Days"} using the split() method: `String[] words s.split("\\s+");`

- **Output** the processed text files into a specified directory.

- **Contain comments** for each method in your application.

3. Encrypting a Message

1. To encode a plaintext character using the Polybius Square, locate the character in the matrix and read off the letter on the ***far left side on the same row*** and then the letter at the ***top of the same column***, i.e. *each plaintext character is represented by two cipher characters*. For example, the plaintext "OBJECT" will generate the sequence of pairs {FG, VX, XA, DF, GV, XG};

2. ***Create a matrix*** from a code word with the enciphered codes underneath. In this case the code word *JAVA* will be used for both encryption and decryption. As each plaintext character has is represented by two enciphered codes, it creates a degree of *diffusion* in the cypher.

J	A	V	A
F	G	V	X
X	A	D	F
G	V	X	G

3. Perform a ***columnar transposition***, by sorting the plaintext alphabetically. This step *fractionates* the cypher by splitting up the two enciphered codes associated with each plaintext character.

A	A	J	V
G	X	F	V
A	F	X	D
V	G	G	X

4. The final cyphertext is formed by ***reading off each column***:

{GAVXFGFXGVDX}

4. Decrypting a Message

The decryption of a message requires that the columnar transposition in Step 3 above is undone. This can be performed as by reading each set of codes into the correct column denoted by the index and then checking each pair of code along each row against the Polybius Square.

J	A	V	A
2	*0*	*3*	*1*
F	G	V	X
X	A	D	F
G	V	X	G

Note that, because the encrypted message is 12 characters long and the code word JAVA is four characters long, we can compute the number of rows required in the decoding matrix as the *message-length/code-word-length*. Use the modulus operator (%) to test if there is a remainder – if so, an additional row will be required.

5. Deployment and Submission

- ***The project must be submitted by midnight on Thursday 31st August 2023.*** The project must be submitted as a Zip archive ***(not a rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the *"File Encryption using 2D Arrays - (50%) Assignment Upload"* heading in the "Assignment" section of Moodle.

- The name of the Zip archive should be <*id*>.zip where <*id*> is your ATU student number.

Do NOT submit the assignment as an Eclipse project or submit any text files or other resources. If you do, you will lose marks. You will also lose marks if you hard-code any environmental variables in your project like system paths and file names.

- The Zip archive must have the following structure:

Marks	Category
src	A directory that contains the packaged **source code** for your application.
README.pdf	A **PDF file** detailing the main features of your application in **no more than 300 words**. Marks will only be given for features that are described in the README.

6. Scoring Rubric

Component	Marks (100)	Description
Submission	5	Zip archive correctly named and structured
Compiles	5	**All or nothing.** The Java source code compiles without errors and runs from the command line.
README	5	The README file clearly describes the application, its features, is free from spelling and grammatical errors.
UI	15	The user interface allows directory names, file names and any other parameters to be specified. ***Paths and environmental variables are NOT hard-coded.***
Parsers	15	All text files in the specified directory are processed.
Encryption / Decryption	15	Cipher methods implemented correctly.
Output	15	Processed text files are generated correctly in specified directory.
Comments and Formatting	15	All methods and key steps in the application have been fully commented and the code is correctly indented and formatted.
Extras	10	Extra marks will be given for the appropriate application of the ideas and principles we study on this course. Any additional functionality must be documented in the README.