**Course Name:** Information Security
**Course ID:** CPCS 425
**Term:** Fall 2023
**Course Instructor:** Dr. Reemah Alhebshi
**Email:** ralhebshi@kau.edu.sa

# Project: Cipher Application

## Instructions:

- The deadline for this project is **Saturday, November 11, 2023 at 11:59 PM**.
- Any form of **plagiarism will result in receiving zero** in the project.
- **WARNING**: Late submission will not be accepted. Any project submitted after the cutoff time will receive zero.
- This project report has to be submitted in PDF format via Blackboard.
- Your package also should include the source files, output screen shots, and sample input and output files.
- You will be given 20 minutes to demonstrate your project. Questions asked will be related to the working of your project (program).
- This project worth 20% of the overall module marks (100%).

## Goal

In this project, you will be writing a short Java program to perform simple encryption and decryption on a text file message. Your program will read in a message file, encrypt it as described below and then read back in the encrypted file and perform a decryption.

## Learning outcomes

At the end of this project, you will be able develop codes to encrypt and decrypt text messages using some classical encryption techniques.

## The Scenario

Personal privacy is of utmost importance in the global networked world. One of the best tools to help people safeguard their personal information is the use of cryptography software. Modern cryptography algorithms require knowledge of advanced mathematics and programming. In this assignment you will gain a very brief introduction in the production of cryptography software.

## Input File

The input file for this program will be named message.txt. However, your program must be designed to allow it to easily work with other input filenames without requiring the code of the class to be changed. This implies that the message.txt filename must not be "hard wired" into the code without the possibility of being changed. The following assumptions regarding the input file apply.

The input file consists of lines of text. It may be assumed that the shortest line of text in the file to be encrypted will be of length no less than four characters. The number of lines of text in the file is unknown. The end of file/data while statement processing loop pattern must be used. A trivial sample input file is shown below:

```
abcdefghijklmnopqrstuvwxyz
```

## Encryption Process

Below is the description of each step of the encryption process. Where necessary an example is given to clarify the process described in the step.

1. Remove any leading or trailing whitespace from the line.
2. Convert all letters in the string to UPPERCASE.

   Step 2 Input Line: `"abcdefghijklmnopqrstuvwxyz"`
   Step 2 Output Line: `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`

3. Move the first half of the string to be the last half. (Note: for lines of odd length the line must be divided such that the first half being moved contains one more character than the last half.)

   Example 1:
   Step 4 Input Line: `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`
   Step 4 Output Line: `"NOPQRSTUVWXYZABCDEFGHIJKLM"`

4. Swap the first 2 characters of the line with the last two characters.

   Step 5 Input Line: `"NOPQRSTUVWXYZABCDEFGHIJKLM"`
   Step 5 Output Line: `"LMPQRSTUVWXYZABCDEFGHIJKNO"`

5. Swap the two characters immediately to the left of the middle of the string with the two characters that immediately follow them.
   Example 1:
   Step 6 Input Line: `"LMPQRSTUVWXYZABCDEFGHIJKNO"`
   Step 6 Output Line: `"LMPQRSTUVWXABYZCDEFGHIJKNO"`

6. Perform the following character substitutions:

| Original | Substitution |
|---|---|
| A | @ |
| E | = |
| I | ! |
| J | ? |
| O | * |
| P | # |
| R | & |
| S | $ |
| T | + |
| V | ^ |
| X | % |
| (space) | _ |

Below gives an example of applying all six steps sequentially to the alphabet string:

Encryption Input Line: `"abcdefghijklmnopqrstuvwxyz"`
Encryption Output Line: `"LM#Q&$+U^W%@BYZCD=FGH!?KN*"`

## The Intermediate File

The encryption of the message file will be written to an intermediate file named "cipher.txt". As with the other filenames, the Cipher class must be designed to allow it to easily work with other intermediate filenames without requiring the code of the class to be changed. Since the intermediate output file will serve as input to the decryption step, no header file information must be written to the file. A sample intermediate file that corresponds to the above input file is shown below:

```
LM#Q&$+U^W%@BYZCD=FGH!?KN*
```

## Decryption Process

Below is the description of each step of the decryption process. Be aware that the cipher that is being used is not symmetric, (i.e., encrypting a message and then decrypting the encrypted text may not always yield the exact original message.) Where necessary an example is given to clarify the process described in the step.

1. Perform the following character substitutions:

| Original | Substitution |
|---|---|
| @ | A |
| = | E |
| ! | I |
| ? | J |
| * | O |
| # | P |
| & | R |
| $ | S |
| + | T |
| ^ | V |
| % | X |
| _ | (space) |

2. Remove any leading or trailing whitespace from the line.

3. Swap the two characters immediately to the right of the middle of the string with the two characters that immediately precede them. (Note: for lines of odd length the line must be divided such that the first half contains one more character than the last half.)

4. Swap the first 2 characters of the line with the last two characters.

5. Move the first half of the string to be the last half. (Note: in this step for lines of odd length the line must be divided such that the first half contains one less character than the last half.)

   Step 5 Input Line: `"@BCD=FGH!?KLMN*#Q&$+U^W%YZ"`
   Step 5 Output Line: `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`

6. Convert all letters in the string to lowercase.

   Step 6 Input Line: `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`
   Step 6 Output Line: `"abcdefghijklmnopqrstuvwxyz"`

Below gives an example of applying all six steps sequentially:

Encryption Input Line: `"LM#Q&$+U^W%@BYZCD=FGH!?KN*"`
Encryption Output Line: `"abcdefghijklmnopqrstuvwxyz"`

## The Output File

The output file is named "decrypt.txt". An output file, which corresponds to the above input and intermediate files, is shown below.

```
abcdefghijklmnopqrstuvwxyz
```

The files for this encryption/decryption sample will be available on the course web site. You can use this sample (as well as others of your own devising) as an input source in your own test cases by creating a BufferedReader attached to the message.txt URL. Note, however, that it is not sufficient to do your testing only using this sample--you must create additional tests using your own input (see the hints on testing your solution below).

## Solution Requirements

- You must provide a class called "Cipher" to serve as the main entry point for your solution.
- Your Cipher class must provide a method with the following signature:

```
public void encrypt( BufferedReader inStream,
                     PrintWriter    outStream )
     throws Exception
{
     // ...
}
```

- The Cipher.encrypt() method must correctly translate all of the input characters from the provided message inStream, and produce the corresponding encrypted output on outStream.
- Your Cipher class must also provide a method with the following signature:

```
public void decrypt( BufferedReader inStream, PrintWriter
outStream ) throws Exception
{
      // ...
}
```

- The Cipher.decrypt() method must correctly translate all of the input characters from the provided encrypted message inStream, and produce the corresponding decrypted output on outStream.
- The Cipher.encrypt() and Cipher.decrypt() methods must not throw any exceptions, except for those that arise from calls sent to inStream.

The Cipher class is only responsible for storing data for the cipher technique described in this project. Behavior for handling other cipher techniques must NOT be included in the Cipher class.

## Testing Hints

When it comes to testing, remember to write one or more test cases for each method that your write in your solution. Preferably, you should write these tests before (or as) you write the method itself, rather than saving testing until your code works. As you work on larger and larger programs, it is important to build skills in convincing yourself that the parts you have already written work as you intend, even if the full solution has not been completed.

Also, in addition to trying to think of various cases that your methods should add formatting to, also write test cases for scenarios where a method **should not** take action (or should signal an error condition, if that is the behavior intended).

Consider the following test method (which assumes your text fixture includes a p4Cipher object created from your Cipher class):

```
public void testEncryption()
{
    try
    {
        // create the streams needed
        BufferedReader inStream  =
            IOHelper.createBufferedReaderForString(
                    "abcdefghijklmnopqrstuvwxyz" );
        StringWriter   result    = new StringWriter();
        PrintWriter    outStream = new PrintWriter( result );

        // run the method to get results
        p4Cipher.encrypt( inStream, outStream );

        inStream.close();
        outStream.close();

        // test that the result is what was expected
        assertEquals( result.toString(),
                              "LM#Q&$+U^W%@BYZCD=FGH!?KN*" );
    }
    catch ( Exception e )
    {
        // If this happens, something went wrong;
        fail();  // treat as a failed test
    }
}
```