EECS 2030: Lab 2

may be done in groups of up to three students

Motivation

This lab will let you practise the following:

- create a utility class
- reviewing array and String manipulation in Java
- implementing static features
- write simple unit tests
- documenting your class properly using Javadoc

Part 1: Getting Started

Download a zip file containing the Lab 2 Eclipse project.

Import the project into Eclipse by doing the following:

1. Under the **File** menu choose **Import...**
2. Under **General** choose **Existing Projects into Workspace** and press **Next**
3. Click the **Select archive file** radio button, and click the **Browse...** button.
4. In the file browser that appears, navigate to your download directory (exactly where this is depends on what computer you working on; on the lab computers the file will probably appear in your home directory)
5. Select the file **Lab2_S23.zip** and click **OK**
6. Click **Finish**.

Explore the existing methods and the test cases, try to understand the purpose of each line. For example, what parameters the constructors are expected to take, what the output or return values should be, and why some operation(s) should be disallowed.

Text Encryption

In this lab you will have to implement an application for encrypting and decrypting a previously encrypted text (represented as a String), based on a specified encryption/decryption key.

The file was encrypted using a combination of a Substitution cipher[1] and a columnar Transposition cipher[2].

The encryption procedure is the following:

- Read the encryption key (this is a symmetric system, and the same key is used both for encryption and for decryption); let's assume, the key is "ABCDEFGH".

[1] https://en.wikipedia.org/wiki/Substitution_cipher
[2] https://en.wikipedia.org/wiki/Transposition_cipher#Columnar_transposition

- Calculate the hash value of the key (this is a random-looking number that is dependent on the string given). In this case, the hash value was taken from the output of the `hashCode` method:

  ```
  int hash = key.hashCode();
  ```

  in this case, the value is 2042300548.

- Use the value as the seed to initialize the Random object:

  ```
  random = new Random(hash);
  ```

  or `random.setSeed(hash);`

  At this point we have a random number generator; however, the sequence of its pseudo-random numbers is determined by the seed above.

- Now we use the numbers this generator generates to create a *substitution pattern* from the following alphabet (ending with space) `“ABCDEFGHIJKLMNOPQRSTUVWXYZ ”`.
  - generate two random numbers between 0 and 26 (the seed was set, right?)
  - swap the letters at these positions in the string above[3]
  - repeat 100 times
  - should get `“GCWHAKSXJMDLFUB ITVYRPZENQO”` after 100 iterations
  - This means, A letters in the original will be replaced with a G, B with C, C with W, and so on:

    ```
    “ABCDEFGHIJKLMNOPQRSTUVWXYZ ”
    “GCWHAKSXJMDLFUB ITVYRPZENQO”
    ```

- Similarly, we should create a pattern for column order during the transposition step:
  - start with pattern {0, 1, 2, 3, 4, 5, 6, 7}
  - set the Random seed to the same initial value
  - generate two random numbers between 0 and 7
  - swap the numbers at these positions in the array above
  - repeat 100 times
  - should get [3, 2, 7, 4, 1, 0, 5, 6] after 100 iterations
  - This means, the columns will be read in the order 3, 2, 7, and so on – instead of sequentially

- Read the text in chunks of 64 characters
  - say, the first 64 characters are

  ```
  “IN CRYPTOGRAPHY, A SUBSTITUTION CIPHER IS A METHOD OF ENCODING B”
  ```

[3] You might want to use a `StringBuilder` class instead of `String`

- Apply the substitution:

  “JUOWTN YBSTG XNOOGOVRCVYJYRYJBUOWJ XATOJVOGOFAYXBHOBKOAUWBHJUSOC”

- Apply the transposition (write row-by-row, then read columns in order 3, 2, 7, …):

```
J U O W T N   Y
B S T G   X N O
O G O V R C V Y
J Y R Y J B U O
W J   X A T O J
V O G O F A Y X
B H O B K O A U
W B H J U S O C
```

- Result: “WGVYXOBJOTOR GOHYOYOJXUCT RJAFKUUSGYJOHBJBOJWVBWNXCBTAOS NVUOYAO”

- For decryption, need to reverse the transposition and substitution steps.

For this lab you need to implement two methods (currently empty), that encrypt or decrypt a text string based on the supplied key.

An example input and the output it produces (these are also present in the supplied unit tester):

Original, “plain text” (from `https://en.wikipedia.org/wiki/Substitution_cipher`):

```
In cryptography, a substitution cipher is a method of encoding
by which units of plaintext are replaced with ciphertext,
according to a fixed system; the "units" may be single letters
(the most common), pairs of letters, triplets of letters,
mixtures of the above, and so forth. The receiver deciphers the
text by performing the inverse substitution.
```

Encrypted:

```
WGVYXOBJOTOR GOHYOYOJXUCT RJAFKUUSGYJOHBJBOJWVBWNXCBTAOS NVUOYAO
XYGGGXYWZJLOLYTWOKATO OJJVJTWOABOU Y JAGNROEAZXOWOUAAWETXBYOHJYH
YAFRGUYAOEAOFJYXONXVAOOVBHOUNSAOSJYOOVAYUKVAOOLOOOOJOLTFGVYYCAVB
B OOYYJBWOKVAAFOUVYJKVTXFGLOVYEKOOBTLLOVYOOA OOAFJAYOAYOBTYTBTRY
CHYTOTAAGUTOTAY OOYJJXCTBOXAHVETOGBAAXOOAOKXP ANPVOWAOYKABOAWYOB
SPCBOOOOUURJOOOOAAYOOOOOOAVUOOOOJJVYOOOOFOOROOOOYTYOOOOOXVJOOOOO
```

After decryption:

```
IN CRYPTOGRAPHY  A SUBSTITUTION CIPHER IS A METHOD OF ENCODING B
Y WHICH UNITS OF PLAINTEXT ARE REPLACED WITH CIPHERTEXT  ACCORDI
NG TO A FIXED SYSTEM  THE  UNITS  MAY BE SINGLE LETTERS  THE MOS
T COMMON   PAIRS OF LETTERS  TRIPLETS OF LETTERS  MIXTURES OF TH
E ABOVE  AND SO FORTH  THE RECEIVER DECIPHERS THE TEXT BY PERFOR
```

```
    MING THE INVERSE SUBSTITUTION
```

Here, any symbols other than 26 English letters and space are replaced with spaces, and all text is converted to uppercase. Even though this make the process lossy, it makes the implementation simpler. Please do the same in your implementation. If the input plaintext is not in multiples of 64 characters, assume it ends with an appropriate number of spaces (use *padding*). The length of the encrypted string must a multiple of 64, and most likely the decrypted result will contain the padding spaces added during the encryption step.

Unit Tester

A unit tester class is also provided. Currently it contains only two test cases (one for encryption, one for decryption).

Think about how you are going to test for the requirements listed in this lab (e.g., how to handle special cases).

Notes

- Try to reduce code duplication whenever possible
- Use helper methods whenever appropriate
- Use the best practices you learned so far, regarding coding style, comments, etc.
- If you cannot complete one or more of the methods, at least make sure that it at least returns some value of the correct type; this will allow the tester to run, and it will make it much easier to evaluate your code. For example, if you are having difficulty with `decrypt` method then make sure that the method returns some string value.

If you have questions, don't hesitate to post your questions on the course forum on eClass or ask teaching assistants during your lab sessions.

Submission

Find all the `java` files in your project and submit them electronically via eClass (do not zip them).

If working in a group, make only one submission and include a **group.txt** file containing the names and the student numbers of the group members. The deadline is firm.

Grading

The lab will be graded using *the Common Grading Scheme for Undergraduate Faculties*[4]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

Academic Honesty

Direct collaboration (e.g., sharing your work results across groups) is not allowed (plagiarism detection software may be employed). However, you're allowed to discuss the lab requirements, approaches you take, etc. You may not use any code from outside sources, even if it is your own code you wrote for another assignment, project, hobby, etc.

[4] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/