ASSIGNMENT 3

Computer Science Fundamentals II

This assignment is due on March 24th, 2020 by 11:55pm.
See the bottom of this document for submission details.

Learning Outcomes

To gain experience with

- Solving problems with the queue data type
- Experience with the idea of encoding and decoding data
- The design of algorithms in pseudocode and their implementations in Java

Do not import any libraries unless specifically asked to do so. Any additional imports will be heavily penalized (-10 or more).

Introduction

Encryption is an important consideration when transmitting information over insecure mediums. One of the ways that this has been done historically is with the use of a cipher. This assignment details making a custom cipher called the 'Western Cipher', and you will be tasked with implementing the encoding and decoding algorithm, alongside using a Circular Array Queue as described in class. This cipher will be used to encode messages strictly consisting of **upper case alphabetical characters** (i.e., A-Z, no numbers, punctuation, etc) **and spaces**, i.e., ' '.

The custom cipher we are implementing has the following rules:

1. All letters are always shifted forwards 5 steps in the alphabet. For example, 'A' would be shifted forward to 'F', and 'B' would be shifted to 'G'.
2. All letters are shifted an additional 2 steps forward for every spot in the length of the message they are, starting with an index of 0. So, for example, the 'AB' from above would be shifted to 'FI'.
3. All values listed below are replaced with the following values according to the table:

A	1
E	2
I	3
O	4
U	5
Y	6

This means that ‘AB’ would now become ‘1I’
This takes priority over the previous rules for the individual letter substituted, so there is no shift in the alphabet before the conversion. Letters shifted to become values in table 3 **should not** be converted to numbers.

4. If the previous letter was converted to a numerical value, the letter that follows should be shifted backwards by twice the amount that the letter was converted to. So, for example, ‘AB’ becomes first ‘1B’, and then B is shifted forward 5 steps from rule 1, 2 steps from rule 2, and then 2 backwards by this rule, 4. So the encoded message would now be ‘1G’.

5. If the previous letter was converted to a numerical value, and the next value is also to be converted to a numerical value, use the following table for encryption instead of following rule 4:

A	3
E	4
I	5
O	6
U	1
Y	2

So ‘AA’ would become ‘13’, ‘AE’ would become ‘14’, and ‘YA’ would become ‘63’. ‘YOU’ would become ‘661’.

These are the rules your cipher must follow. We recommend writing out pseudocode for this process, as the listed order from 1-5 may not be the ideal one from which to convert this into an algorithm.

For the second part of the assignment, you must also implement a decoder method which takes an encoded string and returns it to it’s original form. This should undo all of the operations above in the appropriate order such that a message can be encoded and decoded without any loss of information or mistakes, i.e., ‘AA’ becomes ‘12’ and ‘12’ becomes ‘AA’ again.

Provided files

The following is a list of files provided to you for this assignment. You do not need alter these files in any way and should assume we will mark them with unmodified ones. Studying these will help improve your understanding of Java.

- Classes provided:
  - *EmptyCollectionException.java, TestQueue.java, TestWesternCipher.java, QueueADT.java*
  - *Codes.java is provided and is available under Sample Code on the course website, it is not needed to complete this assignment and is for your reference*

Classes to implement

A description of the classes that you are required to implement for full marks in this assignment are given below. You may implement more classes if you want; you must submit the code for these with your assignment.

In all these classes, you may implement more private (helper) methods as desired, but you may **not** implement more public methods. You may **not** add additional static methods, or instance variables, or static variables. Penalties will be applied if you implement these.

For this assignment, you may not use Java's *Queue* class, although reading and understanding the code there may help you better understand queues. You also may not use any other pre-made Java collections libraries. The data structure required for this assignment is a circular queue, described below:

CircularArrayQueue.java

This class represents a Queue implementation using a circular array as the underlying data structure. This class must implement the QueueADT and work with the generic type (T).

This class must have the following *private* variables:

- front (int)
- rear (int)
- count (int)
- queue (T array)
- DEFAULT_CAPACITY (final int) with a value of 20

The class must have the following *public* methods:

- CircularArrayQueue (constructor) – no parameters required in this first constructor. Initialize the front to 1, rear to the default capacity (DEFAULT_CAPACITY), count to 0, and the queue array using the final int variable DEFAULT_CAPACITY as the array's capacity.

- CircularArrayQueue (second constructor) – same as the first constructor described above, except that this one takes in an int parameter for the initial capacity rather than using the default capacity. Front and rear are set to 1 and initialCapacity respectively.
- enqueue – takes in an element of the generic type and adds that element to the rear of the queue. If the queue is full before adding this item, then call expandCapacity.
- dequeue – throws an EmptyCollectionException if the queue is empty; otherwise remove and return the item at the front of the stack.
- first – throws an EmptyCollectionException if the queue is empty; otherwise return the item at the front of the queue without removing it.
- isEmpty – returns true if the queue is empty, and false otherwise.
- size – returns the number of items on the queue.
- getFront – returns the front index value (NOTE: this is not part of the QueueADT but is still required for this assignment).
- getRear – returns the rear index value (NOTE: this is not part of the QueueADT but is still required for this assignment).
- getLength – returns the current length (capacity) of the array (NOTE: this is not part of the QueueADT but is still required for this assignment).
- toString – returns the string containing "QUEUE: " followed by each of the queue's items in order from front to rear with ", " between each of the items and a period at the end of the last item. If the queue is empty then print "The queue is empty" instead.
- expandCapacity (private) – create a new array that has 20 more slots than the current array has, and transfer the contents into this new array and then point the queue instance variable to this new array by resetting the front and rear appropriately. You may set the front to 1 and the rear to count when expanding the array instead of the DEFAULT_CAPACITY based front and rear.

WesternCipher.java

You should import java.io.*; in this class

This class represents a cipher that can encode and decode information with the algorithm provided above. It must have the following *private* variables:

- encodingQueue (CircularArrayQueue, type Character)
- decodingQueue (CircularArrayQueue, type Character)

The class must have the following *public* methods:

WesternCipher (constructor) – No parameter constructor, it initializes both the encoding and decoding queues with a capacity of 10 and as type Character

WesternCipher (constructor with argument) – Takes an integer as input and initializes both the encoding and decoding queues with the capacity provided and as type Character

encode(String input) – Takes a string as input, splits the string into individual characters, applies the Western Cipher algorithm described above, rejoins the individual characters into a string and returns it. While possible to implement without a queue, this method must enqueue every character into the queue and then encode while dequeueing.

decode(String input) – Takes a string as input, splits the string into individual characters and undoes the Western Cipher algorithm described above. It then rejoins the individual decoded characters, gathers them into a string and returns it. While possible to implement without a queue, this method must enqueue every character into the queue and then decode while dequeueing.

Main method – **This requirement is not assessed in the tests**. The main method must prompt the user about whether it would like to encode or decode a string, take the string and encode/decode as appropriate, and then prompt the user if they would like to enter another string. If no is selected, the program should exit.

Hints:

Look at the provided cipher code for a starting point. You will need to remember %

You can do integer math by converting the starting character ‘A’ or the ending character ‘Z’ and adding or subtracting the appropriate number of letters to move.

The decoder has to undo the encode operations exactly.

Marking notes

Marking categories

- Functional specifications
  - Does the program behave according to specifications?
  - Does it produce the correct output and pass all tests?
  - Are the class implemented properly?
  - Are you using appropriate data structures?
- Non-functional specifications
  - Are there comments throughout the code?
  - Are the variables and methods given appropriate, meaningful names?
  - Is the code clean and readable with proper indenting and white-space?
  - Is the code consistent regarding formatting and naming conventions?
- Penalties
  - Lateness: 10% per day
  - Submission error (i.e. missing files, too many files, ZIP, etc.): 5%
  - "package" line at the top of a file: 5%
  - Additional global scope variables, static methods or variables: 25%

- Importing unnecessary or unasked for dependencies: 50%

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through cheating-detection software.

Submission (due March 24th, 2021 at 11:55pm ET)

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the assignment.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope
- Submitting the files in an incorrect submission page or website will receive a penalty.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- CircularArrayQueue.java
- WesternCipher.java
- Any custom classes used.

Grading Criteria

- Total Marks: [20]
- Functional Specifications:
  - [3] CircularArrayQueue.java
  - [5] WesternCipher.java
  - [10] Passing Tests
- Non-Functional Specifications:
  - [0.5] Meaningful variable names, private instance variables
  - [0.5] Code readability and indentation
  - [1] Code comments