CMPE 242
Spring 2020
Programming Homework #1

This assignment is due by 23:55 on Sunday, 22 March.

Note: This assignment will not be evaluated without a face-to-face demo.

This assignment is designed to help your understanding of the concept of linked list, stack and queue data structures.

PART I

The IT department of TEDU decided to build its own simple version of an email server. They divide the emails into 3 categories, namely: INBOX, ARCHIVE and TRASH. You are asked to implement a system that manages emails.

a) Email Class

First, you must implement an Email Class. The email will be implemented within the Email class, which holds:

- `subject : String`
- `ID : Integer`
- `message : String`
- `time : Integer  //milliseconds`
- `flag : boolean`

The methods should include a default constructor, and the `set…(), get…()` methods for each of the attributes.

b) LinkedListOfEmails Class

First, implement a **Singly-linked** `LinkedListOfEmails` class (together with a **private** class Node). Each node should store an `Email` object as its data. **Consider that the emails in the lists are sorted in a descending order on the arrival time (i.e. newest first, oldest last).**

As mentioned above, the emails are divided into three categories, namely: INBOX, ARCHIVE and TRASH. Therefore, in your program (in Part C), you should create three linked lists, with these names: `Inbox, Archive, Trash`.

The `LinkedListOfEmails` class should support the following methods:

- addEmail(Email E): Adds a new email to the list. Note that emails in all the lists are sorted in descending order on the arrival time (i.e. newest first, oldest last).

- read(int id): Show current Email details with the given email id, and set true current Email read-Flag and change current index to next.

- Email delete(int id): Delete the Email with the given email id, and return the corresponding Email object.

- void showAll(boolean flag): Print all emails details in all categories. If flag is true, all emails are shown. If flag is false, only unread emails are shown.

c) EmailApplication Class

First, as mentioned above, the emails are divided into three categories, namely: INBOX, ARCHIVE and TRASH. Therefore, in your program, you should first create three empty linked lists, with these names: `Inbox, Archive, Trash`.

Then, your program should read input from the standard input line-by-line. Each line of input will contain:

`N`	New email arrived. For example:
	
	`> N //Hello from CMPE242.//1234//This is a welcome email from the course.//2364675//`
	`> N //Homework 1 posted.//1237//This is your first homework for the course.//2264672//`

`R <id>`	Read an email with the given id. For example:
	
	`> R 1234`
	`Email id: 1234`
	`Subject: Hello from CMPE242`
	`Body: This is a welcome email from the course.`
	`Time received: 2364675.`
	`Status: Read`
	
	`> R 1235`
	`No such email.`

`A <id>`	Archive the email with the given id. This should move the email from the Inbox to the Archive. Note that the two lists are sorted in descending order. For example:
	
	`> A 1234`
	`Email 1234 archived.`

`D <id>`	Delete the email. This should move the email from the Inbox to the Trash. Note that the trash is also sorted in descending order.

`S <Folder>`	Show the contents of the folder in descending order of arrival times. The parameter `<Folder>` can be one of: `Inbox, Archive, Trash.`
	
	`Email Subject Body Time Read`
	`1234 Hello from CMPE242. This is a welcome email. 2364675 No`
	`1237 Homework 1 posted. This is your first ho... 2264675 Yes`
	`...`
	
	You should print the truncated form of the subject and body strings to 25 and 40 characters, respectively. Check the Java System.out.printf() method details.

`U <Folder>`	Show all unread emails in the folder in descending order of arrival times. The parameter `<Folder>` can be one of: `Inbox, Archive, Trash.`
`C <Folder>`	Clear the contents of the folder. The parameter `<Folder>` can be one of: `Inbox, Archive, Trash.` If the parameter is `Trash,` the program should empty it. If the parameter is `Inbox` or `Archive` , then the program should move all emails from the target folder to the `Trash.`

d) EmailTester Class

You must also provide a tester class for Part I that demonstrates the functionality of your implementation. Be sure to include all the extreme cases.

PART II

The second part of the homework is about stacks. Note: Also for this section, you must use your own implementation of stacks. The stack should use generics and use a linked-list based implementation.

(Note: you must also provide a tester class for Part II that demonstrates the functionality of your implementation. Be sure to include all the extreme cases.)

a) Calculator (Calc) Class

You task is to implement a basic calculator that supports integer operands like 1, 647, and -42 as well as the (binary) integer operators +, -, *, /, and %. The style of arithmetic expressions our calculator will evaluate is also called a post-fix notation. Stacks are great for doing this job! Your task is to write a driver (client) program that uses the generic Stack interface we discussed in class, and one of the given implementations to perform these calculations as specified here.

Your program should be called **Calc** and work as follows:

- The user enters input that consisting of operands and operators, presumably in post-fix notation.
- If the user enters a valid integer, you push that integer onto the stack.

- If the user enters a valid operator, you pop two integers off the stack, perform the requested operation, and push the result back onto the stack.
- If the user enters the S as an input, you print the current state of the stack.
- If the user enters the P as an input, you pop the top element off the stack and print only that element (not the entire stack) followed by a new line.
- If the user enters the E, you exit the program.

Note that there are a number of error conditions that your program must deal with gracefully for full credit. This means that you'll have to print error messages to the user.

Examples

```
$ java Calc
S
[]
10
S
[10]
20 30
S
[30, 20, 10]
*
S
[600, 10]
+
S
[610]
P
610
S
[]
E
$
```

b) CalcTester Class

You must also provide a tester class that demonstrates the functionality of your implementation. Be sure to include all the extreme cases.

c) Any other necessary classes (Stack etc.)

Be sure to include any other necessary classes to implement Part II.

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. This assignment is due by 23:55 on Sunday, March 22th.

2. You should submit your homework to course moodle page before the deadline. No hardcopy submission is needed. You should send files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., zip, rar).

3. The below rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

4. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.

5. For this assignment, you must use your own implementation of linked lists, stacks and queues. In other words, you cannot use any existing APIs or code from other sources such as the **java.util.ArrayList, java.util.Stack** or **java.util.LinkedList** classes in the Java standard libraries. But you are allowed to use the codes given in our textbook and/or our lecture slides. You ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).

6. You MUST use a linked list based implementation in your solution. You will get no points if you implement array-based solutions (using fixed-sized arrays, dynamically sized arrays, data structures such as vector from the standard library, using ArrayList class, etc.).

7. We will test your implementation by writing your and our own driver .java tester files. For this reason, your classes' name MUST BE as shown in the homework description.

8. The submissions that do not obey these rules will not be graded.

9. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

10. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-------------------------------------------------------
// Title: Stack
// Author: Hamid AHMADLOUEI
// ID: 2100000000
// Section: 0
// Assignment: 1
// Description: This class defines a String stack
```

```
//-------------------------------------------------------
```

11. Since your codes 1will be checked without your observation, you should report everything about your implementation. Well comment your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void setVariable(char varName, int varValue)
//-----------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
//-----------------------------------------------------------
{
      // body of the function
}
```

- Indentation, indentation, indentation...

- This homework will be graded by your TAs, Hamid Ahmadlouei and İbrahim İleri (hamid.ahmadlouei@tedu.edu.tr, ibrahim.ileri@tedu.edu.tr). Thus, you may ask them your homework related questions. You are also welcome to ask your course instructor Tolga Çapın and Tansel Dökeroğlu for help.

GRADING RUBRIC

Performance Element	Weight	Master (4/4)	Advanced (3/4)	Developing (2/4)	Beginner (1/4)	Insufficient (0/4)
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	10%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
Code Design	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
PART I	**Email Application**					
Abstract Data Type: Email Lists	10%	Uses own linked list for implementing email lists. Implements singly linked list and provides all functionality. Uses generics.	Uses own linked list for implementing email lists. Implements singly linked list and provides all functionality. Does not use generics, or some design differences.	Implements doubly linked lists or major deviations from the specification.	Uses standard Java libraries for linked lists.	Not even an attempt.
Functionality	25%	All user commands are implemented with no missing functionality. Runs without any crash. Lists are in descending order.	Missing some minor user commands or minor output problems (e.g. no fixed string size). Runs without any crash. Lists are in descending order	Attempted to implement all user commands but about half of them do not work. Major deviations from the specification.	Only few commands are implemented correctly. Compiles but several warnings.	No working solution or does not compile.

Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set.	Provided a tester class. Able to identify key test points.	Sporadic	No test case presented.
PART II	**Calculator Application**					
Abstract Data Type: Stack	10%	Uses own linked list based implementation of stack. Implements singly linked list and provides all functionality. Uses generics.	Uses own linked list for implementing stack. Implements singly linked list and provides all functionality. Does not use generics, or some design differences.	Implements doubly linked lists or major deviations from the specification.	Uses standard Java libraries for stack.	Not even an attempt.
Functionality	15%	All user commands are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all user commands but some of them do not work.	Only few commands are implemented correctly. Compiles but several warnings.	No working solution or does not compile.
Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested with each set.	Provided a tester class. Able to identify key test points.	Sporadic	No test case presented.

About late submissions: 20 points will be deducted for each late day.