CMPE 242

Programming Homework 1

This assignment is due by 23:55 on Sunday, 4 April

You are welcome to ask your HW related questions. Please use one of these options:

1. Moodle Homework **Question Forum**: HW Question-and-Answer (Q&A) Forum on Moodle is always available. Use the "Forum" link at the course Moodle page.
2. Homework **RECITATION HOURS**: There will be two Q&A RECITATION HOURs on the following days:
   - CMPE242-HW1-OfficeHour1: Mar 22, 06:00-07:00 PM, Zoom ID: 952 6936 0016
   - CMPE242-HW1-OfficeHour2: Mar 26, 06:00-07:00 PM, Zoom ID: 928 5510 3025

Note: Please make sure that you have read the HW document well before participating.

PART I

Step 1

For the first part you should implement the Generic Stack ADT using a linked list. You must implement your own linked list implementation from scratch, you may not use the built-in Java linked list class. Name your stack implementation as **myStack** and the class should implement the following operations:

`boolean isEmpty()`	Returns "true" if and only if the stack is empty.
`int size()`	Returns the number of items in the stack.
`void push(Item)`	Pushes an item at the top of the stack.
`Item pop()`	Removes and returns the top element from the top of the stack.
`Item peek()`	Returns the most recently inserted item on the stack without removing it.
`print()`	Prints the elements of the stack starting from top to bottom.

Note 1: Be sure to test all methods and all exceptions as well.

Note 2: make sure to understand the concept of generics before implementing the class.

Step 2:

In this step, you will write a **VaccineStock** application that will use the **myStack** class you created above for management the COVID-19 vaccine stock. (*Note: you would not normally use a stack for this purpose in real life, but we would like you to experience the use of stacks.)*

First you will create a class called **VaccineStock**. This class will have three private data members and three functions:

`serialNumber`	An integer which holds the vaccine serial number.
`countryName`	A string that holds the name of the country that vaccine was manufactured.
`numberOfVaccines`	An integer that holds the number of vaccines produced.
`void popItem()`	Pops the item off the stack and displays it.
`void pushItem()`	pushes the item onto the stack
`int action()`	Displays the action menu and returns the user's choice

This program should be ask to enter **ADD/DELETE/EXIT** command in an infinite loop. If the user enters the option **ADD**, the program adds a new item to the inventory **myStack**; in **DELETE** entry remove an item from the inventory **myStack**. The loop should continue until the user writes **EXIT**. When adding an item, the program should ask the user for the information it needs for the three data members of the **VaccineStock** class and add a new item to the stack. When removing an item from the stack, the program should display all of the information in the **VaccineStock** object that was popped from the stack. When the program ends, it should pop all of the remaining items of the stack and display its data. Below you can see the example output format:

```
$ java VaccineStock
Enter COMMAND?
ADD
Enter ITEM DATA?
191
CHINA
15000
Enter COMMAND?
ADD
Enter ITEM DATA?
192
TURKEY
8000
Enter COMMAND?
DELETE
192, TURKEY, 8000
Enter COMMAND?
EXIT
191, CHINA, 15000
```

PART II

For Part II, you should implement the Queue ADT using the resizing array approach. You should start with array size of 4 and the size of array can be increased (double) or decreased (halve) if needed, as explained in class.

Step 1:

Write a generic Queue class. Name your class as **myQueue**. The **myQueue** class should implement the following operations. Note that you must implement your own implementation.

`boolean isEmpty()`	Returns "true" if and only if the queue is empty.
`boolean isFull()`	Returns "true" if and only if the queue is full.
`int size()`	Returns the number of items in the queue.
`void enqueue()`	Inserts the element at the back of the queue.
`Item dequeue ()`	Deletes one element from the front of the queue.
`Item peek()`	Returns front item on the queue without deleting it.
`int action()`	Displays the action menu and returns the user's choice.
`void print()`	Print the elements of the queue from front to back.

Note 1: Be sure to test all methods and all exceptions as well.

Note 2: make sure to understand the concept of generics before implementing the class.

Step 2:

Now write a **Vaccine** application that will use the **myQueue** class you created above to simulate the management of the line of students who wait for COVID-19 vaccine in health center of the university.

Here are the assumptions:

- The health center daily vaccination capacity is X (i.e. it can vaccinate only X people on a given day).
- The students come in the priority risk groups that have a name and a size. For example Group1 with size 40, Group2 size 30 …
- It is guaranteed (i.e. you can assume) that the group size is not more than health center daily vaccine capacity (X).
- Health center must vaccinate groups with highest risk priorities first.
- Health center must fill its daily vaccinate capacity.

The program input file is `COVID19.txt`. The first line of the file contains the daily capacity number of the Health center. The following lines contain a name and a number, corresponding to the name of the group and the number of student in the group. The groups are in descending order of priority (i.e. the first group has the highest priority).

Input:

```
50
Group1 30
Group2 10
Group3 50
Group4 20
```

The output should have the following format: For each day, with a listing of the groups name and size which vaccinated on that day. See the example output format. Finally, print the total number of students that vaccinated.

Output:

```
Day1: Group1 30 Group2 10
Day2: Group3 50
Day3: Group4 20
Total Student: 110
Total Day: 3
```

WHAT TO HAND IN

A zip file containing:

- ➔ The Java source of your programs.
- ➔ The Java sources should be WELL DOCUMENTED as comments, as part of your grade will be based on the level of your comments.

The zip file should be uploaded to Moodle.

IMPORTANT

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

1. 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).
2. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.
3. If necessary, you may define additional data members and member functions.
4. For this assignment, 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 ...).
5. The submissions that do not obey these rules will not be graded.
6. 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.
7. 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: Linked list, STACK, QUEUE
// Author: Hamid Ahmadlouei
// ID: 2100000000
// Section: 0
// Assignment: 1
```

```
// Description: This class defines a linked list
//-------------------------------------------------------
```

8. Since your codes will 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...

9. This homework will be graded by your TAs, İbrahim İleri and Hamid Ahmadlouei (ibrahim.ileri@tedu.edu.tr, hamid.ahmadlouei@tedu.edu.tr). Thus, you may ask them your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course instructor Tolga Çapın and Ulaş Güleç for help.

GRADING RUBRICs

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	5%	Every class and method has a detailed explanation	Most classes and methods have an explanation,	Attempted to document the classes and methods, but	Only few comments.	Not even an attempt.

		(pre- and post-conditions, sample I/O, etc).	but missing in some parts.	they are not clear.		
Code Design	10%	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.
Abstract Data Type: linked list, array,	20%	Uses array and link list data structure for implementation own solution. And provides all functionality.	Uses array and link list data structure for own implementation. And provides most of expected functionalities.	Implements with major deviations from the specification.	Used different data structure to solve this problem.	Not even an attempt.
Functionality	50%	All functions 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 functions but some of them do not work.	Only few functions 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.