CSC 151 Programming Assignment 2

Objectives:

- Implementing solutions with Stacks, Queues, LinkedLists and Dictionaries
- Using java.util.LinkedList in traffic simulation
- Using java.util.Stacks for selling or ordering T-shirts
- Using java.util.Hashtable for implementing a Q&A session

THIS ASSIGNMENT HAS 3 PARTS

Do not forget to add parameterless constructors for your classes!!!

PART 1: STACKS

Definition

In this assignment you will use Stacks for selling or ordering T-shirts. You will have two classes called **TShirts** and **TShirtStack**. Any **TShirts** object has a size and color. Size can be and integer between 0-4 (0:S 1:M 2:L 3:XL 4:Big) and color can be a String. When you go shopping you will see the stacks of TShirts in 3 colors (Red, Green, Blue) and you will keep track of the sizes available and sold out in arrays. When a T-shirt is sold, update your counters in these arrays and check if you are below the limit. If any size is below 3 add 3 **TShirts** object to each color stack and update your counters.

Considering the specifications provided above, please write the following classes.

Class TShirts

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- Name of your file will be **TShirts.java** (case-sensitive)
- It will have **one private integer instance variable** *size* and **one private String variable** *color*.
- *size* can be an integer between **0 and 4** (use constant values for max and min values and verify these whenever you need to modify this value)
- *color* is a String and will have the value **"Red", "Green", "Blue"** and case insensitive.
- Provide one **public constructor** with 2 parameters for *size* and *color* and initialize the instance variables with the given values. Use set methods inside.
- Provide **public set/get** methods for each instance variable and a **setAll** method for setting both and do not forget to check restrictions for size.
- Override the **toString** method. When a **TShirts** object with *size*=0 and *color*="Red" is converted to a String it will be "size=S color=Red" (a space before color and \n after color value)
- Test your class in the main method.
- Test your class by Junit

Class TShirtStack

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- Name of your file will be **TShirtStack.java** (case-sensitive)
- It will have **three private Stack of TShirts instance variables** and their names are *red, green, blue* and two int arrays named as *sizesInStock* and *soldOut* for keeping counters. Use **java.util.Stack**
- Declare a constant *STOCK_LIMIT* as 3
- Provide a **public constructor** with no parameters and initialize the stacks and arrays
- Provide an **addTShirt** method that has a **TShirts** parameter and returns void. Add this object to one of the stacks depending on the color and update the array *sizesInStock* depending on the size of the object.
- Provide a **sell** method that has a String parameter as color and returns void. Sell the **TShirts** object on top of the stack of that color. Update the arrays after this operation and call order method which will check if any **TShirts** size is below *STOCK_LIMIT* and order accordingly.
- Provide an **order** method that has no parameters and returns void. Check the *sizesInStock* array and if any size is below *STOCK_LIMIT*, add *STOCK_LIMIT* number of **TShirts** of this size to each color stack. Use your **addTShirt** method. Assume that in *sizesInStock* array 0th index has 2 which is less than *STOCK_LIMIT*=3. Then add 3 small size **TShirts** to each color Stack(*red, green, blue*). This means you are adding 9 TShirts as a total.
- Override the **toString** method. Return the String representation of this object as follows:
  - If you recently created TShirtStack object, String representation will be:

    ```
    TShirt Stacks by color
    red=
    green=
    blue=
    sizesInStock=S:0 M:0 L:0 XL:0 Big:0
    soldOut=S:0 M:0 L:0 XL:0 Big:0
    ```
  - After adding a red and small size **TShirts** object it will be

    ```
    TShirt Stacks by color
    red=S
    green=
    blue=
    sizesInStock=S:1 M:0 L:0 XL:0 Big:0
    soldOut=S:0 M:0 L:0 XL:0 Big:0
    ```
- Test your class in the main method.

- Test your class by JUnit

<u>PART 2: QUEUES AND LINKEDLISTS</u>

<u>Definition</u>

In this assignment you will have any number of vehicles on a road. The vehicle’s speed can be minimum 5 mph and maximum 90 mph(A bit high don’t try this in real life!!!). They will line up one after the other. A vehicle can join the other any time and from any position, but speed adjustments are required depending on the speeds and position. You will have only one class called **Vehicles**. You will have a LinkedList for keeping the vehicles as an instance variable. When a new vehicle joins the list from position **pos**(an integer starting from zero), the vehicle should check the speed of the other vehicle in the front and slow down or speed up if it is faster or slower. The other vehicles should also decrease their speeds if they are faster/slower than the newly joined vehicle. See the rules and possible scenarios given below.

Considering the specifications provided above, please write the following class.

<u>Class Vehicles</u>

- Use the sample outputs provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- Name of your file will be **Vehicles.java** (case-sensitive)
- It will have **one private Integer LinkedList instance variable** named as *vehicles*. Use **java.util.LinkedList<Integer>** class. The data in this list will be the speed of the vehicles as an integer.
- The speed can be an integer between **5** and **90** (use constant values for max and min values and verify these whenever you need to modify this value or add a vehicle). No vehicle can stop(you have unlimited gas☺)
- Provide two **public constructors**
    - One with no parameter and initialize your **LinkedList** *vehicles* as empty.
    - One with an **ArrayList** parameter. This parameter has any number of speeds for various vehicles. First sort them in descending order and add these vehicles starting from the fastest one. For example, your **ArrayList** has [10, 25, 30, 12] then sort this as [30, 25, 12, 10] and using your add method add them to your **LinkedList** *vehicles*. You may use **Collections.sort()** and **Collections.reverse()**
- Override the **toString** method. When a **Vehicle** object with *vehicles*=[10, 9, 8] is converted to a String it will be “[10, 9, 8]”
- Provide a method named **addNewVehicle** which will have an integer speed and integer position parameter and the vehicle with the given speed will be added to the given position

- For each case do not forget to do the following: If speed is not in the valid range adjust to valid min or max value before adding.
- If list is empty, you can add the vehicle to first position
- If the position is higher than the size of the list, add as the last one but check the one ahead of it and adjust the speed as 1 less than that one.
- If the position is somewhere in between the other vehicles, first adjust its speed 1 less than the one immediately ahead and then speed up/slow down the other vehicles that come after by one unit
- If you need to slow down some vehicles but they reached to minimum leave the rest as the minimum.
- Examples:
    - Create a Vehicles object from an ArrayList=[10, 25, 30, 12]
        - add 30 to 0
        - add 25 to 1➔25 adjusted 29 speed up
        - add 12 to 2➔12 adjusted 28 speed up
        - add 10 to 3➔10 adjusted 27 speed up
        - result===[30, 29, 28, 27]
    - Create a Vehicles object as empty
        - add 20 to 0➔no adjustment
        - [20]
        - add 10 to 0➔20 adjusted 9 slow down
        - [10, 9]
        - add 30 to 2➔30 adjusted 8 slow down
        - [10, 9, 8]
        - add 5 to 3➔5 adjusted 7 speed up
        - [10, 9, 8, 7]
        - add 30 to 0➔all the rest adjusted speed up
        - [30, 29, 28, 27, 26]
        - add 98 to 2➔98 adjusted to max limit first and adjusted to 28 and the rest slow down
        - [30, 29, 28, 27, 26, 25]
        - add 3 to 10➔3 is less than min adjust to 5 first, size is less than 10 add as the last and adjusted 24 speed up
        - [30, 29, 28, 27, 26, 25, 24]
        - add 25 to 1➔25 adjusted 29 speed up rest slow down
        - [30, 29, 28, 27, 26, 25, 24, 23]
        - add 10 to 0➔all adjusted slow down, but final four can't slow down any more so leave as 5
        - [10, 9, 8, 7, 6, 5, 5, 5, 5]

- Other methods like set/get/equals are OPTIONAL.
- Test your class in the main method.
- Test your class by JUnit

PART 3: HASHTABLES

Definition

In this assignment you will use Hashtables for preparing a Question & Answer session. You will be given a long String which has the Yes/No questions and the next questions to be asked.

Example:

"Q1,Is it big?,Q2,Q5-Q2,Is it white?,Q3,Q4-Q3,Whale,,-....

In this String questions are separated by – and for each question you have a comma separated list including question id, question text, next question id to ask if the answer is yes/no. In the given example above **Q1** is the id of the first question and ***Is it big?*** is the question text and **Q2** is the next question id if the answer is yes and **Q5** is the next question id if the answer is no. After – next question whose id is **Q2** starts.

Considering the specifications provided above, please write the following classes.

Class QA

- Use the sample output provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- Name of your file will be **QA.java** (case-sensitive)
- This is the representation of a binary question that has either a yes or no answer.
- It will have **4 private String instance variables** *qid*, *text*, *yesQID* and *noQID*. Each question has an id which is not numeric(*qid*), the question text to ask the user(*text*), and the next question id for the yes and no answers(*yesQID* and *noQID*)
- Provide a **public parameterless constructor**
- Provide one **public constructor** with 4 String parameters for *qid*, *text*, *yesQID* and *noQID* and initialize the instance variables with the given values.
- Provide one **public constructor** with one String parameter for a question that has *qid*, *text*, *yesQID* and *noQID* parts as comma separated. Split this String and assign the values to the instance variables.
- Provide **public set/get** methods for each instance variable.
- Optional: Override the **toString** method.
- Test your class in the main method.
- Test your class by JUnit

<u>Class QuizHashTable</u>

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- Name of your file will be **QuizHashTable.java** (case-sensitive)
- It will have **a private Hashtable** whose key is a String for qid and value is a QA object. Use java.util.Hashtable and name it as *quiz*.
- Provide a **public constructor** with a String parameter. The parameter has all quiz questions separated by – and each question has the *qid*, *text*, *yesQID* and *noQID* fields as comma separated. First split the String into questions and for each question create a QA object. Then put the question object into the Hashtable by using the qid as the key and QA object as the value.
- Provide a **public method ask** which has two String parameters and returns a String. First parameter is the question id to start asking and the second parameter is a sequence of characters formed from *Y* and/or *N*. These characters are the answers given by the user. Start from the given question, retrieve the data for that question from the Hashtable, add the question text to the output and depending on the next character in the user answer proceed with the Yes or No branch until there are no more answers or no more questions left. Add the answer character to the output and put a newline. If there are no more questions, just add the text of it to the output but not the answer character. Return the output
- Optional: Override the **toString** method.
- Test your class in the main method.
- Test your class by Junit

General Submission Instructions

- You may have groups for the assignments **if it is allowed.** If you have a group state it **explicitly in the honor code** by writing all group members' names. Please name the pdf files accordingly. **However, all group members should submit individually.**
- All submissions require a package called **assignment** and your source codes (.java files) will be under it.
- You should submit
  - Your java files to **Gradescope** as required and
  - Your java files and sample outputs for various scenarios as a **single pdf** file named as **YourFullName(s)_AssignmentN.pdf** with the **honor code** at the top as a comment to Nexus under the appropriate link. Replace N in AssignmentN with the appropriate value. (e.g., if I submit Assignment1 I will submit a pdf file whose name is **ZeynepOrhan_Assignment1.pdf.** If I have a group whose members are me and Jane Doe, then the file name will be **JaneDoeZeynepOrhanAssignment1.pdf** and the honor code will be modified). Use the template .docx file attached and modify it accordingly.
- **If you use any resource to get help, please state it in the honor code. Otherwise, it will be an honor code violation!!!**
- **Gradescope provides a feature to check code similarity by comparing all submissions. If the percentage of your code's similarity is above a threshold, then this will be an honor code violation!!!**
- Make sure that you have no compiler errors.
- When you have compiler error free codes and submit appropriately to Gradescope, your code will be tested automatically and graded. You do not need to know the test cases, but you will see the test results. Informative messages will be provided for success and failure cases.
- You can resubmit your files any number of times if you have failing tests until the due date.
- Your final submission or the one you selected will be graded after the due date of the assignment.
- Please start as early as possible so that you may have time to come and ask if you have any problems.
- Read the late submission policy in the syllabus.
- Please be aware that the correctness, obeying the OOP design and submission rules are equally important

- Gradescope testing will generally be 100% and sometimes manual grading will be done. Please be aware that **your grade can be reduced if you do not follow the instructions and apply good programming practices**.
- Use the starter code if provided.
- Do not use any predefined Collection classes of Java unless stated otherwise.

<u>Assignment specific instructions</u>

- General submission instructions apply.
- **<u>Group work:</u> ALLOWED(You may have a group of 3-4 students)!!! However, you should submit the same files individually!!!**
- **<u>Due date:</u>** February 20, 2021, until 11:50 PM
- **<u>Nexus:</u>** Submit the code files (**TShirts.java, TShirtStack.java, Vehicles.java, QA.java, QuizHashTable.java**) as a single pdf file named as **YourFullName_PRA2.pdf** with the honor code at the top as a comment under Programming Assignment 2 link.
- **<u>Gradescope:</u>** Submit the java files (**TShirts.java, TShirtStack.java, Vehicles.java, QA.java, QuizHashTable.java**) under the Programming Assignment 2
- **<u>Grading:</u>**
  - Gradescope: Yes
  - Manual: Yes
- **<u>Starter code:</u>** Use the code given below

PART I

```
/*
* Honor Code
*/
package assignment;
import java.util.Stack;

public class TShirts {
      //0:S 1:M 2:L 3:XL 4:Big Size You may use a String array



    //Constant values for MAX_SIZE and MIN_SIZE



    //Instance variables


        /**
         * @param size
         * @param color
         */
    //Constructor


    //Set/get methods


    //toString method

        @Override

    //main method. Please change this completely
        public static void main(String[] args) {
                String colors[]= {"Red","Green","Blue"};

                TShirts tsh[]=new TShirts[4];
                for(int i=0;i<tsh.length;i++)
                        tsh[i]=new TShirts(i,colors[i%3]);
                for(int i=0;i<tsh.length;i++)
                        System.out.print(tsh[i]);
    }
}
```

```
package assignment;
import java.util.Arrays;
import java.util.Stack;
public class TShirtStack {
	//0:S 1:M 2:L 3:XL 4:Big Size You may use a String array


	//Constant values for STOCK_LIMIT


	//Instance variables red, green, blue Stacks
	//and int arrays sizesInStock and soldOut


	//Parameterless Constructor initialize arrays and stacks


	//addTShirt method


	//sell method


	//order method

	//toString method
	@Override

	//main method. Please change this completely
	public static void main(String[] args) {
		String colors[]= {"Red","Green","Blue"};
		TShirtStack tsh=new TShirtStack();
		System.out.println(tsh);
		for(int i=0;i<5;i++) {
			tsh.addTShirt(new TShirts(i % (1+TShirts.MAX_SIZE), colors[i%3]));
			System.out.println(tsh);
		}
		tsh.sell("red");
		System.out.println(tsh);
		tsh.sell("green");
		System.out.println(tsh);
		tsh.sell("blue");
		System.out.println(tsh);
	}
}
```

The output of the main of TShirts class above is

```
size=S color=Red
size=M color=Green
size=L color=Blue
size=XL color=Red
```

The output of the main of TShirtStack class above is

```
========================================
TShirt Stacks by color
red=
green=
blue=
sizesInStock=S:0 M:0 L:0 XL:0 Big:0
soldOut=S:0 M:0 L:0 XL:0 Big:0

TShirt Stacks by color
red=S
green=
blue=
sizesInStock=S:1 M:0 L:0 XL:0 Big:0
soldOut=S:0 M:0 L:0 XL:0 Big:0

TShirt Stacks by color
red=S
green=M
blue=
sizesInStock=S:1 M:1 L:0 XL:0 Big:0
soldOut=S:0 M:0 L:0 XL:0 Big:0

TShirt Stacks by color
red=S
green=M
blue=L
sizesInStock=S:1 M:1 L:1 XL:0 Big:0
soldOut=S:0 M:0 L:0 XL:0 Big:0

TShirt Stacks by color
red=S,XL
green=M
blue=L
sizesInStock=S:1 M:1 L:1 XL:1 Big:0
soldOut=S:0 M:0 L:0 XL:0 Big:0

TShirt Stacks by color
red=S,XL
green=M,Big
blue=L
sizesInStock=S:1 M:1 L:1 XL:1 Big:1
```

```
soldOut=S:0 M:0 L:0 XL:0 Big:0

TShirt Stacks by color
red=S,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big,Big
green=M,Big,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big,Big
blue=L,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big,Big
sizesInStock=S:10 M:10 L:10 XL:9 Big:10
soldOut=S:0 M:0 L:0 XL:1 Big:0

TShirt Stacks by color
red=S,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big,Big
green=M,Big,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big
blue=L,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big,Big
sizesInStock=S:10 M:10 L:10 XL:9 Big:9
soldOut=S:0 M:0 L:0 XL:1 Big:1

TShirt Stacks by color
red=S,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big,Big
green=M,Big,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big
blue=L,S,S,S,M,M,M,L,L,L,XL,XL,XL,Big,Big
sizesInStock=S:10 M:10 L:10 XL:9 Big:8
soldOut=S:0 M:0 L:0 XL:1 Big:2
```

PART II

Examples(these are just some example scenarios for you)

No parameter for constructor

[]

****

Parameter for constructor:[10, 35, 20, 17]

add 35 to 0

add 20 to 1

add 17 to 2

add 10 to 3

===[35, 34, 33, 32]

****

Start with empty list and do the followings:

add 18 to 0

[18]

add 9 to 0

[9, 8]

add 25 to 2

[9, 8, 7]

add 6 to 3

[9, 8, 7, 6]

add 20 to 0

[20, 19, 18, 17, 16]

add 100 to 2

[20, 19, 18, 17, 16, 15]

add 1 to 20

[20, 19, 18, 17, 16, 15, 14]

add 23 to 1

[20, 19, 18, 17, 16, 15, 14, 13]

add 9 to 0

[9, 8, 7, 6, 5, 5, 5, 5, 5]

PART III

Here is how I will test for the Hashtable (just an example).

//All questions are given in rawQ String

public static final String ***rawQ***="Q1,Is it big?,Q2,Q5-Q2,Is it white?,Q3,Q4-Q3,Whale, , -Q4,Cat, , -Q5,Is it an animal?,Q6,Q7-Q6,Ant, , -Q7,Dust, , ";

//Possible answers from the user

public static final String ***ans1***="YY";

public static final String ***ans2***="YN";

public static final String ***ans3***="NY";

public static final String ***ans4***="NN";

//First get all questions and in the constructor split them.

//Use - as the separator.

//Store the parts that correspond to single questions in the array

// For each question(array element) call the constructor of QA and create a QA object.

// QA constructor gets a String parameter and splits it into parts by using , as the separator

//Insert these QA objects into Hashtable quiz by using the qid field as the key and QA object as the value

QuizHashTable qz=**new** QuizHashTable(***rawQ***);

//res is the output you will generate and return from the ask method

String res;

// ask method gets a question id to start and an answer sequence

//For the following start from question whose id is Q1 and answer sequence is ans1 i.e. YY

res=qz.ask("Q1", ***ans1***);

System.***out***.println(res);

// Q1 is Is it big?

```
// and first answer is Y

//So ask the question Is it an animal?

//and the second answer is Y

//Since there is no more question stop and give the answer as the text of the current QA object which
is Ant

//Output res will be

/*

Is it big?Y

Is it an animal?Y

Ant

*/


res=qz.ask("Q1", ans2);

System.out.println(res);

// See the explanations above for details. This time we start again from Q1 but answer sequence ans2
is YN

//Output is

/*

Is it big?Y

Is it white?N

Cat

*/



res=qz.ask("Q1", ans3);

System.out.println(res);

// See the explanations above for details. This time we start again from Q1 but answer sequence ans3
is NY

//Output is
```

```
/*

Is it big?N

Is it an animal?Y

Ant

*/



res=qz.ask("Q1", ans4);

System.out.println(res);

// See the explanations above for details. This time we start again from Q1 but answer sequence ans4
is NN

//Output is

/*

Is it big?N

Is it an animal?N

Dust

*/
```

Example 1:
Questions as a String
"Q1,Is it big?,Q2,Q5-Q2,Is it white?,Q3,Q4-Q3,Whale, , -Q4,Cat, , -Q5,Is it an animal?,Q6,Q7-Q6,Ant, , -Q7,Dust, , "
Answers as a String
“YY”
Output

```
Is it big?Y
Is it white?Y
Whale
```

Example 2:
Questions as a String
"Q1,Is it big?,Q2,Q5-Q2,Is it white?,Q3,Q4-Q3,Whale, , -Q4,Cat, , -Q5,Is it an animal?,Q6,Q7-Q6,Ant, , -Q7,Dust, , "
Answers as a String
“YN”
Output

```
Is it big?Y
Is it white?N
```

Cat

Example 3:

Questions as a String

"Q1,Is it big?,Q2,Q5-Q2,Is it white?,Q3,Q4-Q3,Whale, , -Q4,Cat, , -Q5,Is it an animal?,Q6,Q7-Q6,Ant, , -Q7,Dust, , "

Answers as a String

“NY”

Output

```
Is it big?N
Is it an animal?Y
Ant
```

Example 4:

Questions as a String

"Q1,Is it big?,Q2,Q5-Q2,Is it white?,Q3,Q4-Q3,Whale, , -Q4,Cat, , -Q5,Is it an animal?,Q6,Q7-Q6,Ant, , -Q7,Dust, , "

Answers as a String

“NN”

Output

```
Is it big?N
Is it an animal?N
Dust
```