COE528 (Winter 2024) - Lab4

General Lab Rules

All the necessary files for this lab should be in the lab4 directory.

All the Java files in this lab should have the following package declaration:

```
package coe528.lab4;
```

Duration: Two weeks.

Objectives

- Provide and implement abstraction function and rep invariant.

Exercise 1:

Abstract Concept of a stack of distinct strings:

A stack of distinct strings, *p*, is a collection of strings with a *top* where a string cannot exist more than once in the collection. For example, the collection of strings{“ab”, “cd”, “ae”, “bd”} is a stack with *top* = “bd”. The only operations supported are the addition of a string at the *top,* known as *push*, and the deletion of a string from the *top,* known as *pop*.

Implementation of a stack of distinct strings:

The following class, `StackOfDistinctStrings`, represents a stack of distinct strings.

Click on Project > New Project in the Netbeans program and save it as "Ex1" on your lab4 directory.

Create a new class called “`StackOfDistinctStrings`” and copy the code in it. For this class:

a) Write the abstraction function in the Overview clause.

b) Write the rep invariant in the Overview clause.

b) Fill in the body of the method `repOK`().

b) Fill in the body of the method `toString` ().

```
import java.util.ArrayList;

public class StackOfDistinctStrings {

    // Overview: StacksOfDistinctStrings are mutable, bounded
    // collection of distinct strings that operate in
    // LIFO (Last-In-First-Out) order.
    //
    // The abstraction function is:
    // a) Write the abstraction function here
    //
    //
    //
    //
    // The rep invariant is:
    // b) Write the rep invariant here
    //
    //
    //
    //

    //the rep
    private ArrayList<String> items;

    // constructor
    public StackOfDistinctStrings() {
        // EFFECTS: Creates a new StackOfDistinctStrings object
        items = new ArrayList<String>();
    }


    public void push(String element) throws Exception {
        // MODIFIES: this
        // EFFECTS: Appends the element at the top of the stack
        //          if the element is not in the stack, otherwise
        //          does nothing.
        if(element == null) throw new Exception();
        if(false == items.contains(element))
            items.add(element);
    }


    public String pop() throws Exception {
        // MODIFIES: this
        // EFFECTS: Removes an element from the top of the stack
        if (items.size() == 0) throw new Exception();
        return items.remove(items.size()-1);
    }
```

```
    public boolean repOK() {
        // EFFECTS: Returns true if the rep invariant holds for this
        //          object; otherwise returns false
        // c) Write the code for the repOK() here
    }

    public String toString() {
        // EFFECTS: Returns a string that contains the strings in the
        //          stack and the top element. Implements the
        //          abstraction function.
        // d) Write the code for the toString() here
    }
}
```

Exercise 2:

Abstract Concept of a Queue:

A queue is a collection of elements where the first added element is the first to be removed. It follows the FIFO (First-In-First-Out) principle. For example, imagine a queue at a ticket counter; the person who enters the queue first gets served first.

Design and Implementation of a Queue Abstract Data Type (ADT):

Design and implement a queue ADT with the following specifications:

- The queue should support generic types to allow for flexibility in the type of elements it can hold.
- Implement methods to enqueue (add) an element to the rear end of the queue, dequeue (remove) an element from the front end of the queue, and check if the queue is empty.
- Write appropriate documentation for the queue class and methods, including an overview, abstraction function, rep invariant, and any necessary comments.
- Ensure that the queue's internal representation is encapsulated properly.
- Implement the repOK() method to check if the rep invariant holds true for the current state of the queue.
- Implement the toString() method to provide a string representation of the queue's elements.

Your task:

1. Define the abstract concept of a queue based on the provided description.
2. Design the queue ADT by specifying its methods and behaviours.
3. Implement the queue ADT in Java, ensuring proper encapsulation and adherence to the defined specifications.
4. Implement the provided main function to test the functionality of your queue implementation.
5. Submit your solution with the main function and ensure your implementation produces the expected output.

Here's the provided main function:

```
public class Main {
    public static void main(String[] args) {
        // Test the queue implementation
        Queue<Integer> queue = new Queue<>();

        // Enqueue elements
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);

        // Dequeue elements
        int dequeuedElement1 = queue.dequeue(); // 10
        int dequeuedElement2 = queue.dequeue(); // 20

        // Check if the queue is empty
        boolean isEmpty = queue.isEmpty(); // false

        // Expected output
        System.out.println("Dequeued element 1: " + dequeuedElement1);
        System.out.println("Dequeued element 2: " + dequeuedElement2);
        System.out.println("Is the queue empty? " + isEmpty);
    }
}
```

Submitting your lab

Deadline: The submission deadline is 11:59 PM EST, the night before the corresponding lab session during the week of February 26, 2024.

You must include the duly filled and signed standard cover page with your submission. The cover page can be found on the departmental website: Standard Assignment/Lab Cover Page

You must submit your lab electronically on D2L. Please zip up your NetBeans project containing all source files and submit it to the respective assignment folder on D2L.