Lab Exercise 3: Linear Data Structures [10 Marks]

SE2205: Data Structures and Algorithms using Java – Fall 2023

Open Day: October 1, 2023; **Cut off time**: October 10, 2023, Tuesday @11am

---

Prepared by Dr. Quazi Rahman (qrahman3@uwo.ca).

A. Rationale and Background

In this lab Assignment, you will demonstrate your understanding of the Stack and Queue data structures with the aid of an array. Here you will reuse Generic-class Pair <Y, N> that you created in Lab 2.

B. Evaluation and Submission Instructions

Submit your Lab online by carrying out the following instructions:

1. Create a Project with the following name: *username_Lab3*
2. For this question create a package called LA3Q1.
3. Use the names for each class and the associated variables as shown in the UML diagram. Where the name 'Quazi' is shown, replace it with your first name.
4. For this question, use the static header and footer methods you created before and modify it as you see fit.
5. Comments: **Writing short but clear comments for Labs is mandatory**. If the comments are missing, full credit may not be awarded.
6. Once the lab is completed, go to your Lab folder. Select the project folder (e.g. *username_Lab3*). Right-click to select 'Send to' 'Compressed zipped folder'. Make sure it has the same naming convention (e.g., *usernames_Lab3*.zip). Upload this file to OWL as your submission.
7. You will be graded on this lab Assignment based on the comments and running code.
8. Please note the deadline: Tuesday/10/Oct. @11 am, and it is a hard deadline.
9. If your code does not run, you will be awarded a zero-grade.
10. If you submit class-file by mistake, you will be awarded a zero-grade.

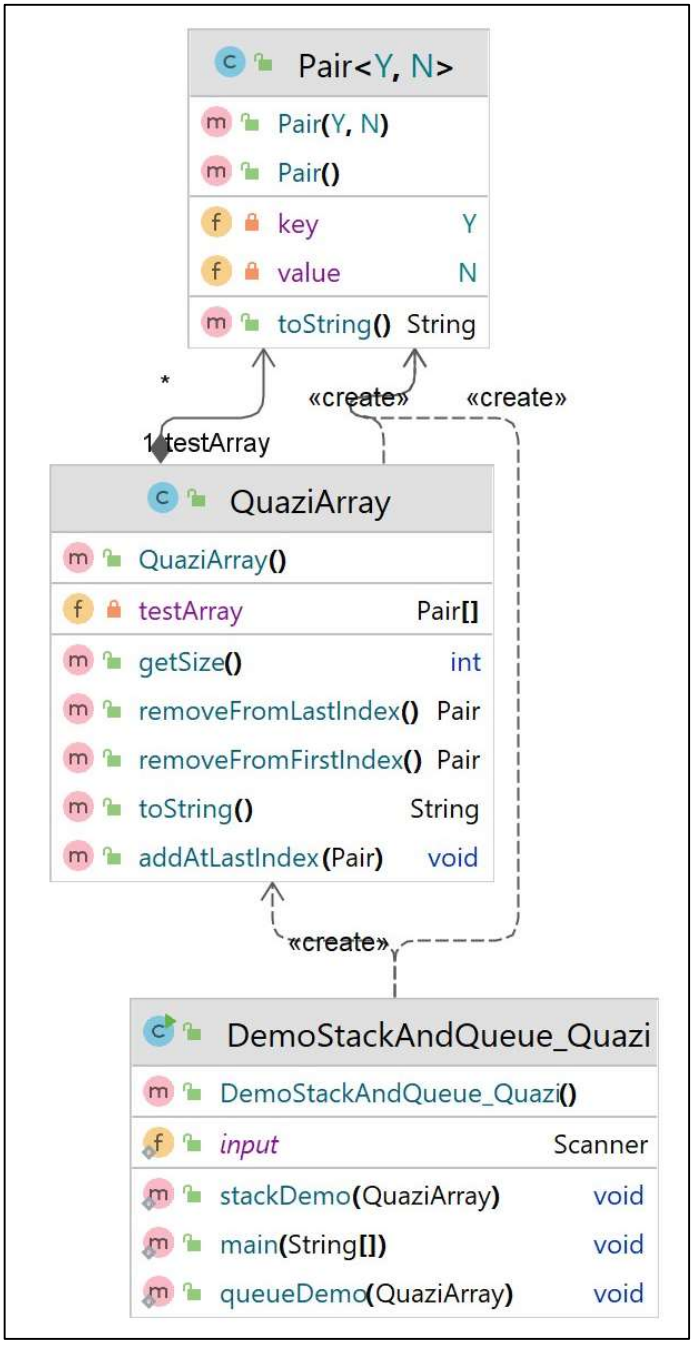
C. Lab Question

1. Question 1 [10 Marks]

Here you need to create three classes as shown in the UML diagram below, and these classes are a generic **Pair <Y, N>** class (reuse the one you created in lab 2), a concrete class called **YourFirstNameArray**, which will depend on Pair Class, and a driver class called **DemoStactAndQueueYourFirstName**. See the UML Diagram. The specifications follow:

(a) Define a generic class called *Pair<Y, N>* with the following specifications (see the class diagram below):

- i) Two private fields: key and value of Y and N types, respectively.
- ii) Two constructors: without parameter and with parameter. For without-parameter one, you can leave the method-body empty.
- iii) The overridden **toString()** method; it should return a String containing Y and N type values as [YR: Y, NM: N], where Y and N are object references, and YR and NM are hardcoded acronyms for year and name, respectively.



(b) Define a second class called ***YourFirstNameArray*** (e.g., I named it using my first name as *QuaziArray*) with the following specifications:

i) A **private Pair**-type array field called **testArray**.

ii) The constructor without parameter that will initialize the array with a size 0.

iii) A helper method **getSize()** which will return the length of the array.

iv) **removeFromLastIndex()** method: It removes the element from the last index of the array and returns the element, and restructures the array with the new length, which is one smaller than the original one.

v) **removeFromFirstIndex()** method: It removes the element from the first index of the array and returns the element, and restructure the array with the new length, which is one smaller than the original one.

vi) **toString()** method: Override the Object's **toString()** method which should return the list as a string as shown in the sample output. (Talk to your instructor if you need any help.)

vii) **addAtLastIndex()** method: It restructures the array with the new length, which is one larger than the original one, and then adds the new element at the last index of the array.

(c) Now define the driver class called ***DemoStackAndQueue_YourFirstName*** with the following specifications:

i) A **public static** field of **Scanner** type duly instantiated.

ii) An empty parameter constructor: this is optional.

iii) The header and the footer methods: You should copy-paste these methods from your previous lab exercise solution and modify it according to the current problem.

iv) **stackDemo()** method: It is a **public static void** method that will accept a *YourFirstNameArray* (e.g., *QuaziArray*) type parameter and demonstrate the stack operations. Check the specifications in section (d).

v) **queueDemo()** method: It is a **public static void** method that will accept a *YourFirstNameArray* (e.g., *QuaziArray*) type parameter and demonstrate the queue operations. Check the specifications in section (e).

vi) **main()** method: Check the specifications in section (f).

(d) Specifications for **stackDemo()** method (See the method signature and the return type in the UML diagram):

i) Assumption: The last index of the array is the top of the stack where the push and pop operations will take place.

ii) You need to carry out the demo with the help of the methods you defined in the *QuaziArray* class.

iii) Once called from main (), this method will receive a *QuaziArray* Type array reference that refers to an array of size zero, and then print the message – "*You have an empty stack*", and it will print the empty stack as [] with the help of toString() method you defined in the *QuaziArray* Class. See the sample output.

iv) Then, from inside a loop it should provide three options via a menu for *push*, *pop* and *exit*. Get the user's choice and validate user's choice (talk to the instructor if there is any confusion).

v) As shown in the sample output, when the user enters ***a*** for push, it will ask the user to enter the *year of study* and *name of the student*. Based on these values the new Pair-item should be pushed to the stack with the help of the **addAtLastIndex()** method from the *QuaziArray* Class.

vi) After this data is pushed, the revised array needs to be displayed on the screen as shown in the sample output with the help of the toString() method defined in the *QuaziArray* class.

vii) If the user chooses to pop a data item, it will return the popped item, and then the revised stack will be displayed. If the stack is already empty but the user wants to pop an item, the code should provide the message (see the sample output) "FYI: The stack is empty".

viii) Now with the help of the stack-demo menu, the user should be able to push and pop the data items till the user decides to exit. Once the user decides to exit, this method will close.

(e) Specifications of **queueDemo()** method (See the method signature and the return type in the UML diagram):

i) Assumption: The last index of the array is the rear of the queue where the data will be enqueued, and the first index of the array is the front of the queue from where the data will be dequeued.

ii) You need to carry out the demo with the help of the methods you defined in the *QuaziArray* class.

iii) Once called from main (), this method will receive a *QuaziArray* Type array reference that refers to an array of size zero, and then print the message – "*You have an empty stack*", and it will print the empty stack as [] with the help of toString() method you defined in the *QuaziArray* Class. See the sample output.

iv) Then, from inside a loop it should provide three options via a menu for *enqueue*, *dequeue* and *exit*. Get the user's choice and validate user's choice (talk to the instructor if there is any confusion).

v) As shown in the sample output, when the user enters ***a*** for enqueue, it will ask the user to enter the *year of study* and *name of the student*. Based on these values the new Pair-item should be enqueued with the help of the **addAtLastIndex()** method from the *QuaziArray* Class.

vi) After this data is enqueued, the revised queue (the array) needs to be displayed on the screen as shown in the sample output with the help of the toString() method defined in the *QuaziArray* class.

vii) If the user chooses to dequeue a data item, it will return the dequeued item from index zero (the front of the queue), and then the revised queue (the array) will be displayed. If the queue is already empty but the user wants to dequeue an item, the code should provide the message (see the sample output) "FYI: The queue is empty".

viii) Now with the help of the queue-demo menu, the user should be able to enqueue and dequeue the data items till the user decides to exit. Once the user decides to exit, this method will close.

(f) Specifications of the main () method:

i) Call the header method.

ii) Test part: Before writing the rest of the code in the main(), test that your code is printing the values correctly. In this case, declare a Pair-type reference variable and instantiate it with an Integer (e.g., 3) and a String (e.g., John) values. Print that variable with the help of toString() method you defined in the Pair class. It should print: [YR: 3, NM: John]. If this test is successful, just comment-out this part of the code, and do the rest of the following.

iii) Declare a QuaziArray type reference variable. E.g., **QuaziArray x**;

iv) Now from inside a loop generate a menu with three choices: 1 for Stack, 2 for Queue and 3 for Exit (see the sample output).

v) If the user chooses either 1 or 2, the reference variable **x** that has been declared in (iii) will be instantiated, and then the **stackDemo()** method (for option 1) or the **queueDemo()** method (for option 2) will be called by passing this instantiated reference there.

vi) Once the user enters 3 as option, the process will stop with a goodbye message.

vii) Call the footer method.

viii) Optional: once you are done – you may add the Exceptions to make your code user friendly.

Sample Output:

```
=====================================================
Lab Exercise 3
Prepared By: Quazi Rahman
Student Number: 999999999
Goal of this Exercise: ........!
=====================================================

This code performs a demo for Stack and Queue:

Main Demo-Menu:
        1: Stack
        2: Queue
        3: Exit
        Enter your choice: 1

You have an empty stack: []

Stack Operation Menu:
```

```
a: Push
b: Pop
c: Exit
Enter your choice: 1
Invalid choice!

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: w
Invalid choice!

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: a
Let's push a data-item into the stack....
Enter year: 2
Enter name: qqq
The current stack: [[YR: 2, NM: qqq]]

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: a
Let's push a data-item into the stack....
Enter year: 3
Enter name: ttt
The current stack: [[YR: 2, NM: qqq][YR: 3, NM: ttt]]
Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: 2
Invalid choice!

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: a
Let's push a data-item into the stack....
Enter year: 2
```

```
Enter name: sss
The current stack: [[YR: 2, NM: qqq][YR: 3, NM: ttt][YR: 2, NM: sss]]

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: b
Let's pop a data-item ....
[YR: 2, NM: sss] is removed! The current stack: [[YR: 2, NM: qqq][YR: 3, NM: ttt]]

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: b
Let's pop a data-item ....
[YR: 3, NM: ttt] is removed! The current stack: [[YR: 2, NM: qqq]]

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: b
Let's pop a data-item ....
[YR: 2, NM: qqq] is removed! The current stack: []

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: b
Let's pop a data-item ....
FYI: The stack is empty!

Stack Operation Menu:
a: Push
b: Pop
c: Exit
Enter your choice: c
Ending Stack-demo! Goodbye!

Main Demo-Menu:
        1: Stack
        2: Queue
        3: Exit
```

```
        Enter your choice: 2

You have an empty queue: []

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: a
Let's enqueue a data-item in the queue....
Enter year: 3
Enter name: zzz
The current queue: [[YR: 3, NM: zzz]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: a
Let's enqueue a data-item in the queue....
Enter year: 2
Enter name: ggg
The current queue: [[YR: 3, NM: zzz][YR: 2, NM: ggg]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: a
Let's enqueue a data-item in the queue....
Enter year: 2
Enter name: ddd
The current queue: [[YR: 3, NM: zzz][YR: 2, NM: ggg][YR: 2, NM: ddd]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: a
Let's enqueue a data-item in the queue....
Enter year: 4
Enter name: aaa
The current queue: [[YR: 3, NM: zzz][YR: 2, NM: ggg][YR: 2, NM: ddd][YR: 4, NM: aaa]]

Queue Operation Menu:
a: Enqueue
```

b: Dequeue
c: Exit
Enter your choice: b
Let's dequeue a data-item ....
[YR: 3, NM: zzz] is dequeued! The current queue: [[YR: 2, NM: ggg][YR: 2, NM: ddd][YR: 4, NM: aaa]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: b
Let's dequeue a data-item ....
[YR: 2, NM: ggg] is dequeued! The current queue: [[YR: 2, NM: ddd][YR: 4, NM: aaa]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: a
Let's enqueue a data-item in the queue....
Enter year: 2
Enter name: uuu
The current queue: [[YR: 2, NM: ddd][YR: 4, NM: aaa][YR: 2, NM: uuu]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: b
Let's dequeue a data-item ....
[YR: 2, NM: ddd] is dequeued! The current queue: [[YR: 4, NM: aaa][YR: 2, NM: uuu]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: b
Let's dequeue a data-item ....
[YR: 4, NM: aaa] is dequeued! The current queue: [[YR: 2, NM: uuu]]

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: b
Let's dequeue a data-item ....

[YR: 2, NM: uuu] is dequeued! The current queue: []

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: b
Let's dequeue a data-item ....
FYI: The queue is empty!

Queue Operation Menu:
a: Enqueue
b: Dequeue
c: Exit
Enter your choice: c
Ending Queue-demo! Goodbye!

Main Demo-Menu:
1: Stack
2: Queue
3: Exit
Enter your choice: 3
Goodbye!

==========
Completion of Lab Exercise 3 is successful!
Signing off - Quazi
==========

/*