SWE2721 Lab 2: Testing a Circular Queue with TestNG

1. Introduction

This is your first verification lab where you will be developing Unit tests. Its purpose is to help you to start thinking about how to test existing software. From doing this, you will review the concepts of TestNG as well as begin to see some of the patterns necessary to ensure correct operation of software.

2. Lab Objectives

- Review the Java queue data structure.
- Develop TestNG test cases to properly verify the operation of a basic Java data structure.
- Use independent learning to understand the operation of a circular queue.

3. Deliverables / Submission

Now that you have completed your lab assignment, tag your final version (which should be on the trunk) with the label LabSubmission1.0. Your test code should be thoroughly commented and cleanly build without warnings, and should have header java doc at the top of the file. Additionally, each file that you have changed should have comments added to it with your name included.

In addition, each person will be responsible for submitting one report with the following contents:

1. Title Page
    a. Student Name
    b. Date
    c. Assignment Title
    d. Repository Link (linkable preferred)
2. Introduction
    a. What are you trying to accomplish with this lab?
        i. This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment. It should be multiple grammatically correct sentences.
3. Testing Strategy
    a. What strategy did you use to create your test cases?
    b. How did you go about organizing the tests in the test file?
4. Fault Locations (For this lab, this can be a simple table with line number(s) and descriptions of the bugs. You do not need to file bug reports)
    a. What problem(s) did your testing find?
    b. Where did you make a change to fix the bug?
5. Discussion
    a. How did the AAA test approach help you to structure your test cases?
    b. On the V Model, what type of test are you writing here? Why did you provide this answer?
6. Things gone right / Things gone wrong.
    a. This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
7. Conclusions
    a. What have you learned from this experience?
    b. What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file in Canvas. If you have any questions, consult your instructor.

4. Problem Overview

One commonly used low level data structure is the circular queue or circular buffer. It is often used in systems which require fast adds and removes, as unlike other forms of queues, additions and removals can be designed to be atomic without the usage of blocks.

For this lab, you have been provided with a full implementation for a Java circular queue class. However, this code has not been tested and will most likely contain errors. (OK, not most likely. It WILL contain errors.) You are responsible for developing TestNG tests to ensure that this class operates properly. If there are problems with the implementation, you will need to go into the source code and perform the appropriate fixes.

It is possible that there are some methods which are not implemented in the source code. If the methods are not implemented, the code must indicate this by throwing an UnsupportedOperationException. For methods which are unsupported, make sure that the appropriate exception is thrown when the methods are invoked.

The queue, as it is structured, **should not** be capable of handling null references. Therefore, you should make certain that null cannot be added to the queue as well, throwing the appropriate exception as is applicable. The Javadoc should reflect this.

Complete Javadoc for the project is available in the doc folder of the repository. You'll want to refer to this as you are performing your implementation.

5. Lab Specifics

Working individually, you are to write a set of test cases which will fully exercise this module. Each method should be tested, and you should have good enough coverage to ensure that the methods are thoroughly tested. Your tests should be structured to use **AAA practices**. You will have a GitHub repository to work from. This repo will have the starting code base and other materials you will need. There is a document showing where your test cases should be placed. Please DO NOT change the packaging of the project.

As you are working, you will need to make sure the tests you write are placed in the 'student' group so that they will execute when 'student' is enabled. To make this work, you will need to make sure the tests are placed in that group as your write them, and also that the group is enabled by modifying the build.gradle file. The build.gradle file controls which tests are executed, and you will need to make the student group executable by making the change shown below.

Initial Provided build.gradle file segment related to testing	Modified file to include student group executing.

test {
    useTestNG()
    {
      includeGroups 'demo'
      excludeGroups
    }
.
.
.
	test {
	    useTestNG()
	    {
	      includeGroups 'demo', 'student'
	      excludeGroups
	    }
	.
	.
	.

When finished, you should have a completely working circular queue implemented in Java. Any bugs that are fixed should be noted in code with a small comment indicating why you were making the change and provide a number relating the bug fix back to the source defect number. (Note: You will not be filing format defect reports for this exercise like you did in Lab 1. Rather, you simply need to keep a table showing the bug ID (a number) and what the problem was that was found, as well as mapping to the code where the change / fix was made.) For example, the following will work:

Bug ID	Bug Description	Line number to fix bug (Note that this will change as other bugs ae fixed.)
1	When attempting to trim the empty string, the module throws a null pointer exception.	Stringizer.java:217
2	When the user passes in a string that is all symbols to the digitizer class, an exception is thrown by the digitizer addDescription method.	Digitizer.java:314159265

The UML for this is shown in Figure 1.

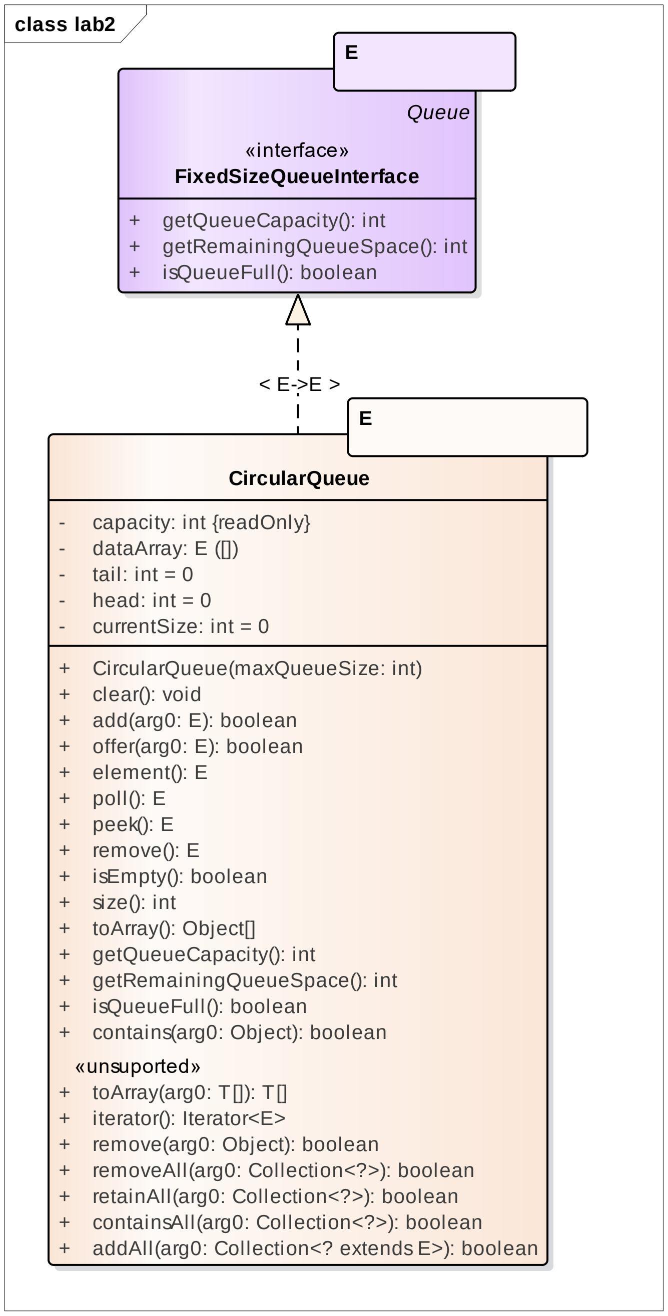
class lab2
E
Queue
«interface»
FixedSizeQueueInterface
+ getQueueCapacity(): int
+ getRemainingQueueSpace(): int
+ isQueueFull(): boolean
< E->E >
E
CircularQueue
- capacity: int {readOnly}
- dataArray: E ([])
- tail: int = 0
- head: int = 0
- currentSize: int = 0
+ CircularQueue(maxQueueSize: int)
+ clear(): void
+ add(arg0: E): boolean
+ offer(arg0: E): boolean
+ element(): E
+ poll(): E
+ peek(): E
+ remove(): E
+ isEmpty(): boolean
+ size(): int
+ toArray(): Object[]
+ getQueueCapacity(): int
+ getRemainingQueueSpace(): int
+ isQueueFull(): boolean
+ contains(arg0: Object): boolean
«unsuported»
+ toArray(arg0: T[]): T[]
+ iterator(): Iterator<E>
+ remove(arg0: Object): boolean
+ removeAll(arg0: Collection<?>): boolean
+ retainAll(arg0: Collection<?>): boolean
+ containsAll(arg0: Collection<?>): boolean
+ addAll(arg0: Collection<? extends E>): boolean

Figure 1 UML Class diagram for the project. (Note: Methods stereotyped as unsupported should be left as unimplemented and should throw an appropriate exception if invoked.