Lab 3: Queue ADT

Topic(s)

Queue Implementations

Readings & Review

- Carrano, *Data Structures and Abstractions with Java, 5th edition*, Chapters 7-8
- Queue ADT & implementation lectures

Objectives

Be able to:

- ✓ Modify the book's Queue ADT (Abstract Data Type) implementation using a circular array with a single unused element to instead use all array elements and a counter to track the current number of entries in the queue
- ✓ Grow the array then populate it using `System.arraycopy()`
- ✓ Answer questions about the Queue ADT and your implementation

Instructions

1. **This is an individual assignment**.
2. ***Read the entire assignment and ask questions about anything that you don't understand (before you start).***
3. To use the provided Eclipse project, download the zip file then carefully and fully follow all instructions in "…Import – Export – Submission Instructions for Labs – DMR…"

   Note: I will not grade submissions which are non-compliant.
4. For your application:
   a. Be sure to follow Good Programming Practices.

      Your code must comply with the Coding Standards/Guidelines posted on Brightspace.
   b. Keep in mind the Plagiarism & Cheating – Policy and Acknowledgment.
5. Coding:
   a. Modify the provided ArrayQueue.java
6. Do a Write-up (Analysis / Summary). The write-up is in a separate assignment in Brightspace.

7. Submit your work by the posted due date/time.
    a. Late submissions will incur a 25% penalty per day.
    b. Projects that don't compile will not be graded.
    c. No submission will be accepted after I make the solution available (typically the next class).
8. Be sure to follow Good Programming Practices.
   Your code must comply with my Coding Standards/Guidelines as discussed in class, used in my provided code, and supported by my Eclipse configuration settings.
9. If you'd like my assistance with your code, please (1) come to an extra help session or (2) contact me to arrange for extra help (limited availability).
10. Post all other questions with respect to this lab in the Labs discussion forum in Brightspace – ***I will not answer questions clarifying this lab or the material it covers by email.***

Note: Do not post or otherwise share this assignment or any code provided to you or implemented by you as part of this assignment. Violation of this restriction will be treated as a violation of the Wentworth Academic Honesty Policy; consequences are the same as those for plagiarism.

Reminder: you are bound by the Plagiarism & Cheating – Policy & Acknowledgment.

If you have any questions, ask!

Have fun! Dave

Problem

ADT Implementation:

Modify the provided Queue ADT implementation (ArrayQueue.java) – use the `TODO` comments to guide you as you make changes (replace TODO with DONE as you complete each task):

- ☐ update the class Javadoc comment:
    - ☐ replace "Your Name" with your full name
- ☐ Modify the provided ADT Queue (uses a circular array to contain its entries) to (**do these one at a time in the listed order**) – run the test program in `ArrayQueue.java` to see if you're getting the expected state after each modification:
    - ☐ enhance `checkCapacity()` to screen out `desiredCapacity`s that are too small (less than `DEFAULT_CAPACITY`) – throw an exception similar to the one when `desiredCapacity` is too large (use the same message replacing "large" with "small")

- ☐ implement `initializeState()` to do the work the 1-arg constructor is doing – then modify the 1-arg constructor to call `initializeState()`
- ☐ modify `clear()`: call `initializeState()` – use the 'best' desiredCapacity available (you will need to describe your reasoning in the write-up)
- ☐ modify all appropriate methods to initialize, increment, decrement, and inspect/use `this.numberOfEntries`
- ☐ modify all necessary methods to switch from the book's implementation which reserves an array element to distinguish between empty and full states to an implementation which uses all elements of the array and uses `this.numberOfEntries` to determine empty and full
  - ☐ modify `isEmpty()`: use `this.numberOfEntries` instead of the other state information – do not use any conditional logic
  - ☐ modify `isArrayFull()`: replace the current test with an appropriate boolean expression using `this.numberOfEntries` – do not use any conditional logic
- ☐ in `ensureCapacity()`: replace the loop with two invocations of `System.arraycopy()` – see below: "Growing a Circular Array using System.arraycopy()" for a full walk-through
- ☐ you may use `main()` to test/debug your code (it has access to the instance state/variables) – I provided an example program that displays the state of the queue as it `enqueue()`s and `dequeue()`s entries so you can see how the array is populated and expanded – you may modify/replace it as you wish
- ☐ do not make any other changes

My Tests

The provided version of `ArrayQueueDMRTests.java` (in …`dmr.tests`) is extensive – I may provide an updated version of the tests – I may use an alternate version to grade your code. I structured the tests to minimize the number of methods you need to have working to pass each test though the composite tests, by their nature, require many methods to work correctly. If your code passes all my tests, you will see something similar to:

```
Summary Test Results

⋮

Successfully completed ## of ## tests (100%) attempted for class ArrayQueue
```

To pass most of my tests, you will need to have most of the methods switched from the book's implementation to the revised implementation. Use the `main()` in `ArrayQueue.java`. Your test/debugging code must compile cleanly but will not be graded. My tests include some information (in the detailed logs) about the steps the tests take but not enough alone to isolate your bugs – it should help inform your choices for how/what to debug. If you're still stuck, post the failing portion of the detailed log in the lab discussion forum on Brightspace (not your code) and I can give you more details about what the test does. Note that my tests inspect the internal state of the `ArrayQueue` object (e.g., the size of the `queue` array) which is why most of these tests fail with the book's implementation even though the implementation works fully.

Do not modify `ArrayQueueDMRTests.java`. These are my tests – you must pass 100% to potentially get full credit on the lab. The tests generate two sets of output: summary information about each test group to the console and detailed information about every test to a log file (in the `test-logs` folder). You may need to select the `test-logs` folder in the Package Explorer or Project Explorer pane then press `F5` (or right-click on the folder then left-click on `Refresh`) so the list of logs gets updated. A new log is created each time you run the tests – the last log will be the most recent – they're named with the date and time in a format that will sort chronologically. You can/should delete old log files – I don't need/use them.

Note: Each test will time-out after a few seconds – if the tests seem to have stalled, please wait, they'll finish. The detail logs contain information about each test including the method under test, the parameter(s) provided to the method, the expected results, and the actual results. If the only tests that succeed in a section match the 'stub' value (an expected value which may occur even if the code is incorrect/incomplete), that test section indicates failing all tests. Because my tests have a built-in time-out, it's difficult to debug while they run. Instead, look in the detailed log for the parameters passed to a method in a failing test and pass the same parameters to that method from `main()` in `ArrayQueue` – you can then use Eclipse's debugging facilities without worrying about the test aborting because it took too long. I do not evaluate any debugging code you may have in `main()`, except that it must compile cleanly – no warning or errors.

Note: Do not submit your project with debugging code in any of the API or private utility methods (not even commented out) – it's ok to leave debugging code in `main()` as long as it doesn't cause compilation warnings/errors.

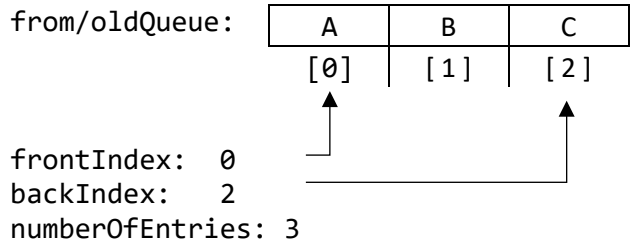
Growing a Circular Array using System.arraycopy()

In the context of the Queue ADT lab, how do we implement the copy portion of the `ensureCapacity()` method in a circular array using two invocations of `System.arraycopy()`?

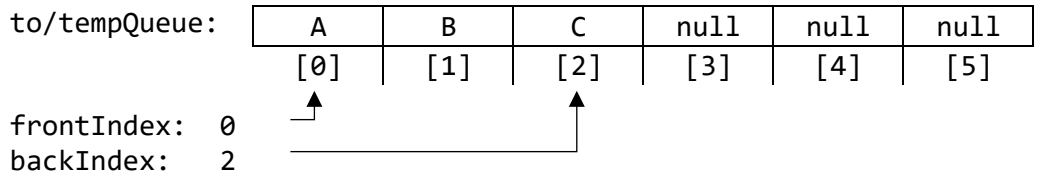
There are two scenarios to consider.

In the first scenario, the front and back indices either haven't wrapped (multiple `enqueue()`s with no `dequeue()`s) or wrapped fully (enough `enqueue()`s and `dequeue()`s to move `frontIndex` to `0` and `backIndex` to the highest index defined for the array). For the following discussion, all entries in this scenario are in the front portion; there are no entries in the back portion.

The following depicts the starting state for a queue array of length 3 (`oldQueue`, `frontIndex`, `backIndex`, `numberOfEntries`):



and the ending state (`tempQueue`, `frontIndex`, `backIndex`):



Observe that we'll be copying:

- array elements from `oldQueue` starting at index zero (0) to array elements in `tempQueue` starting at index zero (0)
- all elements in `oldQueue` which is the length of the `oldQueue` array (stored in `oldSize` for convenience)

Here, the front portion of the queue is all elements in the array (`oldQueue`), starting with index `[0]`; the back is empty (contains no elements).

We want the front entry of the queue to start at the beginning of the new array (`tempQueue`), so we'll always copy into `tempQueue` starting at index `[0]`.

So, our first take can copy the front portion of the queue by setting both the 'source position' (2nd parameter to `System.arraycopy()`) and the 'destination position' (4th parameter) to zero (0); and the length (number of entries to copy) (5th parameter) is all entries (which is in `this.numberOfEntries`).

Since all entries in the queue are in the front portion, the length of the back portion is zero (0) - the 'source position' and 'destination position' aren't important but they must be valid indices into `oldQueue` and `tempQueue`, respectively, otherwise, `System.arraycopy()` will throw an `Exception`. `System.arraycopy()` correctly copies no elements if the length (5th parameter) is zero (`0`).

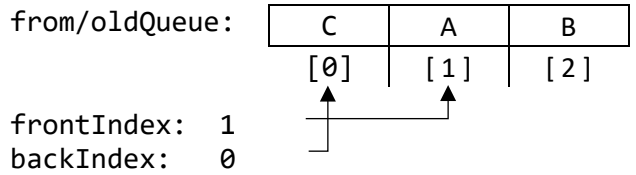
Our solution so far could be:

```
System.arraycopy( oldQueue, 0,          // from
                  tempQueue, 0,         // to
                  this.numberOfEntries ) ;  // how many elements
System.arraycopy( oldQueue, 0,          // from
                  tempQueue, 0,         // to
                  0 ) ;                 // how many elements
```

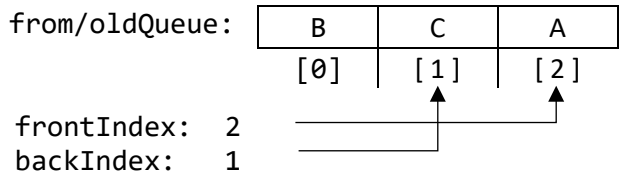
We'll see that for most scenarios, hardcoding the parameters to zero (`0`) isn't correct and `numberOfEntries` alone doesn't give us the correct number of entries to copy for the front portion of the queue.

There are two examples of the second scenario, given an array of three elements (`oldSize` and `numberOfEntries` are the same as in the first scenario). In this scenario, the front portion of the queue contains at least one entry but doesn't start at index 0 in `oldQueue` and the back portion of the queue contains at least one entry and starts at index 0 in `oldQueue`.

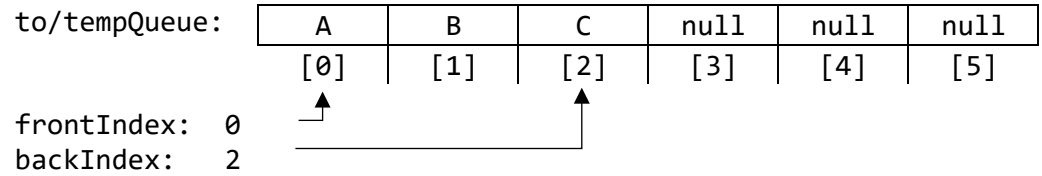
Example 1 has the front portion of the queue in elements 1-2 and the back portion in element 0:



Example 2 has the front portion of the queue in element 2 and the back portion in elements 0-1:



The ending state will always be the same as in the first scenario above:



Observe that we'll be copying the front portion of the queue:

- from `oldQueue` starting at index `frontIndex` to array elements in `tempQueue` starting at index zero (0)
- in the first example, the number of elements in the front of the queue is 2 (indices 1 and 2); in the second example, it's 1 (index 2)

The back portion of the queue is:

- in `oldQueue` starting at index `0` and in `tempQueue` starting at the entry immediately following the last entry in the front portion of the queue
- the number of elements in the back portion is the number of elements not in the front portion which we can calculate by subtracting the number of elements in the front portion from the total number of elements in the queue

So, our revised solution copies the front portion: takes some entries from the 'source position' (2nd parameter to `System.arraycopy()`) and the 'destination position' (4th parameter) is still zero (0); and the length (number of entries to copy) (5th parameter) which is greater than or equal to 1.

We always copy the back portion of the queue starting from 'source position' index zero (`0`) in `oldQueue`. The 'destination position' (index into `tempArray`) is not fixed at zero (`0`) so we must instead calculate it so the entry(ies) from the back portion of the queue are copied into the array elements in `tempQueue` immediately following the index holding the last element of the front portion of the queue. Unlike the first scenario, the length of the back is non-zero.

Our revised solution looks like:

```
System.arraycopy( oldQueue, ?,          // from
                  tempQueue, 0,         // to
                  ? ) ;                 // how many elements
System.arraycopy( oldQueue, 0,          // from
                  tempQueue, ?,         // to
                  ? ) ;                 // how many elements
```

The Provided Eclipse Project

The assignment includes a zip file of an Eclipse project containing:

1. Queue ADT implementation (in the …`queue.adt` package):

   - ArrayQueue.java
     - this is a working version based on the book's implementation

     ☐ make the modifications described above

     ☐ change the TODO comments to DONE comments as you complete each task

2. shared (in the …`queue` package):

   - QueueInterface.java and EmptyQueueException.java
     - These are basically the same as in the textbook
     - used throughout the project
     - do not modify these files
   - ArrayQueueCapacity.java
     - an enumeration (enum) of capacities
     - used by ArrayQueue.java to set `DEFAULT_CAPACITY` and `MAX_CAPACITY`
     - used by ArrayQueueDMRTests.java to exercise ArrayQueue.java
     - do not modify this file

3. …`dmr.tests` package:

    - ArrayQueueDMRTests.java
        - my provided JUnit tests for ArrayQueue
        - for full credit, your implementation must pass all tests
        - do not modify this file
        - to debug your code, use `main()` in `ArrayQueue.java` – my tests in `ArrayQueueDMRTests` timeout after a couple of seconds
    - Folder `test-data-dmr`
        - data for ArrayQueueDMRTests.java
        - do not modify the contents of this folder or store anything in it

4. …`dmr.testing` package:

    - various classes
        - support my testing
        - do not modify any of these classes

5. `doc` folder:

    - contains the Javadoc documentation for the project (excluding testing)
    - you can access the documentation by double-clicking the index.html file
    - you do not need to regenerate the documentation even if you add methods to ArrayQueue.java (though you do need to include Javadoc comment blocks for them 8~)

6. `test-logs` folder:

    - ArrayQueueDMRTests.java generates a detailed log file each time it executes
    - detailed log files are named with the date/time of execution – the last one is the most recent

        ☐ you should delete unnecessary (old) logs while you test to reduce clutter

        ☐ you should delete most/all logs before submitting (it's ok to leave the last/most recent log)

    - you may need to click on the `test-logs` folder name in the Package Explorer pane then press `F5` or use the `File > Refresh` menu option to tell Eclipse to check the folder for updates

---

Good Programming Practices/Requirements

☐ **Your code must compile successfully, or I won't grade it**

☐ Your project must have the correct name

☐ Your class must have the correct name and be in the specified package (already correct – don't change them)

☐ Use mnemonic/fully self-descriptive names for all classes and class members (methods, variables, parameters, etc.) – make sure they all abide by Java de facto standard naming conventions (methods, variables, etc. are lowerCamelCase; classes, interfaces, enum[eration]s are UpperCamelCase; and constants are UPPERCASE_WITH_UNDERSCORES.

☐ You must include a Javadoc comment block for the class (already provided)

  ☐ position the cursor on the blank line immediately before the `public class Xxxx` declaration; type `/**` then press `Enter`. Make sure your actual name (not your username) is in the `@author` tag.

☐ You must include full Javadoc comments for all classes and methods (even `private` methods) in all classes (included for all provided methods)

  ☐ hint: position the cursor on the blank line immediately before the method header line; type `/**` then hit `Enter`.

☐ (optional) Generate the Javadoc for the …`queue.adt` package (uncheck the others) for `private` visible items (it must be in the `doc` folder). Make sure your Javadoc processes without errors (it's ok for the Javadoc utility to complain that there are no comments for the `ArrayQueue`'s data fields).

I provided Javadoc (or its equivalent) for `ArrayQueue.java` and the supporting classes in the …`queue` package – the Javadoc for the provided code is already in the `doc` folder. FYI, the following is considered a Javadoc comment block – it pulls the documentation from the linked location:

```
/*
 * (non-Javadoc)
 * @see edu.wit.scds.dcsn.comp2000.queue.QueueInterface#clear()
 */
```

Do not replace these comments.

☐ Add brief comments to your code, not just so it's easier for other readers, but also so it's easier for you to remember your logic. Use block comments (enclosed in `/* … */`) and/or line comments (starting with `//`). Comments should clarify or set expectations, not simply duplicate the code in English.

☐ Your code must abide by my coding standard. Once you finish coding, tell Eclipse to make it compliant: `Source > Clean Up…` (you need to have already imported my settings)

☐ Make your instance variables `private` (already done)

  ☐ Do not add any instance variables

☐ You must initialize all instance variables in your constructor(s) or a method invoked by the constructor(s) – you are not permitted to use default instance variable values (their zero value) – the no-arg constructor should set up a valid, empty instance - invoke the no-arg constructor as the first executable statement in any other constructors to give you a known, reliable starting state – do not duplicate the code.

☐ Do not submit your project with debugging code in any of the API methods (not even commented out).

☐ Spell check and grammar check your work! Misspellings and grammatical errors (in write-ups and comments in your code) send a strong message to the reader/reviewer that you do sloppy work – they (I) will (perhaps unfairly) assume your code is equally sloppy (and therefore buggy).

Write-up

I will provide a write-up template in a separate assignment in Brightspace.

Submitting Your Work:

☐ Make sure your name is in all project files you create or modify (e.g., `ArrayQueue.java`). Do not add this information to supplied files that you do not modify.

☐ Make a **.zip file** for your project (my grading procedures expect this). ***No other compressed formats are acceptable!*** Follow the "Import – Export – Submission" instructions carefully to minimize errors.

- ☐ Include your entire project folder (root folder and subfolders, not just the contents of the root, src or bin folders).
- ☐ In Brightspace, **attach** your zipped solution file to the submission for this assignment.

Rubric

This lab is worth 100 points:

- Queue ADT: `ArrayQueue.java`
  - 100 points

☐ I will not grade your lab if it doesn't compile - I will tentatively give you a grade of 1 for the entire lab - fix the problems and resubmit - if it still doesn't compile, I will give you a 0. This is all-or-none - your classes must compile successfully.

☐ Similarly, if you don't rename the folder and project with your username (e.g., RosenbergD) or if you don't replace 'Your Name' with your name (e.g., Dave Rosenberg) in the `@author` tags in the Javadoc comment blocks immediately preceding each class, you will receive a grade of 1 and I will not grade your submission. If you resubmit and still haven't corrected the issues, I will give you a grade of 0. This is also all-or-none – you must properly complete all steps.

☐ You must change each `// TODO` comment to `// DONE` – I will only grade methods marked as `DONE`. **Note that your code must follow the** `// DONE` **comment.**

I calculate the grade for each part of the assignment (code) as:

- 80%: works
    - for `ArrayQueue` – based on the %-age correct when you run `ArrayQueueDMRTests` - may be adjusted down for problematic implementations
- 20%: clean code:

☐ your code resembles mine (indentation and spacing) - my code uses space characters instead of tab characters and indents by 4 spaces at a time - Eclipse will do this (follow the instruction in "Installing and configuring the Eclipse IDE" in the Resources section on Brightspace to get my configuration setting – you can then tell Eclipse to format the code (`Source -> Clean Up…`). You can only clean up your code if it compiles cleanly (no red blocks in the right margin).

☐ variable names are mnemonic - self-descriptive - and spelled out ('`i`' is acceptable as a loop variable in the `for` loops)

☐ names use Java de facto standard capitalization (`aVariable` – not `a_variable` – not `AVariable`)

☐ you enclosed all statements controlled by `if`/`while`/`for` in curly braces, even if only a single statement

☐ curly braces must use next-line, indented format (consistent with the code I provided)

☐ proper use of Java language (e.g., don't compare a `boolean` expression/variable against `true` or `false`)

☐ you added brief comments describing your logic (typically not useful line-by-line) and focusing attention on important information (e.g., '`loop starts at index 1 because the 0-th element was already processed`')

☐ you spelled comments and variable names correctly – use proper grammar

- Eclipse automatically spell-checks your comments – its dictionary isn't very extensive but it's a good start – for correctly spelled words that aren't in its dictionary, you can add them (hover over the word until a list of alternatives displays then scroll to the bottom and click on 'add to dictionary')

- Eclipse doesn't spell-check your code (variable names, method names, etc.) – pay attention to your spelling
- misspelled words and poor grammar are often viewed by a reader of your code as an indication of whether your code is likely to be correct – simple conclusion (which may be wrong) is sloppy comments means sloppy code – result: look for a job elsewhere

☐ you only produce specified output - remove (don't just comment out) all debugging statements except in `main()`.

I will not grade any code in `main()` in `ArrayQueue.java` – you may/should use this to (optionally) debug your code. However, any code in `main()` mustn't cause any compilation warnings or errors.

I typically include comments/feedback with your grade (in the Grades section on Brightspace).

Submission

☐ You must export your project from Eclipse into a zip archive - you will upload this zip file to Brightspace. Refer to the "…Import – Export – Submission…" instructions for details. I recommend that after you export the project, using Windows Explorer, open the zip file and make sure everything's where it's supposed to be within it: there should be a single item – the project folder – at the top level; several subfolders (e.g. admin, bin, doc, src, etc.) and .project and classpath settings files in the project folder, and the rest of the folder hierarchy will match the structure of the project in Eclipse. If you're not sure how to do this, ask.

You may resubmit your solution via Brightspace – it's set to accept an unlimited number of submissions. I only accept submissions via Brightspace – do not email them to me. To access the submission form, click on the assignment title.

If your submission is not in the correct format or if pieces are missing, I'll give you a grade of 1 on Brightspace and provide a brief explanation of what's wrong. You will need to resubmit, or I will change the grade to 0.

Late Submissions

☐ Make sure you know by when the assignment is due. Click on the assignment title in Brightspace to verify the due date/time – you can also see how many points the assignment is worth.

I accept late submissions until I post the solution set – typically 2-3 days after the due date – but not after I post the solution.

Late Penalties

There is a 25% per day deduction for late submissions starting as soon as the due date/time has passed. If you submit any time after that – even on the due date but after the due time – you will lose 25%.

To avoid late penalties, submit well in advance of the deadline – 3-6 hours earlier is probably safe. Your first submission must be substantially complete, but I will not grade the initial submission if you resubmit – in this case, I use the first submission to determine on-time/late but I grade the last submission. *If your initial submission contains my unmodified starter code, I may refuse to grade the lab once you resubmit. If your initial submission isn't substantially complete, it may not stop the clock – instead, I'll look at the date/time of the first substantial submission to determine on-time/late.*

Note that network/power/other utility problems, Brightspace access/availability issues, travel delays, "the dog ate my computer" occurrences, and the like may happen – plan accordingly – if they happen at or shortly before the submission deadline, that's unfortunate but irrelevant. Submit several hours before the deadline to make sure you don't have problems then resubmit your completed solution later (even after the deadline). *Do not delete anything* related to the project from your computer until after I post your grade in case there are issues with your submission.

Extenuating Circumstances

If you have a medical issue, family emergency, or other legitimate reason for not getting the assignment in on-time – or within the allotted time – contact me as far in advance, or as soon after an emergency as possible to discuss the options I can offer you. I will do my best, but I cannot guarantee that I'll be able to provide extensions for every assignment.