Programming assignment #3: CSE 12 Spring 2013

Here's a description of the third programming assignment for CSE 12, Spring 2013. In this assignment, you will write generic circular array, stack, and queue implementations. Along the way you will get more experience with implementing Java interfaces, writing JUnit test cases, and using the Adapter design pattern.

>>>Due Fri May 10 5:00 PM

>>> Required solution files: BoundedDequeTester.java Deque12.java Stack12.java Queue12.java

This means that the files that contain your solution to the assignment must be turned in by the due deadline by using the bundleP3 script while logged into your cs12 account on ieng6. Early turnin points are as for previous assignments. The turnin procedure is described below in the section "Turning in your assignment".

It's a good idea to plan to do the assignment well before the deadlines; terminals and tutors can get very busy at the last minute.

In doing this assignment, it's your responsibility to understand the course rules for integrity of scholarship.

Getting started

First, read chapters 6 and 7 the textbook, and corresponding lecture notes; also, you may find it useful to read the Wikipedia entry on double-ended queues. This assignment uses material introduced and discussed there. Create a directory named "P3" under your cs12 home directory to use for developing your solution to this assignment. Your solution to this assignment should be turned in from that directory by the due date and time. You will use the bundleP3 script to do that; see below.

Copy the files `BoundedDeque.java`, `BoundedStack.java`, and `BoundedQueue.java` from the `~/../public/P3/` directory to your P3 directory. These files define interfaces that you will be implementing in this assignment.

BoundedDequeTester.java

Read the documentation in the source code file for the `BoundedDeque<E>` interface, or view the javadoc generated web pages for BoundedDeque. Understand the responsibilities of each method required by the interface. Sketch a test plan for a class that implements this interface. Define a class named `BoundedDequeTester` that extends `junit.framework.TestCase` and that implements your test plan, using an `Deque12` object as a test fixture.

The class you will be testing is `Deque12`, which implements `BoundedDeque`, and which you will be defining as part of this assignment. As a practical matter, you do not need to completely write your `BoundedDequeTester` program before starting to define `Deque12`. In fact, an iterative test-driven development process can work well: write some tests in `BoundedDequeTester`, and implement the functionality in `Deque12` that will be tested by those tests, test that functionality with `BoundedDequeTester`, write more tests, etc. The end result will

(hopefully!) be the same: A good tester, and a good implementation of the class being tested.

As for previous (and future) assignments, make sure that your `BoundedDequeTester` does not depend on anything not specified by the documentation of the BoundedDeque interface and the Deque12 class. For example, do not make use of any other constructors or instance or static methods or variables that the ones required in the documentation.

Deque12.java

Define a generic class `Deque12<E>` that implements the `BoundedDeque<E>` interface. Besides the requirements for the methods and constructor documented in that interface, for this assignment there is the additional requirement that this implementation must use a circular array to hold its elements. (Note also that, as for other assignments, your Deque12 should not define any public methods or constructors other than those in the interface specification.)

A circular array can be rather tricky to implement correctly! It will be worth your time to sketch it out on paper before you start writing code.

You will of course need to create an array with length equal to the capacity of the Deque12 object. You will want to have instance variables for the size of the Deque12 (how many data elements it is currently holding), and to indicate which elements of the array are the current front and back elements. Think about what the values of these instance variables should be if the Deque12 is empty, or has one element, or is full (has size equal to its capacity). And in each case, think carefully about what needs to change for an element to be added or removed from the front or the back. Different solutions are possible, as long as all the design decisions you make are consistent with each other, and with the requirements of the interface you are implementing.

Stack12.java

Read and understand the interface specification in `BoundedStack.java` (or view the javadoc pages for BoundedStack) and then define a generic class `Stack12<E>` that implements the `BoundedStack<E>` interface. Once `Deque12` is implemented and tested and debugged, defining `Stack12` is quite easy. Note that the methods required by the `BoundedStack` interface are different from, though closely related to, the `BoundedDeque` interface methods. Use the Adapter pattern!

Queue12.java

Read and understand the interface specification in `BoundedQueue.java` (Or view the javadoc for BoundedQueue) and then define a generic class `Queue12<E>` that implements the `BoundedQueue<E>` interface. As with Stack12, if `Deque12` is already implemented and tested and debugged, defining `Queue12` is quite easy. Again note that the methods required by the `BoundedQueue` interface are different from, though closely related to, the `BoundedDeque` interface methods, and so the Adapter pattern is applicable here as well.

As is often the case when the Adapter pattern is used, if the adapted class (`Deque12` in this case) is tested and debugged, the adapting class shouldn't need much testing, because almost all of the work is being handled by delegation to the adapted class's methods. However, you should do some testing of your `Stack12` and `Queue12` classes, to make sure that everything is working correctly.

One partial test you can do is use your Queue12 and Stack12 classes to help solve a maze. Copy the file `MazeMaker12.jar` from the `public/P3` directory to your working directory, and run it as

```
java -jar MazeMaker12.jar
```

It will pop up a window showing a grid of cells. Click on "Create" to knock down some walls of the cells to make a maze. Now there are two ways to discover paths in this maze: Depth First Search (DFS) and Breadth First Search (BFS). As you will see, these two strategies explore the maze in very different ways. However, the only difference between the two algorithms is what data structure each uses to to keep track of cells in the maze that it has visited in its search (DFS uses a stack, BFS uses a queue). Click on "BFS" or "DFS", and select a starting point, to initiate a breadth-first or depth-first search of the maze. During the search, the blue dots are the maze cells that are stored on the stack or queue during each step in the search.

(If you would like to see how a correctly functioning stack and queue work in this context, copy `MazeMaker.jar` from that directory and run it.)

Turning in your assignment

After you are done with the assignment, you can turn it in. The process is essentially the same as for previous assignments; when logged into ieng6 or a lab workstation, cd to the directory containing your solution file and run the command

```
bundleP3
```

and carefully follow the dialog.

Grading

There are 15 possible points on this assignment, not including early-turnin points. We will test your solution as follows:

- We will copy your turned-in files into an otherwise empty directory.
- Then we will copy in the `BoundedDeque.java`, `BoundedStack.java`, and `BoundedQueue.java` files from the `public/P3` directory.
- We will then compile your solution with the commands:

  ```
  javac Deque12.java
  javac Queue12.java
  javac Stack12.java
  javac BoundedDequeTester.java
  ```

  (If your solution files do not compile, you will recieve 0 points for this assignment.)
- We will test your `Deque12`, `Queue12`, and `Stack12` classes to make sure that they implement correctly the requirements of this assignment.
- We will test your `BoundedDequeTester` program to make sure it thoroughly tests a `Deque12` class's implementation of `BoundedDeque`.

We will also consider your coding style, including commenting. Commenting and style guidelines are the same as

for previous assignments. Keep in mind that the javadoc comment on a public class should use the `@author` tag to indicate the name of the author of the code (that is, you).

Good luck!