Project 1b: Data Structures Part 2

This is the second part of project 1. Part 1C will be released by Monday evening. While projects 1B and 1C have been combined into a single deadline (2/11) the specification and autograder for 1C will be separate for logistical and conceptual simplicity.

This project is brand new. Please let us know on Piazza if you spot any bugs or issues.

Introduction

Introductory video: (Youtube)

In project 1b, you will build a rudimentary autograder for project 1a. In the skeleton, we have provided the following files:

- StudentArrayDeque.class: A buggy implementation of ArrayDeque.
- StudentLinkedListDeque.class: A buggy implementation of LinkedListDeque.
- ArrayDequeSolution.class: A correct implementation of ArrayDeque.
- LinkedListDequeSolution.class: A correct implementation of LinkedList.
- FailureSequence.java: A utility class for this assignment.
- DequeOperation.java: A utility class for this assignment.

- examples/: A folder with some examples that may be helpful.
- proj1b.jar: For use with IntelliJ. Contains all of the listed .class files above.

Your goal for part 1B is to create at least two Java files: `TestArrayDeque1B.java` and `TestLinkedListDeque1B.java`. There is no specific API (that is, no specific methods that you must implement).

For this project, you are allowed to work with a partner, and your partner must be the same as you had for part a. If you wish to dissolve your partnership from part a, please send an email to the course staff for approval.

Getting the Skeleton Files

As with project 1, pull the skeleton using the command `git pull skeleton master`.

If you're using IntelliJ make sure to reimport the project. After you've done this, right click (or two finger click) on the proj1b jar file and select "Add as Library".

Phase 1: Finding the Bugs

We've provided buggy implementations of the project 1a assignment, namely `StudentArrayDeque.class` and `StudentLinkedListDeque.class`. Note that we've provided the class files only. Despite not having access to the .java files for these buggy implementation, your code that uses these classes will still compile and run just fine.

You should start by making sure you can write code that uses the `StudentArrayDeque` and `StudentLinkedListDeque` classes. An example is provided in the `examples/StudentArrayDequeLauncher.java` file.

In `TestArrayDeque1B.java` , you should give a JUnit test that `StudentArrayDeque` fails. Likewise in `TestLinkedListDeque1B.java` , you should give a JUnit test that `StudentLinkedListDeque` fails. Any tests you write should succeed for a correct implementation, e.g. you can't just do `assertEquals(5, 10)` and consider youself done with project 1B.

In this first phase of this project, you should simply write JUnit tests which are capable of consistently finding at least one bug in each of these two classes. For example, when you run `TestArrayDeque1B` , some assertEquals statement should get triggered into displaying an error.

Important: When building tests, you must select Integer as your test type, `StudentArrayDeque<Integer>` *.*

You are not required to write randomized tests (like those from the project 1A autograder), but this is probably the easiest and laziest way to succeed.

Once you feel reasonably comfortable that your JUnit tests are capable of identifying failures in a buggy implementation, it's time to improve your autograder.

Phase 2: Printing Out a

Of course, simply telling the student that their code fails is only going to lead to tears, sadness, confusion and late night Piazza posts. Thus, you're going to modify your autograder sot that it tells the student something useful.

To do this, we'll take advantage of the `assertEquals(message, expected, actual)` method, which outputs a helpful message to the user.

For an example of how this method works, see `AssertEqualsStringDemo.java` in the examples folder.

The string message provided to assertEquals must be a series of method calls, where the last call in the sequence yields an incorrect return value. For example, if adding 5 to the front, then 3 to the front, then removing from the front yields an incorrect value, then the String message passed to assertEquals should be exactly the following:

```
addFirst(5)
addFirst(3)
removeFirst()
```

You do not need to supply the expected and actual values as part of the String message, since those are passed separately to the assertEquals statement.

You should not attempt to idenftify weird corner cases that cause the Student input to crash. See the FAQ for more.

To make your life easier, we've provided optional helper classes `DequeOperation.java` and `FailureSequence.java`. You are not required to use these files, and you're free to modify them however you wish. However, if you DO use them, you should make sure to submit them to gradescope. For examples of how to use these classes, see `FailureSequenceDemo.java`. If you have previous Java experience, we recommend not using these files, then later comparing your solution to how we did things using these files.

Submission

Submit `TestArrayDeque1B.java`, `TestLinkedListDeque1B.java`, and any supporting files you created or require, including `DequeOperation.java` and `FailureSequence.java`. Do not submit .class files.

Tips

- Start with ArrayDeque. A bug will be easier to find.
- assertEquals will not work the way you'd hope with Deques. For example assertEquals(deque1, deque2) will not return true if all the items are the same. You'll need to write your own comparison method if you want to compare entire deques.
- *It's probably not a good idea to write a Deque comparison function*. Suppose you write a `compareDeques(studentDeque, solutionDeque)` method that returns false. Even if this function returns false, that doesn't give you an operation that causes a failure. It's much easier to test the output of single operations (e.g. student.removeFirst() vs. solution.removeFirst()).
- The StdRandom class is the easiest way to generate random numbers. See the official documentation for a list of methods.
- There's no need to do any exception catching or throwing on this assignment (we haven't learned this in 61B yet).
- Build your failure sequence as you perform operations.

I found a bug in StudentArrayDeque! If I call get(5) when there are only 3 items it crashes. However, I can't figure out how to get assertEquals to report the message in a nice way.

You're required to find a bug for which the student solution returns the wrong answer, not one for which it crashes. I know this is a little artificial, but I felt it was a more interesting autograder component for you to write.

I keep getting a null pointer exception in my TestLinkedListDeque1B and I'm not using get()!

Look very carefully at your error message. You'll notice the error is not happening in the student code, but in your code! The issue is probably that you're doing something like `int x = slld.removeFirst()`. If removeFirst returns a null, then this causes a null pointer exception. Use `Integer x = slld.removeFirst()` instead. We'll discuss the difference between an `Integer` and an `int` after the midterm, but basically `Integer` object can hold null, but ints cannot. Autograders are hard.

You know what, I found a bug in ArrayDequeSolution! The very same one that is in StudentArrayDeque. get(5) crashes on an empty ArrayDequeSolution instead of returning null.

Well, that's embarassing. Rather than fix this, we'll leave this bug in to encourage people to come to this FAQ and see the first question.

How would I write a test for printDeque()?

It would be rather involved, and our autograder autograder isn't quite smart enough to be able to read your output anyway. Stick with the other methods. If you're curious, google "redirect standard output"

I'm getting a "reference to assertEquals is ambiguous" error.

Always try searching the web for mysterious error messages. Recall that self-sufficiency as a programmer is a major goal of 61B. I *think* the first hit on Google should be enough, but certainly post to Piazza if you're still stuck.

The autograder is complaining about my failure sequences.

As you might imagine, the autograder for project 1B is a weirdly complex beast, as it is has to autograde autograder output. To keep things simple *the String argument to a failing assert must contain a failure sequence and ONLY a failure sequence*, and *all tests must fail due to a failing assert*. There should be no failures due to null pointer exceptions. The String argument to your assert statement must contain no extraneous information.

The autograder is still complaining about my failure sequences.

You need to include the operation that caused the failure. For example, if `size()` returns the wrong value, you need to include `size()` in your failure sequence, since you're required to provide "a series of method calls, where the last call in the sequence yields an incorrect return value". Also make sure your failure sequence only appears once!

I tried all that and the autograder is still complaining about my failure sequences.

Copy the reported failure sequence from the online autograder, and write a simply file `Quick.java` which generates a studentDeque, applies the operations listed, and prints the result of the final step. Chances are you'll find that the result is not the same as your test reported in the AG. Most likely you forgot to include an operation, or possibly added operations you didn't actually make.

Project 1b Navigation