Stacks and Queues

CS 151 - Introduction to Data Structures

Assignment 4 - due Friday 2/24

1 Tasks

Before we start, it is important to note that you are not allowed to change the given `Stack.java`, `IntStack.java`, `ArrayStack.java`, `Queue.java`, and `Deque.java`.

Part 1. Copy `ArrayStack.java` and `Stack.java` from `~dxu/handouts/cs151/code/5-1`. Copy `Queue.java` from `~dxu/handouts/cs151/code/5-2`. Write a class called `TwoStacksQueue` that implements the `Queue` interface as follows.

1. Your class will store two `ArrayStack` objects as instance variables but no other. A `TwoStacksQueue` object is a `Queue` and should behave as a `Queue` (FIFO). Since you are using two stacks to simulate a queue, it will certainly not be the most efficient implementation of a queue and that's ok - just as long as you know that and can analyze the runtime appropriately in the `README` - see below. There should not be any other array/ArrayList/linked list used within your implementation.
2. Your `README` should provide a discussion on the design of your data structure, in particular how you implemented `enqueue` and `dequeue` operations. In addition, you should provide a worse-case big-O analysis of each of these operations.

Part 2. Implement the `Deque` ADT (double-ended queue where we can insert and delete at both ends) with an array used in a circular fashion. Copy `Deque.java` from `~dxu/handouts/cs151/code/5-2`, which specifies the the `Deque` interface that you must implement. Name your class `ArrayDeque`.

Study how we implemented the `Queue` ADT using a circular array for reference. You should find the discussion in Section 6.3 of your textbook helpful as well.

Part 3. (Extra Credit) Implement a stack data structure (name it `NewStack`), storing integers, that supports the usual operations `size`, `isEmpty`, `push`, `pop`, `top` and an additional operation `minElement`, which returns (but does not remove) the smallest element currently in the stack. All operations (except for `toString`) should run in $O(1)$ worst case time - note that this means no loops of any kind. There are different designs to achieve the $O(1)$ `minElement`, differing mainly in the amount of additional space. Two main approaches are $O(1)$ space or $O(n)$ space. Explain how your data structure works in your README and justify the $O(1)$ `minElement`, together with the amount of space used. It is acceptable to write a non-generic `NewStack` that only stores integers and doesn't implement the `Stack` interface. In that case, implement the `IntStack` interface instead.

2 Additional Requirements

These requirements apply to the implementations of all three parts (or two parts, if you are not doing extra credit) above.

1. Provide a zero-parameter constructor which constructs an object (of the corresponding class) of a default size. Provide also a one-parameter constructor that constructs an object (of the corresponding class) of the parameter-specified size. Refer to the implemenation of the constructors of `ArrayStack` or `ArrayQueue` for details.

2. Follow your textbook's design and return `null` for any attempted operations on empty

3. Any attempt to insert into a full data structure, as well as any other unavoidable error should result in the Java run-time exception `IllegalStateException` being thrown.

4. Override `toString` to return a `String` that contains the contents of the data structure in the following format, starting from the first-in element as `element1`:
   `(elment1, element2, ..., elementn)`.
   Note that in case of a stack, this means `top` is printed last, not first.

3 Testing

A test program `CheckFormat_A4.java` has been provided for you. Note that this mostly makes sure that your output format is as expected for our autograder.

Although some correctness testing is included, it's minimal. You are expected to write your own tests. More specifically, once you pass `CheckFormat_A4.java`, write a driver program `Main.java` that tests all the methods you have implemented in your `TwoStacksQueue`, `ArrayDeque`, `NewStack` implementations in above parts. You should include enough tests to clearly demonstrate that your implementation works. We will test by replacing your `Main.java` with one of our own, calling your methods and using your classes. Make sure you test thoroughly!

4 Electronic Submissions

1. README: The usual plain text file `README`

- **Your name:**
- **How to compile:** Leave empty if it's just `javac Main.java`
- **How to run it:** Leave empty if it's just `java Main`
- **Known Bugs and Limitations:** List any known bugs, deficiencies, or limitations with respect to the project specifications. Documented bugs will receive less deduction versus uncaught ones.
- **Write-up:** Contents as discussed above for Part 1 and 3

2. Source files: all `.java` files

3. Data files used: none

DO NOT INCLUDE: Please delete all executable bytecode (`.class`) files prior to submission.

To submit, store everything (README and source files) in a directory called `A4`. Then follow the directions here:
`https://cs.brynmawr.edu/systems/submit_assignments.html`