Stacks and Queues

CS 206 - Introduction to Data Structures

Assignment 4 - due Tuesday 3/5

1 Tasks

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

Part 1. Copy `ArrayStack.java` and `Stack.java` from `~dxu/handouts/cs206/code/lec09`. Copy `Queue.java` from `~dxu/handouts/cs206/code/lec10`. Write a class called `TwoStacksQueue` that implements the `Queue` interface as follows. 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. Override `toString` for `TwoStacksQueue` to return a `String` that contains the contents of the current Queue in the following format
`(elment1, element2, ..., elementn)`.

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/cs206/code/lec10`, which specifies the the `Deque` interface that you must implement. Name your class `ArrayDeque`. Override `toString` for `ArrayDeque` to return a `String` that contains the contents of the current Deque in the following format
`(elment1, element2, ..., elementn)`.

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. Implement a new stack data structure (call it `NewStack`), storing integers, that supports the operations `push`, `pop` and an additional operation `minElement`, which returns the smallest element currently in the stack. All operations should run in $O(1)$ worst case time - note that this means no loops of any kind. Explain how your data structure works in your README and justify the $O(1)$. It is acceptable to write a non-generic `NewStack` that only stores integers and doesn't implement the `Stack` interface. Override `toString` for `NewStack` to return a `String` that contains the contents of the current stack in the following format
`(elment1, element2, ..., elementn)`.

Part 4. 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.

2 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://systems.cs.brynmawr.edu/Submit_assignments`