PA6 Stacks and Queues

Due: Wednesday 3/17 by 11:30PM
Submission:

- PA6.pdf - answers to the written questions.

- ArrayStack.java - implementation of a stack backed by an array.

- ListStack.java - implementation of a stack backed by a linked list.

- ArrayQueue.java - implementation of a queue backed by an array.

- ListQueue.java - implementation of a queue backed by a linked list.

Overview

The purpose of this assignment is to give you practice implementing data structures, comparing different implementations of data structures, and determining the runtime complexity of certain methods of data structures.

The assignment consists of a written part answering questions and a coding part. It is suggested to finish the coding part first, since the written questions rely on the code that you have written.

Assignment

Code

Your job is to implement four different classes. Two of the classes represent the Stack ADT (abstract data type) and the other two represent the Queue ADT. As mentioned in class, many data structures are built on either arrays or linked structures. You will implement both types for both the Stack and the Queue and then compare the implementations. We will not specify all of the methods you must implement for these classes in this spec. Instead, we will provide interfaces with the starter code that you must implement. **In addition to the methods from the interfaces, you must also override the toString and equals methods. Lastly, you must implement copy constructors for all of your classes. Our autograder calls these methods. If you don't write these methods, then the autograder won't compile, which gives you a zero on the project.**

ArrayStack.java - implement the StackInterface with a Stack backed by an array.

ListStack.java - implement the StackInterface with a Stack backed by a linked list.

ArrayQueue.java - implement the QueueInterface with a Queue backed by an array.

ListQueue.java - implement the QueueInterface with a Queue backed by a linked list.

You are not allowed to use any other data structures other than an array for the two Array classes. For the linked list classes, you can not use the Java LinkedList class. Instead implement them like they were done in class, using a Node inner class and creating the linked list yourself.

Remember that in order to implement an interface all you have to type is:

```
public class ArrayStack implements StackInterface {
...
}
```

You will then get a compile error because your ArrayStack will have unimplemented methods from the interface. Eclipse should provide a 'quick fix' when you hover over the red line indicating the error. This quick fix can provide all of the method headers which can save some time typing. The quick fix is 'Add unimplemented methods.'

Remember that after you have implemented the interfaces you must also override the toString and equals methods, and write copy constructors for all of your classes.

The toString method should return a string of the format: "{0,1,2,3,4,5}". **Note that there are no spaces in the string.** The autograder counts on this format so be sure to follow it. If there are no elements at all, then print "{}". This also does not include a space. In the above example, '0' would be the bottom of the stack and '5' would be the top. If it were representing a queue, '0' would be the next element to be dequeued (the front of the queue) and '5' would be the element most recently enqueued (the back of the queue).

Two stacks are equal if their sizes and all of their elements are equal. Two queues are equal if their sizes and all of their elements are equal.

Clarifications to Common Questions

As (has been or will be) mentioned in class and the spec (in bold), you are not allowed to use any other data structures. That means you cannot use the Java LinkedList class or the Java ArrayList class or any other data structure. To put it simply, just implement the classes very similarly to how we did in lecture.

At the very top of this spec, we listed the files that you need to turn in. You should not turn in any other files. This means you can't create a Node class in a separate file. Instead, make the Node class as a nested class as shown in lecture. If you believe you need any other classes, these must also be nested/inner classes.

Written

A large part of this PA is the thought process behind implementing data structures and comparing different implementations. Answer the below questions and submit a PDF of your answers. We strongly recommend typing the answers. It makes creating a PDF much easier.

We will still allow handwritten answers but ensure that you have impeccable handwriting. If our UGTAs cannot understand any part of your handwriting they are instructed to mark the answer as incorrect. **If you submit it with a file type other than PDF you will receive a zero for the written portion of this assignment.**

1. ArrayList vs. LinkedList

We can finally answer the question, "Should I use an ArrayList or a LinkedList?"

In class, we went through an implementation of MyArrayList. This implementation was fairly consistent with the way the Java libraries implement an ArrayList. However, the MyLinkedList demo was **not** consistent with how the Java libraries implement the LinkedList class. It can be made much more efficient. If you are curious, old source code for the LinkedList class can be found here. The above link has some weird looking code called generics which we will get to soon enough.

1a

It would be useful to not just have an instance variable tracking the front of our list, but also have an instance variable tracking the back of our list. Why is this helpful? For which methods does it speed up the implementation?

1b

Moving forward, assume that the LinkedList class uses an instance variable to track both the front and the back of our list as mentioned in part A. Create a table that lists the runtime efficiencies (Big O) for both the ArrayList and LinkedList classes for the methods shown in the below table. You should make a table like this one and fill in the Big O where there are '?'. I have done a few for you.

Method	ArrayList	LinkedList
add(int value)	O(1)	?
add(int index, int value)	?	?
clear()	?	?
contains(int value)	?	?
get(int index)	?	?
isEmpty()	O(1)	O(1)
remove(int index)	?	?
toString()	?	?
equals(Object o)	?	?

1c

Using the above table, we can now make comparisons on when it might be a good idea to use an ArrayList or a LinkedList. Give a scenario where you should choose to use an ArrayList instead of a LinkedList. This can be a fully fledged real-world example or just an explanation of data access patterns, for example (completely made up sentence about Maps), "I would want to use a HashMap over a TreeMap whenever I am constantly adding and deleting elements from the map because..."

1d

Give an example or statement about when you would want to use a LinkedList over an ArrayList.

2. ArrayStack vs. ListStack

2a

Make a similar table as you did above. It should look like the below except the '?' should be replaced with the runtime efficiencies.

Method	ArrayStack	ListStack
push(int value)	?	?
pop()	?	?
peek()	?	?
isEmpty()	?	?
size()	?	?
clear()	?	?
toString()	?	?
equals(Object o)	?	?

2b

As you are aware, Java only provides one Stack class. So you cannot choose an ArrayStack or a ListStack depending on data or data access patterns. You have to pick one and go with it for all of your Stack needs. Using the above table and your knowledge about how Stacks are used, would you choose to implement a Stack with an array or a linked list as its backing data structure? Why?

3. ArrayQueue vs. ListQueue

3a

Make one more table! Hopefully you are getting pretty good at this by now.

Method	ArrayQueue	ListQueue
enqueue(int value)	?	?
dequeue()	?	?
peek()	?	?
isEmpty()	?	?
size()	?	?
clear()	?	?
toString()	?	?
equals(Object o)	?	?

3b

If you were creating a new programming language, and writing a Queue class, would you choose to use an array or a linked list as the backing data structure for your queue? Why?

Error Handling

We will use very bad error handling practices in this PA. We have to do this until we learn about exceptions. If the user calls an erroneous method you should ignore it as best as you can. Of course, some of the methods still require a return value. The 'error' return values are specified in the comments of the interfaces.

Grading Criteria

We are not providing testcases for this PA. As long as you implement the StackInterface for your Stack classes and the QueueInterface for the Queue classes, our autograder will be able to call those methods on your classes. That is how we will test your implementations. **But of course don't forget to write a copy constructor and override the equals and toString methods.** You should not print anything in any of your classes. i.e. do not call 'System.out.println' anywhere in your code. You can (and should) use print statements to debug but be sure to remove them all before submitting your files.

We encourage you to write your own JUnit testcases to ensure your classes work properly, but we will not be collecting or grading these test files. We will test your classes with our own testcases.

Your grade will consist of similar style and code clarity points we have looked for in earlier assignments.

Write your own code. We will be using a tool that finds overly similar code. Do not look at other students' code. Do not let other students look at your code or talk in detail about how to solve this programming project. Do not use online resources that have solved the same or similar problems. It is okay to look up, "How do I do X in Java", where X is indexing into an array, splitting a string, or something like that. It is **not** okay to look up, "How do I solve {a programming assignment from CSc210}" and copy someone else's hard work in coming up with a working algorithm.