Lab 05 - Stacks and Queues

Dr. Mark R. Floryan

August 20, 2018

1 PRE-LAB

This week, you will be writing two data structures. The first will be a **linked list based** queue, and the other will be an **array based** stack.

1. Download the starter code and import the project into Eclipse
2. Implement the Queue.java class
3. Implement the Stack.java class
4. Verify your implementation using the provided tester class
5. **FILES TO DOWNLOAD:** stacksandqueues.zip
6. **FILES TO SUBMIT:** lab05.zip

1.1 QUEUE.JAVA

To begin, implement the *Queue* class inside the Queue.java file. The methods you are responsible for are listed below. This Queue **must be a linked-list based queue**. You may use

Java's built-in Linked List (import java.util.LinkedList) or you may use your own implementation from the previous labs.

```
public class Queue<T>{
    public Queue();

    public int size();

    public void enqueue(T data);

    public T dequeue();
}
```

1.2 STACK.JAVA

Next, you will implement a basic stack in java. This stack **must be an array-based stack**. Thus, your stack must support the resizing of the capacity of the underlying array. You **may not use an instance of Vector** in order to accomplish this. Instead, your class must have a field that is the underlying array. We are doing this so that you are implementing both a stack, and also learning how vector is implemented at the same time.

The methods you are responsible for are shown below:

```
public class Stack<T>{
    // The fields of this stack are given to you as follows
    private Object[] data;
    private int size = 0;
    private static final int INITIAL_CAPACITY = 100;

    /* Constructors not shown. Are implemented for you. */

    public void push(T data);

    @SuppressWarnings("unchecked")
    public T pop();

    public int size();

    public int capacity();

    public void resize(int newCapacity);
}
```

1.3 TESTING AND SUBMITTING

We are providing a tester class with a main function. This class will add and remove many random elements to your two data structures and check that they work correctly. The tester will output an error message if anything goes wrong.

To submit, please zip up your entire project and submit on Collab.

2 In-Lab

The goal of this in-lab is to continue practicing with stacks and queues in a laid back environment. As usual, you will:

1. Get into small groups of two
2. The TAs will present you with some programming challenges regarding stacks and queues. These will not be graded, but you are required to attend and to participate.
3. You may think about the challenges before lab below if you'd like, but we highly recommend that you not solve them ahead of time.
4. The TAs will give you time to solve each problem and lead you in sharing solutions with one another.

2.1 Stacks and Queues Coding Challenges

The TAs will lead you in going through the following challenges. It is ok if you do not get through each of these.

1. Implement a basic stack (push and pop) using two queues. How fast can you make each operation?
2. Implement a basic queue (enqueue and dequeue) using two stacks. How fast can you make each operation?
3. Suppose you want to write a queue, except every item contains a *priority* value. Items of higher priority should be dequeue'd before others. Using a linked list, write the enqueue and dequeue functions for this kind of priority queue. What are the runtimes of your operations?
4. Implement a data structure that contains a single array, but within that array we store two stacks. The two stacks are not full until ALL of the underlying array is full (i.e., you cannot just partition the array into two halves).
5. In class, we discussed using a stack to implement a post-fix calculator. Given a post-fix expression as a string with spaces between tokens, write some code that computes the value of the expression using a stack.

3 POST-LAB

The goal of this post-lab is to write a report comparing the efficiency of linked-list based queues to array-based stacks. Theoretically, the runtimes for all operations on both these data structures is constant time. So, it is ok that we are comparing stacks to queues here. What we are really interested in is how much the resize operation on the stack, which is linear time and should slow us down, affects the overall performance of the stack versus the queue.

You will perform an experiment by doing the following:

1. Write some test code to time the various methods of your stack and queue. We are interested in testing enqueue versus push and dequeue versus pop

2. Run a small experiment timing each method. Because these methods are very fast, you'll need to invoke each MANY times before you see a slow enough performance and be able to compare your findings.

3. Write a report summarizing and analyzing your findings

4. **FILES TO DOWNLOAD:** None

5. **FILES TO SUBMIT:** PostLabFive.pdf

For this experiment, you should time your stack and queue by calling each method i. You may play around with i until you find a number that works well, but they should be the same for each method (e.g., don't execute enqueue 10^5 times and push 10^7 times). You should test the following methods:

```
1    /* You will test these methods: */

3    /* Compare the speeds of these two methods */
     Queue.enqueue(T data);
5    Stack.push(T data);

7    /* Seperately, compare the speeds of these two */
     Queue.dequeue();
9    Stack.pop();
```

3.1 REPORT

Summarize your experiment and your findings in a report. Make sure to adhere to these general guidelines:

- Your submission MUST BE a pdf document. You will receive a zero if it is not.
- Your document MUST be presented as if submitted to a professional publication outlet. You can use the template posted in the course repository or follow Springer's guidelines for conference proceedings.
- You should write your report as if it is original novel research.
- The grammar / spelling / professionalism of this document should be sound.
- When possible, do not use the first person. Instead of "I ran the code 60 times", use "The code was executed 60 times...".

In addition to the general guidelines above, please follow the following rough outline for your paper:

- **Abstract**: Summarize the entire document in a single paragraph
- **Introduction**: Present the problem, and provide details regarding the two strategies you implemented.
- **Methods**: Describe your methodology for collecting data. How many method calls, how many executions, how you averaged things, etc.
- **Results**: Describe your results from your execution runs.
- **Conclusion**: Interpret your results. Which methods were fast and which were slow? Did this surprise you? Does this align with the theoretical runtimes of those methods? How large did the lists need to get before you witnessed a slowdown?

Lastly, your paper MUST contain the following things:

- A table (methods section) summarizing the different methods and how many execution runs were done in each group.
- A table (results section) summarizing each method and the averages / std. dev. of runtimes for each (as well as any other data you decided to collect).

- Some kind of graph visualizing the results of the table from the previous bullet.