CSC 212: Programming with Data Structures

Midterm: Spring 2016

Thursday, March 10, 1-2:50pm

- This is an in-class exam on the material through Lab 6.
- It is closed notes, closed Internet, and closed technology, but you may use a "cheat sheet".
- Your cheat sheet must be hand-written, created by you, 8.5 × 11 inches, and double-sided (at most).
- I will come in a few times to answer questions.
- If there is a clarification to be made, I will write it on the board.
- Do not discuss the exam with other students and respect the honor code of doing your own work.
- If you are unable to make progress on any part of the exam, tell me what you tried; describe your thought process.
- Even if you are not finished, your exam must be turned in at the end of lab to maintain fairness.
- For ease of writing, I'll encourage using *uppercase, single-letter* variable names, but just for the exam!

Name	

Part 1	/15
Part 2	/15
Part 3	/20
Part 4	/20
Part 5	/15
Part 6	/15
Total	/100

Part 1: Warmup and Vocab

(a) What does FIFO stand for? What classical data structure has this property?

(b) What does FILO stand for? What classical data structure has this property?

(c) For each line of the code below, use the following phrases to describe what is happening: “declare”, “initialize”, “assign”, “memory allocated”, “get”, “set”

```
int[] X;

X = new int[3];

int[] Y;

Y = X;

Y[0] = 10;

int first = X[0];
```

Draw a diagram of the end result, showing `X` and `Y` clearly. What is `first`?

(d) Student X is writing a `ListTester` class, but they are confused about how to use the following `LinkedList` class to add an element to the *end* of a list. Student X writes:

```
public class ListTester {

    public static void main(String[] args) {
        List myList = new List();
        Node firstClass = new Node("212");
        Node secondClass = new Node("103");
        myList.tail = firstClass;
        myList.tail = secondClass;
    }
}
```

```
public class LinkedList {
    public Node head;
    public Node tail;
    ...

    private void add(String element) {
        ...
    }
    ...
}
```

What is wrong with this code? What changes could be made to `LinkedList` (and/or `Node`) to prevent Student X from doing this?

(e) Justify your answers to the following questions.

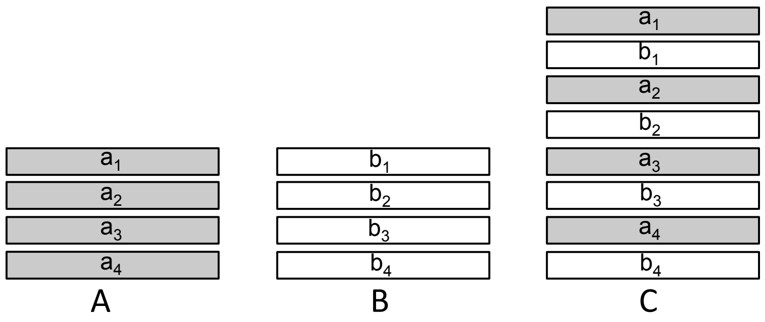
i. Is `JComponent` an interface or a class?

ii. What is an example of a class we have studied that is built upon `JComponent`?

iii. In the model-view-controller paradigm, what piece does your example fall under?

Part 2: Stacks

Write a method that will take in two stacks (`A` and `B`) of the same size (n) and interleave them, creating one stack (`C`). Whatever was at the top of stack `A` should now be at the top of stack `C`, as shown in the picture below for $n = 4$:



It is okay to destroy the original stacks in the process. The only `Stack` methods you may use are peek, push, pop, and isEmpty.

```
____________________ Stack interleave(Stack A, Stack B) {









}
```

- Based on the description and partial signature of the method, what Java keyword(s) should also be included in the method signature? Write on the dotted line.

- What is the runtime of your method, in terms of the size of each stack, n? Justify your answer.

Part 3: Arrays and Iterators

Complete the code below to write an iterator for an array of Strings. In particular, think about whether or not you want to add any fields or methods.

```
import java.util.Iterator;

public class ArrayIterator implements Iterator<String> {

    private String[] array;



    public ArrayIterator(String[] array) {
        this.array = array;


    }

    public boolean hasNext() {





    }

    public String next() {





    }
}
```

- Is it necessary to have an iterator for an array? Are there any advantages? Be specific.

- What Java construct is `java.util.Iterator`? What do `hasNext()` and `next()` look like inside `java.util.Iterator`?

Part 4: Loops and Linked Lists

In this question we'll be implementing a new *generic* data structure called a `Loop<E>`, using a **singly linked list of nodes** as the underlying data structure. Each node should have a pointer to the next node so that they form a continuous loop (like beads on a necklace). However, your loop data structure should have one field, a pointer to the "start" Node.

(a) Draw a diagram showing a single element being added to an empty loop. Then write code that corresponds to this operation (does not have to be encapsulated as a method).

(b) Draw a new diagram showing another element being added to the loop at the "start" (so now there are exactly two elements, starting with the newest one). Then write code to execute this operation (does not have to be a method).

(c) Finally, draw a new diagram that shows an element being added to the start of the loop, but with an arbitrary number of elements already in the loop. Then write code for a **method** that will accomplish this general task. Explain the issues involved, including efficiency (although it's not a requirement that your method be efficient). You can choose to augment the fields of your `Node<E>` data structure or your `Loop<E>` data structure, or leave them as they are, as long as your method accomplishes the goal.

Part 5: Sorting

Inspired by 212, after graduation you decide to work for *Sorters, Inc.* that provides sorting algorithms for every possible situation. One client explains that often their arrays are already mostly sorted (smallest to largest), so the version of insertion sort that you've provided (the one we covered in class) is doing too much extra work. Describe (using code or pseudocode), a modified insertion sort that will be more efficient with somewhat sorted arrays. An example is below:

```
int[] A = {3, 5, 1, 8, 17, 15, 29}
```

Requirements:

- Your algorithm should not modify the original array
- You should return a new sorted `LinkedList`

If you ran your algorithm on the example list above, *exactly* how many comparisons are performed?

Part 6: Runtime analysis

(a) What is the runtime of the shunting yard algorithm, in terms of the number of tokens n? Justify your answer.

(b) What is the runtime of the code below, in terms of n?

```
for (int i=0; i < n; i++) {
    for (int j=0; j < (int)(n/Math.pow(2,i)); j++) {
         array[i][j] = 1;
    }
}
```

Justify your answer. For fun: prove your answer.

More space (tell me what problem this is for):