

Name: \_\_\_\_\_

# CSSE 220—Object-Oriented Software Development

Final Exam, Nov. 16, 2009

This exam consists of two parts. Part 1 is to be solved on these pages. Ask your instructor for scratch paper if you need more room. Part 2 is to be solved using your computer. You will need network access to download template code and upload your solution for part 2. Please disable IM, email, and other such communication programs before beginning the exam.

*Resources for Part 1:* You may use a single sheet of  $8\frac{1}{2} \times 11$  inch paper with notes on both sides. Notes may be printed but must not be smaller than 4 point.

*Resources for Part 2:* Open book, notes, and computer. Limited network access. You may use the network only to access your own files, the course ANGEL site and web pages, the textbook's site, Sun's Java website, and Logan Library's Safari Tech Books Online. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

Parts 1 and 2 are included in this document. You should read over all of the questions before beginning work, but...

You must turn in part 1 before accessing the resources for part 2.

	<b>Problem</b>	<b>Poss. Pts.</b>	<b>Earned</b>
	1	8	_____
	2	6	_____
	3	6	_____
	4	8	_____
	5	9	_____
	6	12	_____
	7	15	_____
	8	12	_____
	9	8	_____
	10	12	_____
	11	8	_____
	<b>Paper Part Subtotal</b>	<b>104</b>	_____
	C1. LDComponent line appears	8	_____
	shape is closed	4	_____
	shape persists on resize	8	_____
	C2. DoublyLinkedList reverse() (2 points per unit test)	8	_____
	C2. DoublyLinkedList isInOrder() (2 points per unit test)	10	_____
	C2. DoublyLinkedList indexOf() (1 point per unit test)	8	_____
	<b>Computer Part Subtotal</b>	<b>46</b>	_____
	<b>Total</b>	<b>150</b>	_____

## Part 1—Paper Part

1. (8 points)

- a. Write a Comparator called `CompareStringsByLength` that compares two Strings by their length. Thus "dog" and "cat" are equal, "dog" comes before "elephant", and "aardvark" comes after "dog". (Note that the first and last of these comparisons are different than the normal String ordering.)

```
class CompareStringsByLength implements Comparator<String> {  
    @Override  
    public int compare(String s1, String s2) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

- b. One version of `java.util.Arrays.sort()` takes an array of objects and a Comparator that can compare those objects. Complete the code below to use your Comparator from part a to sort the array in order of increasing length.

```
public void sortByLength(String[] input) {  
  
  
  
  
  
  
  
  
  
    Arrays.sort(input, _____);  
}
```

2. (6 points) In the Markov project, you used a fixed length queue data structure. Briefly explain how we can use three fields—an array of size  $s$ , along with two int fields—to implement a fixed length queue. (You do not have to give the details of the implementation, just explain how the three fields would be used.)

3. (6 points) Suppose we have a program that creates two threads. One thread increases a field of type `int` by 10 every 50 ms. The other thread decreases the same field by 5 every 25 ms. The two threads are started at the same time.

- a. The field will never deviate by more than 20 from its initial value: **true** or **false**
- b. Justify your answer:

4. (8 points) What is wrong with the recursive algorithm below? (You do not have to fix the problem, just identify what it is.)

```
/**
 * @param arr
 * @return the sum of the elements in the given array
 */
public static int sumElements(int[] arr) {
    return sumElements(arr, 0);
}

// Helper method
private static int sumElements(int[] arr, int i) {
    if (i >= arr.length) {
        return 0;
    } else {
        return arr[i] + sumElements(arr, i);
    }
}
```

Answer:

5. (9 points) Suppose you are implementing a system for a client. You have two different algorithms to choose from for one of the main processing tasks.

- The first algorithm requires  $n^3 + n$  operations to process  $n$  elements of input.
- The second algorithm requires  $3n^2 + 10000n$  operations for the same input.

Your client expects the system to scale to large data sets.

- a. Which algorithm would you use? (circle one)            the first            the second
- b. Briefly explain your answer to part a:

6. (12 points) Consider the following initial array configuration. In each question, assume we want to sort the array in **ascending** order (that is, from smallest to largest).

7	8	1	5	6	3	4	2
---	---	---	---	---	---	---	---

a. Suppose the **selection** sort algorithm from class is applied to the **initial array at the top of the page**. Show the state of the array immediately following the first **three** executions of the outer loop.

--	--	--	--	--	--	--	--

---

b. Suppose the **insertion** sort algorithm from class is applied to the **initial array at the top of the page**. Show the state of the array immediately following the first **four** executions of the outer loop.

--	--	--	--	--	--	--	--

---

c. Suppose the **merge** sort algorithm from class is applied to the **initial array at the top of the page**. Show the state of the two sub-arrays immediately before the final merge.

--	--	--	--

--	--	--	--

7. (15 points) For each of the scenarios below, indicate which of the *data structures* we studied—**array list, linked list, stack, queue, hash set, tree set, hash map, or tree map**—would be best. Justify your answer.

- a. Simulating a paper in-box, where new items are placed on the top of the pile and work items are taken from the top of the pile.

Data Structure: \_\_\_\_\_

Justification:

- b. Tracking the point values for each letter in the game of Scrabble. (In Scrabble common letters like E, S, and T are worth 1 point each. Less common letters are worth more points, with maximum points for Q and Z, which are worth 10 points each.)

Data Structure: \_\_\_\_\_

Justification:

- c. Modeling the waiting list to get into a section of a course that is over-capacity.

Data Structure: \_\_\_\_\_

Justification:

8. (12 points) Predict the output of each of the following code blocks:

a. 

```
Stack<Integer> s = new Stack<Integer>();
s.push(1);
s.push(2);
s.push(3);
System.out.print(s.pop());
System.out.print(s.pop());
s.push(4);
System.out.print(s.pop());
s.push(5);
System.out.print(s.pop());
System.out.print(s.pop());
```

Answer: \_\_\_\_\_

b. 

```
HashMap<Character,String> map =
    new HashMap<Character, String>();
map.put('a', "Turkey");
map.put('b', "Pumpkin Pie");
map.put('a', "Tofurkey");
System.out.print(map.get('b'));
System.out.print(map.get('a'));
```

Answer: \_\_\_\_\_

c. 

```
HashMap<String, HashSet<Integer>> map =
    new HashMap<String, HashSet<Integer>>();
map.put("Turkey", new HashSet<Integer>());
map.put("Dressing", new HashSet<Integer>());
map.get("Turkey").add(10);
map.get("Turkey").add(20);
map.get("Turkey").add(30);
map.get("Dressing").add(40);
map.get("Dressing").add(40);
map.get("Dressing").add(50);
map.put("Turkey", new HashSet<Integer>());
System.out.print(map.get("Turkey"));
System.out.print(map.get("Dressing"));
```

Answer: \_\_\_\_\_

9. (8 points)

a. Suppose an algorithm takes 3 seconds for an input of size 10,000. If the algorithm is  $O(n^3)$ , approximately how many seconds does it take for an input of size 20,000? (Circle one.)

1.5      3      4.5      6      9      12      18      24      30

b. Suppose an algorithm takes 100 ms for an input of size 1,000,000. If the algorithm is  $O(n^2)$ , about how large an input could it process 900 ms. (Circle one.)

500,000   1,000,000   1,500,000   2,000,000   3,000,000   4,500,000   9,000,000   27,000,000

10. (12 points) Consider the following code:

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < N - 1; j++) {  
        foo();  
    }  
}
```

a. Exactly how many times is `foo()` called, in terms of  $N$ ? \_\_\_\_\_

b. Fill in the blank: Your answer to part *a* is  $O(\text{_____})$ .

c. Suppose the condition on the inner loop were changed to  $j < i - 1$ .

Now exactly how many times would `foo()` be called, in terms of  $N$ ? \_\_\_\_\_

d. Fill in the blank: Your answer to part *c* is  $O(\text{_____})$ .

11. (8 points) Suppose the input file to the Markov Chain algorithm contains only the following line:

fo og f fo g f fo og f g f fo f

**Let the prefix length be 3.** For each of the following, indicate whether or not it would be a valid output of the algorithm, assuming the output isn't terminated early:

fo og f fo g f fo f    valid    invalid

fo g f fo f            valid    invalid

fo og f g f fo f        valid    invalid

fo og f g f fo og f    valid    invalid

Name: \_\_\_\_\_

## Part 2—Computer Part

*Resources for Part 2:* Open book, notes, and computer. Limited network access. You may use the network only to access your own files, the course ANGEL site and web pages, the textbook's site, Sun's Java website, and Logan Library's Safari Tech Books Online. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

You must turn in the preceding pages  
before accessing the resources for part 2.

**Instructions.** You must actually get these problems working on your computer. Almost all of the credit for this problem will be for code that actually works. There are several different small methods to write, so you can get a lot of partial credit by getting some of them to work. If you get every part working, comments are not required. If you do not get a method to work, comments may help me to understand enough so I can give you (a small amount of) partial credit.

After you have handed in part 1, **begin part 2 by checking out the project named *FinalExam* from your course SVN repository.** (Ask for help immediately if you are unable to do this.)

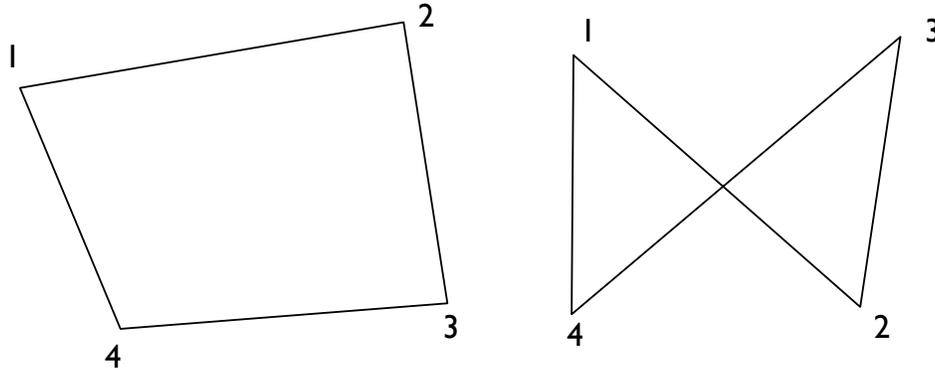
When you have finished the problems, and more frequently if you wish, you should **submit your code by committing it to your SVN repository.** We will check commit logs, so you must be careful not to commit anything after the end of the exam. For grading, we will ensure that the included JUnit tests have not been changed.

---

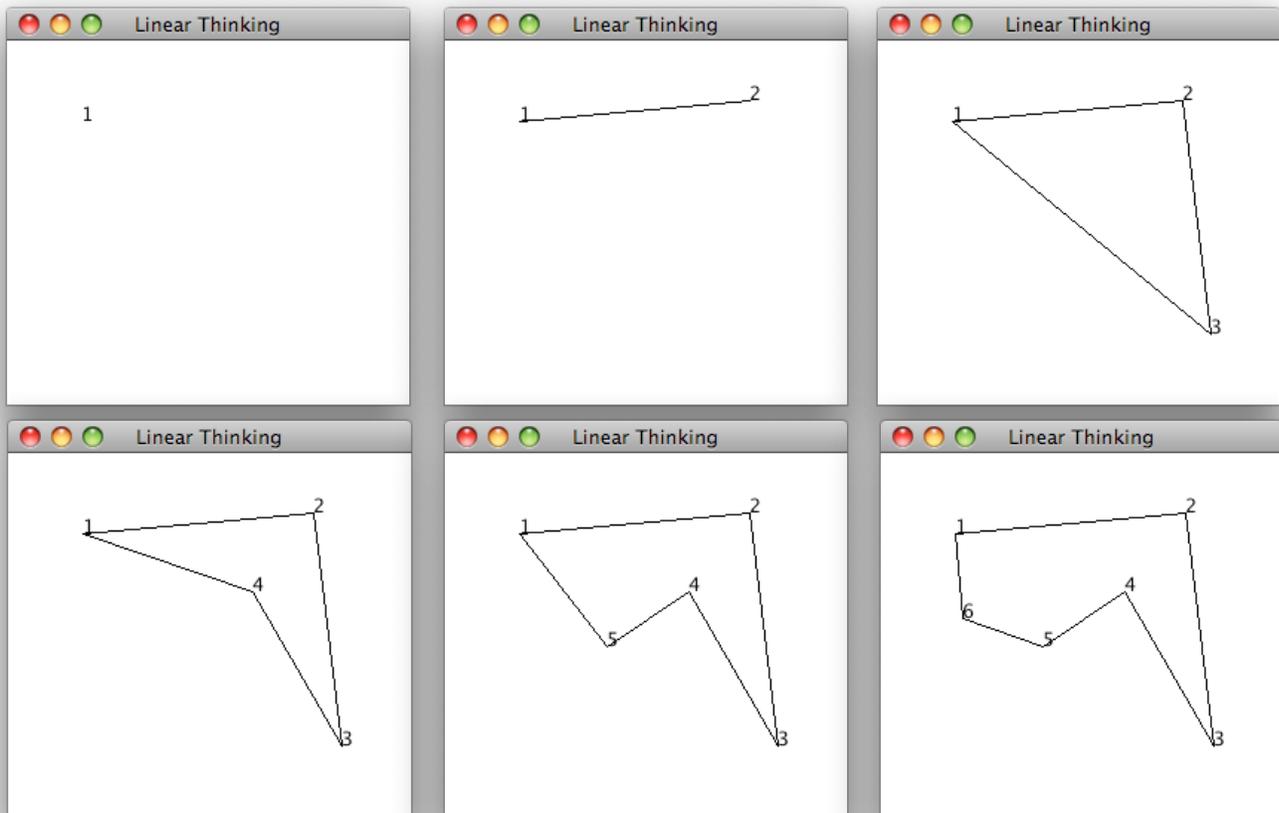
### Problem Descriptions

C1. (20 points) A class `LineDrawerMain` is provided that constructs a `JFrame` and adds a single `LDComponent` to it. `LDComponent` is a subclass of `JComponent`. `LDComponent` is partially implemented. Your task is to finish it by adding an appropriate listener and finishing the `paintComponent` method.

The completed GUI should draw a series of lines based on the user's mouse clicks. After one click, nothing should appear. After two clicks, a line should appear between the location of the first click and the location of the second. A third click changes the shape to a triangle connecting the three points. A fourth click changes the shape to one with four sides. The points should be connected in the order clicked, so this four sided shape might appear as a quadrilateral or as a bow-tie shape as shown in the figures below. (The numbers demonstrate the order of the clicks; you do *not* need to include the numbers in your solution.)



Additional clicks add additional lines in order. For full credit the shape must remain if the window is resized. The figures below shows successive windows after 1 through 6 clicks. (Again, you do *not* need to include the numbers in your solution.)



C2. (26 points) In the list package you will find a version of doubly linked list like the one from class. For this problem, your task is to add some additional methods. The javadoc specifications for those methods are given below.

JUnit tests are provided in list.DoublyLinkedListTest.

```
/**
 * Returns a NEW doubly linked list consisting of the elements of this list
 * in reverse order. Leaves the original list unchanged.
 *
 * @return a new, reversed list
 */
public DoublyLinkedList<E> reverse() {
    // TODO: implement reverse()
}

/**
 * Checks whether the items in this list are in order according to the given
 * Comparator object.
 *
 * @param comp a function object to compare the elements of this list
 *
 * @return true if and only if this list is sorted according to comp
 */
public boolean isInOrder(Comparator<E> comp) {
    // TODO: implement isInOrder()
}

/**
 * Searches this list for an object equal to the given object. Returns the
 * index of that object's position in the list. That is, returns 0 if the
 * given object is equal to the first item in this list, 1 if it is the
 * second, and so on. Returns -1 if the object doesn't appear in the list.
 *
 * @param object
 *
 * @return the index of the object in the list, or -1 if it doesn't appear
 */
public int indexOf(E object) {
    // TODO: implement indexOf(). For full credit you must add and use an
    // appropriate helper method in the Node inner class.
}
```