CSSE 220---Object-Oriented Software Development

# Final Exam -- Part 2, November 20, 2019

**Allowed Resources for Part 2**

Open book, open notes, and open computer. Limited network access. You may use the network only to access your own files, the course Moodle and Piazza sites (but obviously don't post on Piazza) and web pages, the textbook's site, Oracle's Java website, and Logan Library's online books.

**Do not use non--approved websites like search engines (Google) or any website other than those listed above.**

**Instructions**

You must disable Microsoft Lync, IM, email, and other such communication programs before beginning part 2 of the exam. 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 actually get these problems working on your computer. Almost all of the credit for the problems 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 you can earn (a small amount of) partial credit.

Submit your modified source code via Moodle. **Be sure to check your Moodle submission to ensure you submitted the right version of the files before leaving.**

**Part A: 3 Linked List** (18 points) Implement the code for the 3 unimplemented methods in SinglyLinkedList.java -- each problem is worth 6 points. Instructions are included in the comments of each method. Unit tests are included in SinglyLinkedListTest.java.

**Part B: Recursion** (6 points) Implement the code for the unimplemented method in RecursionProblems.java. Instructions are included in the comments of the file and corresponding unit tests are included in RecursionProblemsTest.java.

**Part C: HashMap** (6 points) Implement the code for the unimplemented method in HashMapProblem.java. Instructions are included in the comments of the file and corresponding unit tests are included in HashMapProblemTest.java.

**Part D: Refactoring** (13 points)

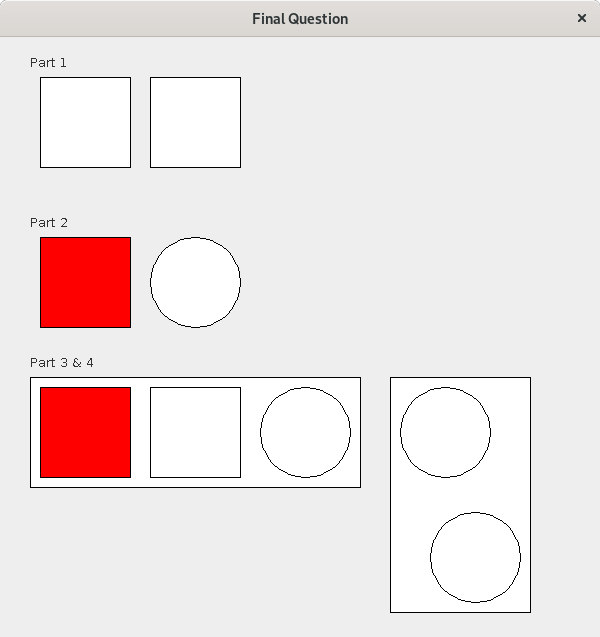
Take a look in OptionsMain and the Options dialog classes. The given code asks the user for an integer, and depending on that integer makes a window that allows the user to select between a list of Strings. Depending on the class they select using buttons, radio buttons, or from a JList list. This code functions correctly but is poorly designed.

1. (5 points) The three classes OptionsButtonDialog, OptionsListDialog, and OptionsRadioButtonDialog have duplication. Introduce a new superclass (AbstractOptionsDialog) and remove the duplication between these classes. For now, you can ignore the partial duplication in showWindow() and focus on the other methods and fields that are exactly duplicated.

2. (5 points) The OptionsMain classes has duplication in the three if clauses. Remove this duplication. HINT: you’ll probably want to make an ArrayList of 3 AbstractOptionsDialog objects, and then select a particular entry from that list.

3. (3 points) the showWindow() methods in the Dialog classes are identical at the beginning, identical at the end, but are different in the middle of the method. Find a way to remove some or all of the duplicate code between the 3 versions of this method.

**Part E: Final Question** (22 points)

In this problem you start with a Rect class that draws a rectangle at a particular position on the screen. RectMain is the main function for this code.

You can see a video of the click behavior in action here:

<https://youtu.be/bV3Qejdz8PU>

**Stage 1** (6 points)

Make instances of the Rect class change to filled in black squares when clicked. To do this you’ll want to register your MouseListener class on the RectComponent JComponent subclass, not the frame (or your clicks will always be a little off). When a click occurs, it should cause the RectComponent to loop across all its contained Rects, find one that matches the click location, and then invoke some sort of click handling method on it. Determine if the click is within the rect using Rect’s containsPoint method. Be sure to call repaint!

**Stage 2** (7 points)

Uncomment the code in RectMain corresponding to this part.

We want to add 2 new kinds of RectComponent, RedRects and CircleRects. RedRects act like regular rects except their background starts as red and they turn pink when clicked. CircleRects act like Rects (including click behavior) but they are a circle rather than a square. Use inheritance to minimize duplication between these 2 classes and Rect.

If you need to, the click detection on CircleRect can be a little innaccurate.

**Stage 3** (6 points)

Uncomment the code in RectMain corresponding to this part.

Add a 3rd kind of Rect called ContainerRect. ContainerRect has other Rects added to it. The size of the container rect is always 10 pixels larger on every size than its contained rects (hint: make the addRect method adjust the size of the container as new Rects are added). ContainerRects draw as a white box, and then the contained rects should draw within in their usual way.

Don’t worry about click behavior in this part.

**Stage 4** (3 points)

Note that this part is pretty complicated for the small amount of points it is worth.

Uncomment the small amount of code in RectMain corresponding to this part, and think about why that code is necessary.

When a container rect is clicked, its background should immediately switch between yellow and white. However, the child rects within the container should not immediately change. Instead 500 milliseconds after the click, the first child rect should change, 500 milliseconds after that next child should change, etc. You don’t have to worry about what the correct behavior is if another click occurs before all the children have finished changing.