Name: ______________________________ Section: ________ CM: __________

CSSE 220—Object-Oriented Software Development

Final Exam – Part 2, Nov. 16, 2016

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.

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 I can give you (a small amount of) partial credit. ***NOTE: CODE THAT DOES NOT COMPILE WILL NOT BE ELIGIBLE FOR PARTIAL CREDIT!***

Begin part 2 by checking out the project named *Final-201710* from your course SVN repository. (Ask for help immediately if you are unable to do this.)

When you have finished a problem, and more frequently if you wish, **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.

Part 2 is included in this document. **Do not use non–approved websites like search engines (Google) or any website other than those listed above.** Be sure to turn in these instructions, with your name written above, to your exam proctor. You should not exit the examination room with these instructions.

Part 2—Computer Part

Problem Descriptions

Part A: 3 Linked List (18 points) Implement the code for the 3 unimplemented methods in StringLinkedList.java – each problem is worth 6 points. Instructions are included in the comments of each method. Unit tests are included in StringLinkedListTest.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: 1 Threads (8 points)

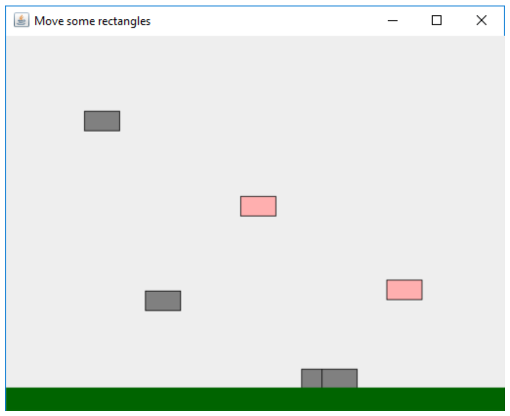
In this problem we have provided you with code that calculates a result based on a array of doubles. The result is (product of all doubles) - (sum of all doubles).

Stage 1 (4 Points)

Modify the function computeSumAndProductInParallel so that it computes the sum and product at the same time using threads. For this stage, you don't need to actually compute the final subtraction correctly, the printouts should simply show that both the sum and the product start together and compute their results. For example:

```
Starting parallel test...
Starting computing product
Starting computing sum
Finished computing product
Finished computing sum
Calculated sum is: 10034.515662779348
Calculated product is: 1.2338255311192952E-180
```

Stage 2 (4 Points) Further modify the code so that the final subtraction computes correctly in the parallel version. The result should be the same as the serial version.



The final stage of GravityRects, including PinkRects and FastRects

Part E: GravityRects (27 points)

In this problem you will implement rectangles that will be affected by gravity.

These parts can be implemented out of order.

Stage 1 (9 Points)

The given code draws 2 GravityRect objects on the screen. Add animation so that these rectangles drop at a constant rate, eventually coming to a stop on the "floor" at the bottom of the screen.

Stage 2 (9 Points)

Modify the code so that when you click on the window, a new GravityRect gets added to the window at the position you clicked. The newly added GravityRect should drop as the existing GravityRects do.

Note that if multiple GravityRects land in the same horizontal position, they do not need to "stack". They should all drop to the "floor" level – one Rect should appear to cover the other.

Stage 3 (9 Points)

We want to introduce 2 new kinds of GravityRects. PinkRects should act like existing GravityRects except they should be colored Pink. FastRects should look like existing GravityRects but they should fall 3x as fast.

Make 2 subclasses of GravityRect to introduce this new behavior. DO NOT just add fields to your existing GravityRect object for color and speed. Minimize the amount of code duplication between the subclasses and the superclass GravityRect.

Modify your click handling code to add these new rects: shift-click should add a PinkRect, Control-click should add a FastRect. HINT: you can detect the shift/control state of a mouseclick by looking at the MouseEvent object (e.g. myMouseEvent.isControlDown()). If your mouse listener in stage 2 doesn't work, you can still earn points for stage 3 by adding some PinkRects and FastRects to the start screen.