

Name: \_\_\_\_\_ Section: \_\_\_\_\_ CM: \_\_\_\_\_

# CSSE 220—Object-Oriented Software Development

## Final Exam – Part 2, Feb. 27, 2014

**Allowed Resources for Part 2.** Open book, open notes, and 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.

**Begin part 2 by checking out the project named *FinalExam-201420* 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 the 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** (15 points) Implement the code for the 3 unimplemented methods in `StringLinkedList.java` – each problem is worth 5 points. Instructions are included in the comments of each method. Unit tests are included in `StringLinkedList.java`.

**Part B: 3 Recursion** (12 points) Implement the code for the 3 unimplemented methods in `Recursion.java` – each problem is worth 4 points. Instructions are included in the comments of each method. Unit tests are included in `RecursionTest.java`.

**Part C: Multithreading** (8 points) The code in `TimedChecker.java` is designed to simulate a system that runs a system integrity check at regular intervals (in this case, every second). Run the code in `TimedChecker` and click the button. You should see something like this in the console:

```
2014/02/25 16:26:56 Doing system check...done
2014/02/25 16:26:57 Doing system check...done
2014/02/25 16:26:58 Doing system check...done
2014/02/25 16:26:59 Doing system check...done
2014/02/25 16:27:00 Doing system check...done
2014/02/25 16:27:01 Doing system check...done
2014/02/25 16:27:02 Doing system check (this could take a while)...
System check started on 2014/02/25 16:27:02 finally complete
2014/02/25 16:27:08 Doing system check...done
```

You can see that in general, the system check runs every second. BUT, when the system check occasionally takes a long time, it delays the other system checks. Look at the system check that starts on 16:27:02 ... the next system check doesn't start till 16:27:08. This is a problem because it means that the system was not being checked during those 6 seconds.

The behavior we want is for other system checks to start and run every second, even if there is another long running system check currently in process. In this system, there is no problem with multiple system checks running at once. So the result ought to look like this:

```
2014/02/25 16:26:59 Doing system check...done
2014/02/25 16:27:00 Doing system check...done
2014/02/25 16:27:01 Doing system check...done
2014/02/25 16:27:02 Doing system check (this could take a while)...
2014/02/25 16:27:03 Doing system check...done
2014/02/25 16:27:04 Doing system check...done
2014/02/25 16:27:05 Doing system check...done
2014/02/25 16:27:06 Doing system check...done
2014/02/25 16:27:07 Doing system check...done
System check started on 2014/02/25 16:27:02 finally complete
2014/02/25 16:27:08 Doing system check...done
```

Use Java threads to modify the system to get the right system check behavior. Each time the system check runs, it should run in its own thread, ensuring that long running system checks do not block other system checks from running.

**Part D: Spaceships** The code you're given is a simple graphics class with 2 kinds of Spaceship objects — TeleportySpaceship and MovingSpaceship. In the given code, one of each of these objects is drawn on the screen but they don't do anything.

- Stage 1 (10 Points) The teleporty spaceship should teleport (move to the clicked location) to wherever the user clicks the mouse button in the window. It should immediately draw under the mouse wherever a click happened. Correctly implement a java MouseListener to make this behavior work.
- Stage 2 (10 Points) The MovingSpaceship should not teleport to wherever a user clicks, but it should move in that direction. The motion can be whatever you find easy to implement (e.g. it could move in a straight line to the click point, or it could move north/south then move east/west, or whatever). It also does not have to move at a constant speed. BUT it should clearly be animated and moving towards the click point. You'll need to use java timers or multithreading to make this work.
- Stage 3 (10 Points) In the code you were given, SpaceshipComponent has two fields (teleporter and mover). However, we want to expand this code to support an arbitrary number of spaceships and new kinds of spaceship classes. We want a single ArrayList that can store both Teleporty and MovingSpaceships. Add a new interface or abstract class for spaceships to make the changes necessary to TeleportySpaceship and MovingSpaceship. Then replace the two fields in SpaceshipComponent with the arraylist and modify your SpaceshipComponent code to work correctly. Feel free to add a couple more MovingSpaceships to SpaceshipComponent's constructor to show it working correctly.

You must turn in these instructions with your name written on the first page before exiting the examination room.