

Name: _____ Section: _____ CM: _____

CSSE 220—Object-Oriented Software Development

Final Exam – Part 2, Nov. 20, 2015

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. **NOTE: CODE THAT DOES NOT COMPILE MAY NOT BE ELIGIBLE FOR PARTIAL CREDIT!**

Begin part 2 by checking out the project named *Final-201510* 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 `StringLinkedListTest.java`.

Part B: 2 Recursion (10 points) Implement the code for the 2 unimplemented methods in `RecursionProblems.java` – each problem is worth 5 points. Instructions are included in the comments of each method. Unit tests are included in `RecursionProblemsTest.java`.

Part C: 1 HashMap (5 points) Implement the code for the single unimplemented method in `HashMapQuestion.java` – this problem is worth 5 points. Instructions are included in the comments of the file and corresponding unit tests are included in `HashMapQuestionTest.java`.

Part D: Multithreading (10 points) The code in `ThreadsCommand.java` is designed to simulate a system that both starts and stops threads. The system has 3 commands:

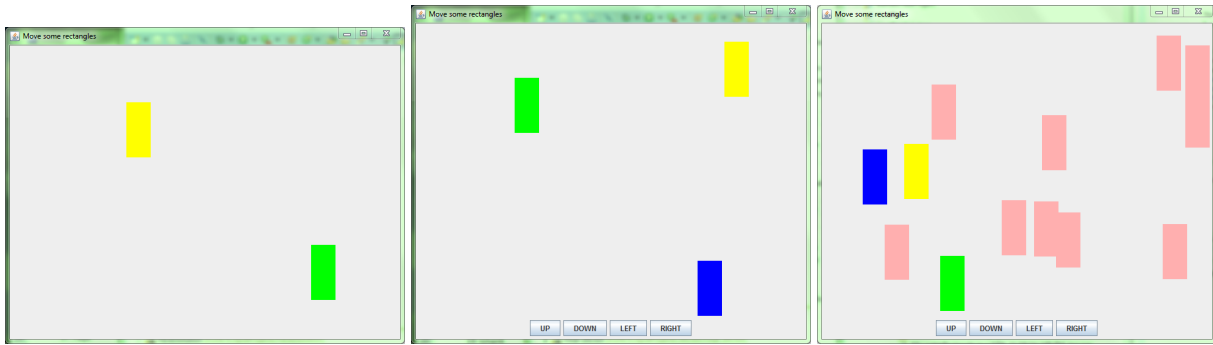
- a. *create*. Creates a new, independently running thread with a new thread number (starts at 1 and goes up with each new thread). Created threads don't really do anything, but every 5 seconds they print a message like "Thread NUM checking in every 5 seconds".
- b. *stopall*. This command stops all running threads. The threads need to exit gracefully – they shouldn't be killed manually using `Thread.stop()`. When a thread is stopped it should print "Thread NUM stopping gracefully". Threads don't have to stop immediately when you execute the *stopall* command — they can stop within 5 seconds. However, once *stopall* is called no existing threads should print checkin messages.
- c. *exit*. Ends the program. Does not need to do anything special with the threads and it's already implemented for you.

Using the system you can start several threads, then stop them, then start some more. For example:

Welcome to ThreadCommand. Enter your commands.

```
create
Thread 1 checking in every 5 seconds
create
Thread 2 checking in every 5 seconds
Thread 1 checking in every 5 seconds
Thread 2 checking in every 5 seconds
Thread 1 checking in every 5 seconds
create
Thread 3 checking in every 5 seconds
Thread 2 checking in every 5 seconds
Thread 1 checking in every 5 seconds
Thread 3 checking in every 5 seconds
Thread 2 checking in every 5 seconds
stopall
Thread 1 stopping gracefully
Thread 3 stopping gracefully
Thread 2 stopping gracefully
create
Thread 4 checking in every 5 seconds
Thread 4 checking in every 5 seconds
stopall
Thread 4 stopping gracefully
exit
```

Implement the create and stopall commands. You can create classes in new files or just modify the file given.



Stage 1 (left) - a rectangle that moves towards mouse clicks. Stage 2 - a rectangle that moves when you press UI buttons. Stage 3 (right) random rectangles of both types. *Note that initial rectangle position is random.*

Part E: Moveable Rectangle

Start by running and understanding the given code. You have been provided with a class called `MoveableRectangle`. `MoveableRectangles` start in random locations, have velocities, and never move off the given window. The code in `MoveableRectangleComponent` makes the `MoveableRectangles` animate.

- Stage 1 (10 Points) Make a new subclass of `MoveableRectangle` called `MouseMoveableRectangle`. `MouseMoveableRectangles` should implement the `MouseListener` interface and respond to mouse clicks on the screen. When the user clicks on the screen, the `MouseMoveableRectangle` should move in the direction of the click. It doesn't need to stop once it gets to the location though - it can continue on past. Add one `MouseMoveableRectangle` to the screen to prove your code works — it should be yellow. If you can't figure out how to get the code working using a subclass of `MoveableRectangle`, you can implement the animated moving rectangle some other way and still get full credit for Part 1. BUT doing so will make it almost impossible to do part 3.
- Stage 2 (10 Points) Make a new subclass of `MoveableRectangle` called `ButtonMoveableRectangle`. `ButtonMoveableRectangles` are controlled by 4 buttons at the bottom of the screen: UP, DOWN, LEFT, RIGHT. If up is pressed, all `ButtonMoveableRectangles` should start going up (and the other buttons should work similarly). Add buttons to the frame to control `ButtonMoveableRectangles`. Add one `ButtonMoveableRectangle` to the screen to prove your code works — it should be blue. If you can't figure out how to get the code working using a subclass of `MoveableRectangle`, you can implement the button controlled rectangle some other way and still get full credit for Part 2. BUT doing so will make it almost impossible to do part 3.
- Stage 3 (10 Points) Now make a 10 element array of `MoveableRectangles` in `RectangleComponent` called `rectangles`. When a new `MoveableRectangleComponent` is constructed, that list should be RANDOMLY populated with `ButtonMoveableRectangles` and `MouseMoveableRectangles`. There should be a 50/50 chance of one or the other appearing at a particular spot in the list. Modify the code so all the rectangles are displayed. The new random rectangles should all be pink. Each random rectangle should still do its correct moving behavior (move based on mouse clicks or move based on buttons).