CSSE 220---Object-Oriented Software Development

Final Exam -- Part 2, November 14, 2018

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.

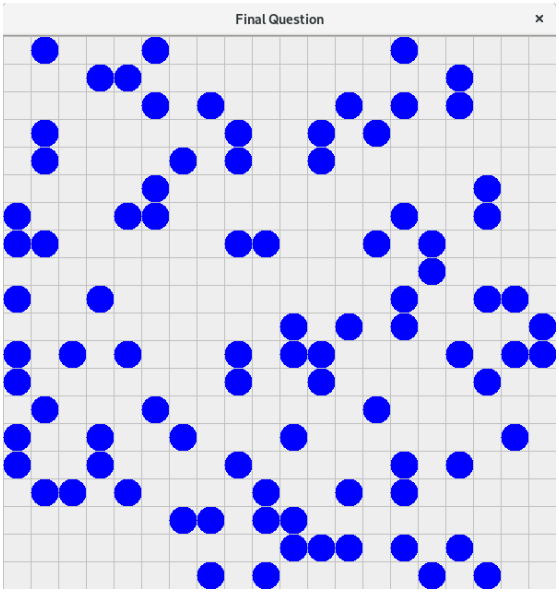
Part D: Refactoring (13 points)

The given code simulates a baker who bakes 3 kinds of foods – cookies, pizza, and souffles. The code is functioning as is but has some problems. FoodCookies, FoodPizza, and FoodSouffle are code-wise very different from each other and therefore these is not much duplication between them. BUT the code that uses these 3 classes in FoodMain has both duplication and type-predicated code. Use either interfaces or inheritance to solve these problems. You should be able to do this with relatively minor changes to the system (e.g., you don't need to rewrite all 3 foods and combine them into one class, say).

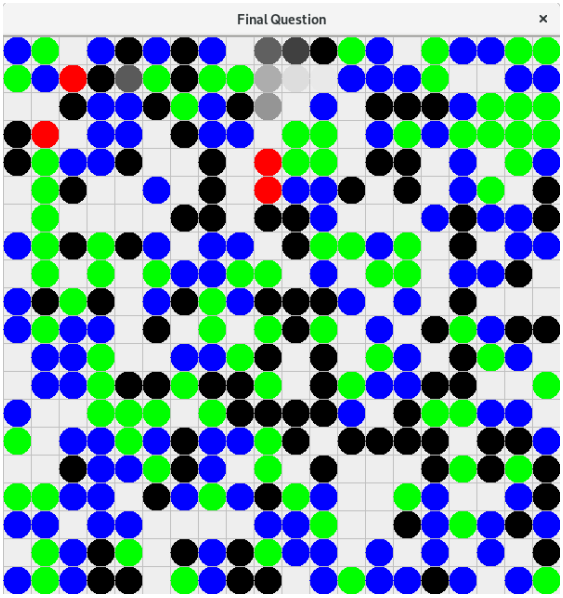
Your change should not change the functionality of the system. Here's a sample system output (note that the souffle baking includes randomness so it might print slightly differently when you run it):

```
mixing flour and water to make dough
adding tomato sauce, cheese, and pepperoni
baked 1 pepperoni pizza!
mixing flour, sugar, butter, and chocolate chips to make dough
baked 24 cookies
baked 24 cookies
baked 24 cookies
baked 24 cookies
baked 24 cookies
mixing flour and water to make dough
adding tomato sauce, cheese, and olive
baked 1 olive pizza!
mixing flour and water to make dough
adding tomato sauce, cheese, and mushroom
baked 1 mushroom pizza!
mixing cream eggs and spices to make souffle
attempting to bake 1 souffle...but it didn't rise
mixing cream eggs and spices to make souffle
attempting to bake 1 souffle...but it didn't rise
mixing cream eggs and spices to make souffle
attempting to bake 1 souffle...success!
```

Part E: Final Question (22 points)



Grid after Stage 1



Grid after Stage 4, and some color changing and animated balls have been clicked.

In this problem we have given you code that draws a grid on the screen. We want you to fill this grid with colored balls, each of which has different behavior.

Stage 1 (6 points)

Populate a selection of grid cells with blue balls. Each grid cell should have a 25% chance of being filled with a blue ball. **NOTE**: although the population of the cells should be random every time you start the program, once populated the grid should remain the same (i.e., if you resize the window the same cells should be populated with blue balls).

Stage 2 (6 points)

Make it so when a cell with a blue ball is clicked with the mouse, it prints "clicked r c" to the console where *r* is the row number of the ball clicked and *c* is the column number. For example, clicking on a ball in the upper left corner should print "clicked 0 0" to the console. Clicking on a cell that does not contain a ball should do nothing. You'll probably want to create a ball class to do this (as you'll need it for the later parts). **NOTE**: if it makes your life easier, you can print clicked if the click happens anywhere within the grid cell that contains a ball, rather than just if it happens to be within the radius of the ball.

Blue ball click demonstration video: https://youtu.be/m09yQU-ffCY

Stage 3 (6 points)

Add color changing balls to the grid. Each grid cell should have a 25% chance of being filled with a color changing ball (though only one ball can fill a particular grid cell). Color changing balls are initially green. Make a new class to represent color changing balls and **use inheritance to reduce duplication between the ball classes as much as possible**. Color changing balls should switch to red when clicked and then back to green if clicked again, then back to red if clicked again, etc. **NOTE:** Don't forget to call repaint!

Green ball demonstration video: https://youtu.be/-Wo_pJAnJdI

Stage 4 (4 points)

Add animated balls to the grid. Each grid cell should have a 25% chance of being filled with an animated ball (though only one ball can fill a particular grid cell). Animated balls are initially black. Make a new class to represent animated balls and use inheritance again to reduce duplication. Animated balls should smoothly animate from black to white, then when the clicked-on ball reaches all white, it should immediately turn back to black, and then repeat the animation process, etc.

Black ball demonstration video: https://youtu.be/I7fCk15c4gg