COMS W1007x: Object-Oriented Programming & Design (Java)
Fall, 2011
Assignment 3: Due Thursday, November 10, 2011, 1:10 pm, in class

Part 1: Theory (45 points)

The following problems are worth the points indicated. They are generally based on the book's exercises at the end of Chapters 4 and 5, under the section heading "Exercises". They cover issues in the design of interfaces and in the use of patterns.

"Paper" programs are those which are written on the same sheet that the rest of the theory problems are written on, and are not to be text-edited, compiled, executed, or debugged. For fairness, there will be no extra credit for doing any work beyond the design and coding necessary to produce on paper the objects or object fragments that are asked for, so computer output will have no effect on your grade. The same goes for the theory portion in general: clear handwriting is sufficient, and computer-generated documents will earn no extra credit.

1) (10 points) Exercise 4.4, except use the class DormRoom, which has a occupancy limit, and a floor number within its building. This is basically a variation on what the book does for the class Country in Sections 4.3 and 4.4. Write a paper program for the class, and another paper program for its tester.

2) (10 points) Exercise 4.4 again, except using Comparator, so that instance of DormRoom can be sorted by either occupancy or by floor, without having to recompile the class DormRoom.

3) (6 points) Some authors suggest that patterns come in groups that are sometimes called super-patterns. Five such super-patterns are (in alphabetical order): Behavioral, Concurrency, Creational, Fundamental, and Structural patterns. Look up the following patterns (which are in alphabetical order), describe them, and say what family they are in: Balking, Builder, Delegation, Facade, Memento. Which of these is the oldest? the newest?

4) (6 points) There are a lot of "anti-patterns". Look up what they are, and define and give one example for each of five of them. Note: do *NOT* define or use any of the following, as they are just too horrible to contemplate, and any exposure to their definitions will probably damage your designing skills forever: Big_Ball_Of_Mud, Boat_Anchor, Gas_Factory, God_Object, Gypsy_Wagon, Magic_Pushbutton, Poltergeist, Stovepipe_System.

5) (6 points) Generalize 5.14 but do not write a paper program. That is, the question becomes: what problems can occur when there are more than one decorator that both use methods of the same base component? Hint: why do bold and italics work as decorators of a font, regardless of their order of application? Further hint: think about sequence diagrams. How can these problems be fixed in general?

6) (7 points) Roughly based on 5.17 (which has an answer online), but using a real world pattern. Consider that people who bring their lunch in a brown bag also often use the same brown bag to hold all the garbage of the meal for disposal. Show how this can be generalized into a pattern by formalizing it with a name, context, problem, and solution. Could this pattern ever be used in software?

Part 2: Programming (55 points)

This assignment is intended to explore both interfaces and GUI development via patterns.

Please recall that all programs must compile, so keep a working version of each part before your proceed to the next. Please also note that softcopy of Javadoc output is required, including class comments, constructor comments (with @param) and method comments (with @param and @return, as appropriate). You also need to provide test examples in both the softcopy and the hardcopy.

Step 1 (18 points): Getting started. Implement the code in section 4.10, pages 166 to 169. The source code can be downloaded from the book's on-line companion. Make sure you properly attribute the source of this code! Then, you should rearrange the code (particularly in AnimationTester) and remove any magic numbers (particularly in CarShape) so that it is easier to understand and is more self-documenting. Test the code and submit some evidence of your testing.

For the magic numbers, you can use the "cheatsheet" about CarShape that is on Courseworks, or your own modification of it. In particular, make sure all CONSTANTS are defined using the all-caps convention.

Step 2 (19 points): Using Step 1, do 5.4, except that your slider controls the car's *speed*. You should check the answer for 5.3 in the book's on-line companion for a few hints. If you do this Step, you do not have to submit separate output from Step 1.

Step 3 (18 points): Creativity Step. Using Step 2, pick *two* of the following; each one is worth 9 points. If you do this Step, you do not have to submit separate output from Step 1 or Step 2. However, you do have to show either one use of an additional interface, or one use of an additional pattern. This must be clearly documented so that the grader understands how you used design methods to modify the original specifications given in Steps 1 and 2.

a) (9 points) Add something like the enchancement described in Exercise 4.14, which changes some or all of the car's color.

b) (9 points) Add the enhancement of Exercise 4.20. Your slider will then simply control the speed of all the cars at once. Nevertheless, each car should start off in a random location with a random speed.

c) (9 points) Add the enhancement of Exercise 4.22 ("wrap around").

d) (9 points) Add the enhancement of Exercise 4.22, except that the car now "bounces" off the left and right walls, shifting direction like a billiard ball but retaining speed.

e) (9 points) Have the car show where it has been by having the last N places still appear on the display, much like "mouse trails" do in the Windows operating system. This usually requires the use of the Queue object in the Java API: tricky, but you would learn a lot.

f) (9 points) Add an obstacle (a simple box) that appears at the push of a button, and which destroys the car when the car hits it, or which the car can bounce off of.

g) (9 points) Add UI options for *two* additional, separate controllable car features that can then appear or disappear, such as roof, radio antenna, fancy wheels, supercharger hood, a trailer, a driver, etc.

h) (9 points) Add a slider that affects just the size of the wheels or some other *individual part* of the car, and the rest of the car adjusts to accommodate it as necessary. For example, if the wheels are bigger, the car should be taller but not necessarily longer.

i) (9 points) Add weather (rain, snow, wind, etc.) whose intensity is controllable by a slider. It is up to you to decide how to display "weather".

j) (9 points) Modify the apparent geometry of the frame to make the window appear infinite, by making the car get progressively smaller as it approaches the border, so that it never really reaches it. This is not as hard as it sounds. This is called a "hyperbolic space", and is sometimes used to display the contents of large databases. Look up Escher's woodcuts in his "Circle Limit" series to get an idea.

k) (9 points) Add terrain that is not level that the car follows--a simple hill, for example. Alternatively, provide a slider for the car vertical position, turning this assignment into a video game. The user then has to keep the car on the fixed hilly road as the car traverses left to right, and the car even changes color automatically if it is too high or too low, as a warning to the user.

General Notes:

For each step, design the system using whatever CRC, UML, or other technology you find helpful; we do not require you to submit the results of these tools, but you should find that them make the Javadoc easier. Then, write the documentation and the system, and text edit its classes into files. Compile them, execute them on your own test data, and debug them if necessary. When you are ready, submit a hard copy of the text of the code and your testing (use appletviewer or some other screen capture tool), and whatever additional documentation is required; submit an electronic copy of the your class and javadoc files. Make sure that the source code is clear and its user interface is comprehensive and informative.

Write your classes clearly, with a large block comment at the beginning describing what the class does, how it does it (i.e. in some form, talk about its CRC and UML), and how it will be tested. Describe the test data in the comment, also. Document each constructor and method, using javadoc tags and other commentary as necessary. Clear programming style will account for a substantial portion of your grade for the programming part of this assignment.

Checklist for submission:

For theory: Hard copy, handed in to Prof or TA by the beginning of class. No extra points for using a text editor. Neatly stapled! Name and UNI attached.

For programming: Hard copy, handed in to Prof or TA by the beginning of class. All code, all testing runs (cut and pasted from console, and/or captured screenshots). No hard copy javadoc is required this time. Neatly stapled, separately from the theory! Name and UNI attached.

Also for programming: Soft copy, submitted to the Course Files Shared Folder on courseworks by the beginning of class, in a tarball created by CUIT Unix tar command ("tar -cvzf uni_HW1.tar.gz whateverMyDirectoryIs"). *Please use "uni_HW3" as the name of your Courseworks submission.* Include softcopy of javadoc html. Name and UNI should be included as comments in every class. The code must compile. The last submission before deadline is the official one that will be executed if needed.