COMS W1007x: Object-Oriented Programming & Design (Java)
Fall, 2011
Assignment 5: Due Tuesday, December 13, 5:00pm, 622 Schapiro CEPSR

Part 1: Theory (36 points)

The following problems are worth the points indicated. They are generally based on the book's exercises at the ends of Chapters 7 and 8, under the section heading "Exercises". They cover issues in the class structure of Java (in Chapter 7) and frameworks (in Chapter 8).

"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) (4 points) Exercise 7.4. Write a paper program, which can be about two lines executable lines long. But make sure you also answer the question.

2) (4 points) Exercise 7.6. The answers can be a single line each.

3) (8 points) Exercise 7.12. Write a paper program.

4) (8 points) Exercise 7.18. Write a paper program, using Implementation #1 on pages 100-104.

5) (8 points) Exercise 8.6. Write a paper program. See pages 124-125 and 328-330 for the code for two existing versions.

6) (4 points) Exercise 8.10.

Part 2: Programming (64 points)

This assignment is intended to explore the applet framework.

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 electronic 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 some test examples: make sure you turn in some screen shots to show the functionality of your system.

Step 1 (15 points): Getting started. Implement the code in section 8.2, pages 323 to 324. The source code can be downloaded from the book's on-line companion. Make sure you properly attribute the source of this code! You will also need to create the appropriate HTML file, modeled on the HTML code on page 322. Verify that the start and stop methods actually affect the timer (although you do not have to prove this by submitting any physical output).

Step 2 (15 points): Using Step 1, replace the single message of Step 1 with two messages, each with its own string, font, fontsize, but the time delay should be in common to both. Test this with two strings of very different lengths; the shorter one should "chase" and "overtake" the longer one (why?). If you do this

Step, you do not have to submit separate output from Step 1.

Step 3 (15 points): Using Step 2, allow the two messages to be given each their own speeds (that is, delays). These delays can now be negative or positive (zero doesn't make sense: why?), and you will therefore also have to handle scrolling in either direction. There are several ways to handle unequal and possibly oppositely signed speeds (does it matter which string is drawn first?); justify your choice. If you do this Step, you do not have to submit separate output from Step 1 or Step 2.

Step 4 (19 points): Creativity Step. Handle collisions in a physically realistic way. Let each string have a "mass" that is equal to its fontsize (its "height") times its length: that is, mass is roughly proportional to the number of pixels it contains. Use the laws of physics to recompute the velocity vectors after you detect a collision. (A good short introduction and the simple equations can be found in en.wikipedia.org/wiki/Momentum, under "Elastic collisions".) Test the program at least twice, changing only the HTML file. The first should show a collision between two shapes with equal mass: the velocities should be exchanged. The second should show a collision between two shapes with very different masses, with a tiny string (short and short of stature) impacting a much bigger (long and high) string; the tiny string should be greatly affected, but even the big string should show some effect, and after a while it should settle into a specific pattern. It is best to do these with strings that start off completely isolated from each other. If you do this Step, you do not have to submit separate output from the prior Steps.

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, electronically submit the text of the code, a copy of the Javadoc output, a copy of your testing (including screen shots), and whatever additional documentation is required. 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. Show some of the test data in the comment, also. Document each constructor and method. 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 by the due date and time. No extra points for using a text editor. Neatly stapled, separately from programming hard copy! Name and UNI attached.

For programming: Hard copy, handed in by the due date and time. All code, all testing runs (cut and pasted from console, and/or captured screenshots). Neatly stapled, separately from the theory hard copy! Name and UNI attached.

Also for programming: Soft copy, submitted to the Course Files Shared Folder on courseworks by the due date and time, in a tarball created by CUIT Unix tar command ("tar -cvzf uni_HW5.tar.gz whateverMyDirectoryIs"). Includes softcopy of javadoc html. Name and UNI should be included as comments in every class. The code must compile. Last electronic submission before deadline is the official one that will be executed if needed.