COMS W1007x: Honors Intro to Computer Science
Fall, 2012
Assignment 4: Due Tuesday, November 27, 2012, 1:10pm, in class

Part 1: Theory (35 points)

The following problems are worth the points indicated. They are generally based on the book's exercises at the ends of Chapters 8 and 9, under the section heading "Exercises" and "Programming Projects". They cover concepts involving arrays, mouse and key events, inheritance, and timers.

"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 Part in general: clear handwriting is sufficient, and computer-generated documents will earn no extra credit.

1) (7 points) Based on EX 8.10: Write a paper method that takes two arrays of the same type, arrayReceive and arraySend, and which copies all the values in arraySend into arrayReceive, starting at location arrayReceive[start]. What should you do if there is not enough room?

2) (7 points) Based on PP 8.12 and Listing 8.16: Write a paper method that draws an equilateral triangle with a horizontal base by using fillPolygon(). The method should accept a size parameter and the coordinates of its highest point.

3) (7 points) Based on Ex 9.5 through 9.7: Draw a UML class diagram for the hierarchy of sports, and show at least one field and one method for at least two separate classes. In particular, these two classes must be at different levels in the hierarchy, so that the fields and classes of the superclass still are valid for the subclass, but that the reverse is not true.

4) (7 points) Based on PP 9.3: Assume you have a class ReadingMaterial with methods getPages(), countWords(), turnPage(). Write the class and at least two method headers (no other code) for a subclass ComicBook which has illustrations with word bubbles, for a subclass Statistics which has tables and formulae, and PopUpBook which has three-dimensional activities but might not have numbered pages.

5) (7 points) Based on the Programming Part below: Assuming you have completed all the Steps, give a paragraph saying what you would have to change at each Step to do a realistic *three* dimensional simulation.

Part 2: Programming (65 points)

This assignment is intended to exercise some the more advanced concepts in arrays, mouse and key events, and timers. It is based on Listings 9.15 and 9.16, except we introduce additional different physical objects that follow the laws of physics.

The goal of this assignment is not only to exercise some of the more complex GUIs of Java, but also to test the limits of physical simulations. One of the purposes here is to see how accurate they can be made, and how much you can push them to their limits before they break. Some of the book's code in the Online Companion can be useful, but in this assignment a good deal of this design will be on your own.

Please recall that all programs must compile, so keep a working version of each Step before your proceed to the next.

Step 1 (5 points): Getting started.

Implement 9.15 and 9.16. Note that on page 478 is says: "The goal should be to create the illusion of movement that is pleasing to the eye." So, you will have to experiment, and tune your constants, and then document your choices of constants within your javadoc. You can pick whatever gif you want, but since this is a 2D simulation, an image of a something flat and round, like a dime or a smiley face or even just a colored dot, is a good choice. Note that the object always starts in the same place and goes in the same direction.

Step 2 (15 points): Adding more points via mouse control.

Once your Step 1 it is running and tuned, add to it a mouse listener like the one in Listing 8.18. This will serve to indicate where to place some new objects. A careful reading of Listing 8.18 shows that none of the special properties of ArrayList are used, so instead use just an array to hold the references to the objects. Note that this is a requirement for this Step, and you will lose points if you continue to use ArrayList! Direction can still be still hard-coded ("moveX" and "moveY"), objects are all of the same icon, and they pass through each other, but they all still bounce off the walls.

Step 3 (15 points): Adding direction via keyboard control.

Modify Step 2 by using something like the keyboard listener in Listing 8.22 to indicate the direction that the next created object will take. This direction value is stored until it is changed. Use the keyboard so that the three-by-three keyboard block of "wersdfxcv" to indicate motion in the eight compass directions of NW N NE, W o E, and SW S SE, respectively, where the center key of "d" means no motion at all. Since this is finite and culturally determined, you can use a switch if you want. So, at any time, any of these keys can be pressed, and it becomes the direction for all subsequent mouse clicks. You don’t have to display these key clicks to the user in any way.

Step 4 (10 points): Add the physics of elastic collisions.

For a quick introduction (and an interesting demo), see:

`http://en.wikipedia.org/wiki/Elastic_collision`

Still assume all masses are equal. How do you detect collisions between any two objects? In general, this takes N^2 tests if there are N objects. Experiment to see two important things. Do you detect all collisions--or do some objects appear still to pass through each other? And, how many objects can you get moving and still give a realistic appearance? You need to document your design decisions and parameters for this Step.

Step d54 (20 points): Creativity step: add inheritance.

The three-by-three keyboard block of "wersdfxcv" still controls the direction of some new objects, and these new object still have a fixed mass and a fixed icon. But the next three-by-three block of "uiojklm,." specifies that the next object to be placed by the mouse has a larger mass and a different icon (say, an American quarter); relative directions are again given by compass direction with respect to the center key of "k".

So far, this does not require inheritance--you can just use fields--but the following does: give it the extra ability (that is, make it a subclass) so that it can change its icon on demand. In particular, every time this larger object is collided with, it shows this graphically in some way; the smaller objects do not. You must use inheritance in this Step, and you will lose points if you just create a single but more powerful class that both kinds of objects are instances of.

Note that you should review the wikipedia page again to make sure that you accommodate the two different masses involved. If you structured your collision detection algorithm properly, this should just change a few lines of code.

General Notes:

The general rules for the previous assignments still apply: If you do a Step you do not have to submit separate output from the previous Steps. Outputs here consist of the usual hard and soft copy of code, softcopy of javadoc, and enough screen shots to show that the system does in fact demonstrate the functionality of the Steps you attempted.

For each Step, design and document the system, text edit its components into files, compile them, debug them, and execute them on your own test data. When you are ready, electronically submit the text of its code, a softcopy of your Javadoc output, a copy of your testing (including any screenshots), and whatever additional documentation is required. Make sure that the source code is clear and any user interface is comprehensive and informative.

Make sure you have properly attributed any code that is legal to incorporate from other sources, including the book's online companion.

Write your classes clearly, with a large block comment at the beginning describing what the class does, how it does it, and justify how it will be tested. Put the comments about testing in a Tester class; if possible, show some test data in the comment also. Document each constructor and method.

Clear programming style will account for a substantial portion of your grade for both the Theory Part and the Programming Part of this assignment. For one reasonable set of suggested practices and stylistic guidelines, see the book's Appendix F. But it is far more important to be consistent: PoLS, everybody!

Checklist:

Here is a checklist for submission. Please note that this is designed to be a *general* guide, and you are responsible for *all* things asked for in the text of the assignment above, whether or not they appear below.

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, separately from programming hard copy! 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), all Javadoc if hardcopy Javadoc is required in hardcopy (for this assignment, it is not). Neatly stapled, and separately stapled from the theory! Name and UNI attached.

Also for programming: Soft copy, submitted to the Assignments page of Courseworks by the beginning of class, in a tarball created by CUIT Unix tar command, given below. Note that "myUNI" should be replaced with your UNI. *Please* use this convention as it makes the TAs' job much easier:

```
tar -cvzf myUNI_HW4.tar.gz whateverMyDirectoryIs
```

Also for programming: Include the softcopy of Javadoc html. Name and UNI included as comments in *every* class. The code must compile. Your last electronic submission before deadline is the official one that will be executed if needed.