CPSC 411, Spring 2006 – Problem Set 1.

For this problem set you will work alone. You may consult fellow classmates in figuring out how to do the problems, but the work you actually hand in should be your's alone.

The main purpose of this assignment is to help you learn how to work with AspectJ.

The homework is due at 11pm January 23rd. No late homework will be accepted. You will use the handin program to turn in the assignment. Handin instructions will be posted to the newsgroup. You should put your name and student number at the top of the first page.

A number of the questions ask you to hand in source code. Please make sure the source code is properly indented and set in a fixed pitch font (courier).

Before beginning this problem set you should get the Problem Set 1 download from the course web site. You will need to import that into Eclipse, in the same way you imported the first compiler drop.

Spend some time using the navigation features of AspectJ. Use the gutter annotations to go from methods to advice. Go to Window>Show View>Other to open the Visualizer and the Visualizer menu. Use that to see all the places in the code (technically called join point shadows) where advice might run.

Problem 1

Once you have the project imported into Eclipse, it should compile properly.

Now run the DriverA file as an AspectJ/Java application. Look at the output log.

How many display updates result from a top-level call to Point.setX? Point.setY? Line.moveBy? Circle.moveBy? These four numbers are the answer to problem 1. (Write them down now, because they will change in later problems!)

Problem 2

The code has an empty DisplayUpdating aspect in the display package.

Refactor the code to eliminate the scattered calls to change() that appear throughout the Shape classes. All those calls should now be in the DisplayUpdating aspect.

As part of the refactoring you should remove the old calls.

For the solution to this problem hand in the source code for your DisplayUpdating aspect.

Problem 3

There is a lot of behavior in the Shape class that corresponds to the typical Observer pattern. Now further refactor the code to move that behavior to the DisplayUpdating aspect.

For the solution to this problem hand in the source code for your revised DisplayUpdating aspect.

Problem 4

Which solution do you prefer – the one from Problem 3 or Problem 4? Provide a well grounded rationale. You may want to use the kind of “refers to” picture used in lecture to make your case.

Problem 5

The ‘legacy’ code you downloaded had some other modularity problems as well. In particular note that the x and y fields of the Point class were not private. Make them private. In doing so, you will discover that you have to fix more than just the Point class.

For the solution to this problem hand in the source code for the moveBy methods of Point, Line and Circle.

Problem 6

Now re-run the Driver A file, and compare the number of display updates for top-level calls to the three moveBy methods. What has changed and why?

By editing only the DisplayUpdating aspect, fix your code so that top level calls to any of the change methods result in only a single display update. Include the in the source code for your revised DisplayUpdating aspect in your answer to this problem.