**CS 220 – Data Abstraction**

**PEX 1 – Virtual Aquarium**

**Prelim 1 Due: 2200, Lesson 6, Thursday, 19 January**

**PEX1 Due: 2200, Lesson 10, Tuesday, 31 January**

|  |
| --- |
| Help Policy: **AUTHORIZED RESOURCES:** Any, except another cadet’s program.  **NOTE:**   * Never copy another person’s work and submit it as your own. * Do not jointly create a program unless explicitly allowed. * You must document all help received from sources other than your instructor. * **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.**  Documentation Policy:  * You must document all help received from any source other than your instructor. * The documentation statement must explicitly describe WHAT assistance was provided, WHERE on the assignment the assistance was provided, and WHO provided the assistance. * If no help was received on this assignment, the documentation statement must state “NONE.” * If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.  Vague documentation statements must be corrected before the assignment will be graded, and will result in a 5% deduction on the assignment.Turn-in Policies:  * On-time turn-in is at 2200 on the due date, same day for both M and T day sections. * Late turn-ins will receive a 25% penalty per 24 hours late unless prior arrangements have been made with your instructor. * There is no early turn-in bonus or extra credit for this assignment. |

1. Objectives

* Be able to design a multi-level class hierarchy
* Be able to read, understand, and modify a significant amount of code written by another programmer
* Be able to refactor existing code to minimize repeated code using inheritance and polymorphism
* Be able to document code using Javadoc

1. Background

This programming exercise implements a virtual aquarium where aquatic creatures swim or crawl on the screen with different movements and different behaviors when meeting another creature. There are six different kinds of aquatic creatures, Angelfish, Clownfish, Crabs, Lobsters, Seahorses, and Snails. The table below summarizes the movements and behaviors of each of these creatures:

|  |  |  |
| --- | --- | --- |
| Creature | Movement | Behavior When Meeting Another Creature |
| Angelfish | Swim horizontally across the upper ¾ of the aquarium, always moving at a slight angle either upward or downward. | Reverses direction. |
| Clownfish | Swim straight across the upper ¾ of the aquarium. | Swims at an angle for a short time, downward if moving left-to-right and upward if moving right-to-left. |
| Crab | Crawl straight forward and backward in the bottom ¼ of the aquarium, randomly pausing for a short period. | Scurry away quickly, moving horizontally at twice their normal speed for a short period. |
| Lobster | Crawl straight forward and backward in the bottom ¼ of the aquarium. Each time they move forward or backward, they also move slightly left or right with a probability of 0.25. Thus, half the time they only move forward or backward and not horizontally. | When a lobster meets another creature, it pauses for a short period. |
| Seahorse | Swim horizontally across the upper ¾ of the aquarium, also moving vertically in an up-and-down, zig-zag motion. | Ignores all other creatures. |
| Snail | Crawl straight forward and backward in the bottom ¼ of the aquarium. | Ignores all other creatures. |

The creatures in the aquarium move in three dimensions and each has three coordinates, x, y, and z. The creatures are sorted so those toward the back of the aquarium are drawn earlier and those toward the front of the aquarium are drawn later. This gives the illusion of depth (visual depth front-to-back, not water depth top-to-bottom) when creatures closer to the front overlap creatures closer to the back.

For this assignment you will be given a fully functional implementation of the virtual aquarium with one problem: Each type of creature is a separate class stored in a separate array which makes sorting all of the creatures in the aquarium a very difficult task. That is, sorting angelfish with respect to angelfish and clownfish with respect to clownfish is easy, but sorting angelfish with respect to clownfish when they are stored in separate arrays is not so easy.

Your task for this assignment will be to design and implement a multi-level class hierarchy that minimizes redundant code using inheritance and polymorphism.

1. Preliminary Exercise

For the preliminary assignment you will begin by reading the provided code for the six creature classes and identifying common and/or redundant code that could be refactored into base classes. From this you will design your class hierarchy and create a UML diagram describing your hierarchy. This diagram must include all base and derived classes you determine are necessary. You do not need to include any classes from the Java library on your UML diagram. All class representations on the diagram must also include all class methods and attributes. Be sure to use correct connecting lines to indicate has-a (uses) and is-a (inheritance) relationships. Finally, ensure you include the class containing the main method on the diagram. *You must draw your UML diagram using the Dia tool discussed in class*. No diagrams created with other software or hand-drawn diagrams will be accepted.

In the top-level NetBeans project folder is a Word document named **Classes.docx** with all of the provided source code. Your task with this document is to identify and highlight portions of code as follows:

Green indicates code and variables that should be refactored into a base class.

Yellow indicates code and variables unique to a single class.

Gray indicates code and variables that will need modification after the refactoring.

Also in the top-level NetBeans project folder is a word document named **Application.docx** with the source code for the main application. Your task with this document is to identify and highlight portions of code as follows:

Green indicates code and variables that are redundant and should be deleted.

Yellow indicates code and variables that must be retained.

Gray indicates code and variables that will need modification after the refactoring.

1. Preliminary Submission Requirements

* Fill in your name, the documentation statement, and self-assessment in the prelim grade sheet file in the top level of the downloaded NetBeans project folder!
* Using the Webpost link on the left side of the course web page, submit the Prelim Grade Sheet, Classes, and Application Word documents and the Dia file containing your UML diagram.
* There will be four files submitted for the preliminary assignment, none of them zipped.

1. Programming Exercise

For the programming assignment you will implement your multi-level class hierarchy from your preliminary design. You will also make substantial modifications to the main aquarium application to take full advantage of your new design. Your finished application should run and look like the solution demonstrated in class. In particular, all creatures in the aquarium should be sorted and painted from back to front. Work hard to ensure you turn in a running version of the aquarium.

1. Programming Exercise Submission Requirements

* Fill in your name, the documentation statement, and self-assessment in the pex grade sheet file in the top level of the downloaded NetBeans project folder!
* Your NetBeans project name must be LastnamePEX1.
* Zip this entire folder to LastnamePEX1.zip and submit it using the Webpost link on the left side of the course web page.
* Submit only one file for the programming exercise, a zip file containing your entire NetBeans project with the PEX Grade Sheet document in the top-level folder.

1. Helpful hints

NetBeans has a Refactor menu with several tools that may prove to be useful in completing this exercise.

To test that various creatures are still behaving the way they should, put creatures in specific places moving in specific directions. That is, in the main application after the creatures are created they are all in random locations. Use the mutator methods to set their location and speed to ensure two creatures meet, for example.

**CS 220 – PEX 1 Prelim – Grade Sheet Name: Gavin Delphia**

Assessments

Criteria Self Instructor Points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UML diagram describes a reasonable class hierarchy for this problem | | **10** |  | **10** |
| UML diagram contains all required information in the correct format | | **5** |  | **5** |
| Code to be refactored is adequately identified in Word document | | **5** |  | **5** |
| Code unique to one class is adequately identified in Word document | | **5** |  | **5** |
| Code needing modification adequately identified in Word document | | **5** |  | **5** |
| **Subtotal:** | | **30** |  | **30** |
| **Adjustments** | **Vague/Missing Documentation:** | **−0** | **−** | **− 2** |
| **Submission Requirements Not Followed:** | **−0** | **−** | **− 2** |
| **Late Penalties:** | **−0** | **−** | **25/50/75%** |
| **Total w/adjustments:** | **30** |  |  |

Documentation Statement:

I did not receive help on this assignment.

Comments from Instructor:

**CS 220 – PEX 1 – Grade Sheet Name:**

Assessments

Criteria Self Instructor Points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| All aquatic creatures move and behave correctly (3 points each) | |  |  | **18** |
| Aquarium application properly creates and stores various creatures | |  |  | **6** |
| Aquarium application properly moves all creatures | |  |  | **6** |
| Aquarium application properly determines when creatures meet | |  |  | **6** |
| Aquarium application properly sorts all creatures | |  |  | **6** |
| Abstract classes and methods used properly | |  |  | **6** |
| Redundant code among creatures that swim is minimized | |  |  | **12** |
| Redundant code among creatures that crawl is minimized | |  |  | **12** |
| Comments accurately describe what the code does | |  |  | **8** |
| **Subtotal:** | |  |  | **80** |
| **Adjustments** | **All Java code meets specified standards:** | **−** | **−** | **− 8** |
| **Vague/Missing Documentation:** | **−** | **−** | **− 4** |
| **Submission Requirements Not Followed:** | **−** | **−** | **− 4** |
| **Late Penalties:** | **−** | **−** | **25/50/75%** |
| **Total w/adjustments:** |  |  |  |

Documentation Statement:

Comments from Instructor: