CSCI 3353 Object Oriented Design
Homework Assignment 6
Due Monday, October 31

I have written an animated fish tank for you to enjoy, which is in the file *fishtank.zip*. The zip file contains the necessary java files, as well as a folder containing image files of fish. In the folder are two images for several kinds of fish – one facing left, and one facing right.

The first thing you should do is get the code to work on your computer. (Note that the image folder needs to be placed, as is, at the top level of your project.) To add a new fish to the tank, click on the "New Fish" menu; to start and stop the animation, click on the "Animation" menu item.

The program is a lot of fun, but the fish are kind of boring. They all swim the same way, and at the same speed. In this assignment you will use factories to customize how the fish are created.

1. The class *Fish* is responsible for fish movement. In particular, the values of the private variables *xspeed*, *yspeed*, *xDirectionChangeFreq* and *yDirectionChangeFreq* determine how a fish will move. A set of values for these variables can be thought of as a *movement style*. You should use the abstract factory pattern to create a *MovementStyle* interface and some implementing classes. You are free to be creative about what style classes you write, but at least supply the following:

- The class *HorizontalMovement*, in which fish move horizontally in the tank. Its constructor takes an argument that denotes the horizontal speed of the fish.
- The class *UpDownMovement*, in which fish move vertically as well as horizontally. Its constructor takes two arguments that denote the horizontal and vertical speed of the fish.
- The class *OscillatingMovement*, which behaves the same as *UpDownMovement*, except that the variables *xDirectionChangeFreq* and *yDirectionChangeFreq* have high values (a value of about 75 works for me), which cause the fish to wiggle back and forth.

The constructor to the *Fish* class will take a *MovementStyle* argument, as is typical for the abstract factory pattern.

2. The *newFish* method in the *FishMenuBar* class contains the code for creating *Fish* objects. You need to modify this code in accordance with the basic factory pattern. In particular, you should create a hierarchy of factory classes, parallel to the hierarchy of *Fish* classes, and headed by the abstract class *FishFactory*. Each factory class should have a method *create* (with no arguments) that creates an instance of its associated *Fish* class.

The *create* method of each factory class determines the movement style of the *Fish* objects it creates. Feel free to be creative about what each factory class does. My only requirement is that each factory class creates its fish having a different style. For example, my *AngelFishFactory* class creates its fish with an *UpDownMovement* style, having random speeds.

3. The reason that *FishFactory* is an abstract class is that I want you to also write a static factory method for it. The method is called *getFactory*, and it should look like this:

```
public static FishFactory getFactory(String fishname) {
     ...
}
```

The *newFish* method in *FishMenuBar* should call this method to get the appropriate fish factory, and then use the factory to create the appropriate *Fish* object. In a sense, this method is a "fish factory factory".

When you are done, create a zip file containing all of the Java files in your program (without package statements). In addition, please write a document containing the following information:

- A description of anything you did that went beyond what was required.
- A class diagram showing the relationships between all of the classes in the program.
- The answer to the following questions. If you wanted to change the program to accommodate another type of fish:
    - What new Java classes would you need to create?
    - What existing Java classes would you need to modify, and what would the modifications be?

Submit the zip file and the document to Canvas.