**The Aquarium（水族馆） Lab Series**

The Aquarium Lab Series is a series of mixed-length laboratory exercises that provide students with an opportunity to put introductory programming concepts to work immediately.

Unzip [AquariumLabSeries.zip](http://max.cs.kzoo.edu/patterns/AquariumLabSeries.zip) file, it will create a folder called AquariumLabSeries  that contains the instructions for the labs, a folder called StartingCode, and a folder containing documentation.  The StartingCode folder contains three classes you will viewing and modifying, a sub-folder with other "black-box classes" needed by the program (that you do not need to study), and a graphics library.

**Labs in the Aquarium Lab Series**

The Aquarium Lab Series consists of several mini-labs, which are meant to be short enough to complete in a single class period, and a longer lab, which may need a longer time-block or several class periods to complete.

**Mini-Lab: [Reading Specifications and Constructing Objects](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\ObjConstLabs.shtml)**

Tasks include:

* Read several class specifications.
* Write client(客户端)code to construct objects and invoke(引用) member functions:
  + Modify the main function to construct three fish in the aquarium.
  + Display the fish in the aquarium.
  + Move all three fish one space.
  + Redisplay the fish in the aquarium.

**Exercise 1**

* Read the initial version of the main method in AquaSimApplication.java. This method has three sections. The first constructs the objects needed to run the simulation. The second actually runs it (or will, when we have added some more functionality to it). The third section wraps up the program, reminding the user how to quit.
* Research the abbreviated(简称) description of the [AquaFish](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaFish.html) class to discover how to construct a fish. Edit the main method in AquaSimApplication.java to declare and construct three fish variables at the end of the first section.
* Test your modified program. Does it look any different? You constructed three fish, but the userInterface is only drawing the aquarium, not the fish.

**Exercise 2**

* Research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to view a fish. Modify the main method to display the three fish you constructed. You want to draw the fish *after* drawing the aquarium (or else the blue water in the aquarium will hide the fish), but *before* repainting the user interface. Test your modified program.
* Now it is time to start running the aquarium simulation. Research the abbreviated specification for the [AquaFish](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaFish.html) class to discover how to make a fish move forward. Add statements to the main method to move your three fish forward once. (Where should these statements be added?)
* Test your modified program. If you don't see the fish move, you probably forgot to repaint the user interface to make the changes visible. You may also see two copies (or one "blurred" copy) of each fish -- the fish in its old location and in its new location. If you see this, you forgot to redraw the aquarium to "erase(删除)" (or cover up) all the fish in their old locations before displaying the fish in their new locations. Add a line to the beginning of the second block of display statements to redraw the aquarium. You may even occasionally only see two fish (or even one!). This can happen when a larger fish overlaps and hides a smaller fish. Run your program a number of times to verify that you usually see three fish.
* Now when you run your program you should no longer have two copies of each fish, but you probably can't really see the fish move because it happens too quickly to see. Research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to ask the user interface to pause after each set of display statements so that you can see the fish before and after they move.

**Mini-Lab: [If Statements](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\ConditionalLabs.shtml)**

Tasks include:

* Have each of the three fish change direction instead of getting stuck at the aquarium walls.
* Use a random number generator to give fish random colors.

**Exercise 3**

* In your previous testing of the program you may or may not have noticed that fish swim only to the right and get stuck at the right wall. To verify the problem, make a copy of the statement that sets the dimensions of the aquarium. "Comment out" the original, and change the dimensions in the copy to be 100 x 480. Copy the code that moves and redisplays the fish to let them move a second time. Now run the program several times and make sure you see the problem.
* Research the abbreviated [AquaFish](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaFish.html) specification to find out how to determine whether a fish is at a wall and how to make it turn around. Using the [Whether or Not](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\ConditionalLabs.shtml" \l "aa) Selection pattern, modify your main method to have each fish reverse direction if it is at a wall. Then move the fish forward, whether it reversed itself or not. Test your program in the narrower aquarium you created above. When you are satisfied that your program is behaving correctly, restore the aquarium to its original size.

**Exercise 4**

* Research the specification for the [Random](http://java.sun.com/j2se/1.3/docs/api/java/util/Random.html) class to discover how to construct a random number generator. A Random object can be used to simulate a coin toss or other random selection. Edit your main function to construct a random number generator right before you construct your three fish.
* Create a new variable that will be a reference to an object of the Color class, but don't actually construct or initialize a Color object. Give your variable an **Intention Revealing Name**. For example, your variable declaration might look like the following line.

Color fishColor;

* Research the [Random](http://java.sun.com/j2se/1.3/docs/api/java/util/Random.html) specification to discover how to randomly generate one of two values (0 or 1). Before you construct each fish, generate a random 0 or 1 first. Using the [Alternative Action](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\ConditionalLabs.shtml" \l "aa) Selection pattern, set your Color variable to Color.red if the random number is 0; set it to Color.blue otherwise. (Color.red and Color.blue are constant Color values defined in the Java Color class.)
* Research the abbreviated [AquaFish](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaFish.html) specification to discover how to specify the color of a fish as you create it. Modify your code to specify the color of each fish using your Color variable.

**Exercise 5**

* Modify your program to create fish with the colors of the rainbow. This time, before you construct each fish, generate a random integer less than 6 (in the range of 0 to 5) first. Using the [Sequential Choice](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\ConditionalLabs.shtml" \l "sc) selection pattern, set your Color variable according to the following table:

|  |  |
| --- | --- |
| **Random Integer** | **Color** |
| 0 | Color.red |
| 1 | Color.orange |
| 2 | Color.yellow |
| 3 | Color.green |
| 4 | Color.blue |
| 5 | Color.magenta |

* Do not generate a new random number for every test; just generate one random number for each fish and test it against the various integer values. You may want to use cut and paste to reinitialize the Color variable for the second and third fish! (This table uses Color.magenta because Color.indigo and Color.violet are not defined colors.)
* Test your program to make sure that your results are what you expect. (What results were you expecting? What tests are necessary to make sure the results are what you expect?)

**Mini-Lab: [For Loops](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\RepetitionLabs.shtml)**

Tasks include:

* Prompt(提示) the user for how many time steps the simulation should run.
* Modify fish movement so that in each time step a fish randomly chooses whether to change direction (unless it would otherwise swim right through a wall, in which case it always changes direction).

**Exercise 6: Simulate Fish Moving Forward**

* Modify the main method to make your program become a simulation of three fish moving in the aquarium over time. Initially, set the number of time steps in the simulation to 10. Choose one of the loop control idioms above and use it correctly. Make sure that your loop body includes moving and displaying the fish (and pausing the animation long enough for you to see it).

#### Exercise 7: Allow User to Control Simulation Steps

* The AquaSimGUI class, which provides the graphical user interface for the Aquarium Lab Series, has a second constructor that allows you to specify whether or not to prompt for the number of simulation steps to run. Research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to use this constructor. Modify your program to indicate that the graphical user interface should prompt the user for the number of simulation steps.
* Research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to ask the graphical user interface for the number of simulation steps specified by the user. Store the result in a local variable with an **Intention Revealing Name**. Then use that number to control your loop.

**Exercise 8: Fish Doing Their Own Thing**

* Review your use of the [Random](http://java.sun.com/j2se/1.3/docs/api/java/util/Random.html) class to determine how to calculate a one in four chance of changing direction for any given fish. Modify your main method to introduce a more complex check: each fish should change direction if it is up against a wall or if it randomly chooses to change direction.

Note: Use the same Random object you constructed earlier to influence fish color.

* Test your modified program.

**Mini-Lab: [Java Arrays](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\MoreFish.shtml)**

Tasks include:

* Prompt the user for how many fish they want in the aquarium.
* Construct an array, specifying the size.
* Insert fish into the array.
* Traverse the array to move and display all the fish in each time step.
* (hint: We do not want the first fish to move 100 times, followed by the second fish moving 100 times, followed by the third fish moving 100 times. Instead, we want all 25 fish to move once, then all 25 fish to move again.)

**Exercise 9**

* The AquaSimGUI class has a third constructor that lets you specify whether or not to prompt the user for the number of fish to put in the aquarium. Research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to use the third constructor. Modify your program to indicate that the graphical user interface should prompt the user for the number of fish as well as the number of simulation steps.
* Now research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to ask the graphical user interface for the number of fish specified by the user. Store the result in a local variable with an **Intention Revealing Name**.

**Exercise 10**

* Step through your array to put newly constructed fish in it, even though the aquarium can now handle more fish than before, your code should be getting considerably shorter!
* Remove the remaining comments around the code that runs through the steps of the simulation, moving and displaying the fish in the aquarium.
* Inside the simulation loop, replace the code that moves the three named fish with a loop that will move all the fish in your collection. (Each will still change direction when it has to or when it randomly chooses to.)

**Full-Length Lab: [Implementing Classes](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\ImplementingClasses.shtml)**

Tasks include:

* Modify the AquaFish class to provide methods that allow fish to ascend and descend in the aquarium.
* Move the logic of deciding how fish move from the main method to the AquaFish class.
* Flesh out a skeleton Simulation class by implementing the constructor and a step method that encapsulates the behavior for a single step through the simulation.

**Exercise 11: Simulation Up and Down Movement**

### Introduction

Our program would be much more interesting if the fish moved up and down as well as side to side. In this exercise, you will implement two new methods in the AquaFish class, ascend and descend, to support this behavior.

* To make the simulation more believable, the distance that a fish moves up or down should be related to its height. Fish come in different sizes, so the size of any particular fish is one of the properties of that fish. Its position in the aquarium is another relevant property for this exercise. Read the [implementation (code) for the AquaFish class](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\StartingCode\\AquaFish.java) to determine which methods or instance variables will be useful in implementing ascend and descend.
* Determine what parameters (if any) you will need for the new ascend method. Then determine what its return type should be. Add an empty ascend method (one that consists of a declaration and empty braces) to the AquaFish class after the moveForward and changeDir methods.
* Implement the ascend method. You may use the moveForward method as a guide if you like, but the ascend method is simpler. The movement amount should simply be the height of the fish. Research the specification for the [AquaPoint](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaPoint.html) class to discover what methods might be useful in implementing the ascend and descend methods.
* Implement the descend method.
* Modify your main method in the AquaSimApplication class to allow fish to ascend or descend before moving forward, according to the following formula:
  + A fish at the surface has a 2/3 chance of descending and a 1/3 chance of staying at the surface.
  + A fish at the bottom has a 1/3 chance of ascending and a 2/3 chance of staying at the bottom.
  + A fish that is neither at the surface nor at the bottom has a 1/3 chance of ascending, a 1/3 chance of descending, and a 1/3 chance of staying at the same depth.

Read the [implementation (code) for the AquaFish class](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\StartingCode\\AquaFish.java) to determine what methods are available to tell whether a fish is at the surface, at the bottom, or somewhere in between.  (These methods do not appear in the partial specification for the AquaFish class that you have used previously.)

* Test your program.

**Exercise 11: Make Fish Responsible for Knowing How to Move**

### Introduction

One of the most important tasks in designing and implementing object-oriented programs is deciding which classes or objects are responsible for executing which behavior. Up until now, all of the behavior that you have added to the Aquarium Simulation program has been in the main method of the AquaSimApplication class. Now it's time to make the program more object-oriented. For example, deciding whether a fish should change direction or ascend or descend when it moves should be the responsibility of the fish, not of the simulation program.

In this exercise, you will implement a move method in the AquaFish class that will encapsulate(封装) all of the behavior related to movement (ascending, descending, changing direction, and moving forward).

* Analyze the main method of the AquaSimApplication class to determine which lines of code should be moved to the move method you will be implementing in the AquaFish class.
* Determine what parameters (if any) you will need for the new move method. Then determine what its return type should be. Add an empty move method (one that consists of a declaration and empty braces) to the AquaFish class before the moveForward method.
* Move the appropriate lines of code from the main method in AquaSimApplication to the move in AquaFish. Notice that the main method was invoking AquaFish methods such as changeDir on a named fish. The move method, though, is tied to the AquaFish class. More specifically, it is always tied to (or executed by) a particular fish. Thus, instead of main invoking methods such as changeDir on a named fish, the fish should now be invoking methods on itself.
* Modify the main method in AquaSimApplication to simply tell each fish to move, letting the fish worry about how it should move.
* Test your program to verify that the behavior is unchanged.

**Exercise 12: Introduce the Simulation Class**

### Introduction

In a well-designed object-oriented program, we usually want the main function to just create some objects and get the ball rolling. Most of the program behavior should be the result of the objects interacting with each other. In the Aquarium Simulation program, though, the main function is actually running the simulation. We have objects that model the fish and the aquarium, but not one that models the simulation itself.

In this exercise, you will write the code for methods in a Simulation class. The constructor will initialize the Simulation object's instance variables and construct the fish in the aquarium. The step method will execute the commands that should happen each timestep in the simulation (moving the fish, in our case).

Once you have implemented the Simulation class, the main method in AquaSimApplication will merely create a number of objects, such as the aquarium, the graphical user interface, and the simulation object, and then ask the simulation object to run the simulation. The main method will also still display the initial configuration of the fish and the modified aquarium after each timestep

* Read [the specification for the Simulation](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\Simulation.html) class to familiarize yourself with what the class should do.
* Download [the incomplete Simulation class implementation](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\StartingCode\\Simulation.java) if you don't have it already.
* Read the incomplete Simulation class implementation and then modify the constructor to initialize the instance variables.

Notice that Simulation objects have three instance variables and that the constructor takes three parameters. Two of the three instance variables represent the same objects as two of the parameters, so you can initialize them directly. For example, the main method in AquaSimApplication will construct an Aquarium and pass it to the Simulation constructor. The Simulation object should refer to the same Aquarium object, not a new one. On the other hand, looking at the parameters to the Simulation constructor, we can see that the main method will not construct an array of fish and pass it to the constructor. Instead, it will pass the number of fish that should be in the array. Therefore, to initialize the allFish instance variable, you will need to construct a new array of fish.

* Move the appropriate lines of code that construct various colored fish from the main method in AquaSimApplication to the Simulation constructor. Double-check whether the instance variables in the Simulation constructor have the same names that the local variables in the main method had. Do not move the code that displays the initial configuration; for now, that code should stay in main.
* Replace the code in the main method that constructed all the fish with a statement that constructs a Simulation object.
* Determine what code in main represents a single timestep in the simulation. Move that code to the step method in Simulation. Adjust any variable names if necessary. Do not move the code that displays the modified aquarium and fish after each timestep; for now, that code should stay in the main method.
* Replace the code in the main method that represented a single timestep in the simulation with a call to the step method in Simulation.
* Test your program to verify that the behavior is unchanged.

**Exercise 13: Allow User to Control Simulation**

* The AquaSimGUI class has a fourth constructor that allows you to specify whether or not to have the graphical user interface interact directly with a Simulation class. Research the abbreviated specification for the [AquaSimGUI](Z:\\gracewu\\gracewu\\my course\\Java\\ExerciseSample\\assignment-examples\\Aqua-code\\AquariumLabSeries\\AquariumLabSeries\\JavaDocs\\AquaSimGUI.html) class to discover how to use this constructor. Modify the main method in AquaSimApplication to indicate that the graphical user interface should use the Simulation class. Once you have done this you no longer need to wait for the user to push the Start button, ask for the number of fish or number of simulation steps, construct the Simulation object, diplay the aquarium, or run the simulation. The graphical user interface will do all of that. Therefore, the construction of the graphical user interface should be the last line of code before the WRAP UP phase.
* Test your program to verify that the behavior is unchanged.