Ocean Life Simulator GUI

1 Introduction

In this assignment you'll be implementing a marine-animal hierarchy to fit into the provided GUI components. When doing the homework, you should not change the provided GUI code except where noted. The main objective of this assignment is to get the GUI provided to work as described while conforming to the guidelines of the hierarchy. Exact implementation may vary.

2 Problem Description

We are providing you with three files:

- Ocean.java
- OceanPanel.java
- ControlPanel.java

You should refer to these files when constructing your hierarchy. The basic requirement is that your hierarchy works with the provided GUI and your overall program runs within the constraints provided (ie population control, etc). To see where objects are instantiated, refer to ClickListener inside the Panel. As you can see, instantiated Fish are passed an x location, a y location, and a Rectangle object. The Rectangle object, bounds, is used to designate the area in which the Fish is confined (in this case, the size of the Ocean). This means that the fish should never be allowed to move outside that area. You only need to modify OceanPanel where it says to in the comments in the code. However, you may add additional helper methods and variables if you so choose. In ControlPanel, you will only need to add more buttons that correspond to the new types of fish that you want to add. You will not need to change the driver class Ocean in any way.

3 Solution Description

Using a well-thought-out OOP design, create a Fish hierarchy. Here are some guidelines:

3.1 Fish Class

Fish should be an abstract class. All Fish should be able to move, collide with other fish, reproduce, eat, or get eaten by other fish! When a fish swims, it should lose health. As time goes on, the fish should get older and eventually die of old age if its lack of health doesn't get to it first. We will be providing pictures (50px) that you may use for the required Fish. However, feel free to find your own or even draw your own.

Disclaimer: The following characteristics for each fish do not necessarily comply with real-life biology.

3.2 Carnivore Class

This class will be an abstract subclass of Fish. Certain Carnivores will be able to eat certain Fish, but only if the prey fish is below a certain health level. Having methods like canEatFish(Fish) and eatFish(Fish) will be helpful. When a fish eats, its health should increase.

3.3 Herbivore Class

This class will be an abstract subclass of Fish. Herbivores should eat seaweed when their health drops below a certain level. This will entirely replenish their health. Therefore, herbivores will only die of old-age or by predation.

3.4 SurgeonFish

Surgeon Fish are herbivores. They rarely reproduce but can have up to 5 children at a time.

3.5 ParrotFish

Parrot Fish are herbivores that reproduce more often than other fish but have will only have 1 child at a time.

3.6 Shark

Sharks are carnivores. They eat any fish, but they are unable to kill the mythical Giant Squids. They do not reproduce very often and only have 1-2 children in the litter.

3.7 GiantSquid

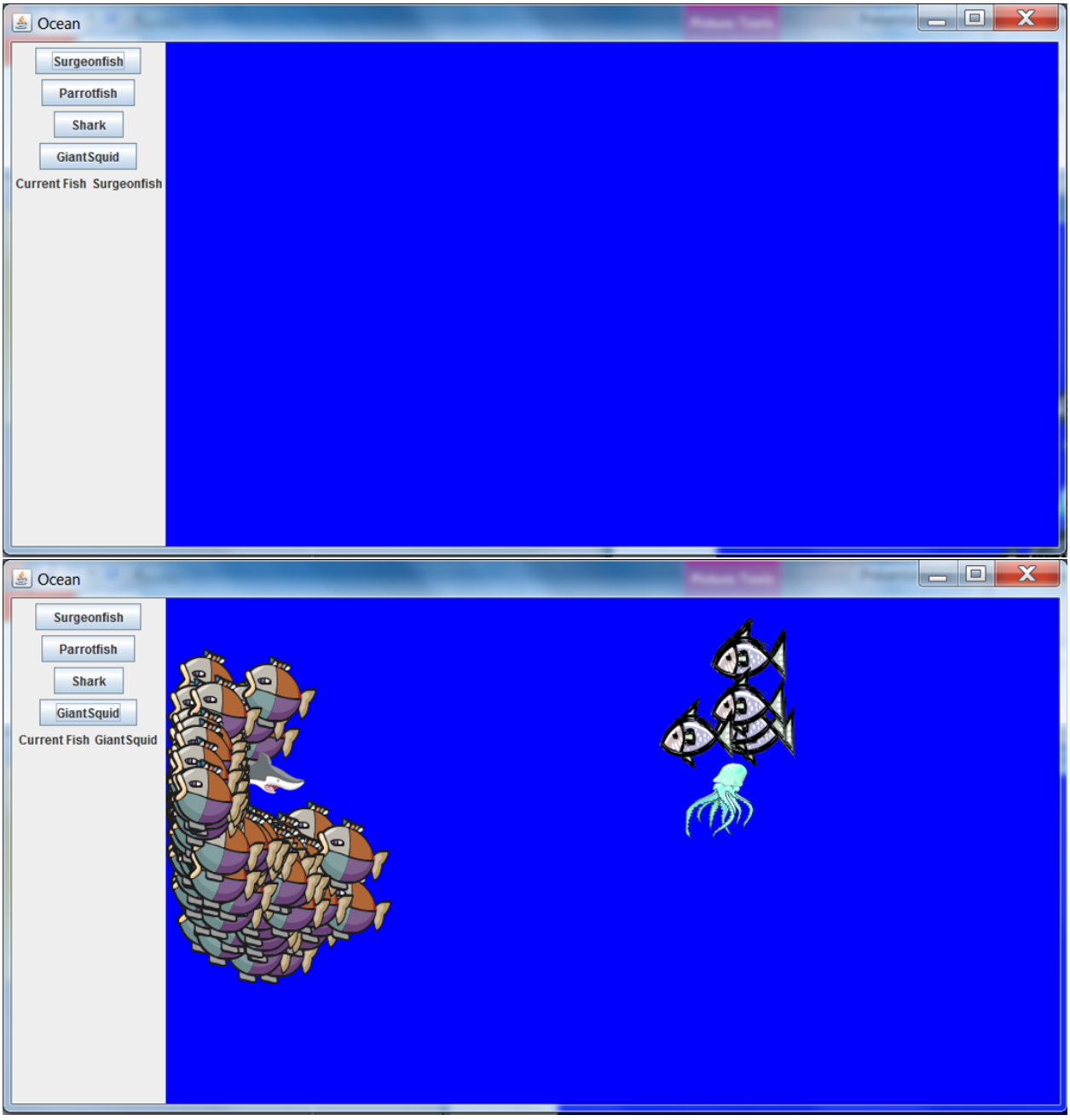
Giant Squids are carnivores that eat any fish, even sharks. However, due to ”technical difficulties”, they cannot reproduce.

3.8 Custom Classes

Here is where it gets fun. You must create two additional Fish species. These fish must have a special ability that differs from the default required Fish (i.e. the GiantSquid's ability to eat anything). Feel free to get creative here! If you're stuck, here's an idea: make a current in the ocean and only certain species can swim against it. Additionally, feel free to go above and beyond with the GUI. We want you to have fun with this one.

4 Output

While we will be flexible when looking at your output, it should at the very least look something like this:



5 Turn-in Procedure

Submit all of the Java source files we provided as well as all new Java source files you created to T-Square. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft.

6 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight Friday. Do not wait until the last minute!