A2: Test The Fish Tank: Let's Fix It

The fish tank part two: Let's fix it (https://markus.teach.cs.toronto.edu/csc207-2019-01/?locale=en)

Last updated: Mon. 28 Jan 2:20pm (setup instructions)

Assignment two focuses on fixing the code of the fish tank. This assignment is worth 7.5% of your final grade, broken down as follows:

- 1.5% of your final grade (20% of the assignment mark) will be awarded for having no spelling mistakes or warnings. See the "running inspections" section. Note that intellij has "quick format" tools, so you do not have to manually fix white space.
- 3% of your final grade (40% of the assignment mark) will be for implementing the requested features and fixing the existing code
- 3% of your final grade (40% of the assignment mark) will be for writing tests for all of the expected behaviors in any public method in the (Bubble, Fish, FollowingFish, HungryFish, Seaweed) classes. You will be evaluated by having those public methods replaced by buggy implementations and seeing if you have a unit test which catches the behavior (from the list below) in the expected test class.

Motivation

Why do we care about testing or software design?

CSC148 gave you the skills necessary to be a programmer. At this point in time, you could make all sorts of neat personal projects alone, without needing to care much about software design.

Our goal in CSC207 is to allow you to work on teams with other programmers. To refactor (e.g., clean up code, fix warnings & typos) is a boring, but important part of working in teams.

Similarly, writing tests ensures that other people working with you can easily understand the desired behavior of your code. This is crucial when you are working on a code base with hundreds of thousands of lines with potentially thousands of other people.

Getting started

1. Log into MarkUs, and click on Assignment 2 (a2), copy the link to your repo, then click on your repo link. Nothing will happen, but this step is important.

2. Open intellij in the usual way: Open a terminal, and type "idea".

3. Go to "File" in the top left, and press "Close Project" to get to the "choose project" view.

4. Once you are in the "choose project" view, press "Checkout from Version Control > Git" and then paste the clone link for the assignment two (**MarkUs link, click on assignment two).** **(https://markus.teach.cs.toronto.edu/csc207-2019-01)** If you are given the option, make the folder name "a2".

5. If prompted, say "yes" to whether you are creating a project from existing sources

6. A dialog box will appear asking whether you want to "import from external model" -- press the second button and click "maven" and press next once

7. After pressing next, ensure the top box is checked "search recursively for project files" and press next as many times as you can.

Starter code

We've provided you with some starter code in MarkUs, which has the following folder structure:

A2 >

src >

main >

java >

fishtank >

(.java files, e.g., from A1)

test >

java >

fishtank >

(tests)

src/main/java/fishtank contains similar files to assignment one, but we've made a few changes:

1. We've refactored FishTank.java to clean up the code smells that were in it. You should compare A2's "FishTank.java" with A1's "FishTank.java" to see an example of what we want after you've refactored the code.
2. We've added an abstract class called FishTankEntity to facilitate rewriting FishTank -- you don't need to write any tests for this class, but you should read it to understand what it does.
3. We've added a new fish called FollowingFish, which should stay exactly 2 units away from the given target fish

In src/test/java/fishtank/FishTest.java, we've provided a sample test file which contains a single (failing, but correct) unit test for Fish, and a single (succeeding) unit test for FollowingFish, which uses a mocked Fish for test purposes.

Running inspections

To run inspections (e.g., to find typos and warnings) -- go to the "Analyze" tab at the top bar, and press "Inspect Code" on the "Whole project"

Make sure you run inspections before every commit -- it's easy to make a last minute change which violates inspections, and we will not return marks for such an avoidable mistake!

Running tests

1. To set up IntelliJ to run the tests, click "Run" --> "Edit Configurations" --> open the only option in the list by clicking on it --> + JUnit --> Test Kind: "All in package" --> "Apply" --> OK

2. Now, you can right click the "java" folder in the "test" folder in the left pane of intellij, and press "Run 'All Tests'"

The results will show along the bottom area of the editor window.

How to edit the code

You may refactor and edit all of the provided code (including making new classes), with two exceptions:

1. You cannot edit the signature (e.g., name, return type, or input parameter types) of the constructor of any provided classes.
2. You cannot edit the signature of any "public" method in any of the provided classes.

We will be automarking your code, so you may get a zero if you make any of the above two changes.

How the fish tank should behave

For each of the following behaviors, ensure that they are correctly implemented, and have a test against them in the correct test class. You will be marked on the correctness of your implementation as well as the correctness of your test:

1. Nothing should leave the bounds of the FishTank (e.g., should not go off the side of the fish tank)
2. Nothing in the fishtank should collide in such a way that the same cell in FishTank contains two different entities (this would delete one of them!) -- In such scenarios, the creatures that would've collided should change directions if possible. If no movement in any direction is possible, they should stay still.
3. Fish and HungryFish should always go right if setGoingRight(true) was just called.
4. Fish and HungryFish should always go left if setGoingRight(false) was just called.
5. Fish and HungryFish should blow a bubble at their location (after moving) about 10% of the time
6. Fish and HungryFish should move up or down about 10% of the time
7. Fish and HungryFish should turn around 10% of the time

8. Fish and HungryFish should eat seaweed that they "bump" into, cutting the seaweed off (e.g., decreasing its length) at the height that the fish bumped into it.
9. Fish and HungryFish should never eat the whole seaweed (e.g., ensure that the length of the seaweed after being eaten is at least one)
10. Seaweed should grow back to their original length (the one passed into the constructor) at the rate of one segment per 200 updates.
11. FollowingFish should always be exactly two squares away from the fish it is following
12. FollowingFish should stay directly above or below the fish that it is following, unless it gets "cornered" along the top or bottom of the map, in which case it should move left or right to maintain exactly two distance (see FollowingFishTest for the distance calculation, we want vertDist + horizDist == 2 always.)
13. Bubbles should go left with 33% chance and right with 33% chance, and always go up
14. Bubbles that go off the map should be destroyed (e.g., call their delete() method. See FishTank's update loop)
15. Bubbles should not collide with fish or other bubbles. In such cases that they would, they should stay still

FAQ

Q: IntelliJ says "fishies" and "MEHUNGRY" are typos -- do I need to fix these?

A: No, we will only be checking for "legitimate" typos and poor names, e.g., "heybub" and "elss" -- we'll be relatively generous with this.

Q: Do I have to test FishTank? How do I write unit tests for a main method?

A: You inherently can't unit test the main method, to do so you'd have to be running an acceptance test. We are not asking you to modify FishTank.java.

Q: Do I need to test "setLocation(...)", ...?

A: No, it is useless to write explicit tests for "get..." and "set..." methods if they have no meaningful logic in them. We will not ask you to write unit tests for these.

Q: If I make a class extend another class, does that count as "changing a public method?"

A: No, feel free to make some classes extend others, as long as the method signatures of *all* of the public methods do not change.

Q: How do I test that something happens 10% of the time, instead of happening all the time?

A: If you run your "experiment" 1000 times, the number of times the event happens should be in the range (50, 150). You can understand this better if you take a probability course such as STA247.

Q: Is it a big deal if I don't mock things?

A: Yes, you will lose marks if a bug in Fish causes FollowingFish's tests to fail, for example.