Homework 02 – Gone Fishing

Authors: David
Topics: Javadocs, inheritance, constructor chaining, copy constructor, static variables, wrapper classes, good class design

Problem Description

The first wave of midterms has passed, and it's time to relax a little. You decide it's time to go fishing, and you return with an impressive haul! You decide to apply your object-oriented programming skills to record information about your new fish friends.

Solution Description

You will complete and turn in 4 classes: `Fish`, `Catfish`, `StripedBass`, and `FlyingFish`. Each of these classes have constructors, methods, and variables that are described below. A lot of the code will be reused through inheritance, so make sure you pay attention to the hints given below!

Notes:

1. All variables should be inaccessible from other classes and require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. Use constructor and method chaining whenever possible! Ensure that your constructor chaining helps you minimize code repetition and maximize code reuse!
4. Helper methods with appropriate visibility are optional.
5. Make sure to add Javadoc comments to your methods and classes!

Fish.java

This class defines a Fish object.

Variables:

- `String name` – the name of this fish
  - This field should be visible to any descendant classes.
  - An invalid value should default this field to "`Nemo`".
    - A value is invalid if it is an empty String, blank String, or null.
  - This value *cannot* be changed once set.
- `Double length` –the end-to-end length of this fish in inches
  - This field should be visible to any descendant classes.
  - An invalid value should default this field to 8.0.
    - A value is invalid if it is not a number, infinite, nonpositive, or null.
      - ***Hint**: Reference the Double API to help implement this logic.*
- `Double weight` –the weight of this fish in ounces
  - This field should be visible to any descendant classes.
  - An invalid value should default this field to 2.0.
    - A value is invalid if it is not a number, infinite, nonpositive, or null.

- `int totalFish` – a value tracking the number of fish that have been introduced to your aquarium
    - This variable should only be subject to modification in the body of `Fish` constructors.
    - ***Hint**: Consider what keyword enables a value to be tracked at the class, rather than instance level.*

Constructors:

- A constructor that takes in `name`, `length`, and `weight`.
    - **Note**: Constructor parameters should be written in the listed order.
- A constructor that takes in no parameters and sets `name` to "`Nemo`", `length` to 5.0, and `weight` to 12.0.
- A copy constructor that deep copies all necessary instance fields.
    - You may assume that the input will not be null.

Methods:

- `formatLength`
    - Returns the end-to-end length of this fish as a String, formatted precisely as follows (without the angle brackets and quotation marks):

      "`<feet> ft <inches> in`"

        - `feet` should be displayed as an integer, and `inches` should be displayed to 2 decimal places with rounding.
            - e.g., A 25.015 inch long fish should yield: "`2 ft 1.02 in`"
        - **Note**: There are 12 inches in 1 foot.
    - ***Hint:** Check the String API. It has a method that should remind you of printf.*
- `formatWeight`
    - Returns the weight of this fish as a String, formatted precisely as follows (without the angle brackets and quotation marks):

      "`<pounds> lbs <ounces> oz`"

        - If the weight of this fish includes only one pound, the unit should instead be "lb".
        - `pounds` should be displayed as an integer, and `ounces` should be displayed to 2 decimal places with rounding.
        - e.g., A 16.015 ounce fish would yield: "`1 lb 0.02 oz`"
        - **Note**: There are 16 ounces in 1 pound.
- `toString`
    - Overrides the corresponding method in its super class, such that a String with the following format is returned (without angle brackets, quotation marks, and trailing or leading whitespace):

      "`I'm a talking fish named <name>. My length is <feet> ft <inches> in and my weight is <pounds> lbs <ounces> oz.`"

        - Floating-point values should be displayed to two decimal places with rounding.
- **No getters or setters should be written.**

Catfish.java

This class defines a Catfish object. It is a subclass of Fish. A Catfish may be the best friend of a StripedBass.

Variables:

- `Double whiskerLength` – the length of this catfish's longest whisker in inches
  - An invalid value should default this field to 8.0.
    - A value is invalid if it is not a number, infinite, nonpositive, or null.

Constructors:

- A constructor that takes in `name`, `length`, `weight`, and `whiskerLength`.
  - **Note**: Constructor parameters should be written in the listed order.
- A constructor that takes in no parameters and sets `name` to "`Bubba`", `length` to 52.0, `weight` to 720.0, and `whiskerLength` to 5.0.
- A copy constructor that deep copies all necessary instance fields.
- You may assume that the input will not be null.

Methods:

- `isShaggy`
  - Returns a boolean reflecting if this catfish is shaggy.
  - A catfish is said to be shaggy if its longest whisker is longer than its end-to-end length.
- `toString`
  - Overrides the corresponding method in its super class, such that a String with the following format is returned (without angle brackets, curly braces, quotation marks, and trailing or leading whitespace):

    "`I'm a talking fish named <name>. My length is <feet> ft <inches> in and my weight is <pounds> lbs <ounces> oz. I'm a catfish whose longest whisker is <whiskerLength>, so I {am/am not} shaggy.`"

    - Floating-point values should be displayed to two decimal places with rounding.
  - ***Hint**: What keyword can you use to help you reuse code?*
- Setter for `whiskerLength`.
  - If the input is invalid, set the field to its default value.
- **No getters should be written**.

StripedBass.java

This class defines a StripedBass object. It is a subclass of Fish. A StripedBass may be best friends with a Catfish.

Variables:

- `int stripeCount` – the count of all stripes on this striped bass
  - An invalid value should default this field to 25.
    - A value is invalid if it is nonpositive.
- `boolean isSaltwater` – whether this striped bass resides in a saltwater body
- `Catfish bestFriend`– a catfish who is the best friend of this striped bass

Constructors:

- A constructor that takes in `name`, `length`, `weight`, `stripeCount`, `isSaltwater`, and `bestFriend`.
  - **Note**: The `bestFriend` argument should be deep copied.
  - **Note**: Constructor parameters should be written in the listed order.
- A constructor that takes in no parameters and sets `name` to "`Striper`", `length` to 30.0, `weight` to 320.0, `stripeCount` to 14, `isSaltwater` to false, and `bestFriend` to null.
- A copy constructor that deep copies all necessary instance fields.
  - You may assume that the input will not be null.

Methods:

- `migrate`
  - The striped bass moves to the opposite kind of water body if and only if it does not have a best friend.
    - For example, if this striped bass were now in a freshwater body and it has a best friend, it would not migrate to a saltwater body but instead stay in the freshwater body.
- `toString`
  - Overrides the corresponding method in its super class, such that a String with the following format is returned (without angle brackets, curly braces, quotation marks, and trailing or leading whitespace):

    "`I'm a talking fish named <name>. My length is <feet> ft <inches> in and my weight is <pounds> lbs <ounces> oz. I'm a {saltwater/freshwater} striped bass with <stripeCount> stripes. I have {no best friend/a best friend named <bestFriendName>}.`"

    - Floating-point values should be displayed to two decimal places with rounding.
  - ***Hint**: What keyword can you use to help you reuse code?*
- **No getters or setters should be written.**

FlyingFish.java

This class defines a FlyingFish object and should be a subclass of Fish. A heavier fish is said to be more powerful, as flying is more difficult for it.

Variables:

- `int flightTime` – the amount of time a flying fish can stay in the air in seconds
  - An invalid value should default this field to 30.
    - A value is invalid if it is nonpositive.

Constructors:

- A constructor that takes in `name`, `length`, `weight`, and `flightTime`.
  - **Note**: Constructor parameters should be written in the listed order.
- A constructor that takes in no parameters and sets `name` to "`Gilbert`", `length` to 12.0. `weight` to 24.0, and `flightTime` to 36.
- A copy constructor that deep copies all necessary instance fields.
  - You may assume that the input will not be null.

Methods:

- `calculatePower`
  - Returns this flying fish's power, represented as a `double`.
    - A flying fish's power is the product of its weight and flight time.
- `fly`
  - Prints a message followed by a newline in the format below (without angle brackets and quotation marks):

    "`Woohoo! <name> flew for <time> seconds.`"

    - `time` should be a random floating-point value in the range $(0, \text{flightTime}]$, displayed to two decimal places with rounding.
- `toString`
  - Overrides the corresponding method in its super class, such that a String with the following format is returned (without angle brackets, curly braces, quotation marks, and trailing or leading whitespace):

    "`I'm a talking fish named <name>. My length is <feet> ft <inches> in and my weight is <pounds> lbs <ounces> oz. I'm a flying fish, and my flight time record is <flightTime>, so my power is <power>.`"

    - `power` should be displayed to 2 decimal places with rounding.
- **No getters and setters should be written.**

Aquarium.java

This class is a test driver, meaning it will be used to test all the code in the above classes. We recommend that you include the following tests at a minimum.

Methods:

- `main`
  - For the Fish class and all its subclasses, create at least one instance of each using each of the constructors.
  - Invoke each public method defined in a class on its respective instance.
  - For each kind of Fish, print its `toString`.

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 15 points.** This means there is a maximum point deduction of 15. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar YourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the number of points we would take off (limited by the Checkstyle cap). In future assignments we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Fish.java`
- `StripedBass.java`
- `Catfish.java`
- `FlyingFish.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

- `java.util.Random`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items.
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.