COMP 2522
Object Oriented Programming
Lab 01

Yue Wang
yue_wang@bcit.ca

Due before noon on Sunday, January 15, 2023

Introduction

Welcome to your first COMP 2522 lab. Today you will apply what you have started learning as we migrate from Python to Java. You will implement your first classes.

Developing software is often a highly iterative process. Iterative software development involves repeated cycles of designing, coding, testing, and integrating small sections or increments of a program. We build a little, show it to our clients, make adjustments, and continue.

By deconstructing a large project into smaller, more manageable tasks, we can abstract away portions of the application and focus on separate, solvable problems. The knowledge we gain from developing and testing these smaller modules can be applied to the development of other parts of the project. Ultimately we generate correct, working, and maintainable software efficiently and quickly.

You have been hired to generate some helpful software for a group of environmentalal scientists. A new and invasive species of *Poecilia*, the live-bearing aquarium fish also known as the guppy, has been discovered in some of the hot springs, pools, and streams near Skookumchuk here in BC. The owners of the land have asked the scientists to study the prolific little fish and determine, among other things, if its reproductive rate will be a threat to native aquatic fauna.

Figure 1: *Guppies* by Per Harald Olsen via Wikimedia [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)]

The scientists are still studying the problem domain, and they haven't finalized the scope of the simulations they want to conduct. But that's okay. Object-oriented analysis, design, and programming are well suited to developing software systems whose requirements are incomplete and changing.

During the first few analysis meetings with the scientists, you identified some fundamental classes in the problem domain. For the first iteration of your software project, you will abstract away much of the potentially complicated logic and design in the application. You will begin by writing one of the classes that represent the basic building blocks of the simulations the scientists will want to run.

1 Submission requirements

This lab must be submitted to D2L at or before 11:59:59 AM on Sunday, January 15, 2023.

Don't wait until the last minute as you never know what might happen (computer issues, no Internet, power outage, ...). Submit often, and please make sure your comments are short and clear and specific.

You must not ignore the code style warnings produced by IntelliJ. **You must style your code so that it does not generate any IntelliJ warnings.**

When code is underlined in red, IntelliJ is telling us there is an error. Mouse over the error to (hopefully!) learn more.

When code is underlined in a different colour, we are being warned about something. We can click the warning triangle in the upper right hand corner of the editor window, or we can mouse over the warning to learn more.

2 Grading scheme

Figure 2: This lab, like all labs, will be graded out of 5

Your lab will be marked out of 5. On a high level, expect that marks will be assigned for:

1. (3 point) Meeting the functional requirements in this lab, i.e., does the code do what it is supposed to do? Does your code pass all 51 unit tests? Did you forget anything else?
2. (1 point) Meeting non-functional Java OOP design requirements such as encapsulation, information hiding, minimizing mutability, minimizing scope, etc.
3. (1 point) Writing code that is self-documenting and well-styled that generates no warnings from IntelliJ or Checkstyle, with correct and complete Javadocs, using short and atomic methods correctly encapsulated in a class that is well-defined.

3 Implementation requirements

Download and import "Lab01.zip" from D2L. You must implement a class called **Guppy.java** which contains the following elements:

1. The following ten public **symbolic constants**. Use these variable names and data types. In Java, symbolic constants are **static** and **final**. Do not use different names or data types, and do not add any other symbolic constants. Use these constants instead of magic numbers inside your code:
   (a) `FIRST_GENERATION` an integer equal to 0
   (b) `YOUNG_FISH_AGE_IN_WEEKS` an integer equal to 10
   (c) `MATURE_FISH_AGE_IN_WEEKS` an integer equal to 30
   (d) `MAXIMUM_AGE_IN_WEEKS` an integer equal to 50
   (e) `MINIMUM_WATER_VOLUME_ML` a double equal to 250.0
   (f) `DEFAULT_GENUS` a String equal to "Poecilia"
   (g) `DEFAULT_SPECIES` a String equal to "reticulata"
   (h) `DEFAULT_HEALTH_COEFFICIENT` a double equal to 0.5

(i) `MINIMUM_HEALTH_COEFFICIENT` a double equal to 0.0

(j) `MAXIMUM_HEALTH_COEFFICIENT` a double equal to 1.0

2. The following eight private **instance variables**. Use these variable names and data types. Do not use different names or data types, and do not add any other instance variables:

   (a) `genus` (first word of the scientific binomial two-part name) a String

   (b) `species` (second word of the scientific binomial name) a String

   (c) `ageInWeeks` an integer

   (d) `isFemale` a boolean

   (e) `generationNumber` an integer

   (f) `isAlive` a boolean

   (g) `healthCoefficient` a double

   (h) `identificationNumber` an integer

3. `numberOfGuppiesBorn`, a private **static integer** that has an initial value of 0. This variable is incremented by 1 each time a new guppy is constructed, and the newly incremented value is assigned to the new guppy's `identificationNumber` instance variable. Further details are provided in the section about constructors (remember that static variables are shared by all objects of a class).

4. **Two (2) constructors**:

   (a) A zero-parameter constructor which sets `ageInWeeks` and `generationNumber` to zero, and sets:

      i. `genus` to `DEFAULT_GENUS`

      ii. `species` to `DEFAULT_SPECIES`

      iii. `isFemale` to `true`

      iv. `isAlive` to `true`

      v. `healthCoefficient` to `DEFAULT_HEALTH_COEFFICIENT`

      vi. `identificationNumber` to the newly incremented value of `numberOfGuppiesBorn`

   (b) A second constructor which accepts the following parameters in the following order:

      i. `newGenus` a String. Format the String passed in `newGenus` correctly (capital first letter, the rest lower case) and assign the new String to the `genus` instance variable.

      ii. `newSpecies` a String. Format the String passed in `newSpecies` correctly (lower case) and assign the new String to the `species` instance variable.

      iii. `newAgeInWeeks` a positive integer. Assign `newAgeInWeeks` to the `ageInWeeks` instance variable. If the parameter passed to the method is negative, assign a 0.

      iv. `newIsFemale` a boolean to assign to the `isFemale` instance variable.

      v. `newGenerationNumber` an integer. If the argument passed as the parameter is less than zero, set the value to **zero**. Otherwise assign `newGenerationNumber` to the `generationNumber` instance variable.

      vi. `newHealthCoefficient` a double. If the argument passed as the parameter is less than `MINIMUM_HEALTH_COEFFICIENT` or greater than `MAXIMUM_HEALTH_COEFFICIENT`, set the value to the closest bound. Otherwise assign `newHealthCoefficient` to the instance variable called `healthCoefficient`.

      vii. This constructor should not accept a parameter for the `isAlive` instance variable. Set the `isAlive` variable to true. Every new Guppy is alive by default.

   (c) When the Guppy is constructed, we must also increment the static `numberOfGuppiesBorn` variable by 1 and and assign the new value to `identificationNumber` in the Guppy being constructed.

5. A method with the header `public void incrementAge()` which increases the value in the `ageInWeeks` instance variable field by 1. If the new value in `ageInWeeks` is greater than `MAXIMUM_AGE_IN_WEEKS`, set the `isAlive` instance variable to false.

6. An **accessor** (getter) for all eight instance variables, and a **mutator** (setter) for `ageInWeeks`, `isAlive`, and `healthCoefficient`:
   (a) Do not create mutators for `genus`, `species`, `isFemale`, or `identificationNumber`. In fact, those instance variables should be declared final!
   (b) Every accessor method name must follow the pattern `getVariableName`.
   (c) Every mutator method name must follow the pattern `setVariableName`.
   (d) Also create a static accessor for the `numberOfGuppiesBorn` static variable. It must be called `public static int getNumberOfGuppiesBorn()`.
   (e) The mutator for `ageInWeeks` must ignore values below 0 and above `MAXIMUM_AGE_IN_WEEKS`.
   (f) The mutator for `healthCoefficient` must ignore values that would cause the `healthCoefficient` variable to exceed its bounds.
7. A method with the header `public double getVolumeNeeded()` which returns the volume of water in millilitres that the guppy needs according to the following formula:
   (a) If the fish is less than 10 weeks old, return `MINIMUM_WATER_VOLUME_ML`.
   (b) If the fish is 10 to 30 weeks old, return `MINIMUM_WATER_VOLUME_ML * ageInWeeks / YOUNG_FISH_AGE_IN_WEEKS`.
   (c) If the fish is 31 to 50 weeks old, return `MINIMUM_WATER_VOLUME_ML * 1.5`.
   (d) If the fish is older than 50 weeks, return 0.0.
8. a method with the header `public void changeHealthCoefficient(double delta)` which adds the value passed in the `delta` parameter to the `healthCoefficient` instance variable. The parameter `delta` may be positive or negative, but if the new value of `healthCoefficient` would be less than or equal to `MINIMUM_HEALTH_COEFFICIENT`, in addition to setting the value of `healthCoefficient` to 0.0, set the `isAlive` instance variable to false. If the new value of `healthCoefficient` would be greater than `MAXIMUM_HEALTH_COEFFICIENT`, set `healthCoefficient` to `MAXIMUM_HEALTH_COEFFICIENT`.
9. a method with the header `public String toString()` which returns a String representation of the guppy. The version generated by IntelliJ is sufficient.
10. **You can test your code by right-clicking the TestGuppy.java JUnit test file included with the template inside the Test folder. There are 51 unit tests. Pass all of them please!**

That's it! Good luck, and have fun!