Homework 04 – Rare Frogs and Speedy Flies

Problem Description

Hello! Please make sure to read all parts of this document carefully. This assignment focuses on topics from L10-11.

In this assignment, you will be applying your knowledge of writing classes, visibility modifiers, constructors, methods, static variables & constants, accessors & mutators, constructor overloading, constructor chaining and toString() to begin real work in Object-Oriented programming! You will create Frog, Fly, and Pond classes that will simulate how they interact with each other in the real world.

Remember to test for Checkstyle errors!

Solution Description

Create files the files Fly.java, Frog.java, and Pond.java from scratch. Pond.java will be a driver class, meaning it creates instances of Fly and Frog objects and "drives" their values based on actions, therefore simulating a pond environment.

You will be creating a number of fields and methods for each file. Based on the description given for each variable and method, you will have to decide whether or not the variables/method should be static and whether it should be private or public. To make these decisions, you should carefully follow the guidelines on these keywords as taught in lecture. In some cases, your program will still function with an incorrect keyword.

`Fly.java`

This Java file defines the behavior and attributes of fly objects that exist within the pond.

Variables
All variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. Hint: There is a specific visibility modifier that can do this!

The `Fly` class must have these variables:

- `mass` – the mass of the fly in grams
    - The type of this variable must allow decimal values for mass
- `speed` – how quickly the fly can maneuver through the air, represented as a double

Constructors
You must use constructor chaining for your constructors. Duplicate code cannot exist in multiple constructors. The constructors ***must*** take the variables in the specified order. (Hint: Refer to L10: Constructors if needed)

- A 2-arg constructor that takes in mass and speed, setting all fields accordingly.

- A 1-arg constructor that takes in only mass and assigns the following default values:
    - `speed`: 10
- A no-arg constructor that assigns the following:
    - `mass`: 5
    - `speed`: 10

Additionally, you can assume all values passed into the constructors will be valid and non-negative.

Methods

Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- `grow` – takes in an int parameter representing added mass, which then changes `speed` as well
    - This method increases the mass of the Fly by the given mass
    - The method changes the speed based on mass as follows:
        - If the `mass` of the Fly is less than 20, speed is increased by 1 for every unit of mass gained up until the mass reaches 20
        - If the `mass` of the Fly is ≥ 20, speed is decreased by 0.5 for every unit of mass added after 20 units
    - You can assume that the the Fly is alive (i.e. mass is greater than 0) when this method is called.
    - This method does not return anything.
- `isDead` – takes in no parameters and returns a value based on whether the Fly is dead
    - If the `mass` is 0, return true. Otherwise, return false.
- `toString` – takes in no parameters and returns a String describing the Fly as follows. Replace the values in brackets <> with the actual value
    - If `mass` is 0, return "I'm dead, but I used to be a fly with a speed of <speed>"
    - Otherwise, return "I'm a speedy fly with <speed> speed and <mass> mass."
    - Do not include the surrounding double quotes in the output. Additionally, specify all decimal values to 2 decimal points.
- Appropriately named getters and setters for all instance variables of Fly.

`Frog.java`

This Java file defines the behavior and attributes of frog objects that exist within the pond.

Variables

All variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. Hint: There is a specific visibility modifier that can do this!

The Frog class must have these variables:

- `name` – the name of this Frog, which can be made of any combination of characters
- `age` – the age of the Frog in months, represented as an int
- `tongueSpeed` – how quickly this Frog's tongue can shoot out of its mouth, represented as a double
- `isFroglet` – a value that represents whether this Frog is a froglet (the stage between tadpole and adult frog), which only has two possible values: true or false

    - A Frog is a froglet if its `age` is within (1 month, 7 months).
    - Whenever age is changed, this variable must be updated accordingly.
- `species` - the name of the species of this Frog,
    - This value must be same for all instances of Frog. (Hint: There is a keyword you can use to accomplish this).
    - By default, this value is "Rare Pepe".

Constructors

You **must use constructor chaining** for your constructors. Duplicate code cannot exist in multiple constructors. The constructors ***must*** take the variables in the specified order.

- A 3-arg constructor that takes in `name`, `age`, and `tongueSpeed` and sets all variables appropriately.
- A 3-arg constructor that takes in `name`, `ageInYears` (representing the age of the Frog in years as a double), and `tongueSpeed` and sets all variables appropriately.
    - When converting `ageInYears` to `age` (in an int number of months), round down to the nearest month without using any method calls
    - Hint: Java can automatically do this for you with casting.
- A 1-arg constructor that takes in just a name with the following default values
    - `age`: 5
    - `tongueSpeed`: 5

Methods

You must use method overloading at least once. Do not create any other methods than those specified. Any extra methods will result in point deductions. All methods must have the proper visibility to be used where it is specified they are used.

- `grow` – takes in a whole number parameter representing the number of months. As the Frog ages, tongueSpeed is also affected
    - This method ages the Frog by the given number of months.
    - The Frog's `tongueSpeed` attribute changes as follows:
        - `tongueSpeed` is increased by 1 for every month the Frog grows until the frog is 12 months old
        - On the other hand, `tongueSpeed` is decreased by 1 for every month the frog ages beyond 30 months old
        - The minimum `tongueSpeed` is 5. Ensure that its value does not drop to less than 5
        - Example: When '`frogA`' is 10 months old and grows 4 months, its `tongueSpeed` should increase by 2. When '`frogB`' is 25 months old and ages by 10, its `tongueSpeed` should decrease by 5.
    - Remember to update `isFroglet` accordingly
    - This method does not return anything
- `grow` – takes in no parameters and ages the Frog by one month and updates `tongueSpeed` accordingly (under the same rules as the other grow method above)
- `eat` – takes in a Fly object as a parameter, which the Frog will attempt to catch and eat.
    - If the Fly is dead, the frog cannot eat it. Therefore, terminate the method if the Fly object passed in is dead.
    - The Fly is caught if the Frog's `tongueSpeed` is greater than the speed of the Fly.
    - If the Fly is caught, set the `mass` of the Fly to 0

        - Additionally, if the Fly's original `mass` is at least 0.5 times the `age` of the Frog, the Frog ages by one month using its `grow()` method.
    - If the Fly is NOT caught, the Fly should grow by a `mass` of 1 (and have an updated `speed` value based on the growth).
        - This should be done using one method.
- `toString` – takes in no parameters and returns a String describing the Frog as follows. Replace the values in brackets <> with the actual value
    - If the Frog is a froglet, return "My name is <name> and I'm a rare froglet! I'm <age> months old and my tongue has a speed of <tongueSpeed>."
    - Otherwise, return "My name is <name> and I'm a rare frog. I'm <age> months old and my tongue has a speed of <tongueSpeed>."
    - Do not include the surrounding double quotes in the output. Additionally, specify all decimal values to 2 decimal points.
- Appropriately named getter and setter for species, which must change the value for all instances of the class.

`Pond.java`

This Java file is a driver, meaning it will contain and run Frog and Fly objects and "drive" their values according to a simulated set of actions. You can use it to test your code and are encouraged to test out different scenarios.

For your submitted Pond.java file, we require the following actions in the main method:

- Create at least 4 Frog instances
    - Frog named Peepo with default attributes
    - Frog named Pepe with the following values
        - age: 10
        - tongueSpeed: 15
    - Frog named Peepaw with the following values
        - age: 4.6
        - tongueSpeed: 5
    - Frog of your liking :)
- Create at least 3 Fly instances:
    - Fly with mass of 1 and speed of 3.
    - Fly with mass of 6 and default attributes.
    - Fly of your liking :)
- Perform the following method calls in order:
    - Set the species of any Frog to "1331 Frogs"
    - On its own line, print the description of the Peepo specified in its toString() method.
    - Have the Frog named Peepo attempt to eat the Fly with a mass of 6.
    - On its own line, print the description of the Fly with a mass of 6 specified in its toString() method.
    - Have Peepo grow by 8 months.
    - Have Peepo attempt to eat the Fly with a mass of 6.
    - Afterwards, print the following (each on its own line):
        - Print the description of the Fly with a mass of 6 specified in its toString() method.
        - Print the description of the Frog named Peepo specified in its toString() method.

        - Print out on a new line the description of your own Frog specified in its toString() method.
    - Have Peepaw grow by 4 months.
    - Afterwards, print the following (each on its own line):
        - Print the description of the Frog named Peepaw specified in its toString() method.
        - Print the description of the Frog named Pepe specified in its toString() method.

Example output

```
1   My name is Peepo and I'm a rare froglet! I'm 5 months old and my tongue
    has a speed of 5.00.
2   I'm a speedy fly with 11.00 speed and 7.00 mass.
3   I'm dead, but I used to be a fly with a speed of 11.00.
4   My name is Peepo and I'm a rare frog. I'm 14 months old and my tongue
    has a speed of 12.00.
5   <output varies based on your custom frog>
6   My name is Peepaw and I'm a rare frog. I'm 59 months old and my tongue
    has a speed of 5.00.
7   My name is Pepe and I'm a rare frog. I'm 10 months old and my tongue has
    a speed of 15.00.
```

Rubric

[45] `Fly.java`

- [4] Correct variables mass and speed with proper visibility, encapsulation, and data types
- [10] Correctly creates constructors with constructor chaining
    - [3] Constructor with params mass and speed
    - [3] Constructor with params mass
    - [4] Constructor with no params
- [12] Correctly implements setters and getters
    - [3] setter method for speed
    - [3] setter method for mass
    - [3] getter method for speed
    - [3] getter method for mass
- [5] Correctly implements toString()
- [10] Correctly implements grow()
- [4] Correctly implements isDead()

[45] `Frog.java`

- [10] Correct variables with proper visibility, encapsulation, and data types
    - [2] name
    - [2] age
    - [2] tongueSpeed
    - [1] isFroglet
    - [3] species
- [10] Correctly implements grow()
    - [8] grow method with one parameter
    - [2] grow method with no parameters

- [10] Correctly implements eat()
- [10] Correctly implements toString()
- [5] Correctly implements setters and getters for species

[10] Pond.java
- [4] Correct Frog objects
- [3] Correct Fly objects
- [3] Correct testing of the objects

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Allowed Imports

To prevent trivialization of the assignment, **you are not allowed to import any classes or packages**.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:
- `var (the reserved keyword)`
- `System.exit`

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded from Canvas under **Files → Homework → Checkstyle**

To run Checkstyle, put the jar file in the same folder as your homework files and run

```
$ java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **10 points**. This means that up to **10** points can be deducted due to Checkstyle errors (1 point per error). Starting HW5, any missing/invalid Javadocs will be marked as a Checkstyle error.

Javadoc

For this assignment, you will be commenting your code with Javadoc. Javadoc comments are a clean and useful way to document your code's functionality. For more information on what Javadoc comments are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the Javadoc overview for your code using the command below, which will put all the files into a folder named "javadoc". Note you should execute this after adding Javadoc comments to your code:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 *This class represents a Dog object.
 *@author George P.Burdell
 *@version 1.0
 */
public class Dog {

    /**
     *Creates an awesome dog(NOT a dawg!)
     */
    public Dog() {
        ...
    }

    /**
     *This method takes in two ints and returns their sum
     *@param a first number
     *@param b second number
     *@return sum of a and b
     */
    public int add(int a,int b) {
        ...
    }
}
```

A more thorough tutorial for Javadoc Comments can be found here.

Take note of a few things:

1. Javadoc comments begin with `/**` and end with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

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

- Frog.java
- Fly.java
- Pond.java

Make sure you see the message stating "HW04 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

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

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds 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.

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.

Important Notes (Do not 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 Piazza for a note containing all official clarifications