Homework 1

Authors: Akul, Tejas, Vince, Connor, Kevin, Alex, Jack, Andrew

Purpose

To complete this assignment, you will apply your knowledge of classes, the `this` keyword, encapsulation, and object equality. As such you will be creating three classes – `Vet`, `Dog`, and `DogDriver`- these classes will simulate a dog going in for a checkup in the real world. As always, use good design and follow the principles of object oriented programming!

Solution Description

For this assignment, you will be creating three classes to simulate how a visit to vet might go in the real world: one class representing the dog- `Dog.java`, one class representing the vet- `Vet.java`, and one class to run the simulation of visiting the vet- `DogDriver.java`. You will also have to adhere to proper checkstyle.

`Dog.java`

This file represents a dog visiting the Java clinic

- Variables (all the fields below should follow the rules of encapsulation)
  - `name` that holds the name of the dog (will always be a String)
  - `type` holds the type of the dog (examples: "Golden Retriever", "Husk", etc.)
  - `age` holds the age of the dog (will always be an `int`)
  - `visitCounter` holds a counter of how many times this dog object has visited the vet (will always be an `int`)
- Constructors
  - A constructor that takes values in the following order- `name, type, age,` and `number of times the dog has visited the vet`. The constructor should initialize the fields with the values passed in.
  - A constructor that takes values in the following order- `name` and `type`. The constructor should initialize the fields with the values passed in.

    In this case a dog is automatically 3 years old, and has visited the vet 2 times previously
  - **In order to receive full credit, you must use constructor chaining**
  - Note: the types the constructor takes in should match the fields they are assigning
- Methods
  - A `toString` method that returns a String representation of a Dog object
    - If the dog has visited the vet before, the String should be:
      `"I am a [type] named [name] and I am [age] years old. I have visited the vet before [visitCounter] times.`
    - If the dog has not visited the vet before, the String should be:
      `"I am a [type] named [name] and I am [age] years old. I have never visited the vet before."`
  - An `equals` method dependent on the `age, name, type` and `number of visits to the vet`
    - You may choose to implement either the overloaded (Takes in a `Dog`) or the overriden (Takes in an `Object`) version of the `equals` method.
  - A method called `visitVet()` for when a dog visits the vet

        - If the dog has not visited the vet before then print to the console `"This is my first visit and I am a little scared!"`
        - If the dog has previously visited the vet, print to the console `"I have been here [visitCounter] times. The vet isn't that scary!"`

- Create accessors and mutator methods for all variables
    - Ensure no negative values are passed in for `age` and `visitCounter`. If a user attempts to set `age` or `visitCounter` to a negative value, nothing should happen, meaning `age` and `visitCounter` should not change.

`Vet.java`

This file represents a vet in the Java clinic

- Variables (all the fields below should follow the rules of encapsulation)
    - `name` that holds the name of the vet
    - `regularDog` the dog object that consistently visits this vet, set if the vet does not already have a `regularDog`
    - `petsSeen` holds the number of dogs that this vet object has seen before
    - `officeVisits` holds a counter of how many visits the java clinic has had as a whole, across all `Vet` objects. (consider what modifier will be used)
- Constructors
    - A constructor that takes values in the following order- `vetName`, `regularDog`, `petsSeen`. The constructor should initialize the fields with the values passed in.
    - A one argument constructor that takes values in `vetName`. The constructor should initialize the fields with the values passed in.
        - `regularDog` should be null
        - `petsSeen` should be 0
    - A constructor with no arguments that initializes the fields with the values passed in.
        - `vetName` should be "James Herriot"
        - `regularDog` should be null
        - 2 `petsSeen` (Do **not** worry about updating `officeVisits` because of this)
    - **In order to receive full credit, you must use constructor chaining**
    - Note: the types the constructor takes in should match the fields they are assigning
- Methods
    - A `toString` method that returns a string representation of a Vet object
        - If the vet does not have pet they regularly see, the String should be: `"Hello, I am [name] and I have worked with [petsSeen] pets."`
        - If the vet does have a pet they regularly see, the String should be: `"Hello, I am [name] and I have worked with [petSeen] pets. I have a regular dog that comes to me named [dog name]."`
        - Note: Do not include the brackets in your printouts
    - An `equals` method dependent **only** on the `name`, `regularDog`, and `petsSeen`.
        - You may choose to implement either the overloaded (Takes in a `Vet`) or the overriden (Takes in an `Object`) version of the `equals` method.
    - A method called `vetVisit` that takes in a Dog object.
        - If the Vet does not have a `regularDog` set the Dog object taken in to `regularDog`
        - This method should call the `dog's visitVet` method

        - Any time a dog visits the vet's office, the `officeVisits` variable and the `petsSeen` variable should be incremented
        - This method should also print out:
          `"A vet named [name] treated [dog name] today."`
    - Create accessors and mutator methods for all variables
        - `Static` fields should have static getters and setters
        - Ensure no negative values are passed in to `petsSeen` and `officeVisits`. If a user attempts to set `petsSeen` or `officeVisits` to a negative value, nothing should happen, meaning `petsSeen` and `officeVisits` should not change.

`DogDriver.java`

- This file is the driver, which will allow you to run your simulation. This will allow you to run and test your code. You will do the following in your `DogDriver`
    - Create 1 Dog object called `d1` named `John` that is a `Husky`
    - Create 1 Vet object called `v1` named `Tim`
    - Call the `visitVet` method on `d1`
    - Print to the console `d1` and `v1`
    - The output to the console should be:

```
I have been here 2 times. The vet isn't that scary!

I am a Husky named John and I am 3 years old. I have visited the vet
before 3 times.

Hello, I am Tim and I have worked with 0 pets.
```

    - Test your code yourself! Neither the auto grader nor our requirements are fully comprehensive.
    - You will not be turning in this `DogDriver.java` file

Import Restrictions

You may not import anything for this homework assignment.

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`

Collaboration

- *Collaboration Statement*
- To ensure that you acknowledge 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].

- Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

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

- `Dog.java`
- `Vet.java`

Make sure you see the message stating "HW01 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. 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:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. 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.

Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is 15 points. If you don't have checkstyle yet, download it from Canvas. Place it in the same folder as the files you want checkstyle. 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 points we would take off (limited to the amount we specified above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

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 Piazza for all official clarifications