Homework 04 – Doggy Lunch Time

Problem Description

Hello! Please make sure to read all parts of this document carefully. This HW assignment focuses on **lessons 10 and 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 a **Doggy.java**, **Treat.java**, and **LunchTime.java** file that will simulate how a Dog acts in real life.

Solution Description

In this homework, you will be creating a `Doggy.java` file, `Treat.java` file, and a `LunchTime.java` file to simulate a typical lunch time of a dog!

`Doggy.java`

This class represents a dog who will be eating a treat.

Variables

All variables must not be 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 Doggy class have the following variables:

- name – the name of the Doggy which should be represented as a String value
- furColor – the color of the Doggy's fur which should be represented as a String value
- glow – a boolean representing if the Doggy is currently glowing
- height – the height of the Doggy in inches represented as a double
- doggyPopulation – class variable representing the total number of Doggy objects as an integer value
    - Initialize variable doggyPopulation to 0
    - **doggyPopulation should increase by 1 each time a Doggy object is created** (**Hint:** Consider how this can be done within the constructor)
- treatPantry – an array of Treats representing the treats available in the shared pantry
    - **All doggies** should be able to access this array
    - Size of the pantry array is 5
    - Pantry should be empty when initialized (all values null)

Constructors

You **must use constructor chaining** in at least two of your constructors. In addition, the constructors ***must*** take the variables in the specified order.

- A constructor that takes in name, furColor, glow, and height and sets all variables appropriately.

- A constructor that only takes in name, furColor, glow. The other variables have the following default values:
    - height: 105.0
- A constructor that takes in no parameters, but has the following default values:
    - name: "Clifford"
    - furColor: "Red"
    - glow: false
    - height: 105.0

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 modifier to be used outside of this class

- `goOutside()`
    - Increase height by 1.5 and set the Doggy's glow to true.
    - Print "I am having way too much fun!"
    - Does not return anything
- `addTreat(Treat t)`
    - Adds the treat in the next available treatPantry slot (first 'null' element).
        - If the pantry is full, print out "Pantry is full."
        - The treat addition was unsuccessful, return false
    - After the treat is added to the pantry, print out the treat brands in the current pantry
        - (Ex: "Current Pantry: Purina, Treat2, Treat3")
    - Prints out the total sum of nutrients after the treat is added.
        - (Ex: "Total Nutrients: 34")
    - The treat addition was successful, return true
- `eatTreat(String shape)`
    - Eats (removes) the first available treat that matches the shape indicated.
        - If the treat makes dogs happy, then the dog will start to glow
        - Otherwise, the dog will not glow
    - The dog will gain 1.5 inches of height for every 10 units of nutrients the treat has
    - After the treat is eaten, print out the current pantry
        - (Ex: "Current Pantry: Purina, Treat3")
    - Prints out the total sum of nutrients after the treat is eaten
        - (Ex: "Total Nutrients: 24)
    - Returns the eaten treat
        - If none of the treats in the pantry matches the shape indicated, return null
- `toString()`
    - Returns a String representation of the Doggy
    - The String should be formatted as follows:
        - If the doggy is glowing, return "I am the [furColor] doggy named [name]! I am currently [height] inches tall."
            - Ex) "I am the Red doggy named Clifford! I am currently 45 inches tall."
        - If the doggy is not glowing, return "Ruff! I am [furColor] [name]!"
            - Ex) "Ruff! I am Red Clifford!"
- **Include getters for all variables and setters only if needed by class Treat**

`Treat.java`

This class represents a treat that a dog will eat.

Variables

All variables must not be 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 Treat class must have these variables:

- **brand** – the name of the Treat's brand which is represented by a String value
- **shape** – the shape of the Dog Treat which is represented by a String value
- **nutrients** – the number of nutrients in the Dog Treat represented as an int
- **happy** – boolean representing if the treat makes dogs happy

Constructors

You *must* use constructor chaining for your constructors. The constructors *must* take the variables in the specified order.

- A constructor that takes in the Treat's brand, shape, nutrients, and happy and sets all the variables appropriately.
- A constructor that takes in the Treat's brand and nutrients, and uses the default values for all other variables.
    - shape: Circle
    - happy: false
- A constructor that takes in no parameters but has the following default values:
    - brand: Bits & Bytes
    - shape: Circle
    - nutrients: 10
    - happy: false

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 modifier to be used outside of this class.

- `puppucino(Doggy dog)`
    - If the treat has less than 5 nutrients: print "Let us go to Starbucks, [dog's name]!"
    - Otherwise print "[dog's name], the [brand] treat is good for you." and set dog's glow to false.
    - Does not return anything
- `birthday(Doggy dog)`
    - If the dog's height is at least 48 inches and the dog is glowing, print "Happy birthday, [dog's name]!!!" and set the Dog's fur color to be "Glittery purple"
    - If the dog's height is less than 48 inches, print "[dog's name] is still a Puppy."
    - Otherwise, print "Today is not your birthday, [dog's name]."
    - Hint: This is a class method
    - Does not return anything
- `toString()`
    - Returns a String representation of Treat
    - The String should be formatted as follows:

    - If the treat makes dogs happy, return "I am a [brand] dog treat in the shape of [shape] with [nutrients] nutrients. I make dogs happier!"
    - Otherwise, return "I am a [brand] dog treat in the shape of [shape] with [nutrients] nutrients."
- **Include getters for all variables and setters only if needed by class Doggy**

`LunchTime.java`

This class acts as a driver class, meaning it will contain and run Doggy and Treat objects and "drive" their values according to a simulated set of actions.

Methods

- Main method must act as such:
    - Must create at least 4 Doggy objects:
        - 1 Doggy using the default constructor
        - 1 Doggy with the 3 arg constructor and the following values
            - `name: Sideways`
            - `furColor: Gray`
            - `glow: false`
        - 1 Doggy with the following values
            - `name: Spot`
            - `furColor: Black`
            - `glow: true`
            - `height: 50.5`
        - 1 Doggy with the following values
            - `name: Luna`
            - `furColor: Brown`
            - `glow: true`
            - `height: 175.8`
    - Must create at least 4 Treat objects:
        - 1 Treat using the default constructor
        - 1 Treat with the 2 arg constructor and the following values
            - `Brand: MilkBoolean`
            - `nutrients: 8`
        - 1 Treat with the following values
            - `brand: Puperoni Bytes`
            - `shape: Circle`
            - `nutrients: 2`
            - `happy: true`
        - 1 Treat with the following values
            - `brand: Ducky Snacks`
            - `shape: Square`
            - `nutrients: 0`
            - `happy: false`

- Perform the following operations in this order:
  - Print doggyPopulation
  - Add all Treats that were created into the treatPantry
  - Call goOutside() on Sideways
  - Call eatTreat() on Spot with Circle as the shape
  - Call birthday() on Ducky Snacks and Clifford
  - Call puppucino() on Bits & Bytes and Luna
  - Call toString() on all Doggys and Treats and print out their respective statements

Output:

```
4
Current Pantry: Bits and Bytes
Total Nutrients: 10
Current Pantry: Bits and Bytes, MilkBoolean
Total Nutrients: 18
Current Pantry: Bits and Bytes, MilkBoolean, Puperoni Bytes
Total Nutrients: 20
Current Pantry: Bits and Bytes, MilkBoolean, Puperoni Bytes, Ducky Snacks
Total Nutrients: 20
I am having way too much fun!
Current Pantry: MilkBoolean, Puperoni Bytes, Ducky Snacks
Total Nutrients: 10
Today is not your birthday, Clifford.
Luna, the Bits and Bytes treat is good for you.
Ruff! I am Red Clifford!
I am the Gray doggy named Sideways! I am currently 106.5 inches tall.
Ruff! I am Black Spot!
Ruff! I am Brown Luna!
I am a Bits and Bytes dog treat in the shape of a Circle with 10 nutrients.
I am a MilkBoolean dog treat in the shape of a Circle with 8 nutrients.
I am a Puperoni Bytes dog treat in the shape of a Circle with 2 nutrients. I make dogs happier!
I am a Ducky Snacks dog treat in the shape of a Square with 0 nutrients.
```

Rubric

[40] `Doggy.java`

- [5] Variables all have the correct declaration and type
- [10] Constructors are all written correctly
  - [5] All constructors are present and take the correct number of arguments
  - [5] Constructor chaining is used
- [25] Methods are implemented correctly
  - [5] `goOutside()` is implemented correctly
  - [5] `addTreat(T)` is implemented correctly
  - [5] `eatTreat()` is implemented correctly
  - [5] `toString()` is implemented correctly
  - [5] Getters and setters are implemented correctly for each variable

[40] `Treat.java`

- [5] Variables all have the correct declaration and type

- [10] Constructors are all written correctly
    - [5] All constructors are present and take the correct number of arguments
    - [5] Constructor chaining is used
- [25] Methods are all implemented correctly
    - [7] `puppucino()` is implemented correctly
    - [7] `birthday()` implemented correctly
    - [7] `toString()` is implemented correctly
    - [4] Getters and setters are implemented correctly for each variable

[20] `LunchTime.java`

- [5] Correct `Doggy` objects
- [5] Correct `Treat` objects
- [5] Correct testing of Treat objects and methods
- [5] Correct testing of Doggy objects and methods

Allowed Imports

To prevent trivialization of the assignment, you **cannot import any package**.

The Checkstyle cap for this homework assignment is 10. Up to 10 points can be lost from Checkstyle errors.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach. 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 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:

- Doggy.java
- Treat.java
- LunchTime.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 (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 a note containing all official clarifications