L14-L15 Homework: Clash of 1331

Problem Description

Hello! Please make sure to read all parts of this document carefully.

Welcome to Clash of 1331! this assignment, you will be applying your knowledge of class hierarchies, interfaces, and polymorphism to train an army of barbarians, archers, and golems.

Solution Description

You will write one interface, Treatable, and five classes: Troop, Barbarian, Archer, Golem, and Infirmary. Each will be written in its own file (named accordingly).

- You must use constructor chaining when it's applicable:
  - If a class has N constructors, it will use "this" constructor chaining N-1 times
  - A class will use "super" constructor chaining at most once. It may only be used in the constructor that has all the appropriate parameters. You will use it if and only if, in that constructor, the implicit "super" constructor chaining is not appropriate
- Visibility for required elements:
  - All classes, methods, and constructors must be public
  - All variables must be private
- You may add to aid you (if you want)
  - Methods, if they are private
  - Class constants (static final variables), if they are private and their type is a primitive type, a wrapper class, or String.
- You may not add any other classes or constructors

You will also be provided with an additional class, TroopConcrete (in an appropriately named file). This class is used for autograding purposes, as we need a way to test the class that we cannot directly instantiate. **You need to familiarize yourself with this class, as it will help you understand autograder error messages related to the type which it is replacing**. This file is **error-free, and error messages related to it are actually errors about the related type**. You must not change it in any way, and you must submit it along with your other files in your submission.

<u>Treatable</u>

- This is an interface that guarantees an object is `Treatable`. It will have:
  - `boolean needsTreatment()` abstract method that will indicate whether a Treatable needs treatment
  - `void treat()` abstract method to treat a Treatable

Troop

- This abstract class represents a troop in our army. It will have:
  - The following instance variables: `name` (`String`), `experienceLevel` (`int`), `health` (`int`).
    - The `experienceLevel` should always lie between 1 and 50. **Any time it will be set to less than 1, it should instead be set to 1, and any time it will be set to more than 50, it will be set to 50**. Tip: always use the setter, which has this behavior.
    - The `health` should always lie between 1 and 100, with same behavior for setting the value as the above
      - **These instructions apply to the entire HW – every indicated change in them comes with an implied bounds check and adjustment as needed**
  - 3-arg constructor taking the instance variables in the order above and setting them
  - `void trainWith(Troop p)` abstract method to have a troop train with another to gain experience
  - `toString` method with the following format:
    - "My name is [`name`], my experience level is [`experienceLevel`], and my health is [`health`]" (without square brackets)
  - `equals` method (from the perspective of Troop, a Troop is equal to another if they have the same `name`, `experienceLevel`, and `health`)
  - Getters and setters for all instance variables
    - The setter for `experienceLevel` and `health` should follow the rules listed on the top for setting the values.

Barbarian

- This class represents barbarians in the army. A `Barbarian` is a Troop and is `Treatable`. It will have:
  - The following instance variables: `isElite` (`boolean`). This attribute will never change and should be marked accordingly.
  - 4-arg constructor with `name`, `experienceLevel`, `health`, and `isElite`
  - 1-arg constructor taking `isElite`, for barbarians with name "Buzz", `experienceLevel` of 1, and 25 of `health`.
  - `boolean isElite()` method returning the `isElite` value.
  - `void trainWith(Troop)` method with the following behavior:
    - A `Barbarian` cannot train if any of these conditions are met:
      - their `health` is below 10 or is exactly 100
      - they are asked to train with someone other than a Barbarian or an Archer
      - their `experienceLevel` is the maximum
    - If any of the conditions are met, the method will do nothing. Otherwise…
    - The Barbarian enjoys training with another `Barbarian`. However, injuries may be sustained from training.
      - If the `Barbarian` trains with another `Barbarian` decrease both barbarian's health by 0.1 * (the `experienceLevel` of the Barbarian that is the parameter to the method). Round the damage taken down before subtracting it.

- Print out "AAAARGH! I just trained with a level [other `Barbarian`'s `experienceLevel`] barbarian, and my health went from [old health] to [new health]" (without square brackets) in its own line.
- After that, increase both Barbarian's experience levels by 5 if none of the barbarians is an elite, and by 8 otherwise

▪ The Barbarian strongly dislikes training with an `Archer` but loves the color purple!
- If the `Barbarian` fights with an `Archer` with purple hair:
  - Increase the `Barbarian`'s health by 10.
  - Print out "YAAARG. My health increased from [old health] to [new health]" (without square brackets) in its own line.
  - Increase the experience level of the barbarian and archer by 1.
- If the `Barbarian` trains with an Archer with another hair color, decrease the Barbarian's health by 15.
  - Print out "AAAARGH! I hate that color!" in its own line.

o `void treat()` method that increases the health by 5
o `boolean needsTreatment()` method. A `Barbarian` needs treatment if their health isn't the maximum
o `void scream()` method that prints out "AAAARGH!" in its own line.
o `toString` method with the following format (without square brackets):
▪ If the `Barbarian` is an elite `Barbarian`:
- "My name is [`name`], my experience level is [`experienceLevel`], and my health is [`health`]. I am an elite barbarian"

▪ Otherwise:
- "My name is [`name`], my experience level is [`experienceLevel`], and my health is [`health`]. I am a regular barbarian"

▪ You must call the super implementation of `toString`
o equals method (a `Barbarian` is equal to another if they have the same `name`, `experienceLevel`, `health`, and `isElite` value)
▪ You must call the super implementation of `equals`

<u>Archer</u>

- This class represents archers in the army. An `Archer` is a Troop and is `Treatable`. It will have:
  - The following instance variables: `hairColor` (`String`).
  - 4-arg constructor with `name`, `experienceLevel`, `health`, and `hairColor`
  - 1-arg constructor taking `hairColor`, for archers with name "Sally", `experienceLevel` of 10, and 15 of `health`.
  - Getter and setter for hairColor
  - `void trainWith(Troop)` method with the following behavior:
    - An `Archer` cannot train if any of these conditions are met:
      - their `health` is below 5 or is exactly 100
      - they are asked to train with someone other than a Barbarian or an Archer
      - their `experienceLevel` is the maximum
    - If any of the conditions are met, the method will do nothing. Otherwise…

        - The `Archer` is best equipped to train with another `Archer`. Specifically, an `Archer` with the same hair color loves training with other archers with that hair color.
            - If an Archer trains with another archer with a different hair color as them, increase both archers' experienceLevel by 2
                - Print out "Oof! I prefer training with other archers with the same hair color as me"
            - If an archer trains with another archer with the same hair color as them, increase both archers' experienceLevel by 4
                - Print out "I like training with other archers with the same hair color as me"
            - For both archers, decrease health by 5
        - The `Archer` doesn't like training with Barbarians
            - If the Archer trains with a `Barbarian`, decrease the `Archer`'s health by 10
            - Print out "Gross. Go away [`Barbarian`'s name]! I hate training with Barbarians!" (without square brackets)
    - `void treat()` method that increases the health by 3
    - `boolean needsTreatment()` method. An `Archer` needs treatment if their health isn't at least 80
    - `toString` method with the following format:
        - "My name is [`name`], my experience level is [`experienceLevel`], and my health is [`health`]. I am an archer with [hairColor] hair" (without square brackets)
        - You must call the super implementation of `toString`
    - equals method (an `Archer` is equal to another if they have the same `name`, `experienceLevel`, `health`, and `hairColor`)
        - You must call the super implementation of `equals`

<u>Golem</u>

- This class represents golems in our army. A `Golem` is a `Troop`. It will have:
    - The following instance variables: `weight` (`int`).
    - 4-arg constructor with `name`, `experienceLevel`, `health`, and `weight`
    - 0-arg constructor, for golems with name "Nelly", `experienceLevel` of 19, 80 of `health`, and 10 tons of weight (weight value is in tons, no conversion needed).
    - Getter and setter for `weight`
    - `void trainWith(Troop)` method with the following behavior:
        - A `Golem` cannot train if any of these conditions are met:
            - their `health` is below 15 or is exactly 100
            - they are asked to train with someone other than a `Golem`
            - their `experienceLevel` is the maximum
        - If any of the conditions are met, the method will do nothing. Otherwise…
        - Increase the experience level of the two golems by 3 and decrease their health by 12
    - `toString` method with the following format:

        - "My name is [name], my experience level is [experienceLevel], and my health is [health]. I am a golem that weighs [weight] tons" (without square brackets)
        - You must call the super implementation of `toString`
    - equals method (a `Golem` is equal to another if they have the same `name`, `experienceLevel`, `health`, and `weight`)
        - You must call the super implementation of `equals`

Infirmary

- Represents infirmaries to treat Treatable troops. It will have:
    - The following instance variables: `infirmaryName` (`String`).
    - The following class variables: `numberOfInfirmaries` (`int`). Should start at 0.
    - 1-arg default constructor taking the infirmary name and increasing the number of infirmaries
    - Getter for `numberOfInfirmaries`
    - `void inspectTroop(Troop)` method:
        - Prints the String representation of the troop to console in its own line
        - If the troop is a Barbarian, the Barbarian should also `scream` after that.
    - `void doTreatment(Treatable)` method:
        - If the instance doesn't need treatment, print in its own line "You are fine. You will not receive treatment at [infirmaryName] infirmary" (no square brackets)
        - Otherwise, treat the instance
    - toString with the following format: "[infirmaryName] infirmary" (no square brackets)
    - equals method (an `Infirmary` is equal to another if they have the same `infirmaryName`)

Testing Your Solution

You will notice that none of the classes have a `main` method. That is because not all classes have to be Java programs, and in this case, none are. To help test your implementations, you can create more classes in which you can create instances of your concrete classes, invoke methods to have them interact with each other, and print to console information that helps verify the correctness of your solution.

Rubric

[40] Type Review (it is very important for your submission that these tests pass. All of these are visible in every submission)

- [10] All types compile, all type headers are correct, all static variables are correct
- [5] Check Treatable
- [5x5] Check constructors (1), methods (3), variables (1) presence in each class

[12x5] Implementations in each of the student-written classes

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 anything.

Feature Restriction

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)

Checkstyle

The Checkstyle deduction limit for this assignment is 25 points, counting Javadoc errors. Refer to the course guide for the use of Checkstyle and review the autograder Checkstyle report on your submissions.

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].

If the autograder encounters issues processing your collaboration statement, begin it with "Collaboration Statement: ".

Allowed Collaboration

When completing homeworks for CS 1331 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

Upload the files listed below to the corresponding assignment on Gradescope:

- Treatable.java
- Troop.java
- TroopConcrete.java (provided but must also be submitted)
- Barbarian.java
- Archer.java
- Golem.java
- Infirmary.java

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.

<u>Important Notes</u>

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit .class files or .jar 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