L10-L11 Homework: Zoo

Problem Description

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

In this assignment, you will be applying your knowledge of writing classes, including visibility modifiers, constructors, methods, variables, accessors & mutators, constructor overloading, constructor chaining, and toString(), to begin writing your own classes with Object-Oriented programming! You will create Animal, Habitat, and Zoo classes that will be used to represent the current state of a zoo.

Solution Description

You will write 3 classes, each in their own file (named accordingly).

- You must use constructor chaining when it's applicable.
- Visibility for required elements:
  - All classes, constructors, and methods should be public.
  - All variables should be private.
- You may add to aid you (if you want):
  - Methods, if they are private.
  - Variables, if they are private static final, and their type is a primitive type, a wrapper class, or String.
- You may not add any other classes or constructors.

Animal

It represents animals in the zoo. The class will have:

- The following instance variables: `species` (`String`), `name` (`String`), `energy` (`int`), `health` (`int`).
  - energy and health should always be between 1 and 100. If they fall below 1, they should be set to 1, and if they go above 100, they should be set to 100.
- The following static variable: `numberOfAnimals` (`int`). It should start at 0 and increase every time an Animal instance is created.
- 4-arg constructor that takes the variables in the order above and sets them.
  - Note that energy and health might be out of the desired range, in that case, you need to set them to an appropriate value as described above
- 2-arg constructor that takes `species` and `name` (in that order) and creates an instance with 50 energy and 100 health.
- 1-arg constructor that takes a name and creates an instance with species "Unknown", 50 energy, and 100 health.
- Getters for all instance variables.
- Setters for `name`, `energy`, and `health`. For `energy` and `health` setters, make sure that the stored value is appropriate.

- Getter for `numberOfAnimals`.
- `void eat(int)` method that receives an amount of food and increases the energy of the animal by 2 times the food amount.
- `void doActivity(int, boolean)` method that receives the duration of the activity and whether it's dangerous. It will decrease the energy of the animal by 5 times the duration. Additionally, if the activity is dangerous, it will decrease the health of the animal by 3 times the duration.
- `void goToZooHospital()` method that sets the health of the animal to 100. Additionally, if the energy is below 60, it will set it to 60.
- `boolean isHungry()` method that returns whether the animal is hungry or not. An animal is hungry if their energy is below 50.
- `toString()` method to provide a `String` representation of instances of the class, with the format "I am [`species`] [`name`]. I have [`energy`] energy and [`health`] health" (without the brackets)

Habitat

It represents habitats in the zoo. The class will have:

- The following instance variables: `name` (`String`), `animals` (`Animal[]`), `animalCount` (`int`).
  - `animalCount` should always be equal to the number of animals in the habitat, which starts at 0 and changes when animals are added or removed from the habitat.
- 2-arg constructor that takes the habitat name and its capacity (int). It will set the name, and assign `animals` with an array such that the length matches the habitat's capacity. `animals` will start with null entries.
- `boolean isFull()` method that returns whether the habitat is full or it has space remaining for more animals.
- `int getCapacity()` method to return the maximum number of animals that can be at one time in the habitat.
- `boolean addAnimal(Animal)` method to add an animal to the habitat. If the habitat is already full or if the animal is null or already in the habitat (the instance is already in the array) the method should return `false`. Otherwise, the animal will be added to the habitat and the method should return `true`. The animal will be added next to the previous animal in the array (or at the beginning of the array if there are no animals yet).
- `boolean removeAnimal(Animal)` method to remove an animal from the habitat. If the animal is not in the habitat, the method should return `false`. Otherwise, the animal will be removed from the habitat and the method should return `true`. When an animal is removed from the habitat, the array should be adjusted so that all animals are next to each other in the array, by changing the location of animals added after the one removed by one. For example, if there are 5 animals and the one removed was at index 2, the one that was at index 3 will be moved to index 2, and the one that was at index 4 will be moved to index 3.
- `void feedAnimals(int)` method that receives an amount of food and makes all animals in the habitat eat that amount of food.
- `Animal[] getAllAnimals()` method that returns an array of all animals in the habitat (this should exclude the null entries in the `animals` array). Note that this should always return a new array instance – it cannot be the `animals` array, even if the habitat is full.
- `Animal[] getHungryAnimals()` method that returns an array of the animals that are hungry. Note that this should always return a new array instance – it cannot be the `animals` array, even if the habitat is full and all animals are hungry.

- `toString()` method to provide a `String` representation of instances of the class, with the format "[`name`] has [`animalCount`] [animal/animals] and has a capacity of [capacity]" (without the brackets, and replacing animal/animals with the correct one based on plurality of animalCount)
- Getters for `name` and `animalCount`. Setter for `name`.

Zoo

Zoo is a Java program (has a main method) that receives the state of the zoo at the beginning of the day from the console, does a few actions, and then prints the state at the end of the day to the console.

The program will receive the number of habitats and number of animals through console arguments, in that order (they are guaranteed to be positive integers). The console input will be a description of the habitats and animals (the inputs will always be valid format, with habitats having the two values and animals having the four values, and String values will be single-word). For example, if there are two habitats and three animals, the first line will be the first habitat, the second line will be the second habitat, the third line will be the first animal, the fourth line will be the second animal, and the fifth line will be the third animal. See the example input/output for format details.

After that, animals will be assigned to habitats using round robin assignment, skipping any habitats that are full when cycling over the habitats (for example, if there are three habitats with capacities 3 1 3, and there are 6 animals, the first animal goes to the first habitat, the second animal goes to the second habitat, the third animal goes to the third habitat, the fourth animal goes to the first habitat, the fifth animal goes to the third habitat, the sixth animal goes to the first habitat). It is guaranteed that the habitats will have a combined capacity for all animals in the zoo.

Once the assignment is complete, the program will print each habitat's String representation at the beginning of the day. Below each habitat will be the information of all the animals in the habitat, each on their own line. See the example input/output for format details.

After that, there will be two actions in the zoo for the day, which will happen in order:

- If the zoo has at least two habitats, the first habitat has at least one hungry animal, and the last habitat isn't full, the first hungry animal from the first habitat will be removed from the first habitat and added to the last.
- All animals in the last habitat will be fed food. Each animal will receive an amount of 10.

Once the actions are complete, the program will print the habitats with their animals once again. See the example input/output for format details.

Example Input/Output

Example Output 1 – User Input is Bolded. Your program should look exactly like this.
Command: java Zoo 3 6

```
H1 3
H2 1
H3 3
Fish Alice 70 80
Tiger Bob 40 50
Parrot Charlie 80 70
Flamingo David 45 65
Ostrich Erin 90 40
Fish Frank 60 60

Habitats and their animals at the beginning of the day:
H1 has 3 animals and has a capacity of 3
I am Fish Alice. I have 70 energy and 80 health
I am Flamingo David. I have 45 energy and 65 health
I am Fish Frank. I have 60 energy and 60 health
H2 has 1 animal and has a capacity of 1
I am Tiger Bob. I have 40 energy and 50 health
H3 has 2 animals and has a capacity of 3
I am Parrot Charlie. I have 80 energy and 70 health
I am Ostrich Erin. I have 90 energy and 40 health

Habitats and their animals at the end of the day:
H1 has 2 animals and has a capacity of 3
I am Fish Alice. I have 70 energy and 80 health
I am Fish Frank. I have 60 energy and 60 health
H2 has 1 animal and has a capacity of 1
I am Tiger Bob. I have 40 energy and 50 health
H3 has 3 animals and has a capacity of 3
I am Parrot Charlie. I have 100 energy and 70 health
I am Ostrich Erin. I have 100 energy and 40 health
I am Flamingo David. I have 65 energy and 65 health
```

Testing Your Solution

You can test Java programs (classes that have a main method) by interacting with the console and verifying that the interaction with the user is correct. For parts of the HW that are not covered by a Java program, you can write your own driver classes to test those components (do not submit any additional class you write).

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import **java.util.Scanner**. You are not allowed to import any other classes or packages.

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 10 points, not 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:

- Animal.java
- Habitat.java
- Zoo.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