Zoo Simulator

Nick Ryan

1 Introduction

Now that you fully understand inheritance, classes, and polymorphism, it's time to do more polymorphism! This homework should reinforce polymorphism ideas and work on developing your OOP class design skills.

2 Problem Description

You've been hired by The 1331 Zoo to create a zoo simulation they can use when visiting schools. This simulation isn't perfect or even that advanced. But it will give younger students a basic idea of the animals and what the animals will do when you try to interact with them.

3 Solution Description

Below are the requirements for each feature. We won't be holding your hand on this homework. It's up to you to decide the gritty details of each class and how they should be connected.

`Animal`

- Every animal must be able to `makeNoise`. They should return the string representation of the sound made by the animal.
- Every animal also needs to be able to `move` in a unique way. You should return a string representation of their movement.
- When you print out the animal it should return the animal's name;
- When you create a new Animal you should increment a counter which keeps track of the total number of Animals instantiated.

`Interfaces`

You'll need to create three interfaces to help distinguish the types of animals

- `Pettable`
  - This interface should have a variable `MAX_TIME_PETTABLE`. (For example you can set this to 10 if you're only allowed to pet an animal for 10 minutes before it's someone else's turn.) This variable should be the same across all instances of Pettable animals. This variable should help you enforce the amount of time you can pet the animal. You should utilize this variable when generating the random amount of time you pet the animal.
  - It should have a method `pet`.
- `Feedable`
  - This interface should have a variable `MAX_AMOUNT_OF_FOOD` which should help you enforce the amount of food you give an animal.
  - It should contain a method `feed`.
- `Viewable`
  - This interface should have a variable `PERCENT_OF_TIME_ASLEEP` which is the probability that the animal is asleep (we all know the large animals sleep most of the day).
  - We also need a `view` method.

Note: Variables declared in an interface are `final` and `static` by default. You should declare them as such yourself so that you don't forget about these traits.

2 `Pettable` Animals

The `pet` method should print out the amount of time you pet the animal, the sound it makes (should be calling makeNoise), and the animal should move closer to you.

You'll need two classes which implement the Pettable interface. The two classes you should create are:

- Sheep
  - `pet`: A sheep really doesn't care about you. While you're petting it there's a 50% chance that it'll walk away before you're done petting. You should inform the user that this happened.
- Pony

2 `Feedable` Animals

The `feed` method should print out the amount of food, and the type of food (for example: a pony eats hay), that is fed to each animal. The amount of food should be randomly generated utilizing the `MAX_AMOUNT_OF_FOOD` variable. The animal should then respond by making it's sound and moving away from you.

You'll need two classes which implement the Feedable interface. The two classes you should create are:

- Pony

- Koi Fish
  - While trying to feed a particular Koi Fish, 25% of the time another fish may come eat the food. If this happens you should inform the user.

2 `Viewable` Animals

The `view` method should print out an action specific to that particular animal (for example: a bear could swim in a pool), we should also see what noise the animal makes, and how the animal is moving around it's enclosure (for example: the tiger is running across the field). However, if that animal is asleep then it should only print out "zzz".

You'll need two classes which implement the Viewable interface. The two classes you should create are:

- Bear
- Tiger
  - A Tiger is twice as likely to be asleep than a bear. How can you utilize the same `PERCENT_OF_TIME_ASLEEP` variable?

Custom Class You'll get to create one more class. This class must extend Animal but can implement any of the interfaces you'd like.

4 Provided Files

4.1 ZooSimulator

We have provided a ZooSimulator.java file which has some of the code you'll need. Make sure to go in and change the code where specified.
NOTE: THIS CLASS WILL NOT COMPILE INITIALLY. You'll need to create the other necessary classes and methods before it will compile. I recommend commenting out the parts that don't compile and then uncommenting when you've added that feature.

ADDITIONAL REQUIREMENTS/NOTES:

- The file contains comments pointing to the code you need to change.
- You can create any instance variables that you feel will help you complete the assignment.
- When you select to pet/feed/watch an animal, randomly select an Animal which implements the necessary interface.
- You'll need to javadoc the methods inside of ZooSimulator.java

5 Example Output

```
$ java ZooSimulator

Welcome to The Atlanta Zoo!

Would you like to:
1. List all of the animals
2. See an output of all animals and the sound they make
3. Pet an animal
4. Feed an animal
5. Watch an animal
6. Quit
Please enter one of the options above:
1

The animals found in this zoo are:
Sheep
Pony
Koi Fish
Bear
Tiger

Would you like to:
1. List all of the animals
2. See an output of all animals and the sound they make
3. Pet an animal
4. Feed an animal
5. Watch an animal
6. Quit
Please enter one of the options above:
3

You pet the sheep for 5 minutes.
Baaaaaaaaaaaaaaaaaaaaa
The sheep shuffles closer to you.

Would you like to:
1. List all of the animals
2. See an output of all animals and the sound they make
3. Pet an animal
4. Feed an animal
5. Watch an animal
6. Quit
Please enter one of the options above:
5

The tiger is asleep.
zzz

Would you like to:
1. List all of the animals
2. See an output of all animals and the sound they make
3. Pet an animal
4. Feed an animal
5. Watch an animal
6. Quit
Please enter one of the options above:
4

You feed the pony 5 pieces of hay.
whinny
The pony trots away.

$
```

6 Tips

- Start coding at the highest level class (Animal) and the interfaces. Then begin implementing each animal one at a time.
- Compile as you go. It can be difficult to debug hierarchies of classes.

7 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **100** points. Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar *.java
Audit done. Errors (potential points off):
0
```

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 the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **100** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

8 Javadocs

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

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

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

The relevant tags that you need to have 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 13.31
 */
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){
        ...
    }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class 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.

8.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.2.jar -j *.java
Audit done. Errors (potential points off):
0
```

There will be a javadoc cap of 20 points for this homework. Please javadoc and checkstyle as you go. Note that you can lose a total of 100 points just for style errors. Don't let this be you. Checkstyle as you go.

9 Turn-in Procedure

Non-compiling or missing submissions will receive a zero. NO exceptions

Submit all of the Java source files you modified and resources your program requires to run to T-Square. Do not submit any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

9.1 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1]Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!