Object-Oriented Pets

Due Tuesday, January 31 at 8 a.m.

CSE 1325 - Spring 2023 - Homework #2 - Rev 1 - 1

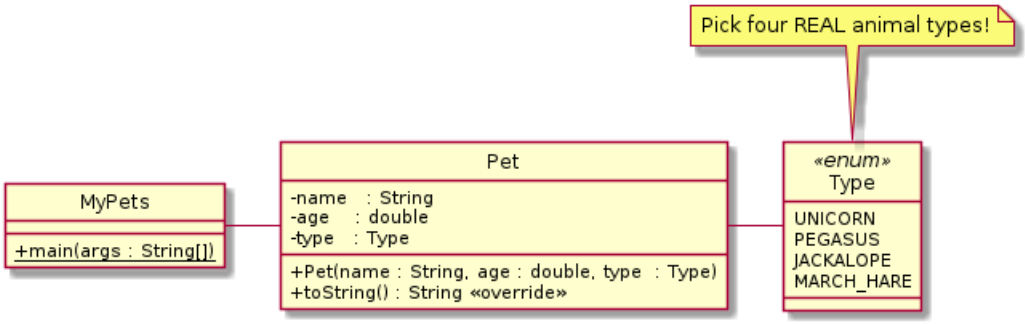
Assignment Overview

Throughout recorded history, humans have associated with other animals as companions for emotional support, social interaction, exercise, and protection. Some people take this to the extreme, and have trouble remembering the names of their menagerie. (The family across the street when I was a child had, as best we could count, 22 cats who came running when they heard the garage door open each evening. We sometimes prepared popcorn for the show.)

Let's help them by keeping a list of the pets' names, ages, and type.

Full Credit

In your git-managed directory **cse1325/P02/full_credit** (capitalization matters!), create public enum Type (in file Type.java) and public class Pet (in file Pet.java) as shown in the class diagram.



You may select any four (or more) types of REAL animals that you like for enum Type. See the collage below for some common companion animals, although you are not limited to the animals shown.

You may NOT add any getters to class Pet to retrieve fields. Instead, you may ONLY use the toString() method to format the fields for printing. The toString() method should return (as an example - but again, pick REAL animals)

```
Celestia is a unicorn age 110
```

Your main method must instance at least one of every Type of animal that your program supports, storing each such object in an array. Then, iterate over the array *using a for-each loop*, printing the text returned *implicitly* by each object's toString method (you may NOT call toString directly), one per line.

Screenshots are no longer required, but a working build.xml IS required. **Copy cse1325-prof/00/code_from_slides/build.xml to your full_credit, bonus, and extreme_bonus directories!**

Bonus

In Java, an enum is just a special class that also has enumerations. But unlike in C, our Java enums can define fields and methods and constructors as well! See the "Adding the Month Number" slide in Lecture 03 which discusses cse1325-prof/03/code_from_slides/date1/Month.java for a good example to help you implement the bonus.

enum Type

Add a private double field named `lifespan` representing how many years the pet will typically live.

Also add a constructor that accepts a double with which to initialize `lifespan`. Look up the typical lifespan for each type of animal your program handles and add parameters to each of your enumeration values that represent that type of animal's typical lifespan. (Be as accurate as you can, but we won't count off regardless of the value you choose.)

Finally, add a getter for field lifespan. This is just a double lifespan() method that returns the value of the field named lifespan. (Don't worry, Java can keep them both straight!) In effect, lifespan is now read-only - the value is set by the enum and can only be read by other code, never changed.

class Pet

Add a constant (public static final) for the human lifespan. Assume that the typical human lifespan is 80 years.

Also add a `public double humanAge()` method that converts the pet's actual age to its age "in human years" using the following equation. So, if a Unicorn lives 295 years and is currently 110 years old, its age would be 80 * 110 / 295 or 29.83 years

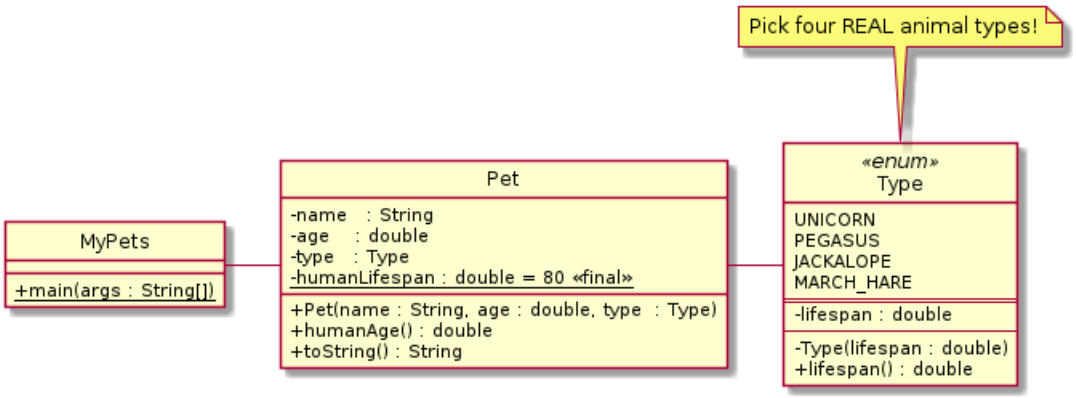
```
humanAge = humanLifespan * age / type.lifespan()
```

Finally, adjust Pet.toString() to include the human age of the pet as well, for example

```
Celestia is a unicorn age 110.000000 or 29.830509 in human-equivalent years
```

(But again, you must select REAL animals.)

An updated class diagram follows.



Add, commit, and push all files. Additional information that you may find helpful follows the Extreme Bonus.

Extreme Bonus

The size of an array must be specified when it is created. Rewrite your MyPets main method to accept *any* number of pets - at least until you run out of memory!

Hints!

The Professor's cse1325-prof

The professor for this class provides example code, homework resources, and (after the due date) suggested solutions via his cse1325-prof GitHub repository.

If you haven't already, clone the professor's cse1325-prof repository with "`git clone https://github.com/prof-rice/cse1325-prof.git`". This will create a new directory called "cse1325-prof" in the current directory that will reflect the GitHub repository contents.

The cse1325-prof directory doesn't automatically update. To update it with the professor's latest code at any time, change to the cse1325-prof directory and type "`git pull`". If that ever fails, just delete the directory and clone it again.

Suggested build.xml

Unlike the Makefile you may have used with C, a single build.xml will cover most (not quite all) of our projects this semester. It's listed here, although you can copy it to your directory from `cse1325-prof/00/code_from_slides` more easily.

IMPORTANT: You need one build.xml file for each directory from which you will run `ant`. For most assignments this semester, you'll need a copy of the same build.xml file in the full_credit, bonus, and extreme_bonus directories. Don't forget to add them to GitHub!

```
<?xml version="1.0"?>
<project name="CSE1325 Project" default="build">
  <presetdef name="javac">
    <javac includeantruntime="false" />
  </presetdef>

  <target name="build" description="Compile source tree java files">
    <javac debug="true" failonerror="true">
      <src path="."/>
    </javac>
  </target>

  <target name="clean" description="Clean output files">
    <delete>
      <fileset dir=".">
        <include name="**/*.class"/>
      </fileset>
    </delete>
  </target>
</project>
```

Common Companion Animals

Here are a few common companion animals collected from https://pixabay.com/photos. No attribution is required, and unlike my usual practice I made no attribution attempt since the selection is so broad! You are NOT limited to the animals shown here, however - be creative!

Full Credit

Formatting Doubles

Don't worry about formatting the animal's age. Whether your toString() method converts it to 110, 110.00, 110.0000, or whatever, we'll be happy with it for now. We'll worry about formatting numbers in a later assignment.

Get build.xml

Copy the build.xml file from cse1325-prof/00/code_from_slides to your cse1325/P02/full_credit directory. This should enable `ant` to compile and link your pet-related classes into .class files. Then run it using `java Pet`. Ensure that `ant clean` deletes all class files.

Write Type.java

This is a simple enum similar to what you would write in C, most likely a one-liner. Yes, even so, it should be in its own file!

Write Pet.java

Ensure that your class, constructor, and toString method are public and your 3 fields are all private. I recommend starting with the constructors (Pet has 1), followed by other public members (Pet has 1), and concluding with private members (Pet has 3).

The constructor need only assign each parameter to its respective private field. Remember that `this.name` (for example) refers to the *field* called name, while a plain `name` refers to the *parameter* called name. Data validation is NOT required yet.

Be sure to mark the toString method as @Override. While Java doesn't require this, it helps the compiler to check misspellings and so it is required for all CSE1325 assignments. Ensure that toString returns all 3 fields in String format, for example, "Fido is a dog age 4.2" (your text is permitted to vary).

Bonus

In **cse1325/P02/bonus**, duplicate your code from cse1325/P02/full_credit.

Modify your Type enum to include one private field (lifespan), one constructor (that assigns its parameter to lifespan), and one method lifespan() (that returns the value of the private lifespan field). This makes the lifespan field effectively read-only from outside the enum.

Your Pet class will be identical to the full_credit version except for the humanLifespan constant (called final static double) and the humanAge() method. See cse1325-prof/03/code_from_slides/date1/Month.java for a good example.

MyPets won't change at all.

Again, no later than once it compiles - add, commit, and push to GitHub!

Completing this task earns up to an additional 10 points.

Extreme Bonus

You *could* write code that fills up an array, then allocates a *bigger* array and fills that up, and so on until you run out of memory. But wouldn't it be cool if such a thing already existed?

The best code you'll ever write is the code you didn't *need* to write because it was in the library!

What you seek is a variable-length or dynamic array. Java has them. Google and StackOverflow are your friends. If you get stuck, ask - **we love questions!**

Completing this task earns up to 15 additional points, depending on how well you impress the grader.