Inheritance

1 Introduction

This assignment will cover inheritance, simple polymorphism, and simple lists. Make sure you review your notes, the lecture slides, and all the concepts presented on inheritance before you start. Be sure you understand what a class is vs. an object/instance. Make sure you understand what is-a means, and how we realize it in Java (i.e, what is the keyword that is used? How do you use it? etc.)

2 Problem Description

From: superchillboss@important.com
To: you@gatech.edu
Subject: Your First Project

Welcome to your internship at Important Inc.!
For your first project, I've gotta bunch of ideas and I figured you could use that inheritance and classes stuff you're learning and help us out.

Basically, I'm building a replacement for T-Square. No reason of course, T-Square is wonderful and never goes down or is slow or buggy or ugly. *Cough.* I'm coming up with a sort of add-on, that deals with some... slightly *different* aspects.
Right. Yes. So obviously I need to manage different people and how they're interacting with the site.

I've tried to break it down for you but definitely hit me up if you've got any questions.

We need a way to represent the following:

- a `Person`. This is just a basic type, obviously we'll have lots of specific "persons". People should have a first name, last name, and username, all of which are immutable.
- a `Professor`, which is-a `Person`, but additionally with a funniness factor as well as an average GPA rating. The funniness factor is just a score of how funny the professor is in lecture. Both of these can be `doubles`, just make sure the GPA stays with valid values, and

that the funniness factor is in the range $[0, 1]$. These change over time, so be sure to allow for that.

- a `Student`, also a `Person`. `Student`s have counts of how many hours they've spent studying vs. not studying. Add a way to get the study as a percentage (use a `double`) of total time, and make sure the hours can be added to but don't go below zero or decrease. The hours should start at 0 for every `Student`.
- a `GradStudent`, which is-a `Student`, except that no grad student's non-study time will ever be greater than zero. Make sure to enforce this.
- a `UndergradStudent`, which is-a `Student`. `UndergradStudent`s have an amount of hope remaining, which should start at 100 for every `UndergradStudent` and only be allowed to decrease, but must not go below zero. This is depressing, but thankfully undergrads also have an amount of pizza consumed count, which starts at zero and can only increase.
- a `TA`, which is-a `UndergradStudent`. `TA`s have a score for three categories: Piazza, Recitation, and Office Hours. These are also scores in the $[0, 1]$ range. These can change too.
- a `Course` has a title, a `Professor`, no more than 15 `TA`s, an additional Head `TA`, and no more than 300 `Student`s. Additionally, there should be a way to get the average values across all students for the study percentage. Finally, be sure it's possible to add students to the course.

Finally, write something to test your code. This could be JUnit Tests (if you know what those are), a fancy GUI, a complex simulator, or just a simple program , but make sure you test all of your code out thoroughly so I can see that and how it works.

Make sure you run that checkstyle thingy. Don't want your code looking ugly, now do we? And be sure to document your code very well. Other developers will be reading this, so they'll need to be able to understand what's going on.

Good luck!
Your really important but super chill boss,
George P. Burdell

3 Solution

Write out a diagram of the inheritance structure before you start the homework! Knowing how all the classes interact with each other will make the implementation much simpler!

Since this is our first inheritance homework, we'd tried to help lay out how to name everything. You'll still need to understand how to connect the classes together, but the following should help as reference:

3.1 `Person`

Data

Name	Type	Mutator	Accessor	Invariants	Description
firstName	`String`	✗	✓	None	The person's first name
lastName	`String`	✗	✓	None	The person's last name
username	`String`	✗	✓	None	The person's username

Methods

Name	Return Type	Parameters
getFirstName	`String`	None
getLastName	`String`	None
getUsername	`String`	None

3.2 `Professor`

Data

Name	Type	Mutator	Accessor	Invariants	Description
funnyFactor	`double`	✓	✓	$\in [0, 1]$	The professor's funny rating
gpa	`double`	✓	✓	$\in [0, 4]$	The professor's average GPA

Methods

Name	Return Type	Parameters
getGPA	`double`	None
getFunnyFactor	`double`	None
setGPA	`void`	`double newGpa`
setFunnyFactor	`void`	`double funnyFactor`

3.3 `Student`

Data

Name	Type	Mutator	Accessor	Invariants	Description
studyHours	`int`	✓	✓	≥ 0, can't decrease	hours the student has spent studying
nonStudyHours	`int`	✓	✓	≥ 0, can't decrease	hours the student has spent not studying

Methods

Name	Return Type	Parameters	Description
getStudyHours	`int`	None	
getNonStudyHours	`int`	None	
getStudyPercentage	`double`	None	
study	`void`	`int hours`	should add the hours to studyHours
relax	`void`	`int hours`	should add the hours to nonStudyHours

3.4 `GradStudent`

This one's a bit different. Everything is inherited, so think about the right way to enforce this new class invariant. Hint: you don't need any new instance data.

3.5 `UndergradStudent`

Data

Name	Type	Mutator	Accessor	Invariants	Description
hope	`int`	✓	✓	$\in [0, 100]$, cannot increase	The undergrad's hope remaining
pizza	`int`	✓	✓	≥ 0, cannot decrease	The undergrad's pizza consumed

Methods

Name	Return Type	Parameters	Description
getHope	`int`	None	
loseHope	`void`	`int hopeLoss`	should subtract hopeLoss from the hope
getPizza	`int`	None	
eatPizza	`void`	`int pizzas`	should add pizzas to the pizza count

3.6 `TA`

Data

Name	Type	Mutator	Accessor	Invariants	Description
piazzaScore	`double`	✓	✓	$\in [0, 1]$	Rating on Piazza
recitationScore	`double`	✓	✓	$\in [0, 1]$	Rating in Recitation
officeHoursScore	`double`	✓	✓	$\in [0, 1]$	Rating in Office Hours

Methods

Name	Return Type	Parameters
getPiazzaScore	`double`	None
getRecitationScore	`double`	None
getOfficeHoursScore	`double`	None
setPiazzaScore	`void`	`double piazzaScore`
setRecitationScore	`void`	`double recitationScore`
setOfficeHoursScore	`void`	`double officeHoursScore`

3.7 `Course`

Data

Name	Type	Mutator	Accessor	Invariants	Description
title	`String`	✗	✓	None	Course title, e.g ”CS 1331”
professor	`Professor`	✗	✓	None	The course’s professor
headTA	`TA`	✗	✓	None	The course’s head TA
tas	`TA[]`	✗	✓	None	The course’s TAs
students	`Student[]`	✗	✓	None	The course’s students

Methods

Name	Return Type	Parameters
getTitle	`String`	None
getProfessor	`Professor`	None
getHeadTA	`TA`	None
getTAs	`TA[]`	None
getStudents	`Student[]`	None
getAverageStudyPercentage	`double`	None
addStudent	`void`	`Student s`

4 Hints/Clarifications

This is probably a bit daunting - you’ve been presented with an obscure specification, with no “sample output” or idea of what things should look like when you’re finished. Thus, make sure you read the details very closely. If you truly understand the concepts presented in class, this homework may be arduous, but it shouldn’t be terribly difficult.

For every class you write, think about what information that class inherits. What do you have to rewrite? What don’t you? How do you communicate with the parent class? etc.

For constructors: remember that you’re responsible for getting information to your parent class.

For instance data: make sure to properly encapsulate and expose only via accessor methods

5 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **40** points. Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.1.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 **40** 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!

6 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.

6.1 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-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

7 Turn-in Procedure

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!

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!