Scheduling Assistant

Angie Palm

1 Introduction

Its often difficult to be able to keep track of your class schedule and all of your grades, so for this homework we are going to build the pieces of a class scheduler that you will be able to enter your classes and grades into to get an idea of how your schedule is shaping up.

You will be given a ClassPeriod.java interface and, you will be implementing this interface in each of the the following classes: Lecture, Lab, and Recitation. These classes will describe how different ClassPeriods work at Georgia Tech.

Do not forget to put the collaboration statement at the top of your submission. See the course syllabus for more details.

2 Problem Description

This homework will be a little different than the previous ones. Instead of us giving you a list of things to implement, we are going to describe what your classes actually need to model and it will be up to you to decide how to actually implement them. We are giving you an interface to implement, so that should give you a concrete place to start with for each class (remember what it means to implement an interface). However, each class will require slightly different constraints and requirements to make them function properly.

Read the descriptions carefully and first draw out how you will model your instance variables to match the description. After you do this, try and determine what methods are needed (in addition to those specified by the interface) to complete the functionality for each class; write out the method signatures for each before actually trying to code them.

There is more room here to be creative and interpret things in your own way. We will be grading for good object-oriented programming (Do you have appropriate instance variables/methods? Do you avoid repeating lots of code within the same class? Are you succinct with your variables and code?) and that each class meets the functionality described below.

We also are not giving you a driver this time around. We will be expecting you to test your code on your own. Make sure you test as you go as opposed to at the end. This will make it much easier to see where things go wrong.

Dont forget to read the very end of the assignment for information on Javadocing, and what we will be expecting of you for this assignment in that regard.

3 Solution Description

3.1 ClassPeriod Interface

This interface will have the following methods; remember that any class that implements the interface will need to have *all* of these methods at a *minimum*.

1. getSubject(): takes in no parameters, returns the subject of the ClassPeriod
2. getHours():takes in no parameters, returns the hours of the ClassPeriod
3. getGrade():takes in no parameters, returns the grade received in that classPeriod
4. getProfessor():takes in no parameters, returns the professor of the ClassPeriod
5. setProfessor(String professor): takes in one String parameter and sets the Professor of the ClassPeriod

3.2 Lecture Class

A lecture is the most basic of class periods. It must adhere to the contract that the ClassPeriod interface provides. You should be able to both set and get a grade for the Lecture. It is important to make sure that a grade for the lecture is an integer number between 0-100. In addition, each Lecture needs to have some way of saying whether or not it is mandatory to attend. When a lecture is created, we must only specify the number of hours, subject, professor, and whether or not lecture is mandatory. When we attempt to print out a lecture (what method specifies this?) it should look like the following:

```
CS 1331 taught by Professor Stasko.  It is a 3 hour course and you
received a 92.  Lecture is MANDATORY.
```

3.3 Recitation Class

A recitation has a teaching assistant, as well as a professor. Additionally, a recitation can either be collaborative or individual there is no in between. A recitation grade is an integer from 0-50 points, which will count as the overall grade for the recitation.

When a recitation is created, the professor, teaching assistant, hours, and subject are required to be specified, as well as whether or not the recitation work is collaborative. When a recitation is printed, it should look like this:

```
Recitation for SubjectName taught by ProfessorName.  The Teaching Assistant
is TAName.  This recitation is NumHours and your current grade is Grade.
The work in this recitation is COLLABORATIVE.
```

3.4 Lab Class

A lab is a little more complicated than a lecture or a recitation. In addition to the professor, a lab also has a lab assistant. A grade for a lab is determined based on a number of lab reports. Each lab report is graded out of 50 points and the overall grade for the lab is simply the average grade for

all of these reports.

There should be a way for someone to add a lab report grade to each lab individually. There should also be a way to retrieve all of the lab report grades for any given lab. Each lab will have 10 lab reports. There should be a way to calculate the grade in the Lab based on the individual lab reports (think about making a helper method that calculates the grade and sets it to an instance variable representing the overall grade! Do you think the method needs to be public or private?). If there are less than 10 reports currently in the lab, the grade will still be calculated as if there are 10 reports.

It is important to stay safe during lab! Each lab has a set of rules (modeled by Strings) that students need to remember and follow. A lab will always have no more than three rules and a student should be able to add them individually.

When a lab is created, the professor, lab assistant, hours, and subject are required to be specified. When a lab is printed, it should look like this:

```
Lab for SubjectName taught by ProfessorName.  The Lab Assistant is
AssistantName.  This lab is NumHours and your current grade is Grade.
The safety rules for this lab are:  1) SafteyRule1, 2) SafetyRule2,
3) SafetyRule3.
```

3.5 Student Class

A Student should contain all of his or her recitations, labs, and lectures. Someone utilizing your Student class should be able to retrieve each of these lists. One student can only hold a max of 21 hours at a time. Each Student should have the ability to add recitations, labs, and lectures individually, as in, you should have functionality to pass in either a single lab, recitation or lecture and have it be added to the appropriate list. The class period should not be added if it takes the student over their 21 hour maximum. The student should also be able to get the total number of hours that he or she is taking. A Student should also have a name and year (Freshman, Sopohomore etc...), in the form of String objects.

When a student is printed, it should look like this:

```
StudentName is taking TotalHours as a Year.
```

4 Tips, and Tricks

- Start this early (again)! We are getting into some of the most important topics of the course, it is imperative that you understand them completely!
- The first thing you should do for this assignment is to *draw everything out* on paper. Write down each of the classes along with the instance variables you think you'll need along with the methods. If you come into office hours looking for help designing this assignment, the TAs will be looking for your sketches as a starting point!
- Keep this assignment in mind as we move forward. A lot of the things we do here can be simplified with a topic we learn very soon (polymorphism). We will use this homework to illustrate the power in this important Object-Oriented concept.

5 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.

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 begin 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.

5.1 Javadoc and Checkstyle

Be sure you download the updated Checkstyle file (6.2.1) from T-Square along with this assignment. It will be used to verify both your Checkstyle and Javadocs. Javadocs will count towards your grade on this assignment.

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
```

6 Checkstyle

We have updated the Checkstyle .jar for this assignment. Please use the one attached to this assignment HW5 in T-Square along with these new instructions.

Checkstyle counts for this homework. You may ignore any errors in provided files (there shouldn't be any). You may be deducted up to 30 points for having Checkstyle errors in your assignment. Each error found by Checkstyle is worth one point. This cap will be raised next assignment. Again, the full style guide for this course that you must adhere to can be found by clicking **here**.

Come to us in office hours or post on Piazza if you have specific questions about what Checkstyle is looking for and how to fix Checkstyle errors.

First, make sure you download the `checkstyle-6.2.1-all.jar` from the T-Square assignment page. Then, make sure you put this file in the same directory (folder) as the `.java` files you want to run Checkstyle on. Finally, to run Checkstyle, type the first line into your terminal while in the directory of your Java files and press enter.

```
$ java -jar checkstyle-6.2.1.jar *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-6.2.1.jar *.java | wc -l
      2
```

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-6.2.1.jar *.java | findstr /v "Starting audit..." | findstr /v "Audit
    done" | find /c /v "hashcode()"
0
```

7 Turn-in Procedure

Submit your `Lecture.java`, `Recitation.java`, `Lab.java` and `Student.java` files on T-Square as an attachment (you may also submit `ClassPeriod.java` if you want to make our lives easier, but are not required to). Do not submit any compiled bytecode (`.class` files), or Checkstyle files. When you're ready, double-check that you have submitted and not just saved a draft.

8 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.)

[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 8PM Thursday. Do not wait until the last minute!

(c) Helps find last minute causes of files not compiling and/or running.