Homework 1: Classes
Due: 5:00 PM, Feb 8, 2019

Contents

Objectives

By the end of this homework, you will understand:

- when to use each of interfaces and abstract classes
- what Java types do and do not enforce within code

By the end of this homework, you will be able to:

- design a class hierarchy for a real-world situation

How to Hand In

In this assignment, there are a few Study Questions. These are for practice, so you do not need to submit anything for these questions. We will post the answers to the Study Questions on Piazza.

In order to hand in your solutions to these problems, they must be stored in appropriately-named files with the appropriate package header in an appropriately-named directory.

The source code files should comprise the hw01.src package, and your solution code files, the hw01.sol package. We mean that all your solution code should have a line at the top saying package hw01.sol; and they should be in the sol/hw01/sol/ directory. For this assignment there is no src code, but in the future, it must have package hw01.src; and be in src/hw01/src/. Note to use the current homework number instead of hw01.

Begin by copying the source code from the course directory to your own personal directory. That is, copy the following files from /course/cs0180/sol/hw01/sol/*.java to ~/course/cs018/workspace/javaproject/sol/hw01/sol:

- BasicTest.java

Do not alter these files!

After completing this assignment, the following solution files should be in your ~/course/cs0180/workspace/javaproject/sol/hw01/sol directory:

- Course.java containing public class Course
- Person.java containing public abstract class Person
Problem Setup

In an organization such as Brown, there seems to be some obvious hierarchy among people: there are full-time employees (including faculty, admins, facilities staff, etc), and there are students (including graduate and undergraduate). A hierarchy of classes seems a good way to capture the people at a university. But some subtle issues arise. For example, instructional staff will need the ability to store and view grades. Clearly, faculty are instructional staff, but so are TAs. TAs are students, however, and students in general shouldn’t be entering grades. This assignment uses the Faculty, Students, TAs, and grading as an example to help you understand the roles of interfaces, abstract classes, and types in Object Oriented design.

Note: We have you develop the code for this assignment in stages. You will turn in one set of files with your cumulative work on all tasks. You do not need to maintain versions of your code from each task separately.

Note: You will see several study questions in the handout. You are not required to submit answers to these, but you should have figured out answers to them, because they check how well you understand the concepts we are studying.

Problems

Courses

Start by creating a class name Course with fields for the department (a string), courseNumber (integer), and number of credits the course gives (a double that should be either .5, 1, or 1.5).
For this class only, make two constructors. The first should simply require all three fields as inputs. The second should take only the department and course number, setting the credits to 1 (by default). Since most courses are worth 1 credit, this makes it easier to create courses with default information. The first constructor should throw an IllegalArgumentException if the number of credits is not one of the expected values.

Task: Define the Course class as just described.

Faculty and Students

Now create a simple class hierarchy for people at a university. Every Person is either a full-time Employee or a Student. To keep the assignment from getting tedious, the only full-time Employees we'll capture are Faculty. We won't distinguish between undergraduate and graduate students. In other words, we have the following hierarchy:

Note that students in this hierarchy take only one course course at a time (because we haven't yet learned how to do lists in Java). The course and grade fields should be initialized to null as part of declaring the field within the class. You can do this by writing something like:

```
Course taking = null;
```

when you set up the field. Fields that are initialized in this way do not need to be arguments to the constructor.
Task: Create a class hierarchy (classes, abstract classes, and interfaces, as needed) to capture the class hierarchy in the diagram. Include fields that capture the information in the diagram. Include constructors where appropriate.

Task: Define a method register for students that takes a course and stores it in the taking field if the student is not already taking another course. Raise a RuntimeException containing the word “course” if the student is already taking something else.

Task: Define a method teach for faculty that takes a course and stores it in the teaching field. Raise a RuntimeException containing the word “course” if the professor is already teaching something else.

Study Question: Since both faculty and students are involved with Courses (teaching or taking) and the two methods you just wrote seem similar, should we have a shared course field in either the Person class or in an abstract class over students and faculty?

Study Question: Why are we creating the register and teach methods, instead of just requiring courses to be passed in as part of the constructors for students and faculty?

TAs

Task: Augment your work to also include TAs (the class name should be singular TA). TAs are students, but they also have a field named assisting that stores which course the TA is currently helping on. Since TAs are only set up when they are known to be assisting courses, we can take the course that a TA is assisting as part of the constructor for a TA.

Note: The TA’s register method should not allow a TA to register for a course they are assisting in.

Grading

Now, we want to extend our classes to support grading. Thinking ahead to when students might be enrolled in multiple courses, we want to store a grade along with the course that the grade was for.

Task: Create a class called GradeReport that has two fields: forCourse and grade. For now, let the grade be just an integer. The constructor should take both fields as inputs.

Task: Add a field grades to Student, which will hold a single GradeReport. This field should be initialized to null when creating Students.

Now, we need to let faculty and TAs assign grades to students.

Task: Add a method called giveGrade to the Student class. This method should take the Faculty or TA who is giving the grade, the Course that the grade is for, and a numeric grade. The method should create a GradeReport and store it in the students grades field, as long as the student is taking the given course and the given Faculty/TA is working on the course. If either of these conditions fails, the method should throw a RuntimeException containing the word “course” (e.g., “instructor is not teaching the course”). If the conditions hold, return the newly created GradeReport.

Figuring out how to define giveGrade to take only faculty or TAs as the person assigning the grade
is part of what we want you to think through for this assignment.

You are welcome to create additional methods in the class hierarchy if needed to write this in good OO style.

**Study Question:** What can go wrong if we use Person as the type of the grade-assigner in giveGrade?

**Enriching Grades**

Our initial setup has simple numeric grades, but in reality there are many forms of grades: numeric grades, S/NC grades, letter grades, written performance reports, and so on. We therefore want to relax our notion of grades to support more options.

For this assignment, we will support two kinds of grades: letter grades (one of “A”, “B”, “C”, “NC”) and S/NC grades, which are captured with two booleans: one for whether the student passed, and another for whether the student has passed with distinction. The distinction boolean should only be true if the passed boolean is also true.

**Task:** Define classes for LetterGrade and SNCGrade, with the components indicated above. The constructor for SNCGrade should take in a string with three possibilities: “SDIST”, “S”, or “NC” (from which you set the fields). Each grade class should have a method isPassing, which takes no input and returns a boolean indicating whether the grade reflects the student having passed. For letter grades, any grade other than “NC” is a passing grade.

**Task:** Modify GradeReport and giveGrade so that grades are now either letter or S/NC grades. We no longer want to support numeric grades (we just used those to get you started more easily).

**Task:** Add a method creditsEarned to the Student class, which takes no input and returns a double. If the student has a GradeReport and has earned a passing grade, this method returns the number of credits for the course. If the student has no GradeReport, the method should raise a RuntimeException containing the string “grade”. If the student has a GradeReport but didn’t pass, the method should return 0.

Your challenge here is to write this in good OO style, meaning that you do not dig into fields further than calling methods on them. Create additional methods in your classes as needed to satisfy this requirement.

**Task:** Only professors are allowed to view students grades; furthermore, they can only view grades of students in their own courses. Add a method to the Faculty class called viewGrade that takes a Student and returns a string (use RuntimeExceptions for illegal cases). If the student is in the professors course and has a GradeReport stored, the method returns a string indicating the grade earned. Otherwise, the method should raise a RuntimeException. The exception should contain the string “enrolled” if the student is not in the professors course, or contain the string “report” if the student has no GradeReport.

**Hint:** You will want to add a method to each class that produces a string version of the grade. In Java, every class as a default method named toString that shows how to display that class as a string (by returning a String—note the capital letter). Write toString methods for each Grade class. For letter grades, just return the string value. For S/NC grades, return one of “NC”, “S”, or “S*” (for S with distinction).
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/cs018/feedback](http://cs.brown.edu/courses/cs018/feedback)