AP Computer Science
Curriculum Module: Gradebook Project

Jill Kaminski
Chaparral High School
Parker, Colorado

Gradebook Project

Project Description

Electronic gradebooks are used by teachers to store grades on individual assignments and to calculate students' overall grades. Perhaps your teacher uses gradebook software to keep track of your grades.

Some teachers like to calculate grades based on what is sometimes called a "total points" system. This is the simplest way to calculate grades. A student's grade is the sum of the points earned on all assignments divided by the sum of the points available on all assignments. For example, suppose that a computer science class has a set of programming projects, quizzes, and tests, and that the sum of the points of all of these assignments is 485 for a semester. A student earns 412 of those points. His grade is 412/485, or about 84.95 percent.

In a total points system, teachers have to carefully figure out how many points each assignment should be worth, in order to make sure that certain assignments don't dominate the students' grades. Some teachers don't like to worry about the point value of each assignment, and they prefer what is sometimes called a "weighted category" system.

In a weighted category system, the teacher defines two or more categories and assigns a percentage weight value to each category. The sum of the weights adds up to 100. Grades are calculated by taking the average score of assignments in each category, multiplying each of those numbers by the weight for its category, and adding those values together.

For example, the computer science class described above could grade with a weighted category system by defining three categories: programming projects, quizzes, and tests. The teacher decides that programming projects should be worth 45 percent, quizzes 10 percent, and tests 45 percent. A student's average on programming projects is .95, on quizzes is .92, and on tests is .85. The student's grade is:

```
PROJECTS   QUIZZES    TESTS    GRADE
(.95 × 45) + (.92 × 10) + (.85 × 45) = 90.2
```

Each value in parentheses is the portion of the total grade for one category.

In this project, you will complete several classes to create an electronic gradebook that can calculate grades using either the total points system or the weighted category system. You will also write a client class to test your other classes. Your client class can even be written to store your own grades in one or more of your classes at school!

Topics addressed in this project include:

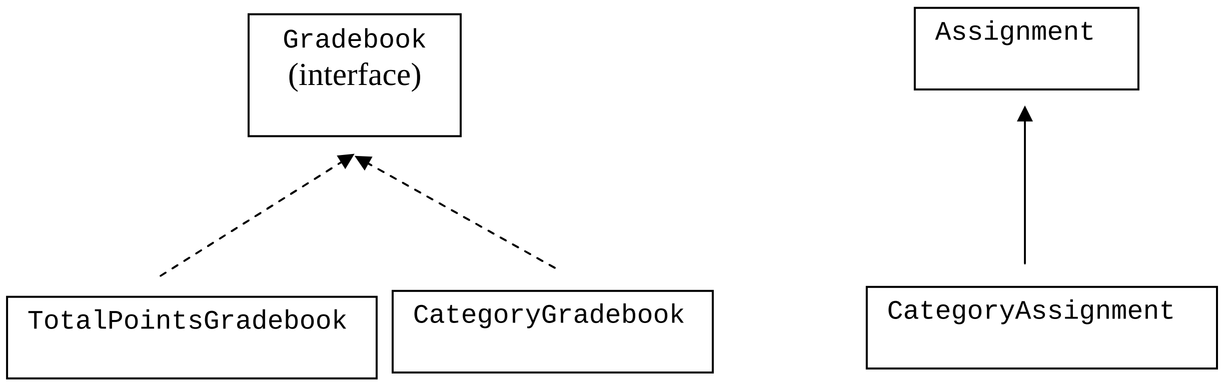
- Interfaces
- Inheritance
- Polymorphism
- Arrays
- ArrayLists

Design Description
Now that you understand the basics of gradebook software, let's take a look at the basic classes that will be used in this project. They have been designed for you, but it's important that you understand the design before you attempt to write the code.

Here are the classes used in this project, and the methods defined in each class:

- Since we're going to make more than one kind of gradebook, we will use an interface named `Gradebook`. The interface contains two methods: `calculateGrade`, which calculates a student's grade according to the rules for the implementation of the `Gradebook`, and `add`, which adds a new assignment to the `Gradebook`.
- We have a `TotalPointsGradebook` class, which implements `Gradebook` and uses the total points system of calculating grades. This is the simpler implementation.
- We also have a `CategoryGradebook` class, which implements `Gradebook` and uses the weighted category system of calculating grades. This is the more difficult implementation.
- We have an `Assignment` class. An `Assignment` has a name, a number of points possible for the assignment, and a number of points earned for an assignment. The `TotalPointsGradebook` class has an `ArrayList` containing `Assignment` objects.
- We have a `CategoryAssignment` class. The `CategoryAssignment` class extends the `Assignment` class. It has everything that the `Assignment` class has. In addition, it has the name of the category to which this `Assignment` belongs. The `CategoryGradebook` class has an `ArrayList` containing `CategoryAssignment` objects.

Here is a diagram of the design of this project:



Teacher Materials
Design Quiz or Discussion Questions

1. Why is an `ArrayList` used to store assignments in the `TotalPointsGradebook` and `CategoryGradebook` classes rather than an array?

   An `ArrayList` is used because we do not know in advance exactly how many `Assignment` objects will be added to the `Gradebook`. An `ArrayList` allows us to add objects as we need them.

   Actually, an array could be used, since all objects in the list are of the same type. But if we used an array, we would have to declare the length of the array to be larger than we ever anticipate needing, and we would have to keep an additional instance variable to represent the actual number of assignments that are currently in the `Gradebook`.

   If the number of assignments ever exceeded our estimate, we would have to declare a bigger array and copy all of the existing `Assignments` into the new array, or our program would crash. These restrictions make an `ArrayList` a better choice.

2. What are the benefits of using a subclass for `CategoryAssignments`? Why not just have `Assignment` and `CategoryAssignment` be similar but unrelated classes?

   The `CategoryAssignment` class needs to have all of the attributes and behaviors of the `Assignment` class, and in addition, will need one additional attribute (the name of the category to which the `Assignment` belongs) and one additional behavior (the ability to return the `Assignment` category). It's much better to use the `Assignment` class as a parent class than to reinvent the wheel and copy all of its attributes and behaviors to a separate class.

   If we make a separate and unrelated `CategoryAssignment` class, and later find we have a bug in the `Assignment` class, we have to remember to fix that bug in two different classes. If we discover later on that we have need of new kinds of `Assignments`, then we have to repeat the process, and still remember to put our bug fixes into each and every new class.

3. What are the benefits of using the `Gradebook` interface? Why not just have the `TotalPointsGradebook` and `CategoryGradebook` be similar but unrelated classes?

   The benefits of this design decision may not be obvious in the class design, or even in the code itself. The benefits are best seen in client code.

   For example, our client class can have an array or an `ArrayList` of `Gradebook` objects to represent a student's schedule of classes. If some classes use a total points system and others use a weighted category system, our array or `ArrayList` will still work.

Another example of the benefits of using interfaces (and also of using abstract classes or parent classes) can be seen when passing a parameter to a method. The formal parameter can be the interface (or abstract class, or parent class), but the actual parameter can be anything that implements or extends the formal parameter. This makes our code very flexible, and also allows for other classes that we may not have even thought of yet!

4. Which classes would you write first, and why?

   I'd suggest writing the `Assignment` class first, and then client code to test it. Then, write the `TotalPointsGradebook` and client code to test it.

   Then, write the `CategoryAssignment` class, as it extends `Assignment`. This is relatively simple, and `Assignment` has already been tested. Write client code to test `CategoryAssignment`. Finally, write `CategoryGradebook` and its client code.

Part 1: `TotalPointsGradebook`

Q: How do you eat an elephant?
A: One bite at a time.

Q: How do you eat a computer?
A: One byte at a time. ☺

Q: How should you write a long program?
A: A little at a time.

Emphasize to your students that it's much, much easier in the long run to break a long program into parts, and to code and test each part before moving on, than to try to write the whole program at once. For the first part of this program, students will write the `Assignment` class and client code to test it, and then write the `TotalPointsGradebook` class and client code to test it.

See the Part 1 folder for the necessary files. To help you differentiate instruction among your students of varying levels of ability, there are two versions of this part of the assignment: Level A and Level B. Level A is the more difficult version. The Part 1 folder contains two folders named Level A and Level B, and each of these folders contains all of the files that students need.

In Level A, students are given instance variables and method headings, but they otherwise write the `Assignment` class, the `TotalPointsGradebook` class, and the client code.

In Level B, students get some extra help. Since the `Assignment` class contains only a constructor and accessor methods and is not very difficult, students will complete that class just as the Level A students do. A client class that invokes the `Assignment` methods and tells students what results are expected has been provided for Level B. After the `Assignment` class has been tested, Level B students write most of the `TotalPointsGradebook` class, but are given some extra hints in the comments. The provided client class also tests the `TotalPointsGradebook` class; this code just needs to be uncommented.

At all levels, instructions in the form of postconditions are provided as comments in the various files. Students can, of course, reference the first two pages of this document, as well as the AP Computer Science Quick Reference (appended to the AP Exam and available at the AP Computer Science Course Home Pages (at
http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/8153.html and
http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html)
as needed, but they should not need additional resources in order to complete this project.

The instructions include vocabulary that is tested on the AP Computer Science Exams, such as **accessor method**, **mutator method**, **precondition**, **postcondition**, and **instantiation**. If your students are not familiar with this terminology, then this project provides a good opportunity for them to learn it.

Emphasize to your students that when code has been provided for them, they should not change it! The `Gradebook` interface given to students is complete and correct. I find that when I give my students code, they are very tempted to change that code when they find problems with the project. You can mitigate the effects of this by having a spare copy of the original files handy, and having students replace their “improvements” with the original when necessary. Explain to them that in the “real world,” software engineers get fired for changing code that works!

The classes will not compile until all non-void methods are written. If you want students to compile after writing each method, then have them write dummy return statements inside these methods, so that the class will compile before being entirely implemented.

Part 2: `CategoryGradebook`

On to Part 2! Students should have built confidence by successfully implementing and testing the `TotalPointsGradebook` at their appropriate level. Now it’s time to move on to the more challenging `CategoryGradebook`. There are three levels of differentiation available in Part 2.

At all levels, students should write and test the `CategoryAssignment` class before writing the `CategoryGradebook` class. I recommend that students use a new folder for this part, so that you can evaluate Part 1 separately from Part 2. They’ll need to put a copy of their `Assignment` class from Part 1 into the new folder.

In Level A, students will implement all methods. As before, they are given method descriptions in comments. Level A students will write the client class to test the `CategoryAssignment` class, and are given hints in comments to help them do this. In the `CategoryGradebook` class, Level A students are given neither the instance variables nor the parameter list for the class constructor. Additionally, they are not given the private helper method shown in the solution, which returns the index number of a given category. They are given a hint that if the `calculateGrade` method would benefit from a private helper method, then that method should be thoroughly documented.

The `CategoryAssignment` class given to Level B students is the same as for Level A, except that students are given the parameter list for the constructor. Students will implement all methods in this class. Level B students will also write the client class to test `CategoryAssignment`, but are given additional help in comments. These comments direct students to the client class for Part 1, and tell them to use this code as an example to write their test code for `CategoryAssignment`. Basically, they can use all of the `Assignment` client code, unchanged except that the object is a `CategoryAssignment` rather than an `Assignment`, and they'll add one extra method call to test the `getCategoryName` method.

Level B students get some extra help in the `CategoryGradebook` class. Specifically, they are given the instance variables and the private helper method `findCategory`. The constructor and the `add` method are the easiest places to start. Tell them that `findCategory` can be used to help `calculateGrade` do its job, and remind them that `findCategory` works, so they should not change it. Some Level B students might need help with `calculateGrade`. You can conduct a mini-class with these students and help them brainstorm a pseudocode solution. Help them along the way. I'd rather challenge students at this level a bit than put them in Level C, but I wouldn't want to frustrate them.

Level B students will also write the client code to test `CategoryGradebook`, again using the client code from Part 1 as an example. If you like, you could give them the client code, but I think that it's important that students get practice in both implementing classes and in writing client code for those classes.

Level C students are given the same `CategoryAssignment` class and client code instructions as Level B students, except that the instance variable is declared for them. At this point, even Level C students who are struggling should be able to complete a simple constructor and accessor method, and they should be able to refer to client code from Part 1 and test their class. They may just need a bit more help than their Level B counterparts.

Level C students are given quite a bit more scaffolding in the `CategoryGradebook` class. They will write the part of the constructor that initialized the instance variables, a line or two of code in the `calculateGrade` method (according to instructions in comments), and the `add` method. They are also given part of the client code to test the classes, but will complete it based on the example provided.

More Ideas

After both parts of this assignment are completed, additional client code can be written to create arrays or `ArrayLists` containing `Gradebooks`. These could be used to simulate either of the below:

- A teacher's gradebook for a single class of 30 students
- A student's set of grades for all of his current classes

This additional assignment is used to illustrate the benefits of using a `Gradebook` interface. We don’t have to know in advance what kind of `Gradebook` objects are in our collection, and no casting is necessary to find out the grade calculated by each of them. See Design Quiz question 3 for more information.

The `CategoryGradebook` in the solution does not calculate grades properly unless at least one `Assignment` in each category has been added. If your students’ code behaves similarly, then they could improve the gradebook so that if no points are available in a category, then the weight for that category does not count when calculating grades.