**Design ideas for an enhance Gradebook class.**

**Design Goals:**

* Represent a grade book like the one supported by Brightspace
* Any number of students
* Any number of assessments
* Each assessment has a numerical grade which may have a fractional part (use double)
* Each student has a “current grade”. This is the weighted average of all their assessment grades (see below).
* Each student has:
  + A unique student ID
  + First name
  + List name
* Each assessment has:
  + A unique name (e.g., “Chapter 1 quiz)
  + A weighting factor used to calculate the current grade.  
    Some instructors use “maximum points possible” as a weighting factor, although this design supports any weighting factors. The only attribute of these values is there value relative to each other. So these weighting factors are equivalent:  
    [10, 20, 20] and [30, 60, 60]  
    A weighting factor of zero causes the assessment to have no influence over the current grade.
* When calculating the **current grade** for a student, only assessments which have been given a grade are used in the calculation. If an instructor wants to have an untaken assessment counted as a zero, a grade of zero must be assigned to it.

**The Application Programming Interface (API):**

* The Gradebook object contains the grades for all students
* The Gradebook constructor: Gradebook(). We only need a default constructor
* Gradebook method:  
   addStudent(String stuID, String firstName, String lastName)
* Gradebook method:  
   addAsmt(String asmtName, double weightingFactor)
* Gradebook method:  
   setGrade(String stuID, String asmtName, double grade)
* Gradebook method:  
   double getGrade(String stuID, String asmtID)
* Gradebook method:  
   double getCurrentGrade(String stuID)

**Assumptions:**

All the addStudent’s and addAsmt’s are done before any grades are inserted via setGrade’s.

Reason: This simplifies the code because we can delay the allocation of the 2-dimensional arrays "grades" and "isGradeSet" until we know the dimensions. Until we know the actual number of students and the actual number of assignments. To relax this assumption, and allow clients to do more addStudents and/or addAsmts after doing some setGrade's, we would need to dynamically reallocate these 2-dimensional arrays (larger), and copy the old data to the larger arrays. All this is possible, just more complicated.

**Data Structures:**

Class Gradebook has a 2-dimensional array of grades, as doubles.

First asmt second asmt third asmt fourth asmt fifth asmt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 94.2 | 91.3 |  |  |  |
|  | 84.3 | 88.2 | 66.3 |  |
| 86.5 |  |  |  |  |

There is a row for each student.

The order of the assignments is the order of the addAsmt() calls.

The order of the students is the order of the addStudent() calls.

Class Gradebook has a 2-dimensional array of Booleans to indicate which grades have been set.

First asmt second asmt third asmt fourth asmt fifth asmt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True | True | False | False | False |
| False | True | True | True | False |
| True | False | False | False | False |

Class Gradebook has an ArrayList Student objects.

|  |
| --- |
| Student |
| Student |
| Student |
| Student |

Class Gradebook has an ArrayList Assessment objects.

|  |
| --- |
| Assessment |
| Assessment |
| Assessment |
| Assessment |

The Student class has these attributes:

* studentID (String)
* firstName (String)
* lastName (String)

The Assessment class has these attributes:

* name (String)
* weighting factor (double)

Here’s some code to illustrate how to use ArrayList

import java.util.ArrayList;

import java.util.ListIterator;

class TryArrayList {

public static void main(String[] args) {

System.out.println("Hello Java");

ArrayList aList = new ArrayList();

aList.add("Apple");

aList.add("Pear");

System.out.println(aList);

System.out.println(aList.get(0));

System.out.println(aList.get(1));

// System.out.println(aList.get(2));

// Try the ListIterator to iterate through elements

ListIterator listIter = aList.listIterator();

while (listIter.hasNext()) {

System.out.println(listIter.next());

}

}

}

**Possible enhancements**

* Calculate the average grade for the course (all the students).  
  getCourseGrade()  
  DONE by Dick Soderman
* Calculate the average grade for a given assignment  
  getAsmtGrade(String asmtName)  
  DONE by Mustafa
* Performance enhancement

You analyzed various sorting and searching algorithms as to their “performance”

For example, the bubble sort was: O(n2)  
That means that if you sort an array which is twice as big, it takes 4 times longer.

You also studied more **efficient** sort algorithms. Some were O(N \* log(N))

In the Gradebook class, methods getStuPos and getAsmtPos are O(N), where N is the number of students and number of assignments, respectively.

This can be improved. How? By using a Java “Map” object.  
A Map allows you to find an element in the Map in O(1) (constant) time.

We can recode the above methods to use a Map.  
This is fairly advanced, but you said you were going to study Maps in Java II, so you might be interested in adding this enhancement.

I have enhanced my solution to do this.

Let me know if you want to do this sometime.