**Recitation 6**

**Topics**

* Encapsulation
* Delegation
* Design interacting classes using
  + Association
  + Composition

In this lab you will be modeling a little part of the Poly CS lab administration.

You are being asked to write software so that the University programmers can model the way your lab workers keep up with your grades.

We've reduced this problem a great deal so that there is not too much for you to do.  
The complete set of code would be really long and would require days or weeks of analysis.

Remember that the focus of the lab is to practice encapsulation and delegation. The first means that you will keep all the data in your types private. The second means that if one object holds a piece of data and you want something done with that data, then you should ask the object to do it for you.

Each semester a set of lab sections is created and a lab worker is assigned to each lab.

***Lab workers*** have names (do you know your lab worker's name?).  
To make the problem less complicated, for this problem no lab worker will be assigned to more than one lab.  
Also assume that lab workers never make any mistakes when entering grades for students.

It's the worker's job to enter grades for their students each week and to printout a gradesheet showing all the students and their grades for each lab.

This is really only about lab grades, so here's the process you must implement in your classes**:**  
Grades should be kept separately for each week - so each week the lab worker adds the student's score for that week to the current grade record for the section he is teaching. Each ***student lab record***consists only of the student's name and their weekly grades. [Assume that no two students in the University have the same name (see, we reduced the problem even more!] The value -1 is used to indicate an absence from a lab - can you write the code so that everyone is absent before the first grade is entered? (this is something that is often done for consistency).

A ***lab section*** will be the "container" for student lab records in this problem.  
One lab section object will contain and manage the student lab records for a specific section. So there's a one-to-one connection between a lab worker and a section (because lab workers in this reduced problem work only one lab each). A lab section is kind of like an accounting package (like MS excel) in that the lab worker uses it to keep his students' grades. The University generates a file for each lab section that has the names of all the students who in that lab. Each lab section is "filled" with students from the corresponding file. There will be one student lab record for each student in the file and all the student lab records for a section will be stored in a lab section object. There's no maximum number of students in a lab section (so, of course, you will implement this as a vector of pointers to student records).

Each section also has a thing called a ***time slot*** which has the day of the week and the hour the lab section begins. Make this be a class with the day of the week being a string. and the hour an int (or more correctly an unsigned). For simplicity times are provided to the constructor in 24-hour time, i.e. 4pm would be represented as 16. For display, we show the times with AM or PM instead of 24-hour notation since most people find that more readable.

The programmers we are working for have requested that a section have a display function which will show the section name, the day and hour, the lab worker's name and a listing of all the students. (Normally the lab worker would be the only way to display the grade data for a section.) The way a section display's output will be slightly different than the output when a lab worker is asked to display the grades for their section [see the sample output below].

The programmers who hired you also want to reuse the Sections and lab workers after they have been created. The programmers (your bosses) want you to provide a reset function for the section object that clears out the data for the student records. The time slot remains the same (forever).

You need to enforce that no lab workers can be created that don't have meaningful names.  
You need to enforce that no sections can be created that don't have meaningful names and time slots.

You are writing code for the programmer who will use your classes to model the labs at Poly.  
We will provide you with a testing program that will show you what sorts of things need to be done and how the programmers you are working for want your classes to work**:** the programmer interface you must provide is shown in the test code.   
[On the upcoming test, you may get the same sort of thing: a main() function will be given to you and you'll have to write the classes that implement what's done in main(). Activities that happen in the "real world" (or some tiny version of it like the lab record keeping) will be modeled by a programmer using your classes.

**Possible scenario (the tester code implements this and its complete output is shown below) :**

Lab worker Moe is assigned to section A2 which meets on Tuesdays at 4:00p.m.  
The students are added. (Or read them in from A2.txt, attached below.)   
Lab worker Jane is assigned section B3 which meets on Thursdays at 11:00a.m.  
The students are added. (Or read them in from B3.txt, attached below.)  
Moe adds the first week's grades for the students that were there (and everyone was there!).  
In the first week of class Bob, Joe, and Sue earned 7 points and Tania earned 9.  
Moe shows us the grades for his section.  
At the end of the second week Moe enters grades for Tania (10), and Bob (10). Sue earned 0 points that week for chatting and email.  
Moe shows us the grades for his section at the end of week two.  
The modeler-programmer tests the reset for joe's section (A2).

**Developing your classes**

Try to work out how to do this on your own. But if you do get stuck on how to get started or how to proceed, you might want to look at  [rec05-guidance.](http://cse.poly.edu/cs1124/Recitations/Recitation05/rec05-guidance.html) (this page is not in Classes.)

In either case: DO NOT WRITE MORE THAN A FEW LINES OF CODE BEFORE YOU SAVE AND COMPILE

The main for the scenario above:

int main() {

// lab workers

LabWorker moe( "Moe" );

LabWorker jane( "Jane" );

// sections and setup and testing

Section secA2( "A2", "Tuesday", 16 );

//secA2.loadStudentsFromFile( "A2.txt" );

secA2.addStudent("John");

secA2.addStudent("George");

secA2.addStudent("Paul");

secA2.addStudent("Ringo");

cout << "\ntest A2\n"; // here the modeler-programmer checks that load worked

secA2.display(); // YOU'll DO THIS LATER AS: cout << secA2 << endl;

moe.addSection( secA2 );

moe.displayGrades(); // here the modeler-programmer checks that adding the Section worked

Section secB3( "B3", "Thursday", 11 );

//secB3.loadStudentsFromFile( "B3.txt" );

secB3.addStudent("Thorin");

secB3.addStudent("Dwalin");

secB3.addStudent("Balin");

secB3.addStudent("Kili");

secB3.addStudent("Fili");

secB3.addStudent("Dori");

secB3.addStudent("Nori");

secB3.addStudent("Ori");

secB3.addStudent("Oin");

secB3.addStudent("Gloin");

secB3.addStudent("Bifur");

secB3.addStudent("Bofur");

secB3.addStudent("Bombur");

cout << "\ntest B3\n"; // here the modeler-programmer checks that load worked

secB3.display(); // YOU'll DO THIS LATER AS: cout << secB3 << endl;

jane.addSection( secB3 );

jane.displayGrades(); // here the modeler-programmer checks that adding Instructor worked

// setup is complete, now a real world scenario can be written in the code

// [NOTE: the modeler-programmer is only modeling joe's actions for the rest of the program]

// week one activities

cout << "\nModeling week: 1\n";

moe.addGrade( "John", 7, 1 );

moe.addGrade( "Paul", 9, 1 );

moe.addGrade( "George", 7, 1 );

moe.addGrade( "Ringo", 7, 1 );

cout << "End of week one\n";

moe.displayGrades();

// week two activities

cout << "\nModeling week: 2\n";

moe.addGrade( "John", 5, 2 );

moe.addGrade( "Paul", 10, 2 );

moe.addGrade( "Ringo", 0, 2 );

cout << "End of week two\n";

moe.displayGrades();

//test that reset works // NOTE: can we check that the heap data was dealt with?

cout << "\ntesting reset()\n";

secA2.reset();

secA2.display();

moe.displayGrades();

} // main

The output would look something like this when you have completely finished all your classes:

test A2  
Section: A2; Tuesday 4PM  
Student: John; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: George; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Paul; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Ringo; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Moe has Section: A2; Tuesday 4PM  
Student: John; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: George; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Paul; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Ringo; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

test B3  
Section: B3; Thursday 11AM  
Student: Thorin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Dwalin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Balin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Kili; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Fili; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Dori; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Nori; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Ori; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Oin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Gloin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Bifur; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Bofur; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Bombur; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Jane has Section: B3; Thursday 11AM  
Student: Thorin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Dwalin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Balin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Kili; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Fili; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Dori; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Nori; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Ori; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Oin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Gloin; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Bifur; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Bofur; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Bombur; Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Modeling week: 1  
End of week one  
Moe has Section: A2; Tuesday 4PM  
Student: John; Grades: -1 7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: George; Grades: -1 7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Paul; Grades: -1 9 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Ringo; Grades: -1 7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Modeling week: 2  
End of week two  
Moe has Section: A2; Tuesday 4PM  
Student: John; Grades: -1 7 5 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: George; Grades: -1 7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Paul; Grades: -1 9 10 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  
Student: Ringo; Grades: -1 7 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

testing reset()  
Section: A2; Tuesday 4PM

Moe has Section: A2; Tuesday 4PM

The modeler programmer who wrote main would need to do more thorough testing by writing more code for more classes and more weeks, of course.  
You write the classes needed for this code to work as it should.

**SOMETHING YOU MAY NOT KNOW BUT NEED:**

* the vector type has a method named clear() that takes no parameters and releases all the data stored *inside* the vector.  
  After v.clear(), v.size() is 0.

**NOTES:**

* you may write the bodies of your class methods inside the class definitions - or - you may declare them in the class and define them after main
* when you reset a Section, should you free up all dynamic memory you used when you put the student records into the vector? Does this mean just clear()-ing the vectors or do you also have to free up the objects that you created on the heap?

**EXTRA:**

If you've finished this lab as above, try to implement this**:**   
In addition to displaying the student records in a section, write a function that the worker can use to display the average for their section for a specific week:

joe.displayAverageForWeek(2);

Average for section A2 for week 2 is: 45.34

(that may not actually the right value).