Recitation 5

Topics

- Design interacting classes using
  - Association
  - Composition
- Encapsulation and Data Hiding
- Delegation
- *If covered in lecture*, provide a destructor and copy constructor for the Section class.

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

You are being asked to write software so that the University programmers can model the way your lab instructors 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, data hiding and delegation. The first two are basic tenets of OOP. Delegation means that if one object holds a piece of data and you want something done with that data, then you *ask the object to do it for you.*

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

Lab instructors 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, so your code does not have to check for such a possibility.

It's the worker's job to enter grades for their students each week and to print out 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 or spreadsheet 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. How should we represent this? Clearly a vector is a good idea. And we *could* (but won't) make it a vector of student records. Why won't we? Well, for this lab, we want you to <u>allocate the student records on the heap</u>.

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.
[In the exams, 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.

The classes

Clearly you need a class to represent a Section and a class to represent a LabWorker. Also you will need a class to represent a TimeSlot and a class to represent a Student record. These last two classes are not directly accessed by the user's code, so can be embedded.

Association vs. Composition

Consider the relationships among our classes. A Section instance is *composed* of, among other things, a TimeSlot. The Section cannot exist without the TimeSlot. In fact, they both come into existence together and late cease existing together.

The bond or *association* between a Lab worker and a Section is quite different. They exist completely indepently and the bond can be made or broken at any point.

These two approaches to establishing connections between different kinds of objects are very common. The nature of an association, such as the spousal relationship from lecture, makes pointers an obvious tool for implementation. Composition, on the other hand, is more likely represented, as we have here, by specifying to type have a field of the other type.

The software design literature goes into much more detail about the different relationships that might exist between classes, but this will suffice for our needs.

Test code

Below is the main for a possible scenario. We have also provided the program as an attachment, with all of the code commented except for the print statements. That will hopefully motivate you to implement and test *incrementally*.

```
// Test code
void doNothing(Section sec) { sec.display(); }

int main() {

    cout << "Test 1: Defining a section\n";
    Section secA2("A2", "Tuesday", 16);
    secA2.display();

    cout << "\nTest 2: Adding students to a section\n";
    secA2.addStudent("John");
    secA2.addStudent("George");
    secA2.addStudent("Paul");
    secA2.addStudent("Ringo");
    secA2.display();
    cout << "\nTest 3: Defining a lab worker.\n";
    LabWorker moe( "Moe" );
    moe.display();

    cout << "\nTest 4: Adding a section to a lab worker.
\n";
    moe.addSection( secA2 );
    moe.display();

    cout << "\nTest 5: Adding a second section and lab 
worker.\n";
    LabWorker jane( "Jane" );
    Section secB3( "B3", "Thursday", 11 );
    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");
    jane.addSection( secB3 );
    jane.display();

    cout << "\nTest 6: Adding some grades for week one\n";
    moe.addGrade("John", 17, 1);
    moe.addGrade("Paul", 19, 1);
    moe.addGrade("George", 16, 1);
    moe.addGrade("Ringo", 7, 1);
    moe.display();

    cout << "\nTest 7: Adding some grades for week three 
(skipping week 2)\n";
    moe.addGrade("John", 15, 3);
    moe.addGrade("Paul", 20, 3);
    moe.addGrade("Ringo", 0, 3);
    moe.addGrade("George", 16, 3);
    moe.display();

    cout << "\nTest 8: We're done (almost)! \nWhat should 
happen to all those students (or rather their records?)\n";

    cout << "\nTest 9: Oh, IF you have covered copy 
constructors in lecture, then make sure the following call 
works\n";
    doNothing(secA2);
    cout << "Back from doNothing\n\n";

} // main
```

Output

When you have completely finished all your classes The output for the above test program should look like the following:

```
Test 1: Defining a section
Section: A2, Time slot: [Day: Tuesday, Start time: 4pm], Students: 
None

Test 2: Adding students to a section
Section: A2, Time slot: [Day: Tuesday, Start time: 4pm], Students:
Name: John, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: George, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: Paul, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: Ringo, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 

Test 3: Defining a lab worker.
Moe does not have a section

Test 4: Adding a section to a lab worker.
Moe has Section: A2, Time slot: [Day: Tuesday, Start time: 4pm], 
Students:
Name: John, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: George, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: Paul, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: Ringo, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 

Test 5: Adding a second section and lab worker.
Jane has Section: B3, Time slot: [Day: Thursday, Start time: 
11am], Students:
Name: Thorin, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
Name: Dwalin, Grades: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 
```

```
Name: Balin, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Kili, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Fili, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Dori, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Nori, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Ori, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Oin, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Gloin, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Bifur, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Bofur, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Bombur, Grades: −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1

Test 6: Adding some grades for week one
Moe has Section: A2, Time slot: [Day: Tuesday, Start time: 4pm],
Students:
Name: John, Grades: 17 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: George, Grades: 16 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Paul, Grades: 19 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Ringo, Grades: 7 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1

Test 7: Adding some grades for week three (skipping week 2)
Moe has Section: A2, Time slot: [Day: Tuesday, Start time: 4pm],
Students:
Name: John, Grades: 17 −1 15 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: George, Grades: 16 −1 16 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Paul, Grades: 19 −1 20 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Ringo, Grades: 7 −1 0 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1

Test 8: We're done (almost)!
What should happen to all those students (or rather their
records?)

Test 9: Oh, IF you have covered copy constructors in lecture, then
make sure the following call works
Section: A2, Time slot: [Day: Tuesday, Start time: 4pm], Students:
Name: John, Grades: 17 −1 15 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: George, Grades: 16 −1 16 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Paul, Grades: 19 −1 20 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Name: Ringo, Grades: 7 −1 0 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Section A2 is being deleted
Deleting John
Deleting George
Deleting Paul
Deleting Ringo
Back from doNothing
```

```
Section B3 is being deleted
Deleting Thorin
Deleting Dwalin
Deleting Balin
Deleting Kili
Deleting Fili
Deleting Dori
Deleting Nori
Deleting Ori
Deleting Oin
Deleting Gloin
Deleting Bifur
Deleting Bofur
Deleting Bombur
Section A2 is being deleted
Deleting John
Deleting George
Deleting Paul
Deleting Ringo
```