Part C - Encapsulation

Constructors

Workshop 4

In this workshop, you are to code a class that represents an enrollment and processes transactions on that enrollment.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- code a class for simple objects
- define a constructor that sets an object to a safe empty state
- overload a constructor to receive information from a client

SUBMISSION POLICY

The “in-lab” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (a 20% late deduction will be assessed). The “at-home” portion of the lab is **due the day before your next scheduled workshop**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly backup your work.

ENROLLMENT CLASS

Download or clone workshop 4 from https://github.com/Seneca-244200/OOP-Workshop4
Open Workshop4/in_lab directory and complete the code in `Enrollment.h` and `Enrollment.cpp.`

Design and code a class named `Enrollment` that holds information about a student’s enrollment in course sections at a college. Place your class definition in a header file named `Enrollment.h` and your function definitions in an implementation file named

`Enrollment.cpp`. Include in your coding all of the statements necessary for your code to compile under a standard C++ compiler, and within the `ict` namespace.

The class `Enrollment` has the following six data members:

- name_: A C-style string holding the name of the course in which the student is enrolled. You may assume that the string is no longer than 30 characters, excluding the null terminator.
- code_: A C-style string holding the alphanumeric course code for the course in which the student is enrolled. You may assume that the string is no longer than 10 characters, excluding the null terminator.
- year_: An integer indicating the year in which the course is being offered **(>= 2015)**.
- semester_: An integer indicating the semester in which the course is being offered **(> 0 and < 4)**.
- slot_: An integer indicating the time-slot this course occupies **(> 0 and < 6)**.
- enrolled_: A Boolean value to indicate whether the student is currently enrolled (a value of `false` indicates the student has been withdrawn from the course).

Enrollment has the following member functions (methods) that are already implemented in `Enrollment.cpp.`

- `bool isEnrolled() const` – a query that returns the value of `enrolled`.
- `bool isValid()const` - returns `true` when the object is NOT in a safe empty state.
- `void setEmpty()`- sets the object to a safe empty state.
- `void display(bool nameOnly=false) const` - a query that displays the course name and code as well as the year, term and time-slot when nameOnly is false. If nameOnly is true, it will display the name only. This function displays `Invalid enrollment!` if the object is in a safe empty state (when it is invalid);
  Enrollment is in a safe empty state when the char strings are empty and other values are set to zero or false.

IN-LAB SECTION (80%)

Implement the following member functions: (make sure you reuse your code and DO NOT re-implement the logics alrady implemented)

- `void set(const char* name, const char* code, int year, int semester, int time, bool enrolled = false) -` a setter function that stores valid data in an `Enrollment` object. The first parameter receives the address of a C-style string containing the course name, the second parameter receives the address of a C-style string containing the course code, followed by 3 integers receiving year, term, time-slot and enrolled respectively. If the last argument is not

provided, _enrolled is set to false.
If any of the data is invalid, this function will set the object to a safe empty state.

- `bool hasConflict(const Enrollment &other) const` – a query that accepts an unmodifiable reference to the other `Enrollment` object and returns whether two course sections have a conflict.
  Two sections have a conflict if they have the same year, semester, and time-slot.
  If any of the objects are invalid this function returns false.

- Include two constructors in your class: a no-argument constructor that sets the object to a safe empty state and a five-argument constructor that receives the address of a C-style string containing the course name, a second address of a C-style string containing the course code, 3 integers indicating year, semester, and time-slot respectively. The five-argument constructor should set the value indicating that the student is enrolled to `false`.

Note that you must check the validity of users' input. If null pointer arguments are provided, or validation fails, the object is set to a ***safe empty state***.

CLIENT MODULE

Here is a sample of implementation file for the `w4_in_lab.cpp` main module that you should use to test your implementation:

```
// OOP244 Workshop 4: Constructors
// File: w4_in_lab.cpp
// Version: 1.0
// Date: 2/8/2016
// Author: Fardad Soleimanloo
// Description:
// This file tests in-lab section of your workshop
/////////////////////////////////////////////


#include <iostream>
#include "Enrollment.h"
#include "Enrollment.h"  // this is on purpose
using namespace std;
using namespace ict;
void displayEnrollments(const Enrollment& e1, const Enrollment& e2);
int main(){
  // constructors
  Enrollment e1("College English", "EAC150", 2015, 1, 5);
  Enrollment e2, e3;
  Enrollment badData[] = {
    Enrollment("Intro to Programming in C", "IPC144", 2015, 1, 5),
    Enrollment((char*)0, "IPC144", 2015, 1, 5),
    Enrollment("Intro to Programming in C", (char*)0, 2015, 1, 5),
    Enrollment("", "IPC144", 2015, 1, 5),
    Enrollment("Intro to Programming in C", "", 2015, 1, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2014, 1, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2015, 0, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2015, 4, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2015, 1, 0)
  };

  cout << "Testing Enrollment objects" << endl << endl;

  // testing two invalid Enrollments
  displayEnrollments(e3, e2);

  // testing isValid and inisValid Enrollments
  displayEnrollments(e1, e2);

  // setting the second one to OOP244
  e2.set("Object Oriented Programming", "OOP244", 2015, 1, 5);

  displayEnrollments(e1, e2);

  // setting the enrollment to true;
  e2.set("Object Oriented Programming", "OOP244", 2015, 1, 5, true);

  displayEnrollments(e1, e2);

  // removing conflict;
  e2.set("Object Oriented Programming", "OOP244", 2015, 1, 6);

  displayEnrollments(e1, e2);

  // making sure all the conditions are met for an inisValid Enrollment.
  cout << endl << "Only index 0 should be isValid:" << endl;
  for (int i = 0; i < 9; i++){
    cout <<"index: " << i << " - "<< (badData[i].isValid() ? "V" : "Not v") <<
"alid." << endl;
  }
  return 0;
}
void displayEnrollments(const Enrollment& e1, const Enrollment& e2){
  cout << "---------------------------------------------" << endl;
  e1.display();
  cout << (e1.isEnrolled() ? "E" : "Not e") << "nrolled" << endl;
  e2.display();
  cout << endl<<  "There is " << (e1.hasConflict(e2)? "" : "not ") << "a conflict!"
<< endl;
}
```

Output Example:
(Your output should exactly match the following)

```
Testing Enrollment objects
```

```
----------------------------------------------
Invalid enrollment!
Not enrolled
Invalid enrollment!

There is not a conflict!
----------------------------------------------
College English
EAC150, Year: 2015 semester: 1 Slot: 5
Status: Not enrolled.
Not enrolled
Invalid enrollment!

There is not a conflict!
----------------------------------------------
College English
EAC150, Year: 2015 semester: 1 Slot: 5
Status: Not enrolled.
Not enrolled
Object Oriented Programming
OOP244, Year: 2015 semester: 1 Slot: 5
Status: Not enrolled.

There is a conflict!
----------------------------------------------
College English
EAC150, Year: 2015 semester: 1 Slot: 5
Status: Not enrolled.
Not enrolled
Object Oriented Programming
OOP244, Year: 2015 semester: 1 Slot: 5
Status: Enrolled.

There is a conflict!
----------------------------------------------
College English
EAC150, Year: 2015 semester: 1 Slot: 5
Status: Not enrolled.
Not enrolled
Invalid enrollment!

There is not a conflict!

Only index 0 should be valid:
index: 0 - Valid.
```

```
index: 1 - Not valid.
index: 2 - Not valid.
index: 3 - Not valid.
index: 4 - Not valid.
index: 5 - Not valid.
index: 6 - Not valid.
index: 7 - Not valid.
index: 8 - Not valid.
```

IN-LAB SUBMISSION

To test and demonstrate the execution of your program, use **`w4_in_lab.cpp`** file. This file should produce the same outputs as shown above.

If not on matrix already, upload your **`Enrollment.h`** and **`Enrollment.cpp`** and **`w4_in_lab.cpp`** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit w3_in_lab <ENTER>
```

and follow the instructions.

AT-HOME SECTION: TO ENROL

Add the following member functions to the Enrollment class:

- **`void withdraw() -`** a modifier that withdraws the student from the course, and displays an appropriate message.
- **`int enrol(const Enrollment* enrollments, int size);`** – a function that sets the **`enrolled`** flag to **`true`** and returns **`0`**, unless there is a conflict between the **`Enrollment`** object (this object) and one of the elements in the **`enrollments`** array. If a conflict is found, the function returns the sequence number of the element in conflict (i.e. returns index+1) and the **`enrolled`** flag remains **`false`**. The parameter **`size`** represents the size of the **`enrollments`** array.

Test your program with **`w4_at_home.cpp`** to make sure the **`enrol`** and **`withdraw`** functions work correctly.

CLIENT MODULE

```
// OOP244 Workshop 4: Constructors
// File: w4_in_lab.cpp
// Version: 1.0
// Date: 2/8/2016
// Author: Fardad Soleimanloo
// Description:
// This file tests in-lab section of your workshop
/////////////////////////////////////////////


#include <iostream>
#include "Enrollment.h"
#include "Enrollment.h"  // this is on purpose
using namespace std;
using namespace ict;
void displayEnrollments(const Enrollment& e1, const Enrollment& e2);
int main(){
  // constructors
  Enrollment e1("College English", "EAC150", 2015, 1, 5);
  Enrollment e2, e3;
  Enrollment badData[] = {
    Enrollment("Intro to Programming in C", "IPC144", 2015, 1, 5),
    Enrollment((char*)0, "IPC144", 2015, 1, 5),
    Enrollment("Intro to Programming in C", (char*)0, 2015, 1, 5),
    Enrollment("", "IPC144", 2015, 1, 5),
    Enrollment("Intro to Programming in C", "", 2015, 1, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2014, 1, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2015, 0, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2015, 4, 5),
    Enrollment("Intro to Programming in C", "IPC144", 2015, 1, 0)
  };

  cout << "Testing Enrollment objects" << endl << endl;

  // testing two invalid Enrollments
  displayEnrollments(e3, e2);

  // testing isValid and inisValid Enrollments
  displayEnrollments(e1, e2);
  // setting the second one to OOP244
  e2.set("Object Oriented Programming", "OOP244", 2015, 1, 5);

  displayEnrollments(e1, e2);

  // setting the enrollment to true;
  e2.set("Object Oriented Programming", "OOP244", 2015, 1, 5, true);

  displayEnrollments(e1, e2);

  // removing conflict;
  e2.set("Object Oriented Programming", "OOP244", 2015, 1, 6);

  displayEnrollments(e1, e2);

  // making sure all the conditions are met for an inisValid
Enrollment.
  cout << endl << "Only index 0 should be isValid:" << endl;
  for (int i = 0; i < 9; i++){
    cout <<"index: " << i << " - "<< (badData[i].isValid() ? "V" :
"Not v") << "alid." << endl;
  }
  return 0;
}
void displayEnrollments(const Enrollment& e1, const Enrollment& e2){
  cout << "----------------------------------------------" << endl;
  e1.display();
  cout << (e1.isEnrolled() ? "E" : "Not e") << "nrolled" << endl;
  e2.display();
  cout << endl<<  "There is " << (e1.hasConflict(e2)? "" : "not ") <<
"a conflict!" << endl;
}
```

Sample output:

```
Please enter the number of course sections: 5
Please enter the following 5 course sections for enrollment:
Course section 1:
Subject Name: aaa aaa
Subject Code: aa
Year: 2015
Semester: 3
Slot: 3
```

```
Enrolled!
Continue? (y/n)y
Course section 2:
Subject Name: bbb bbb
Subject Code: bb
Year: 2015
Semester: 2
Slot: 2
Enrolled!
Continue? (y/n)y
Course section 3:
Subject Name: ccc ccc
Subject Code: cc
Year: 2015
Semester: 2
Slot: 2
There is a conflict with the following, please change enrollment time:
bbb bbb
bb, Year: 2015 semester: 2 Slot: 2
Status: Enrolled.
Course section 3:
Subject Name: ccc ccc
Subject Code: cc
Year: 2015
Semester: 1
Slot: 1
Enrolled!
Continue? (y/n)n
------------------------------
aaa aaa
aa, Year: 2015 semester: 3 Slot: 3
Status: Enrolled.
------------------------------
bbb bbb
bb, Year: 2015 semester: 2 Slot: 2
Status: Enrolled.
------------------------------
ccc ccc
cc, Year: 2015 semester: 1 Slot: 1
Status: Enrolled.
------------------------------
Invalid enrollment!
------------------------------
Invalid enrollment!
aaa aaa
aa, Year: 2015 semester: 3 Slot: 3
```

```
Status: Not enrolled.
```

AT-HOME SUBMISSION: REFLECTION (20%)

Please provide brief answers to the following questions in a text file named `reflect.txt.`

1) What is meant by a **safe empty state**?
    a. What is the safe empty state in your program?
    b. Why did you select this state?
2) Describe a case where having multiple `Enrollment` constructors would simplify xthe code of clients that use it.

SUBMISSION

To test and demonstrate execution of your program use the same data as the sample output above.

If not on matrix already, upload your `Enrollment.h`, `Enrollment.cpp,` `reflect.txt` and `w4_at_home.cpp` to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit w3_at_home<ENTER>
```

Following the instructions and follow the instructions.