COMP 2404 -- Assignment #4

Due: Tuesday, April 9, 2019 at 12:00 pm (noon)

Goal

You will modify your student auditing program from a previous assignment or from the base code to interact with a Façade class that acts as an interface to a simulated cloud-based storage service. You will also implement a class template and overloaded operators.

Learning Outcomes

With this assignment, you will:

- practice the integration of your code with an existing class
- work with the Façade design pattern
- implement a class template and overloaded operators in C++

Instructions

1. Prepare to integrate the new Façade class

We are going to change the program so that it can synchronize with a simulated cloud-based storage service. Our Façade class will be the `StuServer` class, which is provided for you. This class will simulate the retrieval of a master list of existing student information from cloud storage at the beginning of the program, and your program will add these students to the local storage. Your program will then run as usual, with the user adding new students. Of course, the `StuServer` class that you are given will only ***simulate*** this behaviour, so there is *no actual cloud service*. However, you will be modifying your program to integrate with the student server, as if it was working for real.

You will need to download from *cuLearn* the `a4Posted.tar` file, which contains the `StuServer` class. It also contains an example program (`testUtil.cc`) for parsing strings using the `stringstream` library class. You will need to parse a different format of string in step #2 of this assignment, and you can use the given program as an example.

2. Integrate the new Façade class

You will make the following changes to the `Control` class:

- Add a `StuServer` object as a data member of the `Control` class; you must declare this data member **after** the `Storage` object is declared in your `Control` class, otherwise your students will be destroyed before the `StuServer` is finished printing them at the end of the program.
- Modify the `Control` class constructor so that it retrieves all the students from the student server (and the simulated cloud storage) at the beginning of the program. The data for each student is formatted as a string, so your program will have to parse each string to retrieve the data and create the corresponding `Student` object. The constructor will:
  - declare a STL vector of strings to hold the students that are retrieved from cloud storage
  - call the student server's `retrieve` function with that vector passed by reference; this function will populate the vector with the data from the cloud storage

- loop over the retrieved vector of strings, where each string corresponds to the student and course data for one student; parse each string to retrieve the student and course information, and add each student to storage
  - each student string is formatted as follows:

    `id coursecode1 term1 grade1 instructor1 coursecode2 term2 grade2 instructor2 ... 0`

    where `id` is the student id, and what follows is a list of any number of courses, represented as course code, term taken, grade obtained, and instructor for that course, with the list of courses terminated with a course code of zero
  - the `testUtil` program provided gives an example of parsing a simple string

For your program to work with the new `StuServer` class, you will have to modify your Makefile to add the `StuServer` object file to the linking command.

3. Modify the List class as a class template

You will modify the `List` class to make it a class template. Note that the data will no longer be assumed to be a pointer! You should be able to use any data type in the list. If you did not implement the `List` class in a previous assignment, you can adapt the `Array` class from the tutorials.

4. Create the new CourseList class

You will create a new `CourseList` class that derives from the templated `List` class of `Course` pointers. The `CourseList` class will **not** be a class template. If you did not implement the `List` class in a previous assignment, you can adapt the `Array` class from the tutorials.

The `CourseList` class will contain two member functions:

- a `computeGPA()` function that returns the average of all course grades (between 0 and 12) for the student, excluding the withdrawals; you will use the function prototype: `float computeGPA();`
- a `computeNumFW()` function that returns the number of courses that the student has failed or from which the student has withdrawn; you will use the function prototype: `int computeNumFW();`

You will modify the `Student` class to store the student's courses in a `CourseList` object, instead of a `List` object.

You will also remove these functions from the `List` class, as they will called on the `CourseList` object instead.

5. Add overloaded operators

You will modify the `Storage`, `Student`, and `List` classes to replace all the add functions with the overloaded `+=` operator. Change all the classes that use these functions so that they now use the new operators. Make sure that you enable cascading.

6. Test the program

- You will provide the `in.txt` file with sufficient datafill for a minimum of 15 different students, each with at least 5 different courses. The nature of the data and its ordering in the file must **thoroughly** test your program. If parts of your program cannot be adequately tested because of inadequate datafill, you will lose 100% of the marks for some marking components.
- Check that the student and course information is correct when the storage is printed out.
- Make sure that all dynamically allocated memory is explicitly deallocated when it is no longer used. Use `valgrind` to check for memory leaks.

Constraints

- your program must follow correct encapsulation principles, including the separation of control, UI, and entity object functionality
- do not use any classes, containers, or algorithms from the C++ standard template library (STL), except where explicitly permitted
- do not use any global variables or any global functions other than `main()`
- do not use structs; use classes instead
- objects must always be passed by reference, not by value
- your classes must be thoroughly documented in every class definition
- all basic error checking must be performed
- existing functions must be reused everywhere possible

Submission

You will submit in *cuLearn*, before the due date and time, the following:

- one `tar` or `zip` file that includes:
  - all source, header, and data files, including the code provided
  - a Makefile
  - a readme file that includes:
    - a preamble (program and revision authors, purpose, list of source/header/data files)
    - compilation. launching, and operating instructions

NOTE: Do **not** include object files, executables, swap files, or duplicate files in your submission.

Grading (out of 100)

Marking components:

- 50 marks: correct integration with the `StuServer` class
- 12 marks: correct modifications to the `List` class
- 18 marks: correct implementation of the `CourseList` class
- 20 marks: correct implementation of overloaded operators

Execution requirements:

- all marking components must be called, and they must execute successfully to receive marks
- all data handled must be printed to the screen for marking components to receive marks

Deductions:

- Packaging errors:
  - 10 marks for missing Makefile
  - 5 marks for a missing readme
  - 10 marks for consistent failure to correctly separate code into source and header files
  - 10 marks for bad style or missing documentation

- Major programming and design errors:
  - 50% of a marking component that uses global variables, global functions, or structs
  - 50% of a marking component that consistently fails to use correct design principles
  - 50% of a marking component that uses prohibited library classes or functions
  - 100% of a marking component that is *replaced* by prohibited library classes or functions
  - 50% of a marking component where unauthorized changes have been made to the base code
  - up to 10 marks for memory leaks, depending on severity
- Execution errors:
  - 100% of a marking component that cannot be tested because it doesn't compile or execute in VM
  - 100% of a marking component that cannot be tested because the feature is not used in the code
  - 100% of a marking component that cannot be tested because data cannot be printed to the screen
  - 100% of a marking component that cannot be tested because insufficient datafill is provided