COMP 2404 -- Tutorial #5

Object Design Categories

Learning Outcomes

After this tutorial, you will be able to:

- design a program using control, view, and entity objects
- create a UML class diagram to document your design

Instructions

1. You will begin with the code you saved from Tutorial #4.

2. Using paper and pencil, draw a UML class diagram of the existing code, as we covered in section 3.4 of the course material. Remember, functions are not the same as classes, so `main()` is **not a class**! Now add to your diagram a `Control` class that will be responsible for all control flow, specifically the work that's done in `main()`. Now add a `View` class to deal with user I/O. Think about what functions will be required in each class. You will keep adding to your UML diagram as you modify the code in each of the steps below.

3. Create a new `View` class that is responsible for interacting with the user. The `View` class will contain:
   - a member function for displaying the main menu and reading the user's selection
   - a member function for reading all information from the user about one book
   - a member function for printing out the library; this function will take the library as a parameter, and it will use *delegation*, as seen in Tutorial #3, to ask the `Library` class to print to the screen

   Except for printing the library at the end of the program, only the `View` class will interact with the user. You must change the program so that user I/O goes through this class.

   After the `Control` and `View` classes are correctly implemented, your code should have no global functions other than `main()`.

4. Change the `main()` function so that its only responsibility is to declare a `Control` object and call its `launch()` function.

5. Create a new `Control` class in the program to implement the control flow from the `main()` function. The `Control` class will contain:
   - a data member for the `Library` object that used to be declared in `main()`
   - a data member for a new `View` object that will be responsible for user I/O
   - a `launch()` member function that implements the program control flow and does the following:
     - use the `View` object to display the main menu and read the user's selection, until the user chooses to exit
     - if required by the user, create a new `Book` object and add it to the library using existing functions
     - use the `View` object to print the content of the library to the screen at the end of the program

   The `Control` class will perform all user I/O using the `View` class. It will not interact with the user directly.

NOTE: Steps #6 and #7 are for bonus marks only. They are not part of the regular tutorial.

6. **[For bonus marks only]** Now we have a small problem. On one hand, we are required to handle all user I/O in the `View` class. On the other hand, correct encapsulation dictates that the `View` object should not know anything about *how* the books are stored in the library. It's the job of the library's `Array` object to know how to traverse the book collection, **not** the job of the `View` object. So how can we get the `View` object to print out the books if it can't traverse the book array? We also can't ask a collection class like `Array` to print to the screen, as this violates encapsulation rules as well. To solve this problem, we introduce some new formatting functions.

   In our solution, the `Control` object will tell the entity and collection objects to format the data they contain into one big string, and the `Control` object will pass that string to the `View` object to print to the screen. The formatting functions will be similar in nature to the `toString()` functions in Java.

   You will change the code as follows:
   - rename the `print()` functions in the `Book`, `Library` , and `Array` classes to the following:
     ```
     void format(string& outStr)
     ```
   - re-write the `format()` functions so that they format the object data in the same way that it was printed to the screen, but instead of outputting, they place the formatted data into the `outStr` output parameter
     - you can use the `stringstream` class to help with the correct formatting (more details below)
     - the `format()` functions will not print anything to the screen
     - the `outStr` parameter must be declared in the calling function
   - change the code in the `Control` object to format the library data before asking the `View` to print it, using a new member function in the `View` class that simply prints a given string to the screen
   - the library's `format()` function will call the array object's `format()` function
   - the array object's `format()` function will loop through the book array, call the `format()` function on each book, and append each book's formatted string to the formatted string for the entire library

7. **[For bonus marks only]** To use the `stringstream` class, you must include the `<sstream>` header file. In each formatting function, you will declare a `stringstream` object on which you can use the stream insertion operator (`<<`), which is the same operator that we've been using with `cout`. As a result, we can place formatted data into the `stringstream` object in the same way that we've been printing to standard output. At the end of the function, we can convert our `stringstream` object to a regular string using its `str()` member function. For example, the function to format `Book` data could look as follows:
   ```
   stringstream ss;
   ss << setw(3)     << id
      <<"  Title: "  << setw(40) << title
      <<", Author: " << setw(20) << author
     <<", Year: "    << year;
   outStr = ss.str();
   ```

8. Build and run the program. Check that the books are ordered correctly when the library is printed out at the end of the program.

9. Make sure that all dynamically allocated memory is explicitly deallocated when it is no longer used. Use `valgrind` to check for memory leaks.

10. Package together the tutorial code into a tar file, and upload it into cuLearn. Save your work to a permanent location, like a memory stick or your Z-drive.