COMP 2404 -- Tutorial #9

Façade Design Pattern

Learning Outcomes

After this tutorial, you will be able to:

- practice the integration of your code with an existing class
- work with the Façade design pattern

Instructions

1. You will begin with the code you saved from previous tutorials, or with the base code. If you're starting from the base code, you must first implement the `Book` inheritance hierarchy from Tutorial #7, and the `Book` hierarchy's polymorphic `lessThan()` functions from Tutorial #8.

2. 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 `BookServer` class, which is provided for you. This class will simulate the retrieval of a master list of the user's books from cloud storage at the beginning of the program. The program will then run as usual, with the user adding new books. Then at the end of the program, the book server will simulate the sending of all of the user's books (both the original ones and the new ones) back to the cloud storage, where they can be retrieved the next time the program runs. Of course, the `BookServer` 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 book server, as if it was working for real. You will need to download from cuLearn the `t09Posted.tar` file, which contains the `BookServer` class, as well as the `Array` class that it requires.

   Download and un-tar the `t09Posted.tar` file. This tar file contains the header and object files for the `BookServer` and `Array` classes. Modify your Makefile so that the linking step that creates the executable links in the `BookServer.o` and `Array.o` files. That way, when you change your code to use these classes, you will be ready to link them into your executable.

   In order to work correctly, the `BookServer` and `Array` classes expect the following:

   - your data classes must be named in accordance with previous tutorial instructions (`Book`, `FictionBook`, `NonFictionBook`)
   - your `Book` class must provide a `print()` function, in accordance with previous tutorials
   - both `FictionBook` and `NonFictionBook` constructors must have the following prototypes:

     ```
     FictionBook(int id, string callNumber, string title, string author, int year)
     NonFictionBook((int id, string callNumber, string title, string author, int year)
     ```

3. **If you are starting from a previous tutorial:** Most of the changes for integrating with the Façade class will occur in the `Control` class. However, you will need to implement some utility functions in other classes first. Add a `copy` member function to the `List` class. This function will have the prototype: `void copy(Array&)`. It will traverse the linked list and add every `Book` pointer that is currently in the list to the parameter array. Note that this will be a shallow copy. We are not creating new books! We are simply copying all the pointers from one container (the list) into another (the array).

4. **If you are starting from a previous tutorial:** Add a `copyBooks` member function to the `Library` class. The function will have the prototype: `void copyBooks(Array&)`. It will call its book list's `copy` function, which we implemented in the previous step.

5. **If you are starting from a previous tutorial:** Now we can make the changes to the `Control` class:

   - add a `BookServer` object as a data member of the `Control` class; you must declare this data member **after** the `Library` objects are declared in your `Control` class, otherwise your books will be destroyed before the `BookServer` is finished printing them
   - add a default constructor to the `Control` class that retrieves all of the user's books from the book server (and the simulated cloud storage) at the beginning of the program. The constructor will:
     - declare two local `Array` objects: one to hold the fiction books and one to hold the non-fiction books that are currently in cloud storage
     - call the book server's `retrieve` function with those two `Array` objects; this function will populate the arrays with the data from the cloud storage
     - loop over each array separately, and add each book currently in the array to the correct library (for example, the books in the fiction books array should be added to the lounge library, and the books in the non-fiction books array should be added to the SCS library)
   - add a destructor to the `Control` class that sends all the books from the lounge and SCS libraries to the cloud storage using the book server at the end of the program. The destructor will:
     - declare two local `Array` objects: one to hold the fiction books and one to hold the non-fiction books that will be sent to the cloud storage
     - use the `copyBooks` function of each library to copy the library's books to the corresponding `Array` object (for example, books from the lounge library will be copied to the fiction books array)
     - call the book server's `update` function with the two `Array` objects; this function will simulate the upload of the data to the cloud storage

   The book server prints out everything from its master list at the end of the program. Check that all the books are printed out in the correct order.

6. **If you are starting from the base code:**
   - declare a `BookServer` object as a local variable in your `main()` function
   - write two global functions that are called from `main()`, one that retrieves books from the book server at the beginning of the program, and one that updates the book server at the end of the program
   - in the global function called at the beginning of your program:
     - declare two local `Array` objects: one to hold the fiction books and one to hold the non-fiction books that are currently in cloud storage
     - call the book server's `retrieve()` function with those two `Array` objects; this function will populate the arrays with the data from the cloud storage
     - loop over each `Array` object separately, and add each book currently in the `Array` object to the correct primitive book array declared in the `main()` function (for example, the books in the fiction book `Array` object should be added to the primitive array of fiction books, and the books in the non-fiction `Array` object should be added to the primitive array of non-fiction books)
     - think carefully about what data from `main()` needs to be passed as parameters to this function; do **not** use global variables

- in the global function called at the end of your program:
  - declare two local `Array` objects: one to hold the fiction books and one to hold the non-fiction books that will be sent to the cloud storage
  - copy every book from the `main()` function's primitive array of fiction books into one of the `Array` objects, and do the same for the primitive array of non-fiction books into the other `Array` object
  - call the book server's `update()` function with the two `Array` objects; this function will simulate the upload of the data to the cloud storage

  The book server prints out everything from its master list at the end of the program. Check that all the books are printed out in the correct order.

7. Build and run the program. Check that each book is in the correct library, and that the books are ordered correctly, both in the forward direction and the backward direction, when both libraries are printed out at the end of the program. Fiction books should be ordered by author, and non-fiction books by call number. Check that the book server's list of books contains all the books.

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

9. 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.