COMP 2404 -- Tutorial #3

Simple Collection Class

Learning Outcomes

After this tutorial, you will be able to:

- create a simple collection class with data and behaviour
- use *delegation* to implement functionality in the appropriate objects

Instructions

1. Collection classes are a very important tool for encapsulating behaviour in an OO program. The goal of these classes is to hide from the rest of the program how a particular collection (array, linked list, stack, etc.) is implemented. In this tutorial, we will create a new collection class called `Array`. We will move the responsibility for managing the elements of the book array away from the `Library` class and into the new collection class. You will begin with the code you saved from Tutorial #2.

2. Create a new `Array` class. You will need both a header file and a source file for this class. You can refer to the course material (section 2.1) and the coding examples done in class to see what belongs in the header file and what belongs in the source file.

   The class will contain two data members:
   - a data member called `elements`, which is an array of `Book` objects
   - a data member called `size`, which is the current number of books in the array

   Since the book array is moving from the `Library` class, you will also move the constant array size definition (`MAX_ARR_SIZE`) into the correct location.

3. Write the following functions for the `Array` class:
   - a constructor that initializes the data member(s) that require initialization
   - an `add(Book&)` function that adds the given book parameter to the back of the book array
   - a `print()` function that prints out all the books in the array to the screen

   Note: These functions will be identical to the ones you wrote for the `Library` class in Tutorial #2.

4. Modify the `Library` class so that it does not manipulate the book array elements directly:
   - remove the two data members that keep track of the books, and replace them with an `Array` object
   - look at the constructor and decide if the code is still needed; if not, it should be removed

5. Look at the `Library` class's `addBook` function and see what it does. What would be the best way to update this function? One option would be to have the `main()` function retrieve the `Array` object and call its `add` function directly. However, that would be poor design, since we would be shifting the responsibility for accessing the book elements even farther away from where the data is stored. It would also mean that any changes to the `Array` class would have an impact on a larger portion of the program, which would also be a poor design. Instead, we will make no changes to `main()`, and we will use a common OO programming technique called *delegation* in the `Library` class.

Modify the `Library`'s `addBook` function to call the `add` function on its `Array` object. This delegates the responsibility for adding a book to the array to the class that knows the most about storing books, which is the `Array` class.

6. Modify the `Library`'s `print` function so that it delegates printing responsibility to the `Array` object's `print` function. If your code prints out a heading (for example, "LIBRARY:") when the books are printed to the screen, make sure that this heading is not printed by the `Array` class. Printing a heading would be the job of the `Library` class. The `Array` class is only responsible for printing out the elements that it contains.

7. Update the Makefile so that the new `Array` class gets compiled and linked into the executable, as we saw in the course material in section 1.1.

8. Build and run the program. Check that the book information is correct when the library is printed out at the end of the program.

9. Package together the tutorial code into a tar file. Start up a browser in the VM, log into cuLearn, and go to the tutorial page. Select the tutorial submission link, and upload your new tar file.

10. Save your work to a permanent location, like a memory stick or your Z-drive.