ACP exercise: `BookSeller`, OOP with C++ classes

CompSys 202 / MechEng 270

Introduction

We are going to use the C++ features we have been learning so far, particularly instance/static variables, instance/static methods, getters/setters, constructors/destructors, equality, and following good coding practices.

What will our program do?

We want to model a book shop that manages books and purchases:

- Maintain the book seller's cash balance.
- When the book seller purchases a new book:
  - The cash balance is reduced (by the cost price).
  - The book is added to the collection of books on offer.
  - Give each book a unique ID.
- When the book seller sells a book:
  - The book is sold by scanning the book's unique ID.
  - The cash balance is increased (by the sale price).
  - The book is marked as sold in the collections of books on offer.
- Given a unique ID, grab the book that corresponds to that ID (if any).
- Given a keyword in a book's title, search and grab a book that matches the keyword (if any).
- Determine the ***total*** number of **sold and unsold** books (include duplicate books).
- Determine the ***total*** number of **unsold** books (include duplicate books).
- Given a book, sell all copies of that book (equality based on title, author, publication year).

Example usage:

```
#include "BookSeller.h"
#include "Book.h"

#include <iostream>
using namespace std;

int main() {
  BookSeller shop("Books 'r' Us", 100.0);
  cout << "Starting balance: $" << shop.getCashBalance() << endl;

  Book *hp1 = new Book("Harry Potter Philosopher Stone", "J. K. Rowling", 10.50, 29.95, 1997); // 0
  Book *hp2a = new Book("Harry Potter Chamber Secrets", "J. K. Rowling", 11.50, 31.95, 1998);  // 1
  Book *hp2b = new Book("Harry Potter Chamber Secrets", "J. K. Rowling", 9.50, 31.95, 1998);   // 2
  Book *phil = new Book("Philosophy 101", "Paul Kleinman", 8.75, 15.99, 2013);                 // 3

  shop.purchaseStock(hp2b);
  cout << "Balance after purchase: $" << shop.getCashBalance() << endl;

  cout << "Total books: " << shop.totalNumberOfBooks() << endl;
  Book *result = shop.retrieveBookFromID(2);
  if (result != 0)
    cout << "ID search found: " << result->getTitle() << endl;

  bool success = shop.scanAndSellBook(2);

  result = shop.searchKeyword("Secrets");
  if (result != 0) {
    cout << "Keyword search found: " << result->getTitle() << endl;
    int num = shop.sellAllCopiesOfBook(result);
    cout << "Sold " << num << " copies" << endl;
  }
```

Useful links

- `http://www.cplusplus.com/reference/vector/vector/`
- `http://www.cplusplus.com/reference/string/string/`

What to do

1. Create a **BookSeller** class, separating definition/declaration into separate files.
   (a) Declare and define constructor **BookSeller(const std::string &shopName, double cashBalance);**
   (b) Declare and define **double getCashBalance() const;**
   (c) Create an instance of **BookSeller**, and print the starting cash balance as in the example above.
2. Create a **Book** class, separating definition/declaration into separate files.
   (a) Declare and define constructor, using title, author, costPrice, salePrice, year.
   (b) Assign a unique ID to the book when it is constructed, using a static variable.
3. Create a few **Book** instances, as in the example above.
   (a) Add a **getID()** function for the Book class, making it a **const** function.
   (b) Test that each book has a unique ID returned.
4. Add a **void purchaseStock(Book *book)** function in **BookSeller**.
   (a) Decrement the cash balance (will need to add **getCostPrice()** for **Book**).
   (b) Save the book inside a vector. *Hints:*
      i. Inside **BookSeller.h**: **#include <vector>** and declare **std::vector<Book*> collection**
      ii. Inside **BookSeller.cpp**: use **push_back()**
5. Implement a function **int totalNumberOfBooks() const;** inside **BookSeller**
   (a) Either return the vector's **size()**, or implement a **static int numberBooksEverMade();** inside **Book**.
6. Implement a function **Book* retrieveBookFromID(int id) const;** inside **BookSeller**.
   (a) Loop through each book in the collection. If a book's ID matches the id we want, return the book.
   (b) If a book with that id wasn't found, return null pointer (zero).
   (c) Test it by retrieving existing/missing books in the collection:
      i. Print the title of the result to check (add **string getTitle() const;** inside **Book**)
      ii. Safe-guard this printing with an if-statement, just in case the returned pointer was null.
7. Implement a function **bool scanAndSellBook(int id);** inside **BookSeller**.
   (a) Reuse the **retrieveBookFromID(int id)** function to get the Book.
   (b) If no book is returned, then return **false** from **scanAndSellBook()**.
   (c) Add a **sold** status to **Book**, with its respective setter and getter functions.
      i. You will also need to initialise this variable in the **Book**'s constructor.
   (d) For the retrieved book, check it isn't already sold.
      i. If it is already sold, return **false** from **scanAndSellBook()**.
      ii. Otherwise, update its status to sold, update cashBalance, and return **true** from **scanAndSellBook()**.
8. Implement a function **int totalNumberOfUnsoldBooks() const;** inside **BookSeller**.
   (a) Loop through all books and count how many books aren't sold yet. Return that count.
9. Implement a function **Book* searchKeyword(const std::string &keyword) const;** inside **BookSeller**.
   (a) Loop through the collection. If you **find()** keyword in the title, return the book. Otherwise return null.
   (b) Don't worry about making it case-insensitive.
10. Implement a function **int sellAllCopiesOfBook(Book *book);** inside **BookSeller**.
    (a) Define equality for Book, using **operator==(const Book &other)** comparing title, author, year.
    (b) Inside **sellAllCopiesOfBook**, define a count for the number of copies (initially zero).
    (c) For each "equal book" in the collection, attempt to sell it using the existing **scanAndSellBook()** function. Check if it was successful (return value of **scanAndSellBook()**), and if so, increment the count.
    (d) Finally, return the copy count.
11. What about the destructors? We will make the design decision that if a BookSeller closes down, then the Books no longer exist.