CSCI 145 Week 7
Lab 5

Goals for this Lab

1. Write a class and use objects.
2. Understand difference between `public` and `private` methods.
3. Understand difference between `static` and non-`static` methods.

Task Description

Your task is to write a Java class named `BookCollection` (in a file `BookCollection.java`) which defines a data type for a collection of books. Your `BookCollection` class must use the resources provided by the `Book` class. Your `BookCollection` class must provide the resources needed by the provided class `BookStuff`.

The file lab5.zip contains the following files, discussed in more detail below:

- `Lab5.pdf` – this description
- `Book.java` and `BookStuff.java` – these files provide the `Book` class that your `BookCollection` class will need to use and the class `BookStuff` that uses your `BookCollection` class. Do not modify either of these files.
- `BookCollection.java` – the beginning of a `BookCollection` class.
- `List1.txt`, `List2.txt`, `Change1.txt`, and `Change2.txt` – these four files are text files that are used by the `BookStuff` class to test the `BookCollection` class.

Book.java

This class defines a book type that contains the data fields that are used to describe a book and the operations a book can perform.

Note: Do not modify this file.

BookStuff.java

This class file contains the main method. It is a client of `BookCollection` and `Book` classes. This class will not compile with the provided `BookCollection` class. It expects that the methods described below will be available.

`BookStuff` reads in the details of several books from two data files whose names are given on the command line. The data files `List1.txt` and `List2.txt` are provided for this purpose. This class will also manipulate the book collections in accord with instructions given in change files. The data files `Change1.txt` and `Change2.txt`, which have been provided, describe the changes.

All the I/O (Input/Output) and command line arguments are handled by `BookStuff`. For each data file, `BookStuff` adds the books described in the file to a `BookCollection` object. `BookStuff` performs the operations on the books that are described in the corresponding change file. Operations include: changing the price of books, adding stock for books, and selling books. `BookStuff` displays the contents of each collection.

After creating and manipulating two separate `BookCollection`s, `BookStuff` merges the two

collections and displays all three collections.

Note: All of this capability is in the provided `BookStuff` class. It has been tested and found to work correctly. Your task is to develop `BookCollection` to provide the resources needed by `BookStuff`. You may modify BookStuff to help you with creating your `BookCollection` class. However, your final version must work with the version I provided.

BookCollection.java

You need to develop and complete this class for the lab exercise. It will need to use the `Book` class. Your class should not perform any I/O, except for debugging purposes, which you should remove or comment out before submitting the lab.

This class must provide the book collection methods needed by `BookStuff`. You could deduce those needs from studying the file `BookStuff.java`, but to save you the time and trouble, here is a list of the required fields and methods in `BookCollection.java`.

Constants	A pre-determined `public static final` constant `LIMIT` which is the maximum size of a collection when it is created. This constant is present in the provided file.
Data Fields	All data fields must be `private`, and cannot be directly accessed/visible from outside of the class. The following fields are required: 1. An array of `Book` type that is used to hold the collection of Books. 2. The actual size of the collection, initially set to zero. It should not at any time exceed the preset limit constant.
Exceptions	The provided file defines three exceptions: 1. `BookNotFound` – this exception is thrown when a book needed by a method cannot be found in the collection. 2. `DuplicateBook` – this exception is thrown when an attempt is made to add a duplicate of a book to a collection. 3. `CollectionFull` – this exception is thrown when an attempt is made to add a book to a full collection. The sample `changePrice` method in `BookCollection.java` shows how to throw one of these exceptions. The description of methods, below, states when these exceptions are to be used.
Constructor **BookCollection(int size)**	Create an empty book collection of the given size. The size should not exceed the preset maximum size given by `LIMIT`.
Method **void changePrice(String isbn, double price)**	Search the collection for a book with the given ISBN and, if found, change its price, to the given price. If the book is not found, throw the exception `BookNotFound`.

Method **void changeStock(String isbn, int quantity)**	Search the collection for a book with the given ISBN and, if found, add quantity to its stock. The parameter quantity can be positive (books added to stock) or negative (books sold). If the book is not found, throw the exception `BookNotFound`. If adding quantity to the stock would result in negative stock, the Book object will throw an `InsufficientStock` exception. You do not have to handle this exception, it will be handled by `BookStuff`.
Method **int getSize()**	Return the actual number of books in the collection. (Do not return the size the collection was created with.)
Method **double getStockValue()**	Returns the total dollar value of the books in the collection. This value is computed by adding the values (see `getStockValue` method of `Book`) of all the books in the collection.
Method **void addBook (Book book)**	Adds the given book to the collection, provided there is room in the collection and the book is not already there. If the collection has already reached its limit, throw the `CollectionFull` exception. If the book is already in the collection as determined by ISBN, throw the `DuplicateBook` exception.
Method **Book objectAt(int i)**	Return the book at the given index in the collection, provided the index falls into the legitimate range of the book collection. If the index is not in range, throw an `IndexOutOfBoundsException`. (This exception is defined in `java.lang` and can be used without an `import` statement.)
Method **static BookCollection merge(BookCollection collection1, BookCollection collection2)**	Merge the given book collections, returning a new collection which contains all the books found in either or both collections. The size of the new collection must be sufficient to hold all the books in the merged collection but can be larger. If any book is found in both of the collections, only one entry is added to the new collection. The stock for that book is the sum of the stock in the two merged collections and the price is the minimum price for that book in the two collections. The books in the new collection must new `Book` objects. It is not permitted that books be shared with the other collections. (This can be accomplished using the `Book(Book)` constructor of the `Book` class.
Suggested method: **private Book findBook(String isbn)**	More than one method involves searching for a book (using the ISBN) in a collection. It will make your job easier if you write a findBook method as a helper function. This method should be `private`, since it will be used only by the methods within the `BookCollection` class. This method should return `null` if the book cannot be found in the collection. This will allow `findBook` to be used by `addBook` and `merge`, where not finding the book is expected (`addBook`) or OK (`merge`).

Suggested Steps

1. Get your program to compile. You can do this by adding stubs for all the needed methods to `BookCollection.java`. Here are a couple of sample stubs to get you started:

```
public void addBook (Book book) {
}

public double getStockValue() {
   return 0.0;
}
```

2. Get the constructor, `addBook`, `findBook`, `getSize`, and `objectAt` methods working. (`findBook` is needed by `addBook`.) You will need to add the data fields in order to implement these methods. These methods give you enough to construct the collections from the list files and display the collections. If you comment out the calls to `changeCollection` in `BookStuff.java` you can run the program and display the collections.
3. Implement `changePrice` and `changeStock`. This will allow `changeCollection` in `BookStuff` to run.
4. Implement `getStockValue`. This will allow the total value of the collections to be displayed correctly. (Note: You can switch steps 3 and 4 if you want.)
5. Implement `merge`. This is the most complicated to the methods. Once this is done and everything is working correctly, the assignment is complete.

Notes

1. If may seem like there's a lot to do. But, except for `merge`, all the methods can be done in less than a dozen lines of code. So, just do them one at a time. Steps 2 and 5, above, will be the most time consuming.
2. You can run the program as follows:

```
java -cp . BookStuff List1.txt Change1.txt List2.txt Change2.txt
```

3. There are two sizes associated with a collection. The underlying size is specified when the collection is constructed. This is the same as the length of the array of `Book`s data field. The actual size is the number of valid (non-`null`) values in the array. Calls to `addBook` increase the actual size. The methods `findBook`, `getStockValue`, `objectAt`, and `getSize` all use the actual size.
4. Make sure to create a new `Book` object when adding a book to the merged collection in `merge`. This is needed to avoid problems when books are present in both collections and price and/or stock can be changed. This is why all three collections are displayed at the end – to ensure that the merge operation did not change the original two collections.

Turn in your program on Canvas

Turn in your `BookCollection.java` file. Look for the Assignment link on the Canvas page for Lab 5 and click that to turn in the file you just created.

Grading

Your program is due by 11:59pm, Sunday, February 21st. The grading will be:

- 30% – load and display a collection (Steps 2 and 4)
- 20% – modify a collection (Step 3)
- 30% – merge collections (Step 5)
- 20% – overall program organization and presentation