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. (Note: All four files have Linux style line endings. Remember: if you open any of these using NotePad on Windows, the file will appear as one long line.)
- `sample_output.txt` – this file is a sample of the output of running `BookStuff.java` with a working `BookCollection` class.

Read *Suggested Steps* and *Notes*, both below, before starting on the lab.

Source Files

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. You will need to review this file to see the operations available on `Book`s.

Note: Do not modify this file.

BookStuff.java

This class 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 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 `Book` and `BookStuff` to help you with creating your `BookCollection` class. However, **your final version must work with the provided versions.**

BookCollection.java

The provided class is a skeleton for what you need to develop. Stubs have been provided for all the methods described below. In addition, a complete version of the `changePrice` method has been provided as an example, including how to throw an exception.

Completing BookCollection.java

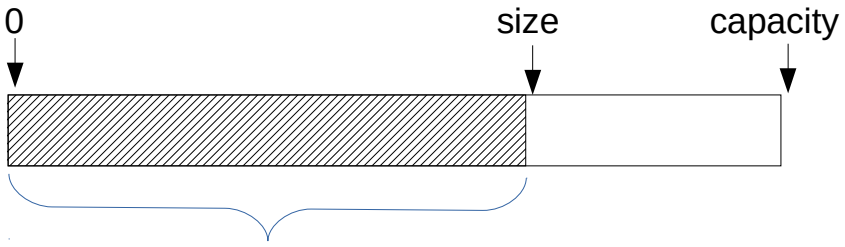
Description

You need to complete this class for the lab exercise. This class uses 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.

The idea behind this class is to maintain a collection of books as an array. The array will have a fixed size which is referred to as the capacity of the collection. The capacity is the same as the length of the underlying array.

In addition, the collection has a size which is the number of `Book` objects actually in the array. This should be the same as the number of valid (non-`null`) values in the array. Calls to the `addBook` method increase the actual size. The `findBook`, `getStockValue`, `objectAt`, and `getSize` methods all use the size.

You should note that there are no methods that remove books from the collection. Thus, the easiest way to maintain the collection is as follows:



Remember that arrays are indexed from zero. Thus, zero is the "first" location in the array, size is the index of the next empty location in the array and capacity is the "location after the last location" in the array. The capacity and the length of the array are the same.

Since `Book`s will never be removed, you don't have to worry about messy issues like removing an entry from the middle of the array.

Methods and Fields

If you want to, you can skip this section until you start implementing the class. At that point, you should review the details for each method before implementing it. The comments in `BookCollection.java` duplicate the information found in the table below. However, there may be additional information in the table that is not found in those comments.

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 capacity of a collection when it is created. This constant is present in the provided file.
Data Fields	All data fields must be `private`, so that they are not directly accessible/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 `Book`s. 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` – an exception thrown when a needed book cannot be found in a collection. 2. `DuplicateBook` – an exception thrown when an attempt is made to add a duplicate of a book to a collection. 3. `CollectionFull` – an exception thrown when an attempt is made to add a book to a full collection. The sample `changePrice` method in the supplied `BookCollection.java` shows how to throw one of these exceptions. The description of methods, states when these exceptions are to be thrown.

Constructor **`public BookCollection (int capacity)`**	Create an empty book collection of the given capacity. The capacity should not exceed the preset maximum capacity given by `LIMIT`.
Method **`public 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 **`public 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 **`public int getSize()`**	Return the actual number of books in the collection. (Do not return the capacity the collection was created with.)
Method **`public 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 **public void addBook (Book book)**	Adds a new `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 capacity, 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 **`public 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 capacity 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 `Book` objects 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`). A stub for this method has been include in the `BookCollection` class provided with the assignment.

Suggested Steps

1. Compile and execute the provided code. You should be able to compile the code without error. Running the code should produce some uninteresting output and terminate with a `NullPointerException`.
2. Get `BookCollection` (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 two calls to `changeCollection` and the call to `BookCollection.merge` 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: Steps 3 and 4 can be done in either order.)
5. Implement `merge`. This is the most complicated to the methods. Once this is done and everything is working correctly, the assignment is complete. (See note 4 below for some hints on how to do the `merge` method.)

If you have time available at the end of the lab, you can start on Assignment 2.

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. (And, even `merge` can be done in about 20.) So, just do them one at a time. Steps 2 and 5, above, will be the most time consuming.
2. The provided `BookCollection.java` has stubs for the methods you need to write. The methods that return values return either zero or `null`. That is required to make the compiler happy. Your version of the methods should return the correct value, which could be zero or `null` in the right circumstances.

3. You can run the program from the command line as follows:

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

4. The easiest way to implement `merge` is as follows:

   a) First, create a new collection with capacity that is the sum of the size of the two individual collections. This guarantees that the new combined collection can hold all the `Book`s in the two individual collections.

   b) Copy the first collection into the new collection. Create a new `Book` object that is a copy of the original book and add that to the new collection. Use the `Book(Book other)` constructor in the `Book` class to do this. Creating a new book 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.

   c) Loop through the second collection. For each book in the second collection, check to see if it is already in the combined collection. If it is, then merge the data from the second book into the first. If it is not, then create a new `Book` object that is a copy of the original book and add that to the new collection.

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. You should not turn in any additional files. In particular, don't turn in `Book.java` or `BookStuff.java`.

Grading

Your program is due by 11:59pm, Wednesday, February 22. 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
- There will be a 10% deduction for <u>not</u> turning your program in correctly.