CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 5

Object Oriented Thinking

Due: Wednesday, November 2, 11:59 PM

Learning Goals:

- Implement classes that interact with each other.
- Write unit tests using objects and instance methods.

NOTE: This assignment must be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Get Started with Starter Code (0 points)

1. Make sure there is no problem with your Java software development environment. If there is any, review Assignment 1 or come to the office/lab hours before you start Assignment 5.
2. First, navigate to the `cse8b` folder that you have created in Assignment 1 and create a new folder titled `assignment5`.
3. Download the starter code. You can download the starter code from Piazza → Resources → Homework → `assignment5.zip`. The starter code should contain four files: `Assignment5.java`, `Book.java`, `LibraryReader.java`, and `Library.java`. Place the starter code within the `assignment5` folder that you have just created.
4. Compile and run the starter code, and you should expect the following output:

```
$ javac *.java
$ java Assignment5
FAILED: addBook() for input 1
Failed test.
```

Part 1: Implement Library Application (80 points)

Overview

In this assignment, you will implement a very minimalistic simulator version of a library management system. This will allow you easily keep track of the books available at the library as well as the list of books that are issued by different readers.

Note: you must NOT change any data field or method signature in the starter code. As such, please observe the starter code and read the instructions below to make sure you understand what each field means before you start to implement.

What's a library without books? (15 points)

In `Book.java`, you will be implementing **2 constructors, 4 instance getter methods, and 1 instance setter method**. This class is a POJO (Plain Old Java Object) class without any great functionality of its own but acts as a template to model a 'Book' in a library.

Each `Book` object, which is an instance of the `Book` class, contains the following fields (all are provided in the starter code; do not change any of these):

1. `private String name`: the name of the book
2. `private String authorName`: the name of the book's author
3. `private int isbn`: the ISBN number of the book
4. `private boolean isAvailable`: if the book is available in the library

Notice how each member field is declared `private`**.** This means that the member is only visible *within* the class, not from any other class. In other words, you will need to use accessors (i.e., getter methods) and mutators (i.e., setter methods) to access and modify, respectively, these `private` members. **You must also use the** `this` **keyword to access member variables hidden by local variables.**

The `Book` class contains the following member methods:

Note: (Please implement and complete the method body but do not change the method headers)

1. **`public Book()`**

This is the no-arg constructor of the `Book` class. The constructor needs to set the `name`, `authorName`, `isbn`, and `isAvailable` member variables as follows.
    - `name` must be set to `null`
    - `authorName` must be set to `null`
    - `isbn` must be set to `-1`
    - `isAvailable` must be set to `false`

2. **`public Book(String name, String authorName, int isbn)`**
   This is another constructor of the `Book` class. This constructor needs to set the `name`, `authorName`, and the `isbn` member variables using the constructor parameters. Finally, the constructor must set the `isAvailable` member variable of the newly created book to `true`. Remember, you must use the `this` keyword to access member variables hidden by local variables.

3. Four getters/accessors
    - **`public String getName()`**
    - **`public String getAuthorName()`**
    - **`public int getISBN()`**
    - **`public boolean getIsAvailable()`**

   Each getter method must simply return the corresponding `private` field of this `Book` object.

4. One setters/mutator for `isAvailable`
    - **`public void setIsAvailable(boolean isAvailable)`**

   This method is used to set the availability of the book (within a library), which is discussed further below. Again, remember you must use the `this` keyword to access member variables hidden by local variables.

Let's build a library! (25 points)

Here, we implement methods inside `Library.java` to add and remove books from the library, clear the list of books in the library, get details of a book in the library, and change details of a book in the library.

Each `Library` object, which is an instance of the `Library` class, contains the following fields (all are provided in the starter code; do not change any of these):

1. `private String name`: the name of the library
2. `private Book[] bookList`: an array of `Book` objects to hold books in the library
3. `private int bookCount`: the number of books in the library

Notice how each member field is declared `private`. This means that the member is only visible *within* the class, not from any other class. In other words, you will need to use accessors

(i.e., getter methods) and mutators (i.e., setter methods) to access and modify, respectively, these `private` members. **You must also use the `this` keyword to access member variables hidden by local variables.**

Note: There are two constants `MAX_BOOK_LIMIT` and `BOOKLIST_FULL` defined in the beginning of the class. We will describe below how to use them.

The `Library` class contains the following member methods:

Note: (Please implement and complete the method body but do not change the method headers)

1. **`public Library()`**
   This is the no-arg constructor of the `Library` class. The constructor needs to set the `name`, `bookList` , and `bookCount` member variables as follows.
   - `name` must be set to `DEFAULT_LIBRARY_NAME`
   - `bookList` must be an array of `Book` type and length `MAX_BOOK_LIMIT`
   - `bookCount` must be set to `0`, indicating there are no books in the library.

2. **`public Library(String name)`**
   This is another constructor of the `Library` class. First, the constructor needs to set the `name` member variable using the constructor parameter. Next, you need to initialize the `bookList` array and set the size of the array using the `MAX_BOOK_LIMIT` constant provided at the beginning of the class. Last, set the `bookCount` member variable to `0`, indicating that there are no books in the library.

3. Three getters/accessors
   - **`public String getName()`**
   - **`public Book[] getBookList()`**
   - **`public int getBookCount()`**

   Each getter method must simply return the corresponding `private` field of the `Library` object.

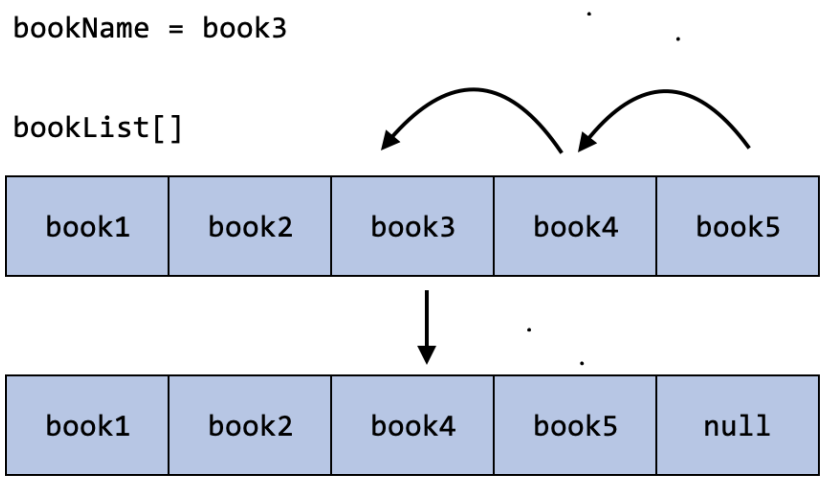
4. One setter/mutator
   - **`public void setName(String name)`**

   The setter method must set the `name` member variable to the one provided in the method argument. Again, remember you must use the `this` keyword to access member variables hidden by local variables.

5. **`public void addBook(Book newBook)`**
   This method implements the functionality to add the book `newBook` to the library. Use the `bookCount` member variable to track the position in the `bookList` array where `newBook`

will be added. Before adding `newBook`, check if the book list is full by comparing `bookCount` to the `MAX_BOOK_LIMIT` constant. If so, then print to the console the `bookList` is full using `System.out.println()` and the `BOOKLIST_FULL` constant. Make sure to update `bookCount` after adding `newBook` to the array**.** However, if a book with the same **name** as `newBook` is already present in `bookList`, then you do not need to do anything further so just return from the method.

6. **`public void removeBook(String bookName)`**
   This method implements the functionality to remove the book named `bookName` from the book list. To accomplish this, first search the array referred to by `bookList` in order to identify the index where the book represented by the `bookName` is present. If no such book is in the book list, then you do not need to do anything further so just return from the method. However, if the index containing `bookName` is found, then overwrite the book with the next book in the `bookList` (if any) and continue shifting all the books that are present in the subsequent positions by one less index. Once the `book` objects are shifted, replace the element in the array referred to by `bookList` with `null` (so that there are no two objects of the same value). The following figure illustrates removing a book from the array referred to by `bookList`.



   Remember to update the `bookCount` member variable once the remove operation is done.

7. **`public Book getBookInformation(String bookName)`**
   This method implements the functionality to retrieve a book from the library with the name `bookName`. Given the name of the book, search the array referred to by `booklist` to find the corresponding `Book` object. Return the `Book` object contained in `bookList` corresponding to `bookName`. Return `null` if a book named `bookName` is not in `bookList`.

8. **`public void clear()`**
   This method implements the functionality to clear the entire book list (i.e., it removes all the books from the library). When this method is invoked, the entire book list represented by `bookList` is cleared. The clear operation is performed by reinitializing the `bookList` array to a new array whose length is equal to `MAX_BOOK_LIMIT`. Remember to reinitialize `bookCount` to `0` since the book list is now cleared.

Finally, we'll implement a reader who has access to our library! (40 points)

Next, we will implement methods inside `LibraryReader.java` using the two classes we just finished implementing. For simplicity, we will assume that we only have one reader at a time who will interact with the library.

A `LibraryReader` object contains the following fields (all are provided in the starter code; do not change any of these):

1. `private Library library`: a library object
2. `private Book[] readerBookList`: an array of `Book`s to hold books in the library that the reader has borrowed
3. `private int readerBorrowCount`: the number of books the reader has borrowed. Any reader is only allowed to borrow a maximum of `3` books (defined in variable `MAX_BOOK_BORROW_LIMIT`)

In the starter code, we have provided two constructors: a no-arg constructor and a constructor that takes in a `Library` object as a reference. We give instructions on how to implement these constructors below. The autograder will use these constructors to test your implementations.

The `LibraryReader` class must contain the following member methods:

Note: (Please implement and complete the method body but do not change the method headers)

1. **`public LibraryReader()`**
   This is the no-arg constructor of the `LibraryReader` class. The constructor needs to set the `library`, `readerBookList` , and `readerBorrowCount` member variables as follows.
   - `library` must be set to `null`
   - `readerBookList` must be an array of `Book` type and length `MAX_BOOK_BORROW_LIMIT`
   - `readerBorrowCount` must be set to `0`, indicating there are no books that that reader has borrowed.

2. **`public LibraryReader(Library library)`**
   This is another constructor of the `LibraryReader` class. First, the constructor needs to set the `library` member variable using the constructor parameter. Next, you need to initialize the `readerBookList` array and set the size of the array using the `MAX_BOOK_BORROW_LIMIT` constant provided at the beginning of the class. Last, set the `readerBorrowCount` member variable to `0`, indicating that there are no books the reader has borrowed..

3. Three getters/accessors
   - **`public Library getLibrary()`**
   - **`public Book[] getReaderBookList()`**
   - **`Public int getReaderBorrowCount()`**

   These getter methods must simply return the corresponding `private` data field of the `LibraryReader` object.

4. **`public int borrowBookFromLibrary(String bookName)`**
   This method lets the user borrow a book from the library. There are multiple things you need to keep in mind when you implement this method.
   - Ensure the `readerBorrowCount` is less than `MAX_BOOK_BORROW_LIMIT`. If not, then the method returns `-1`.
   - Ensure the book the reader is asking for is available (*hint:* look at the `Book` object to find which variable is used to represent availability).
   - If you find the book with the same name as `bookName` and it is available, let the reader borrow the book by adding the book to the `readerBookList`. How will the `readerBookCount` change when you do this?
   - Once the book is borrowed, it is no longer available in the library.
   - Return `1` after successfully borrowing the book from the library. If the book is not available in the library, then return `0`.

5. **`public Book[] getAvailableBooks()`**
   When the reader wants to borrow a book, they might want to know what books are **available** in the library. This method returns a list of available books from the library. (*Hint:* Which variable would you use to check whether a book is available? What is the data type of each element in the library?)

6. **`public int returnBookToLibrary()`**
   Once the reader has finished reading, they might want to return a book to the library. For simplicity, you will remove the first element (0th index) of `readerBookList`. You will then shift the index of the rest of the books the reader has borrowed. Once the `book` objects are shifted, replace the element in the array referred to by `readerBookList` with `null`. There are additional things you need to keep in mind when you implement this method.
   - Ensure that the reader has books they have borrowed. If there are no books they have borrowed, the return `-1`.

- Once the reader returns a book, it is again available in the library. How will the `readerBookCount` change when you do this?
- Return `1` after successfully returning the book to the library.

Part 2: Compile, Run and UnitTest Your Code (10 points)

Just like in previous assignments, **in this part of the assignment, you need to implement your own test cases in the method called `unitTests` in the `Assignment5.java` class. We will be writing unit tests to test only `Library.java`.**

In the starter code, a test case is already implemented for you. You can regard it as an example to implement other cases. Recall, the general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output. Additionally, the starter code also provides detailed comments on how the test cases were formed. *Read them carefully.* It will be helpful in writing the rest of the test cases.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method must return `true` only when all the test cases are passed. Otherwise, it must return `false`. **To get full credit for this section, you must create at least four test cases that cover different situations.** In other words, you will need to create at least **three** more tests that test `addBook`, `removeBook`, `getBookInformation`, and `clear`.

To compare arrays by the equality of contents, you must use `Arrays.deepEquals()`. See the given unit tests for examples.

If a test is not passing, try temporarily printing the result of your method(s) and comparing them to the expected output.

You can compile all the files present in the starter code and run your unit tests from `main()` using the following commands: (Make sure you are in the correct directory, else navigate to the starter code using cd)

```
> javac *.java
> java Assignment5
```

The first command `javac *.java` compiles all the files in the folder with a `.java` extension, which is what is required. After the `.java` files compile and `.class` files are generated, run the `main()` in `Assignment5` using `java Assignment5`. The below screenshot shows the same:

```
$ ls
Assignment5.java        Book.java               Library.java            LibraryReader.java
$
$ javac *.java
$ ls
Assignment5.class       Book.class              Library.class           LibraryReader.class
Assignment5.java        Book.java               Library.java            LibraryReader.java
$
$ java Assignment5
All unit tests passed.
```

Even though you're not required to write test cases for `LibraryReader.java`, it is advised that you do to test the correctness of your implementation. The autograder will check these methods for correctness *thoroughly* (including various edge cases).

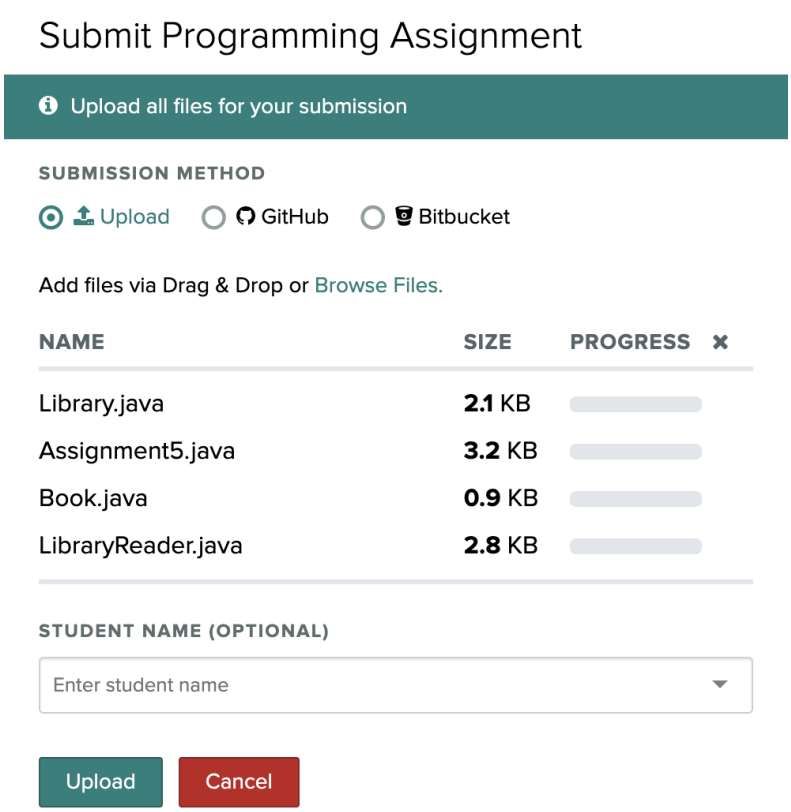
Submission

You're almost there! Please follow the instructions below carefully and use the **exact submission format**. Because we will use scripts to grade, **you may receive a zero** if you do not follow the same submission format.

1. Go to Gradescope via Canvas and click on Assignment 5.
2. Click the DRAG & DROP section and directly select the required files (`Assignment5.java`, `Book.java`, `LibraryReader.java`, and `Library.java`). Drag & drop is fine. Please make sure you do not submit a zip, just the four files in one Gradescope submission. Make sure the names of the files are correct.
3. You can resubmit an unlimited number of times before the due date. Your score will depend on your final (most recent) submission, even if your former submissions have higher scores.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of basic tests. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

5. Your submission should look like the screenshot below. **If you have any questions, then feel free to post on Piazza!**


Submit Programming Assignment
Upload all files for your submission
SUBMISSION METHOD
Upload
GitHub
Bitbucket
Add files via Drag & Drop or Browse Files.
NAME
SIZE
PROGRESS
Library.java
2.1 KB
Assignment5.java
3.2 KB
Book.java
0.9 KB
LibraryReader.java
2.8 KB
STUDENT NAME (OPTIONAL)
Enter student name
Upload
Cancel