CSCI 1933 Lab 5

Arrays, Sorting, and Some File I/O

Lab Rules

You may work individually or with *one* partner in lab. We suggest that you have a TA evaluate your progress on each milestone before you move onto the next. The labs are designed to be completable by the end of lab. If you are unable to complete all of the milestones by the end of lab, you have **until the last office hours on Monday** to get those checked off by **any** of the course TAs. We suggest you get your milestones checked off as soon as you complete them since Monday office hours tend to become crowded. If you worked with a partner, both of you must be present when getting checked off to receive credit. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Moodle for this lab.

Attendance

Per the syllabus, students with more than 3 unexcused lab absences over the course of the semester will automatically fail the course. The TAs will take attendance during lab; it is expected that students will be actively working on the lab material. *Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance.*

Introduction

As a college student, you probably have a lot of books. For this lab, we've decided we'd like to use our computers to represent and search through our numerous books. To facilitate this, you have been provided with a `Book` class. It contains 3 private variables: a title, an author, and a rating. It also contains getters, setters, a `toString` method, and a `compareTo` method (we'll get into that later).

1 Creating a Bookshelf Class

We will store our books in a `Bookshelf` class. Your `Bookshelf` will consist of an array of `Book` objects. It should also contain two constructors – one should take in no arguments, and should initialize the `Book` array with a default size of 20. The other should take in a single `int` argument which pertains to the size of the array. Optionally, you can also add a constructor which takes in a `Book` array to use as the member `Book` array. Your class should also contain the following methods:

- `public boolean add(Book newBook)` – Attempt to add a `Book` to the first empty slot in the `Bookshelf`. If it is successful, it should return true, and false if not. Do not allow books to be added if the `Bookshelf` is full.

  Hint: How can you maintain a constant time complexity? What member variable is required?

- `public Bookshelf getBooksByAuthor(String author)` – return a new `Bookshelf` object containing only the books which were written by a given author. If no books were written by the given author, the returned `Bookshelf` should be empty.
- `public String toString()` – Build a string of all of the `Book` objects in the array. Separate the each `Book` with a single newline character.

Milestone 1:
Write up a few JUnit tests for your methods and show them to your TA.

2 Sorting Our Bookshelf Class

Our bookshelf might be functional, but it may as well not even be called a shelf. There's no order to it! This could be a pile of books and no one could tell the difference. Therefore, we need to sort it. Handily enough, our `Book` class comes with a special `compareTo` method. I'm sure you've probably used operators like $<$ or $>$ all the time. These operators, unfortunately, can't be used on objects. This is where methods such as `compareTo` come into play. `compareTo` is very similar to the `equals` method, except it returns an int based on how the two objects compare. If object a is less than object b, then `a.compareTo(b)` should return a negative number. If not, it should return a positive number, or zero.

The two `compareTo` methods already exist in the `Book` class. It is recommended you take a look at them to see how they are used. For this section of the lab, implement a `sort(char sortBy)` method in your `Bookshelf` class which reorders the objects in the `Bookshelf` so that they correspond to the ascending order of the `compareTo` method. Feel free to use any **reasonable** sorting algorithm you have learned to accomplish this task.

Caution: Remember to make sure you aren't trying to sort any `null` elements at the end of your array!

Milestone 2:
Write at least two tests for your `sort` method, and show them to your TA.

3 File I/O

Now we have a working `Bookshelf` class, and that's fantastic. However, hardcoding every book's title, author, and rating and then recompiling every time we want to make a change is incredibly time consuming. This is where file I/O comes in handy. Instead of hardcoding every book's information into our, we can simply create a method that reads in that information from a file. (This will also be used in the project, so pay attention).

Your task for this section is to write two methods in a `BookshelfReader` class. The class does not need a constructor, or any other methods:

- `public static Bookshelf readBooksFromFile(String fileName)`
- `public static void writeShelfToFile(Bookshelf b, String fileName)`

The first method will take in a file name (such as the provided `bookinput.txt`) and create a new `Bookshelf` object containing all the books in the file. To do so, you will need to import `File` and `Scanner`. An example setup for file input is listed below. Once you have a `Scanner` for the file, you can use it just as you would any other `Scanner`. The files will be setup just like your `toString` method, and you may assume both the amount of books in any file is never greater than 20 and commas are never used as a part of a title or author name.

The second method will take in a `Bookshelf` object and a file name (such as `output.txt`), and print out all of the books inside the `Bookshelf` to the file. You will need to import `PrintWriter` for this task. A `PrintWriter` object can be called with `print` or `println` methods, much like you would call them on `System.out`.

Example of using `PrintWriter` and `Scanner` on a `File`:

```java
// assume our filename is stored in the string fileName
Scanner s = null; // declare s outside try-catch block
try {
        s = new Scanner(new File(fileName));
} catch (Exception e) { // returns false if fails to find fileName
        return false;
}
// Now use s in the same way we used Scanners previously for user input
// To write to an arbitrary textfile, do the following:
// assume our filename is stored in the string fileName
PrintWriter p = null; // declare p outside try-catch block
try {
       p = new PrintWriter(new File(fileName));
} catch (Exception e) {
       return false;
}
```

Milestone 3:
Write a main method which reads in a `Bookshelf` from `bookinput.txt`, sorts the bookshelf by rating, then writes it to `output.txt`.

4 Honors

Note: This section and milestone are only required for students in the honors section. Students in other sections do not need to do this, but are still encouraged to work on it if interested and time permits.

Now that we have a way to add books to our shelves, we should provide a way to pull one off it. Implement the following method in `Bookshelf`:

```
public Book remove(String author, String title)
```

This function should search the array, looking for the book with a certain author and title. If the book isn't found, return `null`. If it is, remove it from the books array and return the `Book` object. Additionally, the `remove` method should shift all books to fill the gap left by the empty book. Do what is necessary so that add still runs in O(1) time. Be sure to rerun all of your previous unit tests to make sure you didn't break any functionality.

Milestone 4:
Write three units tests that check the following:

- Does the returned book match the one that was requested?
- After calling remove, does the book no longer exist in the array?
- After removing a few books, does my add method still work?