Lab04

Due tonight, Thursday, 9/15 at 11:59pm

All classes **must** be in package *lab04*. For any assignment or lab, unless explicitly mentioned otherwise, all classes are expected to be in a package of the exact same name, including lowercase and a possibly leading zero. Points will be deducted otherwise.

A word on exceptions

Recall that there are two sides to programming with exceptions: the client code that *throws*, or triggers, the exception, and the calling user code that receives that exception and optionally *catches* it. Beware the following common misuses of exceptions common to beginners.

1.

```
public double average(int[] array) {
    if (array == null) {
        throw new NullPointerException("null array");
    } else {
        // ...
    }
}
```

1.

```
public double average(int[] array) {
    try {
        if (array.length > 0) {
            // ...
        }
        return -1;
    } catch (NullPointerException e) {
        return -1;
    }
}
```

So what is wrong with these?

1. If something is null, Java will automatically generate a NullPointerException and throw it when you try to do something with nothing. Checking for null only to throw the exception Java would have thrown is redundant and thus unnecessary.
2. A **N**ull**P**ointer**E**xception *is an exceptional error, one that should happen rarely if ever. Seeing a NullPointerException is a sign of an error on the programmer's (your) part. If the code you wrote throws a NPE, change it to check for null and only proceed if it is *not* null.* If a method you're calling with null throws a NPE, you need to make sure before the call that the arguments you're providing are non-null.

For example, correct the above to

```
public double average(int[] array) {
    if (array != null && array.length > 0) {
        // ...
    } else {
        return -1;
    }
}
```

ArrayLists

ArrayLists have similarities and differences in comparison to arrays. They are both data structures (containers) that hold entities, each of the same arbitrary type. While arrays are fixed length and lie contiguously in memory, ArrayLists grow and shrink in size as necessary with no guarantee of where the entities are stored.

While arrays can hold both Objects and primitives like int and double, ArrayLists can only hold Objects. If want them to hold primitives, we need to use the primitive's equivalent class such as Integer, Double, Boolean, or Character.

Anytime you want to use ArrayLists, you need to import the class:

```
import java.util.ArrayList;
```

Let's compare the syntax and usage of arrays vs. ArrayLists.
Let `array` name a general array with *N* elements and `list` name a general ArrayList, both holding elements of type *T*

Concept	arrays	ArrayLists
Empty creation	`T[] array = new T[N]`	`ArrayList<T> list = new ArrayList<>()`

Values creation	`T[] array = { t0, t1, t2, ... }`	`ArrayList<T> list = new ArrayList<> (Arrays.asList(t0, t1, t2, ...))`
		
		`Or also`
		`ArrayList<T> list = (ArrayList<T>) Arrays.asList(t0, t1, t2, ...)`

get element i	`array[i]`	`list.get(i)`
set element i to v	`array[i] = v`	`list.set(i, v)`
get number of elements	`array.length`	`list.size()`
add element e	N/A	`list.add(e)`
print elements	`System.out.println(Arrays.toString(array))`	`System.out.println(list)`
index error	`ArrayIndexOutOfBoundsException`	`IndexOutOfBoundsException`

Read the table carefully to note all the differences and parallelisms. It can be tough to juggle the different syntaxes and easy to confuse which belongs to which.

We'll jump right into using ArrayLists to create a simple Library program.

Create a class `Book` with two private instance variables: a String title and int numPages. Create a constructor that accepts a String and int to initialize the two instance variables. If the integer passed in to the constructor is <= 0, throw a new IllegalArgumentException with an appropriate message.

Provide a getter for each instance variable: `getTitle` and `getNumPages`

Provide a method `public String toString()` that will return a formatted String holding the books information. This format is the title of the book, followed by a single space, followed by the number of pages within parentheses. For example, "Harry Potter (754)".

Create a class `Library`. Don't forget to import the ArrayList class. This class has one private instance variable, an ArrayList holding `Book`s. Provide a constructor with no arguments that initializes the ArrayList to a new empty ArrayList.

Provide a method `public void addBook(Book b)` that accepts a `Book` and adds it to the `Library's` ArrayList collection of `Book`s only if the given `Book` is not null.

Provide a method `public int numBooks()` that returns the number of books in the library.

Provide a method `public void printLibrary()` that prints out each book in the library. Refer to the above table on how to do this in one line.

Provide a method `public double averageLengthOfBooks()` that computes and returns the average number of pages over all the books in the library. The ArrayList cannot be null since our constructor initializes it nor can any books be null since `addBook` checks for that. However, there could be no books in the library, in which case you need to return -1.0.

Provide a method `public Book checkoutBook(int index)` that removes the book at the given index and returns that book. You may assume the index is within bounds. The ArrayList class offers a `remove(int index)` method, among many others. Read how it works here and try to apply it to this situation.

Provide a main method to test the code. Create a Library and at least three books to add to the Library. After adding the books, print out the contents of the Library with `printLibrary. Then` print the result of `averageLengthOfBooks` **as well as** the value you expect. Now check out one of the books and store the Book that was removed. Print out the updated Library and the removed Book to make sure the correct book was removed.