TCSS143
Fundamentals of Object-Oriented Programming-Theory and Application
Programming Assignment 6

DUE: Tuesday, November 4, 2014 by 11:59 p.m.

The purpose of this programming project is to demonstrate the functionality of ArrayList structure.

REQUIREMENTS

You will submit all source code files (**3 in all**) as a **single zipped file named "Programming6.zip**" through the Programming Assignment 6 Submission link on Canvas.

Not only will you be graded on program correctness (Program executes correctly, proper use of methods, classes, inheritance, etc.) but also, good programming documentation techniques including proper indentation, correct locations of braces, meaningful identifier names, general comments prior to methods, specific comments on complex code, javadoc, etc. If 2 or more incorrect documentation issues are spotted, 10 points will be deducted from the grade. Detailed grading of documentation will not take place.

DETAILS

You will create a simple simulated Library class that stores a collection (ArrayList) of books. The Library class also can perform various operations on the collection.

Each book is an instance of the Book class which you will also create. The Book class has a field for the title and an ArrayList field for the author(s) (because many books have multiple authors, an ArrayList to hold them seems a logical choice). The Book class also supplies various methods for operations on Books, described shortly.

1. Book.java

public class Book implements Comparable<Book> {

Fields: 2 private fields for the title of the book and the author's name(s). Because many books have multiple authors, you will need to implement an ArrayList of String to hold 1 or more author names. Once a book is published, the title and authors can't change. For this reason you should declare both fields to be final (NOTE: making these final has implications on where they are listed and how their names are documented).

Methods:

public Book(final String theTitle, final ArrayList<String> theAuthors)
Should simply throw a new IllegalArgumentException(), if either argument is invalid (null, empty string, etc.). If valid arguments, this constructor should set both fields. **Be sure to properly instantiate the ArrayList of author names field before you add anything to it! Do not just assign the parameter to the field or, do not clone it.**

public String getTitle()
Returns the book title

public ArrayList<String> getAuthors( )
Returns the ArrayList of authors.

public String toString( )
Returns a String representation of the title and each author for a single Book ONLY. This is because a Book object is just that, a single Book.

public int compareTo(final Book theOther)
Because Book Implements Comparable<Book>, a compareTo method must be written. Here, the titles will be compared, i.e. the implied parameter's title field with the "other" title field. However, in the event there a multiple titles that are actually different books (i.e. the books are not the same but, have the same title), you should then compare the author(s) names for order. In this case you need only compare the first author of each book (our test data will not have two equal titles with equal authors).

public boolean equals(final Object theOther)
Implemented as a correct equals method (see chapter 9). Once the instanceOf is correctly identified and the appropriate cast is applied, equals should compare both the title and the name ArrayList for equality and return the result (don't forget that some books have multiple authors).

2. Library.java

public class Library {

Fields: A single private ArrayList of Book field is all that is necessary. This will hold all the Book objects in the library.

Methods:
public Library(final ArrayList<Book> theOther)
Library should simply throw a NullPointerException(); if theOther is null. Otherwise, the Library's Book ArrayList should take on all the books in the parameter named other. **NOTE: You MUST create a new ArrayList of Book for the ArrayList of Book field in Library during this process.** Also, once an object of ArrayList<Book> is created you will add individual Book objects (from the parameter) to this list. An alternative method can simply pass "other" to the ArrayList constructor as was discussed in class.

public Library( )
Creates an empty ArrayList of books.

public boolean add(final Book theBook)
First checks for null or empty Strings and calls the appropriate exception. Otherwise it adds the Book argument to the end of the library ArrayList.

Notice it returns a boolean (whether or not the add operation succeeded). **Do Not** just return the constant "true." See the add method of ArrayList: http://docs.oracle.com/javase/6/docs/api/

public ArrayList<Book> findTitles(final String theTitle)
Generates an ArrayList of all books which have titles that match (exactly) with the passed argument and returns this list to the calling program. The String compareTo method is useful here.

public void sort( )
Sorts the library's book ArrayList in ascending order according to the title field (sort these titles just as they are, i.e. don't worry about articles such as The or A, etc.). As illustrated in the textbook, p. 666 (Don't let this number concern you :)), you will use the Collections sort.

public String toString( )
returns a properly formatted String representation of all the books in the library (Title followed by authors).

Neither of these classes should include any println or print statements. All output should be generated in the LibraryDriver.java class described on the next 2 pages.

Also, DO NOT use packages. Do not use detailed/complex try/catch blocks. Do not throw any other exceptions other than NullPointerException and IllegalArgumentException for parameters that are null and Lists that have no elements, respectively. These are topics not yet discussed in detail.

3. LibraryDriver.java

You will write a test driver program that will fully test all the methods of both classes using both valid and invalid data. The data (books and the authors) will be read from two different input files named "LibraryIn1.txt" and "LibraryIn2.txt." These input files are zipped together with these instructions and should be accessible in the same folder in which these instructions reside.

"LibraryIn1.txt" contains all the books and their authors that you will initially insert into your library. The format of this file is; every 2 lines represents 1 book: first line is the title, second line is a list of author(s) of the book.

The list of author(s) is a String which contains 1 or more author names. Author names are separated with an asterisk '*' which can be used to isolate a name so it can be inserted as an element of the ArrayList<String> of authors. The asterisk was deliberately chosen to force you to either research how to properly specify it as a delimiter in the useDelimiter method (which some of you may desire to use but, be forewarned, the asterisk requires special consideration under this usage) OR to simply iterate through the String while searching for the char '*' using the String methods: indexOf, charAt, substring, and length. The later is what I would prefer as it teaches you more in depth problem solving skills. This process should be done through a call to a static method included in the LibraryDriver class. This method should receive a String (representing the entire String of authors for a particular book) and return an ArrayList of String where each element contains a unique author name.

"LibraryIn2.txt" will be used to insert more books into the library after the initial list (read from the first file) has been inserted, printed out, sorted, and printer out again. You should close the first file before opening the second. Further processing (testing of your classes) will take place on the Library after this file is read, such as output of the new library of books, findtitle, sort, etc.

All output (test results) should be sent to an output file named "LibraryOut.txt" .

A skeleton version of the LibraryDriver.java file is described and seen below.

You also will need to write code to perform many of the other tests. This program simply loads the library with a small number of books, prints out the contents of the library, sorts the books in the library, and again, prints out the books, reads more books from a second file, tests the Book and Library methods, etc..

A complete version should be able to add single books to the library and also display a list of books/authors that have the same title by using the findTitles(String title) method of the Library class.

```
import java.util.*;
import java.io.*;
public class LibraryDriver{
  public static void main(String[] theArgs){
    Scanner inputFile = null ;
    PrintStream outputFile = null;
    try{
      inputFile = new Scanner(new File("LibraryIn1.txt"));
      outputFile = new PrintStream(new File("LibraryOut.txt"));
    } catch (Exception e){
        System.out.println("Difficulties opening the file!  " + e);
        System.exit(1);
    }
    ArrayList<String> authors = new ArrayList<String>();
    ArrayList<Book> books = new ArrayList<Book>();
    while (inputFile.hasNext()){
      // Read title
      // Read author(s)
      // Insert title & author(s)into a book
      // Add this book to the ArrayList<Book> of books
      }

// Instatiate a Library object filled with the books read thus far
// and write the contents of the library to the output file

// Sort the current contents of the library
// and write the contents of the sorted library to the output file

// Close the first input file and open the second input file.
// Read the titles and authors from the second input file,
// add them to the library, and write the contents of the
// library to the output file.
    inputFile.close();


    .  .  .  etc.

// Sort the library and write it to the output file

// The following tests the findTitles method:
// Write only the "Acer Dumpling" books to the output file

// Write only the "The Bluffs" books to the output file

// Close all open files and end main.
    inputFile.close();
    outputFile.close();

    }

// Header for method that separates author names and
// returns an ArrayList<String> containing the author names
    public static ArrayList<String> getAuthors(String s){

      // YOU FILL IN THE DETAILS HERE

    }
  }
```

For those who finish early and want something else to do, reformat your output to display the author names as firstname lastname; firstname lastname; etc.

Sample Ouput next page - - ->

Sample Output - - - > next page

```
PRINTS INITIAL BOOK LIST:
"The Hobbit," by Tolkien, J.R.
"Acer Dumpling," by Doofus, Robert
"A Christmas Carol," by Dickens, Charles
"Marley and Me," by Remember, SomeoneIdont
"The Bluff," by Dougan, Thomas; Peters, Paul; Elliot, Sam
"Acer Dumpling," by Sagan, Carl; Smith, Tom
"Building Java Programs," by Reges, Stuart; Stepp, Marty
"Java, How to Program," by Deitel, Paul; Deitel, Harvery


PRINTS SORTED BOOK LIST:
"A Christmas Carol," by Dickens, Charles
"Acer Dumpling," by Doofus, Robert
"Acer Dumpling," by Sagan, Carl; Smith, Tom
"Building Java Programs," by Reges, Stuart; Stepp, Marty
"Java, How to Program," by Deitel, Paul; Deitel, Harvery
"Marley and Me," by Remember, SomeoneIdont
"The Bluff," by Dougan, Thomas; Peters, Paul; Elliot, Sam
"The Hobbit," by Tolkien, J.R.


PRINTS WITH NEW BOOKS UNSORTED:
"A Christmas Carol," by Dickens, Charles
"Acer Dumpling," by Doofus, Robert
"Acer Dumpling," by Sagan, Carl; Smith, Tom
"Building Java Programs," by Reges, Stuart; Stepp, Marty
"Java, How to Program," by Deitel, Paul; Deitel, Harvery
"Marley and Me," by Remember, SomeoneIdont
"The Bluff," by Dougan, Thomas; Peters, Paul; Elliot, Sam
"The Hobbit," by Tolkien, J.R.
"Acer Dumpling," by Dude, Cool; Daddy, Big
"The Bluff," by Zwayer, Ted; Starr, Ringo; Stork, Tony


PRINTS ALL SORTED BOOK LIST:
"A Christmas Carol," by Dickens, Charles
"Acer Dumpling," by Doofus, Robert
"Acer Dumpling," by Dude, Cool; Daddy, Big
"Acer Dumpling," by Sagan, Carl; Smith, Tom
"Building Java Programs," by Reges, Stuart; Stepp, Marty
"Java, How to Program," by Deitel, Paul; Deitel, Harvery
"Marley and Me," by Remember, SomeoneIdont
"The Bluff," by Dougan, Thomas; Peters, Paul; Elliot, Sam
"The Bluff," by Zwayer, Ted; Starr, Ringo; Stork, Tony
"The Hobbit," by Tolkien, J.R.


PRINTS ALL ACER DUMPLINGS:
"Acer Dumpling," by Doofus, Robert
"Acer Dumpling," by Dude, Cool; Daddy, Big
"Acer Dumpling," by Sagan, Carl; Smith, Tom

PRINTS ALL THE BLUFFS:
"The Bluff," by Dougan, Thomas; Peters, Paul; Elliot, Sam
"The Bluff," by Zwayer, Ted; Starr, Ringo; Stork, Tony
```