Instructions for Lab Exercise 2
*Sorting and Searching Lists*
*Computer Science 2334*
*Due by: Friday, 10 February 2017, 4:00 pm CST*
***This lab is an individual exercise. Students must complete this assignment on their own.***

## *Learning Objectives:*

1. To understand the use of **ArrayList**s, how to create them, sort them, and search them.
2. To understand how to implement the **Comparable** interface and override the `compareTo()` method.
3. To learn how to use the `sort()` and `binarySearch()` methods of the **Collections** class.
4. To demonstrate this knowledge by completing a series of exercises.

The `sort()` and `binarySearch()` methods in the **Collections** class will be useful for completing the objectives in course projects. The **List** interface and other methods of the **Collections** class will be used extensively in later projects as well.

## *Instructions:*

This lab exercise *requires you to have a laptop with an Internet connection*. You will work to complete the exercises described in these instructions. Some exercises will require you to fill in the associated answer sheet with answers while others will require you to work on a Java project in Eclipse. You will submit a PDF copy of the completed answer sheet to a dropbox in D2L as well as files from the completed Java project to Mimir.

1. Download and save the answer sheet (`Lab2-CS2334-answerSheet.odt`).

2. Download the `Lab2-eclipse.zip` project archive from the class website and import it into Eclipse. This archive contains the **Book**, **BookList**, **BookFileReader**, and **Lab2Driver** classes. You will modify these files as a part of this lab exercise and submit the project archive when you are finished.

3. On the answer sheet, place a UML diagram for the four classes provided. (You may create this diagram using specialized UML software or general drawing tools, draw it by hand and scan it in, etc., so long as it is clearly legible on the submitted PDF answer sheet.)

4. Generics are a way to use general classes to deal with objects of specific types. In the **Book** class, the name of the genres of each book are stored in an **ArrayList**. **ArrayList**s are general in that they can hold objects of any type. However, we should specify which types of objects *this particular* **ArrayList** will hold. What is an appropriate type for an **ArrayList** of genre names? Replace the "raw type" **ArrayList**s in the **Book** class with generic **ArrayList**s by putting this type between less than and greater than symbols as if they were parentheses. That is, if you said that an appropriate type for a genre is **SnickerDoodle**, then you would replace each instance of `ArrayList` with `ArrayList<SnickerDoodle>`. (Note that **SnickerDoodle** is not actually an appropriate type for genre.) Make this same change to the **BookFileReader** class.

5. Similarly, in the **BookList** class, books of class **Book** are stored in an **ArrayList**. Replace the "raw type" **ArrayList**s in the **BookList** class with generic **ArrayList**s by putting the correct type between less than and greater than symbols as if they were parentheses. Note that the type for these **ArrayList**s is not the same as the type of the **ArrayList**s in **Book** or **BookFileReader**.

6. We have briefly discussed the **Comparable** interface in the class and the associated `compareTo()` method. What would be a good way of determining whether one book should be listed before or after another book? This is called the "Natural" ordering for the class. On the answer sheet, describe your method in English (you will write code for the method in a few moments). Make sure that your method for comparing takes into consideration the

aspects of the book relevant to ordering. Note that you should not consider genre for ordering, as different individuals may classify books into different genres.

7. Complete the implementation of the **Book** class. Make sure you fill in the class and method header comments where information is missing. First, read the entire `Book.java` file. After reading the file, add code to complete the implementation of the `toString()` and `compareTo()` methods. Also, be sure to show that **Book**s will be **Comparable**!

8. Why are getters and setters not required for the **Book** class? (Put your answer on your answer sheet.)

*Note:*
Before you can search a **List** using `binarySearch()`, you must sort the **List** by calling the `sort()` method of the **Collections** class. This method will call the `compareTo()` method of each item that is present in the **List**. Sample code that uses `sort()` is given below.

```
Collections.sort(list);
```

In order to search a **List** to find a particular object you must call the `binarySearch()` method of the **Collections** class. This method takes as a parameter an object (called the key) that represents the object we are searching for. If `binarySearch()` finds the key in the list, it will return the index to the item in the list that matches the key, otherwise it will return a negative integer. (In class at a later date, we will talk about how this negative integer is computed.) Sample code that uses `binarySearch()` to search for an item in a list is given below.

```
int index = Collections.binarySearch(list, key);
```

Yep, that is it. One line.

## *Instructions, continued:*

9. Complete the implementation of **BookList**, looking for all the TODO labels and following the directions for each.

10. Complete the implementation of the `main()` method in the `Lab2Driver`, looking for all the TODO labels and following the directions for each in order to try out your **Book** classes.

11. Ensure that there are no warnings generated for your code. **Do not suppress warnings.** Fix your code so that warnings are not necessary.

12. Export your completed answer sheet to PDF and submit it to the D2L dropbox for Lab 2 on or before **4:00 pm, Friday, 10 February 2017**.

13. Upload the completed Java source code files to Mimir for online checks on or before **4:00 pm, Friday, 10 February 2017**. (Note that the Mimir checks are not yet online. You will be notified when they are.)

Note that if your code is running correctly for the file provided, your output will look like this:

```
Error in input file. Cannot add same book twice.


Unsorted List:
There and Back Again, B.B., 3018, Autobiography
The Theory and Practice of Oligarchical Collectivism, Emmanuel Goldstein, 1937, History
The Chemical and Bacteriological Conditioning of the Embryo: Practical Instructions for Beta Embryo-
Store Workers, Anonymous, 602, Biological Sciences
On the Use of Mirrors in the Game of Chess, Milo Temesvar, 1934, History
Analysis: The Arrakeen Crisis, Princess Irulan, 10282, Politics
The Hitchhiker's Guide to the Galaxy, Megadodo Publications, 1999, Guides
Encyclopedia Galactica, Encyclopedia Galactica Publishing Co., 1997, Encyclopedia
Encyclopedia Galactica, Sebastian Lelong, 1942, Encyclopedia
Home Life and Social Habits of British Muggles, Wilhelm Wigworthy, 1987, Sociology


Sorted List:
Analysis: The Arrakeen Crisis, Princess Irulan, 10282, Politics
Encyclopedia Galactica, Encyclopedia Galactica Publishing Co., 1997, Encyclopedia
Encyclopedia Galactica, Sebastian Lelong, 1942, Encyclopedia
Home Life and Social Habits of British Muggles, Wilhelm Wigworthy, 1987, Sociology
On the Use of Mirrors in the Game of Chess, Milo Temesvar, 1934, History
The Chemical and Bacteriological Conditioning of the Embryo: Practical Instructions for Beta Embryo-
Store Workers, Anonymous, 602, Biological Sciences
The Hitchhiker's Guide to the Galaxy, Megadodo Publications, 1999, Guides
The Theory and Practice of Oligarchical Collectivism, Emmanuel Goldstein, 1937, History
There and Back Again, B.B., 3018, Autobiography


Searching
Key is Analysis: The Arrakeen Crisis, Princess Irulan, 10282,
Analysis: The Arrakeen Crisis, Princess Irulan, 10282, Politics
```