Lab Exercise #3 – Sorting and Searching Lists

Computer Science 2334

<u>Due by: Friday, February 7, 2014, 4:30 pm CST</u>

Members: ____________________

____

____

____

____

____

Objectives:

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

Instructions:

This lab exercise requires a laptop with an Internet connection. Once you have completed the exercises in this document, your group will submit it for grading. All group members should legibly write their names at the top of this lab handout.

Make sure you read this handout and look at all of the source code posted on the class website for this lab exercise before you begin working.

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

1. Download the `Lab3-eclipse.zip` project archive from the class website and import it into Eclipse. This archive contains the `VacationDestination.java` and `Lab3Driver.java` files. You will modify these files as a part of this lab exercise and submit the project archive when you are finished. But, before you start modifying these files, first answer the questions listed below.

2. Below, draw the UML diagram for the **VacationDestination** class.

3. Generics are a way to use general classes to deal with objects of specific types. In the **VacationDestination** class, the names of the landmarks of each country 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 landmark names? Replace the "raw type" **ArrayList**s in the **VacationDestination** 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 landmark name is **SnickerDoodle**, then you would replace each instance of `ArrayList` with `ArrayList<SnickerDoodle>`. (Note that **SnickerDoodle** is not actually an appropriate type for landmark name.)

4. 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 vacation destination should be listed before or after another vacation destination? This is called the "Natural" ordering for the class. Describe your method below in English (you will write code for the method in a few moments). Make sure that your method for comparing takes into consideration all aspects of the vacation destination relevant to ordering.

5. As a group, complete the implementation of the **VacationDestination** class. Make sure you fill in the class and method header comments where information is missing. First, read the entire `VacationDestination.java` file. After reading the file, add code to complete the implementation of the `toString()` and `compareTo()` methods. Why are getters and setters not required for this particular class?

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.

6. Complete the implementation of the `main()` method in the `Lab3Driver.java` test program in order to try out your **VacationDestination** class. Follow the steps specified below to finish the `main()` method.
a) Read through the entire listing of `Lab3Driver.java`.
b) Add code to create seven additional **VacationDestination** objects, initialize them, and add them to the list `destinations`. The code to create and add one **VacationDestination** has already been provided. You need to add code for seven more, for a total of eight.
c) Analyze the code that will print out the list. You need to understand how this works.
d) Add the code necessary to sort the list.
e) Provide the code that will print out the sorted list.
f) Analyze the code necessary to search the list for the “key” that is created in the base code. You need to understand how this works.
g) Add code that prints out the results of this search.

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

8. For bonus points, you may add code to the end of main to print the list again but this time use a `foreach` loop rather than a list iterator.

9. Submit the project archive following the steps given in the Submission Instructions on or before **4:30 pm, Friday, February 7, 2014** through D2L (learn.ou.edu).

10. Turn in this lab handout (with completed answers) to your lab instructor during lab hours or to Professor Hougen at his office by 4:30 pm on Friday, February 7, 2014 or earlier by bringing it to class on Thursday afternoon or to his office and handing it to him or sliding it under his office door if he is not available.