CS 3230 Lab #3
Library Functions: Searching and Sorting

Two tasks that are frequently found in large applications are searching and sorting. These tasks are common but not so common that they form part of a programming language directly. Instead, they are provided as library functions. The two are typically considered together for two reasons: First, many searching algorithms, like the binary and the interpolative search, require that the array or list in which they search be sorted or ordered. The second is more complex.

Sorting and searching functions ultimately compare array or list elements two at a time: are the elements the same and if not, which one comes first. The answer to this question is simple when the array or list elements are numbers or strings. However, when the array or list contains objects, the answer is not so straightforward. Objects are typically compared by comparing one or more of their attributes. But which ones?

When a general purpose searching and sorting library is created, the creator is faced with the problem of making it truly general. This implies that the library should be capable of dealing with numbers and with any object and with any set of attributes in that object. The solution is to require the user of the library functions to provide an ordering function. Given two objects from the array or list, the ordering function indicates if they are the same or which one comes first. It examines selected attributes within the object to make that determination. Based on the information provided by the ordering function, searching and sorting library functions can complete their tasks. Searching and sorting are typically considered together because they use the same ordering function.

The relationship between the three sections of code is illustrated at the right.

1. The user’s application calls the library code.

2. The searching and sorting library functions repeatedly call the user’s ordering function.

3. The user’s application, however, never directly calls the ordering function.

There are two ways to create the ordering function in a Java program:

1. implement the Comparable interface (in java.lang) and define a compareTo method; this implements the class’s “natural ordering”

2. create a new “helper” class that implements the Comparator interface (in java.util) and define the compare method
Assignment

1. Create a public class `Person` that implements the `Comparable` interface
   a. Instance Variables
      i. private int id;
      ii. private String name;
      iii. private String street;
      iv. private String city;
      v. private String state;
      vi. private String phoneNumber;
   b. Constructors:
      i. public Person(int num, String n, String s, String c, String st, String p)
      ii. public Person(String n) (initialize the name and sets the other values to null or 0 as appropriate)
      iii. public Person(int n) (initialize the id number and sets the other values to null)
   c. Methods
      i. public String toString() (which concatenates all Person instance variables)
      ii. public int getID() (accessor method for the id instance variable)
   d. The Comparable Interface
      i. Comparable is a generic interface (as of Java 5): `Comparable<Person>`
      ii. public int compareTo(Person person)
      iii. Compare two Person objects by name using the `String compareTo` method (this is done in your `compareTo` method) your method should return the value returned by the String method

2. Create a class named CompareInt
   a. implements `java.util.Comparator<Person>` (see p. 212)
   b. Defines a single method: `public int compare(Person p1, Person p2)` that compares two Person objects by id (call the `getID` method defined in class Person)

3. Complete the `SortPerson.java` class from the web page (edit points are noted with comments). Review the `sort` and `binarySearch` methods defined in both the `Arrays` and `Collections` classes of the `util` package. Be sure to read the comments at the top of the file.
   a. Use (i.e., call) the `sort` method to sort the list of Person objects alphabetically by name (use the single parameter sort method)
   b. Use the class `binarySearch` method to search for a name entered on the command line. The method requires two parameters: the list and the key. The key is a Person object that only contains a name
   c. Call the `sort` method to sort the list of Person objects numerically by id (use the two parameter version; the first is an list and the second is a comparator)
   d. Use the `binarySearch` method to search for an id entered on the command line. The method requires three parameters: the list to search, a key, and the comparator