Interfaces—The Sorted List Interface

In this assignment you will write a program that implements the sorted-list interface. In many applications we need quick access to items according to their value, not by their index, which is what arrays and ArrayLists accomplish. For example, we may want to search for the phone number of a friend in our contacts list - we search by their name. This search will be highly efficient if we keep the list in sorted order. We therefore consider the following interface:

// an interface for a data structure that maintains a list of // items in sorted order

public interface SortedListInterface {

    public boolean isEmpty();
    // returns true if the list is empty, false otherwise

    public int size();
    // returns the number of items in the list

    public void add(Comparable item);
    // adds an item to the list
    // precondition: none
    // postcondition: the list of items is in sorted order

    public void remove(Comparable item);
    // removes an item from the list
    // precondition: none
    // postcondition: removes an item that matches the given item from the list
    // the list is still in sorted order

    public boolean contains(Comparable item);
    // determines whether an item which matches the given item is in the list
    // precondition: none
    // postcondition: if an item that matches the given item is in the list,

    public Comparable get(Comparable item);
    // returns an item that matches the given item
    // precondition: none

Assignment 7

// postcondition: if an item matching the given item is in the list, // it returns the corresponding object; otherwise it returns null
}

Your task is to implement two classes: a SortedList class that implements the interface, and a class SortablePerson that will be used for testing your code.

Implementation Details. Your SortedList class needs to store its data in an array of type Comparable, keeping the items in sorted order (according to their compareTo method). It should have a constructor that takes no parameters, initializing an empty SortedList. The step of finding where in the list a given item is needs to be performed using binary search (you need to use it both for finding where in the list to insert a new item, and where an object is for the purpose of removing it). Note that you will need to resize the array as more objects are being added to the list.

Here is some client code that illustrates the use of the class:

SortedList list = new SortedList();
list.add((Integer)(10)); // the Integer class defines a compareTo method
list.add((Integer)(15));
list.add((Integer)(2));
System.out.println(list); // [2, 10, 15]
System.out.println(list.contains((Integer)(10))); // true
System.out.println(list.contains((Integer)(20))); // false
list.remove((Integer)(10)); // remove 10 from the list

As further test of your code you will implement a class called SortablePerson, that stores a person’s first name, last name, and a string of information. It should provide the following methods:

- Constructors SortablePerson(String firstName, String lastName, String info) and SortablePerson(String firstName, String lastName).
- sortByLast() which tells the class to sort names by last name and then by first name if the last names are the same.
- sortByFirst() which tells the class to sort names by first name and then by last name if the first names are the same. To implement these two methods we recommend keeping a static Boolean flag indicating how instances should be sorted. This information needs be shared across instances, so it makes sense to use a static variable.
- equals and compareTo methods.
- Getter and setter methods for firstName, lastName and info.
• And a `toString` method of course.

Our driver code may look something like:

```java
SortedList list = new SortedList();
list.add(new SortablePerson("Asa", "Ben-Hur", "1"));
list.add(new SortablePerson("John", "Smith", "2"));
list.add(new SortablePerson("Mark", "Riley", "3"));
System.out.println(list);
SortablePerson person = (SortablePerson)(list.get(new SortablePerson("Asa", "Ben-Hur")));
System.out.println(person.getInfo());
```

Note the last statement — we are retrieving from the list the object that can be identified by the given first- and last-name combination.

**Specifications, notes, and hints**

Your program needs to meet the following specifications:

• Submit the files `SortedList.java` and `SortablePerson.java`. Make sure each program has the specified methods, so our driver code would work.

• When commenting your code use Javadoc style comments at the beginning of each method.

• Put comments at the top of the file with your name, EID, date and course, and a short (one or two line) description of what the program does. We will be testing the code on the machines in the CS computer lab, so make sure your code runs on those machines.

• Submit your source code files via the checkin program by the due date (read the course syllabus for the late policy).