CSE 143, Summer 2012
Programming Assignment #1: SortedIntList (40 points)
Due Thursday, June 28, 2012, 11:30 PM

For your first programming assignment you are to write a class called `SortedIntList` that is a variation of the `ArrayIntList` class discussed in lecture. **Turn in two files** named `SortedIntList.java` and `SortedIntListTest.java` on the Homework section of the course web site

For this assignment you are to **write a class called** `SortedIntList` that is a variation of the `ArrayIntList` class written in lecture. Your class has two primary differences from the original list:

- A `SortedIntList` must maintain its list of integers in **sorted order** (non-decreasing).
- A `SortedIntList` has an option to specify that its elements should be unique (**no duplicates**).

The new class should have the same public methods as the old class except for the two-parameter `add` method that adds a value at a particular index and the `set` method. Because we want to keep the list in sorted order, we won't allow the client to specify where something should go, so these methods should not be public methods of the new class.

Two of the methods should be rewritten. The single-parameter `add` method should no longer add at the end of the list. It should add the value in an appropriate place to keep the list in sorted (non-decreasing) order. The `add` method also has to pay attention to whether the client has requested "unique" values, in which case it has to be careful not to add any duplicates to the list. Your `add` method should not re-sort the entire list (either by calling `Arrays.sort` or something you wrote yourself). Finding the correct position and inserting the new element at that position is more efficient than resorting the whole list.

In addition, the `indexOf` method should be rewritten to take advantage of the fact that the list is sorted. It should use the much faster **binary search** algorithm rather than the sequential search algorithm that is used in the original `ArrayIntList` class (more on this later). We will also slightly change the postcondition of `indexOf`. The `ArrayIntList` version promises to return the index of the first occurrence of the search value. Because we are using a binary search, we will instead guarantee only that the index is for *an* occurrence of the search value (not necessarily the first one).

The other methods from `ArrayIntList` like the remove method can be used without modification in the new class. You should comment each method in your class. If you borrow the comments from the `ArrayIntList` class, then be sure to comment the new methods in a similar manner.

Your new class will have an extra piece of state information. Each `SortedIntList` will keep track of whether or not it is limited to unique values. Think of it as a sort of on/off switch that each list has. If the unique switch is set to off (false), then the list adds everything, even if that leads to duplicate values. If the unique switch is set to on (true), then calls on `add` that pass values already in the list have no effect (i.e., `add` ensures that no duplicates are added). For example, if you start with an empty list that has the unique switch off, then adding three occurrences of the number 42 will generate the list [42, 42, 42]. Adding those same three occurrences of 42 to an empty list that has the unique switch on will generate the list [42].

This extra piece of state will require the addition of several new methods. Your class should keep the DEFAULT_CAPACITY constant and should have a total of four constructors:

Method	Description
`SortedIntList(boolean unique, int capacity)`	This should construct a list with given capacity and with the given setting for whether or not to limit the list to unique values (true means no duplicates, false means duplicates are allowed). If the capacity is negative, an **`IllegalArgumentException`** should occur.
`SortedIntList(int capacity)`	This should construct a list with the given capacity and with unique set to false (duplicates allowed). If the capacity is negative, an **`IllegalArgumentException`** should occur.
`SortedIntList(boolean unique)`	This should construct a list of default capacity with the given setting for unique (true means no duplicates, false means duplicates are allowed)
`SortedIntList()`	This should construct a list of default capacity with unique set to false (duplicates allowed)

Your class should include the following four new methods:

Method	Description
`int min()`	This method should return the smallest element value contained in the list. For example, in the list [4, 4, 17, 39, 58], the min is 4. If the list is empty, it has no minimum element, so you should throw a **`NoSuchElementException.`**
`int max()`	This method should return the largest element value contained in the list. For example, in the list [4, 4, 17, 39, 58], the max is 58. If the list is empty, it has no largest element, so you should throw a **`NoSuchElementException.`**
`boolean getUnique()`	This method should return the current setting for unique (true means no duplicates, false means duplicates are allowed)
`void setUnique(boolean value)`	This method allows the client to set whether or not to allow duplicates (true means no duplicates, false means duplicates allowed)

The `setUnique` method presents a potential problem for us. Suppose that the client has constructed a list and has added many values, including duplicates. If the client then tries to set unique to true, this is supposed to prevent duplicates. But the duplicates are already there. In this case, the `setUnique` method should remove the duplicates and should guarantee that no additional duplicates are added unless the client changes the setting back to false. If the client changes the setting back to false, that will mean that duplicates can then be added, but this doesn't have to behave like an "undo" operation that would put back duplicates that you previously removed.

You are required to use the built-in `Arrays.binarySearch` method discussed in section for all of your location testing (Where is a value? Where would I insert a new value?). You can find the documentation for it in the Java API documentation that is the first link under "Links" on the class web page. Keep in mind that because this is a static method, you have to use the class name in calling it. For example, to find the location of the value 42 in an array called `data` when we want to explore the first twelve elements, you'd say something like:

```
int index = Arrays.binarySearch(data, 0, 12, 42);
```

Remember that the "from index" is inclusive, which is why we use 0, and the "to index" is exclusive, which is why we use 12.

To get access to the Arrays class, you should include this line at the beginning of your class:

```
import java.util.*;
```

Testing Program (`SortedIntListTest.java`):

Along with your `SortedIntList`, turn in a short **testing file** named `SortedIntListTest.java`. This program should test your implementation of `SortedIntList` by creating at least two lists, adding some elements to them, calling various methods, and checking the expected results. This part of the assignment is worth only a few points, but we want you to practice testing your own code. For full credit, your testing file must contain at least 2 test methods, and you must call at least 3 of the different methods on the list(s) you create.

There are also several provided testing programs on the course web site. You may ask, "Why do I need to write my own testing program when these testing programs are already provided?" Our testing programs are large and complex. It can be good to have a smaller test that runs just a few methods that you are working on. We encourage you to add to this program as you develop your `SortedIntList`. If you want to write a more complex test, that is fine, but not required.

Development Strategy

One of the most important techniques for software professionals is to develop code in stages rather than trying to write it all at once (the technical term is *iterative enhancement* or *stepwise refinement*). It is also important to be able to test the correctness of your solution at each different stage.

We have noticed that many 143 students do not develop their code in stages and do not have a good idea of how to test their solutions. As a result, for this assignment we will provide you with a development strategy and some testing code. We aren't going to provide exhaustive testing code, but we'll give you some good examples of the kind of testing code we want you to write.

We are suggesting that you develop the program in three stages:

1. For this version, we won't worry about the issue of unique values. We just want a basic version of the class that keeps a list in sorted order and that uses binary search to speed up searching. This stage involves all of the following (not necessarily in this order):
    a. Copy `ArrayIntList.java` to `SortedIntList.java` and change all occurrences of `ArrayIntList` to `SortedIntList`. That way you'll have a new class that has the same behavior as the old `ArrayIntList` class (including the two constructors).
    b. Make sure that the two-argument `add` method is no longer a public method.
    c. Modify the one-argument `add` method so that it preserves sorted order. You may need to use binary search if your solution involves two steps (first locate, then insert).
    d. Modify `indexOf` so that it uses the binary search method.
    e. Add the `min` and `max` methods.

2. Modify your code so that it keeps track of whether or not the client wants you to guarantee that the list has unique values. Add the two constructors that involve setting unique and modify `add` so that it doesn't add duplicates if unique is set to true.

3. Modify your code to have `getUnique` and `setUnique`. Remember that if the client calls `setUnique` and sets the value to true, you have to eliminate any duplicates that might be in the list.

References

Textbook Chapter 15 covers the implementation of `ArrayIntList` in detail.

Textbook sections 13.1 and 13.3 discuss the binary search algorithm in more detail. 13.3 discusses how it is implemented and its return values in detail. Seeing how it is implemented may help you understand how to use it appropriately.

Style Guidelines and Grading

You may not use any features from Java's collection framework on this assignment, such as `ArrayList` or other pre-existing collection classes. You also may not use the `Arrays.sort` method to sort your list.

A major focus of our style grading is **redundancy**. As much as possible you should avoid redundancy and repeated logic within your code, such as by factoring out common code from `if/else` statements, creating "helper" methods to capture repeated code, or having some of your methods/constructors call others if their behaviors are related. Any additional methods you add to `SortedIntList` beyond those specified should be `private` so that outside code cannot call them. Note that even constructors can be redundant; if two or more constructors in your class (or the superclass) contain similar or identical code, you should find a way to reduce this redundancy by making them call each other as appropriate.

Properly **encapsulate** your objects by making any data fields in your class `private`. Avoid unnecessary fields; use fields to store important data of your objects but not to store temporary values only used within a single call to one method. Fields should always be initialized inside a constructor or method, never at declaration.

You should follow good **general style** guidelines such as: appropriately using control structures like loops and `if/else` statements; avoiding redundancy using techniques such as methods, loops, and `if/else` factoring; properly using indentation, good variable names, and proper types; and not having any lines of code longer than 100 characters.

Commenting will be more of a style focus on this program (and future programs) than it was in 142. You should comment your code with a heading at the top of your class with your name, section, and a description of the overall program. Also place a comment heading atop each method, and a comment on any complex sections of your code.

Not only will we look for the existence of comments on this assignment, but we also have stricter expectations about the quality of their content. Comment headings should use descriptive complete sentences and should be written in your own words, explaining each method's behavior, parameters, return values, and pre/post-conditions as appropriate. If the method potentially throws any exceptions, comment this and explain what exceptions it throws and under what conditions it will throw them. Be concise but specific in your comments. The `ArrayIntList` class is a good example of appropriate commenting style for this assignment. (Your test program should have an overall descriptive comment header at the top of the class, but it does not need commenting on each testing method or otherwise throughout the code.)

For reference, our `SortedIntList.java` is around **130 lines long** including comments (and has **60 "substantive" lines** according to our Indenter tool on the course web site). But you do not have to match this; it's just listed as a sanity check.