CSE143—Computer Programming II
Programming Assignment #1
due: Thursday, 6/30/11, 11:00 pm

For your first programming assignment you are to create a class called SortedStringArrayList that is a modified version of the StringArrayList class discussed in lecture. The primary differences are that your new class must maintain the list of Strings in sorted (non-decreasing) order and that it will have an extra option to specify whether the strings should be unique or whether duplicates are allowed.

In addition to the SortedStringArrayList class, you are to create a modified version of the StringArrayListTest class called SortedStringArrayListTest. This class should test your new SortedStringArrayList class the same way that the original collection of tests checked the original StringArrayList class.

The new SortedStringArrayList 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 that stores a value at a particular index. Because we want to keep the list in sorted order, we won't allow the client to specify where something should go, so these method should not be a 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 StringArrayList class (more on this later). We will also slightly change the specification of indexOf. The StringArrayList 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).

You may need to compare strings to see which one is "less" than the other. While we can compare ints and other primitive Java types using the < operator (i.e., k<17), for Strings and similar objects we need to use the compareTo method. If s and t are Strings, s.compareTo(t) returns a negative integer value if s is "less" than t, zero if they are equal, and a positive integer value if s is "greater" than t. The ordering corresponds to alphabetical order for strings that contain lower-case letters only, and it is the same ordering used by the binary search algorithm (see below) that you will use in your code.

The other methods from StringArrayList like the remove method can be used without modification in the new class. You should comment each method in your class. Since your new class will be a modified version of the original one, you should feel free to use the code and comments from that class as a starting point. If you borrow the comments from the StringArrayList 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 SortedStringArrayList 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 string "xyz" will generate the list ["xyz", "xyz", "xyz"]. Adding those same three occurrences of "xyz" to an empty list that has the unique switch on will generate the list ["xyz"].

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
SortedStringArrayList (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)
SortedStringArrayList (int capacity)	This should construct a list with the given capacity and with unique set to false (duplicates allowed)
SortedStringArrayList (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)
SortedStringArrayList ()	This should construct a list of default capacity with unique set to false (duplicates allowed)

Your class should include the following two new methods:

Method	Description
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 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 "useful links" on the assignment 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 “xyzzy” 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, "xyzzy");
```

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.*;
```

In terms of correctness, your class must provide all of the functionality described above and the execution time for indexOf must indicate that you are using a binary search rather than a sequential search. In terms of style, we will be grading on your use of comments, good variable names, consistent indentation, and good coding style to implement these operations. Be sure to update and add to comments in the existing code to match the changes you make to the code.

Your SortedStringArrayList class should be stored in a file named SortedStringArrayList.java.

Test Code

In addition to the SortedStringArrayList class, you should produce a set of tests in a class named SortedStringArrayListTest stored in a file SortedStringArrayList.java. The test code will be evaluated for both correctness and style and should contain a comprehensive, but not excessive collection of tests that verify that any SortedStringArrayList class (not just yours, but ones we might provide) works properly. You should base your code on the StringArrayListTest class provided with this assignment, deleting tests that do not apply and adding ones needed to verify that the new list keeps data properly sorted and handles the “unique” setting properly, including when the user calls setUnique(true), which removes duplicates from a list if they are present.

A good set of tests will check both typical cases and edge cases. For instance, does the code work right on an empty list, a list with one item, a list with many items? Does it work right if strings are added to it in a random order, alphabetical order, reverse alphabetical order? Do operations work correctly at the beginning or end of the list as well as in the middle? But there’s no point in having redundant tests that check the same thing. For instance, if a series of tests check the behavior of a list of 5 items, there’s probably no point to repeating basically the same tests on a list of length 6. Try to be sure each test or sequence of tests check something different– either different tests, or the same tests run on data that has different characteristics.

More test code is not necessarily better. You should not need to write huge numbers of tests to check your SortedStringArrayList class for this assignment.

One thing you do not need to include in your SortedStringArrayListTest class is tests that verify that exceptions are thrown when expected. Mechanisms for doing this cleanly so that additional tests can be performed afterwards are beyond what we’ve seen so far.

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.

Many students do not develop code in stages and do not have a good idea of how to test their solutions. In lecture we have already talked about writing tests and developing code incrementally, testing each small change as you go, and if you discover a bug, add a test that fails when the bug is present and works when the bug is fixed, which will guard against the same bug recurring in the future.

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 StringArrayList .java to SortedStringArrayList.java and change all occurrences of StringArrayList to SortedStringArrayList. That way you'll have a new class that has the same behavior as the old StringArrayList 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.

   As you perform these steps, modify the corresponding tests in SortedStringArrayListTest and add new tests as needed to check that your new code works properly. You should also update the existing comments and write new ones while you write the code that goes with them.

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.

Each time you change the code you should create appropriate new tests and run both the new and old tests to verify that your new code works and that code that was previously working is still okay. If you write and run small tests as you make small changes you should find that you are able to develop a working solution to the assignment more smoothly and with less effort than you might expect.

What to Turn In

When you are done, turn in your SortedStringArrayList.java and SortedStringArrayListTest.java source files using the online turnin link on the course "homework" web page.