Laboratory 13

CS 1323, Spring 2015

# Learning Objectives

1. Insert elements into an oversized array, preventing duplicates from being inserted using the Arrays.binarySearch() method. (10 points)
2. Keep an array in sorted order using insertions. (20 points)
3. Perform linear search for partial matches in a sorted array using the startsWith() method in the String class. (20 points)
4. Create a nested loop to perform auto-completion. (10 points)
5. Use at least four methods, with well-chosen parameters and return types. (20 points)
6. Create a menu driven interface. (10 points)

10 points will be awarded for the documentation of your program. That means using good names for variables, proper and consistent indentation of code, and meaningful use of whitespace.

Section 10: Have the TAs check your assignment by the end of the laboratory or submit it on Janux by 11:59 p.m. on April 22.

Section 1: Submit your program on Janux by 11:59 p.m. on April 22.

# Description

You have probably noticed that many pieces of software use auto-completion to accelerate data entry. Examples of programs that typically use auto-completion include eclipse, text messaging, looking up contacts in an address book, and email addresses in Outlook. In this project we will create a program that auto-completes email addresses.

An interface for the program is shown below (user input is in bold)

1. Enter a new email address

2. Find an existing email address

3. Quit

What is your choice?

**1**

Enter the email address:

[**dtrytten@ou.edu**](mailto:dtrytten@ou.edu)

Insertion successful

// Repeat menu

What is your choice?

**1**

Enter the email address:

[**dtrytten@ou.edu**](mailto:dtrytten@ou.edu)

That email address is already inserted

// Repeat menu

What is your choice?

**1**

Enter the email address:

[**dt2@ou.edu**](mailto:dt2@ou.edu)

Insertion successful

//Repeat menu

What is your choice?

**2**

Enter the first letters, one at a time

**d**

[dtrytten@ou.edu](mailto:dtrytten@ou.edu)

[dt2@ou.edu](mailto:dt2@ou.edu)

**t**

[dtrytten@ou.edu](mailto:dtrytten@ou.edu)

[dt2@ou.edu](mailto:dt2@ou.edu)

**2**

Found [dt2@ou.edu](mailto:dt2@ou.edu)

// Repeat menu

What is your choice?

3

Bye.

This program uses an oversize array. Create an array that is larger, and keep track of the number of elements that are currently stored in the array. When the array is constructed, the number of elements stored should be zero. As elements are added to the array, this value will increase. You may assume that 100 elements is sufficient for the array, and this value should be stored in a constant so it can easily be changed later if needed.

## Inserting Data

When you go to insert data in this array, you need to be sure that it isn’t already there. This is best done using Arrays.binarySearch(). Remember that you must have the data in sorted order. The binarySearch algorithm uses the compareTo() method in the String class to determine order, so you must use this same method when you are building the array in sorted order.

String first = “aaa”;

String second = “bbb”;

first.compareTo(second) will return a negative value.

second.compareTo(first) will return a positive value.

second.compareTo(second) will return 0.

Inserting the first element in an array will retain sorted order, but later insertions will not. You therefore have to perform insertions in such a way that the data remains in sorted order. This could be done with a method like Arrays.sort(), but this is inefficient. When data is nearly sorted, one iteration of the outer loop (not the full algorithm) of insertion sort is a great choice. Follow the example below (I’m using integers to keep it simple, but the idea is the same with email addresses). The array originally contained 1, 3, 5, 7. The value 2 is to be added to the array. Notice that the last two entries of the array are not currently in use.

Suppose the value 2 is to be inserted. Rather than putting 2 at the end of the array, we will move other elements to the right until the proper place for 2 is found. This is exactly like one outer loop of insertion sort.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 5 | 7 |  |  |  |
|  |  |  |  | 7 |  |  |
|  |  |  | 5 |  |  |  |
|  |  | 3 |  |  |  |  |
|  | 2 |  |  |  |  |  |

It is possible to perform this operation using an different algorithm. Insert the new element at the end of the array, and swap it with the previous one until it is in sorted order. The steps to this algorithm are shown below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 5 | 7 | 2 |  |  |
|  |  |  | 2 | 7 |  |  |
|  |  | 2 | 5 |  |  |  |
|  | 2 | 3 |  |  |  |  |
|  |  |  |  |  |  |  |

You may use linear search to find all of the addresses that match the current input. You recall that the String class has a method called startsWith() that we used in an earlier project. It could come in handy in the linear search for a partial match.

## Class Constants

Creating a menu driven interface is relatively simple, except that the parameter passing gets ugly without one programming trick. If you declare constants like ADD, QUIT, and SEARCH in the main program, you have to bring them into other methods as parameters. An easy work around for this is to create class data. This is done by putting the parameters in the class body, but not inside of any method. Make them static (constants should always be class data, remember). When they are declared this way, they may be used in any method inside of the class.

**public** **class** AutoCompletion

{

**public static** **final** **int** *ADD* = 1;

**public** **static** **final** **int** *SEARCH* = 2;

**public** **static** **final** **int** *QUIT* = 3;

}