Project 4

CS 1323, Fall 2017

# Learning Objectives

1. Insert elements into an array. (10 points)
2. Prevent duplicates from being inserted using the Arrays.binarySearch() method. (20 points)
3. Perform linear search for partial matches in a sorted array using the startsWith() method in the String class. (10 points)
4. Perform error checking for incorrect menu choices, inserting data that is already in the array, and being unable to insert data because the array is full. (10 points)
5. Use at least four methods, with well-chosen parameters and return types. (20 points)
6. Use constant class data. (10 points)
7. 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, sufficient explanatory comments, proper and consistent indentation of code, and meaningful use of whitespace.

Submit the finished work ***individually*** on Canvas by 11:59 p.m. on Friday, October 20.

# Description

You have probably noticed that many pieces of software allow searches to be completed with partial data. For example, on your cell phone you usually have to enter only a few characters of someone’s name to get a match. We’re going to look at how this can be implemented in Java.

An interface for the program is shown below. This program uses email addresses, but the same principles apply when names are used.

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

1

Enter the email address

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

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

1

Enter the email address

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

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

1

Enter the email address

a@ou.edu

That email address has already been inserted

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

1

Enter the email address

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

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

2

Enter the partial address

a

1. a@ou.edu

2. ab@ou.edu

3. [ac@ou.edu](mailto:ac@ou.edu)

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

2

Enter the partial address

ac

1. [ac@ou.edu](mailto:ac@ou.edu)

Please choose from the following menu of choices:

1. Enter a new email address

2. Find an existing email address

3. Quit.

What is your choice?

3

This program uses a single large array. You need to create an array that is too large, 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.

Email addresses are not case sensitive. For example [AB@ou.edu](mailto:AB@ou.edu) is the same as [ab@ou.edu](mailto:ab@ou.edu), and [Ab@ou.edu](mailto:Ab@ou.edu).

## 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 to use binary search.

The binarySearch method returns a non-negative value when the item is found. When the item is not found, it returns –insertionPoint -1. This means that you can use the value returned by binarySearch to keep the data sorted.

Let’s work through an example. Suppose that an oversize array contains four elements {“a”, “c”, “e”, “f”,…}. If you search for “b”, the value would be inserted at position 1. This means that -2 would be returned (-insertionPoint -1 = -1 -1). When you know the return value for binary search you can find the insertion point. The insertion point is then used to shift data over to make space for the new value. For example, in this case, “f” would shift over, then “e”, then “c”, then “b” would be inserted. This is a simple loop—it just starts at the far end of the array instead of the near end. The detailed operations are shown below. Make sure you understand why we have to work from the right side towards the left instead of left to right as we often do.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | c | e | f |  |
|  |  |  |  | f |
|  |  |  | e |  |
|  |  | c |  |  |
|  | b |  |  |  |

You may use linear search to find all of the addresses that match the current input. The String class has a method called startsWith().

String name = “Abby”;

name.startsWith(“Ab”) returns true. name.startsWith(“AB”) returns false.

## Method Signatures

The signatures for the methods are shown below.

public static void search(String[] data, int size, String target)

This method searches for a partial match of target and prints out any element of data (from 0 to size-1) that starts with the same characters as target.

public static int menuChoice(Scanner keyboard)

This method prints out the menu, allows the user to enter a choice, and checks to see that this choice is one of the legal ones. If the choice is not legal, the user should be given a chance to enter it again. If it is legal, the computation should proceed.

public static int addNewEmail(String[] data, int size, String insertMe)

This method should add insertMe to the array data if the String value is not already in the array. The int that is returned is the size of the array after the insertion. This size could be the same as it was previously (for example, if insertMe is already in the array) or may be one larger.

The fourth method you use will be the main method.

## 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. 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;

}