CSCI 1933 Lab 5

Arrays, Searching, Sorting, and Some file I/O

Rules

All steps should be completed primarily in the lab time; however, we require that at least the starred steps (★) to be completed and checked off by a TA during lab time. Starred steps cannot be checked off during office hours. You have **until the last office hours on Monday** to have non-starred steps checked by ANY TA, but we suggest you have them checked before then to avoid crowded office hours on Monday. You may work individually or with *one* other partner on this lab. When working in pairs, both individuals must go to office hours to get checked off. There is nothing to submit via Moodle for this lab. This policy applies to all labs.

1 Build a Contact Class ★

Create a `Contact` class, we will have the following private variables:

- `String name`
- `long phone`
- `String address`
- `String comments`

as well as the following methods:

- `getters` and `setters` for each member variable
- `public String toString()` – Returns a `String` containing all of the info pertaining to the Contact. The `name` should be the first thing in the `String`.
- `public boolean equals(Contact other)` Returns true if every attribute of `other` is equal to yourself.

> **Tip:** A newline character is represented in Java as \n. A tab character is represented as \t.

You decide what constructors to include in the `Contact` class that make it easy to use. The `Contact` class should all be in a seperate Contact.java file.

Write a main method within Contact.java to test all your methods.

2 Build a ContactListClass ★

Create a class called `ContactList` that contains an array of `Contact` objects.

Include a local variable, `ptr`, that holds the index of the most recent `Contact` object referenced. Initially, set `ptr = -1`. Each time a `Contact` is added, looked up, manipulated, etc., modify `ptr` to the index in the array of the referenced `Contact object`.

Additionally, include the following methods in the `ContactList` class:

- You will want to include **two** Constructors for `ContactList` - a default Constructor that sets the length of the array to 20, and another that takes in a length parameter.
- `public boolean add(Contact c)` – inserts a new `Contact` object into the `ContactList` array.
  Return `true` if successful, and `false` if the `ContactList` array is full.
- `public Contact find(String name)` – returns the `Contact` object found containing `name`. That is, if we have a `Contact jackson` with the name field "Jackson", then running `find("Jack")` will return jackson. Begin searching starting from the object to the right of `ptr`. If needed, wrap around the array to index 0 and back to the current position of `ptr`. If nobody matches, return `null` and leave `ptr` where it was.

  > **Tip:** Java contains a built in `contains(String s)` method. To save time on the lab, you may use this built in method. It is recommended that you try and write your own contains method if you have extra time. For those of you who want to write your own, this is the strategy. You will use nested loops and compare the input string with the string you are comparing too. You will want to make use of the following String methods: `length()` and `charAt(int i)`.

- `public Contact remove()` – removes the "current" contact (the one at the index of `ptr`) and returns it. You may either insert a `null` at the location the item was removed from or compact the array so that there is no longer a hole. Consider which is a more efficient method, and in which cases.

The `ContactList` class should all be in a separate ContactList.java file. Remember to write test cases for `ContactList`.

3 Save Your Contacts to a Text File

It'll be easier to work on this lab if you can save your contact information to a file and read it back in.
To do this, let's add two methods to our ContactList class:

- `public boolean write(String fileName)` – writes the `ContactList` information to a textfile with the name `fileName`. Use your `toString` method. This returns `true` if the write was a success or `false` if there was an error.
- `public boolean read(String fileName)` - reads the `ContactList` information from a textfile with the name `fileName`. This will add new `Contact` objects into the array - so you will need to use `Scanner` to pick apart the data you wrote to the text file in the previous method. This returns `true` if the read was a success or `false` if there was an error.

To do this, we will need the following imports. Place these at the top of your file:

```
import java.util.Scanner;
import java.io.PrintWriter;
import java.io.File;
```

To read from an arbitrary textfile, do the following:

```
// assume our filename is stored in the string fileName
Scanner s = null; // declare s outside try-catch block
try {
  s = new Scanner(new File(fileName));
} catch (Exception e) { // returns false if fails to find fileName
  return false;
}
// Now use s in the same way we used Scanners previously for user input
```

To write to an arbitrary textfile, do the following:

```
// assume our filename is stored in the string fileName
PrintWriter p = null; // declare p outside try-catch block
try {
  p = new PrintWriter(new File(fileName));
} catch (Exception e) {
  return false;
}
```

```
// Now use p the same as we print with System.out.println()
p.print("Hello, "); // does not write a newline character
p.print("World!"); // will go on the same line as "Hello, "

p.println("Goodbye "); same line, addes newline character
p.println("Cruel World"); // writes on the line after "Goodbye "
p.close() // close the file to preserve what was written to it
```

Modify your main method to test these two new methods.

4 More Functionality

Let's add some more functionality to our ContactList class. Add the following methods:

- `public Contact getCurrent()` – returns the `Contact` that is currently being pointed to (the contact at index `ptr`)
- `public Contact get(int i)` – returns the `Contact` at index `i` in the array. If the array is empty, or i is outside the size of the array, return `null`.
- `public Contact next()` – increments `ptr` by one so it now points to the next `Contact`. Return the `Contact` in the new location that `ptr` is pointing to. If `ptr` is at the end of the array, move it to the beginning. If the array is empty, do nothing and return `null`.
- `public Contact previous()` – same thing as `next()`, but decrements `ptr`. Use similar wrapping if `ptr` is at the beginning of the array.

Again, test these methods in the main method of ContactList.

5 Sort it

Include one more method to `ContactList`:

- `public void sort()` – You may use any of the three sorting algorithms discussed in lecture (bubble sort, insertion sort, selection sort). The Contacts should be sorted in "roughly" alphabetical order using the `compareTo(String anotherString)` method.

> **Tip:** Recall that `compareTo` returns a negative integer if this String precedes `anotherString`, 0 if this String is the same as `anotherString`, and a positive integer if this String follows `anotherString`.

Yet again, test your `sort()` method in the main method.