ITSC 1213 - Working with ArrayLists Part 2

Introduction

The goal of this lab is to practice working with algorithms to search and sort an ArrayList of objects.

Concepts covered in this lab:

- Using search algorithms to find a specific element in an ArrayList
- Using sort algorithms to rearrange elements in an ArrayList

Required files or links

- This lab assumes you have successfully completed the Working with ArrayLists Part 1 lab. Consult with the instructional team if you need guidance
- This lab assumes you have successfully completed sections 7.5 and 7.6 in the CS Awesome textbook

Part A: Implement a method that uses sequential search to find a specific order in a list of burger orders.

1. Open the NetBeans Project you called `FastFoodKitchen` created in the previous lab.

2. Open the `FastFoodKitchen` class file and add the following method. The bullet points under the method signature tell you what the method must do.

   `public int findOrderSeq(int orderID)`
   - Using the value of the parameter `orderID`, find the position of this order in the list using sequential search algorithm (refer to section 7.5 if you need a refresher)
   - Return the position of the order
   - If an order does not exist in the list that matches `orderID`, return -1

3. Time to test your new method. Rather than using the main class we used in the previous lab let's create a new class with a main method and give it a unique name (e.g., Module3Test). Now add the following code to the main method to test your new `findOrderSeq` method:

```
public static void main(String[] args) {
    FastFoodKitchen kitchen = new FastFoodKitchen();

    // Part A
    int orderPosition = kitchen.findOrderSeq(2);
    System.out.println("Using sequential search >> order position is "
            + orderPosition);
}
```

Note that the value of `orderPosition` will depend on what value you used to initialize the `nextOrderNum` field. In your output you might see a 2 (if you started at 0), 3 (if you started at 1) or -99 (if you used a different value e.g., 10).

4. Now, run your program to test the newly added method. Experiment with different order numbers and fix any bugs you find.

5. When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part B.

Part B: Implement a method that uses selection sort to rearrange the elements in an array

1. In this part we will add a method to our `FastFoodKitchen` class that implements selection sort to rearrange the orders in the list of burger orders based on the total number of burgers. Add the following method. The bullet points under the method signature tell you what the method must do.

   `public void selectionSort()`
   - Sorts `orderList` using selection sort algorithm. (refer to section 7.6 if you need a refresher). The algorithm should perform the sorting based on the total number of burgers in each order (order size), regardless of their type (e.g. if on order has 2 cheeseburgers and 1 veggieburger the total would be 3)

2. Time to test your new method. To do this we want to be able to access the `orderList` object to check whether it is sorted or not. Since we didn't explicitly ask you to create a getter method for the `orderList` field let's make sure you add that to the `FastFoodKitchen` class.

```
public ArrayList<BurgerOrder> getOrderList() {
    return orderList;
}
```

3. Now, switch over to the main method (in the test class you used in Part A) and add the following code to test your new `selectionSort` method (you can comment the code for Part A, but keep the statement that creates the `FastFoodKitchen` object):

```
// Part B
kitchen.selectionSort();
ArrayList<BurgerOrder> sortedOrders = kitchen.getOrderList();
int orderSize = 0;
for (BurgerOrder order : sortedOrders) {
    orderSize = order.getNumCheeseburgers() + order.getNumHamburger()
            + order.getNumVeggieburgers();
    System.out.println(order.getOrderNum() + " has " + orderSize
            + " burgers.");
}
```

4. Now, run your program to test the newly added method. Verify that the list of order is now sorted based on the order size not on how/when orders were added to the list.
5. When your program works and you are satisfied with the result, show your work to the instructional team and proceed to Part C.

Part C: Implement a method that uses insertion sort to rearrange the elements in an array

In this part we will add a method to our `FastFoodKitchen` class that implements insertion sort to rearrange the orders in the list of burger orders based on the total number of burgers.

1. Add the following method. The bullet points under the method signature tell you what the method must do:
   `public void insertionSort()`
   - Sorts orderList using insertion sort algorithm. The algorithm should perform the sorting based on the total number of burgers in each order, regardless of their type (e.g. if on order has 2 cheeseburgers and 1 veggieburger the total would be 3)

2. Switch over to the main method to test your new method. Comment the line from Part C that calls the `selectionSort` method and add a statement to call your new `insertionSort` method

```
// Part C
kitchen.insertionSort();
```

3. Now, run your program and verify that the list of order is now sorted based on the order size not on how/when orders were added to the list.

4. When your program works and you are satisfied with the result, show your work to the instructional team and proceed to Part D.

Part D: Implement a method that uses binary search to find a specific order in a list of burger orders.

1. Now that we can ensure we have a sorted list we can try another search algorithm that works only with sorted arrays/lists.
2. Let's add a new method to our `FastFoodKitchen` class that performs a binary search on our order list. The bullet points under the method signature tell you what the method must do.
   `public int findOrderBin(int orderID)`
   - Using the value of the parameter `orderID`, find the position of this order in the list using binary search algorithm
   - Return the position of the order
   - If an order does not exist in the list that matches `orderID`, return -1

3. To test your new method. Switch over to the class you used in Part A to test the search method and add the following code to the main method to test your new `findOrderBin` method (if you comment any code make sure to keep the call one of the sort methods before you test your new method):

```
// Part D
orderPosition = kitchen.findOrderBin(2);
System.out.println("Using binary search >> order position is "
        + orderPosition);
```

4. Now, run your program to test the newly added method. Remember to test your code with different values. If you encounter any issues use the debugger tool to inspect your code during execution. This is a great way to reinforce your learning and can help you fix any bugs you might encounter.
5. When your program works and you are satisfied with the result and JavaDoc, show your work to the instructional team to be checked off for this lab.

So what did you learn in this lab?

- How to implement sequential and binary search algorithms for an ArrayList of objects
- How to implement selection and insertion sort algorithms for an ArrayList of objects
- How to write code to specifications (the methods you added to the FastFoodKitchen class)
- Practiced testing and debugging a program