ITSC 1213 - Working with ArrayLists Part 1

Introduction

The goal of this lab is to practice working with ArrayLists.

Concepts covered in this lab:

- Creating classes with mutator and accessor methods
- Using ArrayList methods to access and modify elements in an ArrayList
- Using iteration to traverse ArrayLists

Required files or links

- Part B will require code from this link: https://gist.github.com/nanajjar/c9ee149e05886d7694bafb397798b8e5

Part A: Create a new class that represents an order at a fast-food burger joint. This class will be used in Part B, when we work with a list of orders.

1. Create a new project in NetBeans called `FastFoodKitchen` with a main class. Remember that the main class is the class we will use to add code that will be executed when we run the project. We will refer to this class as the main class but you are free to call it any valid Java class name (e.g., Main). For a review on writing classes and objects review Unit 5 of the CS Awesome textbook.

2. Right-click on the package that has the main class to create a new class. Select New → Java Class. Call your new class `Order`, and click Finish.

3. Now start adding instance variable definitions to your `Order` class.
   Click the blank line below `public class Order {`. Create a private integer field called `numHamburgers`, and initialize it to **0**.

4. We also want to add a getter and setter methods for this field. Add the standard getter method that returns the value of `numHamburgers`.

5. For the setter method we want it to check that if the parameter passed in is less than zero, an error message is printed out and the field value is not changed. If it is not less than zero, then change the field value to the parameter passed.

❖ This would be a good time to pause for a moment and review what just happened. Note how the getters and setters use the parameters that were

passed in to set the values of the fields for this class. Later, when we create objects of this class, you'll see how this capability is used.

6. Add three more private fields to this class: `int numCheeseburgers`, `int numVeggieburgers`, and `int numSodas`, following the same process, and initializing each to **0**.

7. Add getters and setters similar to what we did for the numHamburgers field.

8. Add a private boolean field to the class called `orderToGo`, and initialize it to false. Getter methods for fields with a boolean data type don't normally follow the getter methods naming convention. They are commonly named with a pattern that starts with the word "is" followed by the field name. For example, for our `orderToGo` field the getter method would be `isOrderToGo()`, which makes the method more readable, and obvious that it is going to return a boolean (equivalent to the answer to a yes/no question).

9. Add the `isOrderToGo()` and `setOrderToGo(…)` methods for the `orderToGo` field.

10. Add one more field to this class: a private integer called `orderNum`. Initialize it to a positive number of your choosing and add the getter and setter methods for it as well.

11. Make sure to organize the components of your class so they are in the standard order: all the fields at the top and the getters and setters below them.

12. Add a constructor to this class, below the fields, but above the first getter/setter method. This constructor should take six parameters, one for the number of Hamburgers, Cheeseburgers, Veggieburgers and sodas to put in the food order, and one boolean parameter that specifies if the order is for takeout, and one for the order number. Inside the constructor, set each field to the appropriate parameter.

13. Time to test your new class. Switch over to the main class and add the following code to the main method to test your new `Order` class:

```
public static void main(String[] args) {

    Order order1 = new Order(3, 5, 4, 10, true, 1);
    Order order2 = new Order(0, 0, 3, 3, false, 2);
    Order order3 = new Order(1, 1, 0, 2, false, 3);

    System.out.println(
            "Order{" + "numHamburgers=" + order1.getNumHamburger() + ", numCheeseburgers="
            + order1.getNumCheeseburgers() + ", numVeggieburgers=" + order1.getNumVeggieburgers()
            + ", numSodas=" + order1.getNumSodas() + ", orderToGo=" + order1.isOrderToGo() + ", orderNum=" + order1.getOrderNum() + '}');
    System.out.println(
            "Order{" + "numHamburgers=" + order2.getNumHamburger() + ", numCheeseburgers="
            + order2.getNumCheeseburgers() + ", numVeggieburgers=" + order2.getNumVeggieburgers()
            + ", numSodas=" + order2.getNumSodas() + ", orderToGo=" + order2.isOrderToGo() + ", orderNum=" + order2.getOrderNum() + '}');
    System.out.println(
            "Order{" + "numHamburgers=" + order3.getNumHamburger() + ", numCheeseburgers="
            + order3.getNumCheeseburgers() + ", numVeggieburgers=" + order3.getNumVeggieburgers()
            + ", numSodas=" + order3.getNumSodas() + ", orderToGo=" + order3.isOrderToGo() + ", orderNum=" + order3.getOrderNum() + '}');

}
```

14. Now, run your program to see the information about each new `Order` get printed out:

```
run:
Order{numHamburgers=3, numCheeseburgers=5, numVeggieburgers=4, numSodas=10, orderToGo=true, orderNum=1}
Order{numHamburgers=0, numCheeseburgers=0, numVeggieburgers=3, numSodas=3, orderToGo=false, orderNum=2}
Order{numHamburgers=1, numCheeseburgers=1, numVeggieburgers=0, numSodas=2, orderToGo=false, orderNum=3}
BUILD SUCCESSFUL (total time: 0 seconds)
```

15. Finally, edit this test code and type “order1” followed by a period to see how NetBeans automatically shows you all the methods that are available to you for `Order` objects:

16. You’ll see all the getters and setters that NetBeans created listed here, along withsome other methods like the `toString()`, `equals(…)` and `getClass()` which are inherited from `java.lang.Object`. So, you’ve pretty quickly been able to build a whole class. From this menu choose the `setNumSodas()` method, and change the number of sodas in order1 to be **12**. Test your code and make sure that the information being printed out about order1 reflects the change. Feel free to play around with some of these other methods to test them out as well.

17. 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: Create an ArrayList of Order objects and methods to manage it

1. Add another class to your project called `FastFoodKitchen` (see steps 1 and 2 in Part A for guidance if you don’t remember how). Note that if your main class was called FastFoodKitchen you will need to use a different name for this step (e.g., FastFoodKitchenSimulation) or you can change the name of the main class

for this project (e.g., Main). Consult with the instructional team if you are not sure about your project file structure.

2. At the top of your `FastFoodKitchen` class, add an `ArrayList` of `Order` objects called `orderList`. Make `orderList` private.
3. Add a second field that is an integer called `nextOrderNum`, making it both private and static with an initial value of 1.
    - Can you guess why we would want this field to be static?

4. Add a getter method for the `nextOrderNum` field. We do not need a setter method for this field.

5. Add a new private static method that is called `incrementNextOrderNum()` that does exactly that – adds one to the `nextOrderNum` field. This method doesn't need to return anything (`void`). This method must be called each time after creating a new order, to make sure that no order has the same order id number.

6. Create a constructor for `FastFoodKitchen` that populates `orderList` with an initial set of three orders using the same numbers of items as the three orders in the test code for Part A.

    - There are several ways to accomplish this step. You can either have a no-parameter constructor where you create three `Order` instances and add them to `Order` or you can create a constructor that takes in three `Order` objects as parameters that get added to `orderList` inside that constructor.
    - Regardless of how you complete this step you will need to reference the `nextOrderNum` field either directly (if you're creating the orders in the constructor) or using the `getNextOrderNum()` method if you create the objects and pass them in as arguments. Remember that this is a static method so you can call it using the class name - `FastFoodKitchen.getNextOrderNum()`. You need to do this to specify the order id number when you need to create a `Order` object and you must call `incrementNextOrderNum()` afterwards to update the field value.
7. Now we are ready to add methods to handle placing and managing orders in a fast food kitchen. Add the following methods to `FastFoodKitchen`. The bullet points under each tell you what the method must do.

`public int addOrder(int ham, int cheese, int veggie, int soda, boolean toGo)`

- Using the values of the parameters, create a new `Order`, using the next order number available
- Add this new `Order` to orderList
- Increment `nextOrderNum`
- Return the order number of the order you just created

`public boolean cancelLastOrder()`

- If there are orders in the list
  - Remove the last order
  - Decrement `nextOrderNum`
  - Return true to indicate that an order has been canceled
- If there are no orders in the list return false to indicate the list is empty.

`public int getNumOrdersPending()`

- Return the number of orders currently in the `orderList`

8. Add Javadoc for each of the methods in the class.

9. Now, switch over to your main class and comment out the code from Part A. Copy and paste the code from this code snippet - https://gist.github.com/nanajjar/c9ee149e05886d7694bafb397798b8e5 into the `main()` method. You can delete or comment out the code we have from Part A. Read through the code to get a sense for what it is doing. Note that you will need to add the import statement for the `Scanner` class as we are using it in the code snippet you just copied.
   Pay extra attention where you add this code keeping in mind that it needs to all be inside the main method.

10. Run your program and test the different menu options to make sure that your methods are working. You should be able to add a new order, cancel the last order, get the number of pending orders and orders. Fix any bugs you find.

11. Generate the Javadoc to make sure that your `FastFoodKitchen` class is well documented.
12. When your program works and you are satisfied with the result and JavaDoc, show your work to the instructional team to be checked off and proceed to Part C.

Part C: Use ArrayList methods to manipulate the contents of an array list by referencing specific elements

1. Add the following methods to FastFoodKitchen.
   `public boolean isOrderDone(int orderID)`
   - Iterate through orderList, checking each order.
   - If there is an order in the list whose order number matches `orderID`, the order is still in progress, so return false
   - If there isn't an order in the list that matches orderID, return true

   `public boolean cancelOrder(int orderID)`
   - Iterate through orderList, checking each order.
   - If there is an order in the list whose order number matches orderID, remove the order from the `orderList` and return true
   - If there isn't an order in the list that matches `orderID`, return false

2. Now we need to update the main class by adding menu options for the methods we added. Make the following changes to the `main()` method:
   - Add a new menu option for checking on an order
   - Add a new case to the switch statement for this new option
     - Ask user to enter the order number for the order they want to check on
     - Call the `isOrderDone` method with user entered order number
     - If the method returns true, display "No order was found"
     - If the method return false, display "Your order in being prepared"
   - Add a new menu option for canceling an order
   - Add a new case to the switch statement for this new option
     - Ask user to enter the order number for the order they want to cancel
     - Call the `cancelOrder` method with user entered order number
     - If the method returns true, display "Your order has been successfully cancelled"
     - Otherwise, display "Sorry, we can't find your order number in the system"

3. Run your project and test to make sure that your new methods are working. You should be able to check on or cancel an existing order. Make sure that the other methods still work as expected. Fix any bugs you find.

4. When everything works take some time to appreciate the program you put together. Can you see similar logic used in any restaurant ordering systems that you've encountered? Can you think of other "things" an ordering system can do? Do you think we can extend our FastFoodKitchen to do those "things"?

5. When 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 create an ArrayList of objects
- How to call methods on objects stored in an Array List
- How to add a class to a NetBeans project
- How to override a method in a parent class
- How NetBeans shows you methods relevant to the current object
- Practiced generating Javadocs
- How to write code to specifications (the methods you added to the FastFoodKitchen class)
- Practiced testing and debugging a program