LAB 7

Learning Outcomes

- Implement comparison methods for ordered lists of custom object types
- Translate a complex sequence of ordering rules into Java code
- Create a persistent user input structure using a loop
- Determine alternate approaches to programming specific methods

Pre-Lab

- Create a new Java project called Lab7
- Download the files: ArrayOrderedList.java, ArrayList.java, ArrayIterator.java, OrderedListADT.java, ElementNotFoundException.java, EmptyCollectionException.java, ListADT.java, Card.java, TestCards.java, Person.java, and ContactList.java
- Save these downloaded files into the Lab7 src folder

Exercise 1 – Ordering a deck of cards

1. Open Card.java and TestCards.java in Eclipse and examine the code in both classes.
2. Try running the TestCards class. An exception will be thrown. Read the exception message in the console and click on the line it points to, from the `add()` method in ArrayOrderedList.java, to help you understand why this occurred.
3. Go back into Card.java and modify the class header so that the class `implements Comparable<Card>`
4. The class name will now show an error due to it missing the method that are required from the Comparable interface: `compareTo()`
5. Hover over the class name in the error window and click "Add unimplemented methods" to auto-generate the `compareTo()` stub or just type in the method signature yourself.
6. Before completing this method, there are a few other helper methods to implement.
7. Create an `equals()` method. Make it return true if both the rank and suit are the same between `this` and `other`, and false otherwise.
8. Create two helper methods called `getSuitValue()` and `getRankValue()` respectively. Both methods will have int return types. These methods will be used to order the cards given the specified rules and hints (note that this is a specific scenario and these orderings will not apply to many, if any, actual games):
    - suits are ordered as Diamonds (lowest), Clubs, Hearts, Spades (highest) [note that only the initial letters are being used to represent the suits]

- ranks are ordered as 2-10 in based on their numeric value, Jacks, Queens, and Kings are all considered equal, higher than the numeric cards, and Aces are considered the highest rank
- where applicable, you can choose which values to use for the ranking system so long as they perform the correct ordering.
- try to code these methods on your own, but if you are stuck then read the hint provided in Lab7Hints.txt (note that this file also contains the solutions to the ordered deck)

9. Go back to the `compareTo()` method to fill it in based on the following specifications:
    - Call the `equals()` method to see if the two cards are the same (return 0 if so).
    - Use the `getRankValue()` method on both cards and compare them. If they have different rank values, use this comparison to determine their order. If they have the same rank value, proceed to the next option.
    - Use `getSuitValue()` to compare two cards of the same rank value.
10. Run TestCards.java to check if your ordering system was implemented correctly. The first example, at the top, should produce the following results:
    ```
    7 of D
    7 of D
    7 of H
    10 of C
    K of S
    A of H
    ```
11. The results of the second example are found in Lab7Hints.txt.

Exercise 2 – Ordering a contact list

1. Open Person.java and ContactList.java in Eclipse and examine the code in both classes.
2. The Person class already implements Comparable<Person> and contains an empty `compareTo()` method. This program, once it's complete, will provide 3 options for ordering the contacts: sorting by name, email, or city. We'll revisit compareTo() shortly.
3. First, create 3 private helper methods called `sortByName(Person other)`, `sortByEmail(Person other)`, and `sortByCity(Person other)` and all should have int return types. Use the String `compareTo()` within these methods.
4. Now fill in the Person class `compareTo()` with calls to these private helper methods within a conditional. Use `ContactList.sortBy` to determine how to order the list. What kind of values are we expecting in this sortBy variable? Examine the code in ContactList if you are unsure. You may need to also want to read the steps below and then come back to this step once you understand how this variable works.
5. Go into ContactList.java. There are two methods that have been left incomplete.

6. The `main()` method needs the user input portion added. Use the Scanner class and `char c = input.next().charAt(0);`
   to read in the character typed by the user in the console.
7. Add a loop to continuously prompt the user to enter a character to sort, until they enter a character other than 'n', 'e', or 'c'. Display the message stored in msg every time the user is asked to enter a character.
8. When they do enter one of the 3 valid character options, create a new ContactList object with the corresponding parameters (the contacts array and the entered character).
9. The last part to fill in is the `printContacts()` method. There are multiple ways of printing out the contact list table. How many different approaches can you think of? Which one requires the least amount of code to be added?
10. Run ContactList.java and enter each of the options 'n', 'e', and 'c' and check that the list is printed correctly and ordered on the corresponding parameter (name, email, or city).

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- Card.java
- Person.java
- ContactList.java