Learning Outcomes

- Develop a generic class so that it works with any type
- Examine the structure of singly- and doubly- linked lists and their respective nodes
- Modify existing loops to work more robustly and efficiently with linked lists
- Identify the various cases in removing a node from a list
- Program a removal method that handles each of the identified cases

Pre-Lab

- Create a new Java project called Lab4
- Download the files: Person.java, TestReverse.java, LinearNode.java, BuildLinkedList.java, DoubleLinkedNode.java, and BuildDLL.java
- Save these downloaded files into the Lab4 src folder

Exercise 1 – Creating a class with generics

1. Create a new class called ReversibleArray.
2. In the class header, add the code `<T>` immediately to the right of the class name.
3. Give this class two private instance variables: `T[] array` and `int count`
4. Create a constructor which takes in one parameter of type `T[]` and assign that parameter's value into `this.array`. Set count as the length of the array.
5. Add a toString() method that outputs the array values in the format: elem0, elem1, elem2, etc. (hint: add the comma and space only if you are not on the final iteration of the loop).
6. Create the most important function in this class: `public void reverse () { }`
    a. This method must swap elements within the array, without making a new array or other collection.
    b. The swaps should be made symmetrically about the middle of the array. This means array[0] swaps with array[n-1], array[1] swaps with array[n-2], etc.
    c. Use a temp variable to help with these swaps. What variable type should this temp variable be?
    d. How many swaps must be made to correctly reverse the array?
7. Open TestReverse.java and read over the code in its `main()` function.

8. Run TestReverse to see the output. If your output is incorrect, then debug the program until it works properly. It should print out the following text:
   atom, breeze, clock, daydream, energy
   energy, daydream, clock, breeze, atom
   11, 22, 33, 44, 55, 66, 77
   77, 66, 55, 44, 33, 22, 11
   Jerry Seinfeld, Salma Hayek, Reese Witherspoon, Vince Vaughn
   Vince Vaughn, Reese Witherspoon, Salma Hayek, Jerry Seinfeld

Exercise 2 – Singly Linked Lists

1. Open LinearNode.java and BuildLinkedList.java in Eclipse and examine the code in both classes.
2. The `main()` method in BuildLinkedList created a list with 10 nodes, each storing an integer value from 1 to 10. The `printList()` method prints out this list. Run the program to see the list elements printed out.
3. The problem is that `printList()` is "hard-coded" to loop through 10 elements in the for-loop so this method will not work for any other size of list. Modify this method so that it works for any list length (i.e. not hard-coded). You **cannot** change the method signature (parameters). You also **cannot** add instance variables to the class.
4. In the `main()` method, change the for-loop to create a list of 20 nodes and run the program to ensure that it correctly creates and prints out those 20 nodes.
5. Change it again, this time to 7 and check that it works properly with the 7 nodes.

Exercise 3 – Doubly Linked Lists

1. Open DoubleLinkedNode.java and BuildDLL.java in Eclipse and examine the code in both classes.
2. The DoubleLinkedNode class is complete and does not have to be modified.
3. The BuildDLL class has some provided methods for creating a doubly linked list with a sequence of letters (note that we are using the Character class for this, which is a wrapper for the primitive char type) and printing out the list from front to rear. Run the program to see the default output.
4. Notice that the original list is printing correctly, but the `remove()` method is not provided so subsequent list outputs are incorrect.
5. You must fill in the code for removing an element from the linked list using the following rules and hints:

- Loop through the list until you find the correct node (how can you tell if the node is the one for which we are searching?)
- We will assume that the input parameter 'elem' will always be a valid element that is contained in the list. Normally we would have to account for elements that are not found in the list, but you may ignore this possibility for this lab.
- What are the 3 possible cases of the node's location in the list? How must each of these cases be handled?
- Make the appropriate connections with the previous and/or next node based on its position in the list. (Hint: remember front's previous is null and rear's next is null).

6. Run the program and check that the output is correct. The list after each step should be:

K T E N P A L	(original)
K T E P A L	(after removing 'N')
T E P A L	(after removing 'K')
T E P A	(after removing 'L')

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

- ReversibleArray.java
- BuildLinkedList.java
- BuildDLL.java