Ref: CSC20 PAL WS ListNode Class

Linked Lists

In chapter 10, we discussed how ArrayLists overcome some limitations of the standard array and offer features of dynamic sizing and operations for inserting and deleting elements. However, calling operations like insert/remove over a large number of elements (e.g. a million) can be very slow. This would happen due to the underthehood implementation of an ArrayList as an array which requires elements to be shifted in response to each insert and remove operation.

Here, we introduce the LinkedList Collection. Both ArrayList and LinkedList implement the List Interface, but a LinkedList object consists of separate nodes which are linked to each other by reference. This means when elements are added or removed, the LinkedList simply needs to detach and reattach pointer references, which is a faster operation.

Let's practice adding and getting elements from a LinkedList. Each new added element is quickly attached to the previous one by reference.

```
LinkedList<Integer> myList = new LinkedList<>();
myList.add(2);
myList.add(4);
myList.add(6);
myList.add(8);

Note: We can use this declaration too in the above code:
List<Integer> myList = new LinkedList<>();//How does this help?
```

Now let's get elements at certain indices. The get() method will iterate over the list until it lands at the specified position.

```
Integer first = myList.get(0); // retrieve 2
Integer second = myList.get(1); // retrieve 4
Integer third = myList.get(2); // retrieve 6
```

Ex1) Write code to manually get and print elements from all even indices of the given LinkedList. Make sure to use get() to retrieve the required elements.

```
// myList = 11 -> 22 -> 33 -> 44 -> 55 -> 66 -> 77 -> 88
```

```
// get and print elements [11, 33, 55, 77]
```

Removing elements means simply reassigning an element's pointer reference. For example, if we do:

```
myList.remove(2); // remove third index, i.e. 6
// list is now 2->4->8
```

We could also remove by using remove(Object element):

```
Integer four = 4;
myList.remove(four); // remove element with Object Integer 4
// list is now: 2->8
```

Ex2) Write a small method which takes both a list of integers and a LinkedList of Integer elements. Iterate over the list of integers and remove each one from the LinkedList using remove(Object element). You can assume all integers are actually present in the LinkedList.

Another helpful method is contains(Object element), which returns a boolean indicating if the specified element is present in the list. For example:

```
myList.contains(8); // returns true
```

Ex3) Repeat the previous question, but this time use contains(Object) to check if each integer exists in the Integer LinkedList before removing it.

To iterate over a LinkedList, we can use an **Iterator,** which is provided by Java for list Collections and allows us extremely quick traversal over our LinkedList.

We can have the iterator 'point' to different list elements, moving it forward using *iterator.next()* and removing the current element using *iterator.remove().* When the iterator is first created, you can think of it as positioned right before the first list element.

Here is an example.

```
// myList = 3 -> 2 -> 7 -> 4 -> 9
```

```
Iterator<Integer> iter = myList.iterator(); // iter positioned
before first element

iter.next();   // move on to first element, i.e. 3
iter.remove(); // remove 3
iter.next();   // move on to second element, i.e. 2
```

Ex4) Write code to create an Interator for the given LinkedList and remove its first three elements. Use next() to move forward and remove() to remove elements.

```
// myList = 11 -> 22 -> 33 -> 44 -> 55 -> 66 -> 77 -> 88
```

Here is how to traverse a LinkedList of Integer objects and remove the odd integers. Note the use of *hasNext()* to ensure the element positioned right after the iterator actually exists.

```
Iterator<Integer> iter = myList.iterator(); //create iterator

while(iterator.hasNext()){

     //get first element via iter.next()
     Integer element = iter.next();

     //check if the element iterator is on, is an odd integer
     //if so, remove the element iterator is on
     if(element % 2 == 0){
          iter.remove();
     }
}
```

Ex5) Write a small method which takes in a LinkedList of Integer elements, as well as an Integer variable. Use an Iterator to search the LinkedList for the Integer variable and remove it if found. Remove all instances of it if multiple instances exist in the LinkedList. This method returns void

Ex6) Repeat Q5, but this time takes in a LinkedList of Integer elements and an array of Integers. Use an Iterator to search the LinkedList for all items in the array and remove them once found. Assume they can appear only once in the LinkedList. This method returns void.

Repeat these exercises to insert elements at the front, middle and end of the LinkedList.