



Lab 1 Linked List

Quang D. C.
dcquang@it.tdt.edu.vn

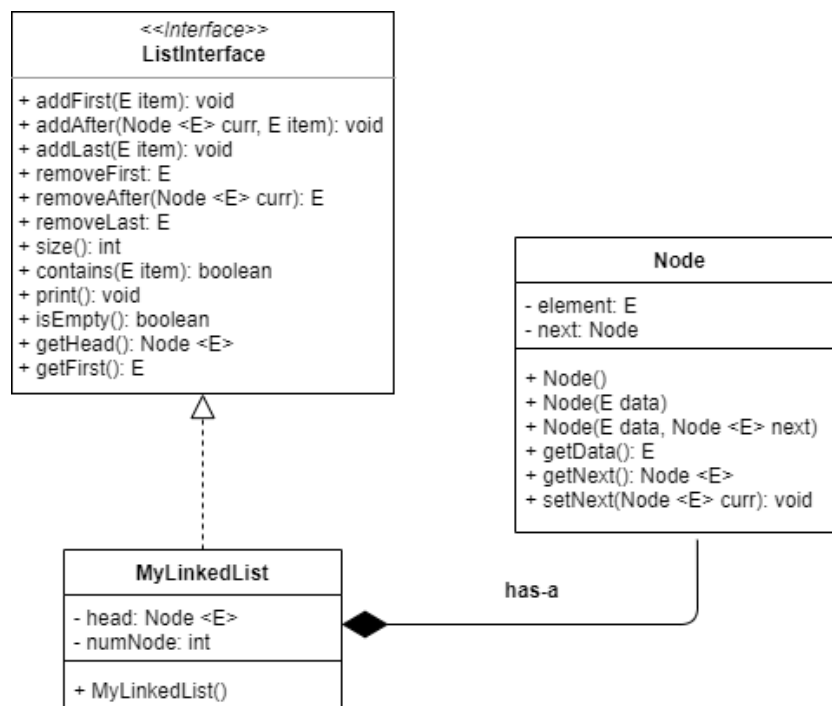
August 24, 2020

After completed this tutorial, you can implement a list ADT with linked list. Please review Generic before starting this tutorial.

1. UML model of Linked list

The following figure presents an UML model of linked list:

- ListInterface represents public functions of linked list, e.g., add new item, remove an item.
- Node class represents an item (node) in linked list.
- MyLinkedList class implements ListInterface and includes items have Node types.



In the next section, we will approach how to implement a linked list based on the above UML model.

2. *Node* class

Node is the basic item in list, thus we need to implement it first.

```
1 public class Node <E> {
2     private E data;
3     private Node <E> next;
4     public Node(){
5         data = null;
6         next = null;
7     }
8     public Node(E data){
9         this(data, null);
10    }
11    public Node(E data, Node <E> next){
12        this.data = data;
13        this.next = next;
14    }
15    public Node <E> getNext(){
16        return next;
17    }
18    public E getData(){
19        return data;
20    }
21    public void setNext(Node <E> n){
22        next = n;
23    }
24 }
```

3. *ListInterface* interface

ListInterface defines the operations (methods) we would like to have in a List ADT.

```
1 import java.util.NoSuchElementException;
2 public interface ListInterface <E> {
3     public void addFirst(E item);
4     public void addAfter(Node <E> curr, E item);
5     public void addLast(E item);
6
7     public E removeFirst() throws NoSuchElementException;
8     public E removeAfter(Node <E> curr) throws
9     NoSuchElementException;
10    public E removeLast() throws NoSuchElementException;
11
12    public void print();
13    public boolean isEmpty();
14    public E getFirst() throws NoSuchElementException;
15    public Node <E> getHead();
16    public int size();
17    public boolean contains(E item);
18 }
```

4. *MyLinkedList* class

This *MyLinkedList* class will implement the *ListInterface* interface.

```
1 import java.util.NoSuchElementException;
2 public class MyLinkedList <E> implements ListInterface<E> {
3     private Node <E> head;
4     private int numNode;
5     public MyLinkedList(){
6         head = null;
7         numNode = 0;
8     }
9     @Override
10    public void addFirst(E item){
11        head = new Node<E>(item, head);
12        numNode++;
13    }
14    @Override
15    public void addAfter(Node<E> curr, E item){
16        if(curr == null){
17            addFirst(item);
18        }
19        else{
20            Node<E> newNode = new Node<E>(item, curr.getNext());
21            curr.setNext(newNode);
22        }
23        numNode++;
24    }
25    @Override
26    public void addLast(E item){
27        if(head == null){
28            addFirst(item);
29        }
30        else{
31            Node<E> tmp = head;
32            while(tmp.getNext() != null){
33                tmp = tmp.getNext();
34            }
35            Node<E> newNode = new Node<>(item, null);
36            tmp.setNext(newNode);
37            numNode++;
38        }
39    }
40    @Override
41    public E removeFirst() throws NoSuchElementException{
42        if(head == null){
43            throw new NoSuchElementException("Can't remove element
44            from an empty list");
45        }
46        else{
47            Node<E> tmp = head;
48            head = head.getNext();
49            numNode--;
50            return tmp.getData();
51        }
52    }
53 }
```

```
52     @Override
53     public E removeAfter(Node<E> curr) throws
NoSuchElementException{
54         if(curr == null){
55             throw new NoSuchElementException("Can't remove element
from an empty list");
56         }
57         else
58         {
59             Node<E> delNode = curr.getNext();
60             if(delNode != null) {
61                 curr.setNext(delNode.getNext());
62                 numNode--;
63                 return delNode.getData();
64             }
65             else{
66                 throw new NoSuchElementException("No next node to
remove");
67             }
68         }
69     }
70 }
71 @Override
72 public E removeLast() throws NoSuchElementException
73 {
74     if(head == null){
75         throw new NoSuchElementException("Can't remove element
from an empty list");
76     }
77     else{
78         Node<E> preNode = null;
79         Node<E> delNode = head;
80         while(delNode.getNext() != null){
81             preNode = delNode;
82             delNode = delNode.getNext();
83         }
84         preNode.setNext(delNode.getNext());
85         delNode.setNext(null);
86         numNode--;
87         return delNode.getData();
88     }
89 }
90 @Override
91 public void print(){
92     if(head != null){
93         Node<E> tmp = head;
94         System.out.print("List: " + tmp.getData());
95         tmp = tmp.getNext();
96         while(tmp != null)
97         {
98             System.out.print(" -> " + tmp.getData());
99             tmp = tmp.getNext();
100         }
101         System.out.println();
102     }
103     else{
104         System.out.println("List is empty!");
```

```
105     }
106 }
107 @Override
108 public boolean isEmpty(){
109     if(numNode == 0) return true;
110     return false;
111 }
112 @Override
113 public E getFirst() throws NoSuchElementException{
114     if(head == null){
115         throw new NoSuchElementException("Can't get element
116 from an empty list");
117     }
118     else{
119         return head.getData();
120     }
121 }
122 @Override
123 public Node<E> getHead(){
124     return head;
125 }
126 @Override
127 public int size(){
128     return numNode;
129 }
130 @Override
131 public boolean contains(E item){
132     Node<E> tmp = head;
133     while(tmp != null){
134         if(tmp.getData().equals(item))
135             return true;
136         tmp = tmp.getNext();
137     }
138     return false;
139 }
```

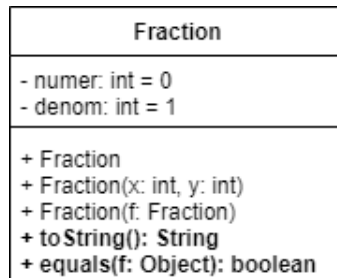
5. Test Integer Linked List

```
1 public class Test {
2     public static void main(String[] args)
3     {
4         MyLinkedList<Integer> list = new MyLinkedList<Integer>();
5         list.addFirst(new Integer(2));
6         list.addLast(new Integer(3));
7         list.print();
8     }
9 }
```

6. Exercise

Exercise 1

Giving **Fraction** class as the following class diagram:



You need to implement a linked list to contain Fraction items.

Exercise 2

Suppose that we have an abstract method with signature as follow:

public E removeCurr(Node<E> curr)

This method removes the node at position *curr*. You need to add this abstract method to your program and implement it.

Exercise 3

Suppose we are having a list of integer numbers, do the following requirements:

- (a) Count the number of even item in the list.
- (b) Count the number of prime item in the list.
- (c) Add item X before the first even element in the list.
- (d) Find the maximum number in the list.
- (e) (*) Reverse the list without using temporary list.
- (f) (*) Sort the list in ascending order.