Lab 1
Linked List

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

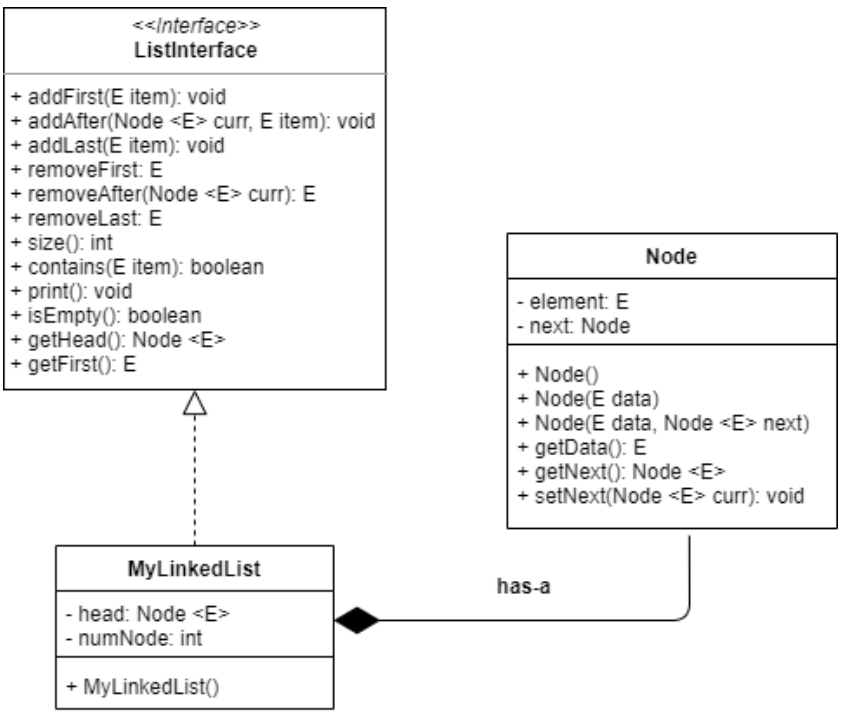
August 24, 2020

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.

```
public class Node <E> {
    private E data;
    private Node <E> next;
    public Node(){
        data = null;
        next = null;
    }
    public Node(E data){
        this(data, null);
    }
    public Node(E data, Node <E> next){
        this.data = data;
        this.next = next;
    }
    public Node <E> getNext(){
        return next;
    }
    public E getData(){
        return data;
    }
    public void setNext(Node <E> n){
        next = n;
    }
}
```

3. *ListInterface* interface

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

```
import java.util.NoSuchElementException;
public interface ListInterface <E> {
    public void addFirst(E item);
    public void addAfter(Node <E> curr, E item);
    public void addLast(E item);

    public E removeFirst() throws NoSuchElementException;
    public E removeAfter(Node <E> curr) throws
   NoSuchElementException;
    public E removeLast() throws NoSuchElementException;

    public void print();
    public boolean isEmpty();
    public E getFirst() throws NoSuchElementException;
    public Node <E> getHead();
    public int size();
    public boolean contains(E item);
}
```

4. *MyLinkedList* class

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

```
import java.util.NoSuchElementException;
public class MyLinkedList <E> implements ListInterface<E> {
    private Node <E> head;
    private int numNode;
    public MyLinkedList(){
        head = null;
        numNode = 0;
    }
    @Override
    public void addFirst(E item){
        head = new Node<E>(item, head);
        numNode++;
    }
    @Override
    public void addAfter(Node<E> curr, E item){
        if(curr == null){
            addFirst(item);
        }
        else{
            Node<E> newNode = new Node<E>(item, curr.getNext());
            curr.setNext(newNode);
        }
        numNode++;
    }
    @Override
    public void addLast(E item){
        if(head == null){
            addFirst(item);
        }
        else{
            Node<E> tmp = head;
            while(tmp.getNext() != null){
                tmp = tmp.getNext();
            }
            Node<E> newNode = new Node<>(item, null);
            tmp.setNext(newNode);
            numNode++;
        }
    }
    @Override
    public E removeFirst() throws NoSuchElementException{
        if(head == null){
            throw new NoSuchElementException("Can't remove element
     from an empty list");
        }
        else{
            Node<E> tmp = head;
            head = head.getNext();
            numNode--;
            return tmp.getData();
        }
    }
```

```
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
67                  remove");
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
    from an empty list");
116         }
117         else{
118             return head.getData();
119         }
120     }
121     @Override
122     public Node<E> getHead(){
123         return head;
124     }
125     @Override
126     public int size(){
127         return numNode;
128     }
129     @Override
130     public boolean contains(E item){
131         Node<E> tmp = head;
132         while(tmp != null){
133             if(tmp.getData().equals(item))
134                 return true;
135             tmp = tmp.getNext();
136         }
137         return false;
138     }
139 }
```

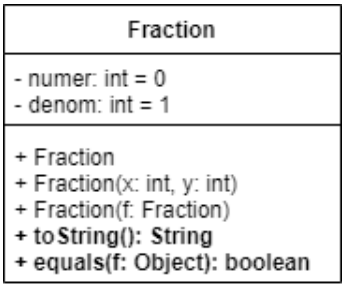
5. Test Integer Linked List

```
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. Excercise

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.