# Data Structures and Algorithms

# Lab 6: Linked List

## I. Objective

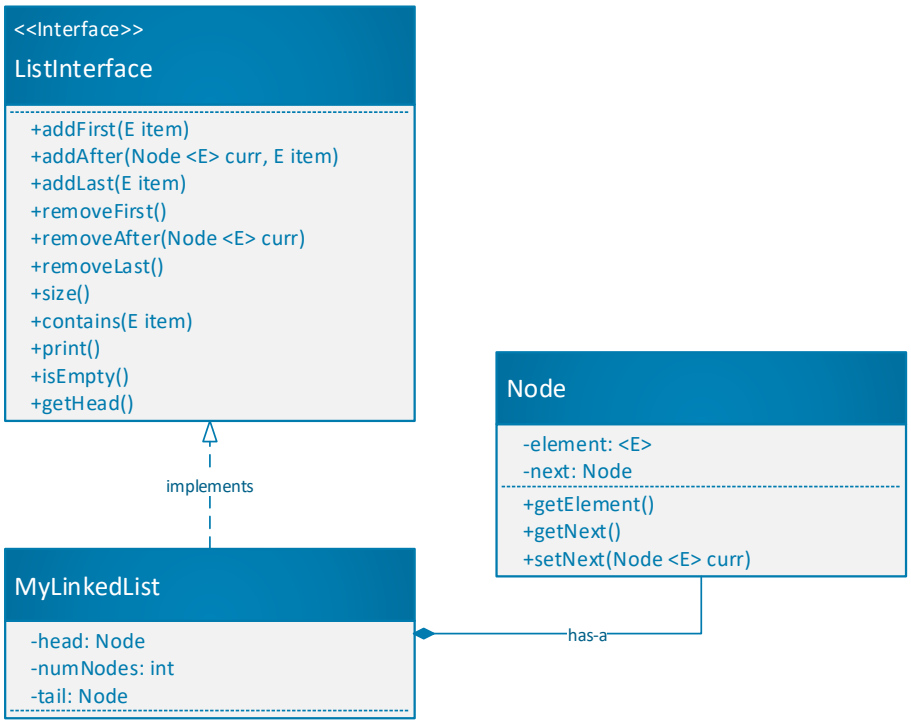
After completing this tutorial, you can:

- Implement a list ADT with linked list.

## II. 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 present how to implement a linked list of **`Fraction`** based on the above UML model.

## III. Fraction class

Before proceeding to the implementation of linked list, first, you need to implement **`Fraction`** class as follows:

| **`Fraction`** |
|---|
| `- numer : int = 0`<br>`- denom : int = 1` |
| `+ Fraction()` |

```
+ Fraction(x : int, y : int)
+ Fraction(another : Fraction)
+ toString() : String
+ equals(f : Object) : boolean
```

## IV. *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;
	}
}
```

## V. *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);

}
```

## VI. *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();
        }
    }

    @Override
    public E removeAfter(Node<E> curr) throws NoSuchElementException
    {
        if(curr == null)
        {
                throw new NoSuchElementException("Can't remove element
                from an empty list");
        }
        else
        {
                Node<E> delNode = curr.getNext();
                if(delNode != null)
                {
                        curr.setNext(delNode.getNext());
                        numNode--;
                        return delNode.getData();
                }
                else
                {
                        throw new NoSuchElementException("No next node to
                        remove");
                }
        }
    }

    @Override
    public E removeLast() throws NoSuchElementException
    {
        if(head == null)
        {
                throw new NoSuchElementException("Can't remove element
                from an empty list");
        }
```

```
        else
        {
            Node<E> preNode = null;
            Node<E> delNode = head;
            while(delNode.getNext() != null)
            {
                preNode = delNode;
                delNode = delNode.getNext();
            }

            preNode.setNext(delNode.getNext());
            delNode.setNext(null);
            numNode--;

            return delNode.getData();
        }
    }

    @Override
    public void print()
    {
        if(head != null)
        {
            Node<E> tmp = head;
            System.out.print("List: " + tmp.getData());
            tmp = tmp.getNext();
            while(tmp != null)
            {
                System.out.print(" -> " + tmp.getData());
                tmp = tmp.getNext();
            }
            System.out.println();
        }
        else
        {
            System.out.println("List is empty!");
        }
    }

    @Override
    public boolean isEmpty()
    {
        if(numNode == 0) return true;
        return false;
    }

    @Override
    public E getFirst() throws NoSuchElementException
    {
        if(head == null)
        {
            throw new NoSuchElementException("Can't get element from
            an empty list");
        }
        else
        {
            return head.getData();
        }
    }

    @Override
```

```
	public Node<E> getHead()
	{
		return head;
	}

	@Override
	public int size()
	{
		return numNode;
	}

	@Override
	public boolean contains(E item)
	{
		Node<E> tmp = head;
		while(tmp != null)
		{
			if(tmp.getData().equals(item))
				return true;
			tmp = tmp.getNext();
		}
		return false;
	}

}
```

## VII. Test class

```
public class Test {

	public static void main(String[] args)
	{
		MyLinkedList<Fraction> list = new MyLinkedList<>();
		list.addFirst(new Fraction(1, 2));
		list.addLast(new Fraction(3, 4));
		list.print();
	}

}
```

## VIII. Exercises

**1.** 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.

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

a. Count the number of even item and odd 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. (*) Reverse the list without using temporary list.

e. Find the minimum number and maximum number in the list.

f. (*) Sort the list in ascending and descending order.