CS162

ASSIGNMENT 5

NAME:

ARCHIT AGRAWAL

ROLL NO. :

202052307

SECTION:

A

Question

1. **Implement a generic Linear linked List**
    a. Create a class named “LInkedList” that uses another class named “Node”.
    b. The class should have (atleast) all the functions that are discussed in the Lecture. For reference, see the lecture video. The lecture slide is attached herewith.

2. **Create an Application class to use the LinkedList class**
    a. The Application class should have a main method.
    b. The Application class should create an object of the LInkedList class.
    c. The Application class should call all the functions of the LinkedList class.

CODE

```
class Node<T>{
    T data;
    Node next;

    public Node(){}

    public Node(T element){
        this.data = element;
    }

    public Node(T element, Node addr){
        this.data = element;
        this.next = addr;
    }
}

class LinkedList<T>{

    protected Node<T> head;
    protected int size;

    public LinkedList(){
        head = null;
        size = 0;
    }

    //The method push adds an element at the starting (index = 0) of the linke
d list.
    public void push(T data){
        Node<T> newNode = new Node<T>(data);
        newNode.next = head;
        head = newNode;
        size++;
    }

    //The method append adds an element at the end (index = size) of the linke
d list.
    public void append(T data){
        Node<T> newNode = new Node<T>(data);
        if (head == null) {
            head = newNode;
        } else {
            Node <T> lastNode = head;
```

```
            while (lastNode.next != null) {
                lastNode = lastNode.next;
            }
            lastNode.next = newNode;
        }
        size++;
    }

    //The method add adds a given element at the given index.
    public void add(int index, T data){

        if(index < 0 || index > size){
            throw new IndexOutOfBoundsException("Index = "+index +" Size = "+s
ize);
        }

        if(index == 0){
            Node<T> newNode = new Node<T>(data);
            newNode.next = head;
            head = newNode;
        } else {
            Node<T> newNode = new Node<T>(data);
            Node<T> prevNode = head;
            int i = 0;
            while(i < index - 1){
                prevNode = prevNode.next;
                i++;
            }
            newNode.next = prevNode.next;
            prevNode.next = newNode;
        }
        size++;
    }

    //The method removeKey removes the first occurence of the element passed t
o it from the linked list.
    public void removeKey(T key){
        if(size == 0){
            return;
        }
        if(head.data == key){
            head = head.next;

        } else {
            Node<T> currNode = head.next;
            Node<T> prevNode = head;

            while(currNode != null){
```

```
                //to prevent NullPointerException
                if(currNode.data.equals(key)){
                    prevNode.next = currNode.next;
                    break;
                } else {
                    prevNode = currNode;
                    currNode = currNode.next;
                }
            }
        }
        size--;
    }

    //The method removeIndex removes the element from the index passed to it a
nd returns the removed element.
    public T removeIndex(int index){
        checkIndex(index);
        T removed;

        if(index == 0){
            removed = head.data;
            head = head.next;
        } else {
            Node<T> currNode = head.next;
            Node<T> prevNode = head;
            int i = 0;
            while(i < index - 1){
                prevNode = prevNode.next;
                currNode = currNode.next;
                i++;
            }
            removed = currNode.data;
            prevNode.next = currNode.next;
        }
        size--;
        return removed;
    }

    //This method prints the linked list in formatted way
    public void printLinkedList() {
        Node<T> currentNode = head;
        System.out.print("head -> ");
        while(currentNode != null){
            System.out.print(currentNode.data + " -> ");
            currentNode = currentNode.next;
        }
        System.out.println("null");
    }
```

```
    public boolean isEmpty(){
        return size == 0;
    }

    public int size(){
        return size;
    }

    public void checkIndex(int index){
        if(index < 0 || index >= size){
            throw new IndexOutOfBoundsException("Index = "+index +" size = "+
size);
        }
    }

    public T get(int index){
        checkIndex(index);
        Node<T> currNode = head;
        int i = 0;
        while(i != index){
            currNode = currNode.next;
            i++;
        }
        return currNode.data;
    }

    public int indexOf(T data){
        Node<T> temp = head;
        int index = 0;
        while(temp != null){
            if(temp.data.equals(data)){
                return index;
            }
            index++;
            temp = temp.next;
        }
        return -1;
    }

}


public class Application{
    public static void main(String []args){
        LinkedList <Integer> list = new LinkedList<Integer>();
        System.out.println("Creating an empty Linked List.");
        System.out.println();
```

```
        System.out.println("Checking if list is empty using isEmpty method");

        if(list.isEmpty()){
            System.out.println("Linked list is empty.");
        } else {
            System.out.println("Linked list is not empty");
        }

        System.out.println();
        System.out.println("Adding 4 elements using method push and append....
");
        list.push(0);
        list.push(2);
        list.append(5);
        list.push(0);
        System.out.println();
        System.out.println("The Linked list");
        list.printLinkedList();
        System.out.println("Size of Linked List = "+ list.size());
        System.out.println();
        System.out.println("Adding 5 more elements using add method...");
        System.out.println();
        list.add(0, 2);
        list.add(5, 2);
        list.add(6, 3);
        list.add(7, 0);
        list.add(8, 7);
        System.out.println("The Linked list");
        list.printLinkedList();
        System.out.println();
        System.out.println("Size of Linked List = "+ list.size());
        System.out.println();

        System.out.println("Removing the first occurence of 2 and 5 using remo
veKey method...");
        list.removeKey(2);
        list.removeKey(5);

        System.out.println();
        System.out.println("The Linked List");
        list.printLinkedList();

        System.out.println();
        System.out.println("Checking if Linked List is empty or not..");
        if(list.isEmpty()){
            System.out.println("Linked list is empty.");
        } else {
            System.out.println("Linked list is not empty");
```

```
        }

        System.out.println("Size of Linked List = "+ list.size);
        System.out.println();

        System.out.println("Using method get to find the element at index 2");
        System.out.println("Element at index 2 is = "+list.get(2));
        System.out.println();

        System.out.println("Using method indexOf to find the index of element
7");
        System.out.println("Index of 7 is = "+list.indexOf(7));
        System.out.println();
        System.out.println("The Linked List");
        list.printLinkedList();
        System.out.println();
        System.out.println("Removing element at index 4 using removeIndex meth
od...");
        System.out.println("Removed Element from index 4 = "+ list.removeIndex
(4));
        System.out.println();

        System.out.println("List after removing the element at index 4");
        list.printLinkedList();
        System.out.println();
        System.out.println("Adding few more elements...");
        System.out.println();

        list.add(0, 2);
        list.add(4, 5);
        list.add(6, 3);
        System.out.println("The Linked List after adding these elements");
        list.printLinkedList();
    }

}
```

OUTPUT

```
Creating an empty Linked List.

Checking if list is empty using isEmpty method
Linked list is empty.

Adding 4 elements using method push and append....

The Linked list
head -> 0 -> 2 -> 0 -> 5 -> null
Size of Linked List = 4

Adding 5 more elements using add method...

The Linked list
head -> 2 -> 0 -> 2 -> 0 -> 5 -> 2 -> 3 -> 0 -> 7 -> null

Size of Linked List = 9

Removing the first occurence of 2 and 5 using removeKey method...

The Linked List
head -> 0 -> 2 -> 0 -> 2 -> 3 -> 0 -> 7 -> null

Checking if Linked List is empty or not..
Linked list is not empty
Size of Linked List = 7

Using method get to find the element at index 2
Element at index 2 is = 0

Using method indexOf to find the index of element 7
Index of 7 is = 6

The Linked List
head -> 0 -> 2 -> 0 -> 2 -> 3 -> 0 -> 7 -> null

Removing element at index 4 using removeIndex method...
```

```
Removed Element from index 4 = 3

List after removing the element at index 4
head -> 0 -> 2 -> 0 -> 2 -> 0 -> 7 -> null

Adding few more elements...

The Linked List after adding these elements
head -> 2 -> 0 -> 2 -> 0 -> 5 -> 2 -> 3 -> 0 -> 7 -> null
PS C:\Users\Archit\Desktop\cprog>
```

Code for throwing Exceptions -:

The changes required in the code above in order to throw exceptions are to be done in Application class only.

Hence, instead of pasting the whole code again and again, only the class Application code is pasted.

- *For throwing IndexOutOfBoundsException in method checkIndex, get, remove*
  Since the method 'get' and 'remove' call the 'checkIndex' method , hence *IndexOutOfBoundsException* will be thrown if any inconsistent index is entered

```
public class Application{
    public static void main(String []args){
        LinkedList <Integer> list = new LinkedList<Integer>();
        list.append(2);
        list.append(0);
        list.append(2);
        list.append(0);
        list.append(5);
        list.append(2);
        list.append(3);
        list.append(0);
        list.append(7);
        list.checkIndex(10); //or list.get(10) or list.remove(10)

    }

}
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index = 10 size = 9
        at LinkedList.checkIndex(Application.java:131)
        at Application.main(Application.java:259)
PS C:\Users\Archit\Desktop\cprog>
```

- *For throwing IndexOutOfBoundsException in method add*

```
public class Application{
```

```
    public static void main(String []args){
        LinkedList <Integer> list = new LinkedList<Integer>();
        list.append(2);
        list.append(0);
        list.append(2);
        list.append(0);
        list.append(5);
        list.append(2);
        list.append(3);
        list.append(0);
        list.append(7);
        list.add(9, 10); //no exception will be thrown because index >= 0 && <
= size and here size is 9
        //after the last statement size becomes 10
        list.add(11, 12); //now exception will be thrown
    }
}
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index = 11 Size = 10
        at LinkedList.add(Application.java:40)
        at Application.main(Application.java:261)
PS C:\Users\Archit\Desktop\cprog>
```