Data Structure Lab4 : Singly Linked List 2022-2023

Topics

1. Implement Node Class
2. Generics
3. Implement SinglyLinkedList Class
4. Implement Basic Methods of SinglyLinkedList
    - isEmpty()
    - size()
    - first()
    - last()
    - addFirst()
    - addLast()
    - removeFirst()

```java
public class SinglyLinkedList<E> {
    private static class Node<E>
    {
        private E element;
        private Node<E>next;

        public Node(E element, Node<E> next) {
            this.element = element;
            this.next = next;
        }

        public E getElement() {
            return element;
        }

        public void setElement(E element) {
            this.element = element;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> next) {
```

Data Structure Lab4 : Singly Linked List 2022-2023

```
            this.next = next;
        }
    }
    private Node<E>head=null;
    private Node<E>tail=null;
    private int size=0;
    public SinglyLinkedList(){}
    public int size(){return size;}
    public boolean isEmpty()
    {
        return size==0;
    }
    public E first()
    {
        if (isEmpty())return null;
        return head.getElement();
    }
    public E last()
    {
        if (isEmpty())return null;
        return tail.getElement();

    }
    public void addFirst(E e)
    {
//        Node<E>n= new Node<>(e,head);
//        head=n;
        head= new Node<>(e,head);
        if (size==0)
            tail=head;
        size++;
    }
    public void addLast(E e)
    {
        Node<E>n= new Node<>(e,null);
        if (size==0)
            head=n;
        else
            tail.setNext(n);
        tail=n;
        size++;

    }
```

Data Structure Lab4 : Singly Linked List 2022-2023

```
    public E removeFirst()
    {
        if (isEmpty())return null;
        E deleted=head.element;
        head=head.getNext();
        size--;
        if (size==0)
            tail=null;
        return deleted;
    }
    public String getAll()
    {
        String all="";
        Node<E>p=head;
        while (p!=null)
        {
            all=all+p.getElement()+"  ";
            p=p.next;
        }
        return all;
    }
}

import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        SinglyLinkedList<String>l= new SinglyLinkedList<>();
        Scanner in= new Scanner(System.in);
        int choice;
        while (true)
        {
            System.out.println("1 add first 2 add last 3 remove
first 4 size 5 Is the list empty? -1 exit");
            System.out.println("input your choice: ");
            choice=in.nextInt();
            switch (choice)
            {
                case 1:
                    System.out.println("input an element : ");
                    l.addFirst(in.next());
                    System.out.println(l.first()+" was added
successfully");
```

Data Structure Lab4 : Singly Linked List 2022-2023

```
                    break;
                case 2:
                    System.out.println("input an element : ");
                    l.addLast(in.next());
                    System.out.println(l.last()+" was added
successfully");
                    break;
                case 3:
                    System.out.println(l.removeFirst()+" was removed
successfully");
                    break;
                case 4:
                    System.out.println(l.size()+" is the size of the
list");
                    break;
                case 5:
                    System.out.println("Is the list empty?
"+l.isEmpty());
                    break;
                case -1:
                    System.out.println("good bye");
                    System.exit(0);

            }
            System.out.println("List elements are : "+l.getAll());
        }
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

   def equals(self, other):

   a, b = self.head, other.head

```
    while a and b:
        if a.data != b.data:
            return False  # القائمتان ليستا متطابقتين
        a = a.next
        b = b.next
    return a is None and b is None  # تأكد من أن الطول متساوٍ
```

2. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.

```
def secondToLast(self):
    if self.head is None or self.head.next is None:
        return None  # لا توجد عقدة ثانية من النهاية
    current = self.head
    while current.next and current.next.next:
        current = current.next
    return current.data
```

3. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.

```
def size(self):
    count = 0
    current = self.head
    while current:
        count += 1
        current = current.next
    return count
```

4. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

```
def rotate(self):
    if self.head is None or self.head.next is None:
        return  # القائمة فارغة أو تحتوي على عقدة واحدة فقط
    old_head = self.head
    self.head = old_head.next
```

```
        current = self.head
        while current.next:
            current = current.next
        current.next = old_head
        old_head.next = None
```

5. Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.

```
def concatenate(self, other):
    if self.head is None:
        self.head = other.head  # إذا كانت القائمة الأولى فارغة
    else:
        current = self.head
        while current.next:
            current = current.next
        current.next = other.head
```

6. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

```
def reverse(self):
    prev = None
    current = self.head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    self.head = prev
```