Data Structure Lab4 : Singly Linked List 2022-2023

Topics

1. Implement Node Class

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

    public Node(T data) {
        this.data = data;
        this.next = null;
    }
}
```

2. Generics

```
public class SinglyLinkedList<T> {
    private Node<T> head;
}
```

3. Implement SinglyLinkedList Class

```
public boolean isEmpty() {
    return head == null;
}
```

4. Implement Basic Methods of SinglyLinkedList
    - isEmpty()

```
public boolean isEmpty() {
    return head == null;
}
```

    - size()

```
public int size() {
```

```
    if (head == null) {
        return 0;
    }
    int count = 0;
    Node<T> current = head;
    while (current != null) {
        count++;
        current = current.next;
    }
    return count;
}
```

- first()

```
public T first() {
    if (isEmpty()) {
        throw new NoSuchElementException();
    }
    return head.data;
}
```

- last()

```
public T last() {
    if (isEmpty()) {
        throw new NoSuchElementException();
    }
    Node<T> current = head;
    while (current.next != null) {
        current = current.next;
    }
    return current.data;
}
```

- addFirst()

Data Structure Lab4 : Singly Linked List 2022-2023

```
public void addFirst(T item) {
    Node<T> newNode = new Node<>(item);
    newNode.next = head;
    head = newNode;
}
```

- addLast()

```
public void addLast(T item) {
    if (isEmpty()) {
        addFirst(item);
        return;
    }
    Node<T> newNode = new Node<>(item);
    Node<T> current = head;
    while (current.next != null) {
        current = current.next;
    }
    current.next = newNode;
}
```

- removeFirst()

```
public T removeFirst() {
    if (isEmpty()) {
        throw new NoSuchElementException();
    }
    T data = head.data;
    head = head.next;
    return data;
}
```

Data Structure Lab4 : Singly Linked List 2022-2023

Homework

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

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

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

class SinglyLinkedList<T> {
    private Node<T> head;

}
```

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.

```
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    SinglyLinkedList other = (SinglyLinkedList) o;

    Node<T> current1 = this.head;
    Node<T> current2 = other.head;

    while (current1 != null && current2 != null) {
        if (!current1.data.equals(current2.data)) {
            return false;
        }
```

```
        current1 = current1.next;
        current2 = current2.next;
    }
    return current1 == null && current2 == null;
}
```

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

```
public T getSecondToLast() {
    if (head == null || head.next == null) {
        throw new NoSuchElementException(); // القائمة فارغة أو تحتوي على عنصر واحد
    }

    Node<T> current = head;
    while (current.next.next != null) {
        current = current.next;
    }
    return current.data;
}
```

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

```
public int size() {
    int count = 0;
    Node<T> current = head;
    while (current != null) {
```

```
        count++;
        current = current.next;
    }
    return count;
}
```

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.

```
public void rotate() {
    if (head == null || head.next == null) {
        return; // لا يوجد شيء يدور
    }

    Node<T> current = head;
    while (current.next != null) {
        current = current.next;
    }
    current.next = head;
    head = head.next;
    current.next = null;
}
```

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

```
public void reverse() {
    Node<T> prev = null;
    Node<T> current = head;
```

```
    Node<T> next = null;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}
```