CS201
Data Structures and Algorithms
Revision Session 2

doubly linked lists
circular linked lists

Doubly Linked Lists
implementation:
double node and linked list

algorithms:
insertion (last, first, middle)
deletion (last, first, middle)

Circular Linked Lists
implementation:
double node and linked list

algorithms:
insertion (last, first, middle)
deletion (last, first, middle)

doubly linked lists

implementation

```java
21 usages  neslihancesurr
public class DoubleNode {
    27 usages
    public DoubleNode previous;
    public DoubleNode next;
    public int data;

    no usages  neslihancesurr
    public DoubleNode(int data){
        this.data = data;
        next = null;
        previous = null;
    }
}
```

```java
no usages  neslihancesurr *
public class DoubleList {
    28 usages
    DoubleNode head;
    19 usages
    DoubleNode tail;

    no usages  neslihancesurr
    public DoubleList() {
        head = null;
        tail = null;
    }
```

insertion algorithms

insert first

1. if empty change the tail
2. if not empty, next of new node points to the (old) head
3. previous of (old) head points to the new node
4. whether the list is empty or not, change the head to point to the new node

```
no usages   neslihancesurr *
public void insertFirst(DoubleNode newNode) {

    if (tail == null) {
        tail = newNode;
    } else {
        newNode.next = head;
        head.previous = newNode;
    }
    head = newNode;
}
```

last

1. if list is empty, make head point to new node
2. if not empty, make next of tail point to new node
3. make previous of new node point to (old) tail
4. whether the list is empty or not, make tail point to new node.

```
no usages  neslihancesurr
public void insertLast(DoubleNode newNode) {

    if (head == null) {
        head = newNode;
    } else {
        tail.next = newNode;
        newNode.previous = tail;
    }

    tail = newNode;
}
```

insert middle

(change fields of new node first to avoid information loss)

1. previous field of new node points to before node
2. next field of new node points to following node
3. previous field of the following node points to new node
4. next field of before node points to new node

```
no usages  new *
public void insertMiddle(DoubleNode newNode, DoubleNode before) {
    newNode.previous = before;
    newNode.next = before.next;
    before.next.previous = newNode;
    before.next = newNode;
}
```

deletion algorithms

delete first

1. if one element head and tail are null
2. if not, following node is the new head
3. new head’s previous node points to null

```
no usages  neslihancesurr
public void deleteFirst() {
    if (tail == head) {
        head = null;
        tail = null;
    } else {
        head = head.next;
        head.previous = null;
    }
}
```

```
no usages  neslihancesurr
public void deleteFirstNode() {
    head = head.next;

    if (head == null) {
        tail = null;
    } else {
        head.previous = null;
    }
}
```

delete last

1. tail points to the node before last
2. if no node before last (only one node), make head point to null
3. cut the connection to the deleted node

```
public void deleteLast() {
    tail = tail.previous;

    if (tail == null) {
        head = null;
    } else {
        tail.next = null;
    }
}
```

delete middle

1. next field of previous node point to the following node
2. previous field of following node point to the previous node

   (no need to change the fields of deleted node)

```
public void deleteMiddle(DoubleNode deleted) {
    deleted.previous.next = deleted.next;
    deleted.next.previous = deleted.previous;
}
```

circular linked lists

insert first

1. if list is empty, the fields of new node are connected to itself
2. if not, next field of new node is linked to following and the previous node is linked to previous

```
public void insertFirst(DoubleNode newNode){
    if (head == null){
        newNode.next = newNode;
        newNode.previous = newNode;
    } else {
        newNode.next = head;
        newNode.previous = head.previous;
        head.previous.next = newNode;
        head.previous = newNode;
    }
    head = newNode;
}
```

3. next field of previous node is linked to new node
4. previous field of following node is linked to new node

delete first

1. if there is one element, delete it
2. if there is more than one, next field of previous node points to new node
3. previous node of following node points to new node
4. update the head

```
public void deleteFirst(){
    if (head.next == head){
        head = null;
    } else {
        head.previous.next = head.next;
        head.next.previous = head.previous;
        head = head.next;
    }
}
```