1. Linked List

**Pengertian**: Sebuah LinkedList adalah kumpulan elemen yang terhubung, di mana setiap elemen memiliki referensi ke elemen berikutnya dalam urutan. Setiap elemen, atau node, terdiri dari data dan referensi ke node berikutnya.

**Single LinkedList:**

Pada single linked list, setiap node memiliki referensi hanya ke node berikutnya dalam urutan.

Implementasi ini lebih sederhana dan membutuhkan ruang memori yang lebih sedikit karena setiap node hanya perlu menyimpan satu referensi.

**Method Penting: Beberapa method penting dalam LinkedList termasuk:**

addFirst(): Menambahkan elemen di awal linked list.

addLast(): Menambahkan elemen di akhir linked list.

removeFirst(): Menghapus elemen di awal linked list.

removeLast(): Menghapus elemen di akhir linked list.

getFirst(): Mendapatkan elemen pertama dalam linked list.

getLast(): Mendapatkan elemen terakhir dalam linked list.

Class awal untuk manual (single)

class Node {

Object data;

Node next;

public Node(Object data) {

this.data = data;

this.next = null;

}

}

Class yang akan menggunakan manual di atas (single)

class LinkedList {

private Node head;

private NodeTail;

public void **addFirst(Object data)** {

Node newNode = new Node(data);

newNode.next = head;

head = newNode;

}

public void **addLast(Object data)** {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

return;

}

tail.next = newNode;

tail = newNode;

}

public void **removeFirst**() {

if (head == null) {

System.out.println("Linked list kosong, tidak ada yang dihapus.");

return;

}

head = head.next;

}

public void **removeLast()** {

if (head == null) {

System.out.println("Linked list kosong, tidak ada yang dihapus.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

Node current = head;

while (current.next != tail) {

current = current.next;

}

current.next = null;

tail = current;

}

public Object **getFirst()** {

if (head == null) {

System.out.println("Linked list kosong.");

return null;

}

return head.data;

}

public Object **getLast()** {

if (tail == null && head == null) {

System.out.println("Linked list kosong.");

return null;

} else {

return tail.data;

}

}

}

**Double LinkedList:**

Pada double linked list, setiap node memiliki referensi baik ke node sebelumnya maupun node berikutnya dalam urutan.

Hal ini memungkinkan traversal maju dan mundur dengan mudah, serta memungkinkan penghapusan node dengan mengacu pada referensi sebelumnya.

Implementasi ini membutuhkan lebih banyak ruang memori karena setiap node perlu menyimpan dua referensi.

Class awal untuk manual (double)

class Node {

Object data;

Node next;

Node prev;

public Node(Object data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

public void **addFirst(Object data)** {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

newNode.next = head;

head.prev = newNode;

head = newNode;

}

}

public void **addLast(Object data)** {

Node newNode = new Node(data);

if (tail == null) {

head = newNode;

tail = newNode;

} else {

newNode.prev = tail;

tail.next = newNode;

tail = newNode;

}

}

public void **removeFirst()** {

if (head == null) {

System.out.println("Linked list kosong, tidak ada yang dihapus.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

head = head.next;

head.prev = null;

}

public void **removeLast()** {

if (tail == null) {

System.out.println("Linked list kosong, tidak ada yang dihapus.");

return;

}

if (head == tail) {

head = null;

tail = null;

return;

}

tail = tail.prev;

tail.next = null;

}

public Object **getFirst()** {

if (head == null) {

System.out.println("Linked list kosong.");

return null;

}

return head.data;

}

public Object **getLast()** {

if (tail == null) {

System.out.println("Linked list kosong.");

return null;

}

return tail.data;

}

**Untuk display (double dan single)**  
public void **display()** {

Node current = head;

while (current != null) {

System.out.print(current.data);

current = current.next;

}

}

2. Stack

**Pengertian**: Stack adalah struktur data linear yang mengikuti prinsip LIFO (Last In First Out). Artinya, elemen yang terakhir dimasukkan adalah yang pertama dikeluarkan.

**Method Penting: Beberapa method penting dalam Stack termasuk:**

push(): Menambahkan elemen ke atas stack.

pop(): Menghapus dan mengembalikan elemen teratas dari stack.

peek(): Melihat elemen teratas dari stack tanpa menghapusnya.

isEmpty(): Memeriksa apakah stack kosong.

3. Queue

**Pengertian**: Queue adalah struktur data linear yang mengikuti prinsip FIFO (First In First Out). Artinya, elemen yang pertama dimasukkan adalah yang pertama dikeluarkan.

**Method Penting: Beberapa method penting dalam Queue termasuk:**

enqueue(): Menambahkan elemen ke dalam antrian.

dequeue(): Menghapus dan mengembalikan elemen pertama dari antrian.

peek(): Melihat elemen pertama dari antrian tanpa menghapusnya.

isEmpty(): Memeriksa apakah antrian kosong.