Single linked list

- a single link list is a list with nodes, each node has a data field and a reference to the next node.

```java
public class Main {

    static class Node {
        int data;
        Node next;

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

    public static void main(String[] args) {

        Node first = new Node(10);
        Node second = new Node(20);
        Node third = new Node(30);
        Node fourth = new Node(40);
        Node fifth = new Node(50);

        first.next = second;
        second.next = third;
        third.next = fourth;
        fourth.next = fifth;

        Node current = first;
        while (current != null) {
        System.out.println(current.data);
        current = current.next;

        }
    }
}
```

Doubly linked list

- this list is as same as the Singly but can be forward or backward Traversal

```java
class Node {
    int data;
    Node next;
    Node prev;

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

class GfG {

    static void forwardTraversal(Node head) {
      Node curr = head;

        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.next;
        }

        System.out.println();
    }

    static void backwardTraversal(Node tail) {

        Node curr = tail;

        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.prev;
        }

        System.out.println();
    }
```

```java
    public static void main(String[] args) {

        Node head = new Node(1);
        Node second = new Node(2);
        Node third = new Node(3);

        head.next = second;
        second.prev = head;
        second.next = third;
        third.prev = second;

        System.out.println("Forward Traversal:");
        forwardTraversal(head);

        System.out.println("Backward Traversal:");
        backwardTraversal(third);
    }
}
```

Circular linked list

- it is as same as the singly but doesn't end with null instead it goes back to the first node

```java
public class CircularLinkedList {
    private Node head;

    private class Node {
        int data;
        Node next;

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

    public void push(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            newNode.next = head;
        } else {
            Node temp = head;
            while (temp.next != head) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.next = head;
        }
    }

    public void printList() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }

        Node temp = head;
        do {
            System.out.print(temp.data + " ");
            temp = temp.next;
        } while (temp != head);
        System.out.println();
    }

    public static void main(String[] args) {
        CircularLinkedList list = new CircularLinkedList();
        list.push(12);
        list.push(15);
        list.push(10);

        list.printList();
    }
}
```

Doubly Circular List

- is as same as a doubly link list but can traverse backward and forward with no end, the first node storing the address to the last node, the last node storing the first node

```java
public class DoublyCircularLinkedList {
    private Node head;

    private class Node {
        int data;
        Node prev, next;

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

    public void push(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            newNode.next = newNode;
            newNode.prev = newNode;
        } else {
            Node last = head.prev;
            last.next = newNode;
            newNode.prev = last;
            newNode.next = head;
            head.prev = newNode;
        }
    }

    public void printList() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }

        Node temp = head;
        do {
            System.out.print(temp.data + " ");
            temp = temp.next;
        } while (temp != head);
        System.out.println();
    }

    public static void main(String[] args) {
        DoublyCircularLinkedList list = new DoublyCircularLinkedList();
        list.push(12);
        list.push(15);
        list.push(10);

        list.printList();
    }
}
```