LINK-LIST

Disusun Oleh :

HAIDAR WAHYU YASARI

F1D02410114

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MATARAM

2025

1. *Source Code*

```
 abstract class Student {
    String name;
    String studentNumber;

    abstract void showMajor();
}

class ComputerScience extends Student {
    void showMajor() {
         System.out.println(name + " (" + studentNumber + ") - Major:
          Computer Science");
    }
}

class Mathematics extends Student {
    void showMajor() {
         System.out.println(name + " (" + studentNumber + ") - Major:
          Mathematics");
    }
}

class Engineering extends Student {
    void showMajor() {
         System.out.println(name + " (" + studentNumber + ") - Major:
          Engineering");
    }
}

// -------------------- Singly Linked List --------------------
class SinglyNode {
    Student data;
    SinglyNode next;

    SinglyNode(Student data) {
        this.data = data;
        this.next = null;
    }
}

class SinglyLinkedList {
    SinglyNode head;

    void insert(Student s) {
        SinglyNode newNode = new SinglyNode(s);
        if (head == null) {
            head = newNode;
            return;
        }
        SinglyNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

    void display() {
        SinglyNode temp = head;
        while (temp != null) {
```

```
            temp.data.showMajor();
            temp = temp.next;
        }
    }
}

// -------------------- Doubly Linked List --------------------
class DoublyNode {
    Student data;
    DoublyNode next, prev;

    DoublyNode(Student data) {
        this.data = data;
        this.next = this.prev = null;
    }
}

class DoublyLinkedList {
    DoublyNode head;

    void insert(Student s) {
        DoublyNode newNode = new DoublyNode(s);
        if (head == null) {
            head = newNode;
            return;
        }
        DoublyNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.prev = temp;
    }

    void displayForward() {
        DoublyNode temp = head;
        while (temp != null) {
            temp.data.showMajor();
            temp = temp.next;
        }
    }

    void displayBackward() {
        if (head == null) return;
        DoublyNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        while (temp != null) {
            temp.data.showMajor();
            temp = temp.prev;
        }
    }
}

// -------------------- Circular Linked List --------------------
class CircularNode {
    Student data;
    CircularNode next;

    CircularNode(Student data) {
```

```
        this.data = data;
        this.next = null;
    }
}

class CircularLinkedList {
    CircularNode head = null, tail = null;

    void insert(Student s) {
        CircularNode newNode = new CircularNode(s);
        if (head == null) {
            head = tail = newNode;
            tail.next = head; // circular link
        } else {
            tail.next = newNode;
            tail = newNode;
            tail.next = head;
        }
    }

    void display() {
        if (head == null) return;
        CircularNode temp = head;
        do {
            temp.data.showMajor();
            temp = temp.next;
        } while (temp != head);
    }
}

// -------------------- Main --------------------
public class Main {
    public static void main(String[] args) {
        // Create student objects
        ComputerScience s1 = new ComputerScience();
        s1.name = "MAMAT";
        s1.studentNumber = "20250101";

        Mathematics s2 = new Mathematics();
        s2.name = "UCUP";
        s2.studentNumber = "20250102";

        Engineering s3 = new Engineering();
        s3.name = "UPIN";
        s3.studentNumber = "20250103";

        ComputerScience s4 = new ComputerScience();
        s4.name = "SUPARMAN";
        s4.studentNumber = "20250110";

        // Singly Linked List
        System.out.println("=== Singly Linked List ===");
        SinglyLinkedList sll = new SinglyLinkedList();
        sll.insert(s1);
        sll.insert(s2);
        sll.insert(s3);
        sll.insert(s4);
        sll.display();

        // Doubly Linked List
        System.out.println("\n\n=== Doubly Linked List (Forward) ===");
```

```
        DoublyLinkedList dll = new DoublyLinkedList();
        dll.insert(s1);
        dll.insert(s2);
        dll.insert(s3);
        dll.insert(s4);
        dll.displayForward();

        System.out.println("\n=== Doubly Linked List (Backward) ===");
        dll.displayBackward();

        // Circular Linked List
        System.out.println("\n\n=== Circular Linked List ===");
        CircularLinkedList cll = new CircularLinkedList();
        cll.insert(s1);
        cll.insert(s2);
        cll.insert(s3);
        cll.insert(s4);
        cll.display();
    }
}
```

The code defines an abstract class Student in Java, which serves as a base class for different types of students. It has two attributes: name to store the student's name and "`studentNumber`" to store their identification number. The class declares an abstract method "`showMajor()`" that must be implemented by any subclass, ensuring that all student types can display their major information.

2. Hasil Program

```
class ComputerScience extends Student {
    void showMajor() {
        System.out.println(name + " (" + studentNumber + ") - Major:
Computer Science");
    }
}
class Mathematics extends Student {
    void showMajor() {
        System.out.println(name + " (" + studentNumber + ") - Major:
Mathematics");
    }
}
class Engineering extends Student {
    void showMajor() {
        System.out.println(name + " (" + studentNumber + ") - Major:
Engineering");
    }
}
```

These classes (`ComputerScience`, `Mathematics`, and `Engineering`) extend the abstract Student class and provide concrete implementations of the `showMajor()` method. Each implementation prints the student's name, student number, and their specific major, demonstrating polymorphism where different student types can be treated uniformly through the Student reference while exhibiting different behaviors.

```
class SinglyNode {
    Student data;
    SinglyNode next;

    SinglyNode(Student data) {
        this.data = data;
        this.next = null;
    }
}
```

The code defines a class `SinglyNode` in Java, which represents a single node in a singly linked list. It has two attributes: data, which stores an object of type Student, and next, which holds a reference to the next node in the list. The constructor initializes the node by setting its data field with the given Student object and assigning null to next, indicating that the node is not yet linked to another/.

```
class SinglyLinkedList {
    SinglyNode head;

    void insert(Student s) {
        SinglyNode newNode = new SinglyNode(s);
        if (head == null) {
            head = newNode;
            return;
        }
        SinglyNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

    void display() {
        SinglyNode temp = head;
        while (temp != null) {
            temp.data.showMajor();
            temp = temp.next;
        }
    }
}
```

The code defines a class `SinglyLinkedList` in Java, which represents a singly linked list structure used to store and manage objects of type Student. It has a head attribute that serves

as the starting point of the list. The `insert(Student s)` method creates a new node using the `SinglyNode` class and adds it to the end of the list; if the list is empty, the new node becomes the head, otherwise the method traverses the list until the last node and links the new node to it. `The display()` method is used to traverse the list from the head to the end, calling the `showMajor()` function of each Student object stored in the nodes, which allows the program to output the major information of each student in the sequence.

```
class DoublyNode {
    Student data;
    DoublyNode next, prev;

    DoublyNode(Student data) {
        this.data = data;
        this.next = this.prev = null;
    }
}
```

The code defines a class `DoublyNode` in Java, which represents a node in a doubly linked list. It stores a Student object in data and has two references, next and prev, for connecting to the next and previous nodes. The constructor initializes the node with the given data and sets both links to null, allowing it to store student objects while supporting two-way traversal.

```
class DoublyLinkedList {
    DoublyNode head;

    void insert(Student s) {
        DoublyNode newNode = new DoublyNode(s);
        if (head == null) {
            head = newNode;
            return;
        }
        DoublyNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.prev = temp;
    }

    void displayForward() {
        DoublyNode temp = head;
        while (temp != null) {
            temp.data.showMajor();
            temp = temp.next;
        }
    }

    void displayBackward() {
        if (head == null) return;
```

```
        DoublyNode temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        while (temp != null) {
            temp.data.showMajor();
            temp = temp.prev;
        }
    }
}
```

The code defines a class `DoublyLinkedList` in Java, which represents a doubly linked list to store objects of type Student. It has a head node as the starting point, and the `insert(Student s)` method adds a new node to the end of the list, linking both forward and backward using next and prev. The `displayForward()` method traverses the list from head to tail, calling each student's `showMajor()` method, while `displayBackward()` starts from the last node and moves back to the head, allowing two-way traversal of the stored data.

```
class CircularNode {
    Student data;
    CircularNode next;

    CircularNode(Student data) {
        this.data = data;
        this.next = null;
    }
}
```

The code defines a class `CircularNode` in Java, which represents a node in a circular linked list. It has two attributes: data, which stores a Student object, and next, which points to the next node in the list. The constructor initializes the node with the given data and sets next to null, meaning it is not yet linked until connected, allowing nodes to form a circular structure where the last node points back to the first.

```
class CircularLinkedList {
    CircularNode head = null, tail = null;

    void insert(Student s) {
        CircularNode newNode = new CircularNode(s);
        if (head == null) {
            head = tail = newNode;
            tail.next = head; // circular link
        } else {
            tail.next = newNode;
            tail = newNode;
            tail.next = head;
        }
    }
```

```
    void display() {
        if (head == null) return;
        CircularNode temp = head;
        do {
            temp.data.showMajor();
            temp = temp.next;
        } while (temp != head);
    }
}
```

The code defines a class `CircularLinkedList` in Java, which represents a circular linked list for storing objects of type Student. It uses two references, head and tail, to keep track of the first and last nodes. The `insert(Student s)` method adds a new node to the list; if the list is empty, the node becomes both head and tail with its next pointing back to itself, otherwise the new node is linked after the tail and the circular connection is maintained by pointing the tail's next back to the head. The `display()` method traverses the list starting from the head and continues until it loops back to the head, calling each student's `showMajor()` method to show the data in a circular sequence.

```
public class Main {
    public static void main(String[] args) {
        // Create student objects
        ComputerScience s1 = new ComputerScience();
        s1.name = "MAMAT";
        s1.studentNumber = "20250101";

        Mathematics s2 = new Mathematics();
        s2.name = "UCUP";
        s2.studentNumber = "20250102";

        Engineering s3 = new Engineering();
        s3.name = "UPIN";
        s3.studentNumber = "20250103";

        ComputerScience s4 = new ComputerScience();
        s4.name = "SUPARMAN";
        s4.studentNumber = "20250110";

        // Singly Linked List
        System.out.println("=== Singly Linked List ===");
        SinglyLinkedList sll = new SinglyLinkedList();
        sll.insert(s1);
        sll.insert(s2);
        sll.insert(s3);
        sll.insert(s4);
        sll.display();

        // Doubly Linked List
        System.out.println("\n\n=== Doubly Linked List (Forward) ===");
        DoublyLinkedList dll = new DoublyLinkedList();
```

```
            dll.insert(s1);
            dll.insert(s2);
            dll.insert(s3);
            dll.insert(s4);
            dll.displayForward();

            System.out.println("\n=== Doubly Linked List (Backward) ===");
            dll.displayBackward();

            // Circular Linked List
            System.out.println("\n\n=== Circular Linked List ===");
            CircularLinkedList cll = new CircularLinkedList();
            cll.insert(s1);
            cll.insert(s2);
            cll.insert(s3);
            cll.insert(s4);
            cll.display();
        }
    }
```

The code defines a Main class in Java, which serves as the entry point of the program. Inside the main method, several objects of subclasses of Student are created, such as `ComputerScience`, Mathematics, and Engineering, each assigned specific names and student numbers. After that, the program demonstrates different linked list implementations: a `SinglyLinkedList`, where nodes are connected in one direction; a `DoublyLinkedList`, which allows traversal both forward and backward; and a `CircularLinkedList`, where the last node connects back to the first. Each list stores the student objects and displays their major information, showing how linked lists can be used to organize and traverse data in multiple ways while demonstrating polymorphism through the Student abstract class.

3. RESULT PROGRAM