

# Pemrograman Lanjut

Lembar Kerja Praktikum Pemrograman Berorientasi Objek dan  
Struktur Data dengan Bahasa Pemrograman Java

Diadaptasi untuk Kurikulum 2024

Penyusun

Raden Ajeng Reina Shafira Gayatri

NIM 245150407141004



Program Studi Sistem Informasi

Departemen Sistem Informasi

Fakultas Ilmu Komputer

Universitas Brawijaya

2025

# Contents

Modul 6: Single dan Double Linked-List.....	3
Percobaan.....	3
Konsep Node dalam Linked-List.....	3
Operasi dalam Single Linked-List.....	5
Operasi dalam Double Linked-List.....	18
Instruksi Pengisian Lembar Kerja.....	5

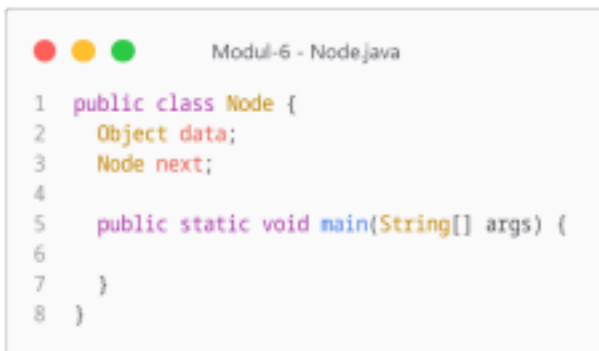
# Modul 6: Single dan Double Linked-List

## Percobaan

### Konsep Node dalam Linked-List

#### *Data dan Analisis Hasil Percobaan*

1. Buatlah sebuah project Java baru dengan nama SingleLinkedListProject
2. Buatlah Class Node di project SingleLinkedListProject dan tuliskan kode berikut:



```
Modul-6 - Node.java
1 public class Node {
2     Object data;
3     Node next;
4
5     public static void main(String[] args) {
6
7     }
8 }
```

File: Node.java

```
public class Node {

    Object data;

    Node next;

    public static void main(String[] args) {

    }

}
```

3. Tambahkan kode program berikut di dalam fungsi main() di dalam class Node.

```
Modul-6 - Node.java

1  public static void main(String[] args) {
2
3      Node node = new Node();
4      node.data = "A";
5
6      System.out.println("node      : " + node);
7      System.out.println("data      : " + node.data);
8      System.out.println("pointer   : " + node.next);
9
10 }
```

Kemudian jalankan program. Apa keluaran dari program? Mengapa demikian? Jelaskan!

Output:

File: Node.java
<pre>node      :Node@4617c264 data      :A pointer   :null</pre>

node mencetak referensi objek Node, jadi hasilnya berupa format Node@<hashcode>. data mencetak nilai "A" karena sudah diisi. pointer (node.next) belum menunjuk ke node lain, jadi nilainya null. Program membuat 1 node dengan data "A" dan pointer null, lalu mencetak semuanya.

## Operasi dalam Single Linked-List

### Data dan Analisis Hasil Percobaan

1. Buat class SingleLinkedList pada project sebelumnya pada file SingleLinkedList.java.

Tuliskan kode program berikut:

```
Modul-6 - SingleLinkedList.java

1  public class SingleLinkedList {
2
3      Node head, tail;
4      int size = 0;
5
6      void init() { head = null; }
7      boolean isEmpty() { return size == 0; }
8      int size() { return size; }
9
10     void addFirst(Node data) {
11         if (isEmpty()) {
12             head = data;
13             tail = data;
14         } else {
15             data.next = head;
16             head = data;
17         }
18         size++;
19     }
20
21     void addLast(Node data) {
22         if (isEmpty()) {
23             head = data;
24             tail = data;
25         } else {
26             tail.next = data;
27             tail = data;
28         }
29         size++;
30     }
31 }
32 }
```

- Tambahkan method main (psvm) pada class SingleLinkedList.

```
Modul-6 - SingleLinkedList.java

1  public static void main(String[] args) {
2
3      SingleLinkedList list = new SingleLinkedList();
4
5      System.out.println("Head: " + list.head);
6      System.out.println("Tail: " + list.tail);
7
8      list.addFirst(new Node());
9      System.out.println("Head: " + list.head);
10     System.out.println("Tail: " + list.tail);
11
12     list.addFirst(new Node());
13     System.out.println("Head: " + list.head);
14     System.out.println("Tail: " + list.tail);
15
16     list.addLast(new Node());
17     System.out.println("Head: " + list.head);
18     System.out.println("Tail: " + list.tail);
19
20 }
```

Jalankan program. Apa keluaran dari program? Mengapa demikian? Jelaskan!

Output:

File: SingleLinkedList.java

```
Head: null
Tail: null
Head: Node@f6f4d33
Tail: Node@f6f4d33
Head: Node@23fc625e
Tail: Node@f6f4d33
Head: Node@23fc625e
Tail: Node@3f99bd52
```

Program mencetak referensi head dan tail pada setiap langkah. Awalnya, keduanya bernilai null karena linked list kosong. Saat `addFirst` dipanggil pertama kali, sebuah node ditambahkan sehingga head dan tail menunjuk ke node yang sama. Pemanggilan `addFirst` kedua menambahkan node baru di depan, menyebabkan head berubah ke node baru sementara tail tetap menunjuk ke node pertama. Kemudian `addLast` menambahkan node baru di akhir, sehingga tail berubah ke node baru tersebut. Setiap cetakan menampilkan referensi objek Node dalam format seperti `Node@<hashcode>` yang menunjukkan alamat memori, dan mencerminkan perubahan posisi head dan tail sesuai operasi yang dilakukan.

- Pada class `Node`, tambahkan constructor yang digunakan untuk menginisialisasi nilai data

pada saat Node diciptakan:

```
Node(Object data) {
    this.data = data;
}
```

File: Node.java

```
public class Node {
    Object data;
    Node next;

    // Constructor untuk inisialisasi data
```

```

Node(Object data) {
    this.data = data;
}

public static void main(String[] args) {
    Node node = new Node("A"); // gunakan constructor

    System.out.println("node      :" + node);
    System.out.println("data      :" + node.data);
    System.out.println("pointer  :" + node.next);
}
}

```

- Ubah isi dari fungsi `main()` di dalam class `SingleLinkedList` menjadi seperti



```

Modul-6 - SingleLinkedList.java

1  SingleLinkedList list = new SingleLinkedList();
2
3  System.out.println("Head: " + list.head);
4  System.out.println("Tail: " + list.tail);
5
6  list.addFirst(new Node("A"));
7  System.out.println("Head: " + list.head.data);
8  System.out.println("Tail: " + list.tail.data);
9
10 list.addFirst(new Node("B"));
11 System.out.println("Head: " + list.head.data);
12 System.out.println("Tail: " + list.tail.data);
13
14 list.addLast(new Node("C"));
15 System.out.println("Head: " + list.head.data);
16 System.out.println("Tail: " + list.tail.data);

```

berikut ini:

Jalankan program, apa keluaran dari program? Mengapa demikian? Jelaskan!

Output:

File: SingleLinkedList.java

```

Head: null
Tail: null
Head: Node@f6f4d33
Tail: Node@f6f4d33

```

```
Head: Node@23fc625e
Tail: Node@f6f4d33
Head: Node@23fc625e
Tail: Node@3f99bd52
```

Diawal, Head dan tail adalah null karena list kosong. Setelah addFirst("A"), node pertama ditambahkan, sehingga head dan tail menunjuk ke node yang sama. Kemudian addFirst("B") menambahkan node baru di depan, membuat head berubah tapi tail tetap. Terakhir, addLast("C") menambahkan node di akhir, sehingga tail berpindah ke node baru. Hasil cetakan adalah referensi objek (Node@...) yang menunjukkan perubahan posisi head dan tail dalam memori.

- Di dalam class SingleLinkedList, lengkapi fungsi/method yang ditujukan untuk, Tulis setiap fungsi/method-nya, dan tunjukkan luaran program yang memperlihatkan keberhasilan proses setiap fungsi/method tersebut. :

- a. Mencari node dengan nilai tertentu, jika tidak ditemukan, maka kembalikan nilai null.

File: SingleLinkedList.java

```
Node find(Object value) {
    Node current = head;
    while (current != null) {
        if (current.data.equals(value)) {
            return current; // Node ditemukan
        }
        current = current.next;
    }
    return null; // Tidak ditemukan
}
```

Output:

File: SingleLinkedList.java

```
Head: null
Tail: null
Head: Node@f6f4d33
Tail: Node@f6f4d33
Head: Node@23fc625e
Tail: Node@f6f4d33
```



Head: Node@23fc625e  
Tail: Node@3f99bd52  
Cari node dengan data 'B': Node@23fc625e  
Cari node dengan data 'X': null

b. Mencari node di posisi ke-n, jika tidak ditemukan, maka kembalikan nilai null.

File: SingleLinkedList.java

```
Node get(int index) {
    if (index < 0 || index >= size) {
        return null; // jika index tidak valid
    }

    Node current = head;
    int count = 0;

    while (count < index) {
        current = current.next;
        count++;
    }
    return current;
}

public static void main(String[] args) {
    SingleLinkedList list = new SingleLinkedList();
    System.out.println("Head: "+list.head);
    System.out.println("Tail: "+list.tail);

    list.addFirst(new Node("A"));
    System.out.println("Head: "+list.head);
    System.out.println("Tail: "+list.tail);

    list.addFirst(new Node("B"));
    System.out.println("Head: "+list.head);
    System.out.println("Tail: "+list.tail);

    list.addLast(new Node("C"));
    System.out.println("Head: "+list.head);
    System.out.println("Tail: "+list.tail);

    System.out.println("\n--- Testing get(index) ---");
```

```

        Node result1 = list.get(0); // Harusnya dapat "B"
        System.out.println("Data di index 0: " + (result1 != null ? result1.data :
"null"));

        Node result2 = list.get(1); // Harusnya dapat "A"
        System.out.println("Data di index 1: " + (result2 != null ? result2.data :
"null"));

        Node result3 = list.get(2); // Harusnya dapat "C"
        System.out.println("Data di index 2: " + (result3 != null ? result3.data :
"null"));

        Node result4 = list.get(3); // Tidak ada, out of bounds
        System.out.println("Data di index 3: " + (result4 != null ? result4.data :
"null"));

    }}

```

Output:

File: SingleLinkedList.java

```

Head: null
Tail: null
Head: Node@f6f4d33
Tail: Node@f6f4d33
Head: Node@23fc625e
Tail: Node@f6f4d33
Head: Node@23fc625e
Tail: Node@3f99bd52

--- Testing get(index) ---
Data di index 0: B
Data di index 1: A
Data di index 2: C
Data di index 3: null

```

c. Menghapus node di posisi (index) ke-n, jika ada/ditemukan.

File: SingleLinkedList.java

```
void removeAt(int index) {
    if (index < 0 || index >= size) {
        System.out.println("Index tidak valid!");
        return;
    }

    if (index == 0) {
        // Hapus node pertama
        head = head.next;
        if (head == null) {
            tail = null;
        }
    } else {
        Node current = head;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = current.next.next;

        if (index == size - 1) {
            tail = current; // update tail jika node terakhir dihapus
        }
    }
    size--;
}

System.out.print("Isi list sebelum dihapus index ke-1: ");
list.printList();

list.removeAt(1); // menghapus node di index ke-1

System.out.print("Isi list setelah dihapus index ke-1: ");
list.printList();
```

Output:

File: SingleLinkedList.java

```
Head: null
Tail: null
Head: Node@6d06d69c
Tail: Node@6d06d69c
Head: Node@7852e922
Tail: Node@6d06d69c
Head: Node@7852e922
Tail: Node@4e25154f
Isi list sebelum dihapus index ke-1: B -> A -> C -> null
Isi list setelah dihapus index ke-1: B -> C -> null
```

d. Menghapus node dengan nilai data tertentu jika ada/ditemukan.

File: SingleLinkedList.java

```
boolean remove(Object targetData) {
    if (isEmpty()) return false;

    // Jika node yang dihapus adalah head
    if (head.data.equals(targetData)) {
        head = head.next;
        if (head == null) tail = null; // jika list jadi kosong
        size--;
        return true;
    }

    Node current = head;
    while (current.next != null) {
        if (current.next.data.equals(targetData)) {
            // Jika node yang dihapus adalah tail
            if (current.next == tail) {
```

```

        tail = current;
    }
    current.next = current.next.next;
    size--;
    return true;
}
current = current.next;
}

return false; // jika data tidak ditemukan
}
System.out.println("\nIsi list sebelum hapus:");
list.printList();

System.out.println("Hapus B: " + list.remove("B")); // hapus tengah (head)
System.out.println("Hapus C: " + list.remove("C")); // hapus tail
System.out.println("Hapus Z (tidak ada): " + list.remove("Z")); // tidak ditemukan

System.out.println("\nIsi list setelah hapus:");
list.printList();

System.out.println("Ukuran list sekarang: " + list.size());

```

Output:

File: SingleLinkedList.java

```

Head: null
Tail: null
Head: Node@1b6d3586
Tail: Node@1b6d3586
Head: Node@4554617c
Tail: Node@1b6d3586
Head: Node@4554617c
Tail: Node@74a14482

```

Isi list sebelum hapus:

```
B -> A -> C -> null
Hapus B: true
Hapus C: true
Hapus Z (tidak ada): false
```

Isi list setelah hapus:

```
A -> null
```

```
Ukuran list sekarang: 1
```

e. Menambah node di posisi (index) ke-n.

File: SingleLinkedList.java

```
void addAt(int index, Node data) {
    if (index < 0 || index > size) {
        System.out.println("Index di luar batas!");
        return;
    }

    if (index == 0) {
        addFirst(data);
    } else if (index == size) {
        addLast(data);
    } else {
        Node current = head;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        data.next = current.next;
        current.next = data;
        size++;
    }
}

public static void main(String[] args) {
    SingleLinkedList list = new SingleLinkedList();

    list.addFirst(new Node("A"));           // A
```

```

list.addLast(new Node("B"));           // A -> B
list.addAt(1, new Node("X"));          // A -> X -> B
list.addAt(0, new Node("Y"));          // Y -> A -> X -> B
list.addAt(4, new Node("Z"));          // Y -> A -> X -> B -> Z
list.addAt(10, new Node("Invalid"));   // Index out of bounds

list.printList(); // Final list: Y -> A -> X -> B -> Z -> null
}

```

Output:

File: SingleLinkedList.java

Y -> A -> X -> B -> Z -> null

Output Jika penambahan di index yang tidak valid:

File: SingleLinkedList.java

Index di luar batas!

f. Menambah node setelah node dengan nilai data tertentu.

File: SingleLinkedList.java

```

void addAfter(Object key, Node newNode) {
    Node current = head;
    while (current != null) {
        if (current.data.equals(key)) {
            newNode.next = current.next;
            current.next = newNode;

            if (current == tail) {
                tail = newNode; // Update tail jika node baru ditambahkan setelah
tail
            }
        }
    }
}

```

```

        size++;
        return;
    }
    current = current.next;
}
System.out.println("Node dengan data " + key + " tidak ditemukan.");
}
System.out.println("\nSetelah addAfter 'A' -> 'D':");
list.addAfter("A", new Node("D"));
list.printList(); // Output: B -> A -> D -> C -> null

System.out.println("\nSetelah addAfter 'C' -> 'E':");
list.addAfter("C", new Node("E"));
list.printList(); // Output: B -> A -> D -> C -> E -> null

System.out.println("\nSetelah addAfter 'X' -> 'Z' (gagal):");
list.addAfter("X", new Node("Z"));
list.printList(); // Tidak ada perubahan, karena X tidak ditemukan

```

Output:

File: SingleLinkedList.java

```

Head: null
Tail: null
Head: Node@6d06d69c
Tail: Node@6d06d69c
Head: Node@7852e922
Tail: Node@6d06d69c
Head: Node@7852e922
Tail: Node@4e25154f

Setelah addAfter 'A' -> 'D':
B -> A -> D -> C -> null

```



Setelah addAfter 'C' -> 'E':  
B -> A -> D -> C -> E -> null

Setelah addAfter 'X' -> 'Z' (gagal):  
Node dengan data X tidak ditemukan.  
B -> A -> D -> C -> E -> null

g. Menambah node sebelum node dengan nilai data tertentu.

File: SingleLinkedList.java

```
void addBefore(Object targetData, Node newNode) {
    if (isEmpty()) return;

    // Jika node yang dicari adalah head
    if (head.data.equals(targetData)) {
        addFirst(newNode);
        return;
    }

    Node current = head;
    Node prev = null;

    while (current != null && !current.data.equals(targetData)) {
        prev = current;
        current = current.next;
    }

    if (current != null) { // ditemukan
        prev.next = newNode;
        newNode.next = current;
        size++;
    } else {
        System.out.println("Data '" + targetData + "' tidak ditemukan.");
    }
}
```

```
System.out.println("Sebelum addBefore:");  
list.printList();  
  
list.addBefore("A", new Node("X")); // Tambah sebelum "A"  
  
System.out.println("Setelah addBefore(X sebelum A):");  
list.printList();
```

Output:

File: SingleLinkedList.java

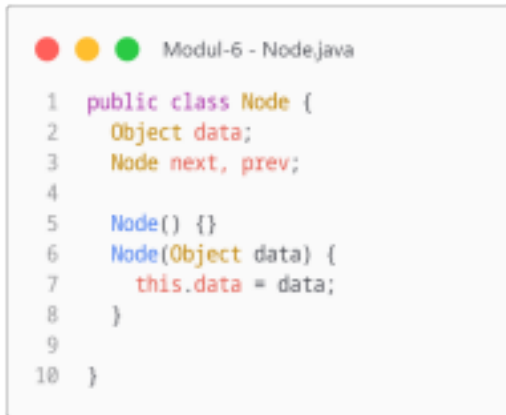
```
Head: null  
Tail: null  
Head: Node@xxxx  
Tail: Node@xxxx  
Head: Node@yyyy  
Tail: Node@xxxx  
Head: Node@yyyy  
Tail: Node@zzzz  
Sebelum addBefore:  
B -> A -> C -> null  
Setelah addBefore(X sebelum A):  
B -> X -> A -> C -> null
```

## Operasi dalam Double Linked-List

Memahami operasi-operasi list sederhana yang ada pada Double Linked-List

### *Data dan Analisis*

1. Buat project baru dengan nama DoubleLinkedListProject, kemudian di dalam project tersebut buatlah class *Node* di dalam file Node.js dan tuliskan kode program berikut ini:

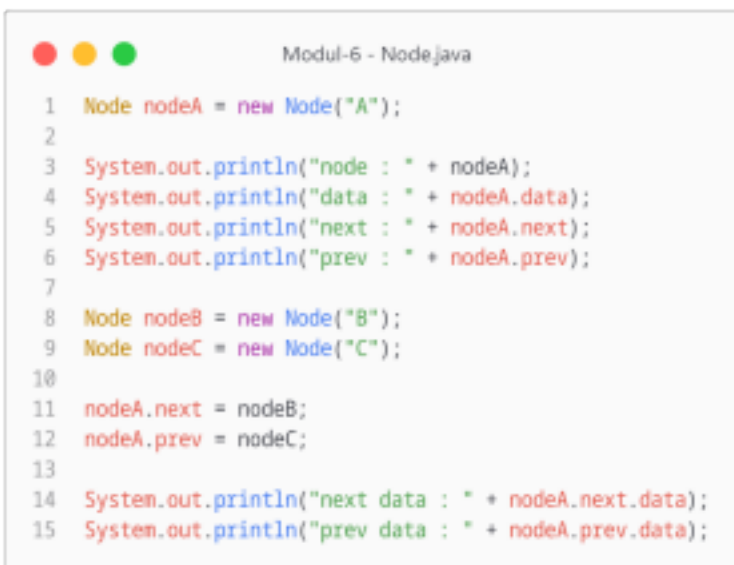


```

1 public class Node {
2     Object data;
3     Node next, prev;
4
5     Node() {}
6     Node(Object data) {
7         this.data = data;
8     }
9
10 }

```

2. Buat method main (psvm) di dalam class Node, dengan definisi body untuk method main() tersebut adalah sebagai berikut:



```

1 Node nodeA = new Node("A");
2
3 System.out.println("node : " + nodeA);
4 System.out.println("data : " + nodeA.data);
5 System.out.println("next : " + nodeA.next);
6 System.out.println("prev : " + nodeA.prev);
7
8 Node nodeB = new Node("B");
9 Node nodeC = new Node("C");
10
11 nodeA.next = nodeB;
12 nodeA.prev = nodeC;
13
14 System.out.println("next data : " + nodeA.next.data);
15 System.out.println("prev data : " + nodeA.prev.data);

```

Apa keluaran dari program? Mengapa bisa demikian?

Output:

File: Node.java
<pre> node : Node@4617c264 data : A next : null prev : null next data : B prev data : C </pre>

Program mencetak informasi objek nodeA. Awalnya, next dan prev masih null sehingga ditampilkan

sebagai null. Setelah nodeA.next diset ke nodeB dan nodeA.prev ke nodeC, maka nodeA.next.data menghasilkan B dan nodeA.prev.data menghasilkan C. Representasi objek nodeA seperti Node@<hashcode> muncul karena toString() tidak dioverride.

3. Buat class DoubleLinkedList di dalam project yang sama dan tuliskan kode program berikut:

```
Modul-6 - DoubleLinkedList.java

1 public class DoubleLinkedList {
2     Node head, tail;
3     int size = 0;
4
5     void init() { head = null; }
6     boolean isEmpty() { return size == 0; }
7     int size() { return size; }
8
9     void addFirst(Node data) {
10         if (isEmpty()) {
11             head = data;
12             tail = data;
13         } else {
14             data.next = head;
15             head.prev = data;
16             head = data;
17         }
18         size++;
19     }
20
21     void addLast(Node data) {
22         if (isEmpty()) {
23             head = data;
24             tail = data;
25         } else {
26             tail.next = data;
27             data.prev = tail;
28             tail = data;
29         }
30         size++;
31     }
32
33     void print() {
34         Node current = head;
35         while (current != null) {
36             System.out.println(current.data);
37             current = current.next;
38         }
39     }
40 }
```

4. Di dalam class DoubleLinkedList tambahkan method main (psvm) dengan body method berisikan kode program berikut:

```

Modul-6 - DoubleLinkedList.java

1  DoubleLinkedList list =
2      new DoubleLinkedList();
3
4  Node nodeA = new Node("A");
5  Node nodeB = new Node("B");
6  Node nodeC = new Node("C");
7
8  list.addFirst(nodeA);
9  System.out.println("head: " + list.head.data);
10 System.out.println("tail: " + list.tail.data);
11
12 list.addLast(nodeB);
13 System.out.println("head: " + list.head.data);
14 System.out.println("tail: " + list.tail.data);
15
16 list.addLast(nodeC);
17 System.out.println("head: " + list.head.data);
18 System.out.println("tail: " + list.tail.data);
19
20 list.print();

```

Jalankan kode program, catat keluarannya! Mengapa keluaran kode program demikian? Jelaskan!

Output:

File: DoubleLinkedList.java
head: A
tail: A
head: A
tail: B
head: A
tail: C
A
B
C

Awalnya list kosong, addFirst(nodeA) membuat head dan tail menunjuk ke A. Kemudian addLast(nodeB) menambahkan B di belakang A, sehingga tail jadi B, A.next menunjuk B, dan B.prev menunjuk A. Lalu addLast(nodeC) menambahkan C di belakang B, membuat tail jadi C, B.next menunjuk C, dan C.prev menunjuk B. Saat print(), program mencetak isi list dari head ke tail sesuai urutan: A → B → C.

5. Ubah statement `list.addFirst()` yang terletak pada baris ke-4 di dalam method `main()` menjadi `list.addLast()`. Jalankan kode program dan catat keluarannya. Ada perbedaan apa antara keluaran program dari tahap No. 4 dengan keluaran program di tahap No. 5 ini? Mengapa demikian?

File: DoubleLinkedList.java

```
head: A
tail: A
head: A
tail: B
head: A
tail: C
A
B
C
```

Tidak ada perbedaan output sama sekali. Karena list masih kosong saat nodeA ditambahkan, baik `addFirst(nodeA)` maupun `addLast(nodeA)` akan mengatur head dan tail ke nodeA. Keduanya memiliki logika identik saat list kosong. Maka hasil akhirnya tetap:

-head adalah "A", dan

-tail adalah "C",

dengan urutan cetak  $A \rightarrow B \rightarrow C$ .

6. Setelah statement yang berfungsi untuk mencetak isi dari list, tambahkan statement berikut ini di dalam method `main()`:

```
System.out.println("Next of B: " + nodeB.next.data);
System.out.println("Prev of B: " + nodeB.prev.data);
```

Jalankan program dan catat keluarannya. Mengapa nilai/keluaran program yang ditampilkan demikian? Jelaskan!

File: DoubleLinkedList.java

```
head: A
tail: A
head: A
tail: B
head: A
```

```
tail: C
A
B
C
Next of B: C
Prev of B: A
```

Nilai Next of B adalah C dan Prev of B adalah A karena nodeB dimasukkan ke dalam list menggunakan addLast, setelah nodeA, dan sebelum nodeC. Dalam struktur double linked list, nodeB berada di tengah, sehingga nodeB.next menunjuk ke nodeC dan nodeB.prev menunjuk ke nodeA. Inilah sebabnya keluaran yang ditampilkan adalah Next of B: C dan Prev of B: A.

7. Buat sebuah method di dalam class DoubleLinkedList yang menampilkan urutan elemen di dalam Double Linked-List dari belakang. Demokan kode programnya dan jelaskan mengapa demikian!

File: DoubleLinkedList.java

```
void printReverse() {
    Node current = tail;
    while (current != null) {
        System.out.println(current.data);
        current = current.prev;
    }
}

public static void main(String[] args) {
    DoubleLinkedList list = new DoubleLinkedList();

    Node nodeA = new Node("A");
    Node nodeB = new Node("B");
    Node nodeC = new Node("C");

    list.addLast(nodeA);
    list.addLast(nodeB);
    list.addLast(nodeC);

    list.print(); // A B C

    System.out.println("Print from tail to head:");
    list.printReverse(); // C B A
}
```

## Output

File: DoubleLinkedList.java

```
A
B
C
Print from tail to head:
C
B
A
```

-list.print(); mencetak dari head ke tail → hasilnya: A B C

-list.printReverse(); mencetak dari tail ke head → hasilnya: C B A

Karena double linked list menyimpan pointer dua arah (next dan prev), maka traversal bisa dilakukan dari depan atau belakang.

8. Di dalam class DoubleLinkedList, lengkapi fungsi/method yang ditujukan untuk:

a. Mencari node dengan nilai tertentu, jika tidak ditemukan, maka kembalikan nilai null.

File: DoubleLinkedList.java

```
Node findByValue(Object value) {
    Node current = head;
    while (current != null) {
        if (current.data.equals(value)) {
            return current;
        }
        current = current.next;
    }
    return null;
}

System.out.println("Find node with value B: " + list.findByValue("B").data);
```



## Output

File: DoubleLinkedList.java
Find node at index 1: B

b. Mencari node di posisi ke-n, jika tidak ditemukan, maka kembalikan nilai null.

File: DoubleLinkedList.java
<pre>Node findByIndex(int index) {     if (index &lt; 0    index &gt;= size) return null;     Node current = head;     for (int i = 0; i &lt; index; i++) {         current = current.next;     }     return current; }</pre> <pre>System.out.println("Find node at index 1: " + list.findByIndex(1).data);</pre>

## Output

File: DoubleLinkedList.java
Find node at index 1: B

c. Menghapus node di posisi (index) ke-n, jika ada/ditemukan.

File: DoubleLinkedList.java
<pre>void removeAt(int index) {     Node target = findByIndex(index);     if (target == null) return;      if (target == head) {</pre>

```

        head = head.next;
        if (head != null) head.prev = null;
        else tail = null;
    } else if (target == tail) {
        tail = tail.prev;
        tail.next = null;
    } else {
        target.prev.next = target.next;
        target.next.prev = target.prev;
    }
    size--;
}

list.removeAt(1);
list.print();

```

#### Output

File: DoubleLinkedList.java

A  
C

d. Menghapus node dengan nilai data tertentu jika ada/ditemukan.

File: DoubleLinkedList.java

```

void removeByValue(Object value) {
    Node target = findByValue(value);
    if (target == null) return;

    if (target == head) {
        head = head.next;
        if (head != null) head.prev = null;
        else tail = null;
    } else if (target == tail) {
        tail = tail.prev;
        tail.next = null;
    } else {
        target.prev.next = target.next;
        target.next.prev = target.prev;
    }
}

```

```
    }  
    size--;  
}  
  
list.removeByValue("C");  
list.print();
```

#### Output

File: DoubleLinkedList.java

A

e. Menambah node di posisi (index) ke-n.

File: DoubleLinkedList.java

```
void addAt(int index, Node data) {  
    if (index <= 0) {  
        addFirst(data);  
    } else if (index >= size) {  
        addLast(data);  
    } else {  
        Node current = findByIndex(index);  
        data.prev = current.prev;  
        data.next = current;  
        current.prev.next = data;  
        current.prev = data;  
        size++;  
    }  
}  
  
list.addAt(1, new Node("D"));  
list.print();
```

#### Output

File: DoubleLinkedList.java

A  
D

f. Menambah node setelah node dengan nilai data tertentu.

File: DoubleLinkedList.java

```
void addAfter(Object value, Node data) {  
    Node target = findByValue(value);  
    if (target == null) return;  
  
    data.next = target.next;  
    data.prev = target;  
  
    if (target.next != null) {  
        target.next.prev = data;  
    } else {  
        tail = data;  
    }  
    target.next = data;  
    size++;  
}  
  
list.addAfter("A", new Node("E"));  
list.print();
```

Output

File: DoubleLinkedList.java

A  
E  
D

g. Menambah node sebelum node yang memiliki nilai data tertentu.

File: DoubleLinkedList.java

```
void addBefore(Object value, Node data) {
    Node target = findByValue(value);
    if (target == null) return;

    data.next = target;
    data.prev = target.prev;

    if (target.prev != null) {
        target.prev.next = data;
    } else {
        head = data;
    }
    target.prev = data;
    size++;
}

list.addBefore("D", new Node("F"));
list.print();
```

#### Output

File: DoubleLinkedList.java

A  
E  
F  
D

Tulis setiap fungsi/method-nya, dan tunjukkan luaran program yang memperlihatkan keberhasilan proses dari setiap fungsi/method tersebut.

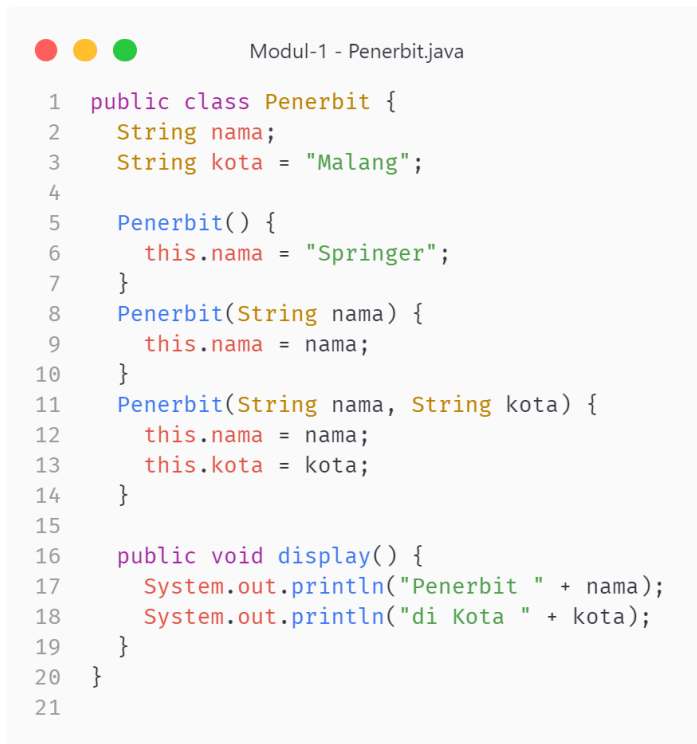
# Instruksi Pengisian Lembar Kerja

*Contoh Style Microsoft Word untuk isian data dan analisis hasil percobaan:*

Paragraf/uraian, menggunakan style Normal. Kode program/luaran program, menggunakan screenshot atau Coding Normal, seperti:

```
public class App {  
    public static void main(String[] args){  
        Mobil mobilA = new Mobil();  
    }  
}
```

Lalu dilanjutkan lagi dengan uraian paragraf menggunakan style Normal. Ukuran gambar screenshot dapat disesuaikan sedemikian rupa sehingga ukuran teks/coding di dalam screenshot sama atau lebih kecil dari ukuran font paragraf dengan style Normal. Jika dimungkinkan, ambil screenshot kode program dengan menggunakan Extension VSCode: CodeSnap agar rapi, seperti pada screenshot contoh kode program yang diperlihatkan berikut ini:



```
Modul-1 - Penerbit.java  
  
1  public class Penerbit {  
2      String nama;  
3      String kota = "Malang";  
4  
5      Penerbit() {  
6          this.nama = "Springer";  
7      }  
8      Penerbit(String nama) {  
9          this.nama = nama;  
10     }  
11     Penerbit(String nama, String kota) {  
12         this.nama = nama;  
13         this.kota = kota;  
14     }  
15  
16     public void display() {  
17         System.out.println("Penerbit " + nama);  
18         System.out.println("di Kota " + kota);  
19     }  
20 }  
21
```