LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 6: IMPLEMENTASI STRUKTUR DATA DOUBLY LINKED LIST DALAM BAHASA PEMROGRAMAN JAVA

OLEH

Abdullah Al Ramadhani (2411533016)

DOSEN PENGAMPU

DR. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. Pendahuluan

Dalam ranah pemrograman, Java menawarkan beragam struktur data yang mendukung pengolahan data secara optimal, salah satunya adalah **Doubly Linked List (DLL)**. Berbeda dari linked list satu arah, DLL memiliki keunikan karena tiap elemennya (node) menyimpan dua referensi sekaligus: satu menunjuk ke node setelahnya, dan satu lagi ke node sebelumnya. Hal ini memberikan keleluasaan dalam menjelajahi dan memodifikasi data.

Sebagai bahasa yang mendukung paradigma **object-oriented programming (OOP)**, Java memungkinkan pengembangan DLL dengan pendekatan berbasis class dan objek. Dalam praktiknya, setiap node bisa dibuat sebagai objek yang menyimpan nilai data serta dua pointer: satu ke node berikutnya (**next**) dan satu lagi ke node sebelumnya (**prev**). Pendekatan ini mencerminkan prinsip OOP seperti enkapsulasi dan interaksi antar objek.

Salah satu keunggulan utama dari DLL dalam Java adalah kemampuannya dalam melakukan traversing ke dua arah. Berkat adanya pointer ke node sebelumnya, kita bisa menelusuri data baik dari depan ke belakang maupun sebaliknya—berbeda dengan single linked list yang hanya mendukung arah maju. Kemampuan ini sangat berguna dalam skenario seperti aplikasi teks editor atau sistem yang membutuhkan pengolahan data dua arah.

Singkatnya, Java mempermudah penerapan struktur Doubly Linked List dengan pendekatan yang terorganisir dan modular. Dukungan terhadap OOP menjadikan Java sangat cocok untuk membangun sistem yang efisien dan fleksibel dengan struktur data terhubung, serta memberikan kemudahan dalam hal pengelolaan dan pemeliharaan program.

B. Tujuan

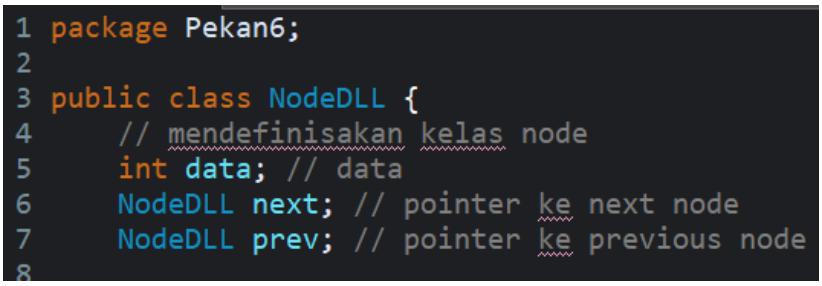
Tujuan dari praktikum ini adalah:

1. Untuk memperkenalkan konsep dasar sekaligus mekanisme kerja dari struktur data Doubly Linked List (DLL).
2. Untuk memberikan gambaran praktis tentang bagaimana struktur DLL dapat direalisasikan dalam bahasa pemrograman Java, lengkap dengan contoh kode yang mencakup operasi fundamental seperti penambahan node di awal, akhir, maupun posisi tertentu, serta penghapusan node di lokasi tertentu, termasuk di bagian kepala dan ekor list.
3. Untuk mengilustrasikan bagaimana Doubly Linked List dapat diimplementasikan menggunakan prinsip pemrograman berorientasi objek (OOP).

C. Langkah-langkah Pengerjaan

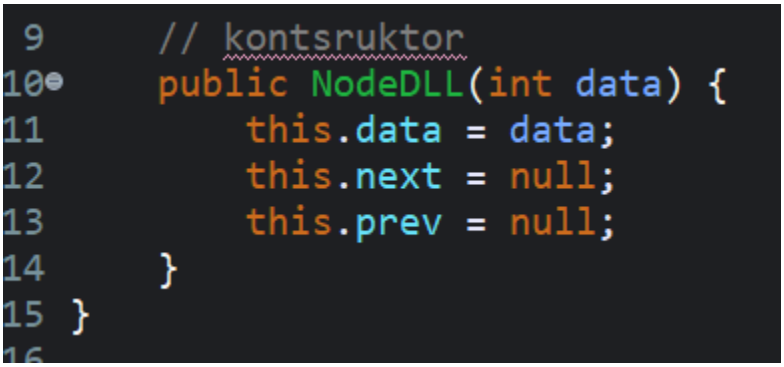
a) NodeDLL

1. Buatlah sebuah kelas baru dengan nama "NodeDLL" yang memiliki tiga atribut utama, yaitu data untuk menyimpan nilai pada node, serta next dan prev yang masing-masing berfungsi sebagai referensi ke node berikutnya dan node sebelumnya.



```
1 package Pekan6;
2
3 public class NodeDLL {
4     // mendefinisakan kelas node
5     int data; // data
6     NodeDLL next; // pointer ke next node
7     NodeDLL prev; // pointer ke previous node
8
```

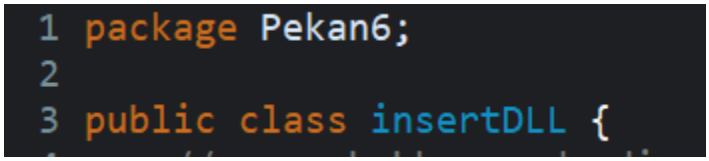
2. Setelah itu, buat konstruktor NodeDLL(int data) yang bertugas untuk membuat node baru dengan nilai tertentu, sekaligus mengatur nilai next dan prev menjadi null sebagai penanda bahwa node tersebut belum terhubung dengan node lain manapun.



```
 9     // kontsruktor
10     public NodeDLL(int data) {
11         this.data = data;
12         this.next = null;
13         this.prev = null;
14     }
15 }
16
```

b) TamabahSLL

1. Buatlah sebuah kelas baru bernama "InsertDLL" yang akan berperan sebagai tempat untuk mengelola penambahan node. Di dalam kelas ini akan terdapat beberapa metode yang digunakan untuk melakukan berbagai jenis operasi penyisipan.



```
1 package Pekan6;
2
3 public class insertDLL {
```

2. Pertama, buat metode insertBegin yang fungsinya adalah menambahkan node di bagian awal list. Metode ini akan membuat node baru dengan data yang diberikan, kemudian mengatur agar node baru tersebut menunjuk ke node head melalui atribut next. Jika head tidak kosong, maka atur atribut prev pada node head agar menunjuk kembali ke node baru. Terakhir, kembalikan node baru sebagai head yang baru.

```
    static NodeDLL insertBegin(NodeDLL head, int data) {
        // buat node baru
        NodeDLL new_node = new NodeDLL(data);
        // jadikan pointer nextnya head
        new_node.next = head;
        // jadikan pointer prev head ke new_node
        if (head != null) {
        head.prev = new_node;
        }
        return new_node;
    }
```

3. Untuk menambahkan node di bagian akhir, buat metode insertEnd. Langkahnya yaitu membuat node baru terlebih dahulu. Jika list masih kosong (head == null), maka node baru menjadi node pertama atau head. Namun jika list sudah berisi, telusuri hingga node terakhir, lalu sambungkan node baru tersebut setelah node terakhir.

```
   // fungsi menambahkan node di akhir
public static NodeDLL insertEnd(NodeDLL head, int newData) {
    // buat node baru
    NodeDLL newNode = new NodeDLL(newData);
    // jika dll null jadikan head
    if (head == null) {
        head = newNode;
    }
    else {
        NodeDLL curr = head;
        while (curr.next != null) {
            curr = curr.next;
        }
        curr.next = newNode;
        newNode.prev = curr;
    }
    return head;
}
```

4. Untuk menyisipkan node di posisi tertentu, buat metode insertAtPosition. Jika posisi yang diminta adalah 1, maka tambahkan node di awal menggunakan metode yang telah dibuat sebelumnya. Jika posisi lebih dari 1, lakukan penelusuran sampai menemukan posisi

```
// fungsi menambahkan node di posisi tertentu
public static NodeDLL insertAtPosition(NodeDLL head, int pos, int new_data) {
    //buat node baru
    NodeDLL new_node = new NodeDLL(new_data);
    if (pos == 1) {
        new_node.next = head;
        if (head != null) {
            head.prev = new_node; }
        head = new_node;
        return head; }
    NodeDLL curr = head;
for (int i = 1; i < pos - 1 && curr != null; ++i) {
        curr = curr.next; }
    if (curr == null) {
        System.out.println("Posisi tidak ada");
        return head; }
    new_node.prev = curr;
    new_node.next = curr.next;
    curr.next = new_node;
    if (new_node.next != null) {
        new_node.next.prev = new_node; }
    return head;
}
```

5. Buat metode printList untuk mencetak seluruh isi dari Doubly Linked List. Metode ini akan menelusuri list dari node head dan mencetak nilai data dari setiap node yang dilewati.

```
public static void printList(NodeDLL head) {
    NodeDLL curr = head;
    while (curr != null) {
        System.out.print(curr.data + " <-> ");
        curr = curr.next;
    }
    System.out.println();
}
```

6. Tambahkan metode main untuk menguji semua operasi yang telah dibuat, mulai dari penambahan node hingga pencetakan isi list

```
69  public static void main(String[] args) {
70      // membuat dll 2 <-> 3 <-> 5
71      NodeDLL head = new NodeDLL(2);
72      head.next = new NodeDLL(3);
73      head.next.prev = head;
74      head.next.next = new NodeDLL(5);
75      head.next.next.prev = head.next;
76      // cetak dll awal
77      System.out.println("DLL awal: ");
78      printList(head);
79      // tambah 1 di awal
80      head = insertBegin(head, 1);
81      System.out.print(
82              "Simpul 1 ditambah di awal: ");
83          printList(head);
84      // tambah 6 di akhir
85      System.out.print(
86              "simpul 6 ditambah di akhir: ");
87      int data = 6;
88      head = insertEnd(head, data);
89      printList(head);
90      // menambah node 4 di posisi 4
91      System.out.println("tambah node 4 di posisi 4: ");
92      int data2 = 4;
93      int pos = 4;
94      head = insertAtPosition(head, pos, data2);
95      printList(head);
96  }
```

7. Jalankan program. Jika semua berjalan dengan benar, maka akan muncul output sesuai dengan operasi yang telah dilakukan sebelumnya.

```
<terminated> insertDLL [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org
Nama: Abdullah Al Ramadhani
Nim: 2411533016
DLL awal:
2 <-> 3 <-> 5 <->
Simpul 1 ditambah di awal: 1 <-> 2 <-> 3 <-> 5 <->
simpul 6 ditambah di akhir: 1 <-> 2 <-> 3 <-> 5 <-> 6 <->
tambah node 4 di posisi 4:
1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6 <->
```

c) Penelusuran DLL

1. Buatlah sebuah kelas baru bernama "PenelusuranDLL" yang akan digunakan untuk menangani proses penelusuran dua arah pada struktur data Doubly Linked List (DLL). Kelas ini akan dilengkapi dengan dua metode utama, yaitu forwardTraversal untuk menelusuri dari awal ke akhir, dan backwardTraversal untuk menelusuri dari akhir ke awal.

```
1 package Pekan6;
2
3 public class PenelusuranDLL {
```

2. Selanjutnya, buat metode forwardTraversal yang akan digunakan untuk menelusuri isi list dari depan ke belakang. Penelusuran dimulai dari head (node pertama), kemudian menggunakan pointer next untuk berpindah ke node berikutnya sambil mencetak data dari tiap node. Proses ini berlanjut hingga pointer mencapai null, yang menandakan bahwa akhir list telah tercapai.

```
5●    static void forwardTraversal(NodeDLL head) {
6         //memulai penelusuran dari head
7         NodeDLL curr = head;
8         //lanjutkan sampai akhir
9         while (curr != null) {
10            //print data
11            System.out.print(curr.data + " <-> ");
12            //pindah ke node berikutnya
13            curr = curr.next;
14        }
15        //print spasi
16        System.out.println();
17    }
```

3. Untuk melakukan penelusuran dari belakang ke depan, buatlah metode backwardTraversal. Di sini, penelusuran dimulai dari tail (node terakhir) dengan memanfaatkan pointer prev untuk bergerak ke node sebelumnya, sambil mencetak data dari masing-masing node. Proses akan berhenti ketika pointer mencapai null, yang berarti kita sudah berada di awal list.

```
19    //fungsi penelusuran mundur
20●   static void backwardTraversal(NodeDLL tail) {
21        //mulai dari akhir
22        NodeDLL curr = tail;
23        //lanjut sampai head
24        while (curr != null) {
25            //cetak data
26            System.out.print(curr.data + " <-> ");
27            //pindah ke node sebelumnya
28            curr = curr.prev;
29        }
30        //cetak spasi
31        System.out.println();
32    }
```

4. Tambahkan metode main untuk menguji kedua metode penelusuran tersebut. Dalam metode ini, buat beberapa node seperti head, second, dan third, lalu hubungkan node-node tersebut menggunakan pointer next dan prev. Lakukan penelusuran maju dimulai dari head, dan penelusuran mundur dimulai dari third.

```
34●     public static void main(String[] args) {
35          //cetak DLL
36          NodeDLL head = new NodeDLL(1);
37          NodeDLL second = new NodeDLL(2);
38          NodeDLL third = new NodeDLL(3);
39
40          head.next = second;
41          second.prev = head;
42          second.next = third;
43          third.prev = second;
44
45          System.out.println("Penelusuran maju:");
46          forwardTraversal(head);
47
48          System.out.println("Penelusuran mundur:");
49          backwardTraversal(third);
50      }
51  }
```

5. Jalankan program. Jika semua berjalan sesuai harapan, maka akan muncul output yang mencerminkan hasil penelusuran baik dari awal ke akhir maupun dari akhir ke awal.

```
Nama: Abdullah Al Ramadhani
Nim: 2411533016
Penelusuran maju:
1 <-> 2 <-> 3 <->
Penelusuran mundur:
3 <-> 2 <-> 1 <->
```

d) HapusDLL

1. Buat sebuah kelas baru bernama "HapusDLL". Kelas ini berfungsi untuk menangani proses penghapusan node dalam struktur Doubly Linked List (DLL). Di dalamnya akan terdapat sejumlah metode untuk menghapus node dari posisi awal, akhir, dan lokasi tertentu, serta menampilkan isi dari list.

```
1 package Pekan6;
2
3 public class HapusDLL {
```

2. Buat metode delHead untuk menghapus node pertama pada list. Pertama-tama, periksa apakah list dalam keadaan kosong. Jika tidak kosong, geser head ke node berikutnya, lalu atur pointer prev dari node baru tersebut menjadi null agar tidak lagi terhubung dengan node yang lama.

```
//fungsi menghapus node awal
public static NodeDLL delHead(NodeDLL head) {
    if (head == null) {
        return null;    }
    NodeDLL temp = head;
    head = head.next;
    if (head != null) {
        head.prev = null;   }
    return head;
}
```

3. Lanjutkan dengan membuat metode delLast yang bertugas menghapus node di bagian akhir list. Metode ini akan menelusuri node hingga mencapai simpul terakhir menggunakan pointer next, kemudian putuskan sambungannya dengan node sebelumnya agar simpul terakhir terhapus dari list.

```
15 //fungsi menghapus di akhir
16 public static NodeDLL delLast(NodeDLL head) {
17     if (head == null) {
18         return null;    }
19     if (head.next == null) {
20         return null;    }
21     NodeDLL curr = head;
22     while (curr.next != null) {
23         curr = curr.next;
24     }
25     //update pointer previous node
26     if (curr.prev != null) {
27         curr.prev.next = null;  }
28     return head;
29 }
```

4. Untuk menghapus node pada posisi tertentu, buat metode delPos. Metode ini akan menelusuri list sampai posisi yang ditentukan. Setelah node di posisi tersebut ditemukan, sesuaikan pointer next dari node sebelumnya dan prev dari node setelahnya agar node yang ingin dihapus dilewati dan tidak lagi menjadi bagian dari list.

```
31 //fungsi menghapus node posisi tertentu
32 public static NodeDLL delPos(NodeDLL head, int pos) {
33     //jika DLL kosong
34     if (head == null) {
35         return head;    }
36     NodeDLL curr = head;
37     //telusuri sampai ke node yang akan dihapus
38     for (int i = 1; curr != null && i < pos; ++i) {
39         curr = curr.next;   }
40     //jika posisi tidak ditemukan
41     if (curr == null) {
42         return head;    }
43     //update pointer
44     if (curr.prev != null) {
45         curr.prev.next = curr.next; }
46     if (curr.next != null) {
47         curr.next.prev = curr.prev; }
48     //jika yang dihapus head
49     if (head == curr)   {
50         head = curr.next;   }
51     return head;
52 }
```

5. Tambahkan metode printList untuk menampilkan semua data dalam list. Proses ini dilakukan dengan menelusuri node mulai dari head dan mencetak data dari setiap simpul hingga mencapai akhir list.

```
54 //fungsi mencetak DLL
55 public static void printList(NodeDLL head) {
56     NodeDLL curr = head;
57     while (curr != null) {
58         System.out.print(curr.data + " ");
59         curr = curr.next;
60     }
61     System.out.println();
62 }
```

6. Terakhir, buat metode main untuk menguji ketiga metode penghapusan yang telah dibuat. Dalam metode ini, buat beberapa node terlebih dahulu, hubungkan mereka menggunakan pointer next dan prev, lalu tampilkan isi awal list. Setelah itu, lakukan penghapusan node pertama, node terakhir, dan node di posisi ke-2, kemudian cetak isi list setiap kali setelah penghapusan dilakukan.

```
64  public static void main(String[] args) {
65
66      System.out.println("Nama: Abdullah Al Ramadhani");
67      System.out.println("Nim: 2411533016");
68
69      NodeDLL head = new NodeDLL(1);
70      head.next = new NodeDLL(2);
71      head.next.prev = head;
72      head.next.next = new NodeDLL(3);
73      head.next.next.prev = head.next;
74      head.next.next.next = new NodeDLL (4);
75      head.next.next.next.prev = head.next.next;
76      head.next.next.next.next = new NodeDLL(5);
77      head.next.next.next.next.prev = head.next.next.next;
78
79      System.out.print("DLL Awal: ");
80      printList(head);
81
82      System.out.print("Setelah head dihapus: ");
83      head = delHead(head);
84      printList(head);
85
86      System.out.print("Setelah node terakhir dihapus: ");
87      head = delLast(head);
88      printList(head);
89
90      System.out.print("menghapus node ke 2: ");
91      head = delPos(head, 2);
92
93      printList(head);
94
95  }
```

7. Jalankan program. Jika berhasil, maka akan muncul hasil yang sesuai dengan urutan operasi yang dilakukan sebagaimana tampak pada contoh output.

```
Nama: Abdullah Al Ramadhani
Nim: 2411533016
DLL Awal: 1 2 3 4 5
Setelah head dihapus: 2 3 4 5
Setelah node terakhir dihapus: 2 3 4
menghapus node ke 2: 2 4
```

D. Kesimpulan

Dari hasil pembahasan dan implementasi yang dilakukan, dapat diketahui bahwa struktur data Doubly Linked List (DLL) memiliki peran penting dalam pengelolaan data yang membutuhkan kemudahan dalam manipulasi, seperti menyisipkan, menghapus, dan menelusuri elemen dari dua arah. Java, sebagai bahasa pemrograman berorientasi objek, memberikan kemudahan dalam membangun DLL lewat penggunaan class dan object, sehingga implementasinya menjadi lebih rapi, modular, dan mudah dikembangkan lebih lanjut.

Melalui proses implementasi yang telah dijalankan, terlihat bahwa DLL mampu menjalankan operasi dasar—seperti menambah node di awal, akhir, maupun posisi tertentu—dan juga menghapus node dari berbagai posisi secara efisien. Fitur traversing ke depan dan ke belakang menambah fleksibilitas penggunaan struktur ini dalam berbagai kebutuhan pengolahan data.

Dengan mempelajari dan menerapkan DLL menggunakan Java, kita tidak hanya memahami cara kerja struktur data dinamis, tetapi juga semakin terbiasa merancang program yang terstruktur dan efisien sesuai prinsip OOP. DLL menjadi fondasi penting dalam pembuatan aplikasi yang membutuhkan pengolahan data yang kompleks dan dinamis.