Laporan Praktikum Pekan 6

Struktur Data

Disusun Oleh :

Saskia Alifah

2411531002

Dosen Pengampu : Dr. Wahyudi, S.T, M.T

Departemen Informatika

Fakultas Teknologi Informasi

Universitas Andalas

Tahun 2025

Doubly Linked List (DLL)

A. Tujuan Praktikum

1. Memahami konsep dasar Doubly Linked List (DLL) sebagai struktur data dinamis yang mendukung traversal dua arah.
2. Mengimplementasikan operasi-operasi dasar pada DLL seperti penambahan, penghapusan, dan penelusuran node menggunakan bahasa pemrograman Java.
3. Menganalisis cara kerja dan efek dari setiap operasi DLL melalui pengamatan hasil keluaran program.

B. Pendahuluan

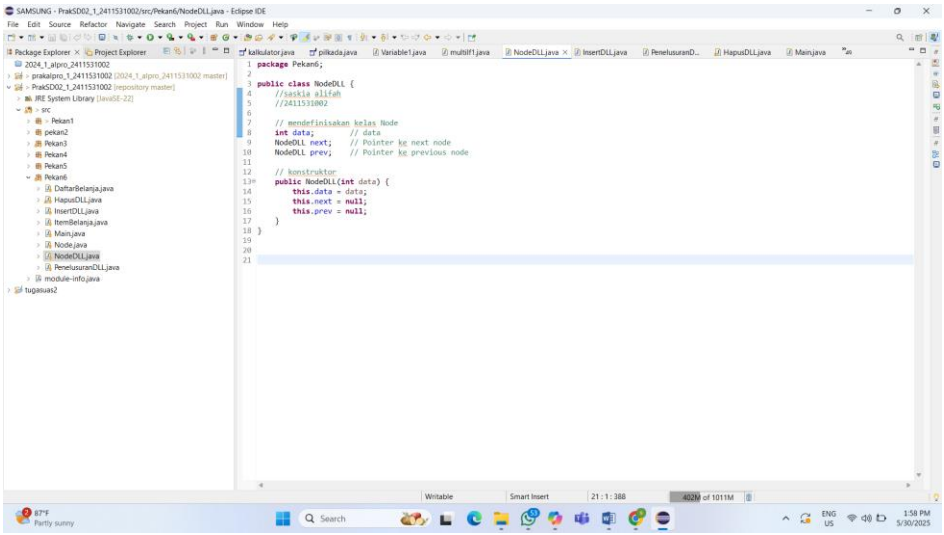
Doubly Linked List (DLL) adalah struktur data linier yang terdiri dari rangkaian node, di mana setiap node memiliki dua pointer: satu menunjuk ke node berikutnya (next) dan satu lagi menunjuk ke node sebelumnya (prev). Hal ini berbeda dengan Single Linked List (SLL) yang hanya dapat ditelusuri satu arah. Dengan dua pointer, DLL memungkinkan traversal dua arah, sehingga operasi penambahan, penghapusan, dan penelusuran node menjadi lebih fleksibel dan efisien. DLL sangat berguna dalam aplikasi yang membutuhkan navigasi dua arah, seperti fitur undo-redo, playlist musik, dan cache LRU.
Pada praktikum ini, mahasiswa mempelajari pembuatan node, penambahan node di awal, akhir, dan posisi tertentu, penghapusan node di berbagai posisi, serta traversal dua arah. Praktikum menekankan pentingnya pengelolaan pointer secara cermat untuk mencegah error seperti kehilangan referensi node, memory leak, atau infinite loop. Dengan pemahaman DLL, mahasiswa akan lebih siap menghadapi struktur data yang lebih kompleks dan aplikasi dunia nyata yang membutuhkan pengelolaan data dinamis.

C. Metode Praktikum

1. Class NodeDLL

SYNTAX:

```
package Pekan6;

public class NodeDLL {
        //saskia alifah
        //2411531002

   // mendefinisakan kelas Node
   int data;       // data
   NodeDLL next;    // Pointer ke next node
   NodeDLL prev;    // Pointer ke previous node

   // konstruktor
   public NodeDLL(int data) {
      this.data = data;
      this.next = null;
      this.prev = null;
   }
}
```



Deskripsi Kelas:

Kelas NodeDLL adalah pondasi utama dari Doubly Linked List. Kelas ini merepresentasikan satu simpul (node) dalam DLL, yang terdiri dari tiga atribut penting: data (menyimpan nilai integer), next (pointer ke node berikutnya), dan prev (pointer ke node sebelumnya). Dengan dua pointer ini, setiap node dapat diakses dari dua arah, sehingga traversal bisa dilakukan dua arah. Kelas ini hanya berfokus pada penyimpanan data dan referensi node, tanpa menyediakan operasi manipulasi secara langsung, sehingga menjadi blok bangunan dasar untuk kelas-kelas lain yang mengelola list secara keseluruhan.

Penjelasan Fungsi:

Konstruktor NodeDLL(int data) bertugas menginisialisasi node baru dengan nilai data yang diberikan, serta mengatur pointer next dan prev ke null. Dengan demikian, setiap kali objek NodeDLL dibuat, node tersebut siap untuk dihubungkan ke node lain dalam list.

Error Handling:

Pada kelas ini, error handling dilakukan secara implisit dengan menginisialisasi pointer next dan prev ke null, sehingga mencegah dereferensi pointer acak yang bisa menyebabkan error runtime. Penanganan error lebih lanjut dilakukan pada kelas lain yang memanipulasi node.

Alur Syntax:

Setiap node baru yang akan ditambahkan ke DLL selalu dibuat menggunakan konstruktor ini. Node yang baru dibuat akan dihubungkan ke node lain oleh kelas lain yang bertugas mengelola DLL.

Output dan Penjelasan:

Kelas ini sendiri tidak menghasilkan output secara langsung. Namun, tanpa kelas ini, tidak ada struktur dasar yang bisa digunakan untuk menyimpan dan menghubungkan data dalam DLL.

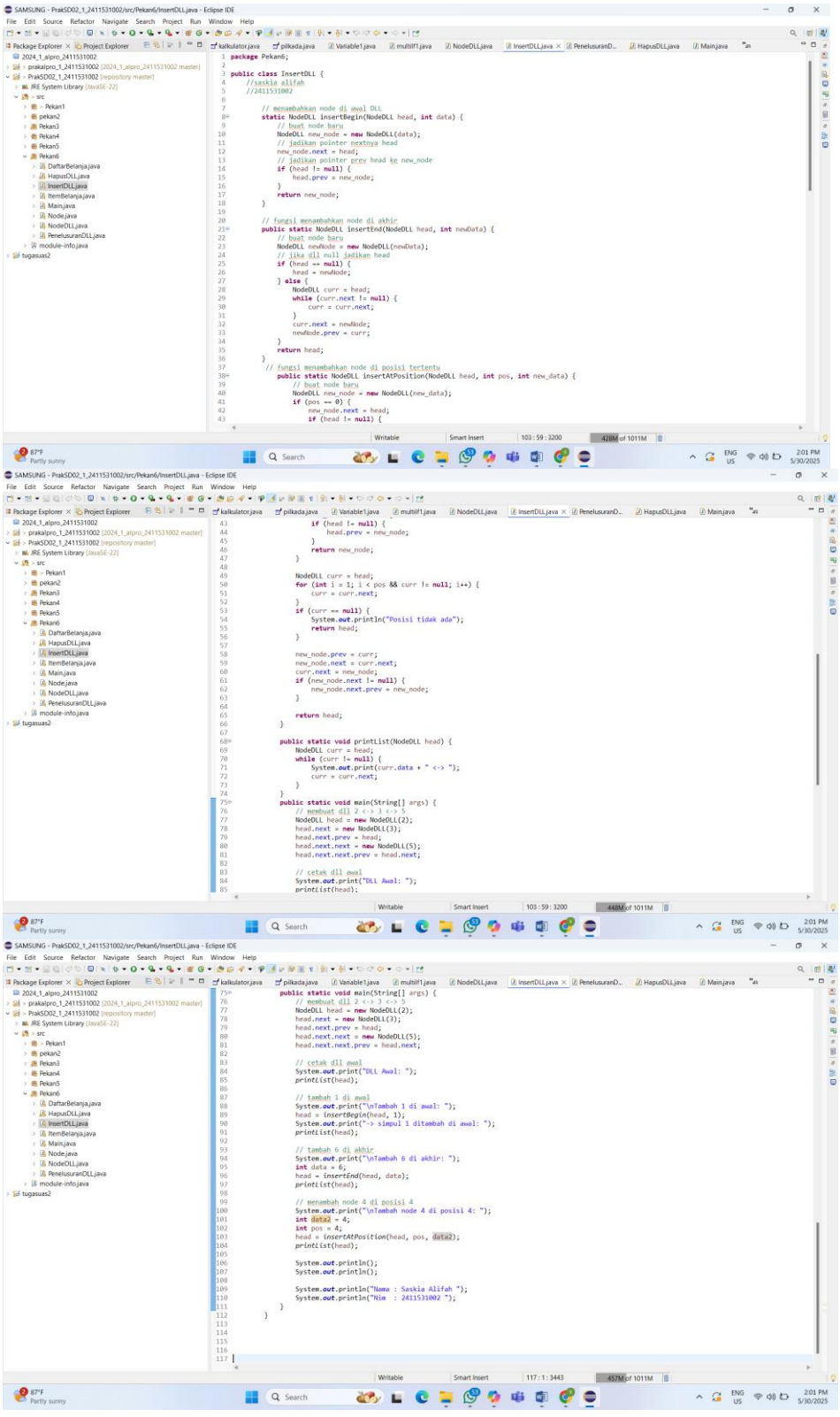
2. **Class InsertDLL**

SYNTAX:

```
package Pekan6;

public class InsertDLL {
```

```
//saskia alifah
//2411531002

    // menambahkan node di awal DLL
    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;
    }

    // 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;
    }
        // 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 == 0) {
                new_node.next = head;
                if (head != null) {
                    head.prev = new_node;
                }
                return new_node;
            }

            NodeDLL curr = head;
            for (int i = 1; i < pos && 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;
    }

    public static void printList(NodeDLL head) {
        NodeDLL curr = head;
        while (curr != null) {
            System.out.print(curr.data + " <-> ");
            curr = curr.next;
        }
    }
    public static void main(String[] args) {
        // membuat dll 2 <-> 3 <-> 5
        NodeDLL head = new NodeDLL(2);
        head.next = new NodeDLL(3);
        head.next.prev = head;
        head.next.next = new NodeDLL(5);
        head.next.next.prev = head.next;

        // cetak dll awal
        System.out.print("DLL Awal: ");
        printList(head);

        // tambah 1 di awal
        System.out.print("\nTambah 1 di awal: ");
        head = insertBegin(head, 1);
        System.out.print("-> simpul 1 ditambah di awal: ");
        printList(head);

        // tambah 6 di akhir
        System.out.print("\nTambah 6 di akhir: ");
        int data = 6;
        head = insertEnd(head, data);
        printList(head);

        // menambah node 4 di posisi 4
        System.out.print("\nTambah node 4 di posisi 4: ");
        int data2 = 4;
        int pos = 4;
        head = insertAtPosition(head, pos, data2);
        printList(head);

        System.out.println();
        System.out.println();

        System.out.println("Nama : Saskia Alifah ");
        System.out.println("Nim  : 2411531002 ");
    }
}
```



Deskripsi Kelas:

Kelas InsertDLL bertanggung jawab untuk seluruh operasi penambahan node ke dalam DLL. Kelas ini menyediakan metode untuk menambah node di awal (insertBegin), di akhir (insertEnd), dan di posisi tertentu (insertAtPosition). Selain itu, terdapat juga metode printList untuk menampilkan isi DLL setelah setiap operasi.

Penjelasan Fungsi:

- insertBegin(NodeDLL head, int data): Menambahkan node baru di posisi paling awal list. Node baru dibuat, pointer next-nya diarahkan ke head lama, dan jika head lama tidak null, pointer prev head lama diubah ke node baru. Fungsi mengembalikan node baru sebagai head baru list.
- insertEnd(NodeDLL head, int data): Menambahkan node baru di posisi akhir. Jika list kosong, node baru langsung menjadi head. Jika tidak, fungsi menelusuri hingga node terakhir, lalu menghubungkan node baru sebagai node berikutnya dan mengatur pointer prev node baru ke node terakhir.
- insertAtPosition(NodeDLL head, int pos, int data): Menyisipkan node baru pada posisi indeks tertentu. Fungsi menelusuri list hingga posisi sebelum posisi target, lalu menghubungkan node baru di antara node sebelumnya dan node berikutnya. Jika posisi melebihi panjang list, fungsi mencetak pesan error dan tidak melakukan perubahan.
- printList(NodeDLL head): Menelusuri dan mencetak seluruh node dalam list, dengan format <data> <->untuk menunjukkan hubungan antar node.

Error Handling:

Pada insertAtPosition, dilakukan validasi posisi. Jika posisi yang diminta tidak ada (melebihi panjang list), fungsi mencetak "Posisi tidak ada" dan tidak melakukan penyisipan node. Pada setiap operasi, pengecekan apakah head null juga dilakukan untuk menghindari dereferensi pointer null.

Penjelasan Fungsi Main:

Fungsi main pada kelas InsertDLL berperan sebagai titik awal eksekusi program sekaligus sebagai demonstrasi penggunaan seluruh metode yang ada di kelas ini. Pada fungsi main, pertama-tama dibuat sebuah Doubly Linked List awal dengan urutan node 2 <-> 3 <-> 5. Setelah itu, list awal dicetak ke layar. Selanjutnya, dilakukan penambahan node 1 di awal list menggunakan insertBegin, dan hasilnya langsung dicetak. Berikutnya, node 6 ditambahkan di akhir list menggunakan insertEnd, dan hasilnya juga dicetak. Kemudian, node 4 disisipkan di posisi ke-4 menggunakan insertAtPosition, dan hasil akhirnya kembali dicetak. Setiap perubahan pada DLL langsung dicetak ke layar sehingga perubahan struktur list dapat diamati secara real time. Di akhir program, identitas praktikan juga dicetak sebagai penutup.

Alur Syntax (Langkah demi langkah sesuai kode):

- Membuat node head dengan data 2.
- Membuat node kedua (data 3), menghubungkan prev-nya ke head.
- Membuat node ketiga (data 5), menghubungkan prev-nya ke node kedua.
- Cetak DLL awal dengan printList(head).
- Tambah 1 di awal dengan insertBegin, update head, cetak hasil.
- Tambah 6 di akhir dengan insertEnd, cetak hasil.
- Tambah 4 di posisi ke-4 dengan insertAtPosition, cetak hasil.
- Cetak identitas praktikan.

Output:

```
<terminated> InsertDLL [Java Application] C:\Users\SAMSUNG\.p2\pool\plugins\org.eclipse.justj.ope
DLL Awal: 2 <-> 3 <-> 5 <->
Tambah 1 di awal: -> simpul 1 ditambah di awal: 1 <-> 2 <-> 3 <-> 5 <->
Tambah 6 di akhir: 1 <-> 2 <-> 3 <-> 5 <-> 6 <->
Tambah node 4 di posisi 4: 1 <-> 2 <-> 3 <-> 5 <-> 4 <-> 6 <->

Nama : Saskia Alifah
Nim  : 2411531002
```

Penjelasan Output:

- **"DLL Awal: 2 <-> 3 <-> 5 <->"**
  Output ini menunjukkan kondisi awal list yang terdiri dari tiga node. Node 2 adalah

head, node 3 di tengah, dan node 5 adalah tail. Pointer next dan prev sudah diatur sehingga traversal dari head ke tail akan mencetak urutan ini.

- **"Tambah 1 di awal: -> simpul 1 ditambah di awal: 1 <-> 2 <-> 3 <-> 5 <->"**
  Setelah operasi penambahan node 1 di awal, node 1 menjadi head baru.
  Pointer next node 1 menunjuk ke node 2, dan pointer prev node 2 menunjuk ke node 1. Traversal dari head ke tail kini menampilkan urutan 1, 2, 3, 5.
- **"Tambah 6 di akhir: 1 <-> 2 <-> 3 <-> 5 <-> 6 <->"**
  Node 6 ditambahkan di akhir list. Node 5 yang sebelumnya menjadi tail kini menunjuk ke node 6, dan node 6 menunjuk kembali ke node 5 melalui pointer prev. Urutan list kini menjadi 1, 2, 3, 5, 6.
- **"Tambah node 4 di posisi 4: 1 <-> 2 <-> 3 <-> 5 <-> 4 <-> 6 <->"**
  Node 4 disisipkan di posisi ke-4 (setelah node 5). Node 5 kini menunjuk ke node 4, dan node 4 menunjuk ke node 6. Pointer prev node 6 diatur ke node 4, dan pointer prev node 4 ke node 5. Ini membuktikan operasi insert di posisi tertentu berjalan lancar.
- **Identitas Praktikan**
  Dicetak pada akhir program sebagai tanda kepemilikan hasil praktikum.

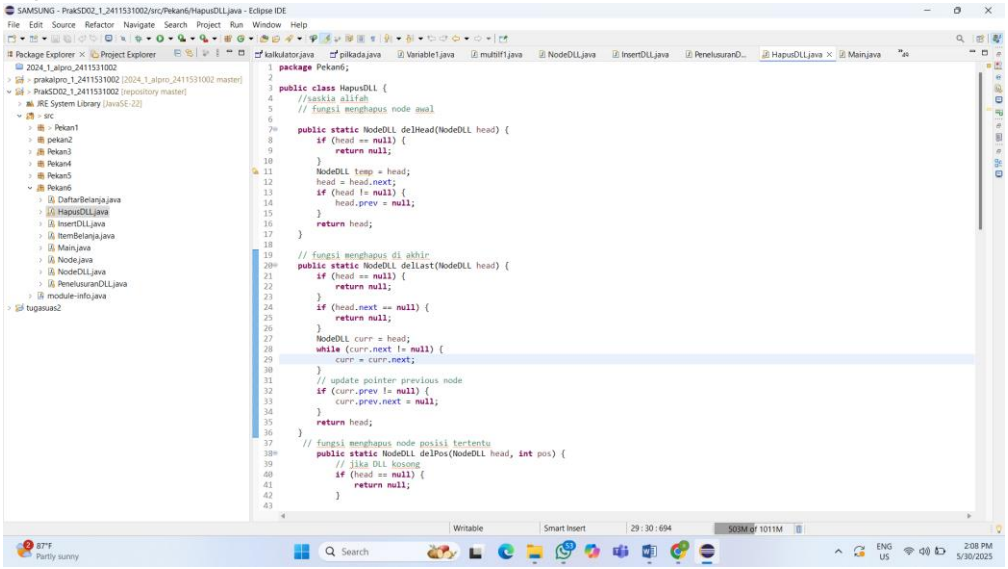
3. Class HapusDLL

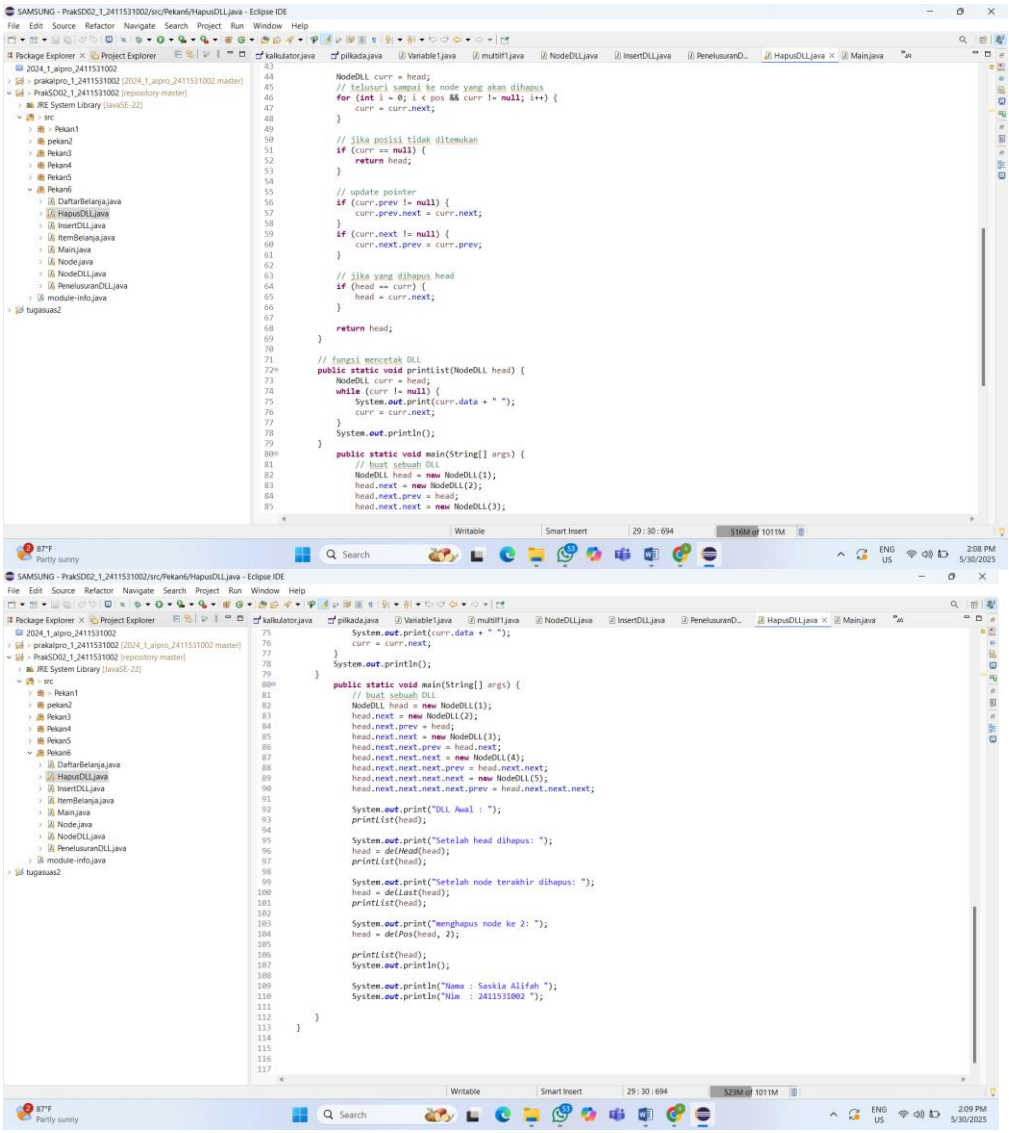
SYNTAX:

```
package Pekan6;

public class HapusDLL {
        //saskia alifah
   // 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;
   }

   // fungsi menghapus di akhir
   public static NodeDLL delLast(NodeDLL head) {
      if (head == null) {
         return null;
      }
      if (head.next == null) {
         return null;
      }
      NodeDLL curr = head;
      while (curr.next != null) {
         curr = curr.next;
      }
      // update pointer previous node
      if (curr.prev != null) {
```

```
            curr.prev.next = null;
        }
        return head;
    }
    // fungsi menghapus node posisi tertentu
    public static NodeDLL delPos(NodeDLL head, int pos) {
        // jika DLL kosong
        if (head == null) {
            return null;
        }

        NodeDLL curr = head;
        // telusuri sampai ke node yang akan dihapus
        for (int i = 0; i < pos && curr != null; i++) {
            curr = curr.next;
        }

        // jika posisi tidak ditemukan
        if (curr == null) {
            return head;
        }

        // update pointer
        if (curr.prev != null) {
            curr.prev.next = curr.next;
        }
        if (curr.next != null) {
            curr.next.prev = curr.prev;
        }

        // jika yang dihapus head
        if (head == curr) {
            head = curr.next;
        }

        return head;
    }

    // fungsi mencetak DLL
    public static void printList(NodeDLL head) {
        NodeDLL curr = head;
        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.next;
        }
        System.out.println();
    }
        public static void main(String[] args) {
            // buat sebuah DLL
            NodeDLL head = new NodeDLL(1);
            head.next = new NodeDLL(2);
            head.next.prev = head;
            head.next.next = new NodeDLL(3);
            head.next.next.prev = head.next;
            head.next.next.next = new NodeDLL(4);
```

```
        head.next.next.next.prev = head.next.next;
        head.next.next.next.next = new NodeDLL(5);
        head.next.next.next.next.prev = head.next.next.next;

        System.out.print("DLL Awal : ");
        printList(head);

        System.out.print("Setelah head dihapus: ");
        head = delHead(head);
        printList(head);

        System.out.print("Setelah node terakhir dihapus: ");
        head = delLast(head);
        printList(head);

        System.out.print("menghapus node ke 2: ");
        head = delPos(head, 2);

        printList(head);
        System.out.println();

                System.out.println("Nama : Saskia Alifah ");
                System.out.println("Nim  : 2411531002 ");

    }
}
```





Deskripsi Kelas:

Kelas HapusDLL bertanggung jawab untuk seluruh operasi penghapusan node pada DLL. Kelas ini menyediakan metode untuk menghapus node di awal (delHead), di akhir (delLast), dan di posisi tertentu (delPos). Terdapat juga metode printList untuk menampilkan isi DLL setelah setiap operasi.

Penjelasan Fungsi:

- delHead(NodeDLL head): Menghapus node pertama dari list. Jika list kosong, mengembalikan null. Jika tidak, head diupdate ke node berikutnya, dan pointer prev pada head baru diatur ke null.
- delLast(NodeDLL head): Menghapus node terakhir. Jika list hanya berisi satu node, mengembalikan null. Jika lebih, menelusuri hingga node terakhir, lalu memutuskan hubungan node terakhir dengan node sebelumnya.
- delPos(NodeDLL head, int pos): Menghapus node pada posisi tertentu. Jika posisi tidak ditemukan, list tidak berubah. Jika node yang dihapus adalah head, head diupdate ke node berikutnya. Jika node di tengah, node sebelum dan sesudah node yang dihapus dihubungkan langsung.
- printList(NodeDLL head): Menelusuri dan mencetak seluruh node dalam list.

Error Handling:

Pada setiap fungsi, pengecekan null dilakukan untuk menghindari operasi pada list kosong. Pada delPos, jika posisi tidak ditemukan, operasi tidak dilakukan. Penanganan kasus list satu node juga dilakukan agar pointer tidak menjadi invalid.

Penjelasan Fungsi Main:

Fungsi main pada kelas HapusDLL mendemonstrasikan operasi penghapusan pada DLL. Pertama, fungsi membuat dan menghubungkan node membentuk list awal dengan urutan 1 2 3 4 5. Fungsi kemudian menampilkan DLL awal dengan printList. Selanjutnya, fungsi menghapus head dengan delHead dan menampilkan hasilnya. Fungsi kemudian menghapus node terakhir dengan delLast dan kembali menampilkan hasilnya. Setelah itu, fungsi menghapus node di posisi ke-2 menggunakan delPos dan hasilnya juga ditampilkan. Di akhir program, identitas praktikan dicetak sebagai penutup.

Alur Syntax (Langkah demi langkah sesuai kode):

- Membuat node head dengan data 1, lalu menambah node 2, 3, 4, dan 5, menghubungkan prev dan next antar node.
- Cetak DLL awal.
- Hapus head dengan delHead, update head, cetak hasil.
- Hapus node terakhir dengan delLast, cetak hasil.
- Hapus node ke-2 dengan delPos, cetak hasil.
- Cetak identitas praktikan.

Output:

```
<terminated> HapusDLL [Java Application] C:\Users\
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 3

Nama : Saskia Alifah
Nim  : 2411531002
```

Penjelasan Output:

- **"DLL Awal : 1 2 3 4 5"**
  Menampilkan seluruh isi list sebelum operasi penghapusan dilakukan. List berisi lima node yang terhubung secara berurutan.
- **"Setelah head dihapus: 2 3 4 5"**
  Node pertama (head) dihapus. Node 2 kini menjadi head baru, pointer prev node 2 diatur ke null. Node 1 sudah tidak terhubung ke list.
- **"Setelah node terakhir dihapus: 2 3 4"**
  Node terakhir (5) dihapus. Tail baru adalah node 4, pointer next node 4 menjadi null. Node 5 sudah tidak terhubung ke list.
- **"menghapus node ke 2: 2 3"**
  Node di posisi ke-2 (yaitu node 4) dihapus. Node 3 menjadi tail, pointer next node 2 menunjuk ke node 3, dan pointer prev node 3 menunjuk ke node 2. List kini hanya terdiri dari dua node.
- **Identitas Praktikan**
  Dicetak pada akhir program sebagai tanda kepemilikan hasil praktikum.

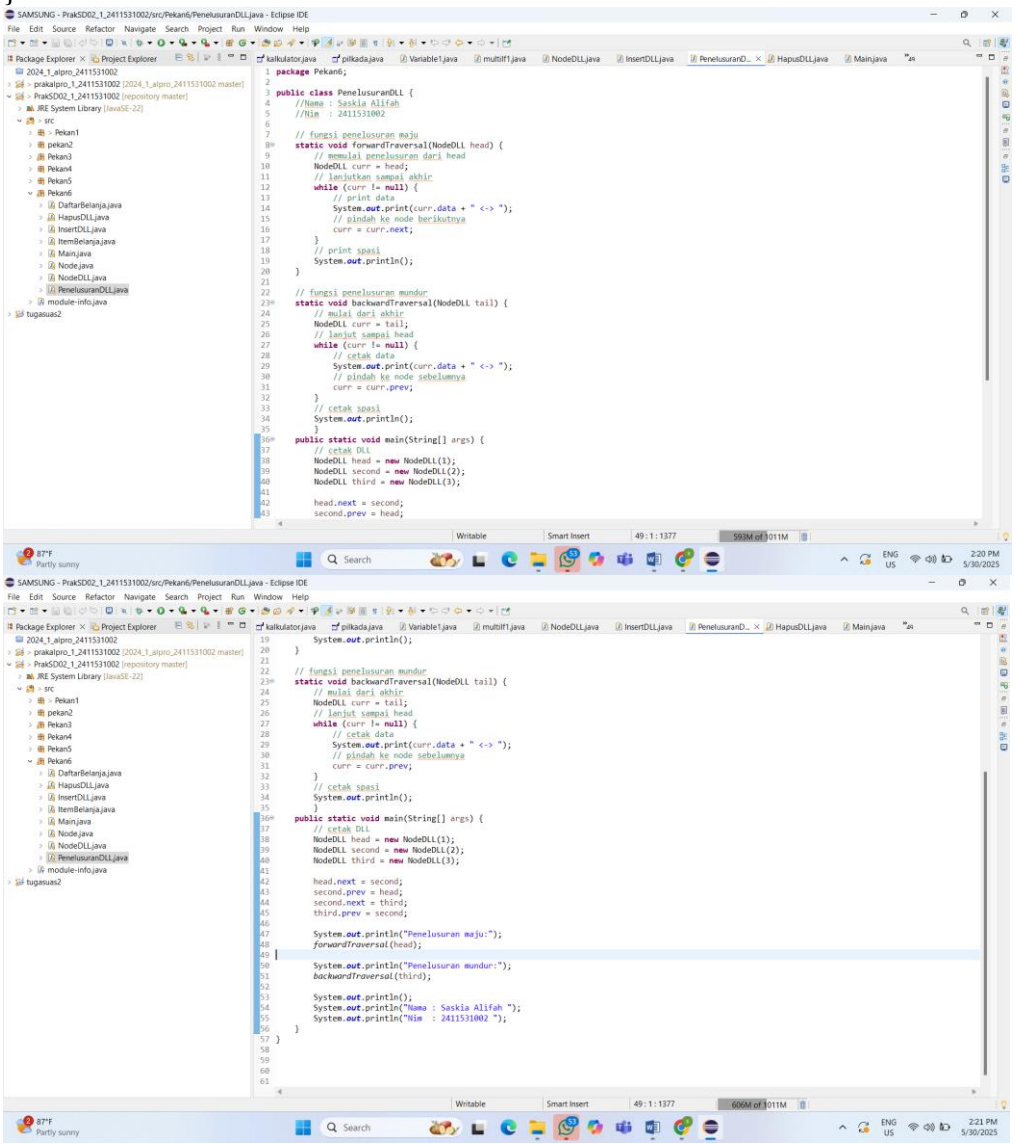
4. Class PenelusuranDLL

SYNTAX:

```
package Pekan6;
```

```
public class PenelusuranDLL {
          //Nama : Saskia Alifah
          //Nim  : 2411531002

    // fungsi penelusuran maju
    static void forwardTraversal(NodeDLL head) {
        // memulai penelusuran dari head
        NodeDLL curr = head;
        // lanjutkan sampai akhir
        while (curr != null) {
            // print data
            System.out.print(curr.data + " <-> ");
            // pindah ke node berikutnya
            curr = curr.next;
        }
        // print spasi
        System.out.println();
    }

    // fungsi penelusuran mundur
    static void backwardTraversal(NodeDLL tail) {
        // mulai dari akhir
        NodeDLL curr = tail;
        // lanjut sampai head
        while (curr != null) {
            // cetak data
            System.out.print(curr.data + " <-> ");
            // pindah ke node sebelumnya
            curr = curr.prev;
        }
        // cetak spasi
        System.out.println();
              }
    public static void main(String[] args) {
        // cetak DLL
        NodeDLL head = new NodeDLL(1);
        NodeDLL second = new NodeDLL(2);
        NodeDLL third = new NodeDLL(3);

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

        System.out.println("Penelusuran maju:");
        forwardTraversal(head);

        System.out.println("Penelusuran mundur:");
        backwardTraversal(third);

        System.out.println();
        System.out.println("Nama : Saskia Alifah ");
        System.out.println("Nim  : 2411531002 ");
    }
```

}



Deskripsi Kelas:

Kelas PenelusuranDLL bertugas untuk melakukan traversal pada DLL, baik secara maju (dari head ke tail) maupun mundur (dari tail ke head). Fungsi ini sangat penting untuk menampilkan isi list dan memastikan pointer next dan prev sudah terhubung dengan benar.

Penjelasan Fungsi:

- forwardTraversal(NodeDLL head): Menelusuri list dari head ke tail dengan mengikuti pointer next, mencetak data setiap node.
- backwardTraversal(NodeDLL tail): Menelusuri list dari tail ke head dengan mengikuti pointer prev, mencetak data setiap node.

Error Handling:

Pada kedua fungsi, dilakukan pengecekan apakah node awal (head atau tail) adalah null. Jika ya, traversal tidak dilakukan sehingga menghindari error runtime.

Penjelasan Fungsi Main:

Fungsi main pada kelas PenelusuranDLL mendemonstrasikan operasi traversal pada DLL. Fungsi membuat node head (1), node kedua (2), dan node ketiga (3), menghubungkan

next dan prev antar node. Kemudian memanggil forwardTraversal untuk menampilkan isi list dari head ke tail, dan backwardTraversal untuk menampilkan isi list dari tail ke head. Di akhir program, identitas praktikan dicetak sebagai penutup.

Alur Syntax (Langkah demi langkah sesuai kode):

- Membuat node head (1), node kedua (2), dan node ketiga (3).
- Menghubungkan next dan prev antar node.
- Penelusuran maju dengan forwardTraversal.
- Penelusuran mundur dengan backwardTraversal.
- Cetak identitas praktikan.

Output:

```
<terminated> PenelusuranDLL [Java
Penelusuran maju:
1 <-> 2 <-> 3 <->
Penelusuran mundur:
3 <-> 2 <-> 1 <->

Nama : Saskia Alifah
Nim  : 2411531002
```

Penjelasan Output:

- **"Penelusuran maju: 1 <-> 2 <-> 3 <->"**
  Fungsi forwardTraversal menampilkan isi list dari head ke tail. Output ini membuktikan bahwa pointer next setiap node sudah terhubung dengan benar, sehingga traversal dari depan ke belakang berjalan lancar.
- **"Penelusuran mundur: 3 <-> 2 <-> 1 <->"**
  Fungsi backwardTraversal menampilkan isi list dari tail ke head. Output ini membuktikan bahwa pointer prev setiap node sudah terhubung dengan benar, sehingga traversal dari belakang ke depan juga berjalan lancar.
- **Identitas Praktikan**
  Dicetak pada akhir program sebagai tanda kepemilikan hasil praktikum.

D. Kesimpulan Praktikum

Praktikum Doubly Linked List ini membuktikan bahwa DLL adalah struktur data yang sangat fleksibel dan efisien untuk operasi dinamis seperti penambahan, penghapusan, dan penelusuran data dari dua arah. Dengan mengimplementasikan berbagai metode pada kelas-kelas yang berbeda, mahasiswa dapat memahami bagaimana setiap operasi memanipulasi pointer prev dan next untuk menjaga integritas struktur list. Praktikum ini juga menegaskan pentingnya pemahaman alur data dan pointer dalam pengembangan aplikasi yang membutuhkan pengelolaan data secara efisien dan dinamis. Selain itu, error handling yang baik sangat penting untuk mencegah kesalahan fatal seperti dereferensi pointer null atau kehilangan node. Dengan penguasaan DLL, mahasiswa siap menghadapi tantangan pengelolaan data yang lebih kompleks di masa depan.