LAPORAN PRAKTIKUM (6) STRUKTUR DATA

Sebuah gambar berisi lambang, logo, makanan, ilustrasi

Konten yang dihasilkan AI mungkin salah.

Oleh:

Nama : Arkan Ubaidillah Warman

NIM : 2411537001

Dosen Pengampu : Dr. Wahyudi MT .

STRUKTUR DATA

(DLL)

DEPARTEMEN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERITAS ANDALAS

PADANG

1. **PENDAHULUAN**
2. **Latar Belakang**

Double Linked List (Daftar Berantai Ganda) adalah salah satu struktur data fundamental yang digunakan untuk menyimpan dan mengelola kumpulan data secara dinamis. Berbeda dengan Single Linked List, Double Linked List memiliki dua pointer (prev dan next) pada setiap simpul (node), sehingga memungkinkan traversing maju dan mundur dengan efisien.

Struktur ini sangat berguna dalam aplikasi yang memerlukan operasi penyisipan (insertion) dan penghapusan (deletion) yang cepat, seperti manajemen memori, implementasi undo-redo, atau navigasi data dua arah. Percobaan ini dilakukan untuk memahami konsep dasar, operasi-operasi utama, serta kelebihan dan kekurangan Double Linked List dibandingkan struktur data lainnya.

1. **Tujuan Percobaan**
   1. Memahami konsep dan implementasi Double Linked List dalam pemrograman.
   2. Menganalisis perbedaan antara Single Linked List dan Double Linked List.
   3. Mengimplementasikan operasi dasar Double Linked List, seperti:

* Penyisipan node (insertion) di awal, tengah, dan akhir.
* Penghapusan node (deletion) berdasarkan nilai atau posisi.
* Traversing (maju dan mundur).
  1. Menilai kelebihan dan kekurangan Double Linked List dalam penggunaan memori dan kompleksitas waktu.

1. **Landasan Teori**

**Definisi Double Linked List**

Double Linked List adalah struktur data linier yang terdiri dari sekumpulan node, di mana setiap node memiliki:

* **Data**: Nilai yang disimpan.
* **Pointer next**: Alamat node berikutnya.
* **Pointer prev**: Alamat node sebelumnya

.

**Operasi Dasar**

1. **Insertion**:

* **Insert at Head**: Menambahkan node di awal list.
* **Insert at Tail**: Menambahkan node di akhir list.
* **Insert at Position**: Menyisipkan node di posisi tertentu.

1. **Deletion**:Menghapus node berdasarkan nilai atau posisi.
2. **Traversal**:Maju (menggunakan next) dan mundur (menggunakan prev).

**Kompleksitas Waktu**

* **Insertion/Deletion di Head/Tail**: O(1).
* **Insertion/Deletion di Posisi Tertentu**: O(n).
* **Searching**: O(n).

**Kelebihan**

* Traversal dua arah lebih fleksibel.
* Operasi penghapusan lebih efisien karena tidak perlu traversal dari head untuk mencari node sebelumnya (seperti pada Single Linked List).

**Kekurangan**

* Membutuhkan memori lebih besar karena penyimpanan pointer tambahan (prev).

1. **LANGKAH- LANGKAH**
2. NodeDLL.java

Sebuah gambar berisi teks, cuplikan layar, software, tampilan

Konten yang dihasilkan AI mungkin salah.

Penjelasan:

1. data: Menyimpan nilai (contoh: 10, 20, dll).
2. next: Menunjuk ke node berikutnya. Jika null, artinya ini node terakhir.
3. prev: Menunjuk ke node sebelumnya. Jika null, artinya ini node pertama.
4. Konstruktor:Saat membuat node baru, next dan prev di-set null karena node belum terhubung ke list.
5. InsertDLL.java

Sebuah gambar berisi teks, cuplikan layar, software, tampilan

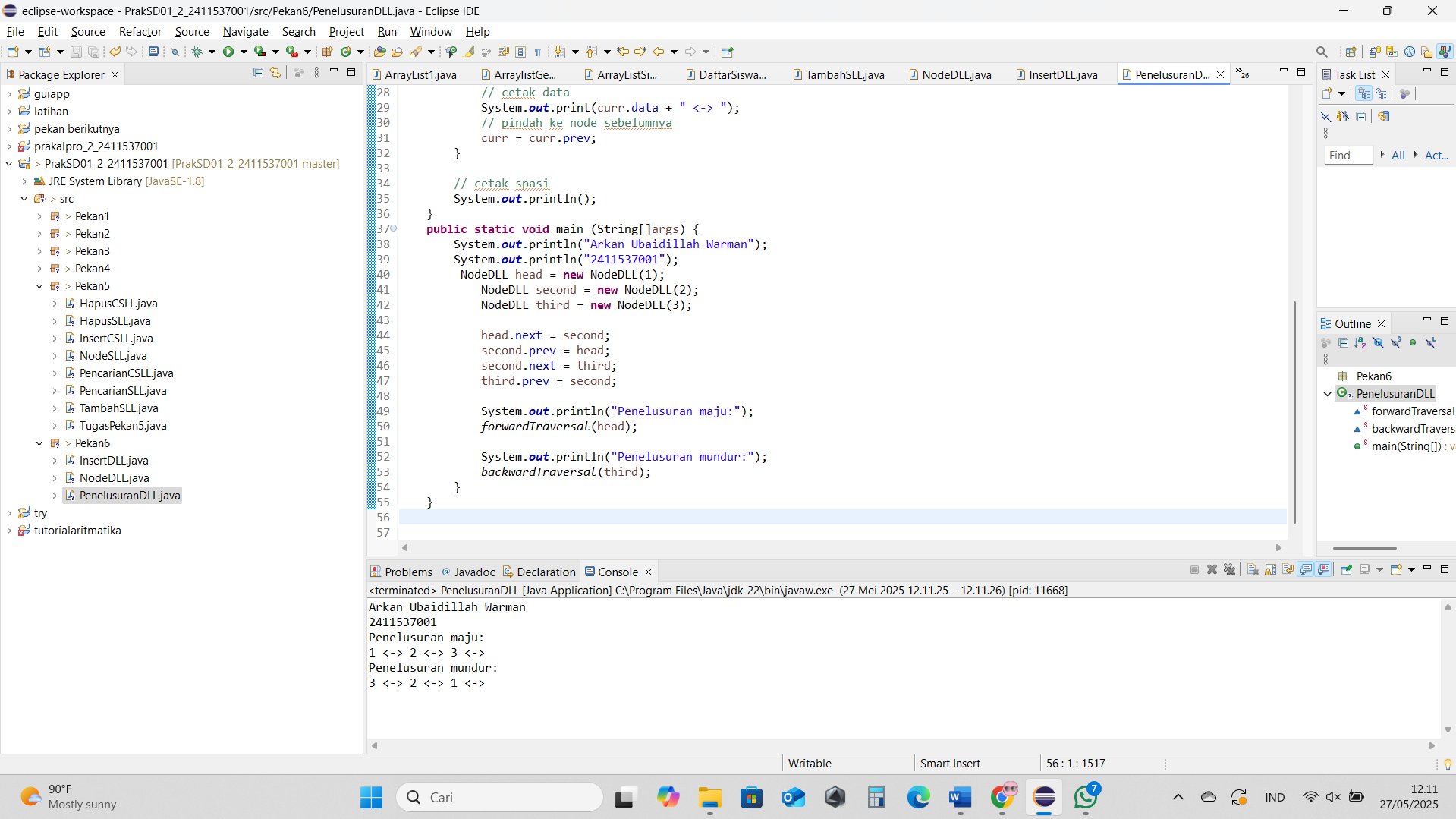
Konten yang dihasilkan AI mungkin salah.

Penjelasan:Kode ini menunjukkan **3 cara menyisipkan node** dalam DLL:

* **Awal**: insertBegin.
* **Akhir**: insertEnd.
* **Posisi Tertentu**: insertAtPosition.

.

1. PenelusuranDLL.java

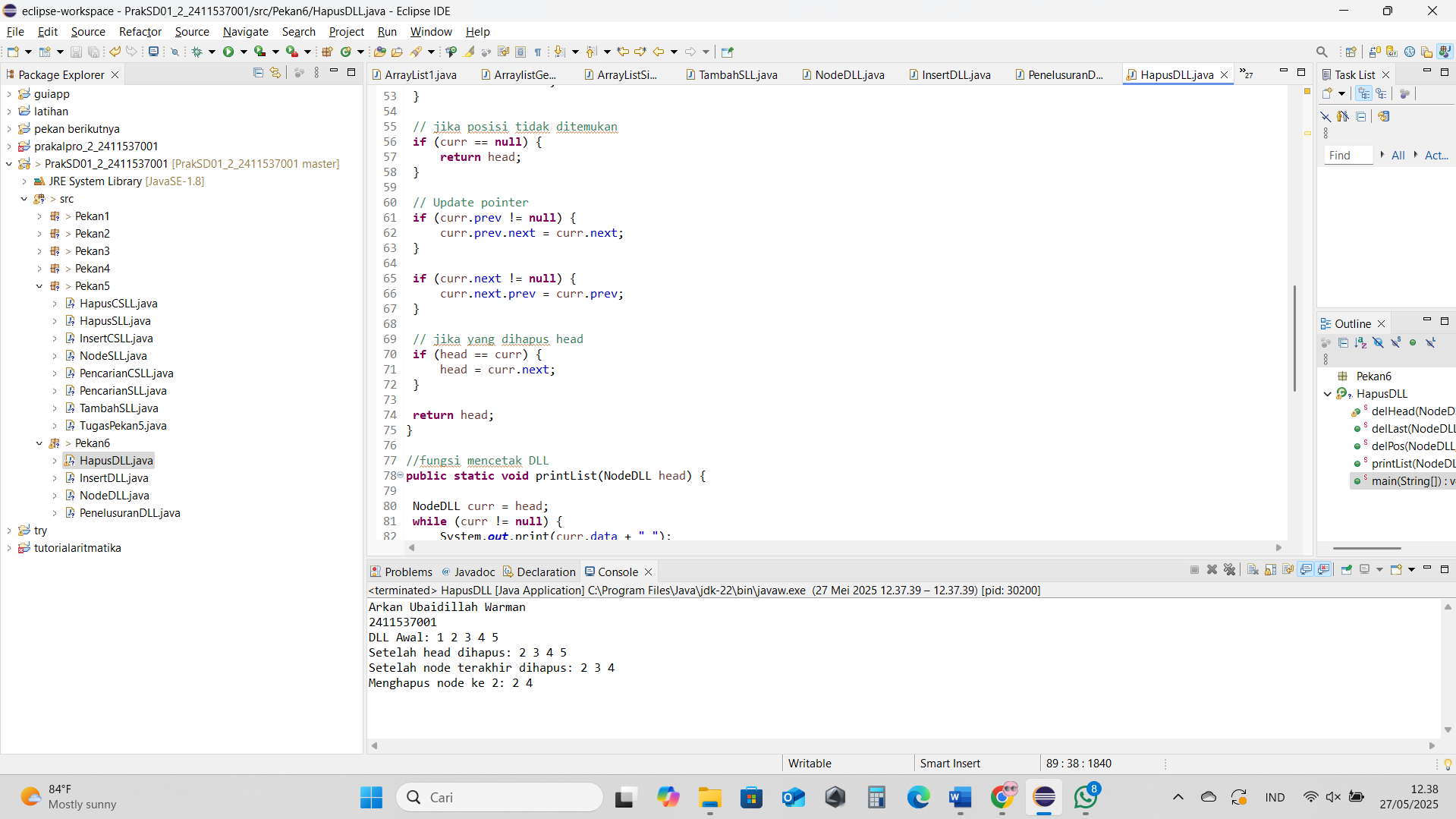


Penjelasan:

* forwardTraversal(): Menelusuri DLL dari **head ke tail** (maju) menggunakan pointer next.
* backwardTraversal(): Menelusuri DLL dari **tail ke head** (mundur) menggunakan pointer prev.
* head menunjuk node 1.
* tail (implisit) menunjuk node 3.

.

1. HapusDLL.java



Penjelasan:

1. **delHead()**

* Menghapus node **pertama** (head)
* Cukup pindahkan head ke node berikutnya dan set head.prev = null
* Kompleksitas: O(1)

1. **delLast()**

* Menghapus node **terakhir**
* Traverse sampai node terakhir, lalu putuskan link dari node sebelumnya
* Kompleksitas: O(n)

1. **delPos()**

* Menghapus node di **posisi tertentu**
* Traverse ke posisi target, lalu hubungkan node sebelum dan sesudahnya
* Kompleksitas: O(n)

1. **KESIMPULAN**
2. Double Linked List memungkinkan manipulasi data lebih fleksibel dibandingkan Single Linked List karena dukungan traversing dua arah.
3. Operasi penyisipan dan penghapusan di awal atau akhir list memiliki kompleksitas waktu O(1), tetapi operasi di posisi tertentu memerlukan O(n).
4. Penggunaan memori lebih besar karena setiap node menyimpan dua pointer.
5. Cocok untuk aplikasi yang memerlukan navigasi maju-mundur (contoh: browser history, playlist musik).

Dengan memahami Double Linked List, pengembang dapat memilih struktur data yang optimal sesuai kebutuhan efisiensi dan fungsionalitas.