LAPORAN PRAKTIKUM

STRUKTUR DATA



DISUSUN OLEH :

Sasya Zamora

2411533014

DOSEN PENGAMPU :

Dr. Wahyudi, S.T.M.T

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

2024

1. TUJUAN

1. Mengimplementasikan struktur data queue secara manual menggunakan array

2. Menampilkan dan mengakses elemen queue menggunakan iterator

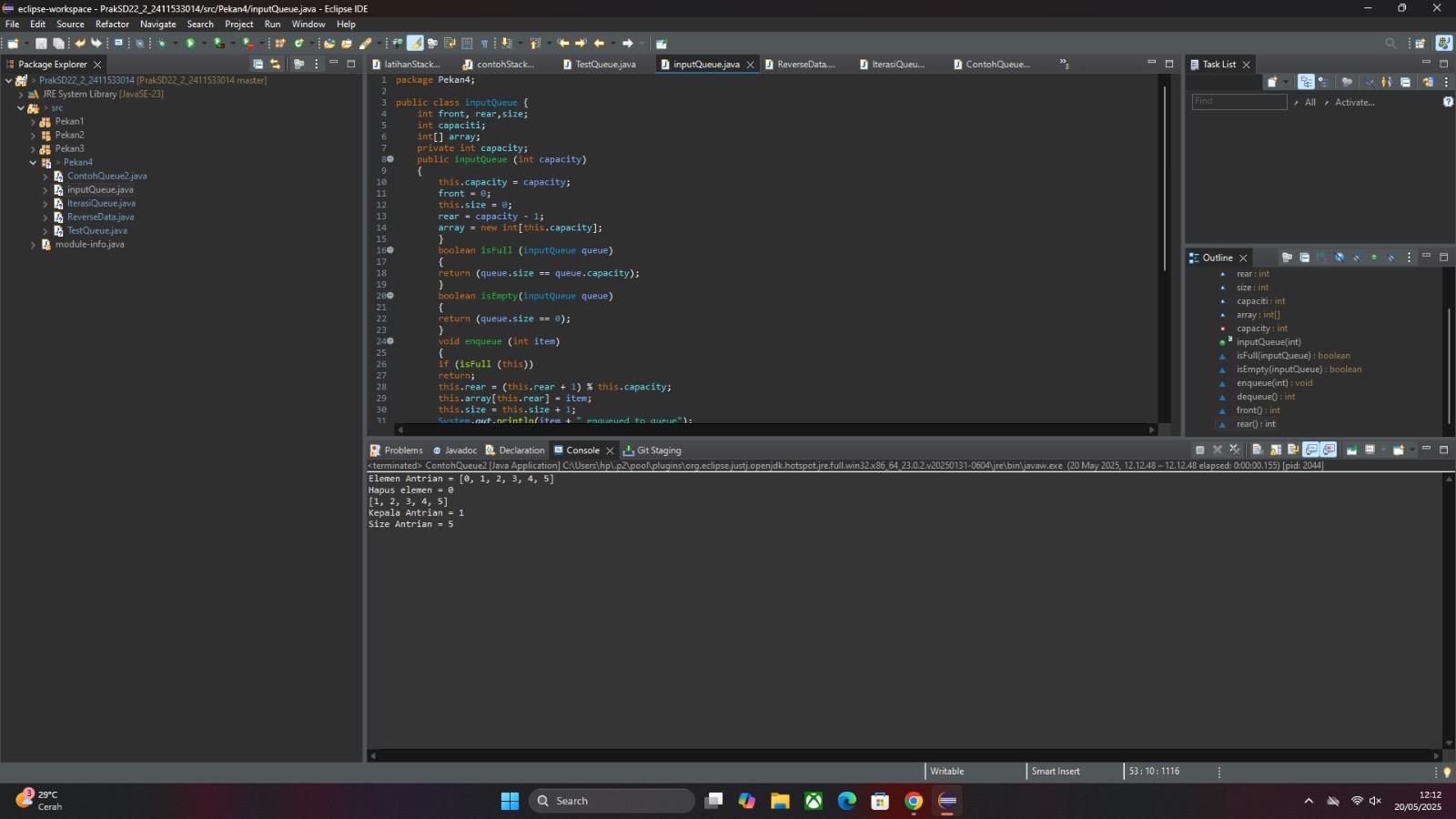
3. Mengaplikasikan operasi dasar queue seperti add(), remove(), peek(), dan size()

4. Menggunakan stack untuk membalik urutan elemen dalam queue

1. PEMBAHASAN

Praktikum ini bertujuan untuk memahami dan mengimplementasikan struktur data antrian (Queue) serta menguji fungsionalitasnya. Antrian merupakan struktur data linear yang mengikuti prinsip FIFO (First-In, First-Out), di mana elemen pertama yang masuk adalah elemen pertama yang keluar. Implementasi dan pengujian difokuskan pada operasi-operasi dasar Queue, yaitu enqueue (menambahkan elemen), dequeue (menghapus elemen), peek (melihat elemen terdepan tanpa menghapus), dan isEmpty (mengecek apakah antrian kosong). Implementasi dilakukan menggunakan array untuk struktur data Queue. Bahasa pemrograman yang digunakan adalah java.

1. LANGKAH PRAKTIKUM
2. InputQueue



1. Deklarasi Kelas InputQueue:

public class InputQueue {

int front, rear, size;

int capacity;

private int[] array;

• front: Menunjukkan indeks elemen terdepan (depan) dari antrian dalam array.

• rear: Menunjukkan indeks elemen terakhir (belakang) dari antrian dalam array.

• size: Menunjukkan jumlah elemen saat ini dalam antrian.

• capacity: Menunjukkan kapasitas maksimum antrian (ukuran array).

• array: Array integer yang digunakan untuk menyimpan elemen-elemen antrian.

2. Konstruktor InputQueue:

public InputQueue(int capacity) {

this.capacity = capacity;

this.front = this.size = 0;

this.rear = capacity - 1; // Inisialisasi rear untuk circular buffer

array = new int[this.capacity];

}

• Konstruktor menginisialisasi antrian dengan kapasitas yang ditentukan.

• front diinisialisasi menjadi 0.

• size diinisialisasi menjadi 0 (antrian kosong).

• rear diinisialisasi menjadi capacity - 1 untuk mendukung implementasi circular buffer. Ini penting agar saat rear mencapai akhir array, ia dapat "membungkus" kembali ke awal array.

• Array integer array dialokasikan dengan ukuran capacity.

3. Metode isFull:

boolean isFull(InputQueue queue) {

return (queue.size == queue.capacity);

}

• Mengembalikan true jika antrian penuh (ukuran antrian sama dengan kapasitasnya), false jika tidak.

4. Metode isEmpty:

boolean isEmpty(InputQueue queue) {

return (queue.size == 0);

}

• Mengembalikan true jika antrian kosong (ukuran antrian adalah 0), false jika tidak.

5. Metode enqueue:

void enqueue(int item) {

if (isFull(this))

return;

this.rear = (this.rear + 1) % this.capacity;

this.array[this.rear] = item;

this.size = this.size + 1;

System.out.println(item + " enqueued to queue");

}

• Menambahkan elemen (item) ke antrian.

• Memeriksa apakah antrian sudah penuh menggunakan isFull. Jika penuh, tidak ada tindakan yang dilakukan.

• this.rear = (this.rear + 1) % this.capacity; Ini adalah inti dari circular buffer. Operasi modulo (%) memastikan bahwa rear akan "membungkus" kembali ke indeks 0 ketika mencapai indeks terakhir array.

• Elemen ditambahkan ke indeks rear dalam array.

• Ukuran antrian (size) diinkremen.

6. Metode dequeue:

void dequeue() {

if (isEmpty(this))

return;

int item = this.array[this.front];

this.front = (this.front + 1) % this.capacity;

this.size = this.size - 1;

System.out.println(item + " dequeued from queue");

}

• Menghapus dan mengembalikan elemen terdepan dari antrian.

• Memeriksa apakah antrian kosong menggunakan isEmpty. Jika kosong, tidak ada tindakan yang dilakukan.

• Elemen dari indeks front diambil dan disimpan dalam variabel item.

• this.front = (this.front + 1) % this.capacity; Mirip dengan enqueue, operasi modulo digunakan untuk implementasi circular buffer.

• Ukuran antrian (size) didekremen.

7. Metode front dan rear:

int front() {

if (isEmpty(this))

return Integer.MIN\_VALUE; //atau nilai lain yang menandakan antrian kosong

return this.array[this.front];

}

int rear() {

if (isEmpty(this))

return Integer.MIN\_VALUE; //atau nilai lain yang menandakan antrian kosong

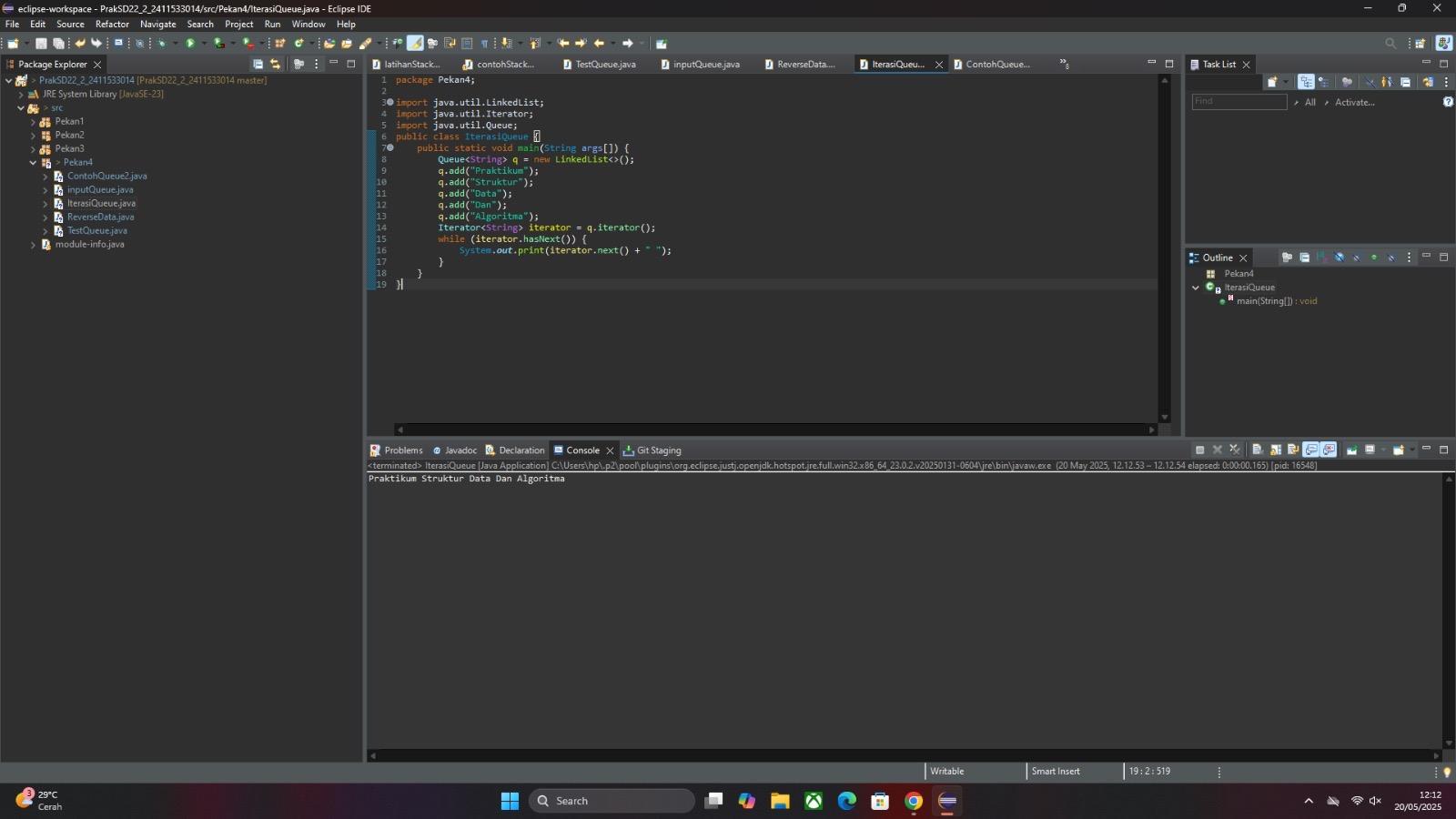
return this.array[this.rear];

}

• Metode front mengembalikan elemen terdepan tanpa menghapusnya.

• Metode rear mengembalikan elemen terakhir tanpa menghapusnya. Keduanya mengembalikan nilai khusus jika antrian kosong.

1. IterasiQueue



1. Mengimpor Kelas yang Diperlukan

import java.util.LinkedList;

import java.util.Iterator;

import java.util.Queue;

Bagian ini mengimpor tiga kelas dari paket java.util:

• LinkedList: Digunakan untuk mengimplementasikan antrian. LinkedList adalah implementasi daftar tertaut ganda yang juga dapat berfungsi sebagai antrian.

• Iterator: Digunakan untuk melintasi elemen-elemen antrian. Iterator menyediakan cara standar untuk menelusuri koleksi.

• Queue: Antarmuka yang mendefinisikan struktur data antrian.

2. Mendefinisikan Kelas IterasiQueue dan Metode main

public class IterasiQueue {

public static void main(String args[]) {

// Kode untuk memproses antrian akan diletakkan di sini

}

}

Ini mendefinisikan kelas utama IterasiQueue. Metode main adalah titik masuk program.

3. Membuat Antrian dan Menambahkan Elemen

Queue<String> q = new LinkedList<>();

q.add("Praktikum");

q.add("Struktur");

q.add("Data");

q.add("Dan");

q.add("Algoritma");

4. Membuat Iterator

Iterator<String> iterator = q.iterator();

Sebuah iterator bernama iterator dibuat untuk melintasi antrian q. Metode iterator() dari antarmuka Queue mengembalikan objek iterator.

5. Mengiterasi dan Mencetak

while (iterator.hasNext()) {

System.out.print(iterator.next() + " ");

}

Loop while mengiterasi selama antrian masih memiliki elemen (iterator.hasNext() mengembalikan true). Di dalam loop:

• iterator.next() mengambil elemen berikutnya dari antrian.

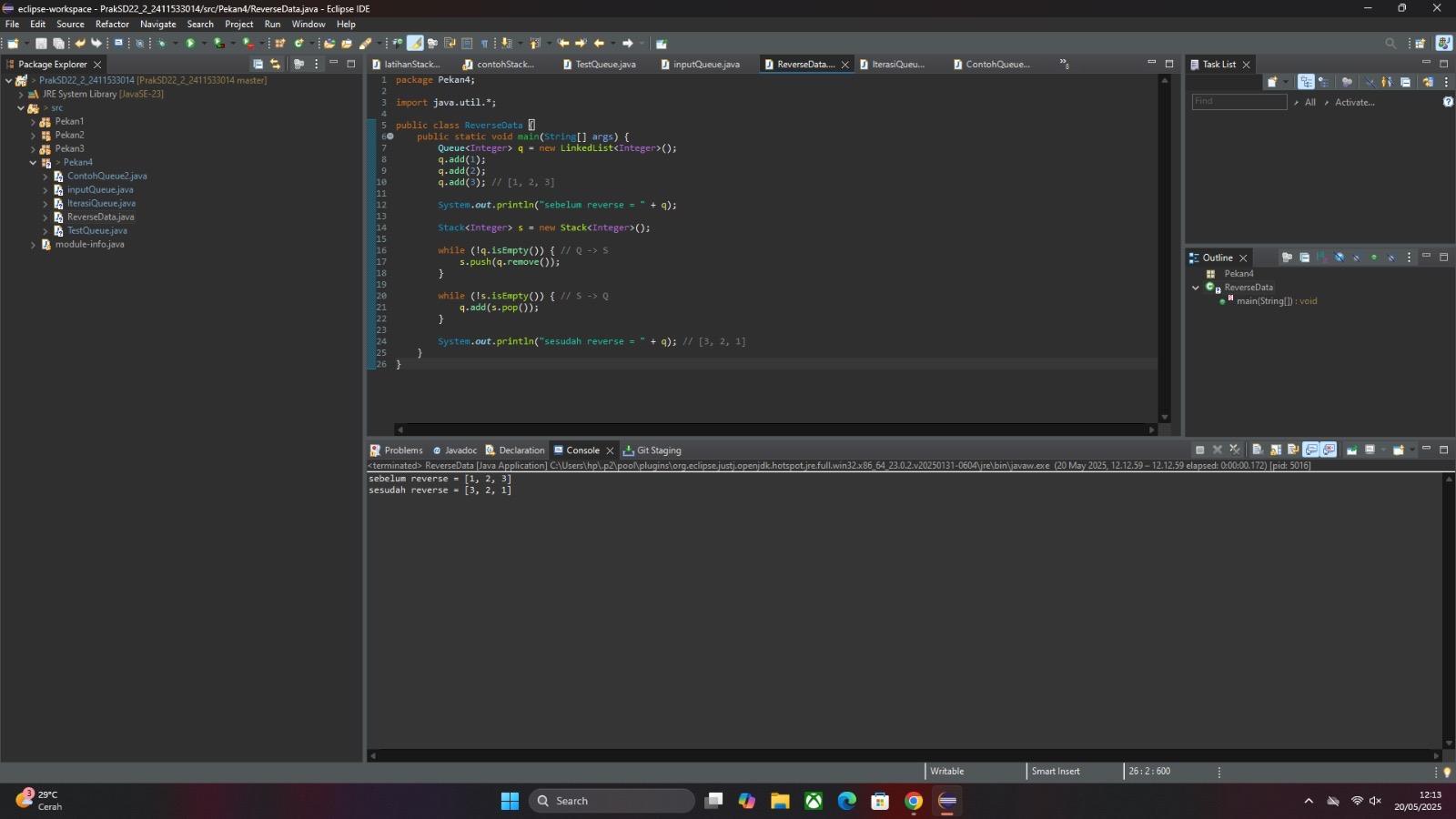
• System.out.print() mencetak elemen yang diambil diikuti dengan spasi.

Loop ini memproses antrian dari depan ke belakang, mencetak setiap elemen secara berurutan.

6. Program akan mencetak keluaran berikut:

Praktikum Struktur Data Dan Algoritma

Setiap kata dari antrian dicetak, dipisahkan oleh spasi. Program ini menunjukkan penggunaan Queue, LinkedList, dan Iterator untuk memproses kumpulan data dalam Java. Keluaran menunjukkan bahwa elemen diproses dengan cara First-In, First-Out (FIFO), yang merupakan perilaku karakteristik dari antrian.

1. ReverseData

1. Import Kelas yang Dibutuhkan

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

Kode ini mengimpor tiga kelas:

• LinkedList: Digunakan untuk membuat antrian (queue). LinkedList merupakan implementasi dari list yang juga bisa digunakan sebagai queue.

• Queue: Antarmuka yang mendefinisikan struktur data antrian.

• Stack: Struktur data tumpukan (stack), yang digunakan sebagai struktur data bantu untuk membalik urutan elemen.

2. Mendefinisikan Kelas ReverseData dan Metode main

public class ReverseData {

public static void main(String[] args) {

// ... (kode selanjutnya) ...

}

}

Ini mendefinisikan kelas utama ReverseData dan metode main sebagai titik awal eksekusi program.

3. Membuat Antrian dan Menambahkan Elemen

Queue<Integer> q = new LinkedList<>();

q.add(1);

q.add(2);

q.add(3);

System.out.println("sebelum reverse = " + q);

// Mencetak antrian sebelum dibalik

Sebuah antrian q yang menyimpan bilangan bulat (Integer) dibuat menggunakan LinkedList. Tiga elemen (1, 2, 3) ditambahkan ke antrian. Pernyataan System.out.println mencetak isi antrian sebelum pembalikan.

4. Membuat Stack dan Memindahkan Elemen dari Queue ke Stack

Stack<Integer> s = new Stack<>();

while (!q.isEmpty()) {

s.push(q.remove());

}

Sebuah stack s dibuat. Loop while berjalan selama antrian q tidak kosong. Di dalam loop:

• q.remove() mengambil dan menghapus elemen terdepan dari antrian.

• s.push() menambahkan elemen yang diambil ke bagian atas stack.

Setelah loop selesai, semua elemen dari antrian telah dipindahkan ke stack, dengan urutan terbalik.

5. Memindahkan Elemen dari Stack Kembali ke Queue

while (!s.isEmpty()) {

q.add(s.pop());

}

Loop while ini berjalan selama stack s tidak kosong. Di dalam loop:

• s.pop() mengambil dan menghapus elemen teratas dari stack.

• q.add() menambahkan elemen yang diambil ke bagian belakang antrian.

Setelah loop ini, semua elemen dari stack telah dipindahkan kembali ke antrian, tetapi sekarang dalam urutan terbalik dari urutan asalnya.

6. Mencetak Antrian Setelah Pembalikan

System.out.println("sesudah reverse = " + q);

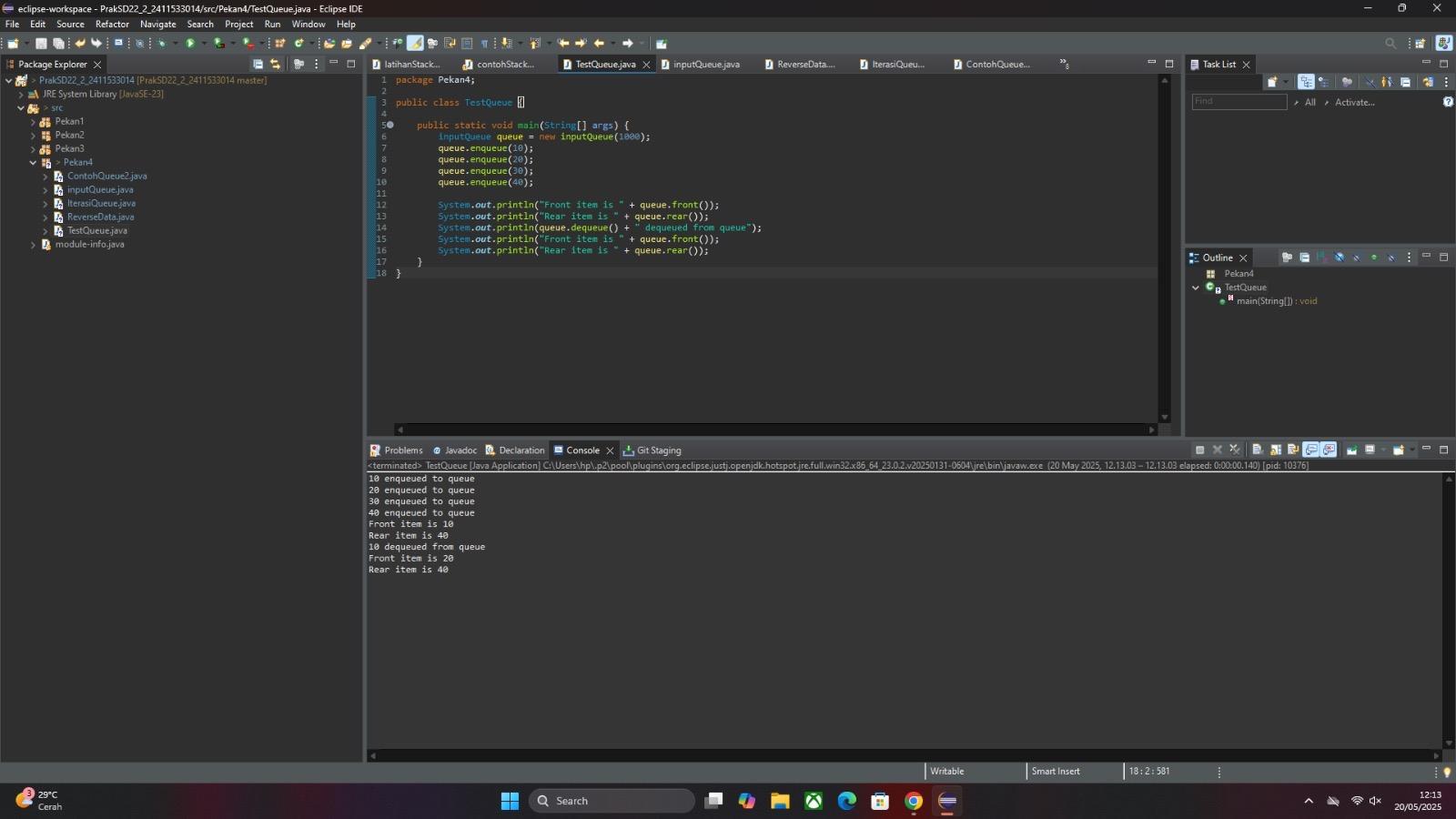
Pernyataan ini mencetak isi antrian q setelah pembalikan.

7. Program akan menghasilkan output:

sebelum reverse = [1, 2, 3]

sesudah reverse = [3, 2, 1]

Ini menunjukkan bahwa kode berhasil membalik urutan elemen dalam antrian. Algoritma ini memanfaatkan sifat LIFO (Last-In, First-Out) dari stack untuk mencapai pembalikan urutan elemen FIFO (First-In, First-Out) dalam queue.

1. TestQueue
2. Import Kelas yang Dibutuhkan

Tidak ada import yang diperlukan dalam kode ini karena diasumsikan InputQueue berada dalam paket yang sama.

1. Mendefinisikan Kelas TestQueue dan Metode main

public class TestQueue {

public static void main(String[] args) {

// ... (Kode selanjutnya) ...

}

}

Ini mendefinisikan kelas utama TestQueue dan metode main sebagai titik awal eksekusi.

1. Membuat Objek InputQueue dan Menambahkan Elemen

InputQueue queue = new InputQueue(1000); // Membuat antrian dengan kapasitas 1000

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.enqueue(40);

Sebuah objek InputQueue dibuat dengan kapasitas maksimum 1000 elemen. Empat elemen (10, 20, 30, 40) kemudian ditambahkan ke antrian menggunakan metode enqueue.

1. Mengambil dan Mencetak Elemen

queue.dequeue(); // Menghapus elemen terdepan (10)

System.out.println("Front item is = " + queue.front()); // Mencetak elemen terdepan

System.out.println("Rear item is = " + queue.rear()); // Mencetak elemen terakhir

Satu elemen dihapus dari antrian menggunakan dequeue. Kemudian, kode mencetak elemen terdepan (front) dan elemen terakhir (rear) dari antrian.

1. Menambahkan dan Mencetak Lagi

queue.enqueue(10); // Menambahkan elemen 10 kembali

System.out.println("Front item is = " + queue.front()); // Mencetak elemen terdepan

System.out.println("Rear item is = " + queue.rear()); // Mencetak elemen terakhir

Elemen 10 ditambahkan kembali ke antrian, dan elemen terdepan dan elemen terakhir dicetak lagi untuk menunjukkan perubahan.

1. Program akan menghasilkan output: Berdasarkan kode dan asumsi bahwa InputQueue berfungsi dengan benar, output yang diharapkan adalah:

20 enqueued to queue

30 enqueued to queue

40 enqueued to queue

10 enqueued to queue

Front item is 20

Rear item is 40

10 dequeued from queue

Front item is 20

Rear item is 40

10 enqueued to queue

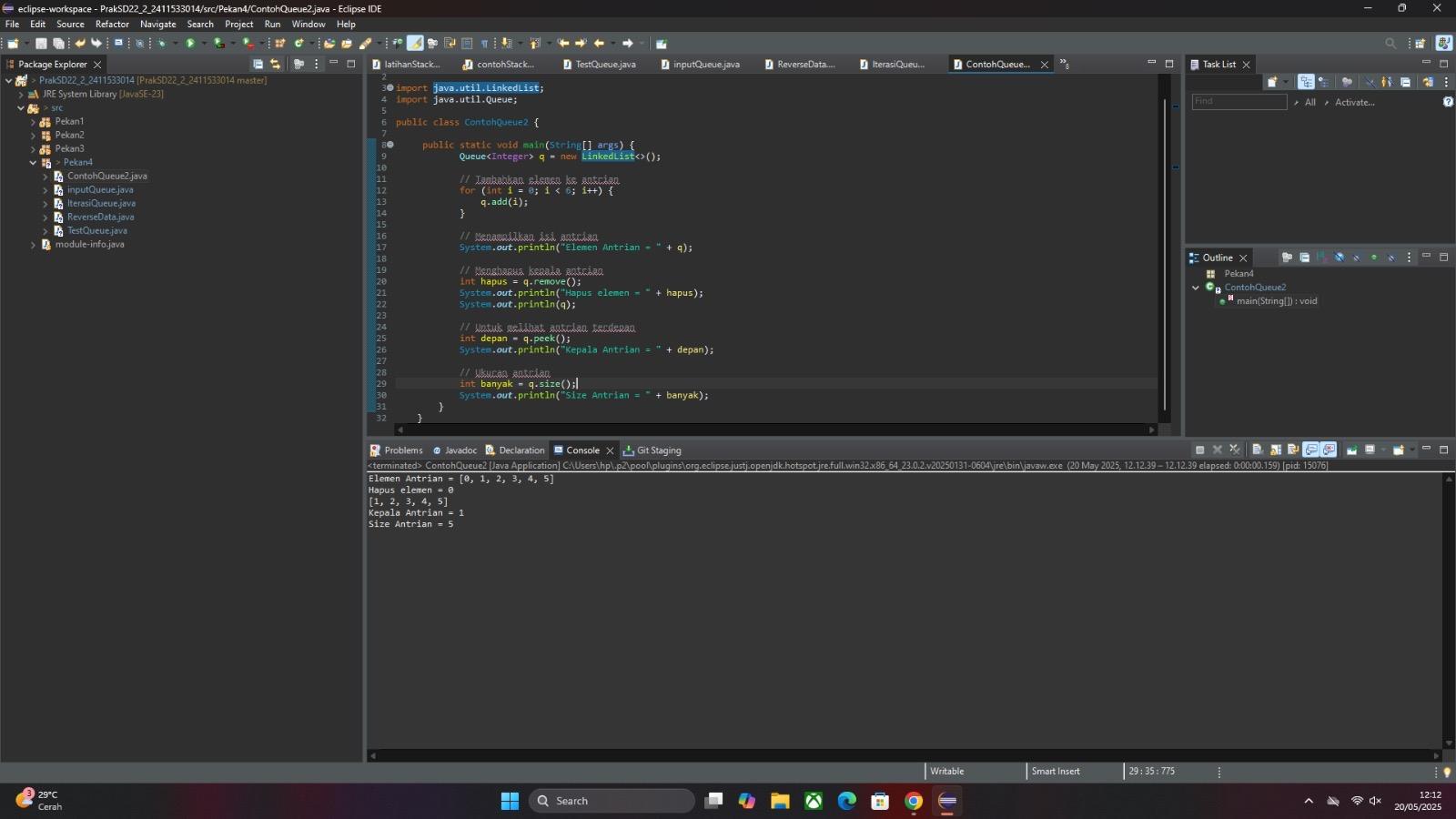
Front item is 20

Rear item is 10

• Baris-baris awal menunjukkan elemen yang ditambahkan ke antrian.

• Setelah dequeue(), elemen terdepan (10) dihapus, sehingga elemen terdepan yang baru adalah 20.

• Setelah menambahkan 10 lagi, 10 menjadi elemen terakhir (rear).

1. ContohQueue2
2. Mengimpor Kelas yang Diperlukan

import java.util.LinkedList;

import java.util.Queue;

Bagian ini mengimpor kelas LinkedList, yang akan digunakan untuk mengimplementasikan antrian, dan antarmuka Queue, yang menyediakan fungsionalitas antrian.

1. Mendefinisikan Kelas ContohQueue2

public class ContohQueue2 {

public static void main(String[] args) {

// ... (sisanya kode)

}

}

Ini mendefinisikan kelas bernama ContohQueue2. Metode main adalah titik masuk program.

1. Membuat Antrian

Queue<Integer> q = new LinkedList<>();

Sebuah antrian bernama q dibuat menggunakan LinkedList untuk menyimpan nilai integer. LinkedList dipilih karena menyediakan implementasi yang efisien untuk menambahkan dan menghapus elemen dari kedua ujungnya.

1. Menambahkan Elemen ke Antrian

// Menambahkan elemen ke antrian

for (int i = 0; i < 6; i++) {

q.add(i);

}

Perulangan ini menambahkan bilangan bulat dari 0 hingga 5 ke antrian q. Metode add() menambahkan elemen ke bagian belakang antrian.

1. Menampilkan Elemen Antrian

// Menampilkan isi antrian

System.out.println("Elemen Antrian = " + q);

Baris ini mencetak isi antrian q saat ini.

1. Menghapus Elemen dari Antrian

// Menghapus kepala antrian

int hapus = q.remove();

System.out.println("Hapus elemen = " + hapus);

System.out.println(q);

Metode remove() menghapus dan mengembalikan elemen di bagian depan (kepala) antrian. Kode kemudian mencetak elemen yang dihapus (hapus) dan antrian yang diperbarui.

1. Melihat Elemen Depan

// Untuk melihat antrian terdepan

int depan = q.peek();

System.out.println("Kepala Antrian = " + depan);

Metode peek() mengembalikan elemen di bagian depan antrian tanpa menghapusnya. Ini memungkinkan Anda untuk memeriksa elemen berikutnya yang akan diproses tanpa memodifikasi antrian.

1. Mendapatkan Ukuran Antrian

// Ukuran antrian

int banyak = q.size();

System.out.println("Size Antrian = " + banyak);

Metode size() mengembalikan jumlah elemen yang saat ini ada di antrian.

1. KESIMPULAN

Kode Java tersebut merupakan program sederhana yang mendemonstrasikan penggunaan LinkedList dalam Java untuk mengimplementasikan struktur data antrian (queue). Program ini menunjukkan operasi-operasi dasar antrian, termasuk menambahkan elemen (enqueue), menghapus elemen (dequeue), melihat elemen terdepan (peek), dan memeriksa ukuran antrian. Penggunaan LinkedList memberikan fleksibilitas dan efisiensi dalam implementasi antrian ini karena kemampuannya untuk menambahkan dan menghapus elemen dengan mudah dari kedua ujungnya. Secara keseluruhan, kode ini memberikan contoh yang jelas dan mudah dipahami tentang bagaimana mengelola antrian dalam pemrograman Java.