Laporan Praktikum
Struktur Data

Disusun Oleh :

Saskia Alifah

2411531002

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

Departemen Informatika

Fakultas Teknologi Informasi

Universitas Andalas

Tahun 2025

Queue

A. Tujuan Praktikum

1. Mempelajari konsep dasar struktur data **Queue (Antrian)**.
2. Mengimplementasikan antrian menggunakan **class bawaan Java (Queue dan LinkedList)** dan **struktur buatan sendiri (array)**.
3. Memahami cara kerja operasi dasar queue seperti **enqueue**, **dequeue**, **peek**, **iterasi**, serta **reverse (pembalikan antrian)**.

B. Pendahuluan

Queue (antrian) adalah struktur data linear yang mengikuti prinsip FIFO (First In, First Out), di mana elemen yang pertama kali masuk akan menjadi elemen pertama yang keluar. Struktur ini sangat umum digunakan dalam berbagai skenario komputasi seperti sistem penjadwalan, antrian cetak, buffer jaringan, dan lainnya. Dalam pemrograman Java, kita dapat mengimplementasikan queue menggunakan library bawaan seperti Queue, LinkedList, dan juga dengan membuat struktur queue manual menggunakan array.

Di Java, Queue adalah interface yang menyediakan metode dasar seperti add(), remove(), peek(), dan poll(). Salah satu implementasi konkret dari Queue adalah LinkedList. Dengan menggunakan LinkedList, kita dapat dengan mudah menambahkan dan menghapus elemen dari kedua ujungnya.

Selain itu, pemahaman tentang implementasi manual dari queue sangat penting untuk memahami bagaimana queue benar-benar bekerja di balik layar. Dengan membangun queue menggunakan array, kita dapat lebih memahami proses rotasi indeks (menggunakan operasi modulus), pengecekan overflow dan underflow, serta bagaimana menjaga konsistensi data dalam kapasitas terbatas.

Metode enqueue adalah proses menambahkan elemen ke belakang antrian, sementara dequeue adalah proses menghapus elemen dari depan antrian. Metode peek atau front digunakan untuk melihat elemen terdepan tanpa menghapusnya. Kita juga bisa menggunakan iterasi untuk menelusuri semua elemen dalam queue secara satu per satu.

Konsep pembalikan antrian (reverse queue) juga menjadi materi penting karena berguna saat kita ingin mengubah urutan proses. Pembalikan ini biasanya dilakukan dengan memanfaatkan stack, yang memiliki prinsip LIFO (Last In, First Out). Elemen dari queue dimasukkan ke dalam stack, kemudian diambil kembali satu per satu untuk membentuk queue baru dengan urutan terbalik.

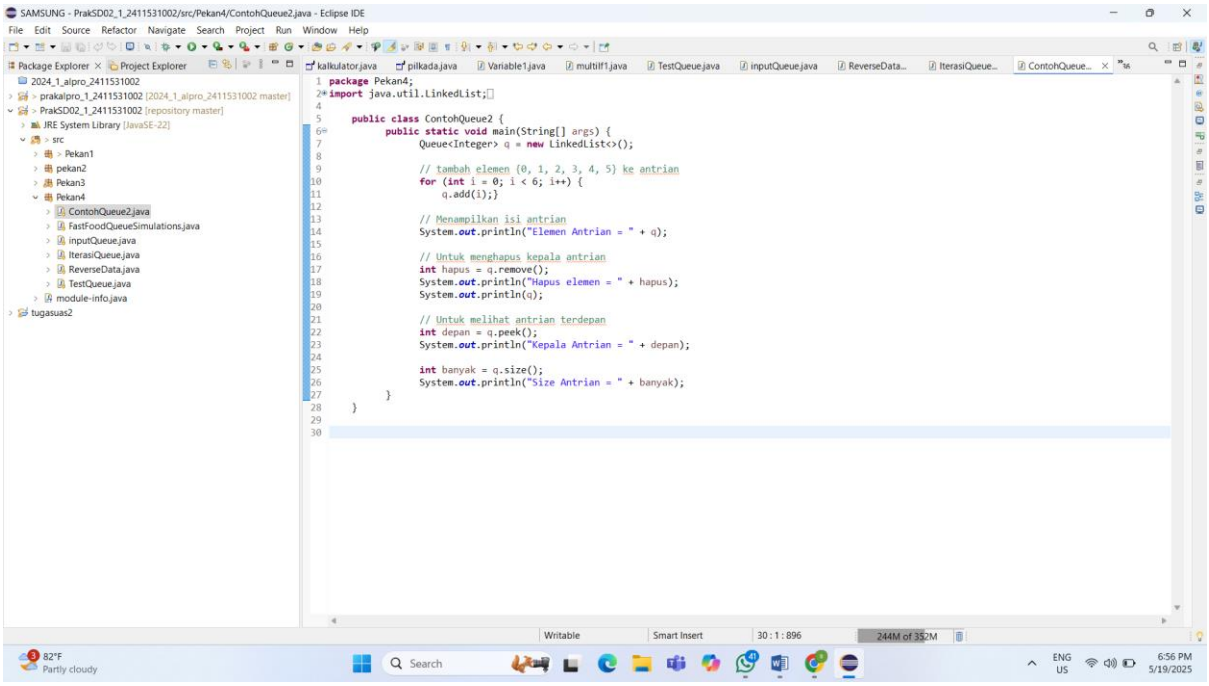
Dengan menggabungkan berbagai metode ini, kita dapat memahami dan mengimplementasikan berbagai macam operasi queue dengan lebih efisien. Kemampuan ini sangat penting untuk menyelesaikan permasalahan komputasi yang melibatkan urutan dan antrian.

C. Metode Praktikum

1. Class ContohQueue2

Kelas ini merupakan contoh implementasi antrian menggunakan antarmuka Queue dari Java dan kelas LinkedList sebagai struktur datanya. Dengan cara ini, kita bisa memanfaatkan fungsi-fungsi bawaan Java seperti add, remove, peek, dan size.

```
package Pekan4;
import java.util.LinkedList;
import java.util.Queue;

public class ContohQueue2 {
    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<>();

        // tambah elemen {0, 1, 2, 3, 4, 5} ke antrian
        for (int i = 0; i < 6; i++) {
           q.add(i);}

        // Menampilkan isi antrian
        System.out.println("Elemen Antrian = " + q);

        // Untuk menghapus kepala antrian
        int hapus = q.remove();
        System.out.println("Hapus elemen = " + hapus);
        System.out.println(q);

        // Untuk melihat antrian terdepan
        int depan = q.peek();
        System.out.println("Kepala Antrian = " + depan);

        int banyak = q.size();
        System.out.println("Size Antrian = " + banyak);
    }
}
```



Pada program ini, digunakan struktur data Queue dengan implementasi dari LinkedList. Pertama-tama, enam angka yaitu 0 hingga 5 dimasukkan ke dalam antrian menggunakan add(). Setelah itu, ditampilkan isi antrian, yang berisi angka dari 0 sampai 5. Kemudian, satu elemen di bagian depan antrian (0) dihapus menggunakan remove(). Antrian kemudian menampilkan isi barunya. Dengan peek(), kita melihat elemen yang sekarang berada di depan antrian (yaitu 1), dan dengan size() kita mengetahui jumlah total elemen saat ini.

Sehingga menghasilkan output :

```
<terminated> ContohQueue2 [Java Application] C:\Use
Elemen Antrian = [0, 1, 2, 3, 4, 5]
Hapus elemen = 0
[1, 2, 3, 4, 5]
Kepala Antrian = 1
Size Antrian = 5
```

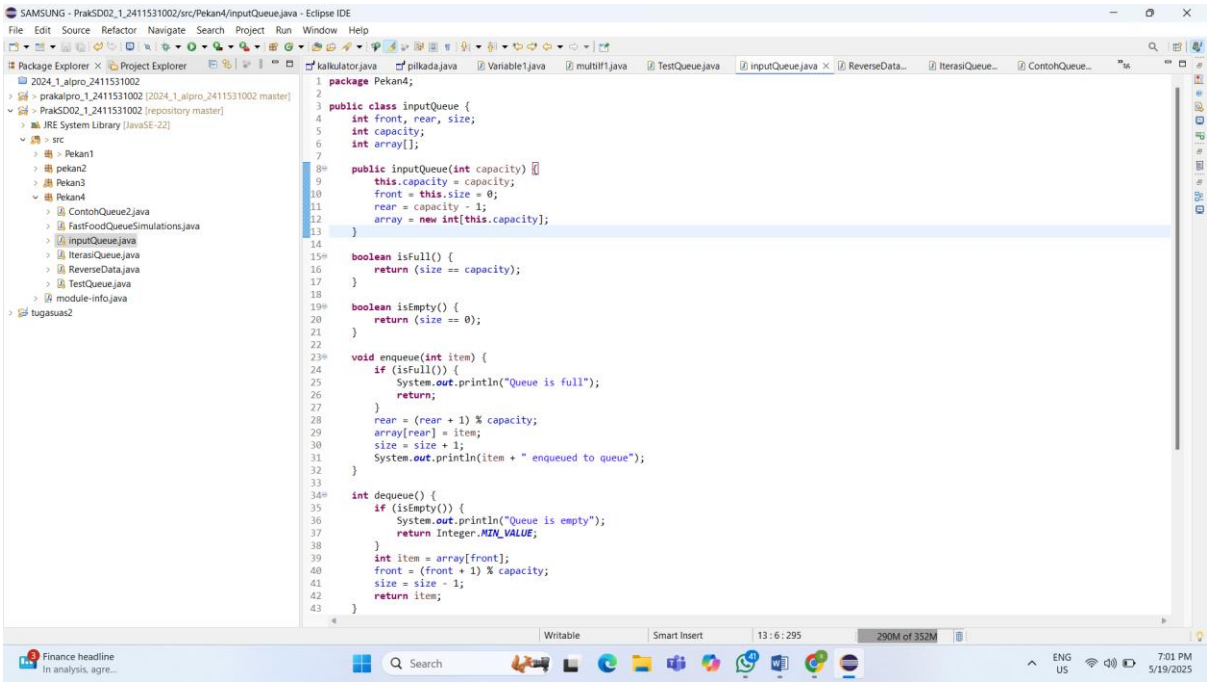
- Antrian awal berisi 6 elemen dari 0 hingga 5.
- Setelah elemen pertama (0) dihapus, tersisa 5 elemen.
- Elemen terdepan sekarang adalah 1.
- Ukuran antrian menjadi 5.

2. Array-based queue

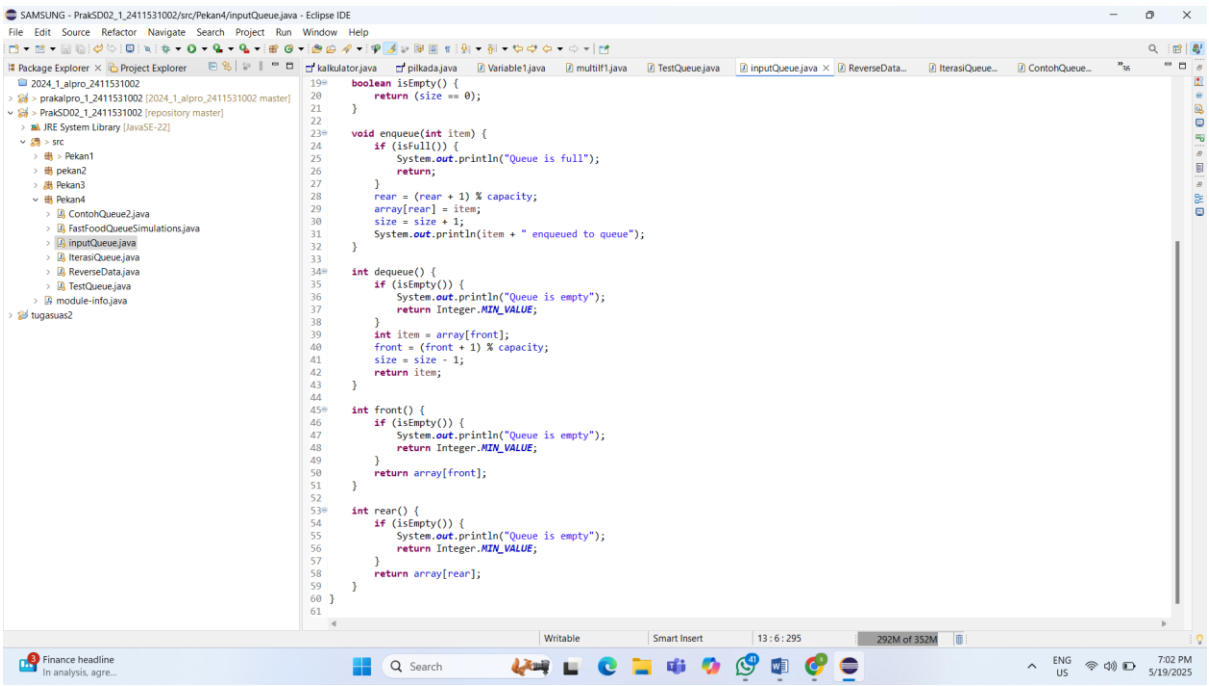
Program berikut merupakan queue yang dibuat menggunakan basis array. Yakni **class inputQueue :**

```
package Pekan4;

public class inputQueue {
   int front, rear, size;
```

```
    int capacity;
    int array[];

    public inputQueue(int capacity) {
        this.capacity = capacity;
        front = this.size = 0;
        rear = capacity - 1;
        array = new int[this.capacity];
    }

    boolean isFull() {
        return (size == capacity);
    }

    boolean isEmpty() {
        return (size == 0);
    }

    void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        rear = (rear + 1) % capacity;
        array[rear] = item;
        size = size + 1;
        System.out.println(item + " enqueued to queue");
    }

    int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return Integer.MIN_VALUE;
        }
        int item = array[front];
        front = (front + 1) % capacity;
        size = size - 1;
        return item;
    }

    int front() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return Integer.MIN_VALUE;
        }
        return array[front];
    }

    int rear() {
        if (isEmpty()) {
            System.out.println("Queue is empty");Z
            return Integer.MIN_VALUE;
        }
        return array[rear];
    }
}
```



```java
package Pekan4;

public class inputQueue {
    int front, rear, size;
    int capacity;
    int array[];

    public inputQueue(int capacity) {
        this.capacity = capacity;
        front = this.size = 0;
        rear = capacity - 1;
        array = new int[this.capacity];
    }

    boolean isFull() {
        return (size == capacity);
    }

    boolean isEmpty() {
        return (size == 0);
    }

    void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        rear = (rear + 1) % capacity;
        array[rear] = item;
        size = size + 1;
        System.out.println(item + " enqueued to queue");
    }

    int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return Integer.MIN_VALUE;
        }
        int item = array[front];
        front = (front + 1) % capacity;
        size = size - 1;
        return item;
    }
```



```java
    boolean isEmpty() {
        return (size == 0);
    }

    void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        rear = (rear + 1) % capacity;
        array[rear] = item;
        size = size + 1;
        System.out.println(item + " enqueued to queue");
    }

    int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return Integer.MIN_VALUE;
        }
        int item = array[front];
        front = (front + 1) % capacity;
        size = size - 1;
        return item;
    }

    int front() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return Integer.MIN_VALUE;
        }
        return array[front];
    }

    int rear() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return Integer.MIN_VALUE;
        }
        return array[rear];
    }
}
```

Method isFull mengembalikan nilai boolean apakah kapasitas dari array yang digunakan sebagai basis Queue sudah penuh. Method isEmpty mengembalikan nilai boolean apakah queue kosong. Method enqueue untuk menambahkan elemen ke queue, sedangkan dequeue untuk mengembalikan elemen dari queue. Terakhir, method front mengembalikan nilai terdepan atau nilai yang pertama di-enqueue, dan method rear mengembalikan nilai terakhir dari queue.

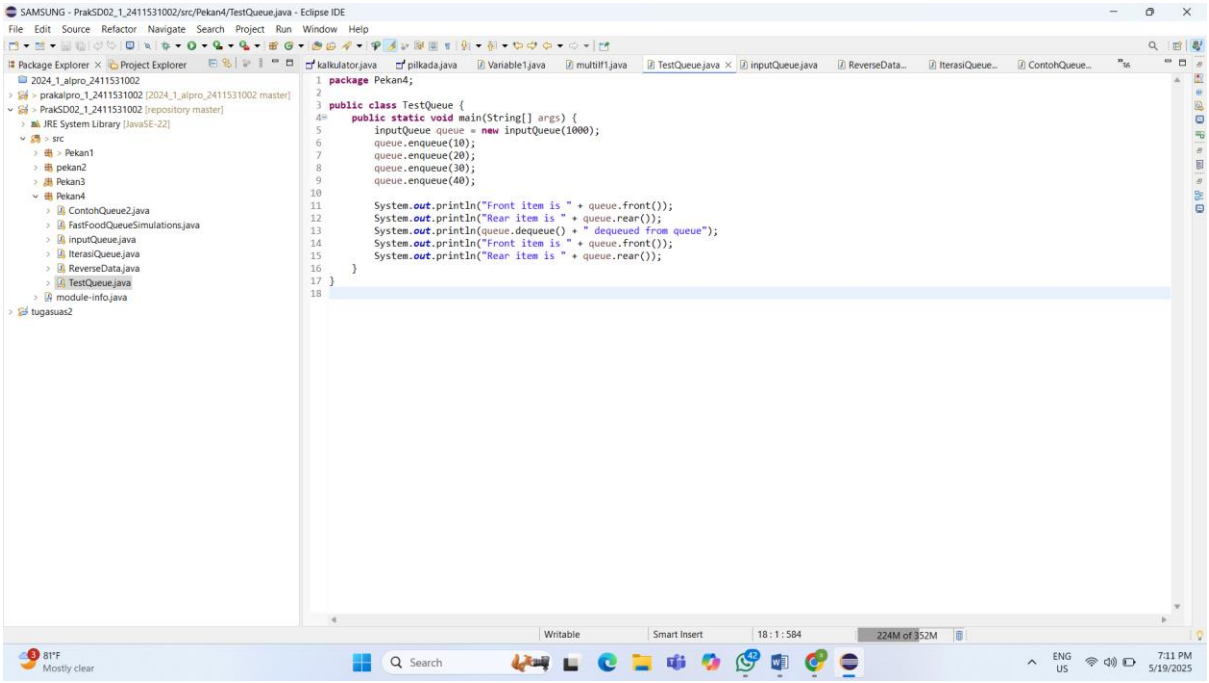
Kelas ini adalah implementasi antrian menggunakan array. Metode enqueue menambahkan elemen ke antrian, dequeue menghapus elemen dari depan. front() dan rear() masing-masing mengembalikan elemen terdepan dan terakhir. Konsep circular queue digunakan untuk efisiensi.

Berikut contoh program yang mengimplementasikan array-based queue yang telah dibuat di kelas inputQueue:

Class inputQueue :

Kelas ini berfungsi untuk menguji fungsionalitas dari inputQueue. Program akan menambahkan beberapa elemen ke antrian, lalu menampilkan elemen depan dan belakang, kemudian melakukan penghapusan (dequeue) satu elemen, dan mencetak elemen depan dan belakang setelah penghapusan.

```
package Pekan4;

public class TestQueue {
   public static void main(String[] args) {
      inputQueue queue = new inputQueue(1000);
      queue.enqueue(10);
      queue.enqueue(20);
      queue.enqueue(30);
      queue.enqueue(40);

      System.out.println("Front item is " + queue.front());
      System.out.println("Rear item is " + queue.rear());
      System.out.println(queue.dequeue() + " dequeued from queue");
      System.out.println("Front item is " + queue.front());
      System.out.println("Rear item is " + queue.rear());
   }
}
```



Output dari program tersebut adalah sebagai berikut:

```
<terminated> TestQueue [Java Application] C:\
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
Front item is 10
Rear item is 40
10 dequeued from queue
Front item is 20
Rear item is 40
```

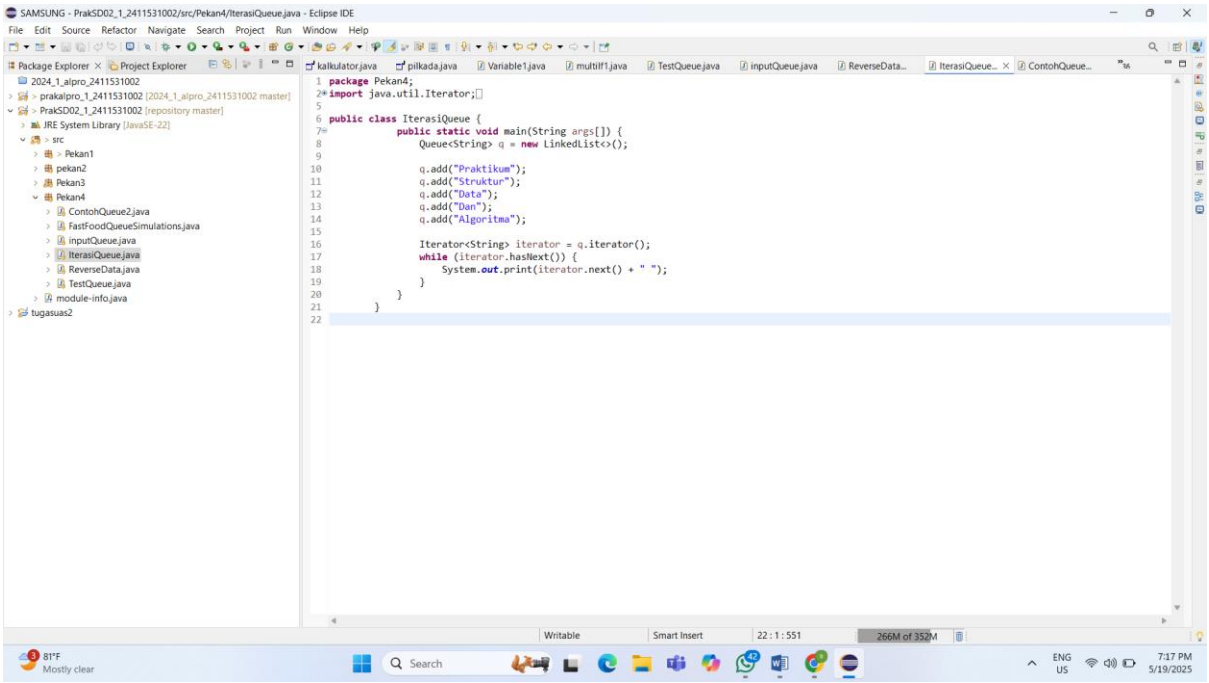
- Setelah empat elemen dimasukkan, antrian memiliki elemen 10, 20, 30, 40.
- Elemen pertama (10) dihapus.
- Elemen terdepan menjadi 20 dan belakang tetap 40.

3. Class iterasiQueue

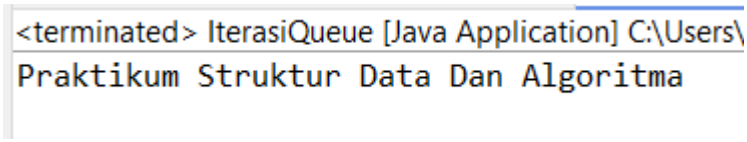
Kelas ini menunjukkan cara menelusuri seluruh isi antrian tanpamenghapusnya menggunakan iterator. Ini berguna ketika kita ingin melihat semua data tanpa mengubah struktur antrian.

```
package Pekan4;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Queue;



public class IterasiQueue {

		public static void main(String args[]) {

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



		    q.add("Praktikum");

		    q.add("Struktur");

		    q.add("Data");

		    q.add("Dan");

		    q.add("Algoritma");
```

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

        while (iterator.hasNext()) {

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

        }

    }

}
```



Sehingga menghasilkan output :



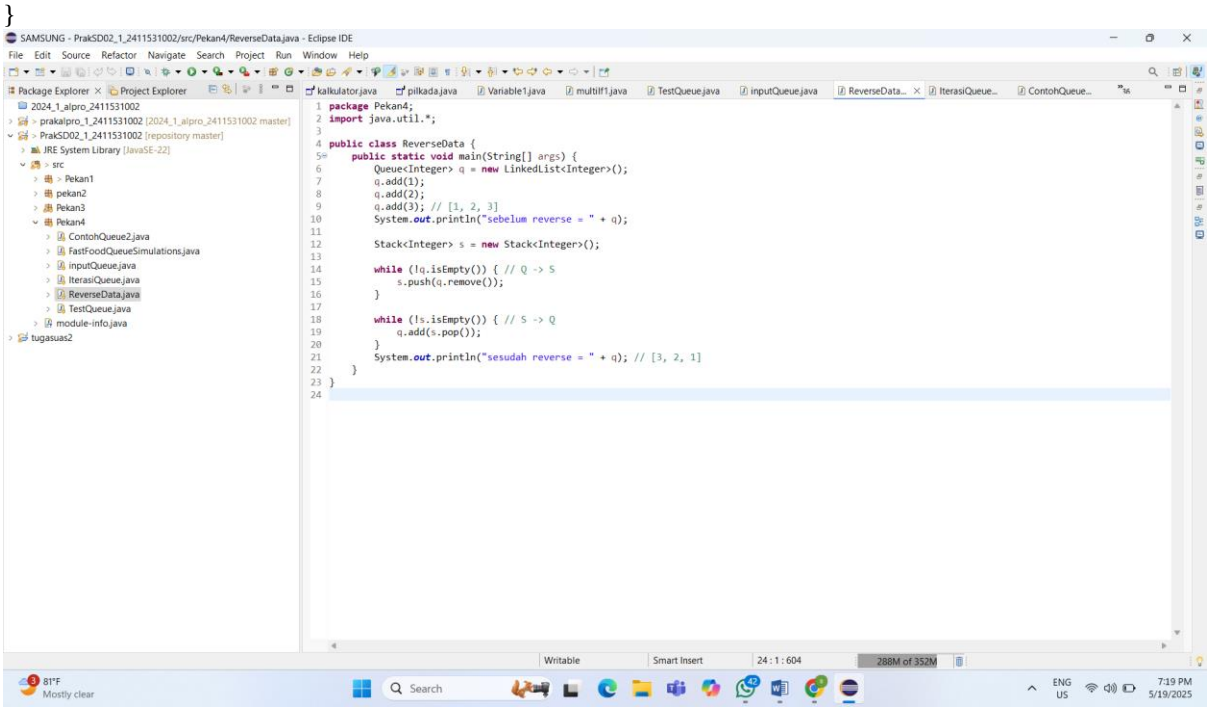
Seluruh elemen dalam antrian ditampilkan secara berurutan tanpa menghapus elemen di dalamnya. Iterator sangat berguna dalam pengolahan data yang membutuhkan pembacaan berurutan.

4. Membalikkan isi queue (reverse)

Program berikut membalikkan isi dari sebuah queue, yakni **class ReverseData**:

```
package Pekan4;
import java.util.*;
```

```java
public class ReverseData {
    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<Integer>();
        q.add(1);
        q.add(2);
        q.add(3); // [1, 2, 3]
        System.out.println("sebelum reverse = " + q);

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

        while (!q.isEmpty()) { // Q -> S
            s.push(q.remove());
        }

        while (!s.isEmpty()) { // S -> Q
            q.add(s.pop());
        }
        System.out.println("sesudah reverse = " + q); // [3, 2, 1]
    }
}
```



Untuk membalikkan isi dari queue diperlukan sebuah stack untuk menyimpan elemen-elemen dari queue. Karena prinsip LIFO pada stack, maka jika elemen-elemen queue dikeluarkan, dan dimasukkan ke dalam stack, lalu dikeluarkan kembali dari stack, maka elemen-elemen dari queue tersebut akan terbalik.

Berikut output dari program tersebut:

```
<terminated> ReverseData [Java Application] C:\
sebelum reverse = [1, 2, 3]
sesudah reverse = [3, 2, 1]
```

- Awalnya antrian berisi 1, 2, 3.
- Elemen dipindahkan ke stack, sehingga urutannya terbalik.
- Elemen dikembalikan ke antrian dari stack, hasilnya adalah antrian dengan urutan terbalik.

D. Kesimpulan Praktikum

Dari praktikum ini, dapat disimpulkan bahwa antrian merupakan struktur data penting yang dapat diimplementasikan baik menggunakan struktur data bawaan Java (Queue, LinkedList) maupun secara manual menggunakan array. Konsep FIFO sangat berguna dalam berbagai aplikasi, dan pemahaman tentang operasi dasar seperti enqueue, dequeue, peek, serta teknik iterasi dan pembalikan data akan sangat bermanfaat dalam pengembangan perangkat lunak. Praktikum ini juga melatih kemampuan logika dalam menyusun alur data dan mengoptimalkan penggunaannya dalam kode program.