

LAPORAN PRAKTIKUM STRUKTUR DATA PEKAN 4
QUEUE



OLEH :
FADIL INSANUS SIDDIK
(2411532013)

MATA KULIAH : PRAKTIKUM STRUKTUR DATA
DOSEN PENGAMPU : Dr. Wahyudi, S.T, M.T.

FAKULTAS TEKNOLOGI INFORMASI
DEPARTEMEN INFORMATIKA
UNIVERSITAS ANDALAS

PADANG, MEI 2025

A. PENDAHULUAN

Struktur data merupakan fondasi penting dalam pengembangan perangkat lunak karena menentukan bagaimana data disimpan, dikelola, dan diakses secara efisien. Salah satu struktur data yang sering digunakan dalam pemrograman adalah Queue atau antrian. Queue merupakan struktur data linear yang mengikuti prinsip FIFO (First In, First Out), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar.

Dalam praktikum ini, mahasiswa mempelajari implementasi dan penggunaan Queue dalam bahasa pemrograman Java. Beberapa pendekatan yang digunakan meliputi penggunaan class bawaan Java seperti `LinkedList` dan `Queue`, serta implementasi manual circular queue menggunakan array. Selain itu, mahasiswa juga mempraktikkan bagaimana cara melakukan iterasi pada antrian menggunakan `Iterator`, membalik antrian menggunakan bantuan struktur data `Stack`, dan mengakses elemen depan (`front`) dan belakang (`rear`) pada antrian.

Pemahaman tentang bagaimana antrian bekerja sangat penting, karena struktur ini banyak digunakan dalam kehidupan nyata, seperti sistem antrian pelanggan, pemrosesan data secara berurutan, serta manajemen tugas dalam sistem operasi. Dengan melakukan praktikum ini, mahasiswa diharapkan tidak hanya memahami konsep teoritis Queue, tetapi juga mampu mengimplementasikannya secara langsung dalam kode program Java.

B. TUJUAN PRAKTIKUM

Tujuan dari praktikum Struktur Data minggu ini adalah sebagai berikut:

1. Memahami konsep dasar Queue dan bagaimana prinsip FIFO diterapkan dalam struktur data.
2. Mempelajari implementasi Queue menggunakan class `LinkedList` dan `Queue` dari Java Collection Framework.
3. Mengimplementasikan Circular Queue menggunakan array secara manual untuk memahami cara kerja antrian dalam bentuk melingkar.
4. Mengenal penggunaan `Iterator` untuk menelusuri elemen dalam Queue.
5. Mempelajari teknik membalik antrian (`reverse Queue`) menggunakan struktur data tambahan yaitu `Stack`.
6. Melatih keterampilan pemrograman Java, khususnya dalam penggunaan struktur data dinamis dan pengelolaan elemen dalam antrian.
7. Menumbuhkan pemahaman logika algoritma, terutama bagaimana elemen dimasukkan dan dikeluarkan dari antrian serta dampaknya terhadap data yang tersisa.

C. LANGKAH KERJA

1. ContohQueue2

```
package Pekan4;
```

- Menyatakan bahwa kelas ini berada dalam package bernama Pekan4.
- Package berguna untuk mengelompokkan kelas-kelas dalam proyek Java.

```
3 import java.util.Queue;  
4 import java.util.LinkedList;
```

- Mengimpor interface Queue dari Java Collection Framework.
- Queue adalah antarmuka untuk struktur data antrian (FIFO).
- Begitu juga dengan import java.util.LinkedList;

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

- Membuat objek q bertipe Queue yang menyimpan elemen Integer.
- Diimplementasikan menggunakan LinkedList karena Queue adalah interface, tidak bisa langsung dibuat objeknya.
- <> adalah diamond operator, otomatis mendeteksi tipe dari kiri (Integer).

```
for (int i = 0; i < 6; i++)  
    q.add(i);
```

- Mengisi antrian dengan angka dari 0 sampai 5 menggunakan method add().

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

- Menampilkan isi antrian setelah pengisian.
- Contoh output: Elemen Antrian [0, 1, 2, 3, 4, 5]

```
int hapus = q.remove();
```

- Menghapus elemen pertama dari antrian dan menyimpannya ke variabel hapus.
- remove() mengikuti prinsip FIFO (First In, First Out).

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

- Menampilkan elemen yang dihapus tadi dari antrian.

```
System.out.println(q);
```

- Menampilkan isi antrian **setelah** penghapusan.

```
int depan = q.peek();
```

- Mengambil elemen **terdepan** dari antrian tanpa menghapusnya menggunakan peek().

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

- Menampilkan elemen paling depan (kepala) saat ini.

```
int banyak = q.size();
```

- Menghitung jumlah elemen dalam antrian setelah penghapusan.

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

- menampilkan jumlah elemen dalam antrian.
- Output:

```
Elemen Antrian [0, 1, 2, 3, 4, 5]
Hapus elemen = 0
[1, 2, 3, 4, 5]
Kepala Antrian = 1
Size Antrian = 5
```

2. inputQueue

```
3 public class inputQueue {
4     int front, rear, size;
5     int capacity;
6     int array[];
```

- Membuat kelas inputQueue untuk implementasi antrian.
- front dan rear: indeks elemen pertama dan terakhir dalam queue.
- size: jumlah elemen yang sedang berada dalam antrian.
- capacity: kapasitas maksimum queue.
- array[]: array untuk menyimpan elemen antrian.

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

- Konstruktor kelas: inisialisasi nilai awal queue.

- `rear = capacity - 1` dilakukan agar saat enqueue pertama, `(rear + 1) % capacity` jadi 0.
- array dialokasikan dengan ukuran sesuai `capacity`.

```
}
boolean isFull(inputQueue queue) {
    return (queue.size == queue.capacity);
}
```

- Mengecek apakah antrian sudah penuh.

```
boolean isEmpty(inputQueue queue)
{
    return (queue.size == 0);
}
```

- Mengecek apakah antrian kosong.

```
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 ke antrian:
 - Cek dulu apakah penuh.
 - Update rear secara melingkar.
 - Tambahkan elemen ke array.
 - Tambah ukuran size.
 - Tampilkan pesan

```
int dequeue()
{
    if (isEmpty(this))
        return Integer.MIN_VALUE;
    int item = this.array[this.front];
    this.front = (this.front + 1) %
    this.capacity;
    this.size = this.size - 1;
    return item;
}
```

- Menghapus (mengeluarkan) elemen dari depan antrian:
 - Cek apakah antrian kosong.
 - Simpan elemen paling depan ke item.
 - Perbarui front secara melingkar.
 - Kurangi size.
 - Kembalikan nilai yang dihapus.

```
int front()
{
    if (isEmpty(this))
        return Integer.MIN_VALUE;
    return this.array[this.front];
}
```

- Mengembalikan elemen paling depan tanpa menghapus.
- Jika kosong kembalikan Integer.Min_VALUE.

```
int rear()
{
    if(isEmpty(this))
        return Integer.MIN_VALUE;
    return this.array[this.rear];
}
```

- Mengembalikan elemen terakhir dalam antrian tanpa menghapus.
- Output: tidak ada output untuk class ini.

3. IterasiQueue

```
import java.util.*;
```

- Mengimpor semua kelas dari package java.util

```
public class IterasiQueue {
```

- Deklarasi kelas public bernama IterasiQueue.

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

- Membuat objek antrian `q` yang menyimpan elemen bertipe `String`.
- Menggunakan `LinkedList` sebagai implementasi dari interface `Queue`.

```
q.add(" Praktikum");
q.add(" Struktur");
q.add(" Data");
q.add(" dan");
q.add(" Algoritma");
```

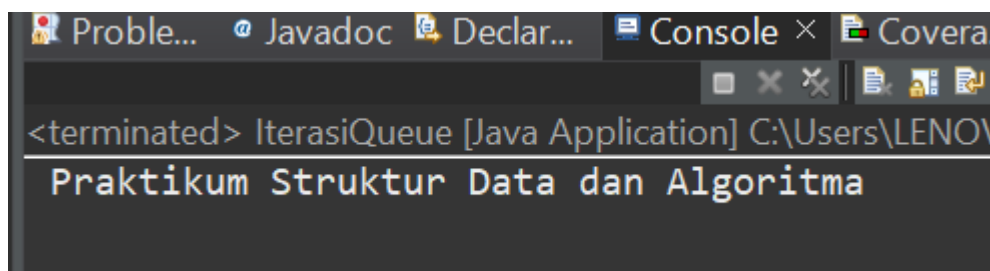
- Menambahkan lima buah `String` ke dalam antrian `q`.
- Semua kata dipisahkan dan diawali spasi agar nanti saat dicetak langsung membentuk kalimat.

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

- Membuat objek `iterator` untuk menelusuri elemen dalam `Queue`.
- `iterator()` menghasilkan objek untuk menelusuri isi `q` dari depan ke belakang.

```
while (iterator.hasNext()) {
    System.out.print(iterator.next() + "");
}
```

- `while (iterator.hasNext())`: loop selama masih ada elemen berikutnya.
- `iterator.next()`: mengambil elemen selanjutnya dari queue.
- `System.out.print(...)`: mencetak elemen tanpa pindah baris.
- `+ ""` sebenarnya tidak diperlukan; hasilnya tetap string.
- Output:



4. ReverseData

```
class ReverseData {
```

- Mendefinisikan class Java bernama ReverseData.

```
public static void main(String[] args) {
```

- Method utama tempat program dimulai saat dijalankan.

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

- Membuat objek `q` bertipe `Queue` yang menyimpan elemen `Integer`.
- Menggunakan `LinkedList` sebagai implementasi dari `Queue`.

```
q.add(1);  
q.add(2);  
q.add(30);
```

- Menambahkan elemen ke dalam antrian secara FIFO (First In First Out).
- Sekarang isi antrian `q` adalah: `[1, 2, 30]`

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

- Menampilkan isi antrian sebelum dibalik.
- Output: `sebelum reverse[1, 2, 30]`

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

- Membuat objek `s` bertipe `Stack` untuk menyimpan elemen secara LIFO (Last In First Out).

```
while (!q.isEmpty()) {  
    s.push(q.remove());  
}
```

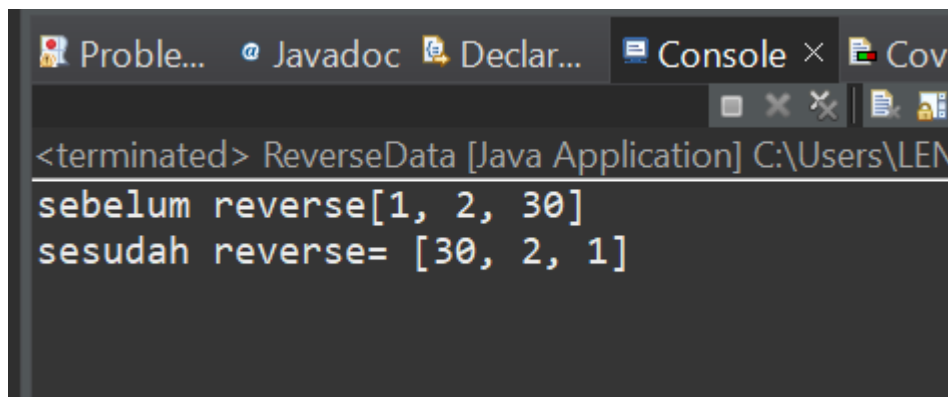
- Selama antrian tidak kosong:
 - Ambil elemen dari antrian (`q.remove()`), dan
 - Dorong ke dalam stack (`s.push(...)`).
- Setelah proses ini, antrian kosong dan stack berisi `[30, 2, 1]` (karena LIFO).


```
while (!s.isEmpty()) {
    q.add(s.pop());
}
```

- Selama stack tidak kosong:
 - Keluarkan elemen dari stack (`s.pop()`), dan
 - Masukkan kembali ke dalam antrian (`q.add(...)`).
- Ini akan membalik urutan antrian menjadi `[30, 2, 1]`.

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

- Menampilkan isi antrian setelah dibalik.
- Output:



```
<terminated> ReverseData [Java Application] C:\Users\LEN
sebelum reverse[1, 2, 30]
sesudah reverse= [30, 2, 1]
```

5. TestQueue

```
public class TestQueue {
```

- Mendefinisikan kelas utama bernama TestQueue.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
```

- Method `main()` adalah titik awal eksekusi program Java.
- Komentar TODO adalah catatan otomatis dari IDE yang bisa diabaikan di sini.

```
inputQueue queue = new inputQueue(1000);
```

- Membuar objek queue dari class inputQueue dengan kapasitas 1000 elemen.
- inputQueue adalah class buatan sendiri yang digunakan untuk menangani queue menggunakan array.

```
queue.enqueue(10);
queue.enqueue(20);
queue.enqueue(30);
queue.enqueue(40);
```

- Menambahkan 4 elemen ke antrian secara berurutan:
 - 10, 20, 30, 40
- Saat ini isi queue: [10, 20, 30, 40]

```
System.out.println("Front item is " + queue.front());
```

- Menampilkan elemen paling depan (front) dari antrian, yaitu 10.

```
System.out.println("Rear item is " + queue.rear());
```

- Menampilkan elemen paling belakang (rear) dari antrian, yaitu 40.

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

- Menghapus elemen paling depan dari antrian (yaitu 10) menggunakan dequeue().
- Menampilkan: 10 dequeued from queue
- Sekarang isi antrian: [20, 30, 40]

```
System.out.println("Front item is " + queue.front());
```

- Menampilkan elemen depan yang baru, yaitu 20.

```
System.out.println("Rear item is " + queue.rear());
```

- Menampilkan elemen paling belakang tetap 40 karena rear tidak berubah.
- Output:

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