LAPORAN PRAKTIKUM STRUKTUR DATA

QUEUE

OLEH:

AHMAD ZAHRAN

2411532004

DOSEN PENGAMPU:

DR. WAHYUDI, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2024

A. Pendahuluan

Queue atau antrian adalah struktur data linier yang memproses elemen berdasarkan urutan masuk (*First In First Out – FIFO*). Dalam praktik pengembangan perangkat lunak, sering kali kita menggunakan library collection seperti LinkedList untuk mengimplementasikan queue. Namun, pemahaman yang lebih dalam akan didapat dengan membuat implementasi queue secara manual menggunakan array.

B. Tujuan

Tujuan dilakukannya praktikum ini adalah sebagai berikut:

1. Memahami Struktur Data Queue pada java
2. Mengimplementasikan Queue

C. Langkah Kerja Praktikum

a. Kelas Contoh Queue

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

1. Pertama buat sebuah Queue dengan nama ‘q’ dengan TipeData Integer, dan menggunakan LinkedList sebagai implementasinya.

```
for (int i = 0; i < 6; i++)
    q.add(i);
    System.out.println("Element Anrtian "+ q);
```

2. Melakukan perulangan dari 0 hingga 5, lalu menambahkan angka tersebut ke dalam antrian q satu per satu menggunakan add(). Dan Cetak semua element dalam antrian.

```
int hapus = q.remove();
System.out.println("Hapus element = "+ hapus);
System.out.println(q);
```

3. Menghapus elemen pertama dari antrian dan menyimpannya dalam variable hapus dan cetak nilai setelah penghapusan.

```
int depan = q.peek();
System.out.println("element depan = "+ depan);
```

4. Menggunakan **peek()** untuk melihat elemen paling depan tanpa menghapusnya.

```
int banyak = q.size();
System.out.println("Banyak element = "+ banyak);
```

5. Menghitung dan mencetak jumlah elemen dalam antrian saat ini menggunakan **size.**

b. Kelas inputQueue

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

1. Deklarasi Variabel yang mana disini ada front, rear, size, capacity, dan array

```
    this.capacity = capacity;
    front = this.size = 0;
    rear = capacity - 1;
    array = new int [this.capacity];
}
```

2. Inisiasi kapasitas antrian lalu untuk front, size nya diatur ke 0 untuk menandakan antrean masih kosong dan rear nya di atur ke capacity -1 agar saat enqueuer pertama, nilai rear bias menjadi 0, membuat array dengan ukuram sesuai kapasitas untuk menyimpan elemen antrian.

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

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

3. Mengecek kondisi Queue.

```
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");
}
```

4. Menambahkan elemen ke Antrian

```
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;
```

5. Menghapus elemen dari Antrian

```
int front()
{
    if (isEmpty(this))
        return Integer.MIN_VALUE;

    return this.array[this.front];
}
```

6. Mengakses Elemen terdepan

```
int rear()
{
    if (isEmpty(this))
        return Integer.MIN_VALUE;

    return this.array[this.rear];
}
```

7. Mengakses elemen terakhir

c. TestQueue

```
inputQueue queue = new inputQueue(1000);
queue.enqueue(10);
queue.enqueue(20);
queue.enqueue(30);
queue.enqueue(40);
```

1. Membuat Queue dengan kapasitas (1000) dan menambahkan elemen menggunakan ‘enqueue’.

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

2. Menampilkan elemen paling depan dan belakang dari queue

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

3. Menghapus elemen dari antrian mencetaknya.

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

4. Menampilkan kembali elemen paling depan dan belang setelah penghapusan.

d. IterasiQueue

```
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");
```

1. Membuat Queue ‘q’ dengan tipedata String dan mengisi Queue dengan mengguanakn **“add”.**

```
        Iterator<String> iterator = q.iterator();
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
```

2. Menggunakan **Iterator** untuk membaca dan mencetak semua elemen satu persatu

e. Kelas ReverseData

```
public class ReverseData {

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

1. Membuat Queue ‘q’ dan mengisinya dengan data integer.

```
        Stack<Integer> s = new Stack<Integer>();
        while (!q.isEmpty()) {
            s.push (q.remove());
        }
```

2. Lalu pindahkan semua elemen queue ke stack untuk membalik urutan.

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

3. Pindahkan kembali dari Stack ke Queue agar menghasilkan queue dengan urutan terbalik

D. Kesimpulan

Praktikum ini membahas berbagai cara penggunaan struktur data Queue di Java, baik secara manual (dengan array) maupun menggunakan fitur dari Java Collections Framework. Implementasi manual membantu mahasiswa memahami cara kerja internal queue, termasuk pengelolaan indeks secara melingkar. Sementara itu, penggunaan kelas bawaan seperti Queue, LinkedList, dan Stack memberikan efisiensi dan kemudahan dalam pengolahan data.

Dengan tambahan pemanfaatan Iterator dan stack, praktikum ini memperkenalkan keterampilan lanjutan yang sering dibutuhkan dalam algoritma dan rekayasa perangkat lunak, seperti iterasi, pembalikan data, dan pengujian struktur dinamis.