LAPORAN PRAKTIKUM STRUKTUR DATA
"*Queue*"

OLEH :

EZZA ADDINI

2311532001

DOSEN PENGAMPU: Dr. WAHYUDI, M. T.

DEPARTEMEN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

PADANG

2024

A. PENDAHULUAN

Dalam struktur data, queue dalam bahasa Java adalah struktur data yang mengikuti prinsip FIFO (First In, First Out), yang berarti elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang keluar. Java menyediakan beberapa implementasi queue dalam package java.util, seperti LinkedList, PriorityQueue, dan ArrayDeque. Queue digunakan untuk menyimpan elemen secara terurut dan memprosesnya satu per satu sesuai urutan kedatangan.

Operasi pada queue mencakup berbagai tindakan yang bisa dilakukan untuk menambahkan, mengakses, dan menghapus elemen. Berikut adalah operasi dasar yang biasanya tersedia dalam implementasi queue di Java:

1. Menambahkan Elemen (Enqueue):
    - *add*(E e): Menambahkan elemen ke akhir queue. Melempar IllegalStateException jika tidak ada ruang tersedia.
    - *offer*(E e): Menambahkan elemen ke akhir queue. Mengembalikan false jika tidak ada ruang tersedia, tanpa melempar pengecualian.
2. Menghapus Elemen (Dequeue):
    - *remove*(): Menghapus dan mengembalikan elemen pertama dalam queue. Melempar NoSuchElementException jika queue kosong.
    - *poll*(): Menghapus dan mengembalikan elemen pertama dalam queue. Mengembalikan null jika queue kosong.
3. Mengakses Elemen Pertama:
    - *element*(): Mengembalikan elemen pertama tanpa menghapusnya. Melempar NoSuchElementException jika queue kosong.
    - *peek*(): Mengembalikan elemen pertama tanpa menghapusnya. Mengembalikan null jika queue kosong.

B. TUJUAN

Tujuan dari praktikum ini adalah:

1. Membuat program latihan queue.
2. Membuat program input untuk queue yang dibuat.
3. Membuat program reverse data pada queue.

C. LANGKAH KERJA

a. Program Latihan Queue

1. Buat package dan class baru, namakan sesuai dengan yang diperintahkan atau yg diinginkan. Jangan lupa untuk menambahkan “import java.util.*;” pada baris setelah package.

```
1 package Pekan3;
2 import java.util.*;
3 public class TestQueue {
```

2. Deklarasikan nama queue yang akan digunakan.

```
public static void main(String[] args) {
    InputQueue queue = new InputQueue(1000);
```

3. Inisialisasikan elemen-elemen yang akan dimasukkan ke dalam stack.

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

4. Buatkan sintaks untuk menampilkan output program.

```
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());
```

5. Untuk menjalankan program dibutuhkan kelas yang berisi input untuk queue.

b. Membuat Program Input untuk Queue

1. Buat class baru dan namakan sesuai nama queue yang telah dibuat pada program sebelumnya, lalu buat “import java.util.*;” pada baris setelah package.

```
1 package Pekan3;
2 import java.util.*;
3 public class InputQueue {
```

2. Deklarasikan tipe data yang akan digunakan.

```
int front, rear, size;
int capacity;
int array[];
```

3. Buat sintaks untuk menginisialisasi objek queue dengan kapasitas seperti berikut.

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

4. Buatkan metode untuk mengecek apakah sebuah queue sudah penuh.

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

5. Buatkan metode untuk mengecek apakah sebuah queue masih kosong.

```
17●boolean isEmpty(InputQueue queue) {
18      return (queue.size == 0);
```

6. Buatkan metode enqueue untuk memeriksa apakah queue sudah penuh sebelum menambahkan elemen baru.

```
20●void enqueue(int item) {
21      if (isFull(this))
22          return;
23      this.rear = (this.rear + 1) % this.capacity;
24      this.array[this.rear] = item;
25      this.size = this.size + 1;
26      System.out.println(item + " enqueued to queue");
```

7. Buatkan metode dequeue untuk menghapus dan mengembalikan elemen dari depan queue, sekaligus memeriksa apakah queue kosong sebelum mencoba menghapus elemen.

```
28●int dequeue() {
29      if (isEmpty(this))
30          return Integer.MIN_VALUE;
```

8. Buatkan metode dequeue dan pastikan bahwa elemen dihapus dari depan antrian dan indeks serta ukuran diperbarui secara akurat.

```
32 int item = this.array[this.front] % this.capacity;
33 this.size = this.size - 1;
34 return item;
35 }
```

9. Buatkan metode front untuk mendapatkan elemen depan dari queue tanpa menghapusnya.

```
36●int front () {
37      if (isEmpty(this))
38          return Integer.MIN_VALUE;
```

10. Buat sintaks untuk mengimplementasikan metode *rear*() untuk mengembalikan elemen belakang dari queue tanpa menghapusnya.

```
42●int rear() {
43      if (isEmpty(this))
44          return Integer.MIN_VALUE;
45
46 return this.array[this.rear];
47 }
48 }
```

11. Jalankan program pada class sebelumnya.

```
Console ×
<terminated> TestQueue [Java Ap
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 10
Rear item is 40
```

c. Program Reverse Data pada Queue

1. Buat class baru dan namakan sesuai perintah atau yang diinginkan, lalu buat “import java.util.*;” pada baris setelah package.

```
1 package Pekan3;
2 import java.util.*;
3 public class ReverseData {
```

2. Deklarasikan tipe data queue yang akan digunakan.

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

3. Buatkan sintaks untuk menambahkan elemen ke dalam queue.

```
q.add(1);
q.add(2);
q.add(3); // [1, 2, 3]
```

4. Buat sintaks untuk menampilkan output data sebelum di-reverse.

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

5. Deklarasikan tipe data stack untuk menampung data sementara dari queue.

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

6. Buat sintaks while loop untuk men-transfer elemen dari queue q ke stack s.

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

7. Buat sintaks while loop untuk men-transfer elemen dari stack s ke queue q.

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

8. Buat sintaks untuk menampilkan output program setelah data di-reverse.

```
    System.out.println("Sesudah reverse: " + q);
}
```

9. Jalankan program.

```
Console ×
<terminated> ReverseData [Java Applica
Sebelum reverse: [1, 2, 3]
Sesudah reverse: [3, 2, 1]
```

D. KESIMPULAN

Dari praktikum yang telah dilakukan, maka dapat diambil kesimpulan bahwa "queue" adalah kumpulan elemen yang disusun secara linear menurut prinsip FIFO (First In, First Out), yang berarti elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang keluar.

Pada praktikum kali ini, kita membuat program queue dengan operasi enqueue dan dequeue. Untuk menjalankan program tersebut, dibutuhkan kelas yang berisi input untuk queue. Maka, dibuatlah kelas program yang berisikan inputan untuk queue yang telah dibuat sebelumnya agar program dapat dijalankan. Kemudian, dibuat juga program untuk me-reverse urutan data pada queue dengan cara memindahkan elemen anggota queue ke stack, kemudian dipindahkan lagi ke queue dengan urutan yang berkebalikan.