

LAPORAN PRAKTIKUM STRUKTUR DATA
QUEUE



OLEH :

JOVANTRI IMMANUEL GULO

NIM 2411532014

MATA KULIAH STRUKTUR DATA

DOSEN PENGAMPU :

Dr. Ir. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, 20 MEI 2025

A. Pendahuluan

Struktur data queue (antrian) memainkan peran penting dalam mengelola data secara terorganisir dan efisien. Queue bekerja dengan prinsip First In, First Out (FIFO), di mana elemen yang pertama masuk akan menjadi elemen yang pertama keluar. Konsep ini banyak diterapkan dalam berbagai bidang, seperti sistem antrean pelanggan, manajemen proses dalam sistem operasi, dan jaringan komputer.

B. Tujuan


Tujuan dari dilakukannya praktikum ini adalah

1. Mampu memahami cara mengimplementasikan queue.
2. Mampu menggunakan konstruktor dan method queue.
3. Mampu mengimplementasikan queue dalam kehidupan sehari-hari.

C. Langkah – langkah Pengerjaan

Berikut adalah langkah-langkah dalam pengerjaan praktikum kali ini:

1. Pertama, buat file (kelas Java) dengan nama **ContohQueue2**.

 ContohQueue2.java

2. Import package yang akan kita gunakan.

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

3. Buat public class dengan nama filenya dan juga class mainnya. Di awal, kita perlu deklarasi Queue dengan konstruktor LinkedList.

```
6 public class ContohQueue2 {
7
8=     public static void main(String[] args) {
9         Queue<Integer> q = new LinkedList<>();
10        for(int i = 0; i < 6; i++) {
```

4. Gunakan perulangan for untuk menambahkan i ke dalam antrian.

```
10        for(int i = 0; i < 6; i++) {
11            q.add(i);
12        }
```

5. Tampilkan queue, lalu hapus elemen terdepan, lalu tampilkan elemen yang dihapus.

```
13        System.out.println("Element antrian: " + q);
14        int hapus = q.remove();
15        System.out.println("Hapus elemen: " + hapus);
```

6. Intip elemen pada queue dengan menggunakan *peek*, kemudian tampilkan

```
17        int depan = q.peek();
18        System.out.println("Kepala antrian: " + depan);
```

7. Tampilkan jumlah queue.

```
19        int banyak = q.size();
20        System.out.println("Size antrian: " + banyak);
```

8. Hasil dari class nya adalah sebagai berikut.

```
Element antrian: [0, 1, 2, 3, 4, 5]
Hapus elemen: 0
[1, 2, 3, 4, 5]
Kepala antrian: 1
Size antrian: 5
```

9. Kedua, buat kelas dengan nama **InputQueue**.

10. Deklarasikan front, rear, size, capacity, dan array dengan tipe data integer.

```

4    int front, rear, size;
5    int capacity;
6    int array[];

```

11. Konstruktor ini menginisialisasi antrian dengan kapasitas tertentu, mengatur front dan size ke nol, serta rear ke kapasitas dikurangi satu. Array dibuat sesuai kapasitas yang diberikan.

```

8    public inputQueue(int capacity) {
9        this.capacity = capacity;
10       front = this.size = 0;
11       rear = capacity - 1;
12       array = new int[this.capacity];
13   }

```

12. Method ini memeriksa apakah antrian sudah penuh dengan membandingkan ukuran saat ini dengan kapasitas maksimum dan mengembalikan nilai true jika penuh atau false jika masih ada ruang.

```

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

```

13. Fungsi ini mengecek apakah antrian kosong dengan melihat apakah size bernilai nol. Jika kosong, mengembalikan true, jika ada elemen, mengembalikan false.

```

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

```

14. Method enqueue digunakan untuk menambahkan item ke dalam antrian apabila belum penuh. Posisi rear diperbarui menggunakan operasi modulo agar tetap dalam batas kapasitas, lalu item disimpan pada indeks rear. Ukuran antrian bertambah satu dan sistem mencetak konfirmasi bahwa item telah ditambahkan.

```

    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");
    }

```

15. Method dequeue menghapus elemen pertama dari antrian jika tidak kosong. Elemen di front disimpan sementara sebelum dihapus, lalu posisi front diperbarui menggunakan operasi modulo agar tetap dalam batas kapasitas. Ukuran antrian berkurang satu dan method ini mengembalikan elemen yang dihapus atau Integer.MIN_VALUE jika antrian kosong.

```

    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;
    }

```

16. Method ini mengembalikan elemen pertama dalam antrian tanpa menghapusnya. Jika antrian kosong, nilai yang dikembalikan adalah Integer.MIN_VALUE.

```

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

```

17. Method ini mengembalikan elemen terakhir dalam antrian tanpa menghapusnya. Jika antrian kosong, nilai yang dikembalikan adalah Integer.MIN_VALUE.

```

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

```

18. Ketiga, buat class Java baru dengan nama **TestQueue**.
 19. Kita menggunakan class yang telah dibuat tadi, yaitu **InputQueue**, menggunakan method **enqueue**, **front**, **rear**, dan **dequeue**.

```

1 package pekan4;
2
3 public class TestQueue {
4     public static void main(String[] args) {
5         inputQueue queue = new inputQueue(1000);
6         queue.enqueue(10);
7         queue.enqueue(20);
8         queue.enqueue(30);
9         queue.enqueue(40);
10        System.out.println("Front item is " + queue.front());
11        System.out.println("Rear item is " + queue.rear());
12        System.out.println(queue.dequeue() + " dequeued from queue");
13        System.out.println("Front item is " + queue.front());
14        System.out.println("Rear item is " + queue.rear());
15    }
16 }

```

20. Berikut adalah outputnya.

```

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

```

21. Keempat, buat class dengan nama **IterasiQueue**.
 22. Program ini menggunakan **Queue** dari **LinkedList** untuk menyimpan beberapa string dan kemudian melakukan iterasi menggunakan **Iterator**.

```

9     public static void main(String[] args) {
10        Queue<String> q = new LinkedList<>();
11        q.add("Praktikum");
12        q.add("Struktur");
13        q.add("Data");
14        q.add("Dan");
15        q.add("Algoritma");

```

23. Bagian ini membuat antrian bertipe **String** menggunakan **LinkedList** dan menambahkan lima elemen ke dalamnya. Iterator digunakan untuk mengakses elemen dalam antrian satu per satu menggunakan **hasNext()**, lalu mencetak elemen yang sedang diakses menggunakan **next()**.

```

17         Iterator<String> iterator = q.iterator();
18         while(iterator.hasNext()) {
19             System.out.println(iterator.next() + " ");
20         }

```

24. Berikut adalah outputnya.

```

Praktikum
Struktur
Data
Dan
Algoritma

```

25. Kelima, buat class dengan nama **ReverseData**.

26. Program ini membalik urutan elemen dalam **Queue** dengan bantuan **Stack**, karena struktur stack menggunakan prinsip Last In, First Out (LIFO).

```

10         Queue<Integer> q = new LinkedList<Integer>();
11         q.add(1);
12         q.add(2);
13         q.add(3);
14         System.out.println("Sebelum reverse: " + q);

```

27. Bagian ini membuat antrian bertipe **Integer** menggunakan **LinkedList** dan menambahkan tiga elemen. Elemen awal ditampilkan sebelum proses pembalikan.

```

16         Stack<Integer> s = new Stack<Integer>();
17         while(!q.isEmpty()) {
18             s.push(q.remove());
19         }

```

28. Elemen dari antrian dipindahkan ke dalam stack satu per satu. Saat **remove()** dipanggil, elemen dikeluarkan dari antrian dan ditambahkan ke stack menggunakan **push()**. Elemen dalam stack dipindahkan kembali ke antrian menggunakan **pop()**, sehingga urutannya terbalik.

```

20         while(!s.isEmpty()) {
21             q.add(s.pop());
22         }
23         System.out.println("Sesudah reverse: " + q);
24     }

```

Elemen yang telah dibalik ditampilkan sebagai output akhir.

29. Berikut adalah outputnya.

```

Sebelum reverse: [1, 2, 3]
Sesudah reverse: [3, 2, 1]

```

D. Kesimpulan

Dari praktikum yang telah dilakukan, maka dapat diambil kesimpulan bahwa penggunaan queue (antrian) dalam struktur data sangat efektif dalam menangani pemrosesan pelanggan secara berurutan. Dengan menerapkan konsep First In, First Out (FIFO), sistem dapat memastikan bahwa setiap pelanggan dilayani sesuai urutan kedatangannya.