LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 4: PEMROGRAMAN ANTRIAN (QUEUE) DENGAN JAVA

OLEH

Abdullah Al Ramadhani (2411533016)

DOSEN PENGAMPU

DR. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. Pendahuluan

Dalam dunia pemrograman, struktur data berperan penting untuk mengelola dan memproses data secara efisien. Salah satu struktur data yang umum digunakan adalah **queue** atau antrian. Queue merupakan struktur data linear yang menerapkan prinsip First In First Out (FIFO), yaitu elemen yang pertama masuk akan menjadi yang pertama keluar. Dalam bahasa pemrograman Java, implementasi dan penggunaan queue dapat dilakukan melalui beberapa cara, baik secara manual dengan menggunakan array, maupun dengan memanfaatkan kelas bawaan Java seperti `LinkedList` dan `Queue`. Selain itu, kode program juga biasanya mencakup berbagai operasi dasar pada queue, seperti **enqueue**, **dequeue**, **peek**, serta iterasi antrian menggunakan **Iterator**, bahkan termasuk teknik membalik urutan elemen queue dengan bantuan struktur data **stack**.

B. Tujuan

Tujuan dari praktikum ini adalah sebagai berikut:

1. Memahami konsep dasar dari struktur data queue (antrian), termasuk cara kerjanya yang mengikuti prinsip First In First Out (FIFO), serta berbagai operasi dasar yang bisa dilakukan, seperti enqueue, dequeue, peek, dan lainnya.
2. Mampu mengimplementasikan queue dalam bahasa pemrograman Java, baik dengan cara manual menggunakan array, maupun dengan memanfaatkan pustaka bawaan Java seperti LinkedList, Queue, dan Stack.
3. Menganalisis bagaimana queue bekerja dalam berbagai situasi penggunaan, seperti mengelola antrian elemen, membalik urutan data dalam antrian, hingga menelusuri elemen menggunakan iterator.

C. Langkah-langkah Pengerjaan

a. Input Queue
    1. Buatlah sebuah package dan class baru dengan nama package "Pekan4" dan nama class "inputQueue". Class ini akan berfungsi untuk menyimpan elemen secara berurutan sesuai urutan masuk (FIFO) menggunakan array, serta menyediakan method-method dasar seperti enqueue, dequeue, front, rear, dan pengecekan apakah queue dalam kondisi penuh atau kosong.

2. Deklarasikan variabel-variabel utama yang dibutuhkan dalam struktur queue. Beberapa variabel tersebut antara lain front sebagai penunjuk elemen paling depan, rear sebagai penunjuk elemen terakhir, size untuk menyimpan jumlah elemen saat ini, capacity sebagai batas maksimal kapasitas queue, dan array[] sebagai array bertipe int untuk menyimpan data.

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

3. Buat constructor untuk class inputQueue yang menerima parameter capacity. Konstruktor ini akan digunakan untuk mengatur kapasitas queue, mengatur posisi awal front dan rear, serta membuat array baru sebagai tempat penyimpanan elemen queue.

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

4. Buat method isFull() yang berfungsi untuk mengecek apakah queue sudah mencapai kapasitas maksimal. Method ini akan mengembalikan nilai true jika jumlah elemen saat ini sama dengan kapasitas yang telah ditentukan.

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

5. Buat method isEmpty() untuk mengecek apakah queue dalam keadaan kosong. Method ini akan mengembalikan true apabila nilai size sama dengan 0.

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

6. Buat method enqueue() yang digunakan untuk menambahkan elemen ke dalam queue. Method ini akan terlebih dahulu mengecek apakah queue penuh dengan memanggil isFull(). Jika masih ada ruang, maka nilai rear akan digeser ke posisi berikutnya dengan logika sirkular, elemen disimpan ke dalam array di indeks tersebut, dan size akan bertambah.

```
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");
}
```

7. Selanjutnya buat method dequeue() yang bertugas menghapus elemen dari bagian depan queue. Method ini akan mengecek terlebih dahulu apakah queue kosong menggunakan isEmpty(), lalu mengambil nilai pada posisi front, memindahkan front ke indeks berikutnya secara sirkular, mengurangi nilai size, dan mengembalikan elemen yang dihapus.

```
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;
    }
```

8. Terakhir, buat method front() dan rear() yang berfungsi untuk menampilkan elemen terdepan dan elemen terakhir dalam queue. Kedua method ini hanya mengembalikan nilai tanpa menghapusnya. Jika queue dalam keadaan kosong, method akan mengembalikan nilai Integer.MIN_VALUE.

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

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

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

b. Program TestQueue

1. Buatlah sebuah class baru bernama "TestQueue" yang digunakan untuk menguji implementasi dari class inputQueue.

2. Buat objek dari class queue dan tambahkan beberapa elemen ke dalamnya menggunakan method enqueue(). Gunakan konstruktor untuk membuat queue dengan kapasitas sebesar 1000, lalu masukkan beberapa data ke dalam antrian tersebut.

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

3. Tampilkan elemen pertama dan terakhir dalam antrian menggunakan method front() dan rear(). Kedua method ini hanya menampilkan nilai tanpa menghapus elemen dari queue.

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

4. Hapus satu elemen dari queue dengan menggunakan method dequeue() dan tampilkan nilai yang dikeluarkan. Method ini akan mengambil dan menghapus elemen yang berada di posisi terdepan.

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

5. Setelah proses penghapusan, tampilkan kembali elemen terdepan dan elemen terakhir di queue. Gunakan kembali method front() dan rear() untuk melihat posisi elemen yang baru setelah penghapusan.

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

6. Jalankan program, dan perhatikan hasil output-nya. Output yang ditampilkan akan menunjukkan proses dasar dari struktur data queue yang telah dibuat, mulai dari penambahan elemen ke dalam antrian, melihat posisi elemen pertama dan terakhir, hingga proses penghapusan elemen dari antrian.

```
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 dequeue from queue
Front item is 20
Rear item is 40
```

c. Program ReverseData

1. Buat sebuah class dengan nama "ReverseData" yang memiliki method main(). Di dalamnya, kamu akan membalik urutan data dalam queue dengan bantuan stack. Jangan lupa untuk menyertakan import library Java yang dibutuhkan, seperti Queue, LinkedList, dan Stack, karena Queue dan LinkedList digunakan sebagai struktur antrian, sedangkan Stack akan membantu proses pembalikan data.

2. Inisialisasi sebuah queue bertipe Integer menggunakan LinkedList sebagai implementasinya. Tambahkan beberapa angka ke dalam antrian, misalnya 1, 2, dan 3, dengan method add().

```
Queue<Integer> q = new LinkedList<Integer>();
q.add(1);
q.add(2);
q.add(3); // [1, 2, 3]
```

3. Cetak isi antrian sebelum dibalik, menggunakan System.out.println(). Hasilnya akan menampilkan data sesuai urutan input, yaitu [1, 2, 3].

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

4. Buat stack bertipe Integer. Stack ini digunakan untuk membalik data karena elemen terakhir yang masuk akan keluar lebih dulu (prinsip LIFO – Last In First Out).

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

5. Pindahkan semua elemen dari antrian ke stack. Gunakan perulangan untuk menghapus elemen satu per satu dari queue menggunakan remove() dan langsung memasukkannya ke stack dengan push(). Setelah proses ini selesai, stack akan berisi [3, 2, 1].

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

6. Kembalikan elemen dari stack ke queue. Dengan cara ini, urutan akan terbalik karena elemen yang terakhir masuk ke stack akan menjadi yang pertama masuk ke queue. Gunakan pop() untuk mengambil data dari stack lalu add() untuk memasukkannya kembali ke antrian. Hasil akhirnya, queue akan berisi [3, 2, 1].

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

7. Tampilan isi antrian setelah dibalik untuk melihat perubahan urutannya.

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

8. Jalankan program dan periksa hasil output-nya. Tampilan akan menunjukkan proses pembalikan data dari antrian dengan memanfaatkan stack sebagai media bantu.

```
Sebelum reverse [1, 2, 3]
sesudah reverse = [3, 2, 1]
```

d. Program IteratorQueue

1. Mulailah dengan membuat class baru bernama IteratorQueue. Class ini nantinya digunakan untuk menampilkan isi antrian menggunakan iterator, yaitu sebuah alat bantu dalam Java yang memungkinkan kita menelusuri elemen-elemen dalam struktur data koleksi. Sebelum lanjut, pastikan kamu mengimpor beberapa library dari java.util yang diperlukan: Queue untuk membuat antrian, LinkedList sebagai implementasinya, dan Iterator untuk proses penelusuran elemen.

2. Buat sebuah objek queue bertipe String, lalu tambahkan beberapa data ke dalamnya. Gunakan LinkedList sebagai implementasi dari Queue, dan tambahkan elemen string satu per satu menggunakan method add(). Kamu bebas mengisi data apa pun, misalnya nama-nama buah, kota, atau apapun yang kamu suka.

```
Queue<String> q = new LinkedList<>();
q.add("Praktikum");
q.add("Struktur");
q.add("Data");
q.add("Dan");
q.add("Algoritma");
```

3. Setelah queue terisi, buatlah objek iterator dari queue tersebut. Caranya cukup dengan memanggil method iterator() dari objek queue yang sudah dibuat tadi. Objek iterator ini akan kamu gunakan untuk membaca elemen dalam antrian satu per satu tanpa mengubah isinya.

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

4. Gunakan struktur perulangan while untuk mencetak isi dari antrian. Selama iterator masih memiliki elemen yang bisa diakses (hasNext() bernilai true), ambil elemen tersebut dengan next() dan tampilkan ke layar. Dengan cara ini, seluruh isi queue akan bisa ditampilkan secara berurutan.

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

5. Jalankan programnya, dan hasilnya akan memperlihatkan seluruh elemen dalam queue yang ditelusuri menggunakan iterator. Ini adalah cara yang sangat efisien jika kamu hanya ingin membaca isi antrian tanpa memodifikasinya.

```
<terminated> IteratorQueue [Java Application] C:\Users\Lenc
Praktikum Struktur Data Dan Algoritma
```

e. ContohQueue2

1. Mulai dengan membuat class baru bernama ContohQueue2. Class ini akan jadi program utama yang menunjukkan bagaimana cara kerja dasar dari struktur data antrian (queue), mulai dari menambahkan data, menghapus, melihat elemen paling depan, sampai menghitung jumlah elemen yang ada. Jangan lupa juga untuk mengimpor library dari java.util yang dibutuhkan, seperti Queue dan LinkedList.

2. Di dalam class tersebut, buatlah objek queue bertipe Integer, dan gunakan LinkedList sebagai implementasinya. Setelah itu, isi antrian dengan angka menggunakan perulangan for. Misalnya dari angka 0 sampai 5, lalu tambahkan setiap angka tersebut ke dalam queue dengan method add().

```
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);
```

3. Setelah semua elemen dimasukkan, gunakan System.out.println() untuk menampilkan isi antrian. Di tahap ini, kamu bisa melihat urutan data yang sudah masuk ke queue.

```
// menampilkan isi antrian.
System.out.println("Elemen Antrian " + q);
```

4. Selanjutnya, hapus satu elemen dari antrian menggunakan method remove(). Karena queue mengikuti prinsip FIFO (First-In-First-Out), elemen pertama yang masuk akan menjadi yang pertama keluar.

```
// UNTUK MENGHAPUS KEPALA ANTRIAN
int hapus = q.remove();
System.out.println("Hapus elemen = " + hapus);
```

5. Tampilkan lagi isi queue setelah penghapusan tadi. Kini, queue hanya akan berisi lima elemen karena satu elemen paling depan telah dikeluarkan.

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

6. Untuk melihat elemen yang sekarang berada di posisi depan tanpa menghapusnya, gunakan method peek(). Ini berguna kalau kamu ingin tahu isi terdepan dari antrian tanpa mengubah isinya.

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

7. Terakhir, tampilkan jumlah elemen yang tersisa di dalam queue. Gunakan method size() untuk mengetahui berapa banyak data yang masih ada setelah penghapusan.

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

8. Jalankan programmu, dan perhatikan hasil output-nya. Kamu akan melihat bagaimana data ditambahkan, dikeluarkan, dicek posisinya, serta dihitung jumlahnya secara langsung melalui queue.

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

D. Kesimpulan

Dari seluruh pembahasan dan implementasi kode yang sudah dilakukan, kita bisa melihat bahwa struktur data queue punya peran penting dalam pengelolaan data yang membutuhkan urutan pemrosesan, khususnya dengan prinsip First In First Out (FIFO). Melalui praktik langsung menggunakan Java, baik dengan pendekatan manual lewat array, maupun dengan memanfaatkan kelas-kelas bawaan seperti LinkedList dan Queue. Kita jadi lebih paham bagaimana queue bekerja dan mengapa struktur ini sangat berguna saat harus menangani antrian data.

Beberapa operasi dasar seperti enqueue, dequeue, peek, dan iterasi menggunakan iterator terbukti bisa menunjang berbagai kebutuhan logika pemrograman. Bahkan saat dibutuhkan, queue juga bisa dimanipulasi, misalnya membalik urutan elemen dengan bantuan struktur data lain seperti stack. Penerapan langsung seperti ini bukan hanya memperkuat konsep teoritis, tapi juga menunjukkan bagaimana struktur queue benar-benar bisa diterapkan di banyak kasus nyata dalam pengembangan program.