MODUL 6 LAPORAN
PRAKTIKUM STRUKTUR DATA

Disusun oleh :

Nama : Fidelia Ping

NIM : 245410012

Kelas : Informatika 1

PROGRAM STUDI INFORMATIKA

PROGRAM SARJANA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA

YOGYAKARTA

2025

MODUL 6
PEMANFAATAN ARRAY/LARIK SEBAGAI STACK (TUMPUKAN) DAN QUEUE (ANTRIAN)

A. TUJUAN PRAKTIKUM

Mahasiswa dapat mengimplementasikan tumpukan dan antrian untuk berbagai keperluan dengan menggunakan bahasa pemrograman Java

B. DASAR TEORI

Stack (Tumpukan)

Tumpukan atau stack adalah teknik peletakan data dimana seolah-olah data yang satu diletakkan 'diatas' data yang lain. Dalam tumpukan semua penyisipan dan penghapusan data dilakukan hanya melalui satu pintu yang disebut top (puncak) tumpukan. Tumpukan dapat dibayangkan seperti tumpukan buku dimana hanya buku teratas yang dapat diperoleh kembali dengan satu langkah. Buku-buku yang terdapat dibawah hanya dapat diambil setelah buku yang berada diatasnya diambil (dikeluarkan) terlebih dahulu. Dengan bentuk yang demikian tumpukan dapat dikatakan sebagai struktur data yang bersifat LIFO (Last In First Out). Dalam tumpukan terdapat 2 buah operasi data yaitu (1) operasi push dan (2) operasi pop. Operasi push adalah operasi untuk menambahkan sebuah data ke dalam tumpukan, sedangkan operasi pop adalah operasi untuk mengambil (menghapus) data dari dalam tumpukan.

Queue (Antrian)

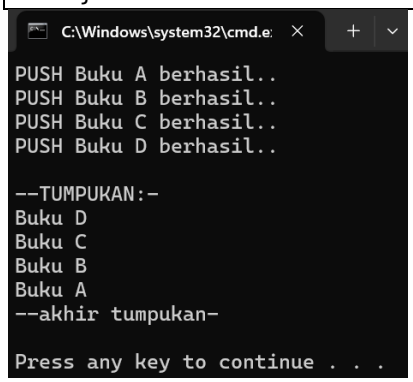
Antrian atau Queue adalah teknik peletakan data dimana seolah-olah data yang satu diletakkan 'dibelakang' data yang lain. Dalam antrian semua penyisipan dan penghapusan data dilakukan melalui dua pintu yaitu belakang dan depan. Pintu belakang digunakan sebagai jalan masuk data, sedangkan pintu depan digunakan sebagai jalan keluar data. Antrian dapat dibayangkan seperti antrian para perima zakat. Orang yang ingin mendapatkan zakat harus mengantri dengan cara masuk melalui sisi belakang. Apabila seorang penerima telah terlayani maka dia harus meninggalkan antrian melalui sisi depan. Dengan bentuk yang demikian antrian dapat dikatakan sebagai struktur data yang bersifat FIFO (First In First Out) artinya pengantri yang datang paling awal akan keluar paling dahulu. Dalam antrian terdapat 2 buah operasi data yaitu (1) operasi enQueue dan (2) operasi deQueue. Operasi enQueue adalah operasi untuk menambahkan sebuah data ke dalam antrian dan dilakukan pada pintu belakang, sedangkan operasi deQueue adalah operasi untuk mengambil (menghapus) data dari dalam antrian dan dilakukan pada pintu depan

C. PEMBAHASAN LISTING

Praktik 1

```
public class program_tumpukan
{ public static int N = 5;
```

```
    public static int atas = -1;
    public static void push (String tumpukan[], String data)
    { if (atas == N-1) //jika tumpukan penuh
    { System.out.println("maap, tumpukan penuh, PUSH " + data
    + " tidak dapat dilakukan");
    }
    else //jika tumpukan tidak penuh
    { atas = atas + 1;
    tumpukan[atas] = data;
    System.out.println("PUSH " + data + " berhasil..");
    }
    }
    public static String POP (String tumpukan[])
    { String hasil;
    if (atas < 0 ) //jika tumpukan kosong
    { hasil = "TUMPUKAN KOSONG, POP GAGAL DILAKUKAN";
    }
    else //jika tumpukan tidak kosong
    { hasil = tumpukan[atas];
    atas = atas - 1;
    }
    return (hasil);
    }
    public static void lihatTumpukan(String tumpukan[])
    { System.out.println("");
    System.out.println("--TUMPUKAN:-");
    for (int i=atas; i>=0; i--)
    { System.out.println(tumpukan[i]);
    }
    System.out.println("--akhir tumpukan-");
    System.out.println("");
    }
    public static void main (String[] args)
    { String tumpukan[] = new String[10];
    push (tumpukan, "Buku A");
    push (tumpukan, "Buku B");
    push (tumpukan, "Buku C");
    push (tumpukan, "Buku D");
    lihatTumpukan(tumpukan);
    }
    }
```



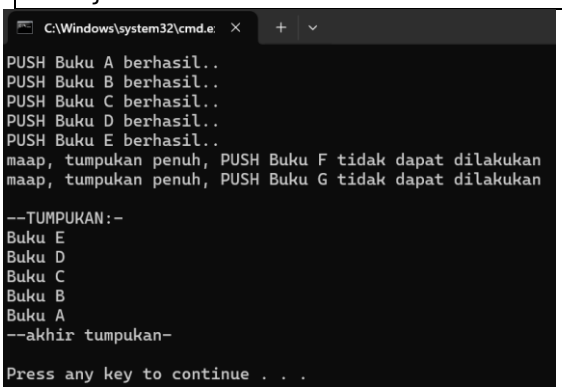
Pembahasan : Ketika program dijalankan, proses push dilakukan berurutan untuk buku A, B, C, dan D. Karena ukuran maksimum tumpukan (N) ditetapkan sebesar **5**, maka keempat buku tersebut **Buku A, Buku B, Buku C, dan Buku D** semuanya

berhasil dimasukkan ke dalam tumpukan, karena kapasitasnya belum penuh. Setiap kali operasi `push()` dilakukan, nilai variabel `atas` bertambah satu, dimulai dari -1 menjadi 0 (saat Buku A masuk), kemudian 1 (Buku B), 2 (Buku C), dan akhirnya 3 (Buku D). Dengan demikian, setelah semua buku dimasukkan, posisi "atas" berada di indeks 3, yaitu pada Buku D, yang menjadi elemen paling atas dalam tumpukan, sesuai konsep LIFO (Last In, First Out) di mana elemen terakhir yang dimasukkan berada di puncak tumpukan.

Praktik 2

```
public class program_tumpukan
{ public static int N = 5;
public static int atas = -1;
public static void push (String tumpukan[], String data)
{ if (atas == N-1) //jika tumpukan penuh
{ System.out.println("maap, tumpukan penuh, PUSH " + data
+ " tidak dapat dilakukan");
}
else //jika tumpukan tidak penuh
{ atas = atas + 1;
tumpukan[atas] = data;
System.out.println("PUSH " + data + " berhasil..");
}
}
public static String POP (String tumpukan[])
{ String hasil;
if (atas < 0 ) //jika tumpukan kosong
{ hasil = "TUMPUKAN KOSONG, POP GAGAL DILAKUKAN";
}
else //jika tumpukan tidak kosong
{ hasil = tumpukan[atas];
atas = atas - 1;
}
return (hasil);
}
public static void lihatTumpukan(String tumpukan[])
{ System.out.println("");
System.out.println("--TUMPUKAN:-");
for (int i=atas; i>=0; i--)
{ System.out.println(tumpukan[i]);
}
System.out.println("--akhir tumpukan-");
System.out.println("");
}
public static void main (String[] args)
{ String tumpukan[] = new String[10];
push (tumpukan, "Buku A");
push (tumpukan, "Buku B");
push (tumpukan, "Buku C");
push (tumpukan, "Buku D");
push (tumpukan, "Buku E");
push (tumpukan, "Buku F");
push (tumpukan, "Buku G");
lihatTumpukan(tumpukan);
}
```

```
    }
```



Pembahasan : Ketika program dijalankan, proses push dilakukan berurutan untuk Buku A hingga Buku G. Karena kapasitas maksimum tumpukan ditetapkan dengan nilai `N = 5`, maka hanya **lima buku pertama** yaitu **Buku A, Buku B, Buku C, Buku D, dan Buku E** yang **berhasil dimasukkan ke dalam tumpukan**. Sementara **Buku F dan Buku G gagal di-push**, karena pada saat itu nilai variabel `atas` sudah mencapai `N-1` (yaitu 4), menandakan bahwa tumpukan sudah penuh. Akibatnya, setiap kali mencoba menambahkan data baru, program menampilkan pesan **"maap, tumpukan penuh, PUSH Buku F tidak dapat dilakukan"** dan hal yang sama juga terjadi pada Buku G. Hal ini terjadi karena pada bagian kode berikut:

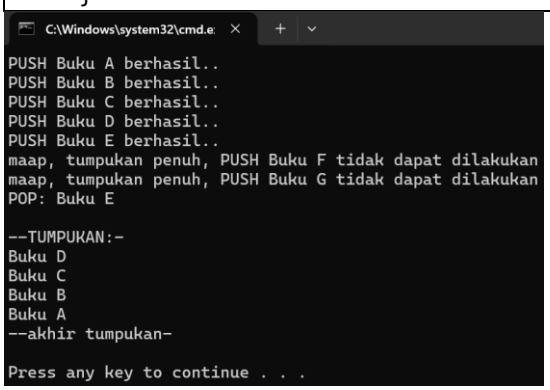
```
if (atas == N-1)
    System.out.println("maap, tumpukan penuh, PUSH " + data + " tidak dapat dilakukan");
```

baris tersebut berfungsi sebagai **pengecekan kondisi penuh (overflow)**, yang mencegah penambahan elemen baru melebihi kapasitas maksimum tumpukan. Jadi, penyebab buku F dan G gagal dimasukkan adalah karena **tumpukan sudah mencapai batas maksimal (N = 5)** dan program memang dirancang untuk **menolak data baru ketika penuh**.

Praktik 3

```
    public class program_tumpukan
    { public static int N = 5;
    public static int atas = -1;
    public static void push (String tumpukan[], String data)
    { if (atas == N-1) //jika tumpukan penuh
    { System.out.println("maap, tumpukan penuh, PUSH " + data
    + " tidak dapat dilakukan");
    }
    else //jika tumpukan tidak penuh
    { atas = atas + 1;
    tumpukan[atas] = data;
    System.out.println("PUSH " + data + " berhasil..");
    }
    }
    public static String POP (String tumpukan[])
    { String hasil;
    if (atas < 0 ) //jika tumpukan kosong
    { hasil = "TUMPUKAN KOSONG, POP GAGAL DILAKUKAN";
```

```
    }
    else //jika tumpukan tidak kosong
    { hasil = tumpukan[atas];
    atas = atas - 1;
    }
    return (hasil);
    }
    public static void lihatTumpukan(String tumpukan[])
    { System.out.println("");
    System.out.println("--TUMPUKAN:-");
    for (int i=atas; i>=0; i--)
    { System.out.println(tumpukan[i]);
    }
    System.out.println("--akhir tumpukan-");
    System.out.println("");
    }
    public static void main (String[] args)
    { String tumpukan[] = new String[10];
    push (tumpukan, "Buku A");
    push (tumpukan, "Buku B");
    push (tumpukan, "Buku C");
    push (tumpukan, "Buku D");
    push (tumpukan, "Buku E");
    push (tumpukan, "Buku F");
    push (tumpukan, "Buku G");
    System.out.println("POP: " + POP(tumpukan));
    lihatTumpukan(tumpukan);
    }
    }
```



Pembahasan : Ketika program dijalankan, proses **push** memasukkan Buku A hingga Buku G, namun karena kapasitas maksimum tumpukan hanya **5 elemen (N = 5)**, maka hanya **Buku A sampai Buku E** yang berhasil masuk. Buku F dan Buku G gagal dimasukkan karena tumpukan sudah penuh. Setelah itu, program menjalankan perintah POP, yang berarti mengambil atau menghapus elemen paling atas dari tumpukan. Maka, **buku yang ter-pop adalah Buku E**, karena Buku E merupakan **elemen terakhir yang dimasukkan** ke dalam tumpukan sesuai prinsip **LIFO (Last In, First Out)**, yaitu data yang terakhir masuk akan keluar terlebih dahulu.

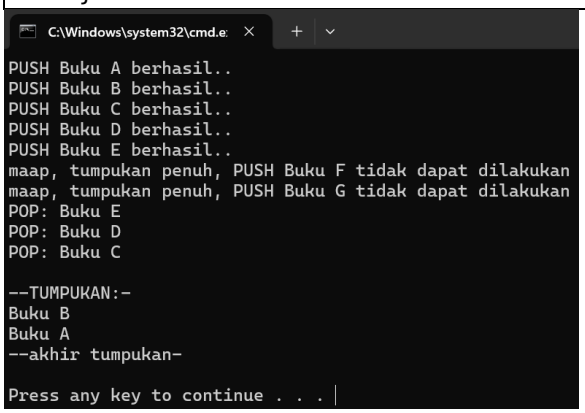
Setelah operasi POP dilakukan, **isi tumpukan berkurang satu elemen**, sehingga kini hanya tersisa **Buku D, Buku C, Buku B, dan Buku A**. Hal ini terjadi karena di dalam method POP, terdapat baris kode:

`atas = atas - 1;`
yang berfungsi **menurunkan posisi indeks "atas"** setelah data paling atas dihapus. Dengan demikian, variabel **`atas` yang sebelumnya bernilai 4 (menunjuk ke Buku E)** kini **berubah menjadi 3 (menunjuk ke Buku D)**. Jadi, kondisi tumpukan saat ini berkurang satu elemen di bagian atas, dan "atas" sekarang berada di posisi **indeks 3**, yaitu pada Buku D.

Praktik 4

```
public class program_tumpukan
{ public static int N = 5;
public static int atas = -1;
public static void push (String tumpukan[], String data)
{ if (atas == N-1) //jika tumpukan penuh
{ System.out.println("maap, tumpukan penuh, PUSH " + data
+ " tidak dapat dilakukan");
}
else //jika tumpukan tidak penuh
{ atas = atas + 1;
tumpukan[atas] = data;
System.out.println("PUSH " + data + " berhasil..");
}
}
public static String POP (String tumpukan[])
{ String hasil;
if (atas < 0 ) //jika tumpukan kosong
{ hasil = "TUMPUKAN KOSONG, POP GAGAL DILAKUKAN";
}
else //jika tumpukan tidak kosong
{ hasil = tumpukan[atas];
atas = atas - 1;
}
return (hasil);
}
public static void lihatTumpukan(String tumpukan[])
{ System.out.println("");
System.out.println("--TUMPUKAN:-");
for (int i=atas; i>=0; i--)
{ System.out.println(tumpukan[i]);
}
System.out.println("--akhir tumpukan-");
System.out.println("");
}
public static void main (String[] args)
{ String tumpukan[] = new String[10];
push (tumpukan, "Buku A");
push (tumpukan, "Buku B");
push (tumpukan, "Buku C");
push (tumpukan, "Buku D");
push (tumpukan, "Buku E");
push (tumpukan, "Buku F");
push (tumpukan, "Buku G");
System.out.println("POP: " + POP(tumpukan));
System.out.println("POP: " + POP(tumpukan));
System.out.println("POP: " + POP(tumpukan));
lihatTumpukan(tumpukan);
}
```

```
    }
```



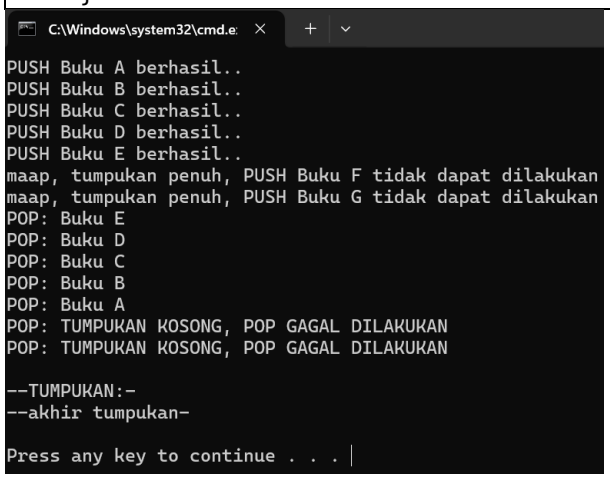
Pembahasan : Ketika program dijalankan, proses **push** memasukkan data berurutan dari **Buku A** hingga **Buku G**, namun karena kapasitas maksimum tumpukan (`N = 5`), hanya **Buku A, B, C, D, dan E** yang berhasil masuk. Buku F dan Buku G gagal karena tumpukan sudah penuh. Setelah itu, dilakukan **tiga kali operasi** `POP()`, yang artinya tiga buku paling atas diambil (dikeluarkan) dari tumpukan secara berurutan. Hasilnya, **buku yang ter-pop adalah Buku E, Buku D, dan Buku C**, karena prinsip kerja tumpukan adalah **LIFO (Last In, First Out)** data terakhir yang dimasukkan akan keluar terlebih dahulu.

Setelah tiga kali `POP`, isi tumpukan kini **tersisa Buku B dan Buku A**. Kondisi ini berbeda dari percobaan sebelumnya karena jumlah data di dalam tumpukan berkurang tiga elemen akibat operasi penghapusan. Perubahan ini terjadi karena setiap kali `POP()` dijalankan, variabel **`atas` dikurangi satu (`atas = atas - 1`),** sehingga posisi atas bergeser turun. Sebelumnya `atas` bernilai 4 (menunjuk ke Buku E), setelah tiga kali `POP`, nilai `atas` kini menjadi **1**, yang berarti posisi "atas" sekarang berada pada **Buku B** elemen teratas dalam tumpukan yang tersisa.

Praktik 5

```
    public class program_tumpukan
    { public static int N = 5;
    public static int atas = -1;
    public static void push (String tumpukan[], String data)
    { if (atas == N-1) //jika tumpukan penuh
    { System.out.println("maap, tumpukan penuh, PUSH " + data
    + " tidak dapat dilakukan");
    }
    else //jika tumpukan tidak penuh
    { atas = atas + 1;
    tumpukan[atas] = data;
    System.out.println("PUSH " + data + " berhasil..");
    }
    }
    public static String POP (String tumpukan[])
    { String hasil;
    if (atas < 0 ) //jika tumpukan kosong
    { hasil = "TUMPUKAN KOSONG, POP GAGAL DILAKUKAN";
```

```
    }
    else //jika tumpukan tidak kosong
    { hasil = tumpukan[atas];
    atas = atas - 1;
    }
    return (hasil);
    }
    public static void lihatTumpukan(String tumpukan[])
    { System.out.println("");
    System.out.println("--TUMPUKAN:-");
    for (int i=atas; i>=0; i--)
    { System.out.println(tumpukan[i]);
    }
    System.out.println("--akhir tumpukan-");
    System.out.println("");
    }
    public static void main (String[] args)
    { String tumpukan[] = new String[10];
    push (tumpukan, "Buku A");
    push (tumpukan, "Buku B");
    push (tumpukan, "Buku C");
    push (tumpukan, "Buku D");
    push (tumpukan, "Buku E");
    push (tumpukan, "Buku F");
    push (tumpukan, "Buku G");
    System.out.println("POP: " + POP(tumpukan));
    System.out.println("POP: " + POP(tumpukan));
    System.out.println("POP: " + POP(tumpukan));
    System.out.println("POP: " + POP(tumpukan));
    System.out.println("POP: " + POP(tumpukan));
    System.out.println("POP: " + POP(tumpukan));
    System.out.println("POP: " + POP(tumpukan));
    lihatTumpukan(tumpukan);
    }
    }
```



Pembahasan : Ya, pada program tersebut **ada proses `POP` yang gagal dilakukan**. Hal ini terjadi karena jumlah pemanggilan `POP()` dalam program lebih banyak daripada jumlah data yang berhasil masuk ke dalam tumpukan.

Secara rinci, hanya **5 buku** yang berhasil di-*push* ke dalam tumpukan, yaitu **Buku A, Buku B, Buku C, Buku D, dan Buku E**, karena kapasitas maksimum tumpukan (`N`

= 5). Buku F dan Buku G gagal dimasukkan sebab tumpukan sudah penuh. Namun dalam program, operasi POP() dipanggil **sebanyak 7 kali**, padahal isi tumpukan hanya 5 elemen.

Akibatnya, 5 operasi POP pertama berhasil mengeluarkan seluruh isi tumpukan secara berurutan dari atas ke bawah (**Buku E, Buku D, Buku C, Buku B, dan Buku A**), tetapi saat POP() dijalankan untuk ke-6 dan ke-7 kalinya, **tumpukan sudah kosong** variabel atas bernilai **-1**. Karena kondisi ini, program menampilkan pesan:

TUMPUKAN KOSONG, POP GAGAL DILAKUKAN

Hal ini terjadi karena bagian kode berikut:

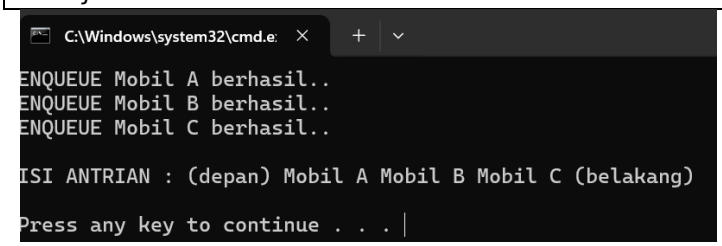
```
if (atas < 0)
    hasil = "TUMPUKAN KOSONG, POP GAGAL DILAKUKAN";
```

berfungsi untuk **mencegah pengambilan data dari tumpukan kosong (underflow).** Jadi, kegagalan POP disebabkan oleh kondisi tumpukan kosong, di mana tidak ada lagi data yang dapat dikeluarkan.

Praktik 6

```
public class program_antrian {
    public static int N = 5;
    public static int belakang = -1;

    public static void ENQUEUE(String antrian[], String data) {
        if (belakang == N - 1) { // jika antrian penuh
            System.out.println("Maaf, antrian penuh, ENQUEUE " + data +
" tidak dapat dilakukan");
        } else { // jika antrian tidak penuh
            belakang = belakang + 1;
            antrian[belakang] = data;
            System.out.println("ENQUEUE " + data + " berhasil..");
        }
    }

    public static String DEQUEUE(String antrian[]) {
        String hasil;
        if (belakang < 0) { // jika antrian kosong
            hasil = "ANTRIAN KOSONG, DEQUEUE GAGAL DILAKUKAN";
        } else { // jika antrian tidak kosong
            hasil = antrian[0];
            // menggeser data ke depan
            for (int i = 0; i <= belakang - 1; i++) {
                antrian[i] = antrian[i + 1];
            }
            belakang = belakang - 1;
        }
        return hasil;
    }

    public static void lihatAntrian(String antrian[]) {
        System.out.println(); // perbaikan di sini
        System.out.print("ISI ANTRIAN : (depan)");
        for (int i = 0; i <= belakang; i++) {
            System.out.print(" " + antrian[i]);
```

```
            }
            System.out.println(" (belakang)");
            System.out.println();
        }

        public static void main(String[] args) {
            String antrian[] = new String[5];
            ENQUEUE(antrian, "Mobil A");
            ENQUEUE(antrian, "Mobil B");
            ENQUEUE(antrian, "Mobil C");
            lihatAntrian(antrian);
        }
    }
```

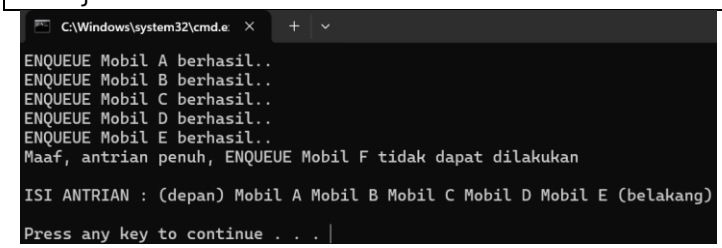


Pembahasan : Setelah program dijalankan, terlihat bahwa proses **ENQUEUE** berhasil menambahkan tiga data yaitu Mobil A, Mobil B, dan Mobil C ke dalam antrian, sehingga kondisi antrian saat ini berisi ketiga mobil tersebut dalam urutan masuk (depan ke belakang: Mobil A, Mobil B, Mobil C). Tidak ada data yang gagal masuk karena kapasitas maksimum antrian sebesar 5 elemen belum tercapai, sehingga kondisi `belakang == N - 1` belum terpenuhi. Nilai variabel `belakang` saat ini adalah 2, menandakan posisi terakhir data dalam antrian berada pada indeks ke-2. Hal ini menunjukkan bahwa program berjalan sesuai prinsip **FIFO (First In First Out),** di mana data pertama yang masuk akan berada di posisi depan dan akan keluar terlebih dahulu.

Praktik 7

```
    public class program_antrian {
        public static int N = 5;
        public static int belakang = -1;

        public static void ENQUEUE(String antrian[], String data) {
            if (belakang == N - 1) { // jika antrian penuh
                System.out.println("Maaf, antrian penuh, ENQUEUE " + data +
    " tidak dapat dilakukan");
            } else { // jika antrian tidak penuh
                belakang = belakang + 1;
                antrian[belakang] = data;
                System.out.println("ENQUEUE " + data + " berhasil..");
            }
        }

        public static String DEQUEUE(String antrian[]) {
            String hasil;
            if (belakang < 0) { // jika antrian kosong
                hasil = "ANTRIAN KOSONG, DEQUEUE GAGAL DILAKUKAN";
            } else { // jika antrian tidak kosong
```

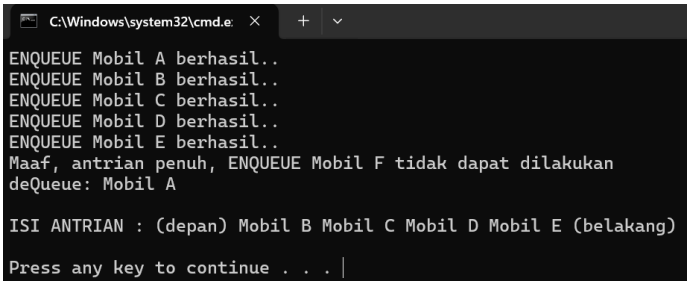
```
                hasil = antrian[0];
                // menggeser data ke depan
                for (int i = 0; i <= belakang - 1; i++) {
                    antrian[i] = antrian[i + 1];
                }
                belakang = belakang - 1;
            }
            return hasil;
        }

        public static void lihatAntrian(String antrian[]) {
            System.out.println(); // perbaikan di sini
            System.out.print("ISI ANTRIAN : (depan)");
            for (int i = 0; i <= belakang; i++) {
                System.out.print(" " + antrian[i]);
            }
            System.out.println(" (belakang)");
            System.out.println();
        }

        public static void main(String[] args) {
            String antrian[] = new String[5];
            ENQUEUE(antrian, "Mobil A");
            ENQUEUE(antrian, "Mobil B");
            ENQUEUE(antrian, "Mobil C");
            ENQUEUE (antrian, "Mobil D");
             ENQUEUE (antrian, "Mobil E");
             ENQUEUE (antrian, "Mobil F");
            lihatAntrian(antrian);
        }
    }
```



Pembahasan : Ketika program dijalankan, proses **ENQUEUE** berhasil memasukkan *Mobil A*, *Mobil B*, *Mobil C*, *Mobil D*, dan *Mobil E* ke dalam antrian, tetapi *Mobil F* gagal masuk karena kapasitas maksimum antrian hanya 5 elemen. Ketika dilakukan **DEQUEUE**, data yang akan keluar pertama kali adalah *Mobil A*, sebab prinsip antrian mengikuti konsep **FIFO (First In First Out)**, yaitu elemen yang masuk paling awal akan keluar lebih dahulu. Setelah proses DEQUEUE dilakukan, seluruh elemen di belakangnya akan digeser satu posisi ke depan sehingga posisi *Mobil B*, *Mobil C*, *Mobil D*, dan *Mobil E* bergeser menggantikan posisi sebelumnya. Perubahan ini terjadi karena adanya perintah penggeseran data di dalam perulangan `for (int i = 0; i <= belakang - 1; i++) { antrian[i] = antrian[i + 1]; }` yang menyebabkan elemen-elemen di antrian menempati posisi baru setelah satu data dihapus. Akibatnya, nilai variabel **belakang** berkurang satu setiap kali proses DEQUEUE dilakukan, menandakan jumlah data dalam antrian juga berkurang.

Praktik 8

```
public class program_antrian {
    public static int N = 5;
    public static int belakang = -1;

    public static void ENQUEUE(String antrian[], String data) {
        if (belakang == N - 1) { // jika antrian penuh
            System.out.println("Maaf, antrian penuh, ENQUEUE " + data +
" tidak dapat dilakukan");
        } else { // jika antrian tidak penuh
            belakang = belakang + 1;
            antrian[belakang] = data;
            System.out.println("ENQUEUE " + data + " berhasil..");
        }
    }

    public static String DEQUEUE(String antrian[]) {
        String hasil;
        if (belakang < 0) { // jika antrian kosong
            hasil = "ANTRIAN KOSONG, DEQUEUE GAGAL DILAKUKAN";
        } else { // jika antrian tidak kosong
            hasil = antrian[0];
            // menggeser data ke depan
            for (int i = 0; i <= belakang - 1; i++) {
                antrian[i] = antrian[i + 1];
            }
            belakang = belakang - 1;
        }
        return hasil;
    }

    public static void lihatAntrian(String antrian[]) {
        System.out.println(); // perbaikan di sini
        System.out.print("ISI ANTRIAN : (depan)");
        for (int i = 0; i <= belakang; i++) {
            System.out.print(" " + antrian[i]);
        }
        System.out.println(" (belakang)");
        System.out.println();
    }

    public static void main(String[] args) {
        String antrian[] = new String[5];
        ENQUEUE(antrian, "Mobil A");
        ENQUEUE(antrian, "Mobil B");
        ENQUEUE(antrian, "Mobil C");
        ENQUEUE (antrian, "Mobil D");
         ENQUEUE (antrian, "Mobil E");
         ENQUEUE (antrian, "Mobil F");
         System.out.println("deQueue: " + DEQUEUE(antrian));
        lihatAntrian(antrian);
    }
}
```

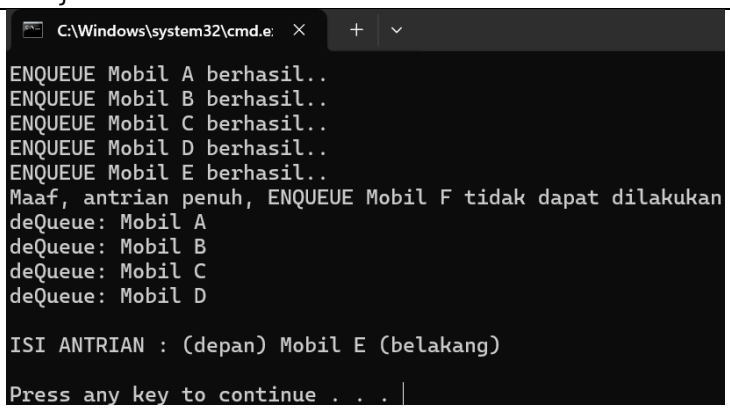


Pembahasan : Setelah program dijalankan, terlihat bahwa mobil yang **ter-deQueue** adalah **Mobil A**, karena antrian bekerja dengan prinsip **FIFO (First In First Out)**, di mana elemen yang pertama kali masuk akan menjadi elemen pertama yang keluar. Setelah proses DEQUEUE dilakukan, posisi data dalam antrian berubah — *Mobil B*, *Mobil C*, *Mobil D*, dan *Mobil E* masing-masing bergeser satu posisi ke depan untuk menempati posisi kosong yang ditinggalkan oleh *Mobil A*. Hal ini terjadi karena adanya perintah penggeseran data di dalam perulangan `for (int i = 0; i <= belakang - 1; i++) { antrian[i] = antrian[i + 1]; }`, yang secara otomatis memindahkan seluruh data ke indeks yang lebih kecil. Akibat proses ini, kondisi antrian berubah dibanding percobaan sebelumnya: jumlah data berkurang satu, urutan data tetap terjaga, dan nilai variabel **belakang** kini berkurang satu (dari 4 menjadi 3), menandakan jumlah elemen aktif di dalam antrian juga berkurang.

Praktik 9

```
    public class program_antrian {
        public static int N = 5;
        public static int belakang = -1;

        public static void ENQUEUE(String antrian[], String data) {
            if (belakang == N - 1) { // jika antrian penuh
                System.out.println("Maaf, antrian penuh, ENQUEUE " + data +
    " tidak dapat dilakukan");
            } else { // jika antrian tidak penuh
                belakang = belakang + 1;
                antrian[belakang] = data;
                System.out.println("ENQUEUE " + data + " berhasil..");
            }
        }

        public static String DEQUEUE(String antrian[]) {
            String hasil;
            if (belakang < 0) { // jika antrian kosong
                hasil = "ANTRIAN KOSONG, DEQUEUE GAGAL DILAKUKAN";
            } else { // jika antrian tidak kosong
                hasil = antrian[0];
                // menggeser data ke depan
                for (int i = 0; i <= belakang - 1; i++) {
                    antrian[i] = antrian[i + 1];
                }
                belakang = belakang - 1;
            }
            return hasil;
        }
```

```
        public static void lihatAntrian(String antrian[]) {
            System.out.println(); // perbaikan di sini
            System.out.print("ISI ANTRIAN : (depan)");
            for (int i = 0; i <= belakang; i++) {
                System.out.print(" " + antrian[i]);
            }
            System.out.println(" (belakang)");
            System.out.println();
        }

        public static void main(String[] args) {
            String antrian[] = new String[5];
            ENQUEUE(antrian, "Mobil A");
            ENQUEUE(antrian, "Mobil B");
            ENQUEUE(antrian, "Mobil C");
            ENQUEUE (antrian, "Mobil D");
             ENQUEUE (antrian, "Mobil E");
             ENQUEUE (antrian, "Mobil F");
             System.out.println("deQueue: " + DEQUEUE(antrian));
             System.out.println("deQueue: " + DEQUEUE(antrian));
             System.out.println("deQueue: " + DEQUEUE(antrian));
             System.out.println("deQueue: " + DEQUEUE(antrian));
            lihatAntrian(antrian);
        }
    }
```

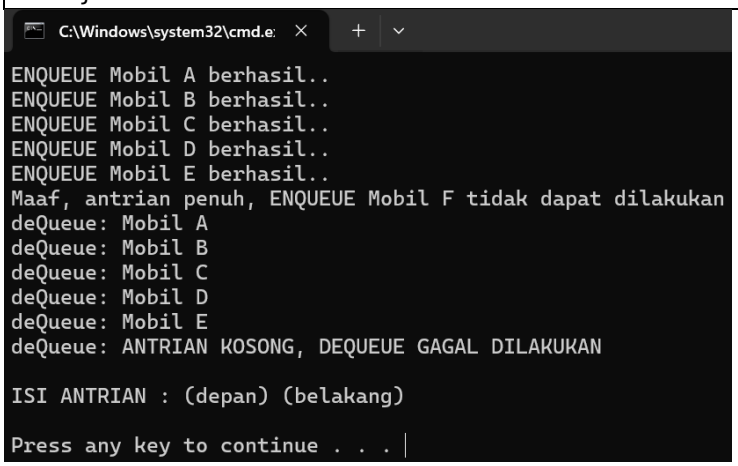


Pembahasan : Jika program di atas dijalankan, maka mobil yang **ter-deQueue** secara berurutan adalah **Mobil A**, **Mobil B**, **Mobil C**, dan **Mobil D**, karena antrian menggunakan prinsip **FIFO (First In First Out)** — artinya, mobil yang pertama kali masuk akan menjadi mobil yang pertama kali keluar. Setelah empat kali proses DEQUEUE, kondisi antrian akan tersisa **Mobil E** saja, karena setiap kali dilakukan penghapusan, elemen di depan (indeks 0) dihapus, dan seluruh elemen di belakangnya digeser satu posisi ke depan melalui `perulangan for (int i = 0; i <= belakang - 1; i++) { antrian[i] = antrian[i + 1]; }`. Akibatnya, posisi data "Mobil E" berpindah ke indeks 0 dan nilai variabel **belakang** berkurang empat kali (dari 4 menjadi 0). Perubahan ini menunjukkan bahwa jumlah data di dalam antrian berkurang seiring proses penghapusan dan posisi data lainnya ikut bergeser agar tetap berurutan dari depan ke belakang.

Praktik 10

```
public class program_antrian {
    public static int N = 5;
    public static int belakang = -1;

    public static void ENQUEUE(String antrian[], String data) {
        if (belakang == N - 1) { // jika antrian penuh
            System.out.println("Maaf, antrian penuh, ENQUEUE " + data +
" tidak dapat dilakukan");
        } else { // jika antrian tidak penuh
            belakang = belakang + 1;
            antrian[belakang] = data;
            System.out.println("ENQUEUE " + data + " berhasil..");
        }
    }

    public static String DEQUEUE(String antrian[]) {
        String hasil;
        if (belakang < 0) { // jika antrian kosong
            hasil = "ANTRIAN KOSONG, DEQUEUE GAGAL DILAKUKAN";
        } else { // jika antrian tidak kosong
            hasil = antrian[0];
            // menggeser data ke depan
            for (int i = 0; i <= belakang - 1; i++) {
                antrian[i] = antrian[i + 1];
            }
            belakang = belakang - 1;
        }
        return hasil;
    }

    public static void lihatAntrian(String antrian[]) {
        System.out.println(); // perbaikan di sini
        System.out.print("ISI ANTRIAN : (depan)");
        for (int i = 0; i <= belakang; i++) {
            System.out.print(" " + antrian[i]);
        }
        System.out.println(" (belakang)");
        System.out.println();
    }

    public static void main(String[] args) {
        String antrian[] = new String[5];
        ENQUEUE(antrian, "Mobil A");
        ENQUEUE(antrian, "Mobil B");
        ENQUEUE(antrian, "Mobil C");
        ENQUEUE (antrian, "Mobil D");
         ENQUEUE (antrian, "Mobil E");
         ENQUEUE (antrian, "Mobil F");
         System.out.println("deQueue: " + DEQUEUE(antrian));
         System.out.println("deQueue: " + DEQUEUE(antrian));
         System.out.println("deQueue: " + DEQUEUE(antrian));
         System.out.println("deQueue: " + DEQUEUE(antrian));
         System.out.println("deQueue: " + DEQUEUE(antrian));
         System.out.println("deQueue: " + DEQUEUE(antrian));
        lihatAntrian(antrian);
    }
```

```
    }
```



Pembahasan : Ya, pada program tersebut terdapat **proses DEQUEUE yang tidak dapat dilakukan**, yaitu pada **dua perintah terakhir** ketika antrian sudah kosong. Hal ini terjadi karena jumlah elemen yang masuk ke dalam antrian hanya sebanyak lima data (Mobil A–Mobil E), sedangkan proses `DEQUEUE` dijalankan sebanyak enam kali. Setelah lima kali penghapusan, seluruh elemen antrian telah keluar, dan variabel **belakang** bernilai `-1`, menandakan bahwa antrian kosong. Saat perintah `DEQUEUE` keenam dijalankan, program mengeksekusi bagian kondisi `if (belakang < 0)` sehingga menampilkan pesan **"ANTRIAN KOSONG, DEQUEUE GAGAL DILAKUKAN"**. Kejadian ini menunjukkan bahwa operasi DEQUEUE hanya bisa dilakukan jika antrian masih memiliki elemen, dan bagian kode yang menyebabkan hal ini adalah pengecekan kondisi kosong di dalam metode `DEQUEUE`.

LATIHAN

-

D. TUGAS

-

D. KESIMPULAN

Kesimpulan yang saya peroleh dari hasil praktikum **pemanfaataan array sebagai stack (tumpukan) dan queue (antrian)** adalah bahwa array dapat digunakan untuk menyimpan dan mengelola data secara terstruktur dengan prinsip yang berbeda pada masing-masing konsep. Pada **stack**, data dikelola dengan prinsip **LIFO (Last In First Out)**, artinya data yang terakhir dimasukkan akan menjadi data yang pertama dikeluarkan. Sedangkan pada **queue**, data dikelola dengan prinsip **FIFO (First In First Out)**, yaitu data yang pertama masuk akan menjadi data yang pertama keluar. Melalui praktikum ini, saya memahami bagaimana proses **push**, **pop**, **enqueue**, dan **dequeue** bekerja menggunakan array, serta pentingnya pengendalian indeks seperti variabel `atas` pada stack dan `belakang` pada queue agar tidak terjadi kesalahan seperti **tumpukan penuh** atau **antrian kosong**. Praktikum ini membantu saya lebih

memahami logika dasar struktur data dan cara mengimplementasikannya dalam program Java.