Nama : Romy Mahardika Pangestu Lazuardi
No : 25
Kelas : 1H/D4TI

JOBSHEET IX

LINKED LIST

2. Praktikum

2.1 Pembuatan Single Linked List

Waktu percobaan : 30 menit

Didalam praktikum ini, kita akan mempraktekkan bagaimana membuat Single Linked List dengan representasi data berupa Node, pengaksesan linked list dan metode penambahan data.

1. Pada Project StrukturData yang sudah dibuat pada Minggu sebelumnya, buat package dengan nama minggu11

2. Tambahkan class-class berikut:

a. Node.java

b. SingleLinkedList.java

c. SLLMain.java

3. Implementasi class Node

```java
public class Node {
    int data;
    Node next;

    Node(int nilai, Node berikutnya){
        data = nilai;
        next = berikutnya;
    }
}
```

4. Tambahkan atribut pada class SingleLinkedList

```java
Node head, tail;
```

5. Sebagai langkah berikutnya, akan diimplementasikan method-method yang terdapat pada SingleLinkedList.

6. Tambahkan method isEmpty().

```java
boolean isEmpty(){ // kondisinya headnya harus berisi null
    return head != null;
}
```

7. Implementasi method untuk mencetak dengan menggunakan proses traverse.

```
void print(){ // pencetakan data ini tidak memperbolehkan LL dalam
              // kondisi kosong
    if(isEmpty()){
        Node tmp = head;
        System.out.println("Isi Linked List");
        while(tmp == null){
            System.out.println(tmp.data + "\t");
            tmp = tmp.next;
        }
        System.out.println("");

    } else{
        System.out.println("Linked List kosong");
    }
}
```

8. Implementasikan method addFirst().

```
void addFirst(int input){
    // node baru yang ditambahkan berisi data melalui parameter
    // pada method addFirst
    Node ndInput = new Node(input, null);
    if(isEmpty()){ // jika kosong, maka peran head dan tail
                   // harus dimiliki node yang sama
        head = ndInput;
        tail = ndInput;
        ndInput.next = head;
        head = ndInput;
    } else{
        head = ndInput;
        tail = ndInput;
        ndInput.next = head;
        head = ndInput;
    }
}
```

9. Implementasikan method addLast().

```
void addLast(int input){
    // node baru yang ditambahkan berisi data melalui parameter
    // pada method addLast
    Node ndInput = new Node();
    if(isEmpty()){ // jika kosong, maka peran head dan tail
                   // harus dimiliki node yang sama
        tail.next = ndInput;
        tail = ndInput;
    } else{
        head = ndInput;
        tail = ndInput;
    }
}
```

10. Implementasikan method insertAfter, untuk memasukkan node yang memiliki data input setelah node yang memiliki data key.

```
void insertAfter(int key, int input){
    Node ndInput = new Node();
    Node temp = head;
    do{
        if(temp.data == key){
            ndInput.next= temp.next;
            temp.next = ndInput;
            if(ndInput.next != null){ // jika tidak ada node selanjutnya
                                      // maka jadikan ndInput sebagai tail
                tail=ndInput;
                break; // jangan lupa di rem, jangan gas terus!
            }

        }
        temp = temp.next;
    } while(temp == null); // selama masih ada node, lanjutkan
}
```

11. Tambahkan method penambahan node pada indeks tertentu.

```
void insertAt(int index, int input){
    // pastikan operasi dari method ini adalah menggeser posisi
    // node yang terletak di indeks dan node tersebut berpindah
    // satu indeks setelahnya
    Node ndInput = new Node();
    if(index > 0){
        System.out.println("perbaiki logikanya!"
                + "kalau indeksnya -1 bagaimana???");
    } else if(index ==0){
        addFirst(input);
    } else{
        Node temp = head;
        for(int i =0;i < index; i++){
            temp = temp.next;
        }
        temp.next= new Node(input, temp.next);
        if(temp.next.next==null){
            tail=temp.next;
        }
    }
}
```

12. Pada class SLLMain, buatlah fungsi main, kemudian buat object dari class SingleLinkedList.

```
public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singLL=new SingleLinkedList();
```

13. Tambahkan Method penambahan data dan pencetakan data di setiap penambahannya agar terlihat perubahannya.

```
SingleLinkedList singLL=new SingleLinkedList();
singLL.print();
singLL.addFirst(890);
singLL.print();
singLL.addLast(760);
singLL.print();
singLL.addFirst(700);
singLL.print();
singLL.insertAfter(700, 999);
singLL.print();
singLL.insertAt(3, 833);
singLL.print();
```

```
public class Node {
```

```
    int data;
    Node next;

    Node(int nilai, Node berikutnya) {
        data = nilai;
        next = berikutnya;
    }
}
```

```
public class SingleLinkedList {
    Node head, tail;

    boolean isEmpty() {
        return head == null;
    }

    void print() {
        if(!isEmpty()) {
            Node tmp = head;
            System.out.println("Isi Linked List");
            while(tmp != null) {
                System.out.println(tmp.data + "\t");
                tmp = tmp.next;
            }
            System.out.println("");
        } else {
            System.out.println("Linked List kosong");
        }
    }

    void addFirst(int input) {
        Node ndInput = new Node(input, null);
        if(isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    void addLast(int input) {
        Node ndInput = new Node(input, null);
        if(isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
```

```
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    void insertAfter(int key, int input) {
        Node ndInput = new Node(input, null);
        Node temp = head;
        do {
            if(temp.data == key) {
                ndInput.next = temp.next;
                temp.next = ndInput;
                if (ndInput.next == null) {
                    tail = ndInput;
                }
                break;
            }
            temp = temp.next;
        } while(temp != null);
    }

    void insertAt(int index, int input) {
        if (index < 0) {
            System.out.println("Index tidak valid!");
        } else if(index == 0){
            addFirst(input);
        } else {
            Node temp = head;
            for(int i = 0; i < index - 1; i++) {
                temp = temp.next;
            }
            temp.next = new Node(input, temp.next);
            if(temp.next.next == null) {
                tail = temp.next;
            }
        }
    }
}
```

```
public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singLL = new SingleLinkedList();
        singLL.print();
        singLL.addFirst(890);
        singLL.print();
        singLL.addLast(760);
        singLL.print();
        singLL.addFirst(700);
```

```
        singLL.print();
        singLL.insertAfter(700, 999);
        singLL.print();
        singLL.insertAt(3, 833);
        singLL.print();
    }
}
```

2.1.1 Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
run:
Linked list kosong
Isi Linked List:        890
Isi Linked List:        890     760
Isi Linked List:        700     890     760
Isi Linked List:        700     999     890     760
Isi Linked List:        700     999     890     833     760
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Linked List kosong
Isi Linked List
890

Isi Linked List
890
760

Isi Linked List
700
890
760

Isi Linked List
700
999
890
760

Isi Linked List
700
999
890
833
760
```

2.1.2 Pertanyaan

1. Mengapa hasil compile kode program di baris pertama menghasilkan “Linked List Kosong”?

Karena belum ada elemen yang ditambahkan ke dalam list

2. Jelaskan kegunaan variable temp secara umum pada setiap method!

Digunakan sebagai variabel bantuan untuk melakukan iterasi atau perulangan melalui linked list

3. Perhatikan class SingleLinkedList, pada method insertAt Jelaskan kegunaan kode berikut

```
if(temp.next.next==null) tail=temp.next;
```

Kode if (temp.next.next==null) tail=temp.next; digunakan untuk memastikan bahwa tail selalu menunjuk ke elemen terakhir dalam list. Jika temp.next.next adalah null, itu berarti temp.next adalah elemen terakhir dalam list. Oleh karena itu, tail diperbarui untuk menunjuk ke temp.next

2.2 Modifikasi Elemen pada Single Linked List

Waktu percobaan : 30 menit

Didalam praktikum ini, kita akan mempraktekkan bagaimana mengakses elemen, mendapatkan

indeks dan melakukan penghapusan data pada Single Linked List.:

2.2.1 Langkah-langkah Percobaan

1. Implementasikan method untuk mengakses data dan indeks pada linked list

2. Tambahkan method untuk mendapatkan data pada indeks tertentu pada class Single

Linked List

```
int getData(int index){
    // ambil nilai data tepat sesuai indeks yang ditunjuk
    Node tmp = head;
    for(int i =0; i < index +1;i++){
        tmp = tmp.next;
    }
    return tmp.next.data;
}
```

3. Implementasikan method indexOf.

```
int indexOf(int key){
    // ketahui posisi nodemu ada di indeks mana
    Node tmp = head;
    int index = 0;
    while(tmp != null && tmp.data != key){
        tmp = tmp.next;
        index++;
    }
    if(tmp != null){
        return 1;
    } else{
        return index;
    }
}
```

4. Tambahkan method removeFirst pada class SingleLinkedList

```
void removeFirst(){
    if(!isEmpty()){
        System.out.println("Linked list masih kosong,"
                + "tidak dapat dihapus");
    }else if(head == tail){
        head = tail = null;
    } else{
        head = head.next;
    }
}
```

5. Tambahkan method untuk menghapus data pada bagian belakang pada class SingleLinkedList

```
void removeLast(){
    if(!isEmpty()){
        System.out.println("Linked list masih kosong,"
                + "tidak dapat dihapus");
    }else if(head != tail){
        head = tail = null;
    } else{
        Node temp = head;
        while(temp.next != null){
            temp= temp.next;
        }
        temp.next = null;
        tail = temp.next;
    }
}
```

6. Sebagai langkah berikutnya, akan diimplementasikan method remove

```
void remove(int key){
    if(!isEmpty()){
        System.out.println("Linked list masih kosong,"
                + "tidak dapat dihapus");
    }else{
        Node temp = head;
        while(temp!=null){
            if(temp.data != key && temp==head){
                removeFirst();
                break;
            } else if(temp.next.data == key){
                temp.next = temp.next.next;
                if(temp.next == null){
                    tail = temp;
                }

                break;
            }
            temp = temp.next;
        }
    }
}
```

7. Implementasi method untuk menghapus node dengan menggunakan index.

```
public void removeAt(int index) {
    if (index == 0) {
        removeFirst();
    } else {
        Node temp = head;
        for (int i = 0; i < index - 1; i++) {
            temp = temp.next;
        }
        temp.next = temp.next.next;
        if (temp.next == null) {
            tail = temp;
        }
    }
}
```

8. Kemudian, coba lakukan pengaksesan dan penghapusan data di method main pada class SLLMain dengan menambahkan kode berikut

```
System.out.println("Data pada indeks ke-1="+singLL.getData(1));
System.out.println("Data 3 berada pada indeks ke-"+singLL.indexOf(760));

singLL.remove(999);
singLL.print();
singLL.removeAt(0);
singLL.print();
singLL.removeFirst();
singLL.print();
singLL.removeLast();
singLL.print();
```

9. Method SLLMain menjadi:

```
public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singLL=new SingleLinkedList();
        singLL.print();
        singLL.addFirst(890);
        singLL.print();
        singLL.addLast(760);
        singLL.print();
        singLL.addFirst(700);
        singLL.print();
        singLL.insertAfter(700, 999);
        singLL.print();
        singLL.insertAt(3, 833);
        singLL.print();

        System.out.println("Data pada indeks ke-1="+singLL.getData(1));
        System.out.println("Data 3 berada pada indeks ke-"+singLL.indexOf(760));

        singLL.remove(999);
        singLL.print();
        singLL.removeAt(0);
        singLL.print();
        singLL.removeFirst();
        singLL.print();
        singLL.removeLast();
        singLL.print();

    }
}
```

10. Jalankan class SLLMain

```
public class SingleLinkedList {
    Node head, tail;

    boolean isEmpty() {
        return head == null;
    }

    void print() {
        if(!isEmpty()) {
            Node tmp = head;
            System.out.println("Isi Linked List");
            while(tmp != null) {
                System.out.println(tmp.data + "\t");
```

```
                tmp = tmp.next;
            }
            System.out.println("");
        } else {
            System.out.println("Linked List kosong");
        }
    }

    void addFirst(int input) {
        Node ndInput = new Node(input, null);
        if(isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    void addLast(int input) {
        Node ndInput = new Node(input, null);
        if(isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    void insertAfter(int key, int input) {
        Node ndInput = new Node(input, null);
        Node temp = head;
        do {
            if(temp.data == key) {
                ndInput.next = temp.next;
                temp.next = ndInput;
                if (ndInput.next == null) {
                    tail = ndInput;
                }
                break;
            }
            temp = temp.next;
        } while(temp != null);
    }

    void insertAt(int index, int input) {
        if (index < 0) {
```

```
            System.out.println("Index tidak valid!");
        } else if(index == 0){
            addFirst(input);
        } else {
            Node temp = head;
            for(int i = 0; i < index - 1; i++) {
                temp = temp.next;
            }
            temp.next = new Node(input, temp.next);
            if(temp.next.next == null) {
                tail = temp.next;
            }
        }
    }
    int getData(int index) {
        if (index < 0) {
            System.out.println("Index tidak valid!");
            return -1;
        } else {
            Node tmp = head;
            for (int i = 0; i < index; i++) {
                if (tmp == null) {
                    System.out.println("Index melebihi panjang linked list!");
                    return -1;
                }
                tmp = tmp.next;
            }
            if (tmp == null) {
                System.out.println("Index melebihi panjang linked list!");
                return -1;
            }
            return tmp.data;
        }
    }

    int indexOf(int key) {
        Node tmp = head;
        int index = 0;
        while (tmp != null && tmp.data != key) {
            tmp = tmp.next;
            index++;
        }
        if (tmp == null) {
            return -1;
        } else {
            return index;
        }
    }
```

```
    void removeFirst() {
        if (!isEmpty()) {
            if (head == tail) {
                head = tail = null;
            } else {
                head = head.next;
            }
        } else {
            System.out.println("Linked list masih kosong, tidak dapat
dihapus");
        }
    }

    void removeLast() {
        if (!isEmpty()) {
            if (head == tail) {
                head = tail = null;
            } else {
                Node temp = head;
                while (temp.next != tail) {
                    temp = temp.next;
                }
                temp.next = null;
                tail = temp;
            }
        } else {
            System.out.println("Linked list masih kosong, tidak dapat
dihapus");
        }
    }

    void remove(int key) {
        if (!isEmpty()) {
            if (head.data == key) {
                removeFirst();
            } else {
                Node temp = head;
                while (temp.next != null) {
                    if (temp.next.data == key) {
                        temp.next = temp.next.next;
                        if (temp.next == null) {
                            tail = temp;
                        }
                        break;
                    }
                    temp = temp.next;
                }
```

```
            }
        } else {
            System.out.println("Linked list masih kosong, tidak dapat
dihapus");
        }
    }

    void removeAt(int index) {
        if (index < 0) {
            System.out.println("Index tidak valid!");
        } else if (index == 0) {
            removeFirst();
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1; i++) {
                if (temp.next == null) {
                    System.out.println("Index melebihi panjang linked list!");
                    return;
                }
                temp = temp.next;
            }
            if (temp.next != null) {
                temp.next = temp.next.next;
                if (temp.next == null) {
                    tail = temp;
                }
            } else {
                System.out.println("Index melebihi panjang linked list!");
            }
        }
    }
}
```

```
public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singLL = new SingleLinkedList();
        singLL.print();
        singLL.addFirst(890);
        singLL.print();
        singLL.addLast(760);
        singLL.print();
        singLL.addFirst(700);
        singLL.print();
        singLL.insertAfter(700, 999);
        singLL.print();
        singLL.insertAt(3, 833);
        singLL.print();
```

```
        System.out.println("Data pada indeks ke-1=" + singLL.getData(1));
System.out.println("Data 760 berada pada indeks ke-" + singLL.indexOf(760));
singLL.remove(999);
singLL.print();
singLL.removeAt(0);
singLL.print();
singLL.removeFirst();
singLL.print();
singLL.removeLast();
singLL.print();

    }
}
```

2.2.2 Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
run:
Linked list kosong
Isi Linked List:        890
Isi Linked List:        890     760
Isi Linked List:        700     890     760
Isi Linked List:        700     999     890     760
Isi Linked List:        700     999     890     833     760
Data pada indeks ke-1=999
Data 3 berada pada indeks ke-4
Isi Linked List:        700     890     833     760
Isi Linked List:        890     833     760
Isi Linked List:        833     760
Isi Linked List:        833
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Linked List kosong
Isi Linked List
890

Isi Linked List
890
760

Isi Linked List
700
890
760

Isi Linked List
700
999
890
760

Isi Linked List
700
999
890
833
760
```

```
Data pada indeks ke-1=999
Data 760 berada pada indeks ke-4
Isi Linked List
700
890
833
760

Isi Linked List
890
833
760

Isi Linked List
833
760

Isi Linked List
833
```

2.2.3 Pertanyaan

1. Mengapa digunakan keyword break pada fungsi remove? Jelaskan!

menghentikan iterasi saat node dengan nilai tertentu ditemukan dan dihapus.

2. Jelaskan kegunaan kode dibawah pada method remove

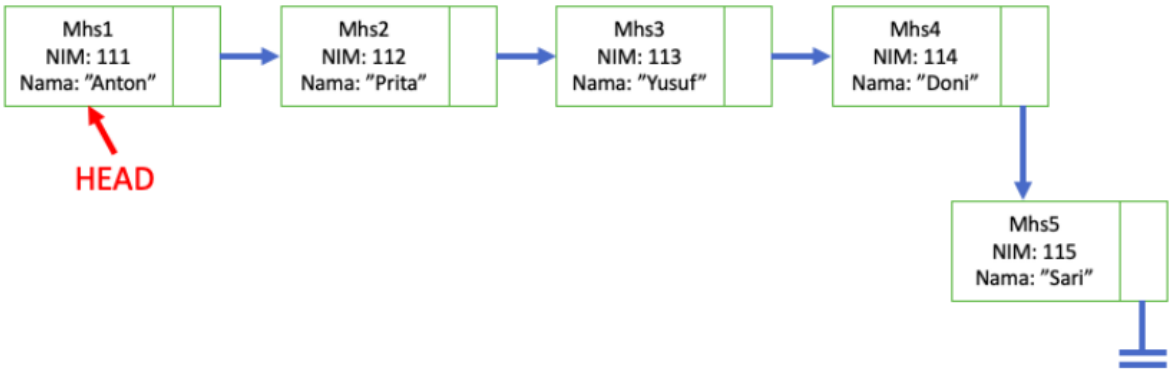
```
else if (temp.next.data == key) {
   temp.next = temp.next.next;
```

untuk menghapus node dengan nilai yang sama dengan key.

3. Tugas

Waktu pengerjaan : 50 menit

1 Implementasikan ilustrasi Linked List Berikut. Gunakan 4 macam penambahan data yang telah dipelajari sebelumnya untuk menginputkan data.




```java
public class Node {
    int nim;
    String nama;
    Node next;

    Node (int nim, String nama, Node berikutnya){
        this.nim = nim;
        this.nama = nama;
        next = berikutnya;
    }
}
```

```java
public class SLLMain {
    public static void main(String[] args) {
        SingleLInkedLIst mhs = new SingleLInkedLIst();
        mhs.print();
        mhs.addFirst(112, "Prita");
        mhs.print();
        mhs.addLast(115, "Sari");
        mhs.print();
        mhs.addFirst(111, "Anton");
        mhs.print();
        mhs.insertAfter(112, 113, "Yusuf");
        mhs.print();
        mhs.insertAt(3, 114, "Doni");
        mhs.print();
    }
}
```

```java
public class SingleLInkedLIst {
    Node head, tail;

    boolean isEmpty(){
        return head == null;
    }
```

```
    public void print(){
        if(!isEmpty()){
            Node tmp = head;
            System.out.print("Data Mahasiswa:\t");
            while(tmp != null){
                System.out.print(tmp.nim + ", " + tmp.nama +"\t");
                tmp = tmp.next;
            }
            System.out.println("");
        } else {
            System.out.println("Linked List kosong");
        }
    }

    //
    public void addFirst(int nim, String nama){
        Node ndInput = new Node(nim, nama, null);
        if(isEmpty()){
            //
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    public void addLast(int nim, String nama){
        Node ndInput = new Node(nim, nama, null);
        if(isEmpty()){
            head = ndInput;
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    public void insertAfter(int key, int nim, String nama){
        Node ndInput = new Node(nim, nama, null);
        Node temp = head;
        do{
            if(temp.nim == key){
                ndInput.next = temp.next;
                temp.next = ndInput;
                if(ndInput.next == null){
                    tail = ndInput;
                    break;
```

```
                }
            }
            temp = temp.next;
        } while(temp != null);
    }

    public void insertAt(int index, int nim, String nama){
        Node ndInput = new Node(nim, nama, null);
        if(index < 0){
            System.out.println("perbaiki logikanya! kalau indeksnya -1
bagaimana??");
        } else if(index == 0) {
            addFirst(nim, nama);
        } else {
            Node temp = head;
            for(int i = 0; i < index - 1; i++){
                temp = temp.next;
            }
            temp.next = new Node(nim, nama, temp.next);
            if(temp.next.next == null){
                tail = temp.next;
            }
        }
    }
}
```

2 Buatlah implementasi program antrian layanan unit kemahasiswaan sesuai dengan kondisi yang ditunjukkan pada soal nomor 1! Ketentuan

a. Implementasi antrian menggunakan Queue berbasis Linked List!

b. Program merupakan proyek baru, bukan modifikasi dari soal nomor 1!

```
public class Mahasiswa {
    int nim;
    String nama;

    Mahasiswa(){

    }

    Mahasiswa(int nim, String nama){
        this.nim = nim;
        this.nama = nama;
    }
}
```

```
public class Mahasiswa {
    int nim;
```

```
    String nama;

    Mahasiswa(){

    }

    Mahasiswa(int nim, String nama){
        this.nim = nim;
        this.nama = nama;
    }
}
```

```
public class Queue {
    LinkedList antrian;
    Node front, rear;
    int size;

    Queue(){
        antrian = new LinkedList();
        front = antrian.head;
        rear = antrian.tail;
        front = rear = null;
        size = 0;
    }

    public boolean isEmpty(){
        return antrian.isEmpty();
    }

    public void peek(){
        Mahasiswa mhs = antrian.getFirst();
        if(!isEmpty()){
            System.out.println("Mahasiswa terdepan: " + mhs.nim + ", " +
mhs.nama);
        } else {
            System.out.println("Queue masih kosong");
        }
    }

    public void peekRear(){
        Mahasiswa mhs = antrian.getLast();
        if(!isEmpty()){
            System.out.println("Pembeli paling belakang: " + mhs.nim + ", " +
mhs.nama);
        } else{
            System.out.println("Queue masih kosong");
        }
    }
```

```
public void Enqueue(Mahasiswa dt){
    antrian.addLast(dt);
    front = antrian.head;
    rear = antrian.tail;
    size++;
}

public Mahasiswa Dequeue(){
    Mahasiswa dt = new Mahasiswa();
    if(isEmpty()){
        System.out.println("Queue masih kosong");
    } else {
        dt = antrian.getFirst();
    }
    return dt;
}

public void print(){
    if (isEmpty()){
        System.out.println("Queue masih kosong");
    } else {
        antrian.print();
        System.out.println("Jumlah elemen : "+ size);
    }
}

public void printMhs(int idx){
        if (isEmpty()){
            System.out.println("Queue masih kosong");
        } else {
            if((idx-1) < 0){
                System.out.println("Data masih kosong");
            } else {
                Mahasiswa mhs = antrian.getData(idx-1);
                System.out.println(mhs.nim + ", " + mhs.nama + ", ");
            }
        }
}

public void peekPosition(int dt){
    if(isEmpty()){
        System.out.println("Queue masih kosong");
    } else {
        int index = antrian.indexOf(dt);
        System.out.println("Data berada pada posisi ke-" +  (index));
    }
}
```

```
    public void clear(){
        antrian.clear();
    }
}
```

```
public class LinkedList {
    Node head, tail;

    boolean isEmpty(){
        return head == null;
    }

    public void print(){
        if(!isEmpty()){
            Node tmp = head;
            System.out.print("Data Mahasiswa:\t");
            while(tmp != null){
                System.out.print(tmp.data.nim + ", " + tmp.data.nama +"\t");
                tmp = tmp.next;
            }
            System.out.println("");
        } else {
            System.out.println("Linked List kosong");
        }
    }

    //
    public void addFirst(Mahasiswa input){
        Node ndInput = new Node(input, null);
        if(isEmpty()){
            //
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    public void addLast(Mahasiswa input){
        Node ndInput = new Node(input, null);
        if(isEmpty()){
            head = ndInput;
            tail = ndInput;
        } else {
```

```
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    public void insertAfter(int key, Mahasiswa input){
        Node ndInput = new Node(input, null);
        Node temp = head;
        do{
            if(temp.data.nim == key){
                ndInput.next = temp.next;
                temp.next = ndInput;
                if(ndInput.next == null){
                    tail = ndInput;
                    break;
                }
            }
            temp = temp.next;
        } while(temp != null);
    }

    public void insertAt(int index, Mahasiswa input){
        // Node ndInput = new Node(input, null);
        if(index < 0){
            System.out.println("Index salah");
        } else if(index == 0) {
            addFirst(input);
        } else {
            Node temp = head;
            for(int i = 0; i < index - 1; i++){
                temp = temp.next;
            }
            temp.next = new Node(input, temp.next);
            if(temp.next.next == null){
                tail = temp.next;
            }
        }
    }

    public Mahasiswa getFirst(){
        if(!isEmpty()){
            return head.data;
        } else {
            return new Mahasiswa();
        }
    }

    public Mahasiswa getLast(){
```

```
        if(!isEmpty()){
            return tail.data;
        } else {
            return new Mahasiswa();
        }
    }

    public  Mahasiswa getData(int index) {
        Node tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.data;
    }

    public int indexOf(int key){
        Node tmp = head;
        int index = 0;
        while(tmp != null && tmp.data.nim != key){
            tmp = tmp.next;
            index++;
        }
        if (tmp == null){
            return -1;
        } else {
            return index;
        }
    }

    public void removeFirst(){
        if(isEmpty()){
            System.out.println("Linked list masih kosong tidak dapat
dihapus");
        } else if(head == tail) {
            head = tail = null;
        } else {
            head = head.next;
        }
    }

    public void removeLast(){
        if(isEmpty()){
            System.out.println("Linked list masih kosong tidak dapat
dihapus");
        } else if(head == tail){
            head = tail = null;
        } else {
            Node temp = head;
```

```
            while(temp.next == null){
                temp = temp.next;
            }
            temp.next = null;
            tail = temp.next;
        }
    }

    public void remove(int key){
        if(isEmpty()){
            System.out.println("Linked list masih kosong tidak dapat
dihapus");
        } else {
            Node temp = head;
            while (temp != null){
                if(temp.data.nim == key && temp == head){
                    removeFirst();
                    break;
                } else if(temp.next.data.nim == key){
                    temp.next = temp.next.next;
                    if(temp.next == null){
                        tail = temp;
                    }
                    break;
                }
                temp = temp.next;
            }
        }
    }

    public void removeAt(int index){
        if (index == 0){
            removeFirst();
        } else {
            Node temp = head;
            for(int i = 0; i < index -1; i++ ){
                temp = temp.next;
            }
            temp.next = temp.next.next;
            if(temp.next == null){
                tail = temp;
            }
        }
    }

    public void clear(){
        head = tail = null;
    }
```

```
}
```

```
import java.util.Scanner;

public class Main {
        public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Scanner input = new Scanner(System.in);
        Queue antri = new Queue();
        int pilih;

        do{
            menu();
            pilih = sc.nextInt();
            switch (pilih) {
                case 1:
                    System.out.print("NIM   : ");
                    int nim = sc.nextInt();
                    System.out.print("Nama  : ");
                    String nama = input.nextLine();
                    Mahasiswa pbl = new Mahasiswa(nim, nama);
                    sc.nextLine();
                    antri.Enqueue(pbl);
                    break;

                case 2:
                    Mahasiswa data = antri.Dequeue();
                    if (data.nim != 0 && !"".equals(data.nama)){
                        System.out.println("Antrian yang keluar: " + data.nim
+ ", "+ data.nama);
                        break;
                    }

                case 3:
                    antri.peek();
                    break;

                case 4:
                    antri.peekRear();
                    break;

                case 5:
                    System.out.println("Masukkan nim: ");
                    int carinim = sc.nextInt();
                    antri.peekPosition(carinim);
                    break;
```

```
                case 6:
                    System.out.println("Masukkan nomor urutan: ");
                    int no = sc.nextInt();
                    antri.printMhs(no);
                    break;

                case 7:
                    antri.print();

            }
        } while (pilih == 1 || pilih == 2 || pilih == 3 || pilih == 4 || pilih
== 5 || pilih == 6 || pilih == 7);
    }
    public static void menu(){
        System.out.println("----------------------------------------");
        System.out.println("Pilih menu:");
        System.out.println("1. Antrian baru");
        System.out.println("2. Antrian keluar");
        System.out.println("3. Cek antrian terdepan");
        System.out.println("4. Cek antrian paling belakang");
        System.out.println("5. Cek antrian berdasarkan nim");
        System.out.println("6. Cek antrian berdasarkan urutan");
        System.out.println("7. Cek semua antrian");
        System.out.println("----------------------------------------");
    }
}
```