Agiftsany Azhar
152011513020/D3-Sistem Informasi

-----Main-----

```
package tugas5_152011513020;

public class Tugas5_152011513020 {

    public static void main(String[] args) {
        System.out.println("-------------------------------------------------");
        System.out.println("Praktikum 5 - Node");
        System.out.println("-------------------------------------------------");

        Node x = new Node(5);
        Node y = new Node(10);
        Node z = new Node(15);

        x.print();
        y.print();
        z.print();

        x.setNext(y);
        y.setNext(z);

        System.out.println(x.getInfo());
        System.out.println(x.getNext().getInfo());
        System.out.println(x.getNext().getNext().getInfo());

        System.out.print("\n\n");
        System.out.println("-------------------------------------------------");
        System.out.println("Praktikum 5 - List");
        System.out.println("-------------------------------------------------");

        List a;

        a = new List();

        a.addFront(5);
        a.addFront(10);
        a.addFront(15);
        a.addFront(20);

        a.addRear(25);
        a.addRear(30);
        a.addRear(35);
        a.addRear(40);

        a.print();

        a.delFront();
        a.print();

        a.delFront();
        a.print();
```

```
        a.delRear();
        a.print();

        a.delRear();
        a.print();

        System.out.print("\n");
        System.out.println("Found       = " + a.found(5));
        System.out.println("Found       = " + a.found(10));
        System.out.println("Found       = " + a.found(15));

        System.out.print("\n");
        System.out.println("Position    = " + a.getPosition(10).getInfo());

        a.delete(5);
        a.print();
    }
}
```

-----Class-----

```
package tugas5_152011513020;

public class Node {
    int info;       // memuat satu data integer
    Node next;      // pointer to next node
    Node prev;

    /**
    * constructor dengan parameter info
    * @param info
    */
    public Node(int info){
        this.info   = info;
        this.next   = null;
        this.prev   = null;
    }

    /**
    * mengubah nilai variable info dengan nilai tertentu yang dimasukkan
    * dari luar melalui parameter input
    * @param info
    */
    public void setInfo(int info){
        this.info    = info;
    }

    /**
    * mengubah variable pointer next menunjuk ke object tertentu sesuai nilai
    * parameter input
    * @param next
    */
    public void setNext(Node next){
        this.next   = next;
```

```
    }

    public void setPrev(Node prev){
        this.prev   = prev;
    }

    /**
    * mengambil nilai info dari sebuah node
    * mengembalikan sebuah nilai integer
    * @return
    */
    public int getInfo(){
        return this.info;
    }

    /**
    * mengambil nilai pointer next, nilainya mungkin null atau merefers
    * pada address/ alamat yang merujuk pada node lain
    * mengembalikan nilai pointer of Node
    * @return
    */
    public Node getNext(){
        return this.next;
    }

    public Node getPrev(){
        return this.prev;
    }

    /**
    * menyetak nilai yang termuat di dalam info
    */
    public void print(){
        System.out.println("Info    = " + this.info);
    }
}

-----Class-----
package tugas5_152011513020;

public class List {
    Node head; // pointer menunjuk node terdepan
    Node tail; // pointer menunjuk node terakhir
    int size; // jumlah node yang terdapat dalam list

    /**
    * constructor create list kosong
    */
    public List(){
        this.head   = null;
        this.tail   = null;
        this.size   = 0;
    }
```

```
/**
* check apakah list dalam keadaan kosong atau tidak
* mengembalikan nilai TRUE atau FALSE
* @return
*/
public boolean isEmpty(){
    return this.size == 0;
}

/**
* menambahkan satu node dengan nilai info ke dalam list pada posisi terdepan
* nilai info didapat melalui passing parameter
* tidak diperlukan check Full karena list bersifat dynamic
* dianggap tidak pernah Full
* @param info
*/
public void addFront(int info){
    if (isEmpty() == true){
        this.head   = new Node(info);
        this.tail   = this.head;
        this.size++;
    }
    else{
        Node t      = new Node(info);
        t.setNext(this.head);
        this.head.setPrev(t);
        this.head   = t;
        this.size++;
    }
}

/**
* menambahkan satu node dengan nilai info ke dalam list pada posisi terakhir
* nilai info didapat melalui passing parameter/**
* tidak diperlukan check Full karena list bersifat dynamic
* dianggap tidak pernah Full
* @param info
*/
public void addRear(int info){
    if (isEmpty() == true){
        this.tail   = new Node(info);
        this.head   = this.tail;
        this.size++;
    }
    else{
        Node t      = new Node(info);
        this.tail.setNext(t);
        t.setPrev(this.tail);
        this.tail   = t;
        this.size++;
    }
}

/**
```

```
    * menghapus satu node dari list pada posisi terdepan
    * bila list dalam keadaan kosong, maka tidak dilakukan aksi apapun
    * */
    public void delFront(){
        if (isEmpty() == false){
            this.head   = this.head.getNext();
            this.head.setPrev(null);
            this.size--;
        }
    }

    /**
    * menghapus satu node dari list pada posisi terakhir
    * bila list dalam keadaan kosong, maka tidak dilakukan aksi apapun
    */
    public void delRear(){
        Node t = this.head;

        if (isEmpty() == false){
            for (int i=1; i<this.size; i++){
                t   = t.getNext();
//                this.size--;
            }
            this.size--;
        }
    }

    /**
    * melakukan check apakah terdapat node dengan nilai info tertentu di dalam
list
    * mengembalikan nilai TRUE bila info ditemukan dan
    * mengembalikan nilai FALSE bila info tidak ditemukan
    * @param info
    * @return
    */
    public boolean found(int info){
        Node t = this.head;
        if(isEmpty() == false){
            for (int i=1; i<this.size; i++){
                if (t.getInfo() == info){
                    return true;
                } else t = t.getNext();
            }
        }
        return false;
    }

    /**
    * mencari node dengan nilai info yang ditentukan dan
    * selanjutnya mengembalikan alamat node yang memuat info tersebut
    * /**
    * function akan mengembalikan nilai null bila data tidak ditemukan
    *
    * @param info
```

```
    * @return
    **/
    public Node getPosition(int info){
        Node t = this.head;
        if(isEmpty() == false){
            for (int i=1; i<this.size; i++){
                if (t.getInfo() == info){
                    return t;
                }
                t = t.getNext();
            }
        }
        return null;
    }

    /**
    * menghapus satu node yang memiliki nilai info tertentu
    * yang dihapus adalah node yang ditemukan pertama dan hanya satu node saja
    * function tidak melakukan tindakan apapun bila node yang dimaksud
    * tidak ditemukan
    *
    * @param info
    */
    public void delete(int info){
        if (found(info) == true){
            Node t  = getPosition(info);
            if(t == this.head){
                delFront();
            }
            else if (t == this.tail){
                delRear();
            }
            else{
                t.getPrev().setNext(t.getNext());
                t.getNext().setPrev(t.getPrev());
                this.size--;
            }
        }
    }

    public void print(){
        Node t = this.head;
        for (int i=1; i<=this.size; i++){
            System.out.print(t.getInfo() + "    ");
            t = t.getNext();
        }
        System.out.print("\n");
    }
}
```