MODUL I

PEMROGRAMAN DASAR

1.1 PERMASALAHAN

1.1.1. Easy Mode

Pertama-tama tambahkan Vadel dalam list, diikuti dengan Loli dan Agus serta Fufu Fafa. Kemudian tambah kak Gem menggunakan method addfirst. Setelah itu hapus Fufu Fafa dari list dengan method deletelast. Terakhir, hapus Kak Gem dengan method deletefirst dan tampilkan list manusia-manusia yang tersisa.

1.2 HASIL PERCOBAAN

1.2.1 Easy Mode

1. Algoritma
   a. Membuat *Class Node* yang berisi variabel *next* dan *method getData*
   b. Selanjutnya membuat *Class* Manusia yaitu turunan dari *Class Node* yang berisi data variabel variabel nama,skill,umur hobi dan method untuk menampilkan data tersebut.
   c. Kemudian membuat *Class Linked List* yang beberapa method seperti *Add First*, *Add Last*, *Delete First*, dan *Delete Last* serta *method Display* untuk menampilkan seluruh data nantinya.
   d. Terakhir membuat *Class Main* dengan membuat objek list yang digunakan untuk menampung data.
   e. Pada *Class Main* juga membuat data dari manusia viral seperti nama, skill, umur dan hobi.
   f. Selanjutnya memasukkan semua manusia viral ke dalam list yang di mana kak Gem di masukkan dengan *method Add First*.Lalu menampilkan semua data tersebut.
   g. Kemudian menghapus data Fufu Fafa dari list dengan menggunakan *method Delete First* dan menghapus data Kak Gem dengan *method Delete Last* dan menampilkan semua data yang tersisa.
2. *Source Code*

```
public abstract class Node {

    public Node next = null;

    public abstract String getData();
}


      public class Manusia extends Node {

          private String nama = "";
          private String skill = "";
          private int umur = 0;
          private String hobi = "";

          public Manusia(String nama, String skill, int umur, String
      hobi) {
              this.nama = nama;
              this.skill = skill;
              this.umur = umur;
              this.hobi = hobi;
          }
```

```
    public String getNama() {
        return nama;
    }

    public String getSkill() {
        return skill;
    }

    public int getUmur() {
        return umur;
    }

    public String getHobi() {
        return hobi;
    }

    @Override
    public String getData() {

          return  String.format("%s\t\t\t|%s\t|%d\t|%s",  nama,
skill, umur, hobi);
    }
}



public class Linkedlist {

    Node head = null;
    Node tail = null;
    Node curr = null;

    public void addFirst(Node n) {
        if (this.head == null) {
            head = n;
            tail = n;
        } else {
            n.next = head;
            head = n;
        }
    }

    public void addLast(Node n) {
        if (this.head == null) {
            head = n;
            tail = head;
        } else {
            tail.next = n;
            tail = n;
        }
    }

    public void deleteFirst(Node n) {
        if (head == null) {
            System.out.println("List masih kosong! ");
        } else {
            if (head == tail) {
                head = tail = null;
            } else {
                head = head.next;
            }
```

```
            }
        }

        public void DeleteLast() {
            if (tail == null) {
                System.out.println("List masih kosong! ");
            } else {
                if (head == tail) {
                    head = tail = null;
                } else {
                    curr = head;
                    while (curr.next != tail) {
                        curr = curr.next;
                    }
                    curr.next = null;
                    tail = curr;
                }
            }
        }

        public void displayList() {
            if (head == null) {
                System.out.println("Daftar masih kosong! ");
                return;
            }

            System.out.println("\nDaftar Manusia Viral:");
            System.out.println("----------------------------------
----------");
             System.out.println("Nama\t\t\t|Skill\t\t\t|Umur\t|Hob
i\t\t");
            System.out.println("----------------------------------
----------");
            curr = head;
            while (curr != null) {

                System.out.println(curr.getData());
                curr = curr.next;
            }
            System.out.println("----------------------------------
----------\n");
        }
    }



    public class Main {

        public static void main(String[] args) {
            Linkedlist list = new Linkedlist();

            Manusia m1 = new Manusia("Kak Gem", "Kasi Paham", 32,
"Bernafas");
             Manusia m2 = new Manusia("Vadel", "Dance Geter", 19,
"Dance");
             Manusia m3 = new Manusia("Loli", "ATM Berjalan", 16,
"Liat Vadel");
            Manusia m4 = new Manusia("Agus", "Agus Sakit   ", 35,
"Donasi");
            Manusia m5 = new Manusia("Fufu Fafa", "Roasting", 30,
"Buka Kaskus");
```

```
            System.out.println("Menambahkan data Manusia Viral");
            list.addFirst(m1);
            list.addLast(m2);
            list.addLast(m3);
            list.addLast(m4);
            list.addLast(m5);
            list.displayList();

            System.out.println("Menghapus Fufu Fafa dari Daftar");
            list.DeleteLast();
            list.displayList();

            System.out.println("Menghapus Kak Gem Dari Daftar");
            list.deleteFirst(m1);
            list.displayList();

        }
    }
```

1.3 ANALISIS DATA

1.3.1 Easy Mode

```
public void addFirst(Node n) {
        if (this.head == null) {
            head = n;
            tail = n;
        } else {
            n.next = head;
            head = n;
        }
    }
```

Script "`public void addFirst(Node n)`" adalah deklarasi metode yang digunakan untuk menambahkan node baru di awal linked list. Pada potongan kode ini, jika "`head == null`", yang berarti list masih kosong, maka node baru akan diinisialisasi sebagai node pertama dengan "`head = n`" dan "`tail = n`", sehingga node tersebut menjadi satu-satunya node dalam list. Namun, jika list sudah berisi node, node baru akan ditambahkan di bagian depan dengan "`n.next = head`" yang menghubungkan node baru dengan node yang saat ini berada di posisi awal (head). Kemudian, "`head`" diperbarui untuk menunjuk ke "`n`", menjadikan node baru sebagai "`head`" dari list.

```
public void addLast(Node n) {
        if (this.head == null) {
            head = n;
            tail = head;
        } else {
            tail.next = n;
            tail = n;
        }
    }
```

Script "`public void addLast(Node n)`" adalah deklarasi metode yang digunakan untuk menambahkan node baru di akhir linked list. Pada potongan kode ini, jika "`head == null`", yang berarti list masih kosong, maka node baru akan diinisialisasi sebagai node pertama dengan "`head = n"** dan **"tail = head`", sehingga node tersebut menjadi satu-satunya node dalam list. Namun, jika list sudah berisi node, node baru akan ditambahkan di akhir dengan "`tail.next = n`" yang menghubungkan node terakhir yang ada dengan node baru. Kemudian, "`tail`" diperbarui untuk menunjuk ke "`n`", menjadikan node baru sebagai *tail* dari list.

```
public void deleteFirst(Node n) {
        if (head == null) {
            System.out.println("List masih kosong! ");
        } else {
            if (head == tail) {
                head = tail = null;
            } else {
                head = head.next;
```

```
            }
        }
    }
```

Script "`public void deleteFirst(Node n)`" adalah deklarasi metode yang digunakan untuk menghapus node pertama dari linked list. Pada potongan kode ini, jika "`head == null`", yang berarti list kosong, maka pesan "List masih kosong!" akan ditampilkan. Namun, jika list memiliki node, ada dua kondisi yang diperiksa:Jika "`head == tail`", artinya hanya ada satu node dalam list. Dalam kasus ini, baik "`head`" maupun "`tail`" di-set menjadi null, mengosongkan list sepenuhnya.Jika list memiliki lebih dari satu node, node pertama akan dihapus dengan mengatur "`head = head.next`", yang berarti node kedua sekarang menjadi head baru dari list, dan node pertama dihapus.

```
public void DeleteLast() {
        if (tail == null) {
            System.out.println("List masih kosong! ");
        } else {
            if (head == tail) {
                head = tail = null;
            } else {
                curr = head;
                while (curr.next != tail) {
                    curr = curr.next;
                }
                curr.next = null;
                tail = curr;
            }
        }
    }
```

Script "`public void DeleteLast()`" adalah metode yang digunakan untuk menghapus node terakhir dari linked list. Jika "`tail == null`", yang berarti list kosong, pesan "List masih kosong!" akan ditampilkan. Jika list memiliki node, dan hanya ada satu node (yaitu "`head == tail`"), maka kedua pointer "`head`" dan "`tail`" di-set menjadi null, mengosongkan list. Namun, jika ada lebih dari satu node, variabel sementara "`curr`" diinisialisasi dengan "head" dan diiterasi melalui list hingga "`curr.next`" menunjuk ke "`tail`". Setelah mencapai node sebelum tail, referensi ke node terakhir dihapus dengan mengatur "`curr.next`" menjadi null, dan pointer "`tail`" diperbarui untuk menunjuk ke node yang sekarang menjadi node terakhir yaitu curr.

```
public void displayList() {
        if (head == null) {
            System.out.println("Daftar masih kosong! ");
            return;
        }
```

Script "`public void displayList()`" adalah metode yang digunakan untuk menampilkan isi dari linked list. Jika "`head == null`", yang berarti list kosong, maka pesan "Daftar masih kosong!" akan ditampilkan dan metode akan dihentikan dengan perintah "`return`". Namun, jika list tidak kosong, metode ini akan melanjutkan untuk menampilkan elemen-elemen dalam list. Dengan mengiterasi dari *head*, setiap *node* akan diakses dan nilainya ditampilkan hingga mencapai *node* terakhir, sehingga pengguna dapat melihat semua elemen yang ada dalam secara berurutan.