JOBSHEET 10

Double Linked Lists

Nama: Rizqi Bagus Andrean

Kelas: TI-1D

Absen: 25

# Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

* + 1. memahami algoritma double linked lists;
    2. membuat dan mendeklarasikan struktur algoritma double linked lists;
    3. menerapkan algoritma double linked lists dalam beberapa *study case*.

# Kegiatan Praktikum 1

Waktu : 90 Menit

# Percobaan 1

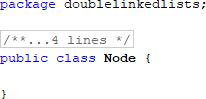
Pada percobaan 1 ini akan dibuat class Node dan class DoubleLinkedLists yang didalamnya terdapat operasi-operasi untuk menambahkan data dengan beberapa cara (dari bagian depan linked list, belakang ataupun indeks tertentu pada linked list).

1. Perhatikan diagram class Node dan class DoublelinkedLists di bawah ini! Diagram class ini yang selanjutnya akan dibuat sebagai acuan dalam membuat kode program DoubleLinkedLists.

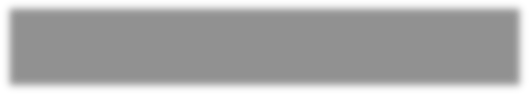
|  |
| --- |
| Node |
| data: int  prev: Node next: Node |
| Node(prev: Node, data:int, next:Node) |

|  |
| --- |
| DoubleLinkedLists |
| head: Node  size : int |
| DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void  add(item: int, index:int): void size(): int  clear(): void  print(): void |

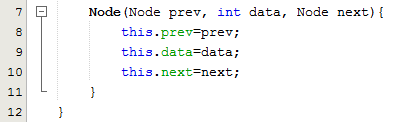
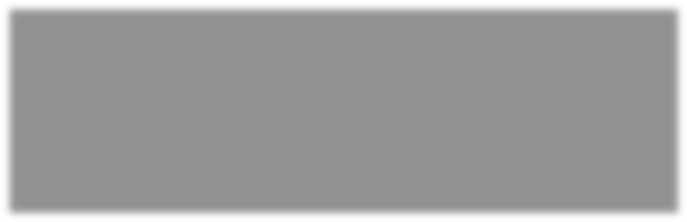
1. Buat paket baru dengan nama **doublelinkedlists**
2. Buat class di dalam paket tersebut dengan nama **Node**



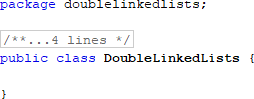
1. Di dalam class tersebut, deklarasikan atribut sesuai dengan diagram class di atas.



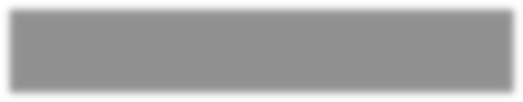
1. Selanjutnya tambahkan konstruktor default pada class Node sesuai diagram di atas.



1. Buatlah sebuah class baru bernama DoubleLinkedLists pada package yang sama dengan node seperti gambar berikut:



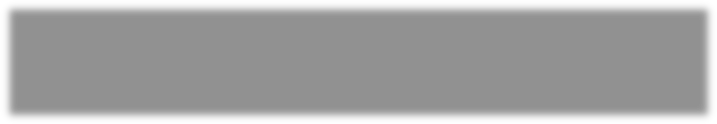
1. Pada class DoubleLinkedLists tersebut, deklarasikan atribut sesuai dengan diagram class di atas.



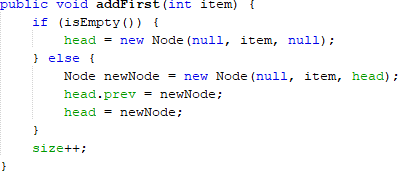
1. Selajuntnya, buat konstruktor pada class DoubleLinkedLists sesuai gambar berikut.

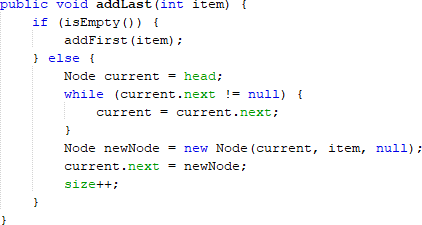


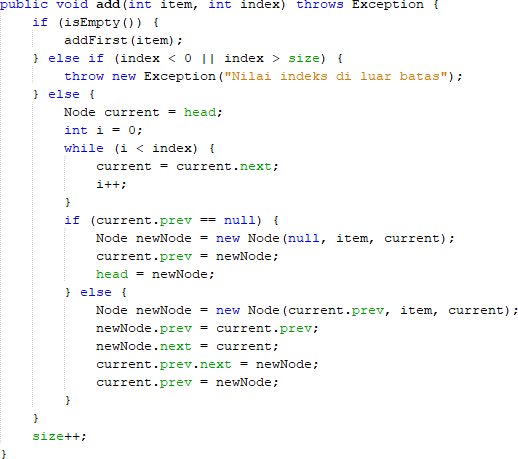
1. Buat method **isEmpty()**. Method ini digunakan untuk memastikan kondisi linked list kosong.



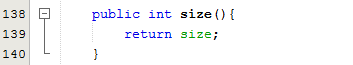
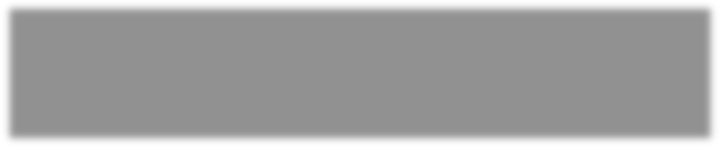
1. Kemudian, buat method **addFirst().** Method ini akan menjalankan penambahan data di bagian depan linked list.



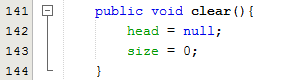
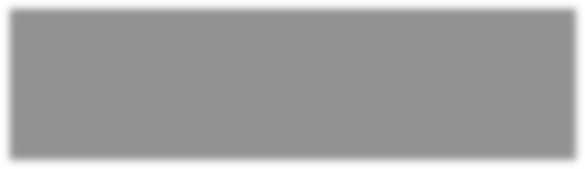
1. Selain itu pembuatan method **addLast()** akan menambahkan data pada bagian belakang linked list.
2. Untuk menambahkan data pada posisi yang telah ditentukan dengan indeks, dapat dibuat dengan method **add(int item, int index)**

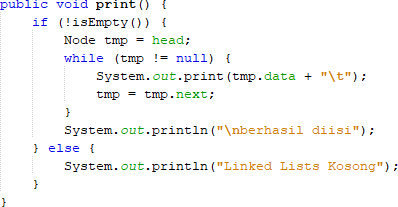
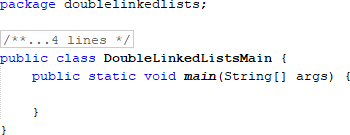


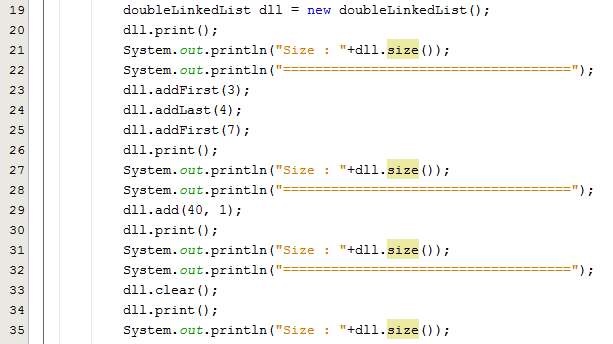
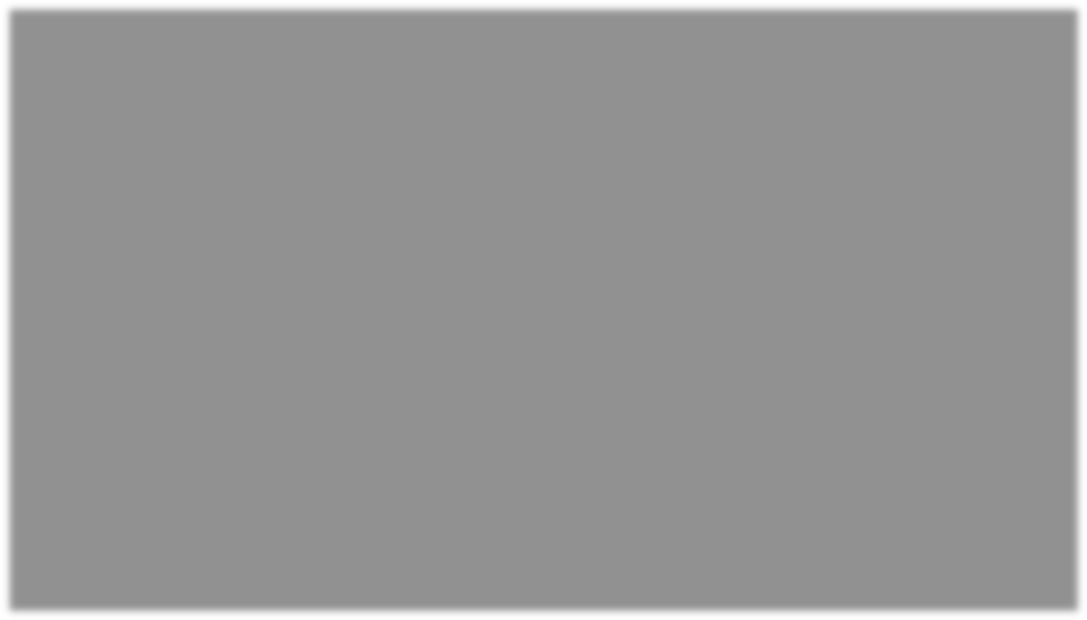
1. Jumlah data yang ada di dalam linked lists akan diperbarui secara otomatis, sehingga dapat dibuat method **size()** untuk mendapatkan nilai dari size.



1. Selanjutnya dibuat method **clear()** untuk menghapus semua isi linked lists, sehingga linked lists dalam kondisi kosong.

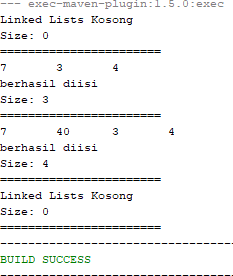


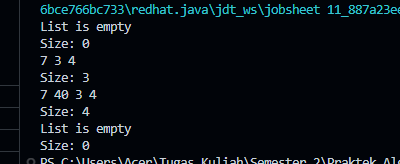
1. Untuk mencetak isi dari linked lists dibuat method **print().** Method ini akan mencetak isi linked lists berapapun size-nya. Jika kosong akan dimunculkan suatu pemberitahuan bahwa linked lists dalam kondisi kosong.
2. Selanjutya dibuat class Main DoubleLinkedListsMain untuk mengeksekusi semua method yang ada pada class DoubleLinkedLists.
3. Pada main class pada langkah 16 di atas buatlah object dari class DoubleLinkedLists kemudian eksekusi potongan program berikut ini.



# Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.





# Pertanyaan Percobaan

1. Jelaskan perbedaan antara single linked list dengan double linked lists!

Single Linked List (SLL) dan Double Linked List (DLL) sama-sama struktur data berurutan, namun berbeda dalam cara menyimpan dan menghubungkan data. SLL hanya memiliki pointer "next" ke node selanjutnya, sedangkan DLL memiliki pointer "next" dan "prev" untuk navigasi dua arah. DLL lebih mudah untuk operasi di tengah list dan penelusuran dua arah, namun membutuhkan memori lebih banyak. SLL lebih hemat memori dan sederhana, tetapi operasi di tengah list lebih rumit dan tidak memungkinkan penelusuran dari akhir ke awal.

1. Perhatikan class Node, di dalamnya terdapat atribut next dan prev. Untuk apakah atribut tersebut?

Di dalam kelas DoubleLinkedList yang diberikan, class Node memiliki dua atribut penting: next dan prev. Atribut ini adalah referensi (penunjuk) ke node lain di dalam list. Atribut next menunjuk ke node berikutnya di dalam list, sementara atribut prev menunjuk ke node sebelumnya di dalam list. Penghubungan dua arah ini memungkinkan untuk menelusuri list dalam dua arah (maju dan mundur), yang merupakan fitur utama dari doubly linked list

1. Perhatikan konstruktor pada class DoubleLinkedLists. Apa kegunaan inisialisasi atribut head dan size seperti pada gambar berikut ini?

inisialisasi head dan size dalam constructor mencerminkan kondisi awal list DoubleLinkedList yang baru dibuat, yaitu list kosong dengan ukuran 0.

1. Pada method **addFirst()**, kenapa dalam pembuatan object dari konstruktor class Node prev dianggap sama dengan null?

Node newNode = new Node(**null**, item, head);

Karena saat addFirt data ditambah pada index pertama dimana index sebelum pertama pasti tidak ada.

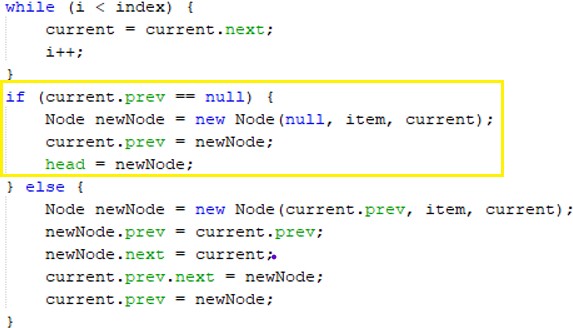
1. Perhatikan pada method **addFirst().** Apakah arti statement head.prev = newNode ?

Berarti sebelum head akan ditambah node baru sehingga headnya akan bergeser.

1. Perhatikan isi method **addLast()**, apa arti dari pembuatan object Node dengan mengisikan parameter prev dengan current, dan next dengan null?

Node newNode = new Node(**current**, item, **null**);

Pembuatan object Node seperti ini dilakukan untuk menyisipkan node baru (newNode) di **akhir** list DoubleLinkedList. Node baru ini disisipkan **setelah** node yang ditunjuk oleh parameter current, dan newNode tidak memiliki node setelahnya (karena berada di akhir list)

1. Pada method **add(),** terdapat potongan kode program sebagai berikut:

jelaskan maksud dari bagian yang ditandai dengan kotak kuning.

Kode itu sebenarnya sama dengan kode yang ada di mthod addFirst() yang digunakan untuk menambah node pada index pertama.

# Kegiatan Praktikum 2

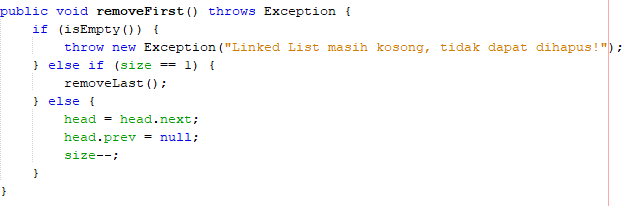
Waktu : 60 Menit

# Tahapan Percobaan

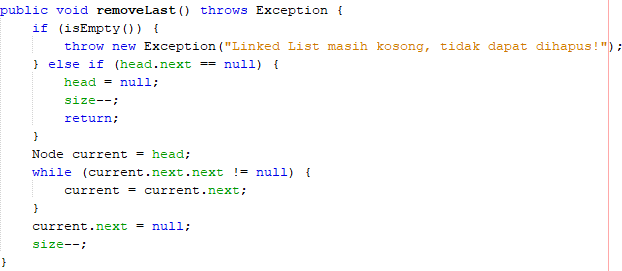
Pada praktikum 2 ini akan dibuat beberapa method untuk menghapus isi LinkedLists pada class DoubleLinkedLists. Penghapusan dilakukan dalam tiga cara di bagian paling depan, paling belakang, dan sesuai indeks yang ditentukan pada linkedLists. Method tambahan tersebut akan ditambahkan sesuai pada diagram class berikut ini.

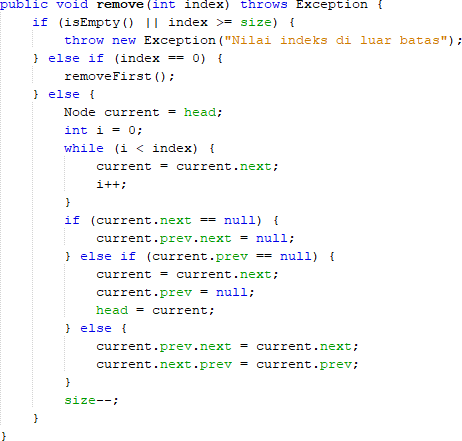
|  |
| --- |
| DoubleLinkedLists |
| head: Node  size : int |
| DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void  add(item: int, index:int): void size(): int  clear(): void print(): void  **removeFirst(): void**  **removeLast(): void remove(index:int):void** |

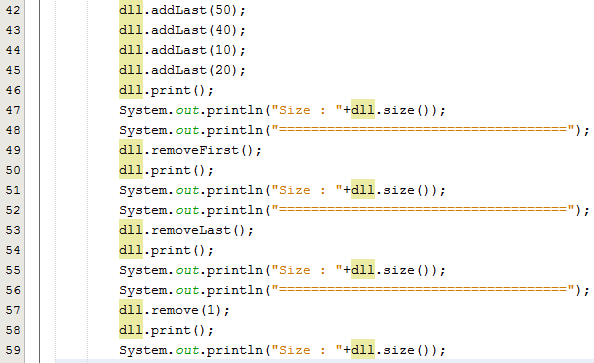
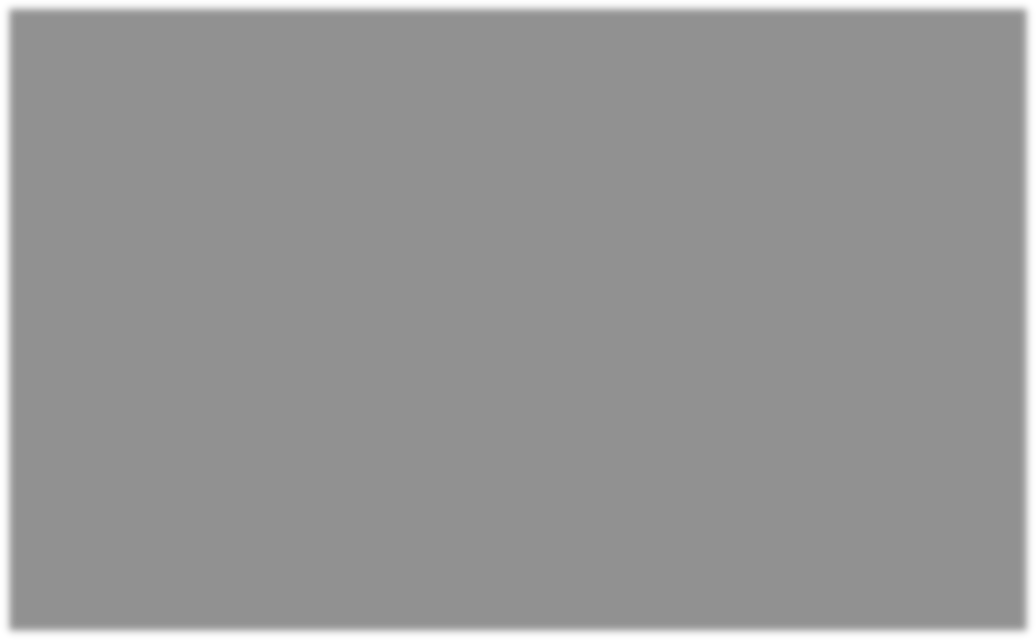
* + - 1. Buatlah method **removeFirst()** di dalam class **DoubleLinkedLists.**



* + - 1. Tambahkan method **removeLast()** di dalam class **DoubleLinkedLists.**

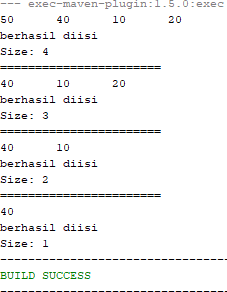


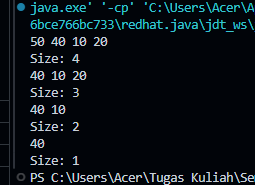
* + - 1. Tambahkan pula method **remove(int index)** pada class **DoubleLinkedLists** dan amati hasilnya.
      2. Untuk mengeksekusi method yang baru saja dibuat, tambahkan potongan kode program berikut pada **main class.**



# Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.





# Pertanyaan Percobaan

* + - 1. Apakah maksud statement berikut pada method **removeFirst()**? head = head.next;

head.prev = null;

Baris pertama head = head.next mengubah referensi head untuk menunjuk ke elemen selanjutnya (next) dari elemen yang saat ini menjadi head. Ini pada dasarnya memutuskan hubungan elemen head lama dari list.

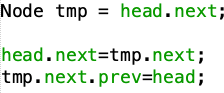
Baris kedua head.prev = null menghapus referensi ke elemen sebelumnya (prev) dari elemen yang sekarang menjadi head baru. Elemen head yang baru sekarang tidak memiliki elemen sebelumnya karena memang menjadi elemen pertama dalam list.

Dengan kedua baris ini, elemen yang tadinya head berhasil dihapus dari list.

* + - 1. Bagaimana cara mendeteksi posisi data ada pada bagian akhir pada method **removeLast()**?

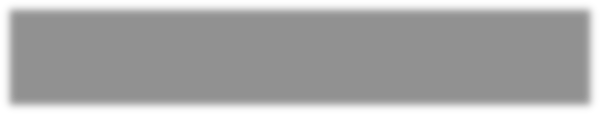
Dengan melakukan perulangan smpai currentnya != nul

* + - 1. Jelaskan alasan potongan kode program di bawah ini tidak cocok untuk perintah **remove**!



Baris ini tidak cocok karena , ia akan menghilangkan elemen sebelumnya karena dipindahkan, sehingga akan kehilangan referensinya

* + - 1. Jelaskan fungsi kode program berikut ini pada fungsi **remove**!



Baris kode tersebut bekerja sama untuk menghapus elemen pada indeks yang ditentukan dari double linked list dengan memperbarui referensi next dan prev dari elemen yang berdekatan dengan elemen yang dihapus.

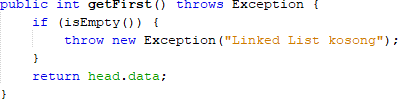
# Kegiatan Praktikum 3

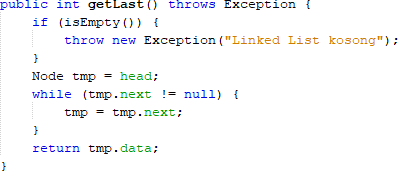
Waktu : 50 Menit

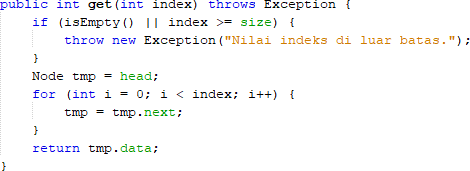
# Tahapan Percobaan

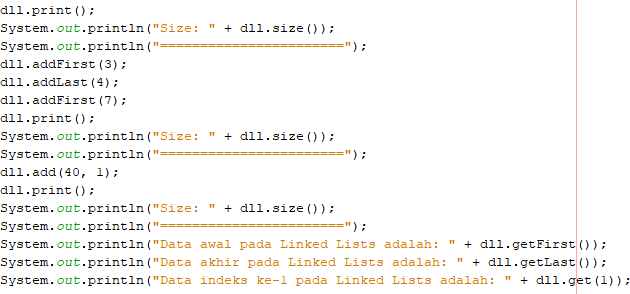
Pada praktikum 3 ini dilakukan uji coba untuk mengambil data pada linked list dalam 3 kondisi, yaitu mengambil data paling awal, paling akhir dan data pada indeks tertentu dalam linked list. Method mengambil data dinamakan dengan **get**. Ada 3 method get yang dibuat pada praktikum ini sesuai dengan diagram class DoubleLinkedLists.

|  |
| --- |
| DoubleLinkedLists |
| head: Node  size : int |
| DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void  add(item: int, index:int): void size(): int  clear(): void print(): void removeFirst(): void removeLast(): void  remove(index:int):void  **getFirst(): int getLast() : int get(index:int): int** |

* + - 1. Buatlah method **getFirst()** di dalam class DoubleLinkedLists untuk mendapatkan data pada awal linked lists.
      2. Selanjutnya, buatlah method **getLast()** untuk mendapat data pada akhir linked lists.

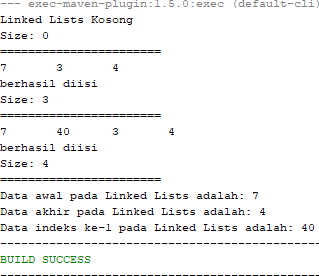


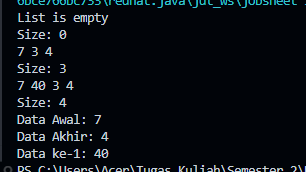
* + - 1. Method **get(int index)** dibuat untuk mendapatkan data pada indeks tertentu
      2. Pada main class tambahkan potongan program berikut dan amati hasilnya!



# Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.





# Pertanyaan Percobaan

* + - 1. Jelaskan method **size()** pada class DoubleLinkedLists!

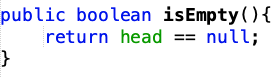
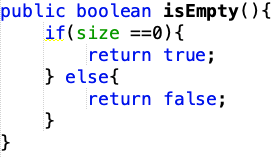
Method size diugunakan untuk menampilkan size dari Node yang tersimpan;

* + - 1. Jelaskan cara mengatur indeks pada double linked lists supaya dapat dimulai dari indeks ke- 1!
      2. Jelaskan perbedaan karakteristik fungsi **Add** pada Double Linked Lists dan Single Linked Lists!

SLL: Bisa menyisipkan di depan, tengah, dan belakang dengan rata-rata O(1) untuk depan dan belakang, dan O(n) untuk tengah, di mana n adalah jumlah elemen. Membutuhkan 1 pointer memori. Cocok untuk penyisipan di depan/belakang yang sering.

DLL: Bisa menyisipkan di depan, tengah, belakang, dan sebelum dengan rata-rata O(1) di semua posisi. Membutuhkan 2 pointer memori. Cocok untuk navigasi bolak-balik dan penyisipan/penghapusan di sembarang posisi.

* + - 1. Jelaskan perbedaan logika dari kedua kode program di bawah ini!



* + - * 1. (b)

1. Program ini mengecek apakah size nya itu == 0, jika iya maka return true, dan jika tidak maka false
2. Program ini sebenarnya mirip, Cuma dia mengecek apakah ada data dinhead atau tidak.

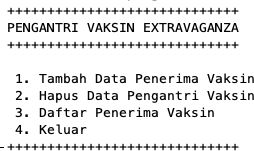
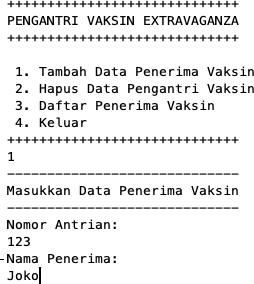
# Tugas Praktikum

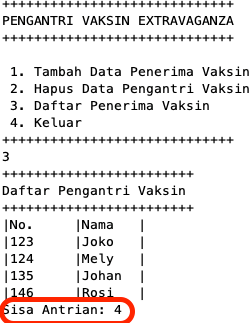
Waktu : 100 Menit

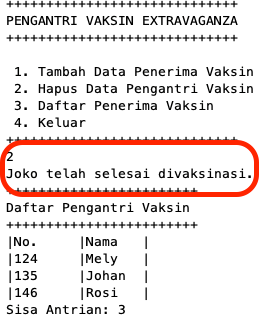
* + 1. Buat program antrian vaksinasi menggunakan queue berbasis double linked list sesuai ilustrasi dan menu di bawah ini! **(counter jumlah antrian tersisa di menu cetak(3) dan data orang yang telah divaksinasi di menu Hapus Data(2) harus ada)**

# Contoh Ilustrasi Program

*Menu Awal dan Penambahan Data*

*Cetak Data (****Komponen di area merah harus ada****)*

*Hapus Data (****Komponen di area merah harus ada****)*



**DLL.java**

package vaksin;

public class DLL {

    Node head;

    int size;

    DLL() {

        head = null;

        size = 0;

    }

    boolean isEmpty() {

        return head == null;

    }

    void addFirst(Orang data) {

        Node newNode = new Node(null, data, head);

        if (!isEmpty()) {

            head.prev = newNode;

        }

        head = newNode;

        size++;

    }

    void addLast(Orang data) {

        if (isEmpty()) {

            addFirst(data);

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            Node newNode = new Node(current, data, null);

            current.next = newNode;

            size++;

        }

        System.out.println("Penerima Vaksin " + data.nama + " telah ditambahkan");

    }

    int size() {

        return size;

    }

    void print() {

        if (!isEmpty()) {

            Node current = head;

            while (current != null) {

                System.out.println(current.data.no + " " + current.data.nama );

                current = current.next;

            }

            System.out.println();

        } else {

            System.out.println("List is empty");

        }

    }

    void removeFirst() {

        if (isEmpty()) {

            System.out.println("List is empty");

        } else if (size == 1) {

            removeLast();

        } else {

            System.out.println(head.data.nama + " telah dihapus");

            head = head.next;

            head.prev = null;

            size--;

        }

    }

    void removeLast() {

        if (isEmpty()) {

            System.out.println("List is empty");

        } else if (size == 1) {

            System.out.println(head.data.nama + " telah dihapus");

            head = null;

            size--;

        } else {

            System.out.println(head.data.nama + " telah dihapus");

            Node current = head;

            while (current.next.next != null) {

                current = current.next;

            }

            current.next = null;

        }

    }

    public static void main(String[] args) {

        DLL dll = new DLL();

        // Tambah Penerima Vaksin

        dll.addLast(new Orang(123, "Joko"));

        dll.addLast(new Orang(124, "Mely"));

        dll.addLast(new Orang(135, "Johan"));

        dll.addLast(new Orang(146, "Rosi"));

        System.out.println();

        // Tampilkan List Penerima Vaksin

        System.out.println("List Penerima Vaksin");

        dll.print();

        System.out.println("Size: " + dll.size());

        System.out.println();

        // Hapus Penerima Vaksin

        dll.removeFirst();

        System.out.println();

        // Tampilkan List Penerima Vaksin

        System.out.println("List Penerima Vaksin");

        dll.print();

        System.out.println("Size: " + dll.size());

    }

}

**Orang.java**

package vaksin;

public class Orang {

    int no;

    String nama;

    public Orang(int no, String nama) {

        this.no = no;

        this.nama = nama;

    }

}

**Node.java**

package vaksin;

class Node {

    Orang data;

    Node next, prev;

    Node(Node prev, Orang data, Node next) {

        this.prev = prev;

        this.data = data;

        this.next = next;

    }

    Node(){}

}

* + 1. Buatlah program daftar film yang terdiri dari id, judul dan rating menggunakan double linked lists, bentuk program memiliki fitur pencarian melalui ID Film dan pengurutan Rating secara descending. Class Film wajib diimplementasikan dalam soal ini.

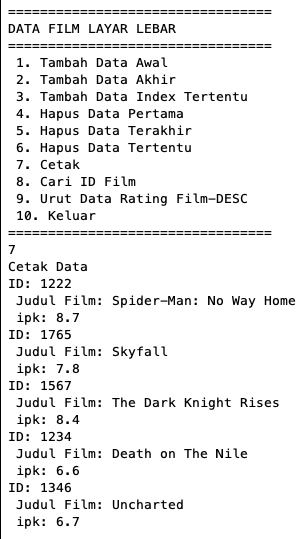
# Contoh Ilustrasi Program

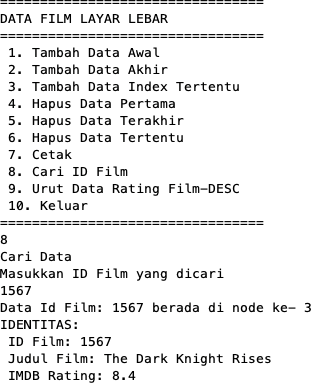
*Menu Awal dan Penambahan Data*

|  |  |
| --- | --- |
|  | Text  Description automatically generated |



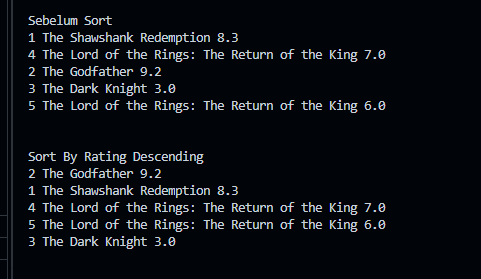
|  |  |
| --- | --- |
|  |  |

*Cetak Data*

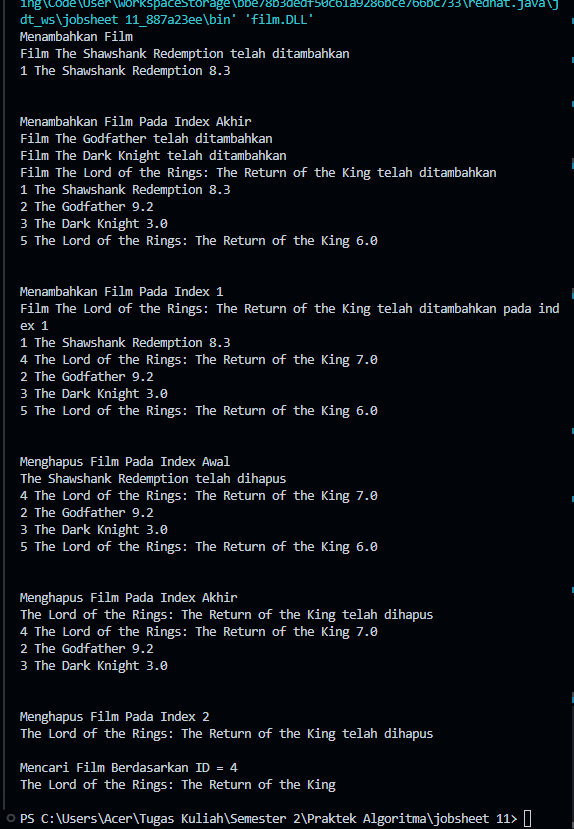
*Pencarian Data*

--- \*\*\* ---

Sort By Desc



Test Menu Lain



**DLL.java**package film;

public class DLL {

    Node head;

    int size;

    DLL() {

        head = null;

        size = 0;

    }

    boolean isEmpty() {

        return head == null;

    }

    void addFirst(Film data) {

        Node newNode = new Node(null, data, head);

        if (!isEmpty()) {

            head.prev = newNode;

        }

        head = newNode;

        size++;

        System.out.println("Film " + data.judul + " telah ditambahkan");

    }

    void addLast(Film data) {

        if (isEmpty()) {

            addFirst(data);

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            Node newNode = new Node(current, data, null);

            current.next = newNode;

            size++;

        }

        System.out.println("Film " + data.judul + " telah ditambahkan");

    }

    void add(Film data, int index) {

        if (index < 0 || index > size) {

            System.out.println("Index out of bound");

        } else if (index == 0) {

            addFirst(data);

        } else {

            Node current = head;

            for (int i = 0; i < index - 1; i++) {

                current = current.next;

            }

            Node newNode = new Node(current, data, current.next);

            current.next = newNode;

            current.next.prev = newNode;

            size++;

        }

        System.out.println("Film " + data.judul + " telah ditambahkan pada index " + index);

    }

    void print() {

        if (!isEmpty()) {

            Node current = head;

            while (current != null) {

                System.out.println(current.data.id + " " + current.data.judul + " " + current.data.rating);

                current = current.next;

            }

            System.out.println();

        } else {

            System.out.println("List is empty");

        }

        System.out.println();

    }

    void removeFirst() {

        if (isEmpty()) {

            System.out.println("List is empty");

        } else if (size == 1) {

           removeLast();

        } else {

            System.out.println(head.data.judul + " telah dihapus");

            head = head.next;

            head.prev = null;

            size--;

        }

    }

    void removeLast() {

        if (isEmpty()) {

            System.out.println("List is empty");

        } else if (size == 1) {

            System.out.println(head.data.judul + " telah dihapus");

            head = null;

            size--;

        } else {

            System.out.println(head.data.judul + " telah dihapus");

            Node current = head;

            while (current.next.next != null) {

                current = current.next;

            }

            current.next = null;

            size--;

        }

    }

    void remove(int index) {

        if (isEmpty()|| index >= size) {

            System.out.println("Index out of bound");

        } else if (index == 0) {

            removeFirst();

        } else {

            Node current = head;

            int i = 0;

            while (i < index - 1) {

                current = current.next;

                i++;

            }

            if (current.next.next == null) {

                removeLast();

            } else if(current.prev == null) {

                removeFirst();

            } else {

                current.next = current.next.next;

                current.next.prev = current;

                size--;

            }

        }

    }

    Film searchFilm(int id) {

        Node current = head;

        while (current != null) {

            if (current.data.id == id) {

                return current.data;

            }

            current = current.next;

        }

        return null;

    }

    void sortByDesc() {

        Node current = head;

        while (current != null) {

            Node next = current.next;

            while (next != null) {

                if (current.data.rating < next.data.rating) {

                    Film temp = current.data;

                    current.data = next.data;

                    next.data = temp;

                }

                next = next.next;

            }

            current = current.next;

        }

    }

}

**Main.java**

public static void main(String[] args) {

        DLL dll = new DLL();

        // Tambah Data Awal

        System.out.println("Menambahkan Film");

        dll.addFirst(new Film(1, "The Shawshank Redemption", 8.3));

        dll.print();

        // Tambah Data di Akhir

        System.out.println("Menambahkan Film Pada Index Akhir");

        dll.addLast(new Film(2, "The Godfather", 9.2));

        dll.addLast(new Film(3, "The Dark Knight", 3));

        dll.addLast(new Film(5, "The Lord of the Rings: The Return of the King", 6));

        dll.print();

        // Tambah Data di Tengah

        System.out.println("Menambahkan Film Pada Index 1");

        dll.add(new Film(4, "The Lord of the Rings: The Return of the King", 7), 1);

        dll.print();

        // Hapus Data Awal

        System.out.println("Menghapus Film Pada Index Awal");

        dll.removeFirst();

        dll.print();

        // Hapus Data di Akhir

        System.out.println("Menghapus Film Pada Index Akhir");

        dll.removeLast();

        dll.print();

        // Hapus Data di Tengah

        System.out.println("Menghapus Film Pada Index 2");

        dll.remove(2);

        System.out.println();

        // Search Film By ID

        System.out.println("Mencari Film Berdasarkan ID = 4");

        System.out.println(dll.searchFilm(4).judul);

        System.out.println();

        // Sort By Rating Descending

        // Sebelum SOrt

        // System.out.println("Sebelum Sort");

        // dll.print();

        // // Setelah Sort

        // System.out.println("Sort By Rating Descending");

        // dll.sortByDesc();

        // dll.print();

    }

**Film.java**

package film;

public class Film {

    int id;

    double rating;

    String judul;

    public Film(int id, String judul,  double rating) {

        this.id = id;

        this.rating = rating;

        this.judul = judul;

    }

}

**Node.java**

package film;

class Node {

    Film data;

    Node next, prev;

    Node(Node prev, Film data, Node next) {

        this.prev = prev;

        this.data = data;

        this.next = next;

    }

    Node(){}

}