JOBSHEET 12 [Double Linked List]

Name : Faricha Aulia

Class : 1I - IT

NIM : 2141720155

12.2.1 Lab-Unit 1

In this experiment, a Node class and a DoubleLinkedLists class will be created in which there are operations to add data in several ways (from the front/first of the linked list,end or a certain index on the linked list).

```
public static void main(String[] args){
    DoubleLL08 dll = new DoubleLL08();
    dll.print();
    System.out.println("Size : " +dll.size());
    System.out.println("==============================");
    dll.addFirst(3);
    dll.addLast(4);
    dll.addFirst(7);
    dll.print();
    System.out.println("Size : " +dll.size());
    System.out.println("==============================");
    dll.add(40, 1);
    dll.print();
    System.out.println("Size : " +dll.size());
    System.out.println("==============================");
    dll.clear();
    dll.print();
    System.out.println("Size : " +dll.size());
    System.out.println("==============================");

public class DoubleLL08 {
  node08 head;
  int size;

  public DoubleLL08(){
    head = null;
    size = 0;
  }

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

  public void addFirst(int item){
    if(isEmpty()){
      head = new node08(null, item, null);
    } else {
      node08 newNode = new node08(null, item, head);
      head.prev = newNode;
      head = newNode;
    } size++;
  }

  public void addLast(int item){
    if(isEmpty()){
      addFirst(item);
```

```
    } else {
        node08 current = head;
        while(current.next != null){
            current = current.next;
        }
        node08 newNode = new node08(current, item, null);
        current.next = newNode;
        size++;
    }
}

public void add(int item, int index){
    if(isEmpty()){
        addFirst(item);
    } else if(index<0 || index>size){
        System.out.println("Index out of bunds");
    } else {
        node08 current = head;
        int i = 0;
        while (i<index){
            current = current.next;
            i++;
        } if (current.prev == null){
            node08 newNode = new node08(null, item, current);
            current.prev = newNode;
            head = newNode;
        } else {
            node08 newNode = new node08(current.prev, item, current);
            newNode.prev = current.prev;
            newNode.next = current;
            current.prev.next = newNode;
            current.prev = newNode;
        }
    }
    size++;
}

public int size(){
    return size;
}

public void clear(){
    head = null;
    size = 0;
}

public void print(){
    if(!isEmpty()){
        node08 tmp = head;
        while(tmp != null){
            System.out.println(tmp.data + " ");
            tmp = tmp.next;
        } System.out.println("\nAll data is displayed");
    } else {
        System.out.println("Linked list is empty");
    }
}
```

```java
public class node08 {
   int data;
   node08 prev, next;

   node08 (node08 prev, int data, node08 next){
      this.prev = prev;
      this.data = data;
      this.next = next;
   }
}
```

12.2.2 Verification

```
Linked list is empty
Size : 0
==============================
7
3
4

Size : 3
==============================
7
40
3
4

All data is displayed
Size : 4
==============================
Linked list is empty
Size : 0
==============================
```

12.2.1 Questions

1. Explain the difference between single linked lists and double linked lists!

   Answer:

   - Single Linked List is a linked list which has only one pointer variable. Where the pointer points to the next node, usually the tail field points to NULL.
   - Double Linked List is a linked list that has two pointer variables, namely a pointer that points to the next node and a pointer that points to the previous node. Each head and tail also points to NULL.

2. Pay attention to the Node class, in which there are next and prev attributes. What are these attributes for?

   Answer:

   - pointer to the node before (prev)
   - pointer to node after (next)

3. Notice the constructor in the DoubleLinkedLists class. What is the use of initializing the head and size attributes as shown in the following image?

```java
public DoubleLinkedLists() {
    head = null;
    size = 0;
}
```

   Answer: To find out whether the Single LinkedList is empty, if the head pointer does not point to a node, the size is 0

4. In the addFirst() method, why is it that when you create an object from the constructor of the Node class, the prev argument is null?

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

Answer: the prev pointer value of HEAD is always NULL, because it is the first data.

5. Pay attention to the addFirst() method. What is the meaning of the following statement?

   head.prev = newNode

   Answer: new node creation when the prev pointer value of HEAD

6. Pay attention to the program code of the addLast() method, what is the meaning of creatinga Node object by filling in the prev parameter with current, and next with null?

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

   Answer:
   - prev parameter filled current returns the value of the current element in the array
   - the next parameter is filled with null so that the function can be used without using a parameter or using a parameter

12.3.1 Lab – Unit 2

In this experiment, several methods will be created to delete data in LinkedLists in the DoubleLinkedLists class. Deletion is carried out in three ways at the very front/first node, at the end, and at a certain index. This additional method will be added according to the following class diagram.

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

```
public void removeFirst(){
    if(isEmpty()){
      System.out.println("Linked list is empty");
    } else if(size == 1){
      removeLast();
    } else {
      head = head.next;
      head.prev = null;
      size--;
    }
  }

  public void removeLast(){
    if(isEmpty()){
      System.out.println("Linked list is empty");
    } else if (head.next == null){
      head = null;
      size--;
      return;
    } else {
```

```
        node08 current = head;
        while(current.next.next != null){
            current = current.next;
        }
        current.next = null;
        size--;
    }
}

public void remove(int index){
    if(isEmpty() || index>=size){
        System.out.println("Index out of bounds");
    } else if (index == 0){
        removeFirst();
    } else {
        node08 current = head;
        int i = 0;
        while(i<index){
            current = current.next;
            i++;
        } if(current.next == null){
            current.prev.next = null;
        } else if (current.prev == null){
            current = current.next;
            current.prev = null;
            head = current;
        } else {
            current.prev.next = current.next;
            current.next.prev = current.prev;
        }
        size--;
    }
}
```

12.3.2 Verification

```
============================
Linked list is empty
Size : 0
============================
50
40
10
20

All data is displayed
Size : 4
============================
40
10
20

All data is displayed
Size : 3
============================
10
20

All data is displayed
Size : 2
============================
10

All data is displayed
Size : 1
============================
```

12.3.3 Questions

1. What does the following statement mean in **removeFirst()** method?

```
head = head.next;
head.prev = null;
```

Answer:

- head - points to the head of the linked list
  head.next - points to the address of the next node
  with head = head.next you change the head to point to the next node
- the prev pointer value of HEAD is always NULL, because it is the first data.

2. How to detect the position of the data that the data is located at the end in the **removeLast() method**?

   Answer: keep an eye on the first node (the "head" node) and go through the list until we find the last node or trace the last node (the "tail" node) in the list and when we add a new node we only need to access the references we have stored directly.

3. Explain the function of the following program code in **remove() method**!

```
current.prev.next = current.next;
current.next.prev = current.prev;
```

Answer: to store the address of the node located
before the node to be deleted

12.4.1 Lab-unit 3

In this Experiment, we will retrieve data on a linked list under 3 conditions: get first data, last data, and data at a certain index in the linked list. The method to retrieve data is called the get method. There are 3 get methods created in this experiment according to the DoubleLinkedListsclass diagram.

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(): voidremoveLast(): void remove(index:int):void **getFirst(): int** **getLast() : int** **get(index:int): int**

```
public  Integer getFirst(){
      if(isEmpty()){
         System.out.println("Linked list is empty");
         return null;
      } else {
         return head.data;
      }
   }

   public Integer getLast(){
      if(isEmpty()){
         System.out.println("Linked list is empty");
         return null;
      } else {
         node08 tmp = head;
         while(tmp.next != null){
            tmp = tmp.next;
         }
         return tmp.data;
      }
   }

   public Integer get(int index){
      if(isEmpty() || index>=size){
         System.out.println("Linked list is empty or index out of bounds");
         return null;
      } else {
         node08 tmp = head;
         for(int i=0; i<index; i++){
            tmp = tmp.next;
         }
         return tmp.data;
      }
   }
}
dll.print();
      System.out.println("Size : " +dll.size());
      System.out.println("==============================");
      dll.addFirst(3);
      dll.addLast(4);
      dll.addFirst(7);
      dll.print();
      System.out.println("Size : " +dll.size());
      System.out.println("==============================");
      dll.add(40,1);
      dll.print();
      System.out.println("Size : " +dll.size());
      System.out.println("==============================");
      System.out.println("Data at the front linked list : " +dll.getFirst());
      System.out.println("Data at the end linked list : " +dll.getLast());
      int index=1;
```

```
System.out.println("Data at index" +index+ "linked list : " +dll.get(index));
dll.clear();
dll.getFirst();
```

Verification

```
All data is displayed
Size : 5
============================
Data at the front linked list : 7
Data at the end linked list : 4
Data at index1linked list : 40
Linked list is empty
```

Questions

1. Explain the **get(index)** method in DoubleLinkedLists class!

   Answer:
2. Explain how to set the index on double linked lists so that the linked list can start at index 1!
   Answer: require more space for each element in the list and element operations such as insertion and deletion are more complex because their setup must be with two references. allows an easier one because it allows moving in both forward and backward directions.
3. Explain the difference between the characteristics of the **Add** method in Double Linked Lists and Single Linked Lists!
   Answer: Double Linked List is more memory usage because it uses 2 pointers while Single Linked List uses only 1 pointer. but both can be both from the front / back
4. Explain the logical difference between the two program codes below!

```
public boolean isEmpty(){
    if(size ==0){
        return true;
    } else{
        return false;
    }
}
```

(a)

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

(b)

Answer:

a. the pointer points to NULL which will be used as a stop condition when reading the contents of the linked list.
b. array to see if it checks for null or empty, but doesn't check its contents. When you iterate over, you just do a check, then iterate over, and when you iterate over, you just check every element

12.1 Assignments

1. Create a vaccination queue program using a queue based on a double linked list according to the illustration and menu below! (the number of queues remaining in the print menu(**Menu 3) and the data of people who have been vaccinated in the Delete Data menu(Menu 2) must exist)**

Program Illustration

Menu 1. Adding data

```
++++++++++++++++++++++++++++
EXTRAVAGANZA Vaccine Queue
++++++++++++++++++++++++++++

 1. Add Vaccine Recipient Data
 2. Delete Vaccination Queue Data
 3. Vaccine Recipient List
 4. Exit
++++++++++++++++++++++++++++
1
----------------------------
Input Vaccine Recipient Data
----------------------------
Queue number / Nomor Antrian:
121
Recipient's name / Nama Penerima:
Jhons
```

*Menu 2. Delete Data (**Components in the red box area must exist**)*

```
EXTRAVAGANZA Vaccine Queue
++++++++++++++++++++++++++++

 1. Add Vaccine Recipient Data
 2. Delete Vaccination Queue Data
 3. Vaccine Recipient List
 4. Exit
++++++++++++++++++++++++++++
2
Jhons has been vaccinated

Queue List
----------------------
122.Melly
123.Rose
124.Willy

remaining queue : 3
```

*Menu 3. Print data (**Components in the red box area must exist**)*

```
++++++++++++++++++++++++++++++
EXTRAVAGANZA Vaccine Queue
++++++++++++++++++++++++++++++

 1. Add Vaccine Recipient Data
 2. Delete Vaccination Queue Data
 3. Vaccine Recipient List
 4. Exit
++++++++++++++++++++++++++++++
3

Queue List
------------------------
121.Jhons
122.Melly
123.Rose
124.Willy

remaining queue : 4
```

```
public class vaccin08 {
   String data;
   vaccin08 prev, next;

   vaccin08 (vaccin08 prev, String data, vaccin08 next){
      this.prev=prev;
      this.data=data;
      this.next=next;
   }
}
public class vaccinDLL08 {
   vaccin08 head;
   int size;

   public vaccinDLL08(){
      head=null;
      size=0;
   }
   public boolean isEmpty(){
      return head==null;
   }

   public void addFirst(String name){
      if(isEmpty()){
         head=new vaccin08(null, name, null);

      }else{
         vaccin08 newNode = new vaccin08(null, name, head);
         head.prev=newNode;
         head=newNode;
      }
      size++;
```

```
}


public void add(String name, int index){
    if(isEmpty()){
        addFirst(name);
    }else if(index<0||index>size){
        return;
    }else{
        vaccin08 current= head;
        int i=0;
        while(i<index){
            current=current.next;
            i++;
        }if(current.prev==null){
            vaccin08 newNode= new vaccin08(null, name, current);
            current.prev=newNode;
            head=newNode;
        }else{
            vaccin08 newNode=new vaccin08(current.prev, name, current);
            newNode.prev=current.prev;
            newNode.next=current;
            current.prev.next=newNode;
            current.prev=newNode;
        }
    }
    size++;
}

public int size(){
    return size;
}

public void clear(){
    head=null;
    size=0;
}
public void print(){
    if(!isEmpty()){
        vaccin08 tmp=head;
        while(tmp!=null){
            System.out.println("."+tmp.data);
            tmp=tmp.next;
        }
    }else{
        System.out.println("LInked list is empty");
    }
}

public void removeLast(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
    } else if (head.next==null){
```

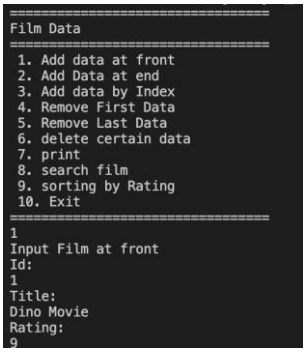
```
        head=null;
        size--;
        return;
    } else {
        vaccin08 current = head;
        while(current.next.next!=null){
            current=current.next;
        }
        current.next=null;
        size--;
    }
}

public void removeFirst(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
    } else if (size==1){
        removeLast();
    } else {
        head=head.next;
        head.prev=null;
        size--;
    }
}

public int search(String key){
    vaccin08 tmp=head;
    int index=0;
    if(head==null){
        System.out.println("The linked list is empty");
    } else {
        while(tmp!=null){
            if(tmp.data==key){
                System.out.println("");
                return index;
            }
            index++;
            tmp=tmp.next;
        }
        if(index==-1){
            System.out.println("Element not found");
        } else {
            System.out.println("Element found at index"+index);
        }
    }return -1;
}

public String getFirst(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
        return null;
    } else {
        return head.data;
```

```
    }
  }

  public String getlast(){
    if(isEmpty()){
      System.out.println("Linked list is empty");
      return null;
    } else {
      vaccin08 tmp=head;
      while(tmp.next!=null){
        tmp=tmp.next;;
      }
      return tmp.data;
    }
  }

  public String get(int index){
    if(isEmpty() || index >= size){
      System.out.println("Linked list is empty or index out of bounds");
      return null;
    } else {
      vaccin08 tmp=head;
      for(int i=0; i<index;i++){
        tmp=tmp.next;
      }
      return tmp.data;
    }
  }
}

import java.util.Scanner;
public class vaccinMain08 {
  public static void menu(){
    System.out.println("++++++++++++++++++++++++++++++");
    System.out.println("EXTRAVAGANZA Vaccine Queue");
    System.out.println("++++++++++++++++++++++++++++++");
    System.out.println("1. Add Vaccine Recipient Data");
    System.out.println("2. Delete Vaccination Queue data");
    System.out.println("3. Vaccine Reccipient List");
    System.out.println("4. Exit");
    System.out.println("++++++++++++++++++++++++++++++");
  }
  public static void main(String[] args) {
    Scanner input08 = new Scanner(System.in);
    vaccinDLL08 dll = new vaccinDLL08();
    int index;
    int choose;
    do{
      menu();
      choose=input08.nextInt();
      input08.nextLine();
      switch(choose){
        case 1:
```

```
            System.out.println("Queue number: ");
            index=input08.nextInt();
            input08.nextLine();
            System.out.println("Recipient's name: ");
            String name=input08.nextLine();
            dll.add(name,index);
          break;
          case 2:
            int pos=1;
            System.out.println( dll.get(pos)+ " has been vaccinated");
            dll.removeFirst();
            System.out.println("Size : "+dll.size);
          break;
          case 3:
            System.out.println("Queue List");
            dll.print();
            System.out.println("Size : "+dll.size);
          break;
        }
      }
      while(choose==1||choose==2||choose==3);
      input08.close();
    }
}
```

2. Create a movie list program consisting of id, title and rating using double linked lists, the program has a search feature with the Movie ID key and a rating sorting feature (descending). Class Film must be implemented.

Add Data



Print Data

```
Film Data
==================================
 1. Add data at front
 2. Add Data at end
 3. Add data by Index
 4. Remove First Data
 5. Remove Last Data
 6. delete certain data
 7. print
 8. search film
 9. sorting by Rating
 10. Exit
==================================
7
Print Data
id: 3
 Title: Upin Ipin Movie
 Rating: 8.0
id: 12
 Title: Paw Patrol
 Rating: 10.0
id: 1
 Title: Dino Movie
 Rating: 9.0
```

```java
public class movie08 {
    String movie;
    int rating;
    movie08 prev,next;

    movie08(movie08 prev, String movie, int rating, movie08 next){
        this.prev=prev;
        this.movie=movie;
        this.rating=rating;
        this.next=next;
    }
}
public class movieDLL08 {
    movie08 head;
    int size;

    public movieDLL08(){
        head=null;
        size=0;
    }
    public boolean isEmpty(){
        return head==null;
    }

    public void addFirst(String movie, int rating){
        if(isEmpty()){
            head = new movie08(null, movie, rating, null);

        }else{
            movie08 newNode = new movie08(null, movie,rating, head);
            head.prev=newNode;
            head=newNode;
        }
        size++;
    }

    public void addLast(String movie){
        if(isEmpty()){
```

```
        addFirst(movie);
    }else{
        movie08 current =head;
        while(current.next!=null){
            current=current.next;
        }
        movie08 newNode = new movie08(current, movie, null);
        current.next=newNode;
        size++;
    }
}

public void add(int index, String movie){
    if(isEmpty()){
        addFirst(movie);
    }else if(index<0||index>size){
        System.out.println("Index out of bounds");
    }else{
        movie08 current= head;
        int i=0;
        while(i<index){
            current=current.next;
            i++;
        }if(current.prev==null){
            movie08 newNode= new movie08(null, movie, current);
            current.prev=newNode;
            head=newNode;
        }else{
            movie08 newNode=new movie08(current.prev, movie, current);
            newNode.prev=current.prev;
            newNode.next=current;
            current.prev.next=newNode;
            current.prev=newNode;
        }
    }
    size++;
}


public int size(){
    return size;
}

public void clear(){
    head=null;
    size=0;
}

public void print(){
    int index;
    index=0;
    if(!isEmpty()){
        movie08 tmp=head;
```

```
        while(tmp!=null){
            System.out.println(index+"."+tmp.movie);
            tmp=tmp.next;
        }
    }else{
        System.out.println("LInked list is empty");
    }
}

public void removeLast(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
    }else if(head.next==null){
        head=null;
        size--;
        return;
    }else{
        movie08 current = head;
        while(current.next.next!=null){
            current=current.next;
        }
        current.next=null;
        size--;
    }
}

public void removeFirst(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
    }else if(size==1){
        removeLast();
    }else{
        head=head.next;
        head.prev=null;
        size--;
    }
}

public void remove(int index){
    if(isEmpty() || index>=size){
        System.out.println("Index out of bounds");

    }else if(index==0){
        removeFirst();
    }else{
        movie08 current=head;
        int i=0;
        while(i<index){
            current=current.next;
            i++;
        }
        if(current.next==null){
            current.prev.next=null;
```

```
        }else if(current.prev==null){
            current=current.next;
            current.prev=null;
            head=current;
        }else{
            current.prev.next=current.next;
            current.next.prev=current.prev;
        }
        size--;
    }
}

public int search(String key){
    movie08 tmp=head;
    int index=0;
    if(head==null){
        System.out.println("The linked list is empty");
    }else{
        while(tmp!=null){
            if(tmp.movie==key){
                System.out.println("");
                return index;
            }
            index++;
            tmp=tmp.next;
        }
        if(index==-1){
            System.out.println("Element not found");
        }else{
            System.out.println("Element found at index"+index);
        }
    }return -1;
}

public String getFirst(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
        return null;
    }else{
        return head.movie;
    }
}

public String getlast(){
    if(isEmpty()){
        System.out.println("Linked list is empty");
        return null;
    }else{
        movie08 tmp=head;
        while(tmp.next!=null){
            tmp=tmp.next;;
        }
```

```
            return tmp.movie;
        }
    }

    public String get(int index){
        if(isEmpty() || index >= size){
            System.out.println("Linked list is empty or index out of bounds");
            return null;
        }else{
            movie08 tmp=head;
            for(int i=0; i<index;i++){
                tmp=tmp.next;
            }
            return tmp.movie;
        }
    }

    public String sorting(){
        movie08 current=null, index=null;
        int temp;
        if(head==null){
            return null;
        }else{
            for(current=head; current.next!=null;current=current.next){
                for(index=current.next;index!=null;index=index.next){
                    if(current.movie>index.movie){
                        temp=current.movie;
                        current.movie=index.movie;
                        index.movie=temp;
                    }

                }
            }
        }
    }
}
import java.util.Scanner;
public class movieMain08 {
    public static void menu(){
        System.out.println("======================");
        System.out.println("Film Data");
        System.out.println("======================");
        System.out.println("1. Add data at front");
        System.out.println("2. Add data at the end");
        System.out.println("3. Add data by index");
        System.out.println("4. Remove first index");
        System.out.println("5. Remove last data");
        System.out.println("6. Delete certain data");
        System.out.println("7. Print");
        System.out.println("8. Search film");
        System.out.println("9. sorting by rating");
        System.out.println("10. Exit");
        System.out.println("======================");
```

```
}

public static void main(String[] args) {
    Scanner input08 = new Scanner(System.in);
    movieDLL08 dll= new movieDLL08();
    String movie;
    int index;
    int choose;
    do{
        menu();
        choose=input08.nextInt();
        switch(choose){
            case 1:
                System.out.println("Enter the film name");
                movie=input08.nextLine();
                dll.addFirst(movie);
            break;
            case 2:
                System.out.println("Enter the film name");
                movie=input08.nextLine();
                dll.addLast(movie);
            break;
            case 3:
                System.out.println("Enter the index");
                index=input08.nextInt();
                System.out.println("Enter the film name");
                movie=input08.nextLine();
            break;
            case 4:
                System.out.println("Removing the first data");
                dll.removeFirst();
            break;
            case 5:
                dll.removeLast();
            break;
            case 6:
                System.out.println("Enter the data index you want to remove");
                index=input08.nextInt();
                dll.remove(index);
            break;
            case 7:
                dll.print();
            break;
            case 8:
                System.out.println("Enter the movie title");
                String key=input08.nextLine();
                dll.search(key);
            break;
        }
        while(choose==1||choose==2||choose==3||choose==4||choose==5);
        input08.close();
    }
}
```