JOBSHEET
Double Linked Lists

12.1 Learning Objectives

After completing this practical session, students will be able to:

1. Understand the double linked list algorithm.
2. Create and declare the structure of a double linked list algorithm.
3. Apply the double linked list algorithm in various case studies.

12.2 Experiment 1

12.2.1 Activity 1

In Experiment 1, a **Student** class, a **Node** class, and a **DoubleLinkedList** class will be created. These classes will include operations for adding data in multiple ways (from the front and the back of the linked list).

1. Observe the class diagram of Student, Node, and DoubleLinkedLists below. This class diagram will serve as a reference for writing the DoubleLinkedLists program code.

Student
nim: String name: String className: String gpa: Double
Student(String nim, String name, String className, Double gpa) print(): void

Node
data: Student prev: Node next: Node

Node()
Node(data: Student)
Node(prev: Node, data: Student, next: Node)

DoubleLinkedLists
head: Node tail : Node
DoubleLinkedLists() isEmpty(): boolean addFirst (): void addLast(): void add(item: int, index:int): void print(): void removeFirst(): void removeLast(): void insertAfter: void

2. Create a new package named **P13** or by using your preferred name
3. Create a class within the package named **Student**. In this class, declare the attributes according to the class diagram above. Also, add the constructor and methods as specified in the diagram.

```
package P13;

public class Student {
    String nim, name, className;
    double gpa;

    public Student(){
    }
    public Student(String nm, String nama, String kls, double ip){
        nim = nm;
        name = nama;
        className = kls;
        gpa = ip;
    }
    void print(){
        System.out.println(nim+" - "+name+" - "+className+" - "+gpa);
    }
}
```

4. Create a class within the package named **Node**. In this class, declare the attributes according to the class diagram above. Then, add the constructor as specified in the diagram.

```
public class Node {
    Student data;
    Node prev;
    Node next;

    Node(){
    }
    Node(Student data){
        this.data = data;
        prev = null;
        next = null;
    }
    Node(Node prev, Student data, Node next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }
}
```

5. Create a new class named **DoubleLinkedLists** in the same package as **Node**. In the **DoubleLinkedLists** class, declare the attributes according to the class diagram above.

```
public class DoubleLinkedLists {
    Node head;
    Node tail;

}
```

6. Add constructor to initialize **head** and **tail** attributes to be **null**;

```
DoubleLinkedLists(){
        head = null;
        tail = null;
    }
```

7. Create a new method isEmpty() that will return true in case the double linked list is still empty.

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

8. Next, create the **addFirst()** method. This method will perform data insertion at the front of the linked list.

```
void addFirst(Student data){
        Node newNode = new Node(data);
        if(isEmpty()){
            head = tail = newNode;
        }else{
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
    }
```

9. In addition, create the **addLast()** method to insert data at the end of the linked list.

```
void addLast(Student data){
        Node newNode = new Node(data);
        if(isEmpty()){
            head = tail = newNode;
        }else{
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
    }
```

10. To insert data after the node that contains the specified key, you can do it as follows.

```
void insertAfter(String key, Student data){
        Node newNode = new Node(data);
        Node temp = head;
        while(temp!=null){
            if(temp.data.nim.equalsIgnoreCase(key)){
               if(temp == tail){
                    addLast(data);
               }else{
                    newNode.next = temp.next;
                    newNode.prev = temp;
                    temp.next.prev = newNode;
                    temp.next = newNode;
               }
            }
            temp = temp.next;
        }
        if(temp == null){
            System.out.println("Insertion failed. Data ("+key+") not 
found!!");
        }
    }
```

11. To display the contents of the linked list, create a **print()** method. This method will print all elements in the linked list, regardless of its size.

```
void print(){
        if(!isEmpty()){
            Node temp = head;
            while(temp!=null){
                temp.data.print();
                temp = temp.next;
            }
            System.out.println("");
        }else{
            System.out.println("Double linked list is currently
empty!!");
        }
    }
```

12. Next, create the **DoubleLinkedListsMain** class to execute all the methods available in the **DoubleLinkedLists** class.

```
public class DoubleLinkedListsMain {
    public static void main(String[] args) {
        DoubleLinkedLists dll = new DoubleLinkedLists();
        dll.print();
        dll.addFirst(new Student("111", "Anton", "TI-1I", 3.57));
        dll.print();
        dll.addLast(new Student("112", "Prabowo", "TI-1I", 3.7));
        dll.print();
        dll.addFirst(new Student("113", "Herco", "TI-1I", 3.89));
        dll.print();
        dll.insertAfter("111", new Student("114", "Rizki", "TI-1I",
3.8));
        dll.print();
        dll.insertAfter("112", new Student("115", "Hanzel", "TI-1I",
3.6));
        dll.print();
        dll.insertAfter("120", new Student("116", "Eiyu", "TI-1I",
3.4));
        dll.print();
    }
}
```

13. Run and observe the output!

12.2.2 Output Verification

Verify your program’s compilation results using the following.

```
Double linked list is currently empty!!
111 - Anton - TI-1I - 3.57

111 - Anton - TI-1I - 3.57
112 - Prabowo - TI-1I - 3.7

113 - Herco - TI-1I - 3.89
111 - Anton - TI-1I - 3.57
112 - Prabowo - TI-1I - 3.7

Insertion failed. Data (111) not found!!
113 - Herco - TI-1I - 3.89
111 - Anton - TI-1I - 3.57
```

```
114 - Rizki - TI-1I - 3.8
112 - Prabowo - TI-1I - 3.7

Insertion failed. Data (112) not found!!
113 - Herco - TI-1I - 3.89
111 - Anton - TI-1I - 3.57
114 - Rizki - TI-1I - 3.8
112 - Prabowo - TI-1I - 3.7
115 - Hanzel - TI-1I - 3.6

Insertion failed. Data (120) not found!!
113 - Herco - TI-1I - 3.89
111 - Anton - TI-1I - 3.57
114 - Rizki - TI-1I - 3.8
112 - Prabowo - TI-1I - 3.7
115 - Hanzel - TI-1I - 3.6
```

12.2.3 Questions

1. Explain the difference between a singly linked list and a doubly linked list.
2. Observe the **Node** class, which contains the attributes **next** and **prev**. What are the purposes of these attributes?
3. Examine the constructor in the **DoubleLinkedLists** class. What is the purpose of this constructor?

```
DoubleLinkedLists(){
    head = null;
    tail = null;
}
```

4. In the **addFirst()** method, what is the meaning of the following code?

```
if(isEmpty()){
    head = tail = newNode;
}
```

5. In the **addFirst()** method, what does the statement **head.prev = newNode** mean?
6. In the **insertAfter()** method, what is the meaning of **current.next.prev = newNode**?
7. In the experiment 1 code, in which method the traversal process implemented? What is the meaning of **temp = temp.next** ?
8. In the **insertAfter()** method, what is the following code for?

```
if(temp == tail){
    addLast(data);
}
```

   Do we have to implement it? What if we remove it?
9. In the **insertAfter()** what is this statement **if(temp.data.nim.equalsIgnoreCase(key))** for?

12.3 Experiment 2

12.3.1 Labs Activities

In this **Experiment 2** several methods will be created to delete elements from the LinkedLists in the **DoubleLinkedLists** class. Deletion will be performed in three ways: at the front, at the end, and at a specified index within the linked list.

1. Create the **removeFirst()** method inside the **DoubleLinkedLists** class.

```
void removeFirst(){
        if(isEmpty()){
            System.out.println("Double linked list is currently
empty!!");
        }else if(head == tail){
            head = tail = null;
        }else{
            head = head.next;
            head.prev = null;
        }
    }
```

2. Create the **removeLast()** method inside the **DoubleLinkedLists** class.

```
void removeLast(){
        if(isEmpty()){
            System.out.println("Double linked list is currently
empty!!");
        }else if(head == tail){
            head = tail = null;
        }else{
            tail = tail.prev;
            tail.next = null;
        }
    }
```

3. Also, add the **remove(int index)** method to the **DoubleLinkedLists** class and observe the results.

```
void remove(int index){
        if(isEmpty()){
            System.out.println("Double linked list is currently
empty!!");
        }else if(index == 0){
            removeFirst();
        }else{
            Node temp = head;
            for(int i=0; i<index; i++){
                temp = temp.next;
            }
            if(temp == tail){
                removeLast();
            }else{
                temp.prev.next = temp.next;
                temp.next.prev = temp.prev;
            }
        }
```

4. Reopen the **main** method in the **DoubleLinkedListsMain** class and add the following statements to delete data:

```
dll.removeFirst();
dll.print();
dll.removeLast();
dll.print();
dll.remove(1);
dll.print();
```

5. Run and observe the output!

12.3.2 Output Verification

Verify your program's compilation results using the following.

```
111 - Anton - TI-1I - 3.57
114 - Rizki - TI-1I - 3.8
112 - Prabowo - TI-1I - 3.7
115 - Hanzel - TI-1I - 3.6

111 - Anton - TI-1I - 3.57
114 - Rizki - TI-1I - 3.8
112 - Prabowo - TI-1I - 3.7

111 - Anton - TI-1I - 3.57
112 - Prabowo - TI-1I - 3.7
```

12.3.3 Questions

1. What is the use of the following statement in the removeFirst() method?

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

2. Why is it important to include conditions and processes like the ones below in both **removeFirst()** and **removeLast()** methods? Explain!

```
else if(head == tail){
    head = tail = null;
}
```

3. In the **removeLast()** method, if there is no **tail** attribute inside **DoubleLinkedLists** class, what approach or changes need to be made to the code of the method?
4. What is the purpose of the initial **if(isEmpty())** check in the **remove(int index)** method?
5. Explain how the method handles the removal of a node at the beginning (**index == 0**) and at the end (**temp == tail**) of the linked list.
6. Describe how the method updates the links between nodes when removing a node from the middle of the linked list.
7. The **remove(int index)** method currently does not check for negative index or index that exceed the size of the DoubleLinkedLists. Please add statements to handle these cases.

12.5 Assignments

1. Add an **add()** function to the **DoubleLinkedList** class to insert a node at a **specific index**.
2. Add a **removeAfter()** function to the **DoubleLinkedList** class to delete the node that comes after the node containing a specified key.
3. Add the methods **getFirst(), getLast(),** and **getIndex()** to retrieve data from the head node, tail node, and node at a specific index, respectively.
4. Add a method **getSize()** to return the number of DoubleLinkedLists data!
5. Add method **indexOf()** to return the index of a specific data specified by key (nim)!

--- *** ---