



Name : Azaria Cindy Sahasika
Number Id : 2341760169 / 06
Class : 1G – Business Information System
Lesson : Algorithm and Data Structure
Material : Material 10 – Double Linked Lists
Github Link : <https://github.com/azariacindy/algorithm-ds>

JOBSHEET 12

Double Linked Lists

1.1. Learning Objective

After learning this lab activity, students will be able to:

1. Understand Double Linked List algorithm
2. Create and declare double linked list algorithm
3. Implement double linked list algorithm in various case studies

1.2. Lab Activities 1

In this lab activity, we will create Node class and DoubleLinkedList class that has operations to insert data in multiple way. (from the beginning or the tail of the list)

1.2.1. Steps

1. Take this class diagram as your reference for creating the **DoubleLinkedList** clas

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

2. Create a new package named **DoubleLinkedList**
3. Create a new class in that package named **Node**
4. In that class, declare the attributes as described in the class diagram

```

4      int data;
5      Node prev, next;

```

5. Next, add the default constructor in Node class

```

7      Node(Node prev, int data, Node next){
8          this.prev=prev;
9          this.data=data;
10         this.next=next;
11     }
12 }

```

6. Create a new class named **DoubleLinkedList** in the same package with the node as following image:

```

package doublelinkedlists;

/**...4 lines */
public class DoubleLinkedLists {

}

```

7. Next, we add the attributes

```

8      Node head;
9      int size;

```

8. Then, add the constructor in class **DoubleLinkedList**

```

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

```

9. Create method **isEmpty()**, this method will be used to check whether the linked list is empty or not

```

16     public boolean isEmpty(){
17         return head == null;
18     }

```

10. Then add method **addFirst()**. This method will be executed when we want to add data in the beginning of the list

```

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

```

11. Let's not forget about adding the data in the end of the list. We can do it after adding these lines of code in **addLast()** method

```

public void addLast(int item) {
    if (isEmpty()) {
        addFirst(item);
    } else {
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
        Node newNode = new Node(current, item, null);
        current.next = newNode;
        size++;
    }
}

```

12. If we want to add a data that specified by a certain index, we will need to provide additional method to do so. It can be done by creating the **add()** method

```

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

```

13. We want to make our list has an easy access to retrieve the length of the list. That's why we create method **size()**

```
public int size() {  
    return size;  
}
```

14. We create a method **clear()** to remove all the data that are exist in linked lists

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

15. Next up, to print the whole data in the list, we need to create a method **print()**.

```
public void print() {  
    if(!isEmpty()) {  
        Node tmp = head;  
        while (tmp != null) {  
            System.out.print(tmp.data + "\t");  
            tmp = tmp.next;  
        }  
        System.out.println("\n successfully added");  
    } else {  
        System.out.println("Linked list is empty");  
    }  
}
```

16. After creating the blueprint classes, we will need one main class so that all of that can be included in the program. Create **DoubleLinkedListMain** class to do so

```
package doublelinkedlists;
```

```
/**...4 lines */  
public class DoubleLinkedListsMain {  
    public static void main(String[] args) {  
          
    }  
}
```

17. Instantiate an object from **DoubleLinkedList** class in the main method. Then apply these program code

```

doubleLinkedList dll = new doubleLinkedList();
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());

```

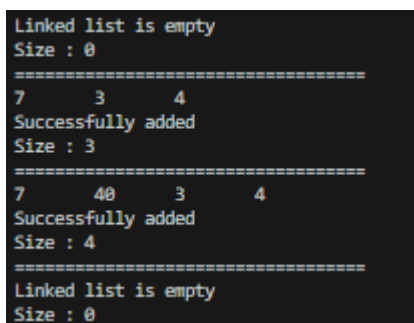
1.2.2. Result

Compile the program and see if the result matches with following image

```

run:
Linked list is empty
Size 0
=====
7      3      4
  successfully added
Size 3
=====
7      40     3      4
  successfully added
Size 4
=====
Linked list is empty
Size 0
BUILD SUCCESSFUL (total time: 1 second)

```



```

Linked list is empty
Size : 0
=====
7      3      4
Successfully added
Size : 3
=====
7      40     3      4
Successfully added
Size : 4
=====
Linked list is empty
Size : 0

```

1.2.3. Questions

1. What's the difference between single linked list and double linked list?

➔ Single linked list

- Each node in a link list contains data and references to the next node in the sequence.
- Navigation is unidirectional, meaning that it can only browse the list in one direction (from end to end).
- Memory usage is lower as each node only stores one reference (to the next node).

➔ Double linked lists

- Each node in a double-linked list contains data, a reference to the next node, and a reference to the previous node.
- Navigation is bidirectional, meaning that you can traverse the list in both directions (from head to tail and tail to head).
- Memory usage is higher as each node stores two references (to the next and previous node).

2. In **Node class**, what is the usage of attribute next and prev?

- ➔ next: This attribute stores a reference to the next node in the list. This attribute is used to browse the list in the forward direction.
- ➔ prev: This attribute stores a reference to the previous node in the list. This attribute is used to traverse the list in the backward direction.
- ➔ These attributes are essential to maintain the structure of a doubly chained list and allow efficient insertion and deletion of nodes at both ends of the list.

3. In constructor of **DoubleLinkedList class**. What's the purpose of head and size attribute in this following code?

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

- ➔ head: This attribute serves as a reference to the first node in the linked list. Initializing it to null indicates that the list is initially empty.
- ➔ size: This attribute keeps track of the number of elements (nodes) in the linked list. Initializing it to 0 indicates that the list starts with zero elements.
- ➔ These attributes are fundamental for managing and manipulating the linked list. The head allows access to the nodes, and the size helps in keeping track of the number of nodes in the list.

4. In method **addFirst()**, why do we initialize the value of Node object to be null at first?

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

- ➔ In the addFirst() method, initialize the Node object with a null value for subsequent references because the new node will be the first node in the list, so there are no subsequent nodes. The head reference of the list will be updated to point to the new node, and the previous reference of the new node will be null because it is the first node.

5. In method **addLast()**, what's the purpose of creating a node object by passing the **prev** parameter with **current** and **next** with **null** ?

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

- ➔ In the addLast() method, it creates a new Node object with the previous reference set to the current node, and subsequent references set to zero. This is because the new node will be the last node in the list, so there are no subsequent nodes. The next reference of the current node will be updated to point to the new node, and the previous reference of the new node will be set to current. In this way, the new node is added to the end of the list, and all existing nodes are linked together in the correct order.

1.3. Lab Activities 2

In this lab activity, we have added some methods from our 1st lab activity. Now, we added some ways for the users to remove a data in the beginning of the list, the tail, or with specified index.

For more details, pay attention to this class 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(): void removeLast(): void remove(index:int):void

1.3.1. Steps

1. Create method **removeFirst()** in class **DoubleLinkedList**

```

public void removeFirst() throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list is still empty, cannot remove");
    }else if(size == 1){
        removeLast();
    }else{
        head = head.next;
        head.prev = null;
        size--;
    }
}

```

2. Create method **removeLast()** in class **DoubleLinkedList**

```

public void removeLast() throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list is still empty, cannot remove");
    }else if(head.next == null){
        head = null;
        size--;
        return ;
    }
    Node current = head;
    while (current.next.next != null) {
        current = current.next;
    }
    current.next = null;
    size--;
}

```

3. Create method **remove()** in class **DoubleLinkedList**, alongside with its parameter

```

public void remove(int index) throws Exception{
    if(isEmpty() || index >= size){
        throw new Exception("Index value is out of bound");
    }else if(size == 0){
        removeFirst();
    }else{
        Node 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--;
    }
}

```

4. To execute additional codes we've just added, also make addition in the main class as well

```

dll.addLast(50);
dll.addLast(40);
dll.addLast(10);
dll.addLast(20);
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("=====");
dll.removeFirst();
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("=====");
dll.removeLast();
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("=====");
dll.remove(1);
dll.print();
System.out.println("Size : "+dll.size());

```

1.3.2. Result

Compile the program and see if the result matches with following image


```

run:
50      40      10      20
    successfully added
Size 4
=====
40      10      20
    successfully added
Size 3
=====
40      10
    successfully added
Size 2
=====
40
    successfully added
Size 1
BUILD SUCCESSFUL (total time: 1 second)

```

```

50      40      10      20
Successfully added
Size : 4
=====
40      10      20
Successfully added
Size : 3
=====
40      10
Successfully added
Size : 2
=====
40
Successfully added
Size : 1

```

1.3.3. Questions

1. What's the meaning of these statements in **removeFirst()** method?

```

89     public void removeFirst() throws Exception {
90         if (isEmpty()) {
91             throw new Exception("Linked list is still empty, can't remove");
92         } else if (size == 1) {
93             removeLast(); // if there's only one element, we can use removeLast to handle it
94         } else {
95             head = head.next; // to point to the second node in the list, the second node becomes the new head of
96             head.prev = null; // to set the prev attribute of the new head node to zero
97             size--; // to reduce the list size by one and reflect the node deletion
98         }
99     }

```

2. How do we detect the position of the data that are in the last index in method **removeLast()**?

➔ After the loop, `current.next` is the last node in the list. Setting `current.next = null` will remove the reference to the last node, effectively removing it from the list.

3. Explain why this program code is not suitable if we include it in **remove** command!

```

Node tmp = head.next;
head.next=tmp.next;
tmp.next.prev=head;

```

➔ because this command specifically deletes the first element (`head.next`). The general delete method needs to take the value to be deleted as input and then browse the list to find a matching node. Once found, it can use logic similar to `removeFirst()` to remove that specific node.

4. Explain what's the function of this program code in method **remove**!

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

- ➔ 'current.prev.next = current.next;': the function updates the next pointer of the node before the deleted node (current.prev) to point to the node after the deleted node (current.next). This effectively bypasses the deleted node in list traversal.
- ➔ 'current.next.prev = current.prev;': the function updates the prev pointer of the node after the deleted node (current.next) to point to the node before the deleted node (current.prev). This maintains the bidirectional connection between nodes in the doubly connected list.

1.4. Lab Activities 3

In this 3rd lab activity, we will test if we can retrieve a data in linked list in various needs. The first is we can get a data in the beginning of the list, at the end of the list, or in specified index of the list. We will create 3 methods to realize the idea. For more details, feel free to check this class 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(): void removeLast(): void remove(index:int):void getFirst(): int getLast() : int get(index:int): int

1.4.1. Steps

1. Create a method **getFirst()** in class **DoubleLinkedList** to retrieve the first data in the list

```
public int getFirst() throws Exception{
    if(isEmpty()) {
        throw new Exception("Linked list still empty");
    }
    return head.data;
}
```

2. Create a method **getLast()** in class **DoubleLinkedList** to retrieve the data in the list

```

public int getLast(int index) throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list still empty");
    }
    Node tmp = head;
    while(tmp.next != null){
        tmp = tmp.next;
    }
    return tmp.data;
}

```

3. Create a method **get(int index)** in class **DoubleLinkedList** to retrieve the data in specified index of the list

```

public int get(int index) throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list still empty");
    }
    Node tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.data;
}

```

4. In the main class, add the program code as follows and see the result

```

public static void main(String[] args) throws Exception {
    DoubleLinkedList dll = new DoubleLinkedList();

    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();

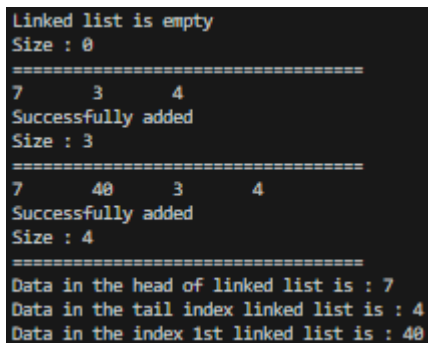
    System.out.println("Size " + dll.size());
    System.out.println("=====");
    System.out.println("Data in the head of linked list is : " + dll.getFirst());
    System.out.println("Data in the tail of linked list is : " + dll.getLast(0));
    System.out.println("Data in the 1st index linked list is : " + dll.get(1));
}

```

1.4.2. Result

Compile the program and see if the result matches with following image

```
run:
Linked list is empty
Size 0
=====
7      3      4
  successfully added
Size 3
=====
7      40     3      4
  successfully added
Size 4
=====
Data in the head of linked list is : 7
Data in the tail of linked list is : 4
Data in the 1st index linked list is : 40
BUILD SUCCESSFUL (total time: 1 second)
```



```
Linked list is empty
Size : 0
=====
7      3      4
Successfully added
Size : 3
=====
7      40     3      4
Successfully added
Size : 4
=====
Data in the head of linked list is : 7
Data in the tail index linked list is : 4
Data in the index 1st linked list is : 40
```

1.4.3. Questions

1. What is the function of method **size()** in **DoubleLinkedList** class ?
 - ➔ The size() method in the DoubleLinkedList class is used to determine the number of elements currently in the list. It loops through the list, starting from the head node, and increments the counter for each node encountered. Finally, it returns the counter value, which represents the total number of elements.
2. How do we set the index in double linked list so that it starts from 1st index instead of 0th index?
 - ➔ Double linked lists, like most data structures, usually use 0-based indexing. This means that the first element has index 0, the second element has index 1, and so on.
3. Please explain the difference between method **Add()** in double linked list and single linked list !
 - ➔ Single linked list
 - A linked list is created, then a new node is added to nodeNew (null if the list is empty). Then nodeNew is updated to become the new first element.
 - ➔ Double-linked lists
 - Creates a linked list, then adds a new node to the first node (null if list is empty). Then the predecessor of nodeNew is set to zero (because it becomes the new first element).
4. What's the logic difference of these 2 following codes?

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

(a)

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

(b)

- a. If size is 0, the list is empty, and true is returned. Otherwise, the list is not empty, and false is returned.
- b. If head is null, the list is empty, and true is returned immediately. If head is not null, the list is not empty, and false is implicitly returned (since the method doesn't have an explicit else statement).

1.5. Assignment

1. Create a program with double linked list implementation that allows user to choose a menu as following image! The searching uses sequential search approach and the program should be able to sort the data in descending order. You may any choose sorting approach you prefer (bubble sort, selection sort, insertion sort, or merge sort)

Adding a data

```
run:
=====
Data manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
1
Insert data in Head postition
34
```

Add data in specified index and display the result

```
run:
=====
Data manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
3
Insert Data
Data node : 66
In index : 1
```

```
=====
Data manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
7
Print data
88
66
32
34
23
67
44
90
99
```

Search Data

```
run:
=====
Data manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
8
Search data : 67
Data 67 is in index-6
```

Sorting Data

```
=====
Data manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
7
Print data :
23
32
34
44
66
67
88
90
99
=====
9
```

Output :

```
=====
Data Manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Print
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
Select menu: 1
Enter data: 34
```

```
=====
Data Manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Print
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
Select menu: 3
Enter data: 80
Enter index: 1
```

```
=====
Data Manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Print
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
Select menu: 7
90 80 90 44 67 23 34 32 66 0
```

```
=====
Data Manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Print
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
Select menu: 8
Enter data to search: 66
Data found
```

```
=====
Data Manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Print
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
Select menu: 9
=====
Data Manipulation with Double Linked List
=====
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Print
7. Print
8. Search Data
9. Sort Data
10. Exit
=====
Select menu: 7
90 90 80 67 66 44 34 32 23 0
```

Code :

jobsheet10_DoubleLinkedLists > assignment06 > J Node06.java > ...

Click here to ask Blackbox to help you code faster

```
1 package assignment06;
2
3 public class Node06 {
4     int data;
5     Node06 prev, next;
6
7     Node06(Node06 prev, int data, Node06 next) {
8         this.prev = prev;
9         this.data = data;
10        this.next = next;
11    }
12 }
```

jobsheet10_DoubleLinkedLists > assignment06 > J DoubleLinkedList06.java > ...

```
3 public class DoubleLinkedList06 {
128     public boolean search(int item) {
129         Node06 tmp = head;
130         while (tmp != null) {
131             if (tmp.data == item) {
132                 return true;
133             }
134             tmp = tmp.next;
135         }
136         return false;
137     }
138 }
```

jobsheet10_DoubleLinkedLists > assignment06 > J DoubleLinkedList06.java > ...

Click here to ask Blackbox to help you code faster

```
1 package assignment06;
2
3 public class DoubleLinkedList06 {
4     Node06 head;
5     int size;
6
7     public DoubleLinkedList06() {
8         head = null;
9         size = 0;
10    }
11
12    public boolean isEmpty() {
13        return head == null;
14    }
15
16    public void addFirst(int item) {
17        if (isEmpty()) {
18            head = new Node06(prev:null, item, next:null);
19        } else {
20            Node06 newNode = new Node06(prev:null, item, head);
21            head.prev = newNode;
22            head = newNode;
23        }
24        size++;
25    }
26
27    public void addLast(int item) {
28        if (isEmpty()) {
29            addFirst(item);
30        } else {
31            Node06 current = head;
32            while (current.next != null) {
33                current = current.next;
34            }
35            Node06 newNode = new Node06(current, item, next:null);
36            current.next = newNode;
37            size++;
38        }
39    }
}
```



```

jobsheet10_DoubleLinkedLists > assignment06 > J DoubleLinkedList06.java > ...
3  public class DoubleLinkedList06 {
41      public void add(int item, int index) throws Exception {
42          if (isEmpty()) {
43              addFirst(item);
44          } else if (index < 0 || index > size) {
45              throw new Exception(message:"Invalid index");
46          } else {
47              Node06 current = head;
48              int i = 0;
49              while (i < index) {
50                  current = current.next;
51                  i++;
52              }
53              if (current.next == null) {
54                  Node06 newNode = new Node06(current, item, next:null);
55                  current.next = newNode;
56              } else {
57                  Node06 newNode = new Node06(current.prev, item, current);
58                  newNode.prev = current.prev;
59                  newNode.next = current;
60                  current.prev.next = newNode;
61                  current.prev = newNode;
62              }
63              size++;
64          }
65      }
66
67      public void removeFirst() throws Exception {
68          if (isEmpty()) {
69              throw new Exception(message:"Linked list is empty, cannot remove!");
70          } else if (size == 1) {
71              head = null;
72              size--;
73              return;
74          }
75          head = head.next;
76          head.prev = null;
77          size--;
78      }

```

jobsheet10_DoubleLinkedLists > assignment06 > J DoubleLinkedList06.java > ...

```
3 public class DoubleLinkedList06 {
80     public void removeLast() throws Exception {
81         if (isEmpty()) {
82             throw new Exception(message:"Linked list is empty, cannot remove!");
83         } else if (head.next == null) {
84             head = null;
85             size--;
86             return;
87         }
88         Node06 current = head;
89         while (current.next != null) {
90             current = current.next;
91         }
92         current.prev.next = null;
93         size--;
94     }
95
96     public void print() {
97         if (!isEmpty()) {
98             Node06 tmp = head;
99             while (tmp != null) {
100                 System.out.print(tmp.data + " ");
101                 tmp = tmp.next;
102             }
103             System.out.println();
104         } else {
105             System.out.println(x:"Linked list is empty");
106         }
107     }
108
109     public void sortDescending() {
110         if (!isEmpty()) {
111             Node06 current = head, index = null;
112             int temp;
113             while (current != null) {
114                 index = current.next;
115                 while (index != null) {
116                     if (current.data < index.data) {
117                         temp = current.data;
118                         current.data = index.data;
119                         index.data = temp;
120                     }
121                     index = index.next;
122                 }
123                 current = current.next;
124             }
125         }
126     }
}
```

jobsheet10_DoubleLinkedLists > assignment06 > J Main06.java > Main06 > main(String[])

Click here to ask Blackbox to help you code faster

```
1 package assignment06;
2 import java.util.Scanner;
3
4 public class Main06 {
5     Run | Debug
6     public static void main(String[] args) {
7         DoubleLinkedList06 dll = new DoubleLinkedList06();
8         Scanner sc = new Scanner(System.in);
9
10        while (true) {
11            System.out.println(x: "=====");
12            System.out.println(x: "Data Manipulation with Double Linked List");
13            System.out.println(x: "=====");
14            System.out.println(x: "1. Add First");
15            System.out.println(x: "2. Add Tail");
16            System.out.println(x: "3. Add Data in nth index");
17            System.out.println(x: "4. Remove First");
18            System.out.println(x: "5. Remove Last");
19            System.out.println(x: "6. Print");
20            System.out.println(x: "7. Print");
21            System.out.println(x: "8. Search Data");
22            System.out.println(x: "9. Sort Data");
23            System.out.println(x: "10. Exit");
24            System.out.println(x: "=====");
25            System.out.print(s: "Select menu: ");
26            int menu = sc.nextInt();
27            sc.nextLine();
28
29            switch (menu) {
30                case 1:
31                    System.out.print(s: "Enter data: ");
32                    int data = sc.nextInt();
33                    dll.addFirst(data);
34                    break;
35                case 2:
36                    System.out.print(s: "Enter data: ");
37                    data = sc.nextInt();
38                    dll.addLast(data);
39                    break;
```

```

jobsheet10_DoubleLinkedLists > assignment06 > J Main06.java > ...
4   public class Main06 {
5       public static void main(String[] args) {
39           case 3:
40               System.out.print(s:"Enter data: ");
41               data = sc.nextInt();
42               System.out.print(s:"Enter index: ");
43               int index = sc.nextInt();
44               try {
45                   dll.add(data, index);
46               } catch (Exception e) {
47                   System.out.println("Error: " + e.getMessage());
48               }
49               break;
50           case 4:
51               try {
52                   dll.removeFirst();
53               } catch (Exception e) {
54                   System.out.println("Error: " + e.getMessage());
55               }
56               break;
57           case 5:
58               try {
59                   dll.removeLast();
60               } catch (Exception e) {
61                   System.out.println("Error: " + e.getMessage());
62               }
63               break;
64           case 6:
65               dll.print();
66               break;
67           case 7:
68               dll.print();
69               break;
70           case 8:
71               System.out.print(s:"Enter data to search: ");
72               data = sc.nextInt();
73               if (dll.search(data)) {
74                   System.out.println(x:"Data found");
75               } else {
76                   System.out.println(x:"Data not found");
77               }
78               break;
79           case 9:
80               dll.sortDescending();
81               break;
82           case 10:
83               System.out.println(x:"Exiting...");
84               return;
85           default:

```

2. We are required to create a program which Implement Stack using double linked list. The features are described in following illustrations:

Initial menu and add Data (push)

```

*****
Library data book
*****
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****
1
-----
Insert new book title
-----
Practical Digital Forensics

```

Print All Data

```
*****
Library data book
*****
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****
4
-----
Info all books
-----
3D Computer Vision
Understanding Software
Algorithms Notes for Professionals
Getting Started with C++ Audio Programming for Game Developers
Practical Digital Forensics
```

See the data on top of the stack

```
*****
Library data book
*****
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****
3
-----
Peek book title from top
-----
```

Pop the data from the top of the stack

```
*****
Library data book
*****
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****
2
-----
Book om top has been removed
-----

*****
Library data book
*****
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****
4
-----
Info all books
-----
Understanding Software
Algorithms Notes for Professionals
Getting Started with C++ Audio Programming for Game Developers
Practical Digital Forensics
BUILD SUCCESSFUL (total time: 1 second)
```

Output :

<pre>===== Library Data Book ===== 1. Add new book 2. Get book from top 3. Peek book title from top 4. Info all books 5. Exit ===== Select menu: 1 ----- Insert new book title ----- Enter book title: Practical Digital Forensics ===== Library Data Book ===== 1. Add new book 2. Get book from top 3. Peek book title from top 4. Info all books 5. Exit ===== Select menu: 3 ----- Peek book title from top ----- Book title from top: Practical Digital Forensics</pre>	<pre>===== Library Data Book ===== 1. Add new book 2. Get book from top 3. Peek book title from top 4. Info all books 5. Exit ===== Select menu: 2 ----- Book on top has been removed ----- Book from top: Practical Digital Forensics ===== Library Data Book ===== 1. Add new book 2. Get book from top 3. Peek book title from top 4. Info all books 5. Exit ===== Select menu: 4 ----- Info all books ----- Getting Strated with C++ Audio Programming for Game Developers Algorithms Notes for Professionals Understanding Software 3D Computer Vision</pre>
--	---

Code :

```
jobsheet10_DoubleLinkedLists > assignment206 > J Node06.java > ...  
Click here to ask Blackbox to help you code faster  
1 package assignment206;  
2  
3 public class Node06 {  
4     String data;  
5     Node06 prev, next;  
6  
7     Node06(Node06 prev, String data, Node06 next) {  
8         this.prev = prev;  
9         this.data = data;  
10        this.next = next;  
11    }  
12 }  
  
jobsheet10_DoubleLinkedLists > assignment206 > J Stack06.java > Stack06 > print()  
Click here to ask Blackbox to help you code faster  
1 package assignment206;  
2  
3 public class Stack06 {  
4     Node06 head;  
5     int size;  
6  
7     public Stack06() {  
8         head = null;  
9         size = 0;  
10    }  
11  
12    public boolean isEmpty() {  
13        return head == null;  
14    }  
15  
16    public void push(String item) {  
17        if (isEmpty()) {  
18            head = new Node06(prev:null, item, next:null);  
19        } else {  
20            Node06 newNode = new Node06(prev:null, item, head);  
21            head.prev = newNode;  
22            head = newNode;  
23        }  
24        size++;  
25    }  
26  
27    public String pop() throws Exception {  
28        if (isEmpty()) {  
29            throw new Exception(message:"Stack is empty, cannot pop!");  
30        } else if (size == 1) {  
31            String data = head.data;  
32            head = null;  
33            size--;  
34            return data;  
35        } else {  
36            String data = head.data;  
37            head = head.next;  
38            head.prev = null;  
39            size--;  
40            return data;  
41        }  
42    }  
}
```

jobsheet10_DoubleLinkedLists > assignment206 > J Stack06.java > ...

```
3 public class Stack06 {
44     public String peek() throws Exception {
45         if (isEmpty()) {
46             throw new Exception(message:"Stack is empty, cannot peek!");
47         }
48         return head.data;
49     }
50
51     public void print() {
52         if (!isEmpty()) {
53             Node06 tmp = head;
54             while (tmp != null) {
55                 System.out.println(tmp.data + " ");
56                 tmp = tmp.next;
57             }
58             System.out.println();
59         } else {
60             System.out.println(x:"Stack is empty");
61         }
62     }
63 }
```

jobsheet10_DoubleLinkedLists > assignment206 > J StackMain06.java > ...

Click here to ask Blackbox to help you code faster

```
1 package assignment206;
2 import java.util.Scanner;
3
4 public class StackMain06 {
5     Run | Debug
6     public static void main(String[] args) {
7         Stack06 stack = new Stack06();
8         Scanner sc = new Scanner(System.in);
9
10        while (true) {
11            System.out.println(x:"=====");
12            System.out.println(x:"          Library Data Book          ");
13            System.out.println(x:"=====");
14            System.out.println(x:"1. Add new book");
15            System.out.println(x:"2. Get book from top");
16            System.out.println(x:"3. Peek book title from top");
17            System.out.println(x:"4. Info all books");
18            System.out.println(x:"5. Exit");
19            System.out.println(x:"=====");
20            System.out.print(s:"Select menu: ");
21            int menu = sc.nextInt();
22            sc.nextLine();
23
24            switch (menu) {
25                case 1:
26                    System.out.println(x:"-----");
27                    System.out.println(x:"  Insert new book title  ");
28                    System.out.println(x:"-----");
29                    System.out.print(s:"Enter book title: ");
30                    String title = sc.nextLine();
31                    stack.push(title);
32                    break;
33                case 2:
34                    System.out.println(x:"-----");
35                    System.out.println(x:"Book on top has been removed");
36                    System.out.println(x:"-----");
37                    try {
38                        String book = stack.pop();
39                        System.out.println("Book from top: " + book);
40                    } catch (Exception e) {
41                        System.out.println("Error: " + e.getMessage());
42                    }
43                    break;
44            }
45        }
46    }
47 }
```



```

jobsheet10_DoubleLinkedLists > assignment206 > J StackMain06.java > ...
4  public class StackMain06 {
5      public static void main(String[] args) {
43         case 3:
44             System.out.println(x:"-----");
45             System.out.println(x:" Peek book title from top ");
46             System.out.println(x:"-----");
47             try {
48                 String book = stack.peek();
49                 System.out.println("Book title from top: " + book);
50             } catch (Exception e) {
51                 System.out.println("Error: " + e.getMessage());
52             }
53             break;
54         case 4:
55             System.out.println(x:"-----");
56             System.out.println(x:"      Info all books      ");
57             System.out.println(x:"-----");
58             stack.print();
59             break;
60         case 5:
61             System.out.println(x:"-----!Exiting!-----");
62             return;
63         default:
64             System.out.println(x:"-----Invalid selection!-----");
65     }
66 }
67 }
68 }

```

3. Create a program that helps vaccination process by having a queue algorithm alongside with double linked list as follows (the amount left of queue length in menu print(3) and recent vaccinated person in menu Remove data (2) should be displayed)

Initial menu and adding a data

```

+++++
Extravaganza Vaccine Queue
+++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
+++++
1
Add new vaccine queue
Queue number : 123
Name : Joko

```

Print data (notice the highlighted red in the result)

```
+++++
Extravaganza Vaccine Queue
+++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
+++++
3
+++++
Current vaccine queue :
| No.      | Name      |
| 123      | Joko      |
| 124      | Mely      |
| 135      | Johan     |
| 146      | Rosi      |
Queue left : 4
+++++
```

Remove Data (the highlighted red must displayed in the console too)

```
+++++
Extravaganza Vaccine Queue
+++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
+++++
2
Joko has been vaccinated !
3
+++++
Current vaccine queue :
| No.      | Name      |
| 123      | Joko      |
| 124      | Mely      |
| 135      | Johan     |
| 146      | Rosi      |
Queue left : 3
+++++
```

Output :

```

=====
Extravaganza Vaccine Queue
=====
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
Select menu: 3
Queue Number: 123, Name: Joko
Queue Number: 124, Name: Mely
Queue Number: 135, Name: Johan
Queue Number: 146, Name: Rosi
Queue length: 4
=====
Extravaganza Vaccine Queue
=====
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
Select menu: 2
Recently vaccinated person: Queue Number: 123, Name: Joko
=====
Extravaganza Vaccine Queue
=====
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
Select menu: 3
Queue Number: 124, Name: Mely
Queue Number: 135, Name: Johan
Queue Number: 146, Name: Rosi
Queue length: 3
=====
Extravaganza Vaccine Queue
=====
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
Select menu: 1
Enter queue number: 123
Enter name to add to vaccine queue: Joko
=====

```

Code :

```

jobsheet10_DoubleLinkedLists > assignment306 > J Node06.java > ...
  Click here to ask Blackbox to help you code faster
1  package assignment306;
2
3  public class Node06 {
4      int queueNumber;
5      String name;
6      Node06 prev, next;
7
8      Node06(Node06 prev, int queueNumber, String name, Node06 next) {
9          this.prev = prev;
10         this.queueNumber = queueNumber;
11         this.name = name;
12         this.next = next;
13     }
14 }

```

jobsheet10_DoubleLinkedLists > assignment306 > J Queue06.java > ...

Click here to ask Blackbox to help you code faster

```
1 package assignment306;
2
3 public class Queue06 {
4     Node06 head, tail;
5     int size;
6
7     public Queue06() {
8         head = tail = null;
9         size = 0;
10    }
11
12    public boolean isEmpty() {
13        return head == null;
14    }
15
16    public void enqueue(int queueNumber, String name) {
17        if (isEmpty()) {
18            head = tail = new Node06(prev:null, queueNumber, name, next:null);
19        } else {
20            Node06 newNode = new Node06(tail, queueNumber, name, next:null);
21            tail.next = newNode;
22            tail = newNode;
23        }
24        size++;
25    }
26
27    public Node06 dequeue() throws Exception {
28        if (isEmpty()) {
29            throw new Exception(message:"Queue is empty, cannot dequeue!");
30        } else if (size == 1) {
31            Node06 temp = head;
32            head = tail = null;
33            size--;
34            return temp;
35        } else {
36            Node06 temp = head;
37            head = head.next;
38            head.prev = null;
39            size--;
40            return temp;
41        }
42    }
}
```

jobsheet10_DoubleLinkedLists > assignment306 > J Queue06.java > ...

```
3 public class Queue06 {
44    public void printQueue() {
45        if (!isEmpty()) {
46            Node06 tmp = head;
47            while (tmp != null) {
48                System.out.println("Queue Number: " + tmp.queueNumber + ", Name: " + tmp.name);
49                tmp = tmp.next;
50            }
51            System.out.println("Queue length: " + size);
52        } else {
53            System.out.println(x:"Queue is empty");
54        }
55    }
56
57    public int getSize() {
58        return size;
59    }
60
61    public String getFront() throws Exception {
62        if (isEmpty()) {
63            throw new Exception(message:"Queue is empty!");
64        }
65        return "Queue Number: " + head.queueNumber + ", Name: " + head.name;
66    }
67 }
```

```

jobsheet10_DoubleLinkedLists > assignment306 > QueueMain06.java > ...
  Click here to ask Blackbox to help you code faster
1  package assignment306;
2  import java.util.Scanner;
3
4  public class QueueMain06 {
5      Run | Debug
6      public static void main(String[] args) {
7          Queue06 queue = new Queue06();
8          Scanner sc = new Scanner(System.in);
9
10         while (true) {
11             System.out.println(x:"=====");
12             System.out.println(x:"      Extravaganza Vaccine Queue      ");
13             System.out.println(x:"=====");
14             System.out.println(x:"1. Add Vaccine queue");
15             System.out.println(x:"2. Remove Vaccine queue");
16             System.out.println(x:"3. Display vaccine queue");
17             System.out.println(x:"4. Exit");
18             System.out.print(s:"Select menu: ");
19             int menu = sc.nextInt();
20             sc.nextLine();
21
22             switch (menu) {
23                 case 1:
24                     System.out.print(s:"Enter queue number: ");
25                     int queueNumber = sc.nextInt();
26                     sc.nextLine();
27                     System.out.print(s:"Enter name to add to vaccine queue: ");
28                     String name = sc.nextLine();
29                     queue.enqueue(queueNumber, name);
30                     break;
31                 case 2:
32                     try {
33                         Node06 removedPerson = queue.dequeue();
34                         System.out.println("Recently vaccinated person: Queue Number: " + removedPerson.queueNumber + ", Name: " + removedPerson.name);
35                     } catch (Exception e) {
36                         System.out.println("Error: " + e.getMessage());
37                     }
38                     break;
39                 case 3:
40                     queue.printQueue();
41                     break;
42                 case 4:
43                     System.out.println(x:"Exiting...");
44                     return;
45                 default:
46                     System.out.println(x:"Invalid selection!");
47             }
48         }
49     }
50 }

```

4. Create a program implementation that list students score. Each student's data consist of their nim, name, and gpa. The program should implement double linked list and should be able to search based on NIM and sort the GPA in descending order. **Students class must be implemented in this program**

Initial menu and adding data

```
=====
Student Data Management System
=====
```

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit

```
=====
1
Insert NIM in head position
NIM : 123
Name : Anang
GPA : 2.77
=====
```

```
=====
Student Data Management System
=====
```

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit

```
=====
3
Insert student's data node
NIM : 743
Name : Freddy
GPA : 2.90
In index : 3
=====
```

```
=====
Student Data Management System
=====
```

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit

```
=====
2
Insert NIM in tail position
NIM : 233
Name : Suparjo
GPA : 3.67
=====
```

Printing data

```
=====
Student Data Management System
=====

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====

7
NIM : 123
Name : Anang
GPA : 2.77
Insert NIM in tail position
NIM : 233
Name : Suparjo
GPA : 3.67
Insert student's data node
NIM : 743
Name : Freddy
GPA : 2.90
In index : 3
All data printed successfully
```

Searching data

```
=====
Student Data Management System
=====

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====

8
Insert NIM to be searched : 565
Data 565 is in node - 5
Identity :
NIM : 565
Name : Ahmad
GPA : 3.80
```

Sorting data

```
=====
Student Data Management System
=====
```

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit

```
=====
9
```

```
=====
Student Data Management System
=====
```

1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit

```
=====
7
```

```
NIM : 233
Name : Suparjo
GPA : 3.67
NIM : 743
Name : Freddy
GPA : 2.90
NIM : 123
Name : Anang
GPA : 2.77
```


Output :

```
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 1
Enter NIM: 123
Enter Name: Anang
Enter GPA: 2.77
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 2
Enter NIM: 233
Enter Name: Suparjo
Enter GPA: 3.67
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 3
Enter index to insert data: 2
Enter NIM: 743
Enter Name: Freddy
Enter GPA: 2.90
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 7
NIM: 123, Name: Anang, GPA: 2.77
NIM: 233, Name: Suparjo, GPA: 3.67
NIM: 743, Name: Freddy, GPA: 2.9
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 8
Enter NIM to search: 743
Student found: NIM: 743, Name: Freddy, GPA: 2.9
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 9
=====
Student Data Management System
=====
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data at specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
=====
Select menu: 7
NIM: 233, Name: Suparjo, GPA: 3.67
NIM: 743, Name: Freddy, GPA: 2.9
NIM: 123, Name: Anang, GPA: 2.77
=====
```

Code :

```
jobsheet10_DoubleLinkedLists > assignment406 > J Node06.java > ...
  Click here to ask Blackbox to help you code faster
1 package assignment406;
2
3 public class Node06 {
4     Student06 data;
5     Node06 prev, next;
6
7     Node06(Node06 prev, Student06 data, Node06 next) {
8         this.prev = prev;
9         this.data = data;
10        this.next = next;
11    }
12 }

jobsheet10_DoubleLinkedLists > assignment406 > J Student06.java > ...
  Click here to ask Blackbox to help you code faster
1 package assignment406;
2
3 public class Student06 {
4     String nim, name;
5     double gpa;
6
7     Student06(String nim, String name, double gpa) {
8         this.nim = nim;
9         this.name = name;
10        this.gpa = gpa;
11    }
12 }

jobsheet10_DoubleLinkedLists > assignment406 > J StudentList06.java > StudentList06 > remoc
  Click here to ask Blackbox to help you code faster
1 package assignment406;
2
3 public class StudentList06 {
4     Node06 head, tail;
5     int size;
6
7     public StudentList06() {
8         head = null;
9         size = 0;
10    }
11
12    public boolean isEmpty() {
13        return head == null;
14    }
15
16    public void addFirst(Student06 student) {
17        if (isEmpty()) {
18            head = new Node06(prev:null, student, next:null);
19        } else {
20            Node06 newNode = new Node06(prev:null, student, head);
21            head.prev = newNode;
22            head = newNode;
23        }
24        size++;
25    }
26
27    public void addLast(Student06 student) {
28        if (isEmpty()) {
29            addFirst(student);
30        } else {
31            Node06 current = head;
32            while (current.next != null) {
33                current = current.next;
34            }
35            Node06 newNode = new Node06(current, student, next:null);
36            current.next = newNode;
37            size++;
38        }
39    }
}
```

```
jobsheet10_DoubleLinkedLists > assignment406 > J StudentList06.java > StudentList06 > removeA
3 public class StudentList06 {
41     public void addAtIndex(int index, Student06 student) {
42         if (index < 0 || index > size) {
43             System.out.println(x:"Invalid index.");
44             return;
45         }
46         if (index == 0) {
47             addFirst(student);
48         } else if (index == size) {
49             addLast(student);
50         } else {
51             Node06 current = head;
52             for (int i = 0; i < index - 1; i++) {
53                 current = current.next;
54             }
55             Node06 newNode = new Node06(current, student, current.next);
56             current.next.prev = newNode;
57             current.next = newNode;
58             size++;
59         }
60     }
61
62     public void removeFirst() {
63         if (isEmpty()) {
64             System.out.println(x:"List is empty.");
65             return;
66         }
67         if (head.next == null) {
68             head = null;
69             size--;
70             return;
71         }
72         head = head.next;
73         head.prev = null;
74         size--;
75     }
}
```

```
jobsheet10_DoubleLinkedLists > assignment406 > J StudentList06.java > Studer
3 public class StudentList06 {
77     public void removeLast() {
78         if (isEmpty()) {
79             System.out.println(x:"List is empty.");
80             return;
81         }
82         if (head.next == null) {
83             head = null;
84             size--;
85             return;
86         }
87         Node06 current = head;
88         while (current.next.next != null) {
89             current = current.next;
90         }
91         current.next = null;
92         size--;
93     }
94
95     public void removeAtIndex(int index) {
96         if (index < 0 || index >= size) {
97             System.out.println(x:"Invalid index.");
98             return;
99         }
100         if (index == 0) {
101             removeFirst();
102         } else if (index == size - 1) {
103             removeLast();
104         } else {
105             Node06 current = head;
106             for (int i = 0; i < index; i++) {
107                 current = current.next;
108             }
109             current.prev.next = current.next;
110             current.next.prev = current.prev;
111             size--;
112         }
113     }
}
```

jobsheet10_DoubleLinkedLists > assignment406 > J StudentList06.java > ...

```
3 public class StudentList06 {
115     public void print() {
116         if (!isEmpty()) {
117             Node06 tmp = head;
118             while (tmp != null) {
119                 System.out.println("NIM: " + tmp.data.nim + ", Name: " + tmp.data.name + ", GPA: " + tmp.data.gpa);
120                 tmp = tmp.next;
121             }
122         } else {
123             System.out.println("Student list is empty");
124         }
125     }
126
127     public Student06 searchByNIM(String nim) {
128         Node06 tmp = head;
129         while (tmp != null) {
130             if (tmp.data.nim.equals(nim)) {
131                 return tmp.data;
132             }
133             tmp = tmp.next;
134         }
135         return null;
136     }
137
138     public void sortByGPA() {
139         if (!isEmpty()) {
140             Node06 current = head, index = null;
141             Student06 temp;
142             while (current != null) {
143                 index = current.next;
144                 while (index != null) {
145                     if (current.data.gpa < index.data.gpa) {
146                         temp = current.data;
147                         current.data = index.data;
148                         index.data = temp;
149                     }
150                     index = index.next;
151                 }
152                 current = current.next;
153             }
154         }
155     }
156 }
```