JOBSHEET 12

Double Linked List

Name
Sherly Lutfi Azkiah Sulistyawati

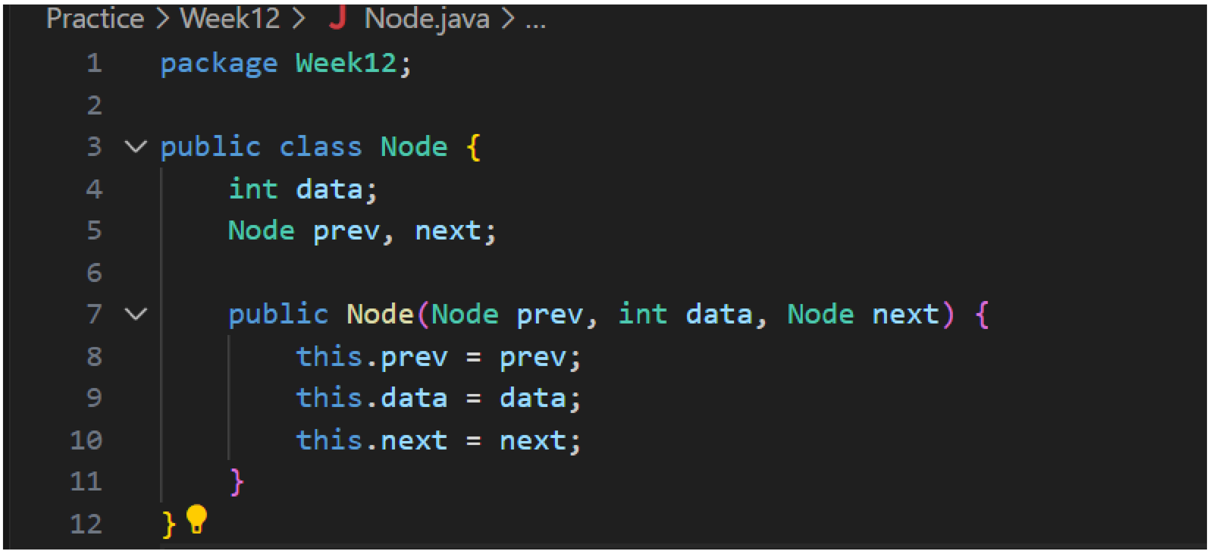
NIM
2341720241

Class
1I

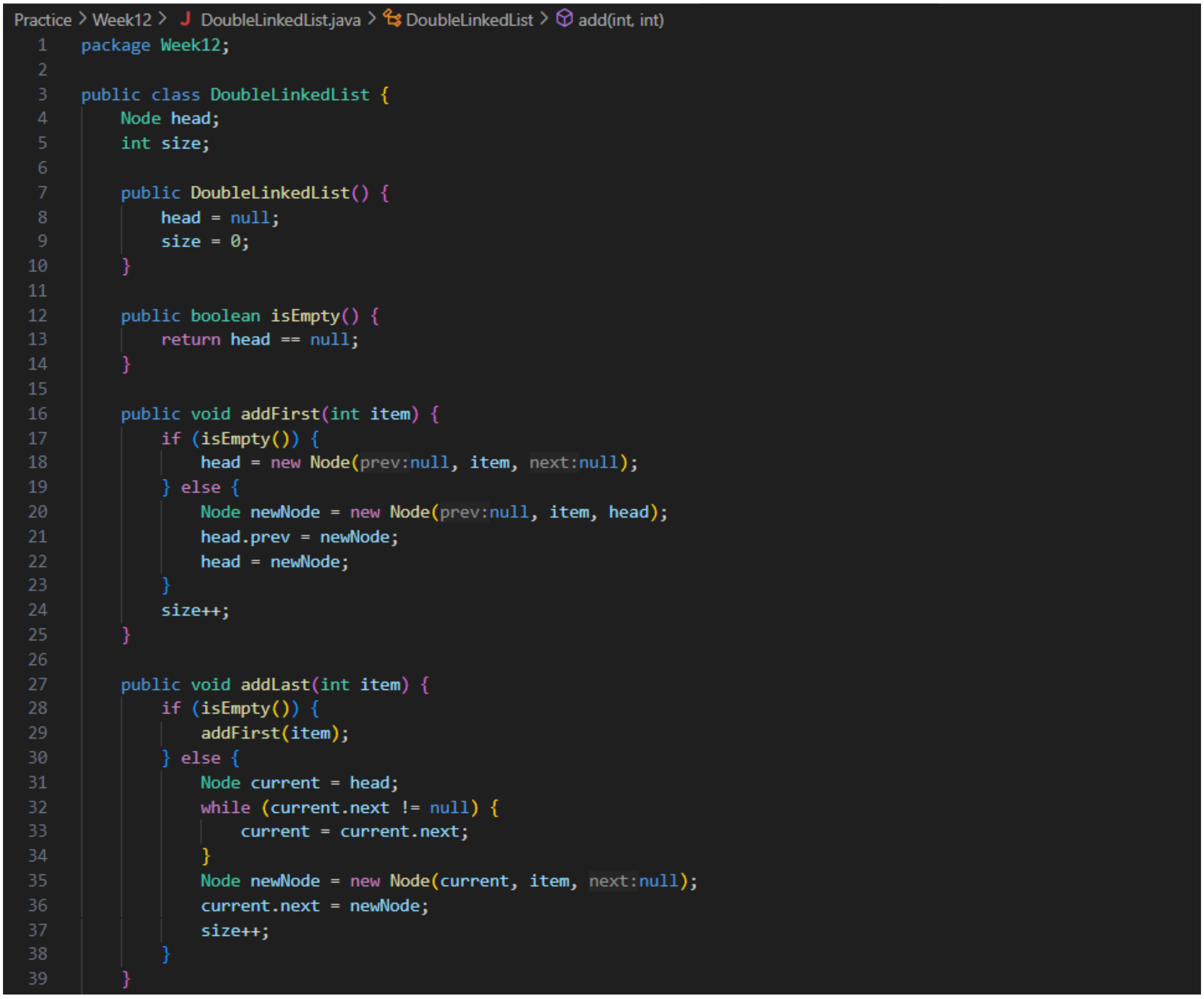
Major
Information Technology

Study Program
D4 Informatics Engineering

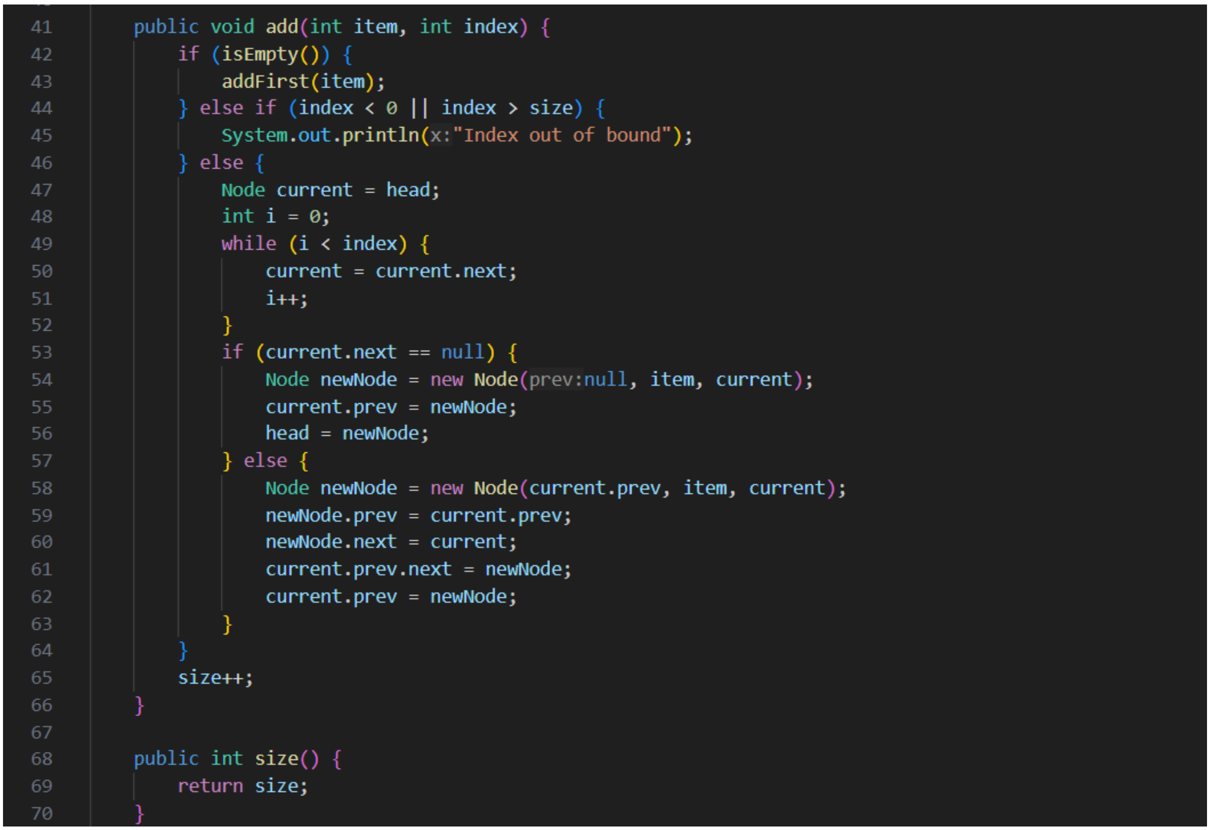
Lab Activity 1



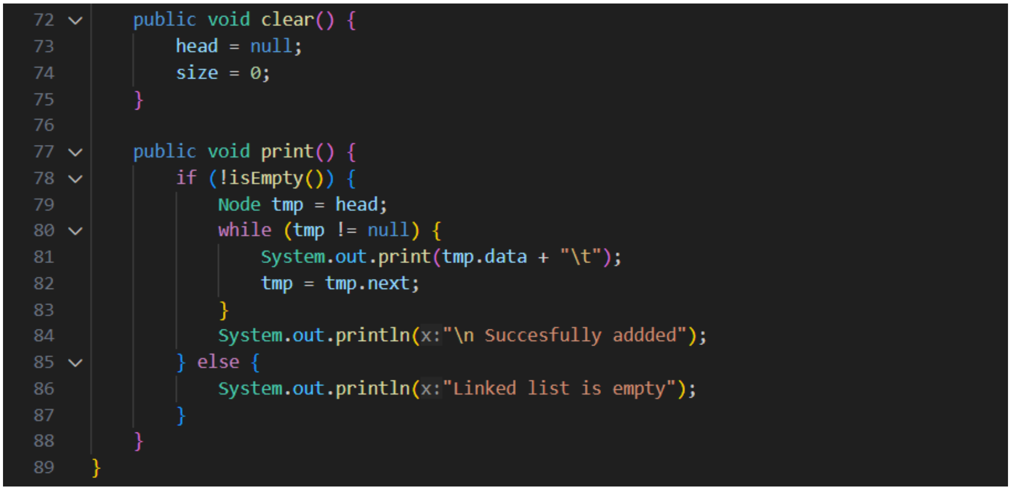
```java
package Week12;

public class Node {
    int data;
    Node prev, next;

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


Practice > Week12 > DoubleLinkedList.java > DoubleLinkedList > add(int, int)

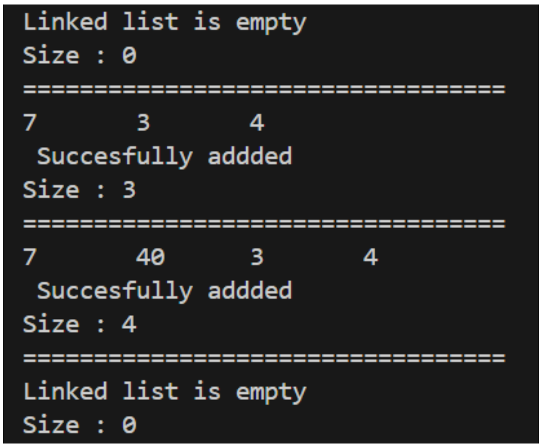
```java
package Week12;

public class DoubleLinkedList {
    Node head;
    int size;

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

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

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

    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, next: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(x:"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(prev: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++;
    }

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



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

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



```
Linked list is empty
Size : 0
==================================
7       3       4
 Succesfully addded
Size : 3
==================================
7       40      3       4
 Succesfully addded
Size : 4
==================================
Linked list is empty
Size : 0
```

Question

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

- **Single Linked List (SLL)**:
  - Each node contains a data part and a pointer to the next node in the sequence.
  - Traversal can only be done in one direction (forward).
  - It is simpler and uses less memory compared to DLL.

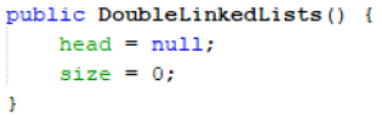
- **Double Linked List (DLL)**:
  - Each node contains a data part, a pointer to the next node, and a pointer to the previous node.
  - Traversal can be done in both directions (forward and backward).
  - It provides more flexibility but uses more memory due to the extra pointer.

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

- **next**:
  - Points to the next node in the linked list.
  - Used to traverse the list in a forward direction.

- **prev**:
  - Points to the previous node in the linked list.
  - Used to traverse the list in a backward direction.

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:** represents the starting point of the linked list. Initializing it to null indicates the list is empty.
- **size:** keeps track of the number of elements in the linked list. Initializing it to 0 indicates the list is empty.

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);

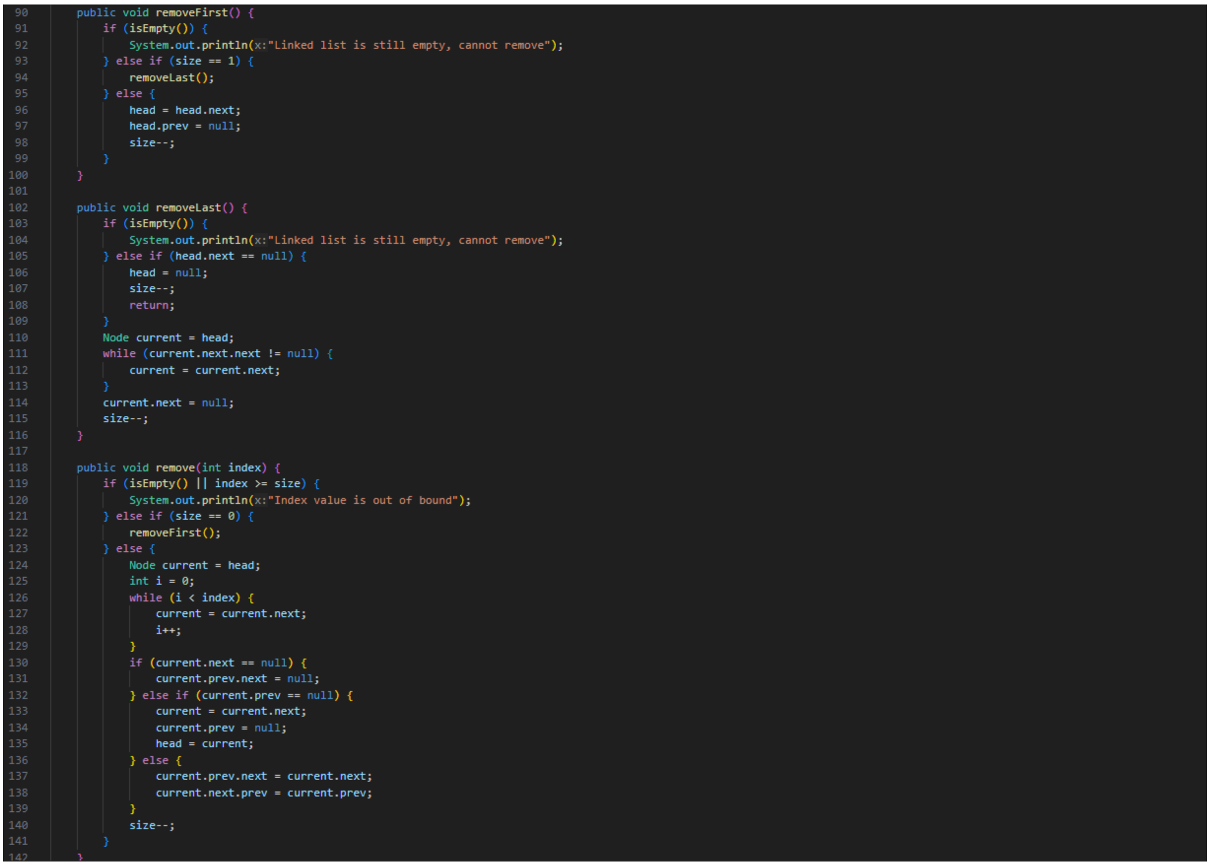
- When adding the first element to the list, the new node's prev is set to null because there are no nodes before it.
- The next is set to head because the new node will be placed at the front, pointing to the current head of the list.

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

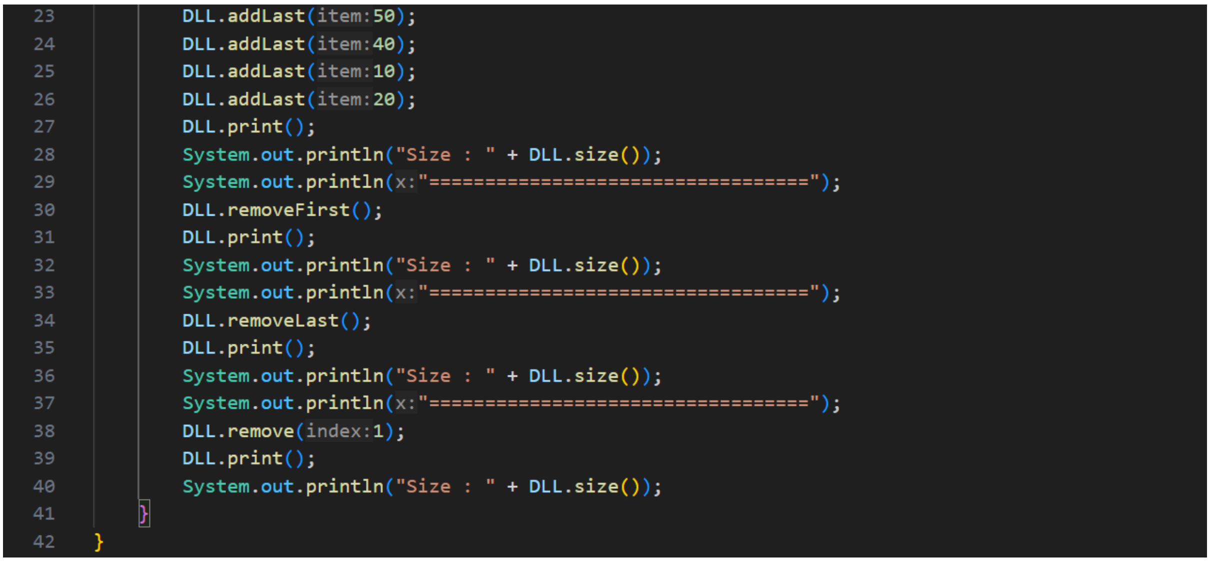
Node newNode = new Node(**current**, item, **null**);

- **current:** This represents the current last node of the list, so the prev pointer of the new node is set to current.
- **null:** The next pointer of the new node is set to null because it will be the new last node in the list.

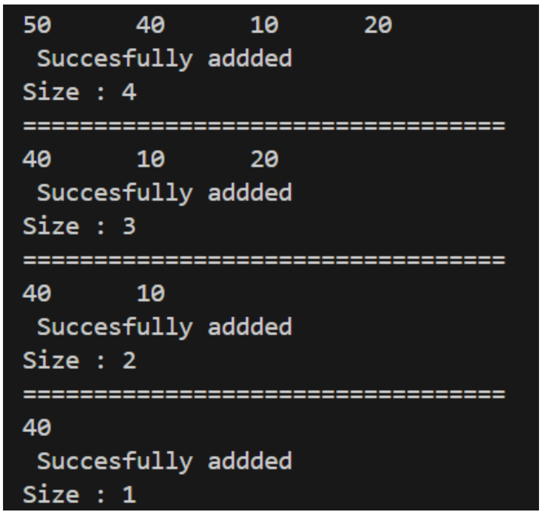
Lab Activity 2



```
    public void removeFirst() {
        if (isEmpty()) {
            System.out.println(x:"Linked list is still empty, cannot remove");
        } else if (size == 1) {
            removeLast();
        } else {
            head = head.next;
            head.prev = null;
            size--;
        }
    }

    public void removeLast() {
        if (isEmpty()) {
            System.out.println(x:"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--;
    }

    public void remove(int index) {
        if (isEmpty() || index >= size) {
            System.out.println(x:"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--;
        }
    }
```



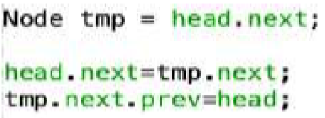
```
        DLL.addLast(item:50);
        DLL.addLast(item:40);
        DLL.addLast(item:10);
        DLL.addLast(item:20);
        DLL.print();
        System.out.println("Size : " + DLL.size());
        System.out.println(x:"==================================");
        DLL.removeFirst();
        DLL.print();
        System.out.println("Size : " + DLL.size());
        System.out.println(x:"==================================");
        DLL.removeLast();
        DLL.print();
        System.out.println("Size : " + DLL.size());
        System.out.println(x:"==================================");
        DLL.remove(index:1);
        DLL.print();
        System.out.println("Size : " + DLL.size());
    }
}
```



```
50      40      10      20
 Succesfully addded
Size : 4
==================================
40      10      20
 Succesfully addded
Size : 3
==================================
40      10
 Succesfully addded
Size : 2
==================================
40
 Succesfully addded
Size : 1
```

Question

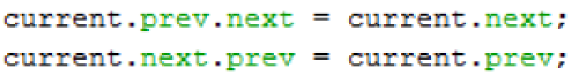
1. What's the meaning of these statements in removeFirst() method?
    - **head = head.next:** Move the head pointer to the next node, effectively removing the first node.
    - **head.prev = null:** Set the prev pointer of the new head node to null because it is now the first node.
    - **size--:** Decrement the size of the list by 1.
2. How do we detect the position of the data that are in the last index in method **removeLast()**?
    - Traverse the list until current.next.next is null, which means current.next is the last node. This allows you to adjust the pointers to remove the last node.
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;
```

    - This code only works if you are removing the second node from the list. It does not handle general cases or edge cases (e.g., if the list has only one node or if the node to be removed is not the second node).

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

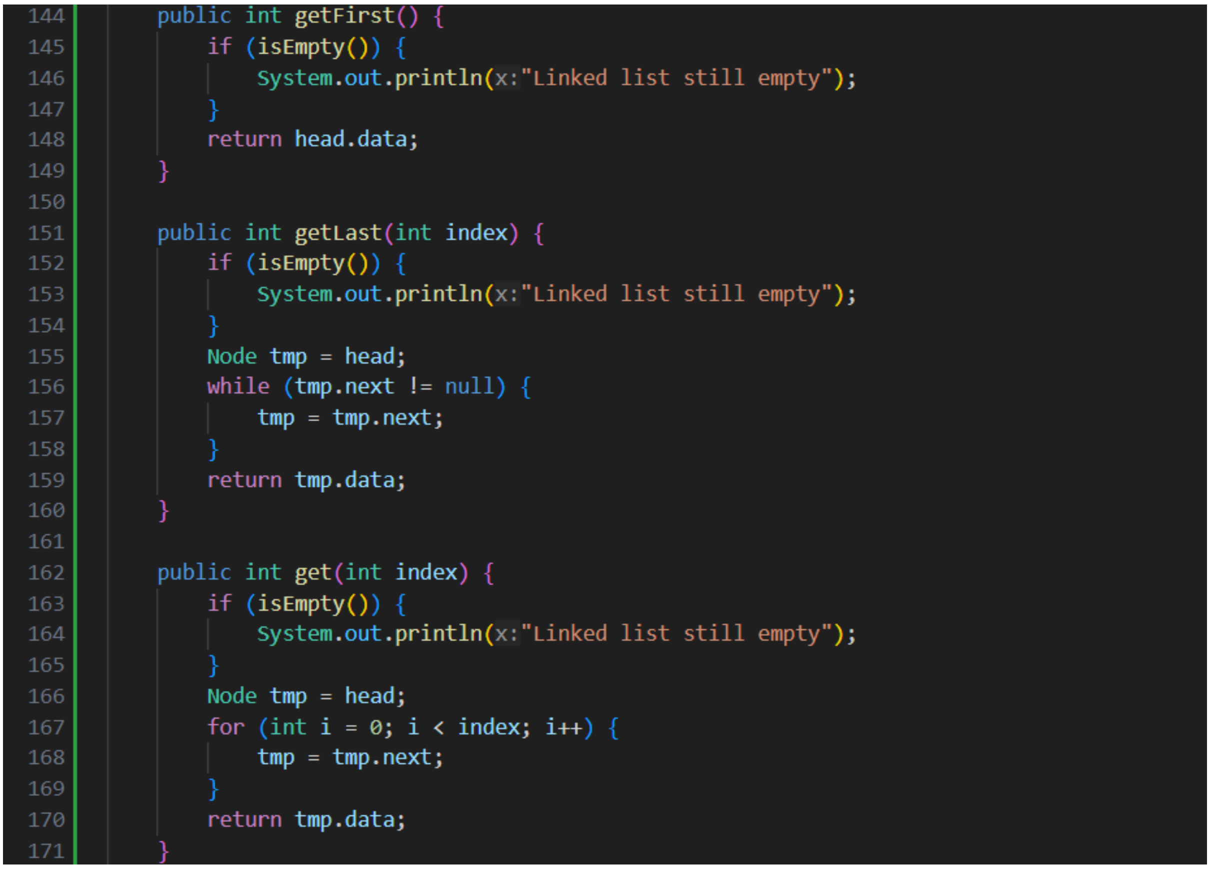


```
current.prev.next = current.next;
current.next.prev = current.prev;
```

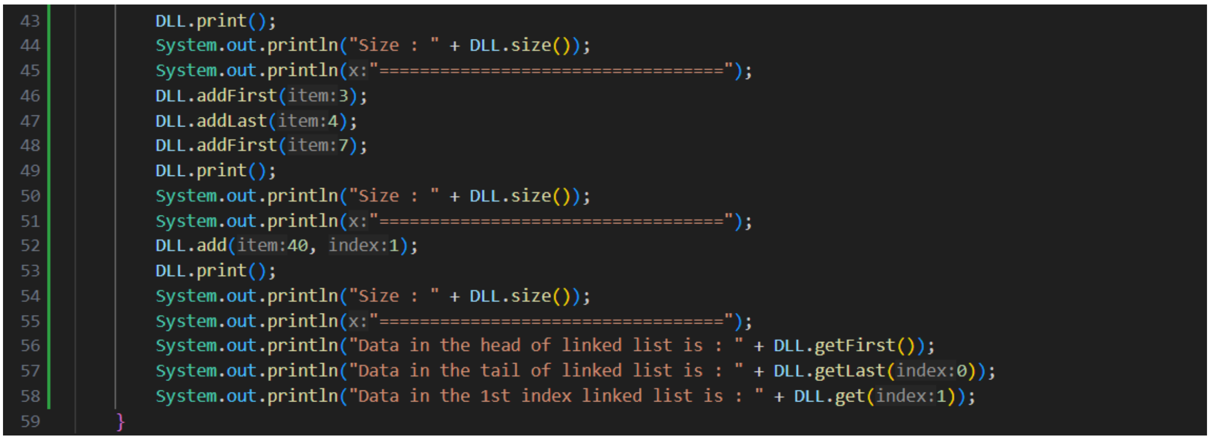
    - **current.prev.next = current.next:** Adjust the next pointer of the previous node to skip the current node and point to the node after the current one.

- **current.next.prev = current.prev:** Adjust the prev pointer of the next node to skip the current node and point to the node before the current one.
- This effectively removes the current node from the list by linking the previous node directly to the next node.

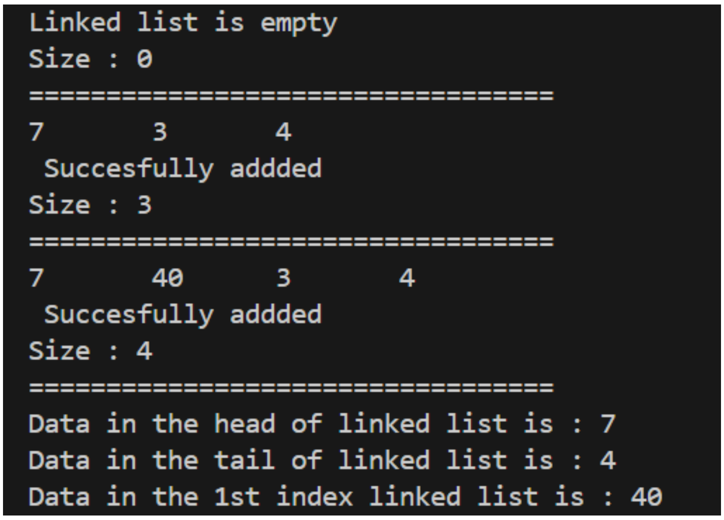
Lab Activity 3



```
    public int getFirst() {
        if (isEmpty()) {
            System.out.println(x:"Linked list still empty");
        }
        return head.data;
    }

    public int getLast(int index) {
        if (isEmpty()) {
            System.out.println(x:"Linked list still empty");
        }
        Node tmp = head;
        while (tmp.next != null) {
            tmp = tmp.next;
        }
        return tmp.data;
    }

    public int get(int index) {
        if (isEmpty()) {
            System.out.println(x:"Linked list still empty");
        }
        Node 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(x:"==================================");
        DLL.addFirst(item:3);
        DLL.addLast(item:4);
        DLL.addFirst(item:7);
        DLL.print();
        System.out.println("Size : " + DLL.size());
        System.out.println(x:"==================================");
        DLL.add(item:40, index:1);
        DLL.print();
        System.out.println("Size : " + DLL.size());
        System.out.println(x:"==================================");
        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(index:0));
        System.out.println("Data in the 1st index linked list is : " + DLL.get(index:1));
    }
```



```
Linked list is empty
Size : 0
==================================
7       3       4
 Succesfully addded
Size : 3
==================================
7       40      3       4
 Succesfully addded
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
```

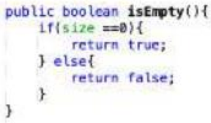
Question

1. What is the function of method **size()** in **DoubleLinkedList** class ?
   - The size() method returns the number of elements in the list
2. How do we set the index in double linked list so that it starts from 1st index instead of 0th index?
   - To make the index start from 1 instead of 0, adjust the indexing logic in the methods where indices are used:

```
public void add(int item, int index) {
    index--; // Decrease index by 1
    // Rest of the code remains the same
}
```

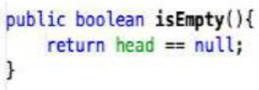
3. Please explain the difference between method **Add()** in double linked list and single linked list !
   - **Double Linked List**:
     - Handles both prev and next pointers.
     - More complex insertion logic to maintain both pointers.
   - **Single Linked List**:
     - Only handles the next pointer.
     - Simpler insertion logic.

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)

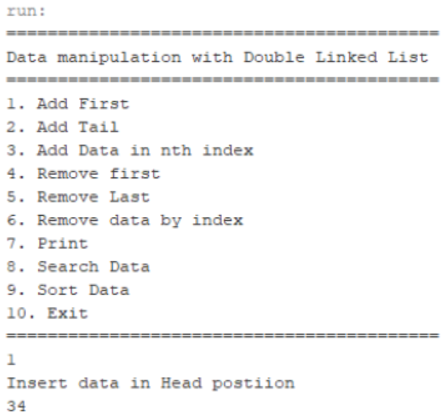
   a) Checks if the size of the list is 0 to determine if the list is empty.

   b) Checks if the head of the list is null to determine if the list is empty.

Assignment

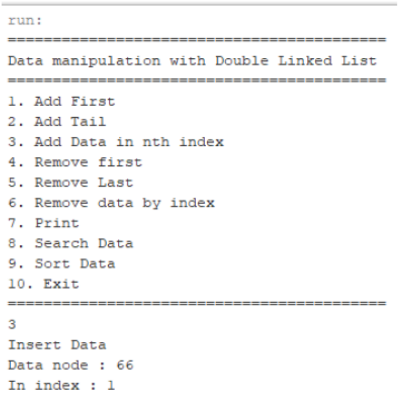
1. Create a program with double linked list implementation that allows user to choose a menu as following image! The searching uses sequential search approach and the program should be able to sort the data in descending order. You may any choose sorting approach you prefer (bubble sort, selection sort,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

Adding a data

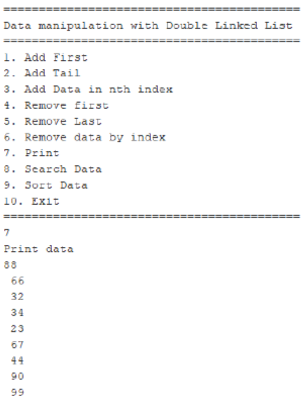


```
run:
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================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

Add data in specified index and display the result

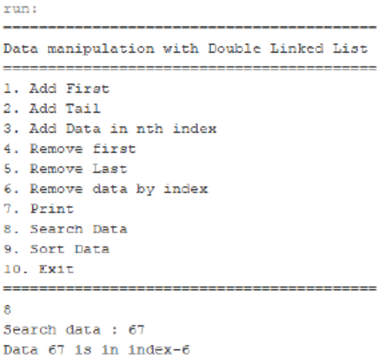


```
run:
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================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



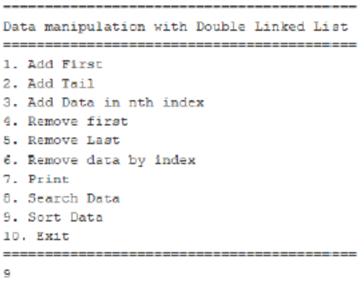
```
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================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

Search Data

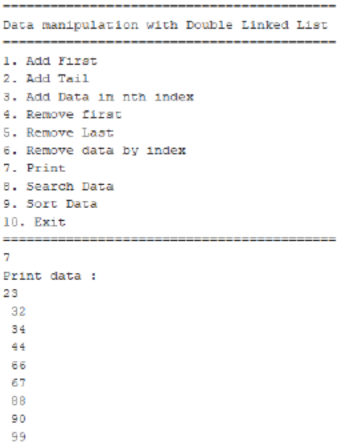


```
run:
-------------------------------------------
Data manipulation with Double Linked List
===========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
===========================================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

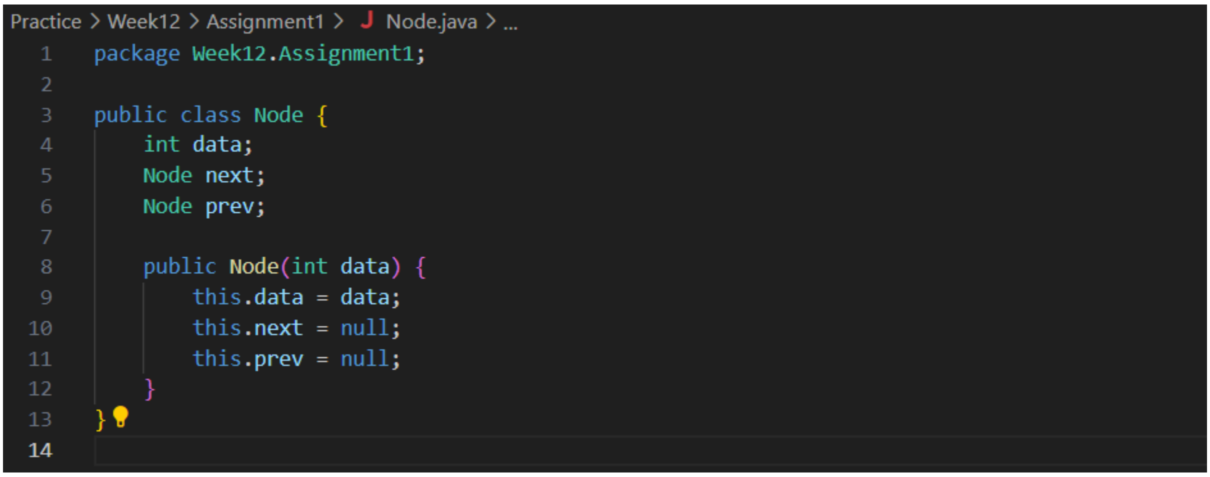
Sorting Data



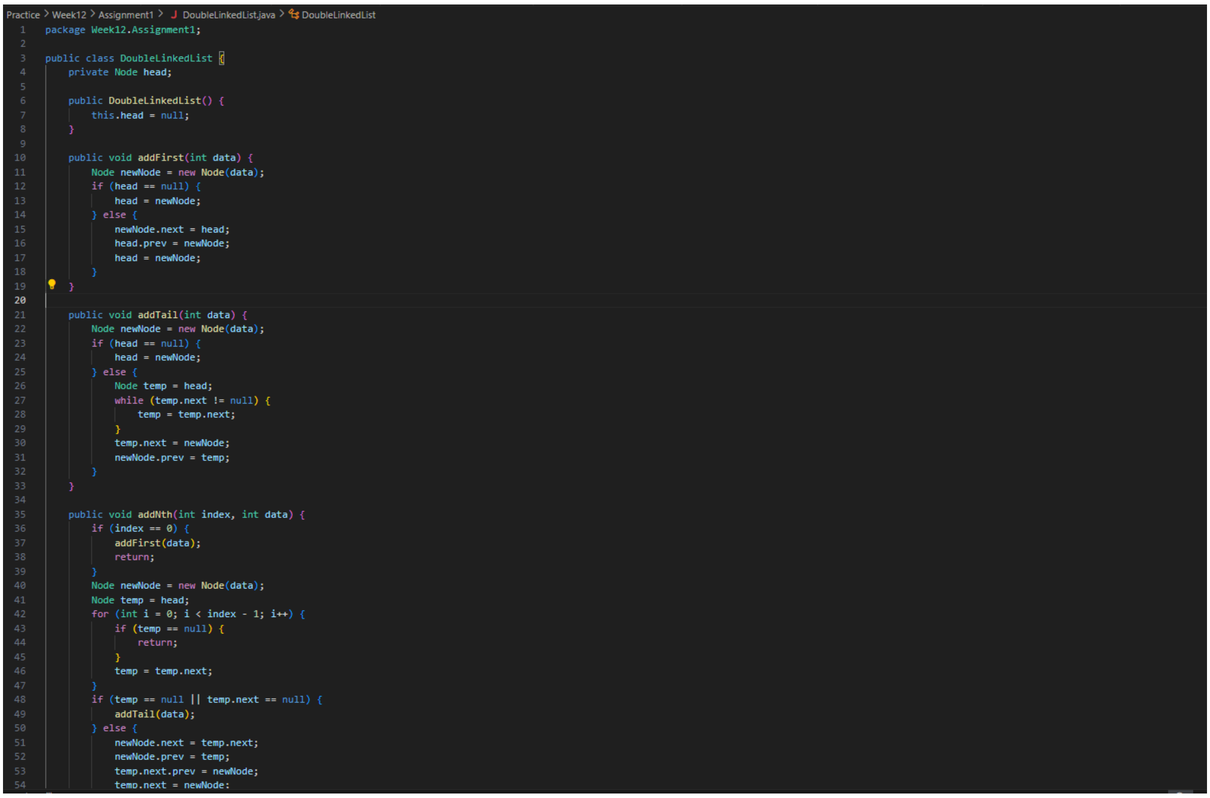
```
-------------------------------------------
Data manipulation with Double Linked List
===========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
===========================================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

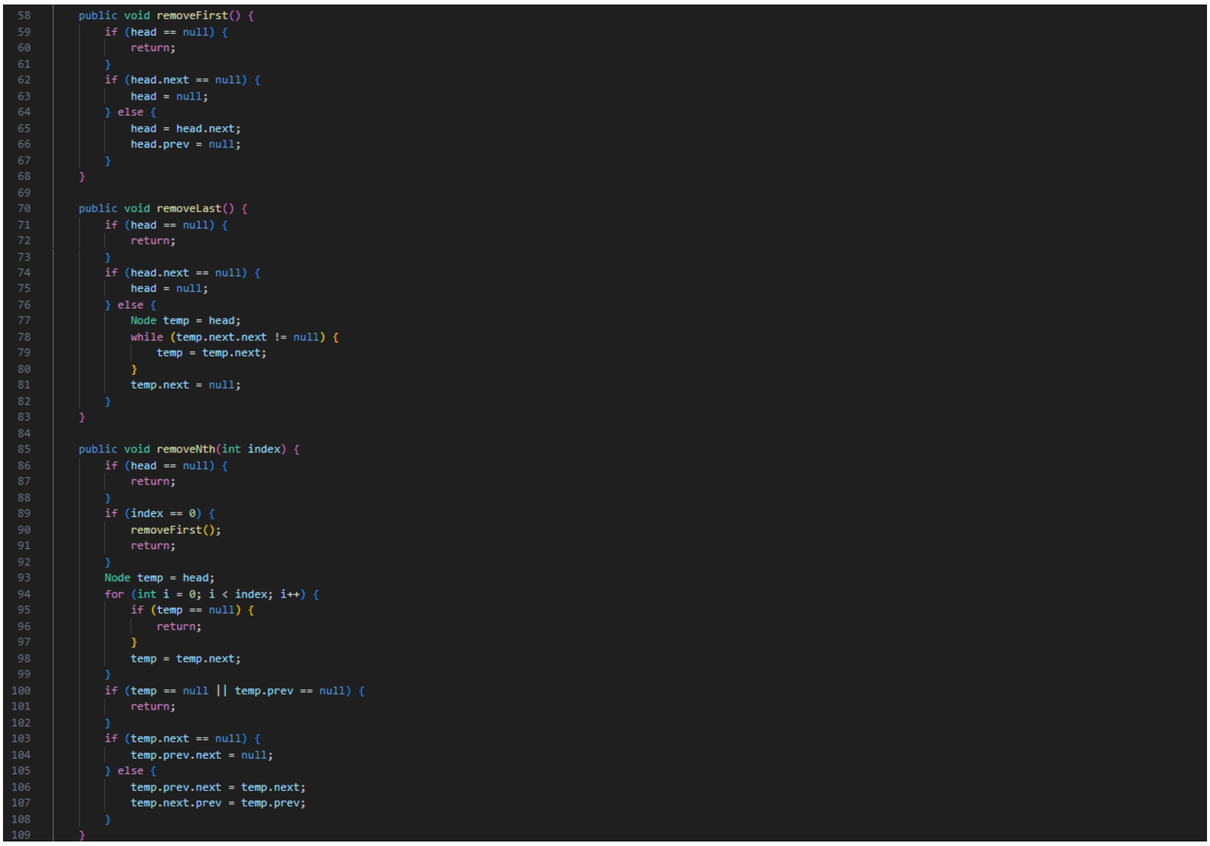


```java
package Week12.Assignment1;

public class Node {
    int data;
    Node next;
    Node prev;

    public Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
```

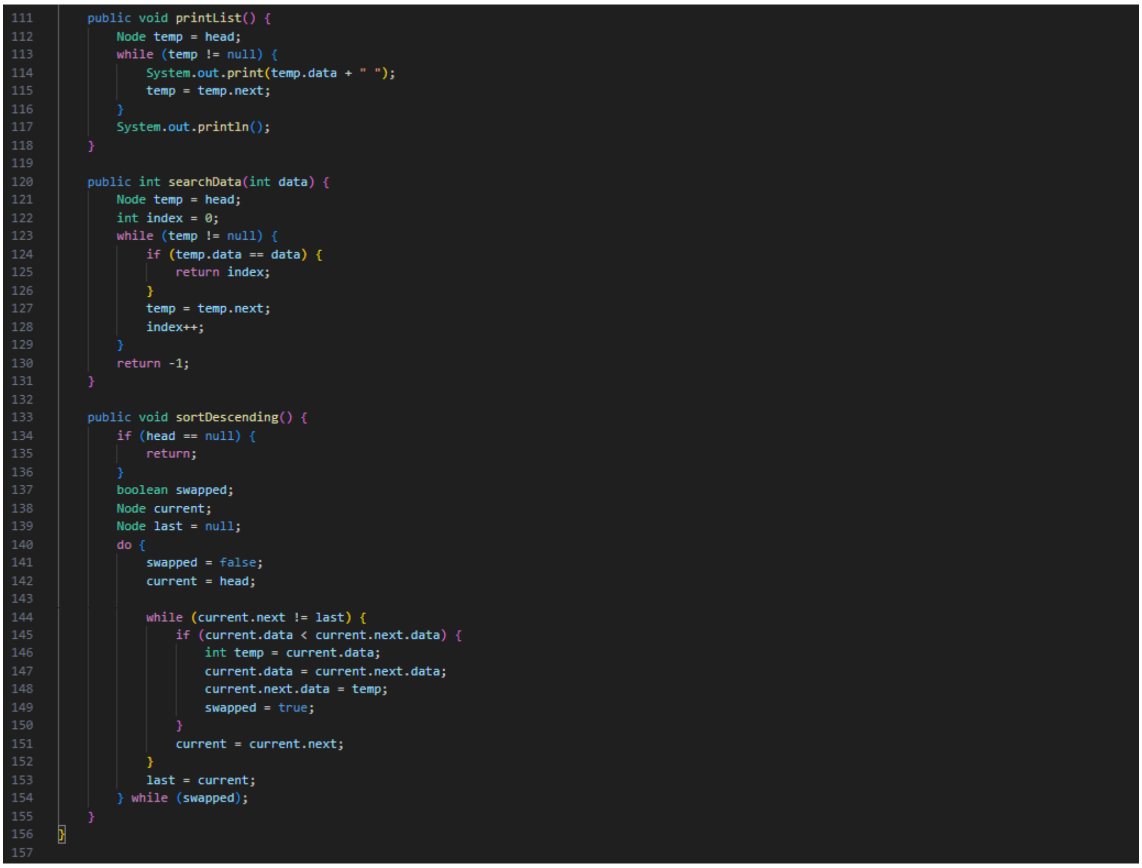


Practice > Week12 > Assignment1 > DoubleLinkedList.java > DoubleLinkedList

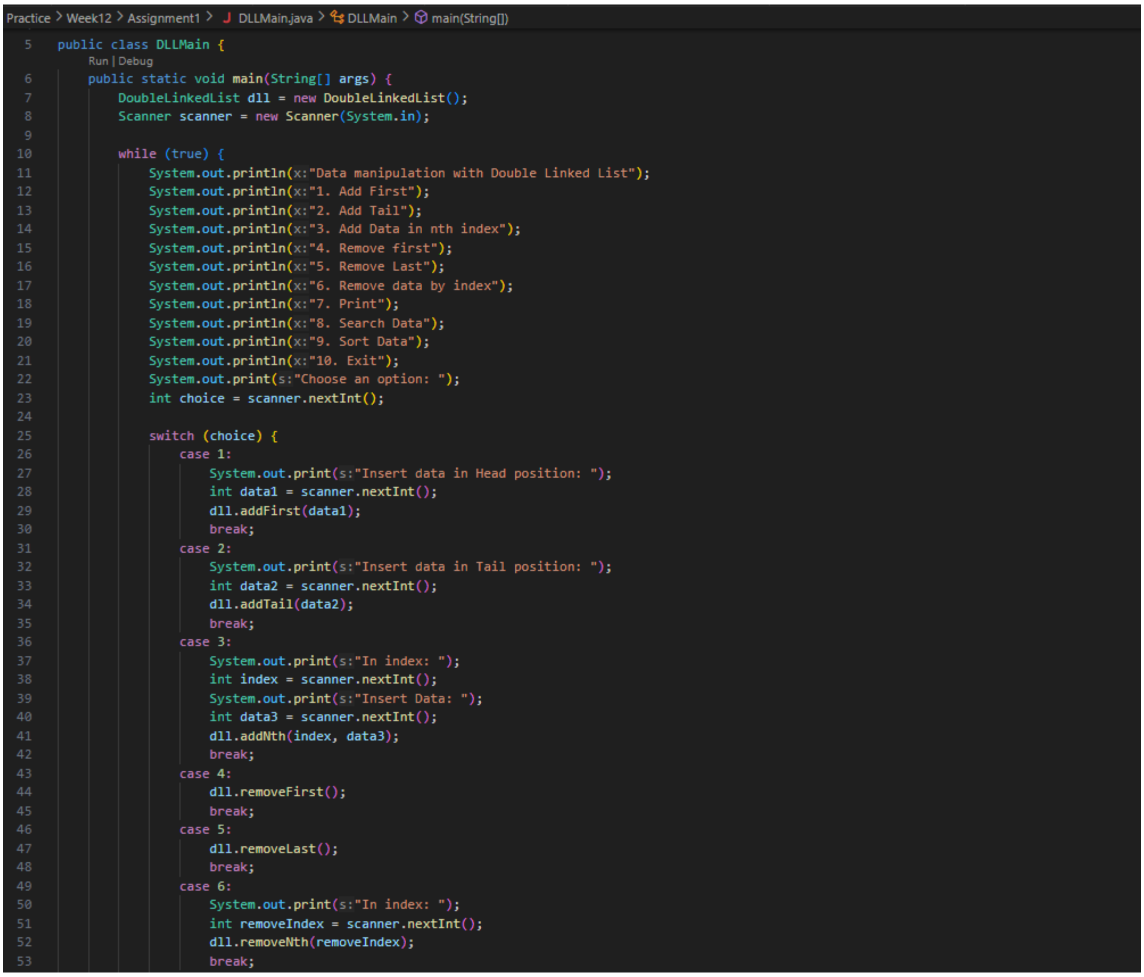
```java
package Week12.Assignment1;

public class DoubleLinkedList {
    private Node head;

    public DoubleLinkedList() {
        this.head = null;
    }

    public void addFirst(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
    }

    public void addTail(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.prev = temp;
        }
    }

    public void addNth(int index, int data) {
        if (index == 0) {
            addFirst(data);
            return;
        }
        Node newNode = new Node(data);
        Node temp = head;
        for (int i = 0; i < index - 1; i++) {
            if (temp == null) {
                return;
            }
            temp = temp.next;
        }
        if (temp == null || temp.next == null) {
            addTail(data);
        } else {
            newNode.next = temp.next;
            newNode.prev = temp;
            temp.next.prev = newNode;
            temp.next = newNode;
```



```java
    public void removeFirst() {
        if (head == null) {
            return;
        }
        if (head.next == null) {
            head = null;
        } else {
            head = head.next;
            head.prev = null;
        }
    }

    public void removeLast() {
        if (head == null) {
            return;
        }
        if (head.next == null) {
            head = null;
        } else {
            Node temp = head;
            while (temp.next.next != null) {
                temp = temp.next;
            }
            temp.next = null;
        }
    }

    public void removeNth(int index) {
        if (head == null) {
            return;
        }
        if (index == 0) {
            removeFirst();
            return;
        }
        Node temp = head;
        for (int i = 0; i < index; i++) {
            if (temp == null) {
                return;
            }
            temp = temp.next;
        }
        if (temp == null || temp.prev == null) {
            return;
        }
        if (temp.next == null) {
            temp.prev.next = null;
        } else {
            temp.prev.next = temp.next;
            temp.next.prev = temp.prev;
        }
    }
```

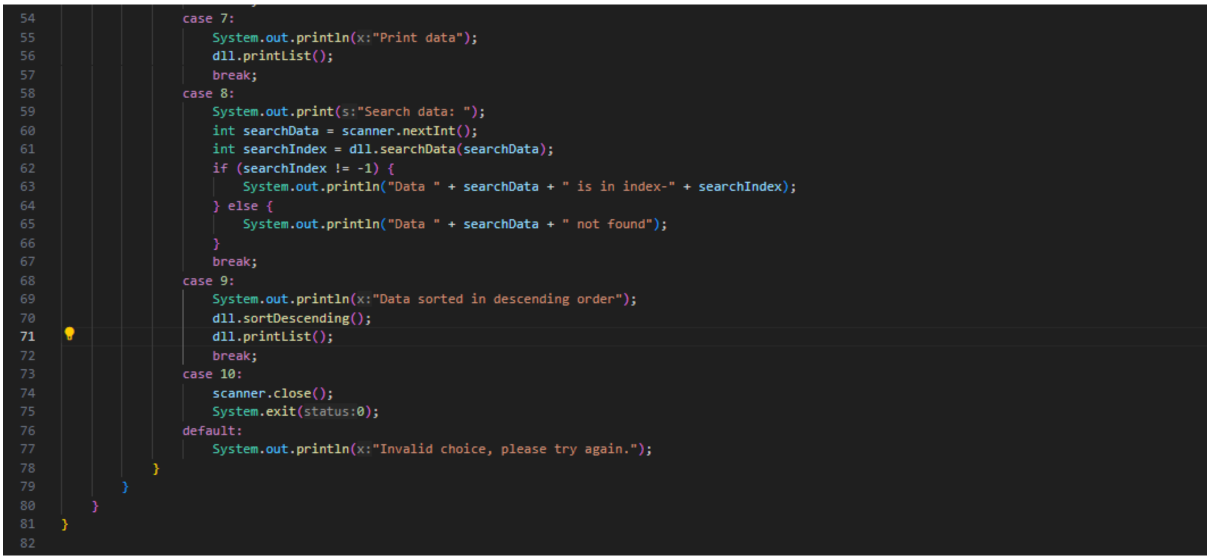


```java
    public void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    public int searchData(int data) {
        Node temp = head;
        int index = 0;
        while (temp != null) {
            if (temp.data == data) {
                return index;
            }
            temp = temp.next;
            index++;
        }
        return -1;
    }

    public void sortDescending() {
        if (head == null) {
            return;
        }
        boolean swapped;
        Node current;
        Node last = null;
        do {
            swapped = false;
            current = head;

            while (current.next != last) {
                if (current.data < current.next.data) {
                    int temp = current.data;
                    current.data = current.next.data;
                    current.next.data = temp;
                    swapped = true;
                }
                current = current.next;
            }
            last = current;
        } while (swapped);
    }
}
```



Practice > Week12 > Assignment1 > DLLMain.java > DLLMain > main(String[])
Run | Debug

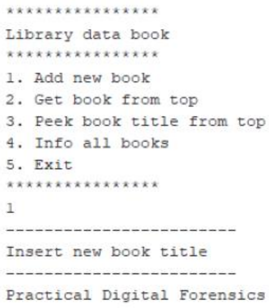
```java
public class DLLMain {
    public static void main(String[] args) {
        DoubleLinkedList dll = new DoubleLinkedList();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println(x:"Data manipulation with Double Linked List");
            System.out.println(x:"1. Add First");
            System.out.println(x:"2. Add Tail");
            System.out.println(x:"3. Add Data in nth index");
            System.out.println(x:"4. Remove first");
            System.out.println(x:"5. Remove Last");
            System.out.println(x:"6. Remove data by index");
            System.out.println(x:"7. Print");
            System.out.println(x:"8. Search Data");
            System.out.println(x:"9. Sort Data");
            System.out.println(x:"10. Exit");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print(s:"Insert data in Head position: ");
                    int data1 = scanner.nextInt();
                    dll.addFirst(data1);
                    break;
                case 2:
                    System.out.print(s:"Insert data in Tail position: ");
                    int data2 = scanner.nextInt();
                    dll.addTail(data2);
                    break;
                case 3:
                    System.out.print(s:"In index: ");
                    int index = scanner.nextInt();
                    System.out.print(s:"Insert Data: ");
                    int data3 = scanner.nextInt();
                    dll.addNth(index, data3);
                    break;
                case 4:
                    dll.removeFirst();
                    break;
                case 5:
                    dll.removeLast();
                    break;
                case 6:
                    System.out.print(s:"In index: ");
                    int removeIndex = scanner.nextInt();
                    dll.removeNth(removeIndex);
                    break;
```



```
                case 7:
                    System.out.println(x:"Print data");
                    dll.printList();
                    break;
                case 8:
                    System.out.print(s:"Search data: ");
                    int searchData = scanner.nextInt();
                    int searchIndex = dll.searchData(searchData);
                    if (searchIndex != -1) {
                        System.out.println("Data " + searchData + " is in index-" + searchIndex);
                    } else {
                        System.out.println("Data " + searchData + " not found");
                    }
                    break;
                case 9:
                    System.out.println(x:"Data sorted in descending order");
                    dll.sortDescending();
                    dll.printList();
                    break;
                case 10:
                    scanner.close();
                    System.exit(status:0);
                default:
                    System.out.println(x:"Invalid choice, please try again.");
            }
        }
    }
}
```

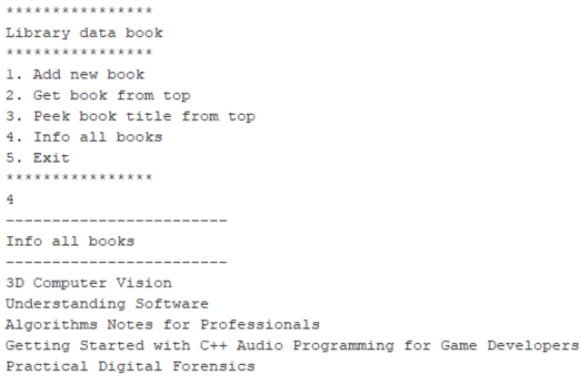
2. We are required to create a program which Implement Stack using double linked list. The features are described in following illustrations:

Initial menu and add Data (push)



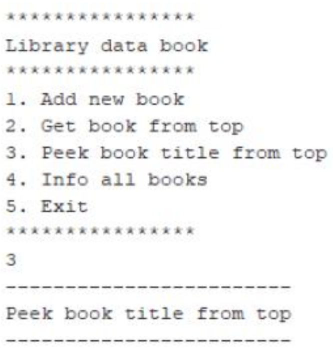
```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
1
------------------------
Insert new book title
------------------------
Practical Digital Forensics
```

Print All Data



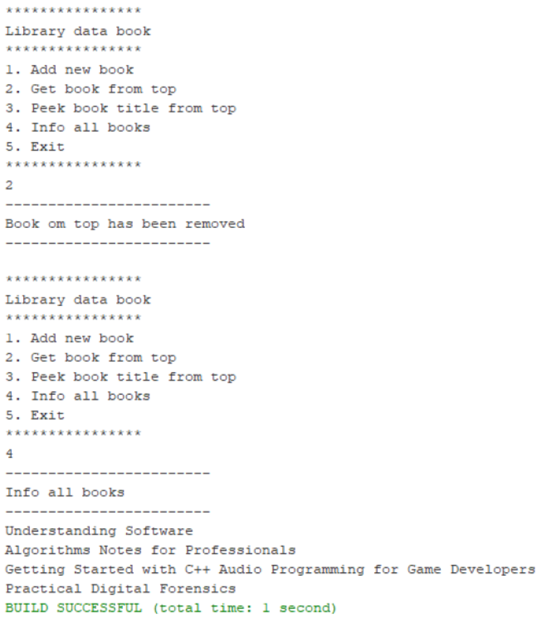
```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
4
------------------------
Info all books
------------------------
3D Computer Vision
Understanding Software
Algorithms Notes for Professionals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

See the data on top of the stack

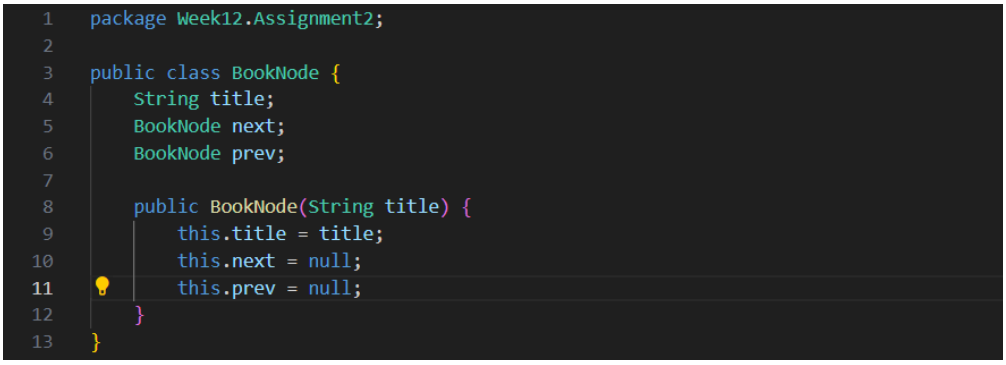


```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
3
------------------------
Peek book title from top
------------------------
```

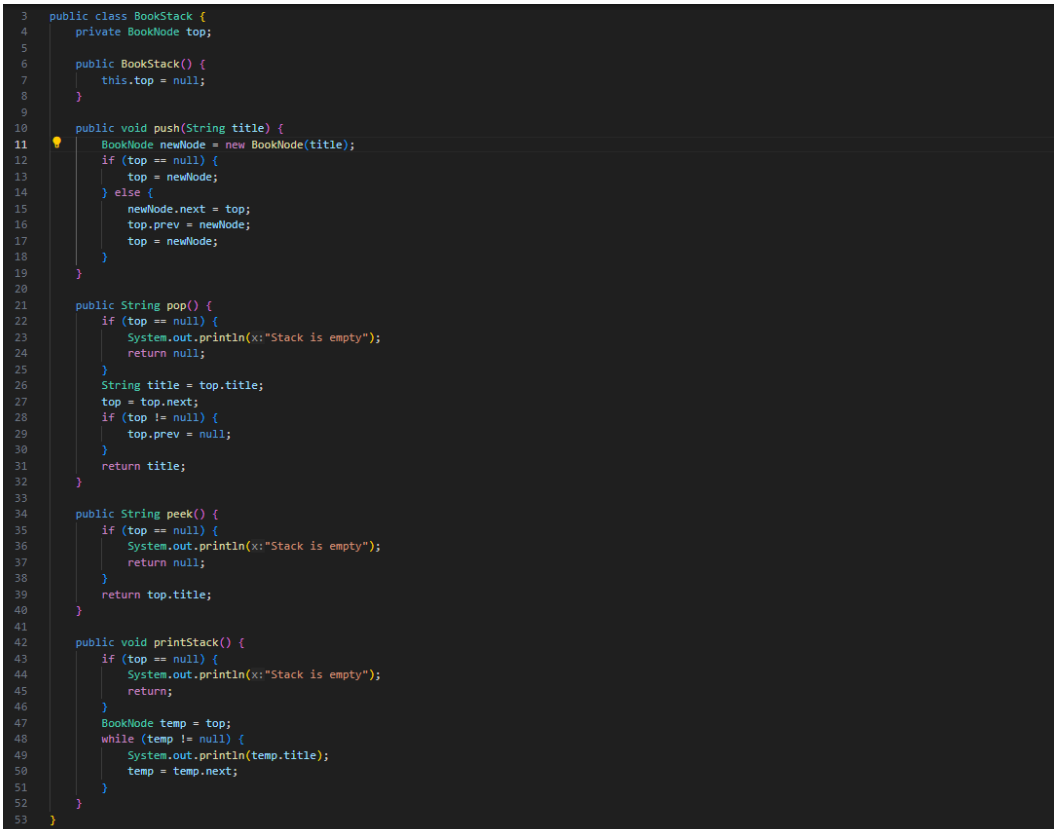
Pop the data from the top of the stack



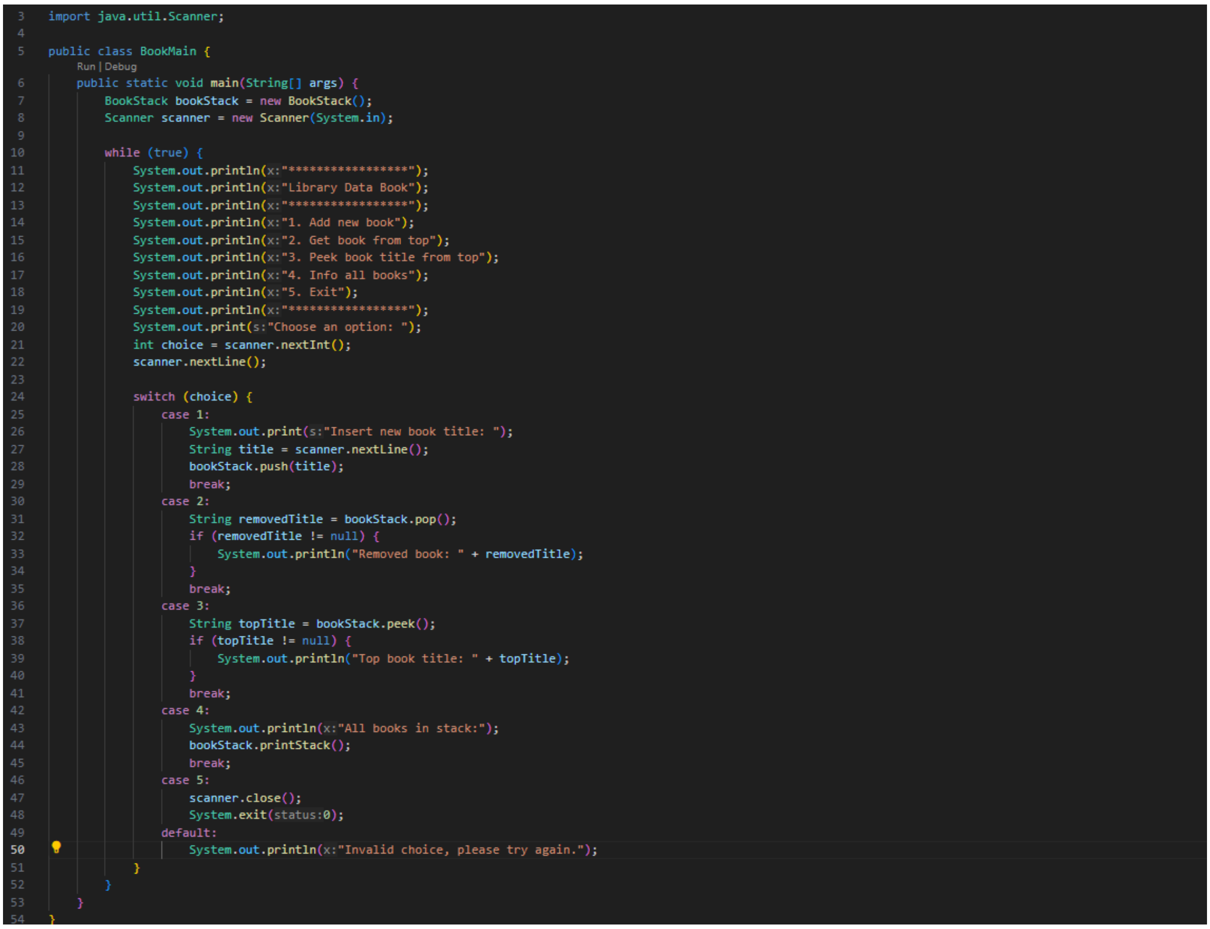
```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
2
------------------------
Book om top has been removed
------------------------

****************
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)
```



```java
package Week12.Assignment2;

public class BookNode {
    String title;
    BookNode next;
    BookNode prev;

    public BookNode(String title) {
        this.title = title;
        this.next = null;
        this.prev = null;
    }
}
```



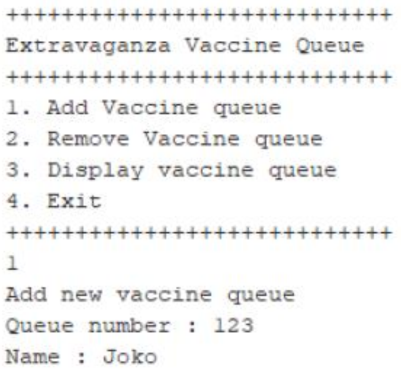
```java
public class BookStack {
    private BookNode top;

    public BookStack() {
        this.top = null;
    }

    public void push(String title) {
        BookNode newNode = new BookNode(title);
        if (top == null) {
            top = newNode;
        } else {
            newNode.next = top;
            top.prev = newNode;
            top = newNode;
        }
    }

    public String pop() {
        if (top == null) {
            System.out.println(x:"Stack is empty");
            return null;
        }
        String title = top.title;
        top = top.next;
        if (top != null) {
            top.prev = null;
        }
        return title;
    }

    public String peek() {
        if (top == null) {
            System.out.println(x:"Stack is empty");
            return null;
        }
        return top.title;
    }

    public void printStack() {
        if (top == null) {
            System.out.println(x:"Stack is empty");
            return;
        }
        BookNode temp = top;
        while (temp != null) {
            System.out.println(temp.title);
            temp = temp.next;
        }
    }
}
```



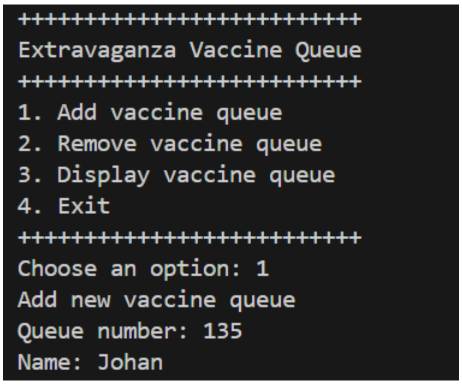
```java
import java.util.Scanner;

public class BookMain {
    Run | Debug
    public static void main(String[] args) {
        BookStack bookStack = new BookStack();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println(x:"*****************");
            System.out.println(x:"Library Data Book");
            System.out.println(x:"*****************");
            System.out.println(x:"1. Add new book");
            System.out.println(x:"2. Get book from top");
            System.out.println(x:"3. Peek book title from top");
            System.out.println(x:"4. Info all books");
            System.out.println(x:"5. Exit");
            System.out.println(x:"*****************");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print(s:"Insert new book title: ");
                    String title = scanner.nextLine();
                    bookStack.push(title);
                    break;
                case 2:
                    String removedTitle = bookStack.pop();
                    if (removedTitle != null) {
                        System.out.println("Removed book: " + removedTitle);
                    }
                    break;
                case 3:
                    String topTitle = bookStack.peek();
                    if (topTitle != null) {
                        System.out.println("Top book title: " + topTitle);
                    }
                    break;
                case 4:
                    System.out.println(x:"All books in stack:");
                    bookStack.printStack();
                    break;
                case 5:
                    scanner.close();
                    System.exit(status:0);
                default:
                    System.out.println(x:"Invalid choice, please try again.");
            }
        }
    }
}
```

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

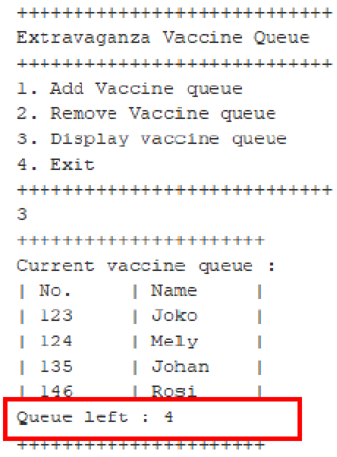


```
++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++
1
Add new vaccine queue
Queue number : 123
Name : Joko
```

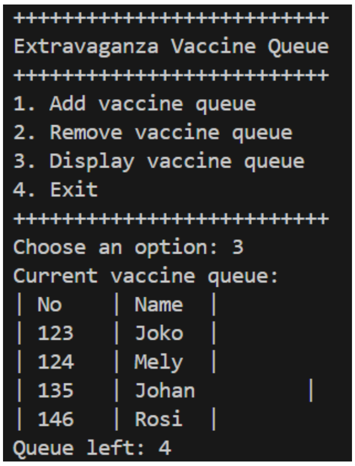


```
++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++
1. Add vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++
Choose an option: 1
Add new vaccine queue
Queue number: 135
Name: Johan
```

Print data (notice the highlighted red in the result)

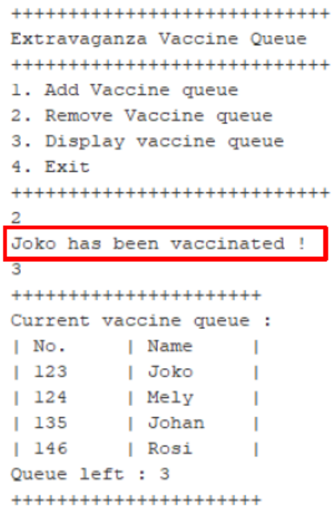


```
++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++
3
++++++++++++++++++++++
Current vaccine queue :
| No.       | Name      |
| 123       | Joko      |
| 124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

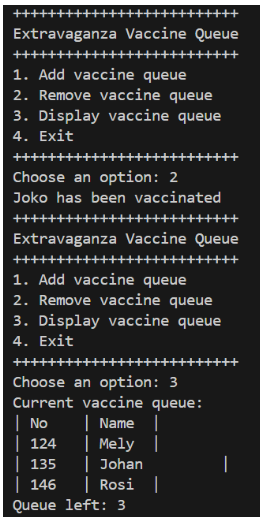


```
++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++
1. Add vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++
Choose an option: 3
Current vaccine queue:
| No     | Name  |
| 123    | Joko  |
| 124    | Mely  |
| 135    |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

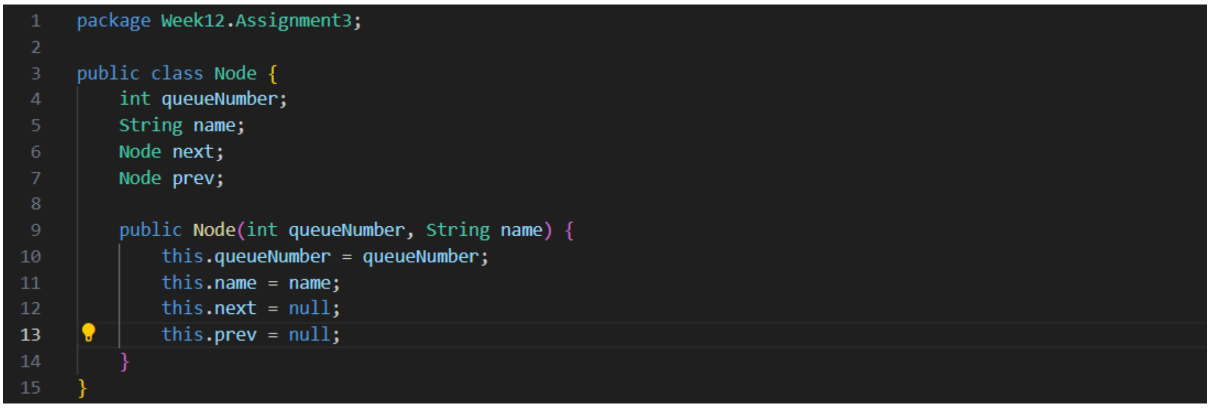
Remove Data (the highlighted red must displayed in the console too)



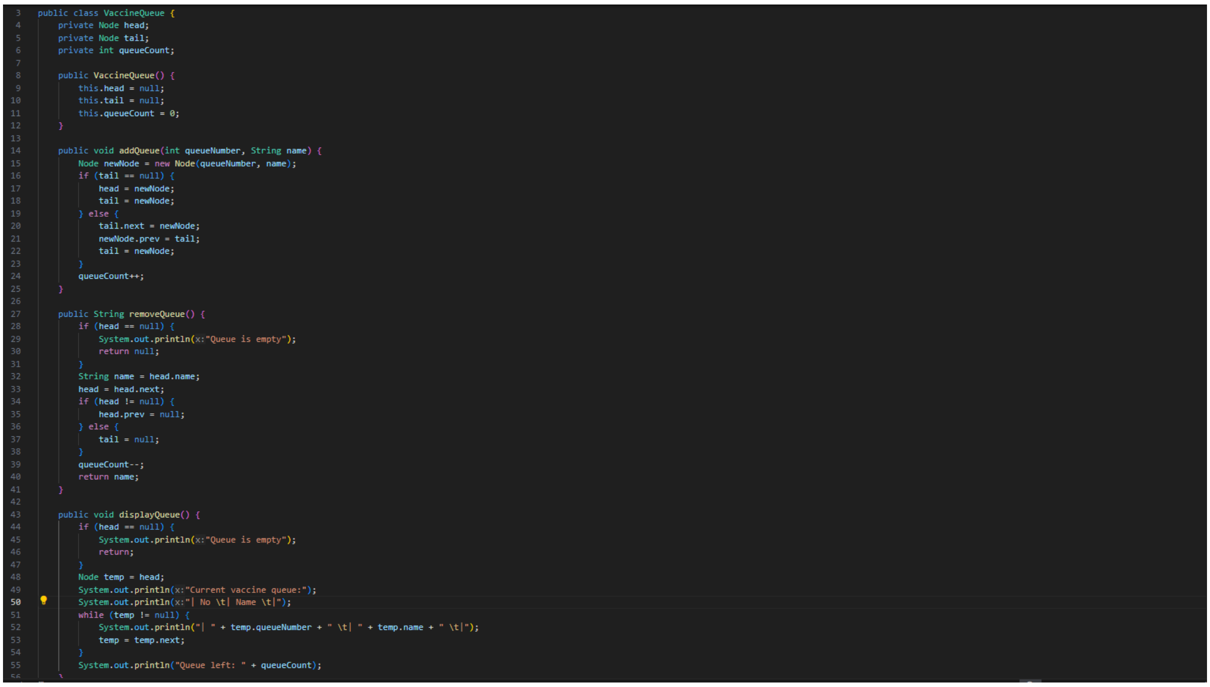
```
++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++
2
Joko has been vaccinated !
3
++++++++++++++++++++++
Current vaccine queue :
| No.       | Name      |
|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



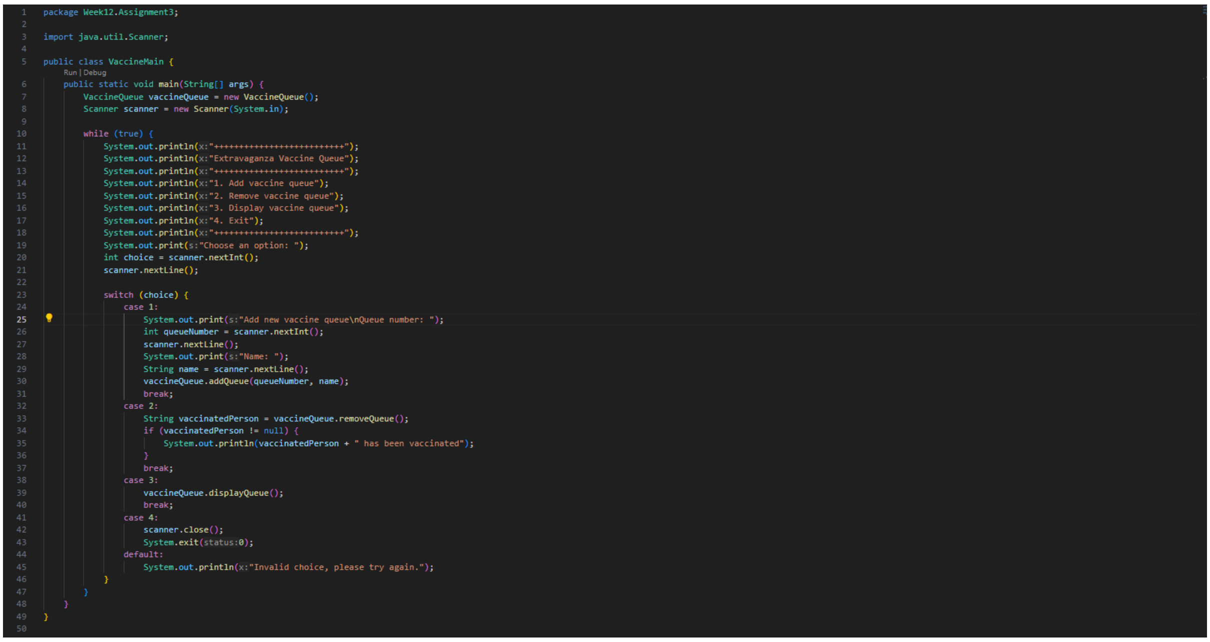
```
++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++
1. Add vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++
Choose an option: 2
Joko has been vaccinated
++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++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



```java
package Week12.Assignment3;

public class Node {
    int queueNumber;
    String name;
    Node next;
    Node prev;

    public Node(int queueNumber, String name) {
        this.queueNumber = queueNumber;
        this.name = name;
        this.next = null;
        this.prev = null;
    }
}
```



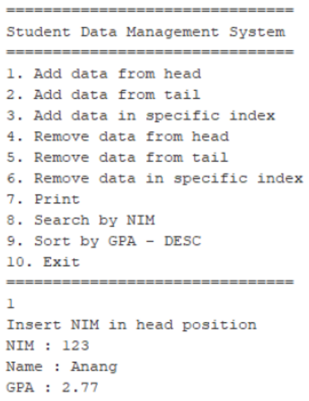
```java
public class VaccineQueue {
    private Node head;
    private Node tail;
    private int queueCount;

    public VaccineQueue() {
        this.head = null;
        this.tail = null;
        this.queueCount = 0;
    }

    public void addQueue(int queueNumber, String name) {
        Node newNode = new Node(queueNumber, name);
        if (tail == null) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
        queueCount++;
    }

    public String removeQueue() {
        if (head == null) {
            System.out.println(x:"Queue is empty");
            return null;
        }
        String name = head.name;
        head = head.next;
        if (head != null) {
            head.prev = null;
        } else {
            tail = null;
        }
        queueCount--;
        return name;
    }

    public void displayQueue() {
        if (head == null) {
            System.out.println(x:"Queue is empty");
            return;
        }
        Node temp = head;
        System.out.println(x:"Current vaccine queue:");
        System.out.println(x:"| No \t| Name \t|");
        while (temp != null) {
            System.out.println("| " + temp.queueNumber + " \t| " + temp.name + " \t|");
            temp = temp.next;
        }
        System.out.println("Queue left: " + queueCount);
```



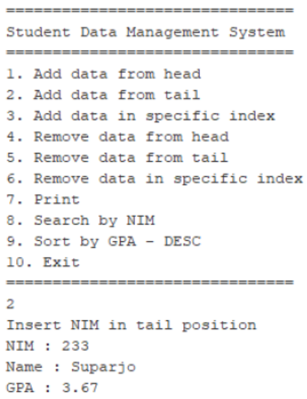
```java
package Week12.Assignment3;

import java.util.Scanner;

public class VaccineMain {
    Run | Debug
    public static void main(String[] args) {
        VaccineQueue vaccineQueue = new VaccineQueue();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println(x:"++++++++++++++++++++++++");
            System.out.println(x:"Extravaganza Vaccine Queue");
            System.out.println(x:"++++++++++++++++++++++++");
            System.out.println(x:"1. Add vaccine queue");
            System.out.println(x:"2. Remove vaccine queue");
            System.out.println(x:"3. Display vaccine queue");
            System.out.println(x:"4. Exit");
            System.out.println(x:"++++++++++++++++++++++++");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print(s:"Add new vaccine queue\nQueue number: ");
                    int queueNumber = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print(s:"Name: ");
                    String name = scanner.nextLine();
                    vaccineQueue.addQueue(queueNumber, name);
                    break;
                case 2:
                    String vaccinatedPerson = vaccineQueue.removeQueue();
                    if (vaccinatedPerson != null) {
                        System.out.println(vaccinatedPerson + " has been vaccinated");
                    }
                    break;
                case 3:
                    vaccineQueue.displayQueue();
                    break;
                case 4:
                    scanner.close();
                    System.exit(status:0);
                default:
                    System.out.println(x:"Invalid choice, please try again.");
            }
        }
    }
}
```

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**

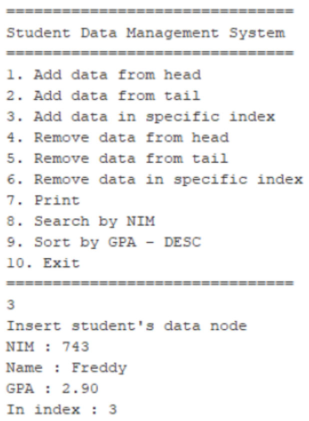
Initial menu and adding data



```
================================
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
```

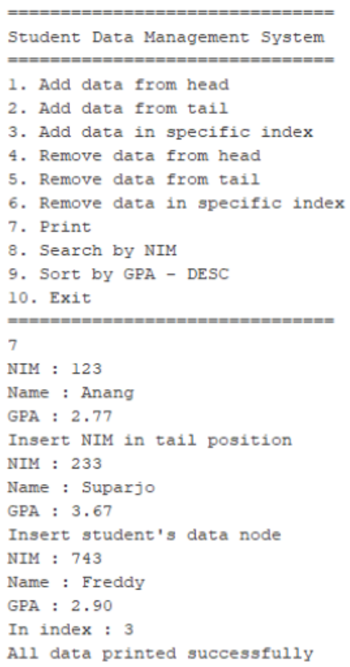


```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
2
Insert NIM in tail position
NIM : 233
Name : Suparjo
GPA : 3.67
```



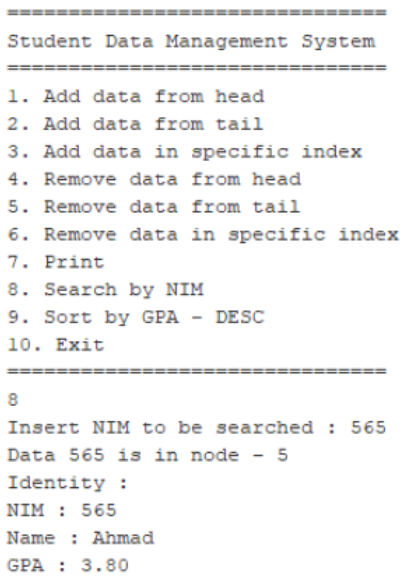
```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
3
Insert student's data node
NIM : 743
Name : Freddy
GPA : 2.90
In index : 3
```

Printing data



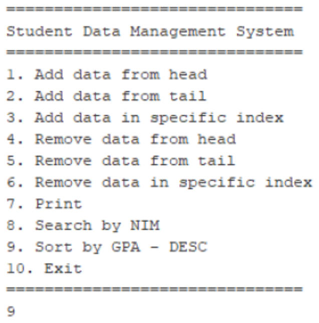
```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - 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

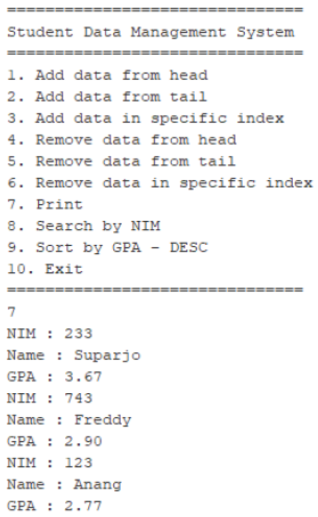


```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
8
Insert NIM to be searched : 565
Data 565 is in node - 5
Identity :
NIM : 565
Name : Ahmad
GPA : 3.80
```

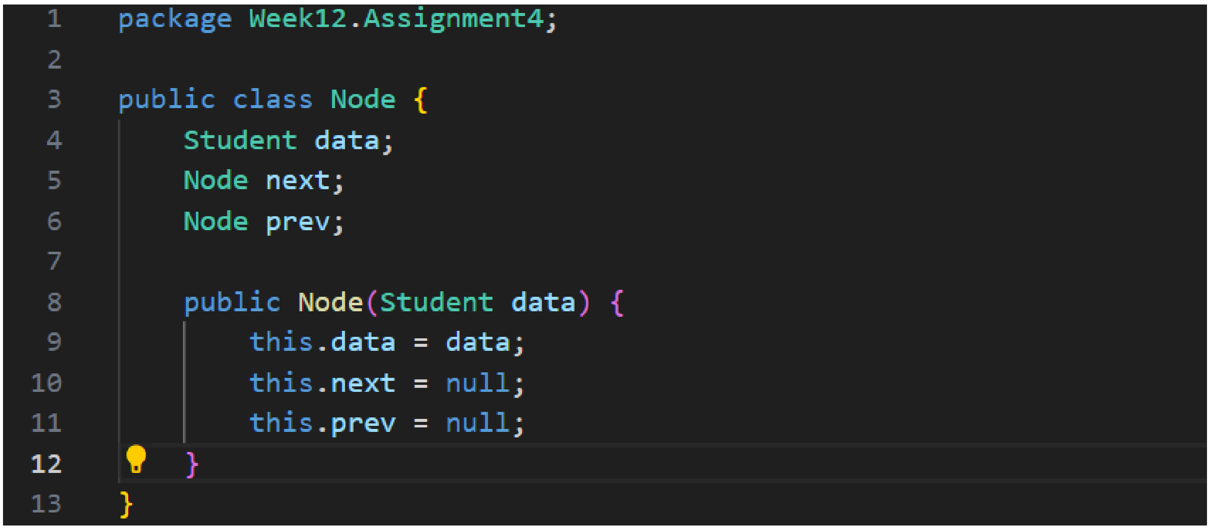
Sorting data



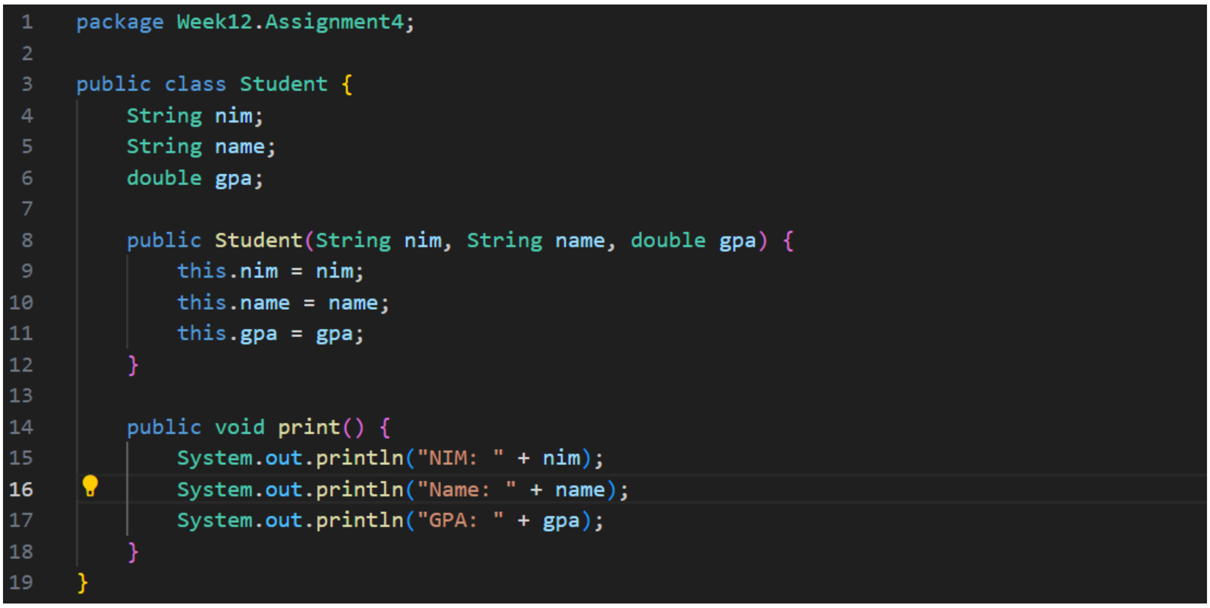
```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA -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



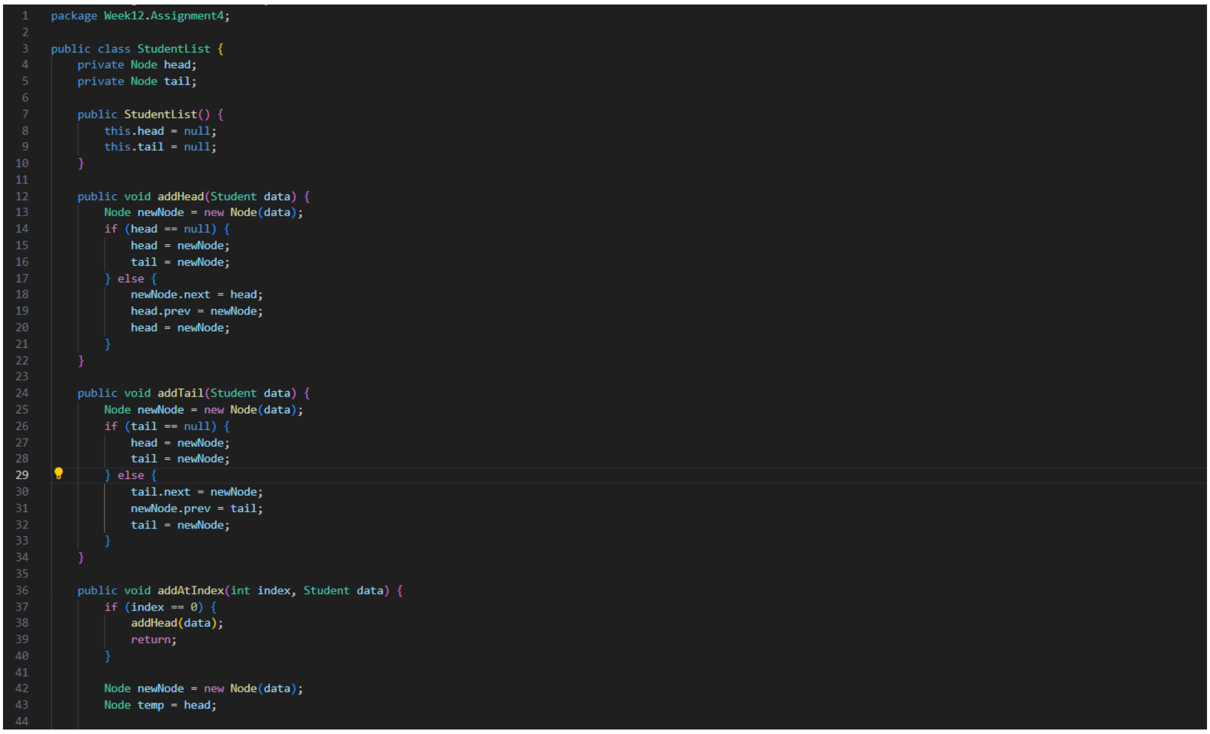
```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - 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
```



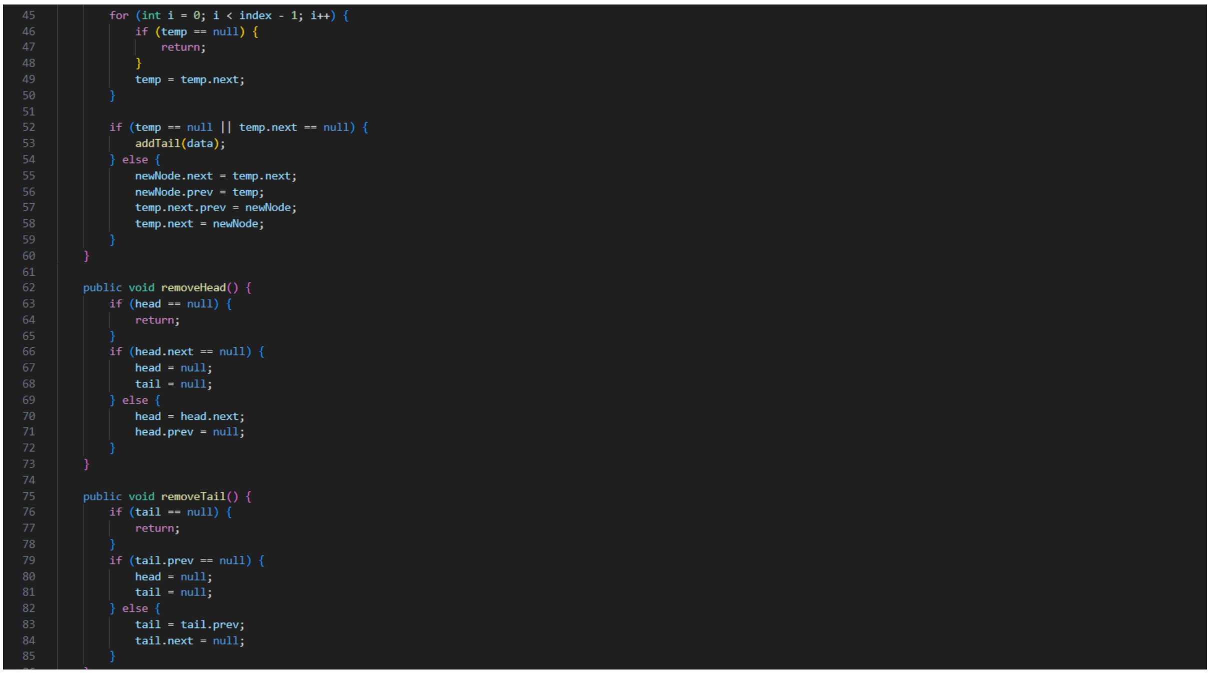
```java
package Week12.Assignment4;

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

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



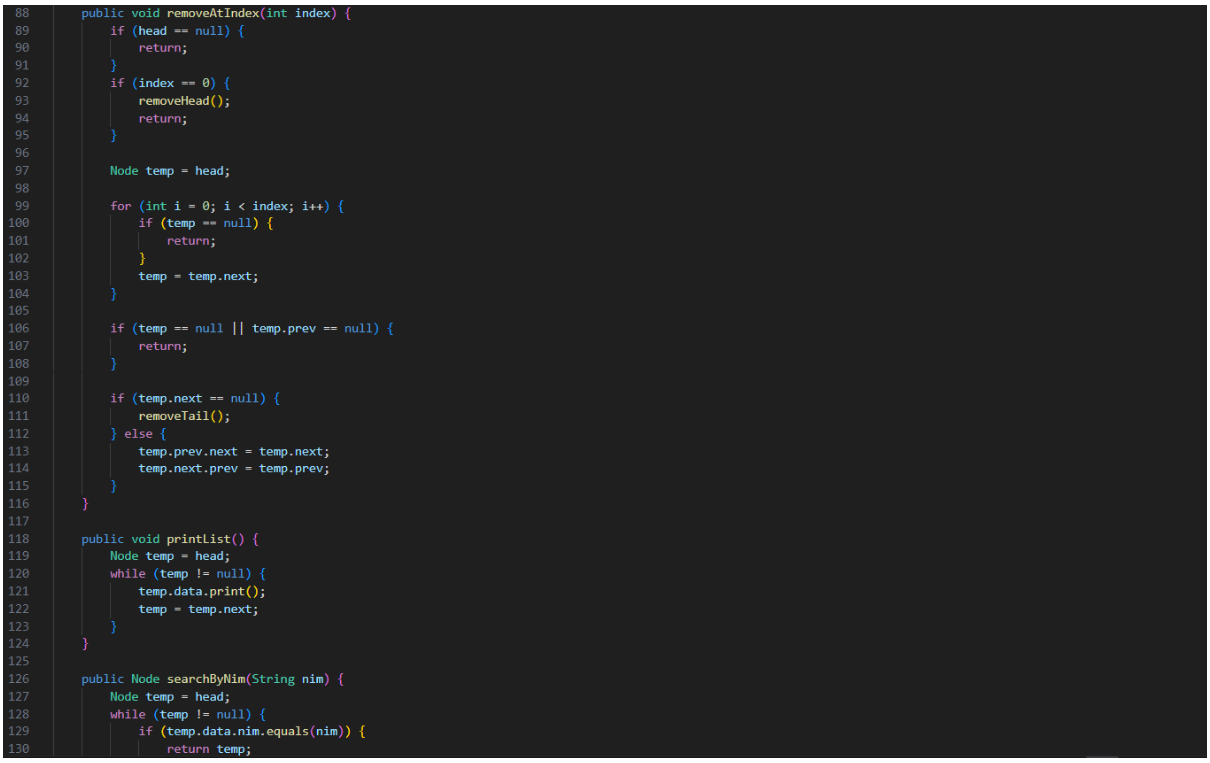
```java
package Week12.Assignment4;

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

    public Student(String nim, String name, double gpa) {
        this.nim = nim;
        this.name = name;
        this.gpa = gpa;
    }

    public void print() {
        System.out.println("NIM: " + nim);
        System.out.println("Name: " + name);
        System.out.println("GPA: " + gpa);
    }
}
```



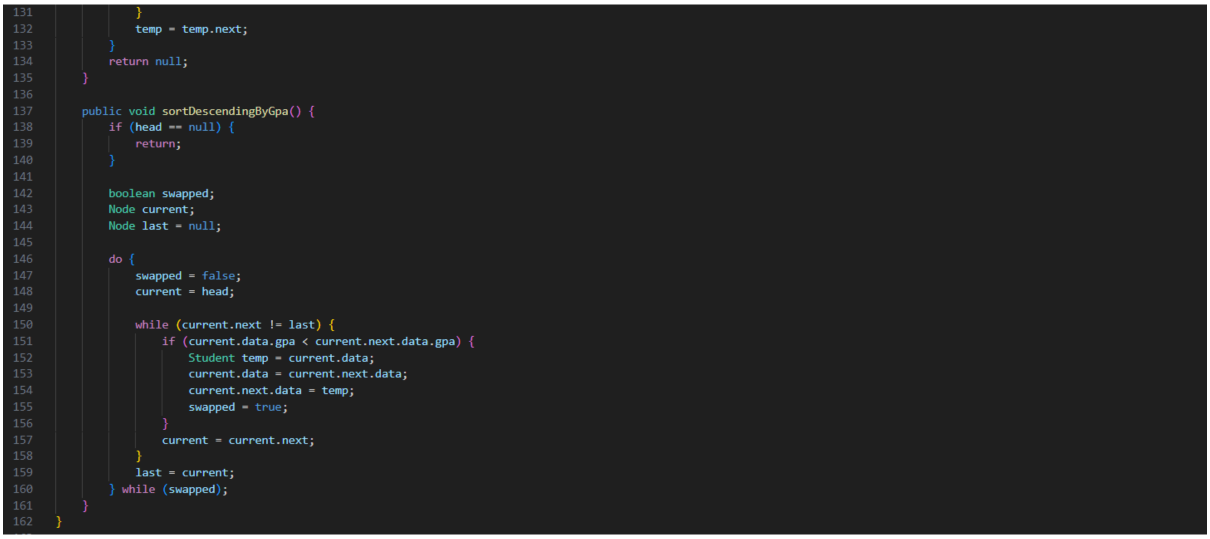
```java
package Week12.Assignment4;

public class StudentList {
    private Node head;
    private Node tail;

    public StudentList() {
        this.head = null;
        this.tail = null;
    }

    public void addHead(Student data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
    }

    public void addTail(Student data) {
        Node newNode = new Node(data);
        if (tail == null) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
    }

    public void addAtIndex(int index, Student data) {
        if (index == 0) {
            addHead(data);
            return;
        }

        Node newNode = new Node(data);
        Node temp = head;

```



```
        for (int i = 0; i < index - 1; i++) {
            if (temp == null) {
                return;
            }
            temp = temp.next;
        }

        if (temp == null || temp.next == null) {
            addTail(data);
        } else {
            newNode.next = temp.next;
            newNode.prev = temp;
            temp.next.prev = newNode;
            temp.next = newNode;
        }
    }

    public void removeHead() {
        if (head == null) {
            return;
        }
        if (head.next == null) {
            head = null;
            tail = null;
        } else {
            head = head.next;
            head.prev = null;
        }
    }

    public void removeTail() {
        if (tail == null) {
            return;
        }
        if (tail.prev == null) {
            head = null;
            tail = null;
        } else {
            tail = tail.prev;
            tail.next = null;
        }
```



```
    public void removeAtIndex(int index) {
        if (head == null) {
            return;
        }
        if (index == 0) {
            removeHead();
            return;
        }

        Node temp = head;

        for (int i = 0; i < index; i++) {
            if (temp == null) {
                return;
            }
            temp = temp.next;
        }

        if (temp == null || temp.prev == null) {
            return;
        }

        if (temp.next == null) {
            removeTail();
        } else {
            temp.prev.next = temp.next;
            temp.next.prev = temp.prev;
        }
    }

    public void printList() {
        Node temp = head;
        while (temp != null) {
            temp.data.print();
            temp = temp.next;
        }
    }

    public Node searchByNim(String nim) {
        Node temp = head;
        while (temp != null) {
            if (temp.data.nim.equals(nim)) {
                return temp;
```



```
            }
            temp = temp.next;
        }
        return null;
    }

    public void sortDescendingByGpa() {
        if (head == null) {
            return;
        }

        boolean swapped;
        Node current;
        Node last = null;

        do {
            swapped = false;
            current = head;

            while (current.next != last) {
                if (current.data.gpa < current.next.data.gpa) {
                    Student temp = current.data;
                    current.data = current.next.data;
                    current.next.data = temp;
                    swapped = true;
                }
                current = current.next;
            }
            last = current;
        } while (swapped);
    }
}
```