JOBSHEET 11

Linked List

Name
Sherly Lutfi Azkiah Sulistyawati

NIM
2341720241

Class
1I

Major
Information Technology

Study Program
D4 Informatics Engineering

Lab Activity 1

Practice > Week11 > Node.java > ...

```java
package Week11;

public class Node {
    int data;
    Node next;

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

```

Practice > Week11 > SingleLinkedList.java > SingleLinkedList > insertAt(int, int)

```java
package Week11;

public class SingleLinkedList {
    Node head; // Initial position in linked list
    Node tail; // Last position in linked list

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

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

    public void addFirst(int input) {
        Node ndInput = new Node(input, next:null);
        if (isEmpty()) { // if linked list is empty
            head = ndInput; // head and tail is equal with node input
            tail = ndInput;
        } else{
            ndInput.next = head;
            head = ndInput;
        }
    }

```

```java
    public void addLast(int input) {
        Node ndInput = new Node(input, next:null);
        if (isEmpty()) { // if linked list is empty
            head = ndInput; //head and tail is equal with node input
            tail = ndInput;
        } else{
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    public void insertAfter(int key, int input) {
        Node ndInput = new Node(input, next:null);
        Node temp = head;
        do {
            if (temp.data == key) {
                ndInput.next = temp.next;
                temp.next = ndInput;
                if (ndInput.next == null) tail = ndInput;
                break;
            }
            temp = temp.next;
        } while (temp != null);
    }

    public void insertAt(int index, int input) {
        if (index < 0) {
            System.out.println(x:"Wrong index");
        } else if (index == 0) {
            addFirst(input);
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1; i++) {
                temp = temp.next;
            }
            temp.next = new Node(input, temp.next);
            if (temp.next.next == null) tail = temp.next;
        }
    }
}
```

```java
public class SLLMain {
    Run | Debug
    public static void main(String[] args) {

        SingleLinkedList singLL = new SingleLinkedList();

        singLL.print();
        singLL.addFirst(input:890);
        singLL.print();
        singLL.addLast(input:760);
        singLL.print();
        singLL.addFirst(input:700);
        singLL.print();
        singLL.insertAfter(key:700, input:999);
        singLL.print();
        singLL.insertAt(index:3, input:833);
        singLL.print();
```

```
Linked list is empty
Linked list content:    890
Linked list content:    890     760
Linked list content:    700     890     760
Linked list content:    700     999     890     760
Linked list content:    700     999     890     833     760
PS D:\College\Semester 2\AlgoritmadanStrukturData>
```

Question

1. Why the output of the program in first line is “Linked list is empty”?

   - The output is "Linked list is empty" because the linked list is empty when the print() method is first called. No elements have been added to the list at that point.

2. Please explain the usage of these following codes in:

```
ndInput.next = temp.next;
temp.next = ndInput;
```

   - These lines of code are used in the insertAfter method to insert a new node (ndInput) after a specified node (temp) with a certain key value in the linked list.

   - ndInput.next = temp.next;: Links the new node (ndInput) to the node that follows temp.

   - temp.next = ndInput;: Links the temp node to the new node (ndInput), inserting ndInput after temp.

3. In SingleLinkedList, what is the usage of this following code in insertAt?

```
if(temp.next.next==null) tail=temp.next;
```

   - This line of code is used to update the tail pointer if the new node is inserted at the last position of the linked list. (i.e., the new node's next is null).

Lab Activity 2

```
76      public int getData(int index) {
77          Node tmp = head;
78          for (int i = 0; i < index; i++) {
79              tmp = tmp.next;
80          }
81          return tmp.data;
82      }
83
84      public int indexOf(int key) {
85          Node tmp = head;
86          int index = 0;
87          while (tmp != null && tmp.data != key) {
88              tmp = tmp.next;
89              index++;
90          }
91
92          if (tmp == null) {
93              return -1;
94          } else{
95              return index;
96          }
97      }
98
99      public void removeFirst() {
100         if (isEmpty()) {
101             System.out.println(x:"Linked list is empty. Cannot remove a data");
102         } else if(head == tail) {
103             head = tail = null;
104         } else{
105             head = head.next;
106         }
107     }
```

```
109     public void removeLast() {
110         if (isEmpty()) {
111             System.out.println(x:"Linked list is empty. Cannot remove a data");
112         } else if(head == tail) {
113             head = tail = null;
114         } else{
115             Node temp = head;
116             while (temp.next != tail) {
117                 temp = temp.next;
118             }
119             temp.next = null;
120             tail = temp;
121         }
122     }
123 
124     public void remove(int key) {
125         if (isEmpty()) {
126             System.out.println(x:"Linked list is empty. Cannot remove a data");
127         } else{
128             Node temp = head;
129             while (temp != null) {
130                 if ((temp.data == key) && (temp == head)) {
131                     this.removeFirst();
132                     break;
133                 } else if(temp.next.data == key) {
134                     temp.next = temp.next.next;
135                     if (temp.next == null) {
136                         tail = temp;
137                     }
138                     break;
139                 }
140                 temp = temp.next;
141             }
142         }
143     }
```

```
145     public void removeAt(int index) {
146         if (index == 0) {
147             removeFirst();
148         } else{
149             Node temp = head;
150             for (int i = 0; i < index - 1; i++) {
151                 temp = temp.next;
152             }
153             temp.next = temp.next.next;
154             if (temp.next == null) {
155                 tail = temp;
156             }
157         }
158     }
159 }
160 
```

```
20          System.out.println("Data in 1st index: " + singLL.getData(index:1));
21          System.out.println("Data 3 is in index: " + singLL.indexOf(key:760));
22          singLL.remove(key:999);
23          singLL.print();
24          singLL.removeAt(index:0);
25          singLL.print();
26          singLL.removeFirst();
27          singLL.print();
28          singLL.removeLast();
29          singLL.print();
30      }
31  }
32 
```

```
Linked list is empty
Linked list content:    890
Linked list content:    890     760
Linked list content:    700     890     760
Linked list content:    700     999     890     760
Linked list content:    700     999     890     833     760
Data in 1st index: 999
Data 3 is in index: 4
Linked list content:    700     890     833     760
Linked list content:    890     833     760
Linked list content:    833     760
Linked list content:    833
PS D:\College\Semester 2\AlgoritmadanStrukturData>
```

Question

1. Why we use break keyword in remove function? Please explain

   - The break keyword is used to exit the while loop immediately after the node with the specified key is found and removed, preventing further unnecessary iterations.

2. Please explain why we implement these following codes in method remove

```
else if (temp.next.data == key) {
   temp.next = temp.next.next;
```

   - This code handles the case where the node to be removed is not the head but is somewhere else in the list.

   - temp.next.data == key: Checks if the data of the node immediately following temp is equal to the key.

   - temp.next = temp.next.next;: Bypasses the node to be removed by setting the next pointer of temp to point to the node after the node to be removed. This effectively removes the node with the key from the list.

3. What are the outputs of method indexOf? Please explain each of the output!

   - The indexOf method returns the index of the first occurrence of a node with the specified key. If the key is not found, it returns -1.

   - Example outputs:

   If the list is [700, 890, 760] and we call indexOf(890), it will return 1 because 890 is at index 1.

   If the list is [700, 890, 760] and we call indexOf(760), it will return 2 because 760 is at index 2.

If the list is [700, 890, 760] and we call indexOf(999), it will return -1 because 999 is not in the list.

Assignment

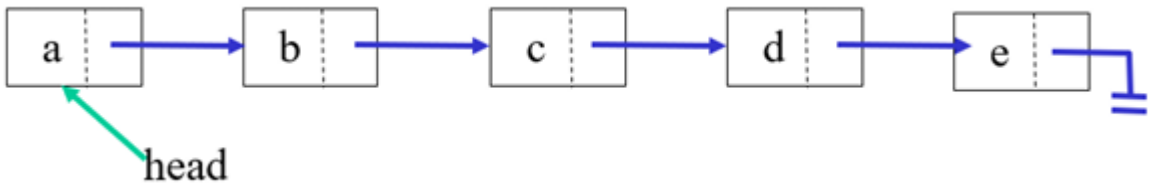
1. Create a method **insertBefore()** to add node before the desired keyword

```
public void insertBefore(int key, int input) {
    if (isEmpty()) {
        System.out.println("Linked list is empty. Cannot insert before " + key);
        return;
    }

    if (head.data == key) {
        addFirst(input);
        return;
    }

    Node ndInput = new Node(input, next:null);
    Node temp = head;
    while (temp.next != null && temp.next.data != key) {
        temp = temp.next;
    }

    if (temp.next == null) {
        System.out.println("Key " + key + " not found in the list.");
    } else {
        ndInput.next = temp.next;
        temp.next = ndInput;
    }
}
```

```
singLL.insertBefore(key:833, input:444);
singLL.print();
singLL.insertBefore(key:1000, input:333); // Key not in list
singLL.print();
```

```
Linked list is empty
Linked list content:    890
Linked list content:    890     760
Linked list content:    700     890     760
Linked list content:    700     999     890     760
Linked list content:    700     999     890     833     760
Data in 1st index: 999
Data 3 is in index: 4
Linked list content:    700     890     833     760
Linked list content:    890     833     760
Linked list content:    833     760
Linked list content:    833
Linked list content:    444     833
Key 1000 not found in the list.
Linked list content:    444     833
```

2. Implement the linked list from this following image. You may use 4 method of adding data we've learnt



Practice > Week11 > Assignment2 > Node.java > Node

```java
package Week11.Assignment2;

public class Node {
    String data;
    Node next;

    public Node(String data, Node next) {
        this.data = data;
        this.next = next;
    }
}

```

Practice > Week11 > Assignment2 > SingleLinkedList.java > ...

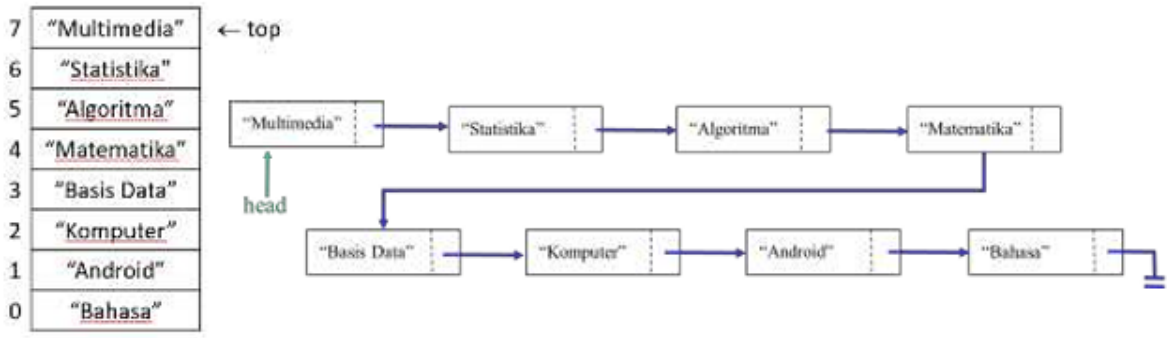
```java
package Week11.Assignment2;

public class SingleLinkedList {
    Node head; // Initial position in linked list
    Node tail; // Last position in linked list

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

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

    public void addFirst(String input) {
        Node ndInput = new Node(input, next:null);
        if (isEmpty()) { // if linked list is empty
            head = ndInput; // head and tail is equal with node input
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    public void addLast(String input) {
        Node ndInput = new Node(input, next:null);
        if (isEmpty()) { // if linked list is empty
            head = ndInput; // head and tail is equal with node input
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    public void insertAfter(String key, String input) {
        Node ndInput = new Node(input, next:null);
        Node temp = head;
        while (temp != null) {
            if (temp.data.equals(key)) {
                ndInput.next = temp.next;
                temp.next = ndInput;
                if (ndInput.next == null) tail = ndInput;
```

```
                break;
            }
            temp = temp.next;
        }
    }

    public void insertAt(int index, String input) {
        if (index < 0) {
            System.out.println(x:"Wrong index");
        } else if (index == 0) {
            addFirst(input);
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1; i++) {
                temp = temp.next;
            }
            temp.next = new Node(input, temp.next);
            if (temp.next.next == null) tail = temp.next;
        }
    }

    public String getData(int index) {
        Node tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.data;
    }

    public int indexOf(String key) {
        Node tmp = head;
        int index = 0;
        while (tmp != null && !tmp.data.equals(key)) {
            tmp = tmp.next;
            index++;
        }

        if (tmp == null) {
            return -1;
        } else {
            return index;
        }
    }

    public void removeFirst() {
        if (isEmpty()) {
            System.out.println(x:"Linked list is empty. Cannot remove data");
        } else if (head == tail) {
            head = tail = null;
        } else {
            head = head.next;
```

```
    public void removeLast() {
        if (isEmpty()) {
            System.out.println(x:"Linked list is empty. Cannot remove data");
        } else if (head == tail) {
            head = tail = null;
        } else {
            Node temp = head;
            while (temp.next != tail) {
                temp = temp.next;
            }
            temp.next = null;
            tail = temp;
        }
    }

    public void remove(String key) {
        if (isEmpty()) {
            System.out.println(x:"Linked list is empty. Cannot remove data");
        } else {
            Node temp = head;
            while (temp != null) {
                if ((temp.data.equals(key)) && (temp == head)) {
                    this.removeFirst();
                    break;
                } else if (temp.next.data.equals(key)) {
                    temp.next = temp.next.next;
                    if (temp.next == null) {
                        tail = temp;
                    }
                    break;
                }
                temp = temp.next;
            }
        }
    }

    public void removeAt(int index) {
        if (index == 0) {
            removeFirst();
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1; i++) {
                temp = temp.next;
            }
            temp.next = temp.next.next;
            if (temp.next == null) {
                tail = temp;
            }
        }
    }
}
```

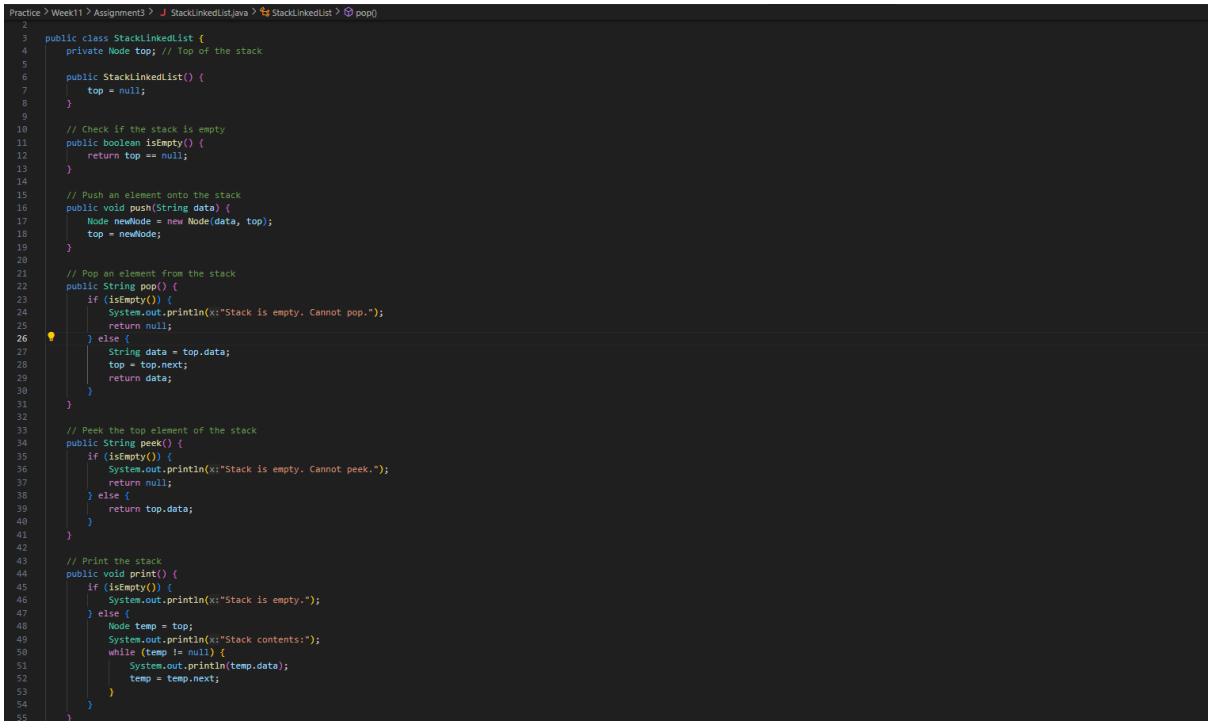
```
Linked list content:    a       b       c       d       e
Linked list content:    a       b       g       h       c       f
Linked list content:    b       g       h       c       d       f
Linked list content:    b       g       h       c       d       e
Linked list content:    b       h       c       d       e
Linked list content:    b       h       d       e
```

3. Create this following **Stack** implementation using Linked List implementation

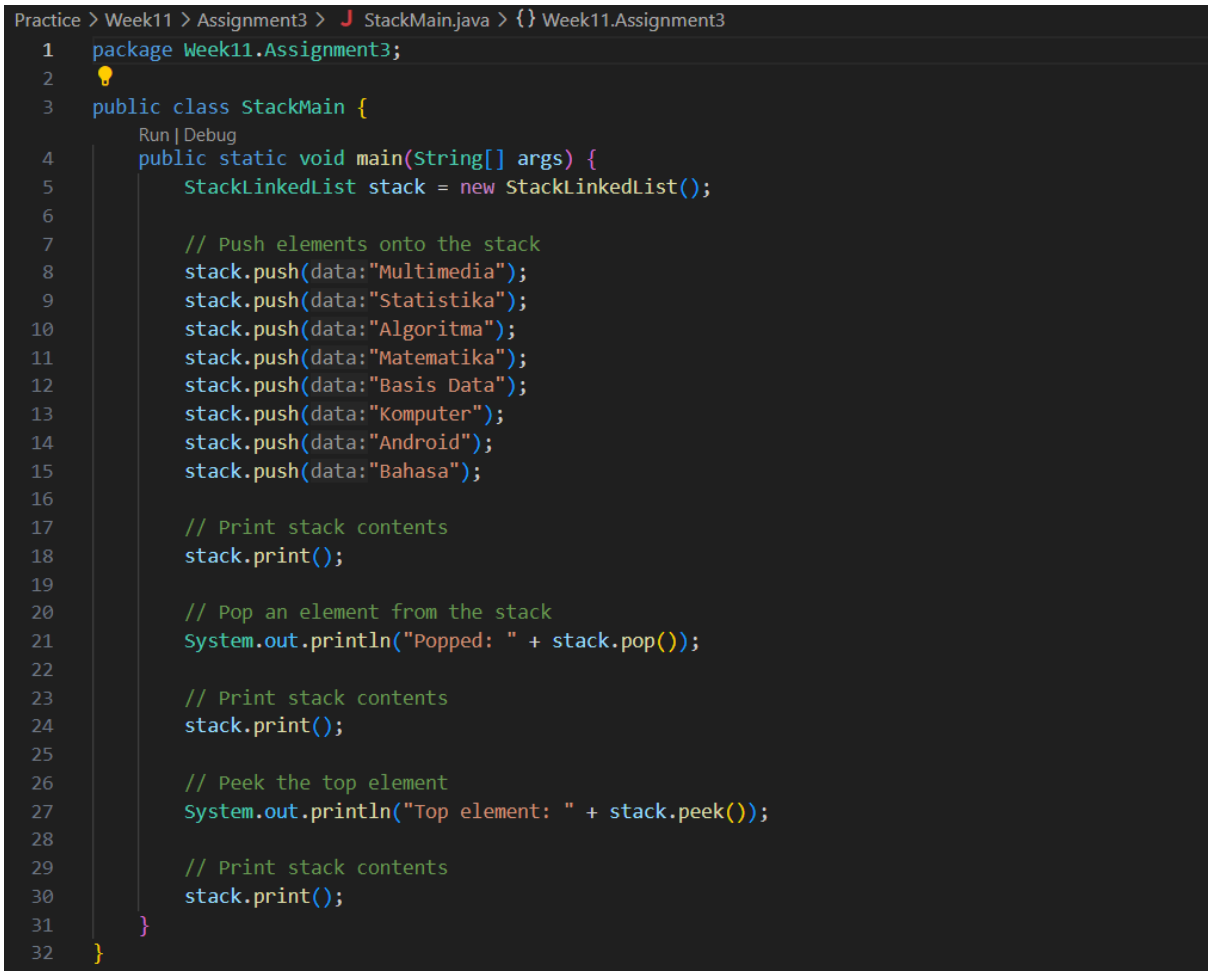


Practice > Week11 > Assignment3 > J Node.java > ...

```java
package Week11.Assignment3;

public class Node {
    String data;
    Node next;

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



```
Practice > Week11 > Assignment3 > StackLinkedList.java > StackLinkedList > pop()

public class StackLinkedList {
    private Node top; // Top of the stack

    public StackLinkedList() {
        top = null;
    }

    // Check if the stack is empty
    public boolean isEmpty() {
        return top == null;
    }

    // Push an element onto the stack
    public void push(String data) {
        Node newNode = new Node(data, top);
        top = newNode;
    }

    // Pop an element from the stack
    public String pop() {
        if (isEmpty()) {
            System.out.println(x:"Stack is empty. Cannot pop.");
            return null;
        } else {
            String data = top.data;
            top = top.next;
            return data;
        }
    }

    // Peek the top element of the stack
    public String peek() {
        if (isEmpty()) {
            System.out.println(x:"Stack is empty. Cannot peek.");
            return null;
        } else {
            return top.data;
        }
    }

    // Print the stack
    public void print() {
        if (isEmpty()) {
            System.out.println(x:"Stack is empty.");
        } else {
            Node temp = top;
            System.out.println(x:"Stack contents:");
            while (temp != null) {
                System.out.println(temp.data);
                temp = temp.next;
            }
        }
    }
```



```
Practice > Week11 > Assignment3 > StackMain.java > {} Week11.Assignment3

package Week11.Assignment3;

public class StackMain {
    Run | Debug
    public static void main(String[] args) {
        StackLinkedList stack = new StackLinkedList();

        // Push elements onto the stack
        stack.push(data:"Multimedia");
        stack.push(data:"Statistika");
        stack.push(data:"Algoritma");
        stack.push(data:"Matematika");
        stack.push(data:"Basis Data");
        stack.push(data:"Komputer");
        stack.push(data:"Android");
        stack.push(data:"Bahasa");

        // Print stack contents
        stack.print();

        // Pop an element from the stack
        System.out.println("Popped: " + stack.pop());

        // Print stack contents
        stack.print();

        // Peek the top element
        System.out.println("Top element: " + stack.peek());

        // Print stack contents
        stack.print();
    }
}
```

```
Stack contents:
Bahasa
Android
Komputer
Basis Data
Matematika
Algoritma
Statistika
Multimedia
Popped: Bahasa
Stack contents:
Android
Komputer
Basis Data
Matematika
Algoritma
Statistika
Multimedia
Top element: Android
Stack contents:
Android
Komputer
Basis Data
Matematika
Algoritma
Statistika
Multimedia
```

4. Create a program that helps bank customer using linked list with data are as follows: Name,address, and customerAccountNumber

Practice > Week11 > Assignment4 > Node.java > { } Week11.Assignment4

```java
package Week11.Assignment4;

public class Node {
    String name;
    String address;
    int customerAccountNumber;
    Node next;

    public Node(String name, String address, int customerAccountNumber, Node next) {
        this.name = name;
        this.address = address;
        this.customerAccountNumber = customerAccountNumber;
        this.next = next;
    }
}
```

Practice > Week11 > Assignment4 > CustomerLinkedList.java > Week11.Assignment4

```java
package Week11.Assignment4;

public class CustomerLinkedList {
    Node head; // Initial position in linked list
    Node tail; // Last position in linked list

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

    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            System.out.println(x:"Customer list:");
            while (tmp != null) {
                System.out.println("Name: " + tmp.name + ", Address: " + tmp.address + ", Account Number: " + tmp.customerAccountNumber);
                tmp = tmp.next;
            }
        } else {
            System.out.println(x:"Customer list is empty");
        }
    }

    public void addFirst(String name, String address, int customerAccountNumber) {
        Node newNode = new Node(name, address, customerAccountNumber, head);
        if (isEmpty()) {
            head = newNode;
            tail = newNode;
        } else {
            newNode.next = head;
            head = newNode;
        }
    }

    public void addLast(String name, String address, int customerAccountNumber) {
        Node newNode = new Node(name, address, customerAccountNumber, next:null);
        if (isEmpty()) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
    }

    public void insertAfter(int customerAccountNumber, String name, String address, int newCustomerAccountNumber) {
        Node newNode = new Node(name, address, newCustomerAccountNumber, next:null);
        Node temp = head;
        while (temp != null) {
            if (temp.customerAccountNumber == customerAccountNumber) {
                newNode.next = temp.next;
                temp.next = newNode;
                if (newNode.next == null) {
                    tail = newNode;
                }
                break;
            }
            temp = temp.next;
        }
    }

    public void remove(int customerAccountNumber) {
        if (isEmpty()) {
            System.out.println(x:"Customer list is empty. Cannot remove data.");
        } else {
            if (head.customerAccountNumber == customerAccountNumber) {
                head = head.next;
                if (head == null) {
                    tail = null;
                }
            } else {
                Node temp = head;
                while (temp.next != null && temp.next.customerAccountNumber != customerAccountNumber) {
                    temp = temp.next;
                }
                if (temp.next != null) {
                    temp.next = temp.next.next;
                    if (temp.next == null) {
                        tail = temp;
                    }
                }
            }
        }
    }
}
```

Practice > Week11 > Assignment4 > J CustomerMain.java > ...

```java
package Week11.Assignment4;

public class CustomerMain {
    Run | Debug
    public static void main(String[] args) {
        CustomerLinkedList customerList = new CustomerLinkedList();

        // Adding customers
        customerList.addFirst(name:"Alice", address:"123 Main St", customerAccountNumber:1001);
        customerList.addLast(name:"Bob", address:"456 Elm St", customerAccountNumber:1002);
        customerList.addLast(name:"Charlie", address:"789 Oak St", customerAccountNumber:1003);
        customerList.print();

        // Insert a customer after a specific account number
        customerList.insertAfter(customerAccountNumber:1002, name:"Diana", address:"101 Pine St", newCusto…1004);
        customerList.print();

        // Remove a customer by account number
        customerList.remove(customerAccountNumber:1001);
        customerList.print();

        customerList.remove(customerAccountNumber:1003);
        customerList.print();
    }
}
```

```
Customer list:
Name: Alice, Address: 123 Main St, Account Number: 1001
Name: Bob, Address: 456 Elm St, Account Number: 1002
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Customer list:
Name: Alice, Address: 123 Main St, Account Number: 1001
Name: Bob, Address: 456 Elm St, Account Number: 1002
Name: Diana, Address: 101 Pine St, Account Number: 1004
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Customer list:
Name: Bob, Address: 456 Elm St, Account Number: 1002
Name: Diana, Address: 101 Pine St, Account Number: 1004
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Customer list:
Name: Bob, Address: 456 Elm St, Account Number: 1002
Name: Diana, Address: 101 Pine St, Account Number: 1004
```

5. Implement **Queue** in previous number with **linked list** concept

```
Customer queue:
Name: Alice, Address: 123 Main St, Account Number: 1001
Name: Bob, Address: 456 Elm St, Account Number: 1002
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Customer queue:
Name: Bob, Address: 456 Elm St, Account Number: 1002
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Customer queue:
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Customer queue:
Name: Charlie, Address: 789 Oak St, Account Number: 1003
Name: Diana, Address: 101 Pine St, Account Number: 1004
```

Practice > Week11 > Assignment4 > CustomerQueue.java > ...

```java
public class CustomerQueue {
    Node head; // Front of the queue
    Node tail; // End of the queue

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

    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            System.out.println(x:"Customer queue:");
            while (tmp != null) {
                System.out.println("Name: " + tmp.name + ", Address: " + tmp.address + ", Account Number: " + tmp.customerAccountNumber);
                tmp = tmp.next;
            }
        } else {
            System.out.println(x:"Customer queue is empty");
        }
    }

    public void enqueue(String name, String address, int customerAccountNumber) {
        Node newNode = new Node(name, address, customerAccountNumber, next:null);
        if (isEmpty()) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
    }

    public void dequeue() {
        if (isEmpty()) {
            System.out.println(x:"Customer queue is empty. Cannot dequeue.");
        } else {
            head = head.next;
            if (head == null) {
                tail = null;
            }
        }
    }
}
```

Practice > Week11 > Assignment4 > CustomerQueueLLMain.java > CustomerQueueLLMain

```java
package Week11.Assignment4;

public class CustomerQueueLLMain {
    Run | Debug
    public static void main(String[] args) {
        CustomerQueue customerQueue = new CustomerQueue();

        // Enqueue customers
        customerQueue.enqueue("Aname:lice", "1address:23 Main St", 10customerAccountNumber:01);
        customerQueue.enqueue("Bname:ob", "4address:56 Elm St", 10customerAccountNumber:02);
        customerQueue.enqueue("Cname:harlie", "7address:89 Oak St", 10customerAccountNumber:03);
        customerQueue.print();

        // Dequeue a customer
        customerQueue.dequeue();
        customerQueue.print();

        // Dequeue another customer
        customerQueue.dequeue();
        customerQueue.print();

        // Enqueue another customer
        customerQueue.enqueue("Dname:iana", "1address:01 Pine St", 10customerAccountNumber:04);
        customerQueue.print();
    }
}
```