Job Sheet 11 Linked List

From:

AL AZHAR RIZQI RIFA'I FIRDAUS

Class:

1 I

Absence:

01

Student Number Identity:

2241720263

Department:

Information Technology

Study Program:

Informatics Engineering

Practicum 1

Code :

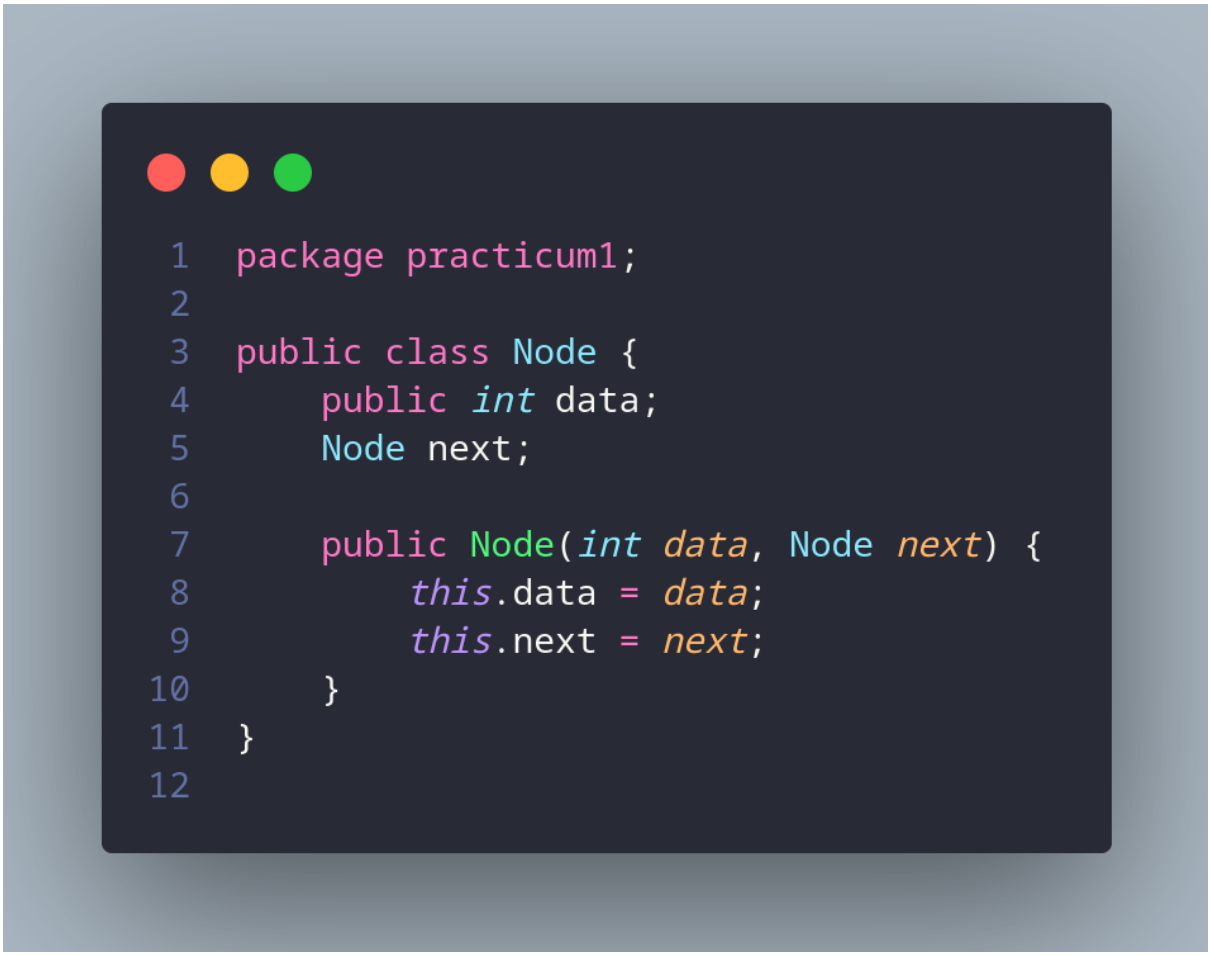
- Main

```java
package practicum1;

public class SSLMain {
    public static void main(String[] args) {
        SingleLinkedList singleLinkedList = new SingleLinkedList();

        singleLinkedList.print();
        singleLinkedList.addFirst(890);
        singleLinkedList.print();
        singleLinkedList.addLast(760);
        singleLinkedList.print();
        singleLinkedList.addFirst(700);
        singleLinkedList.print();
        singleLinkedList.insertAfter(700, 999);
        singleLinkedList.print();
        singleLinkedList.inserAt(3, 833);
        singleLinkedList.print();
    }
}

```

- Single Linked List

```java
package practicum1;

public class SingleLinkedList {
    Node head;
    Node tail;

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

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

    public void addFirst(int input) {
        Node ndInput = new Node(input, null);
        if (isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    public void addLast(int input) {
        Node ndInput = new Node(input, null);
        if (isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

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

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


}

```

- Node



```java
package practicum1;

public class Node {
    public int data;
    Node next;

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

Result :

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessages
/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum/bin practicum1.SSLMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
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
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$
```

Question

1. Why the output of the program in first line is “Linked list is empty”?
    - **Because when a linked list is printed, there is no data that has been created. That’s why the first line of the result is “Linked list is empty”.**

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

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

- **The lines ndInput.next = tmp.next; and tmp.next = ndInput; are used to insert a new node (ndInput) after a specific node (tmp) in the linked list. By using these lines, the code effectively inserts the new node ndInput after the node referenced by tmp in the linked list.**

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

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

- **The line if (tmp.next.next == null) tail = tmp.next; in the insertAt method of the SingleLinkedList class is used to update the tail reference of the linked list if the new node is inserted at the end of the list. By using this code, the tail reference of the linked list is updated whenever a new node is inserted at the end, allowing efficient access to the last node in the list without traversing the entire list.**

Practicum 2

Code :

- SIngle Linked List

```java
// add practicum 2

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

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

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

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

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

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

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

- Main

```java
// add practicum 2
System.out.println("Data in 1st index : " + singleLinkedList.getData(1));
System.out.println("Data 3 is in index : " + singleLinkedList.indexOf(760));
singleLinkedList.remove(999);
singleLinkedList.print();
singleLinkedList.removeAt(0);
singleLinkedList.print();
singleLinkedList.removeFirst();
singleLinkedList.print();
singleLinkedList.removeLast();
singleLinkedList.print();
```

- Result

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessages
/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum/bin practicum1.SSLMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
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     999     833     760
Linked list content :   999     833     760
Linked list content :   833     760
Linked list content :   833
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$
```

Question

1. Why we use break keyword in remove function? Please explain
    - **The break keyword is used in the remove function to exit the loop once the target node is found and removed from the linked list. By using the break statement, the loop is terminated early once the removal operation is completed, improving the efficiency of the algorithm by avoiding unnecessary iterations.**

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

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

- **The lines else if (tmp.data == key) { tmp.next = tmp.next.next; } in the remove method are used to remove a node from the linked list when the node with a matching value (key) is found.**

3. What are the outputs of method indexOf? Please explain each of the output!
    - **The indexOf method is used to find the index of the first occurrence of a node with a specific value in the linked list. The method returns an integer value indicating the index position. Let's explain each of the possible outputs:**

      If the target value is found in the linked list:
      Output: The index position of the first occurrence of the target value in the linked list.
      Example: If the target value is found at index 2, the output will be 2.

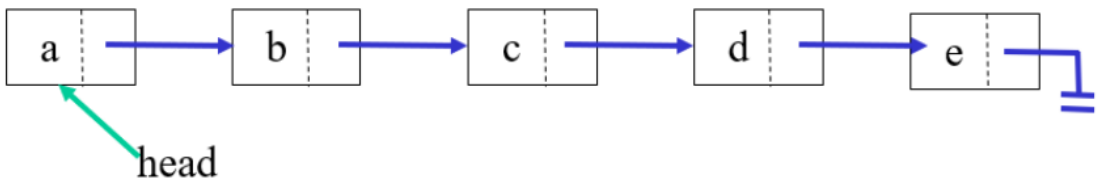
      If the target value is not found in the linked list:
      Output: -1
      Explanation: If the target value is not present in the linked list, the method returns -1 to indicate that the value was not found.

Assignment

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

```java
public void insertBefore(int key, int input) {
    Node ndInput = new Node(input, null);

    // If the list is empty, there is no node to insert before
    if (isEmpty()) {
        System.out.println("Linked list is empty");
        return;
    }

    // If the key is found at the head node, add the new node at the beginning
    if (head.data == key) {
        ndInput.next = head;
        head = ndInput;
        return;
    }

    // Traverse the linked list to find the key and insert the new node before it
    Node tmp = head;
    while (tmp.next != null) {
        if (tmp.next.data == key) {
            ndInput.next = tmp.next;
            tmp.next = ndInput;
            return;
        }
        tmp = tmp.next;
    }

    // If the key is not found in the linked list
    System.out.println("Key not found in the linked list");
}
```

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



- Node

```java
package Assnum2;

public class Node {
    public char data;
    Node next;

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

```

- Single Linked List

```java
package Assnum2;

public class SingleLinkedList {
    Node head;
    Node tail;


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

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

    public void addFirst(char input) {
        Node ndInput = new Node(input, null);
        if (isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    public void addLast(char input) {
        Node ndInput = new Node(input, null);
        if (isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

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

```java
public void insertBefore(char key, char input) {
    Node ndInput = new Node(input, null);

    // If the list is empty, there is no node to insert before
    if (isEmpty()) {
        System.out.println("Linked list is empty");
        return;
    }

    // If the key is found at the head node, add the new node at the beginning
    if (head.data == key) {
        ndInput.next = head;
        head = ndInput;
        return;
    }

    // Traverse the linked list to find the key and insert the new node before it
    Node tmp = head;
    while (tmp.next != null) {
        if (tmp.next.data == key) {
            ndInput.next = tmp.next;
            tmp.next = ndInput;
            return;
        }
        tmp = tmp.next;
    }

    // If the key is not found in the linked list
    System.out.println("Key not found in the linked list");
}


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

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

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

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

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

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

```java
    public void remove(char key) {
        if (isEmpty()) {
            System.out.println("Linked list is empty. Cannot remove a data");
        } else {
            Node tmp = head;
            while (tmp != null) {
                if ((tmp.data == key) && (tmp == head)) {
                    this.removeFirst();
                    break;
                } else if (tmp.data == key) {
                    tmp.next = tmp.next.next;
                    if (tmp.next == null) {
                        tail = tmp;
                    }
                    break;
                }
                tmp = tmp.next;
            }
        }
    }

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

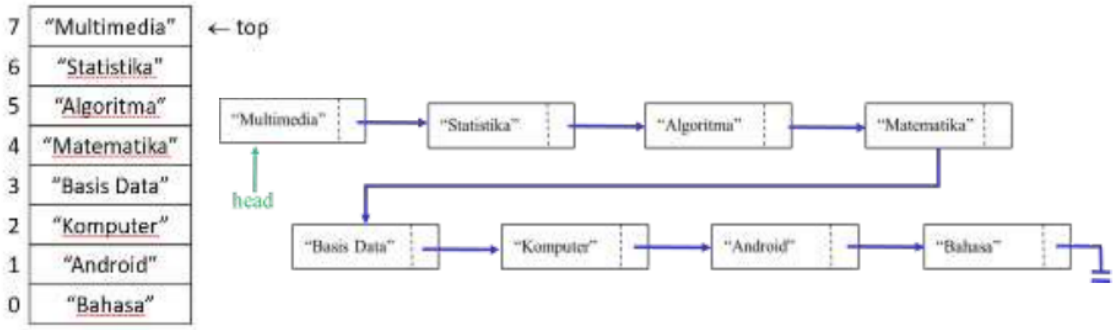
- Main

```java
package Assnum2;

public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singleLinkedList = new SingleLinkedList();

        singleLinkedList.print();
        singleLinkedList.addFirst('a');
        singleLinkedList.print();
        singleLinkedList.addLast('b');
        singleLinkedList.print();
        singleLinkedList.insertAfter('b', 'd');
        singleLinkedList.print();
        singleLinkedList.insertBefore('d', 'c');
        singleLinkedList.print();
        singleLinkedList.insertAt(4, 'e');
        singleLinkedList.print();
    }
}

```

- Result

```
(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessage
s/College/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum/bin Assnum2.SLL
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Linked list is empty
Linked list content :   a
Linked list content :   a       b
Linked list content :   a       b       d
Linked list content :   a       b       c       d
Linked list content :   a       b       c       d       e

(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$
```

3. Create this following Stack implementation using Linked List implementation



- Node

```java
package Assnum3;

public class Node {
    public String data;
    Node next;

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

```

- Single Linked List

```java
package Assnum3;

public class SingleLinkedList {
    Node head;
    Node tail;


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

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

    public void addFirst(String input) {
        Node ndInput = new Node(input, null);
        if (isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head = ndInput;
        }
    }

    public void addLast(String input) {
        Node ndInput = new Node(input, null);
        if (isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

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

```java
public void insertBefore(String key, String input) {
    Node ndInput = new Node(input, null);

    // If the list is empty, there is no node to insert before
    if (isEmpty()) {
        System.out.println("Linked list is empty");
        return;
    }

    // If the key is found at the head node, add the new node at the beginning
    if (head.data == key) {
        ndInput.next = head;
        head = ndInput;
        return;
    }

    // Traverse the linked list to find the key and insert the new node before it
    Node tmp = head;
    while (tmp.next != null) {
        if (tmp.next.data == key) {
            ndInput.next = tmp.next;
            tmp.next = ndInput;
            return;
        }
        tmp = tmp.next;
    }

    // If the key is not found in the linked list
    System.out.println("Key not found in the linked list");
}


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

```java
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 != key) {
        tmp = tmp.next;
        index++;
    }

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

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

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

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

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

```

- Main

```java
package Assnum3;

public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singleLinkedList = new SingleLinkedList();

        singleLinkedList.print();
        singleLinkedList.addFirst("Multimedia");
        singleLinkedList.print();
        singleLinkedList.addLast("Statistika");
        singleLinkedList.print();
        singleLinkedList.insertAfter("Statistika", "Matematika");
        singleLinkedList.print();
        singleLinkedList.insertBefore("Matematika", "Algoritma");
        singleLinkedList.print();
        singleLinkedList.insertAt(4, "Basis Data");
        singleLinkedList.print();
        singleLinkedList.addLast("Komputer");
        singleLinkedList.print();
        singleLinkedList.addLast("Android");
        singleLinkedList.print();
        singleLinkedList.addLast("Bahasa");
        singleLinkedList.print();
    }
}

```

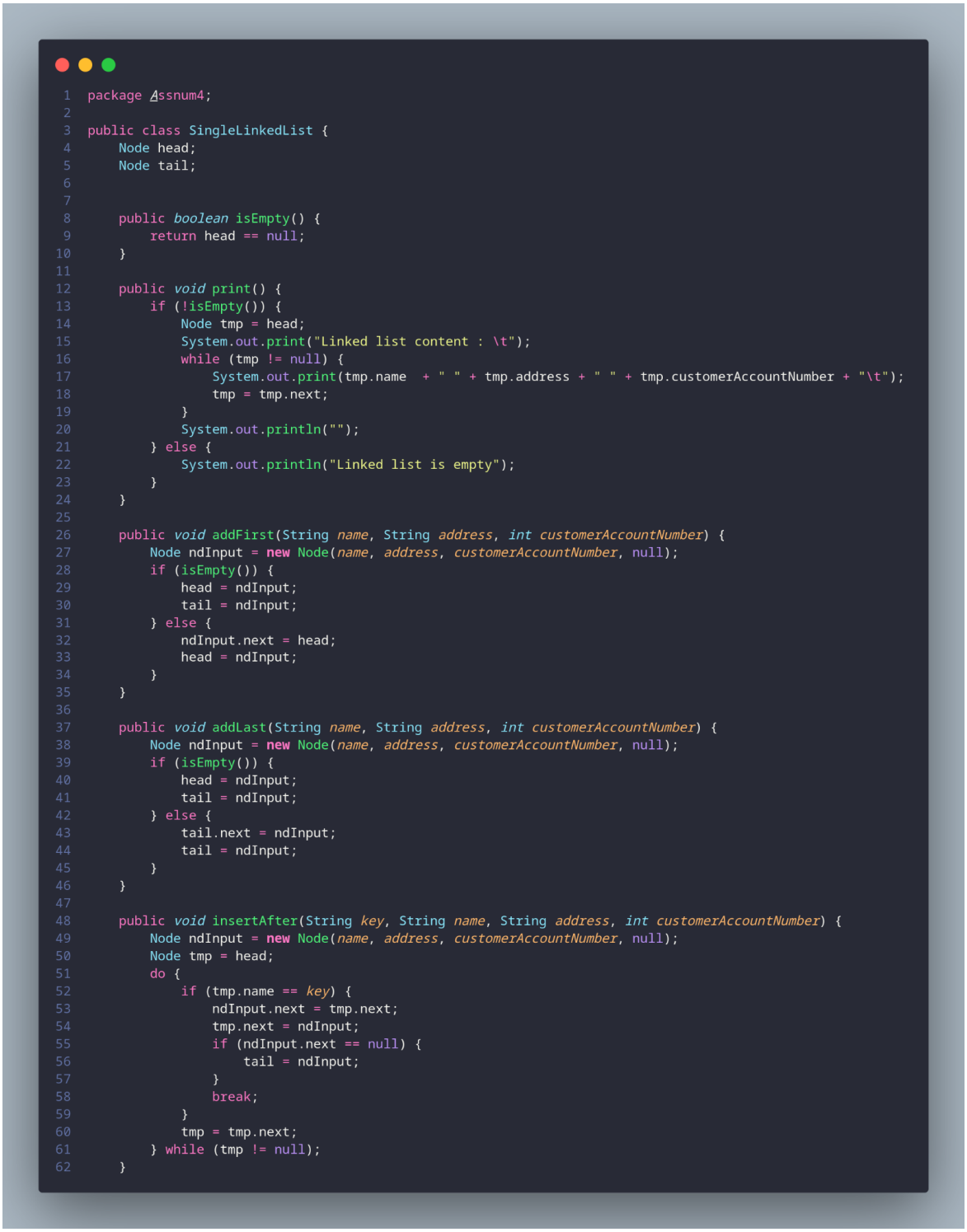
- Result

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /home/zharsuke/Documents/Colleg
ure_and_Algorithm_Practicum/Meet_11/coding/practicum/bin Assnum3.SLLMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Linked list is empty
Linked list content :   Multimedia
Linked list content :   Multimedia      Statistika
Linked list content :   Multimedia      Statistika      Matematika
Linked list content :   Multimedia      Statistika      Algoritma       Matematika
Linked list content :   Multimedia      Statistika      Algoritma       Matematika      Basis Data
Linked list content :   Multimedia      Statistika      Algoritma       Matematika      Basis Data      Komputer
Linked list content :   Multimedia      Statistika      Algoritma       Matematika      Basis Data      Komputer        Android
Linked list content :   Multimedia      Statistika      Algoritma       Matematika      Basis Data      Komputer        Android Bahasa

┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$
```

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

```java
package Assnum4;

public class Node {
    public String name;
    public String address;
    public 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;
    }
}

```

- Single linked List



```java
package Assnum4;

public class SingleLinkedList {
    Node head;
    Node tail;


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

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

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

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

    public void insertAfter(String key, String name, String address, int customerAccountNumber) {
        Node ndInput = new Node(name, address, customerAccountNumber, null);
        Node tmp = head;
        do {
            if (tmp.name == key) {
                ndInput.next = tmp.next;
                tmp.next = ndInput;
                if (ndInput.next == null) {
                    tail = ndInput;
                }
                break;
            }
            tmp = tmp.next;
        } while (tmp != null);
    }
```

```java
public void insertBefore(String key, String name, String address, int customerAccountNumber) {
    Node ndInput = new Node(name, address, customerAccountNumber, null);

    // If the list is empty, there is no node to insert before
    if (isEmpty()) {
        System.out.println("Linked list is empty");
        return;
    }

    // If the key is found at the head node, add the new node at the beginning
    if (head.name == key) {
        ndInput.next = head;
        head = ndInput;
        return;
    }

    // Traverse the linked list to find the key and insert the new node before it
    Node tmp = head;
    while (tmp.next != null) {
        if (tmp.next.name == key) {
            ndInput.next = tmp.next;
            tmp.next = ndInput;
            return;
        }
        tmp = tmp.next;
    }

    // If the key is not found in the linked list
    System.out.println("Key not found in the linked list");
}


public void insertAt(int index, String name, String address, int customerAccountNumber) {
    if (index < 0) {
        System.out.println("Wrong index");
    } else if (index == 0) {
        addFirst(name, address, customerAccountNumber);
    } else {
        Node tmp = head;
        for (int i = 0; i < index -1; i++) {
            tmp = tmp.next;
        }
        tmp.next = new Node(name, address, customerAccountNumber, tmp.next);
        if (tmp.next.next == null) tail = tmp.next;
    }
}
```

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

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

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

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

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

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

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

```

- Main

```java
package Assnum4;

public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singleLinkedList = new SingleLinkedList();

        singleLinkedList.print();
        singleLinkedList.addFirst("Azhar", "malang", 1234);
        singleLinkedList.print();
        singleLinkedList.addLast("Al", "Malang", 1243);
        singleLinkedList.print();
        singleLinkedList.insertAfter("Al", "Rizqi", "Surabaya", 7324);
        singleLinkedList.print();
        singleLinkedList.insertBefore("Rizqi", "Rifai", "Jakarta", 3424);
        singleLinkedList.print();
        singleLinkedList.insertAt(4, "Firdaus", "Jogja", 23324);
        singleLinkedList.print();
        singleLinkedList.addLast("zharsuke", "Malang", 21746);
        singleLinkedList.print();
    }
}

```

- Result

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /home/zharsuke/Documents/College/Semester_2/Data_Struc
ure_and_Algorithm_Practicum/Meet_11/coding/practicum/bin Assnum4.SLLMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Linked list is empty
Linked list content :   Azhar malang 1234
Linked list content :   Azhar malang 1234       Al Malang 1243
Linked list content :   Azhar malang 1234       Al Malang 1243  Rizqi Surabaya 7324
Linked list content :   Azhar malang 1234       Al Malang 1243  Rifai Jakarta 3424      Rizqi Surabaya 7324
Linked list content :   Azhar malang 1234       Al Malang 1243  Rifai Jakarta 3424      Rizqi Surabaya 7324     Firdaus Jogja 23324
Linked list content :   Azhar malang 1234       Al Malang 1243  Rifai Jakarta 3424      Rizqi Surabaya 7324     Firdaus Jogja 23324      zharsuke Malang 21746

┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$
```

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

```java
package Assnum4;

public class Queue {
    private SingleLinkedList list;

    public Queue() {
        list = new SingleLinkedList();
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    public void enqueue(String name, String address, int customerAccountNumber) {
        list.addLast(name, address, customerAccountNumber);
    }

    public void dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot dequeue.");
        } else {
            list.removeFirst();
        }
    }

    public void print() {
        list.print();
    }

}

```

- Main

```java
package Assnum4;

public class SLLMain {
    public static void main(String[] args) {
        // SingleLinkedList singleLinkedList = new SingleLinkedList();

        // singleLinkedList.print();
        // singleLinkedList.addFirst("Azhar", "malang", 1234);
        // singleLinkedList.print();
        // singleLinkedList.addLast("Al", "Malang", 1243);
        // singleLinkedList.print();
        // singleLinkedList.insertAfter("Al", "Rizqi", "Surabaya", 7324);
        // singleLinkedList.print();
        // singleLinkedList.insertBefore("Rizqi", "Rifai", "Jakarta", 3424);
        // singleLinkedList.print();
        // singleLinkedList.insertAt(4, "Firdaus", "Jogja", 23324);
        // singleLinkedList.print();
        // singleLinkedList.addLast("zharsuke", "Malang", 21746);
        // singleLinkedList.print();

        Queue queue = new Queue();

        queue.print();
        queue.enqueue("Azhar", "malang", 1234);
        queue.print();
        queue.enqueue("Al", "Malang", 1243);
        queue.print();
        // queue.dequeue();
        // queue.print();
    }
}

```

- Result

```
┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessage
s/College/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum/bin Assnum4.SL
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Linked list is empty
Linked list content :   Azhar malang 1234
Linked list content :   Azhar malang 1234       Al Malang 1243

┌──(zharsuke㉿asus-vivobook)-[~/…/Data_Structure_and_Algorithm_Practicum/Meet_11/coding/practicum]
└─$
```