NAME :- Jangili Ravali
EMAIL :- jangiliravali9@gmail.com

Day 5 :-

Task 1: Implementing a Linked List

1) **Write a class CustomLinkedList that implements a singly linked list with methods for InsertAtBeginning, InsertAtEnd, InsertAtPosition, DeleteNode, UpdateNode, and DisplayAllNodes. Test the class by performing a series of insertions, updates, and deletions.**

*CustomLinkedList.java ×

```java
package wiproPrgms;

import java.util.List;
class Node {
int data;
Node next;
Node(int data) {
this.data = data;
this.next = null;
}
}
public class CustomLinkedList {
private Node head;
public void insertAtBeginning(int data) {
Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }

    public void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node last = head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = newNode;
    }

    public void insertAtPosition(int data, int position) {
        if (position < 0) {
            System.out.println("Invalid position");
            return;
        }
        if (position == 0) {
            insertAtBeginning(data);
            return;
        }
```

Writable | Smart Insert | 26 : 27 : 557

NAME :- **Jangili Ravali**
EMAIL :- jangiliravali9@gmail.com

```java
    public void updateNode(int oldData, int newData) {
        Node current = head;
        while (current != null) {
            if (current.data == oldData) {
                current.data = newData;
                return;
            }
            current = current.next;
        }
        System.out.println("Data not found");
    }

    public void displayAllNodes() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        CustomLinkedList list = new CustomLinkedList();

        list.insertAtBeginning(9);
        list.insertAtEnd(3);
        list.insertAtEnd(4);
        list.insertAtPosition(2, 1);
        list.displayAllNodes();
```

```
Console ×
<terminated> CustomLinkedList [Java Application] C:\Users\Ravali\.p2\pool\plugins\org.eclipse.justj.o
Picked up _JAVA_OPTIONS: -Xmx512M
9 2 3 4
```

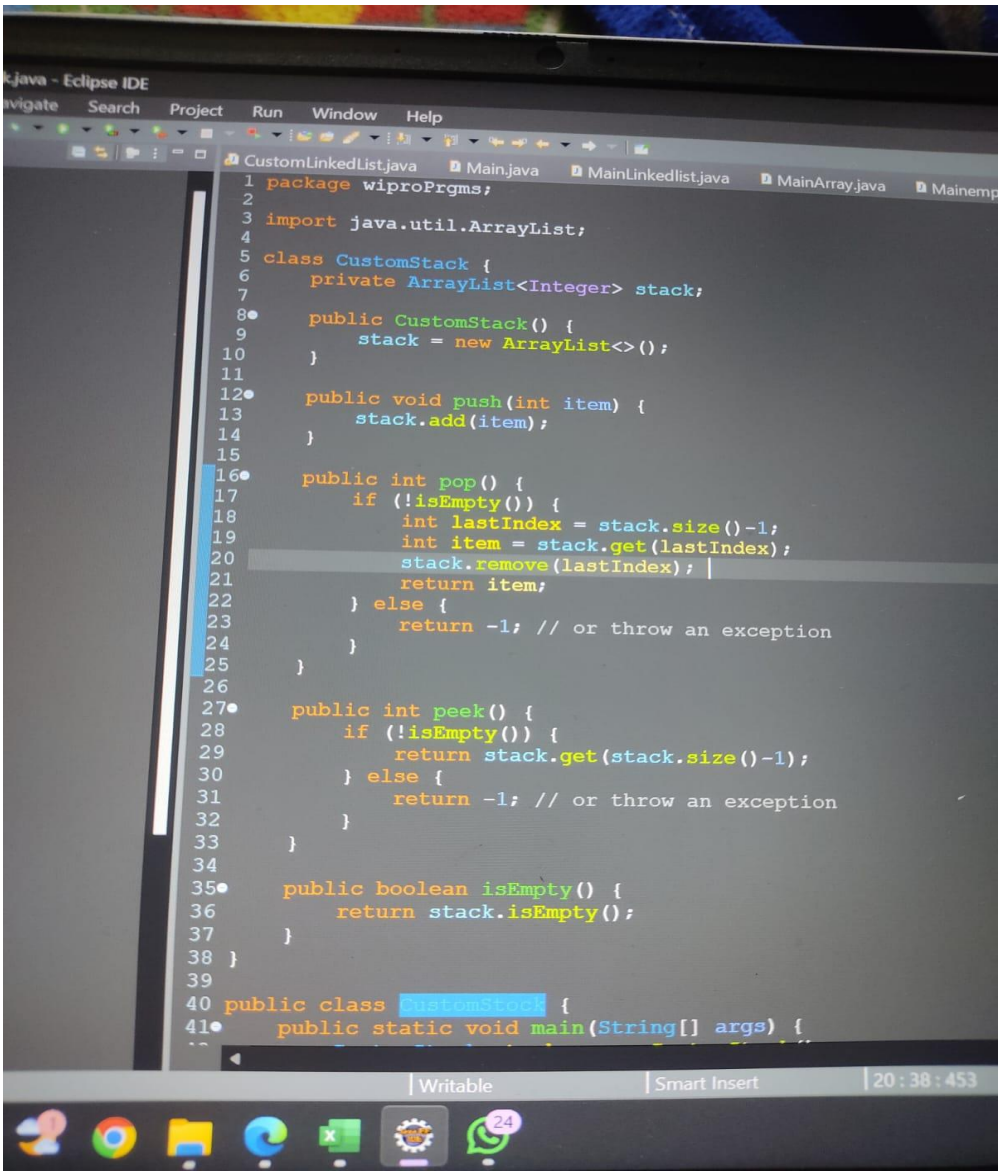
Writable | Smart Insert | 26 : 27 : 557

This Java code implements a singly linked list with methods for insertion at the beginning, end, and specific position, deletion, updating, and displaying all nodes. It also includes a main method for testing the functionalities.

NAME :- Jangili Ravali
EMAIL :- jangiliravali9@gmail.com

Task 2: Stack and Queue Operations

1) **Create a CustomStack class with operations Push, Pop, Peek, and IsEmpty. Demonstrate its LIFO behavior by pushing integers onto the stack, then popping and displaying them until the stack is empty.**




```java
package wiproPrgms;

import java.util.ArrayList;

class CustomStack {
    private ArrayList<Integer> stack;

    public CustomStack() {
        stack = new ArrayList<>();
    }

    public void push(int item) {
        stack.add(item);
    }

    public int pop() {
        if (!isEmpty()) {
            int lastIndex = stack.size()-1;
            int item = stack.get(lastIndex);
            stack.remove(lastIndex);
            return item;
        } else {
            return -1; // or throw an exception
        }
    }

    public int peek() {
        if (!isEmpty()) {
            return stack.get(stack.size()-1);
        } else {
            return -1; // or throw an exception
        }
    }

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

public class CustomStack {
    public static void main(String[] args) {
```

NAME :- **Jangili Ravali**
EMAIL :- jangiliravali9@gmail.com

```java
            return -1; // or throw an exception
        }
    }

    public int peek() {
        if (!isEmpty()) {
            return stack.get(stack.size()-1);
        } else {
            return -1; // or throw an exception
        }
    }

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

public class CustomStock {
    public static void main(String[] args) {
        CustomStack stack = new CustomStack();
        stack.push(1);
        stack.push(2);
        stack.push(3);

        while (!stack.isEmpty()) {
            System.out.println(stack.pop());
        }
    }
}
```

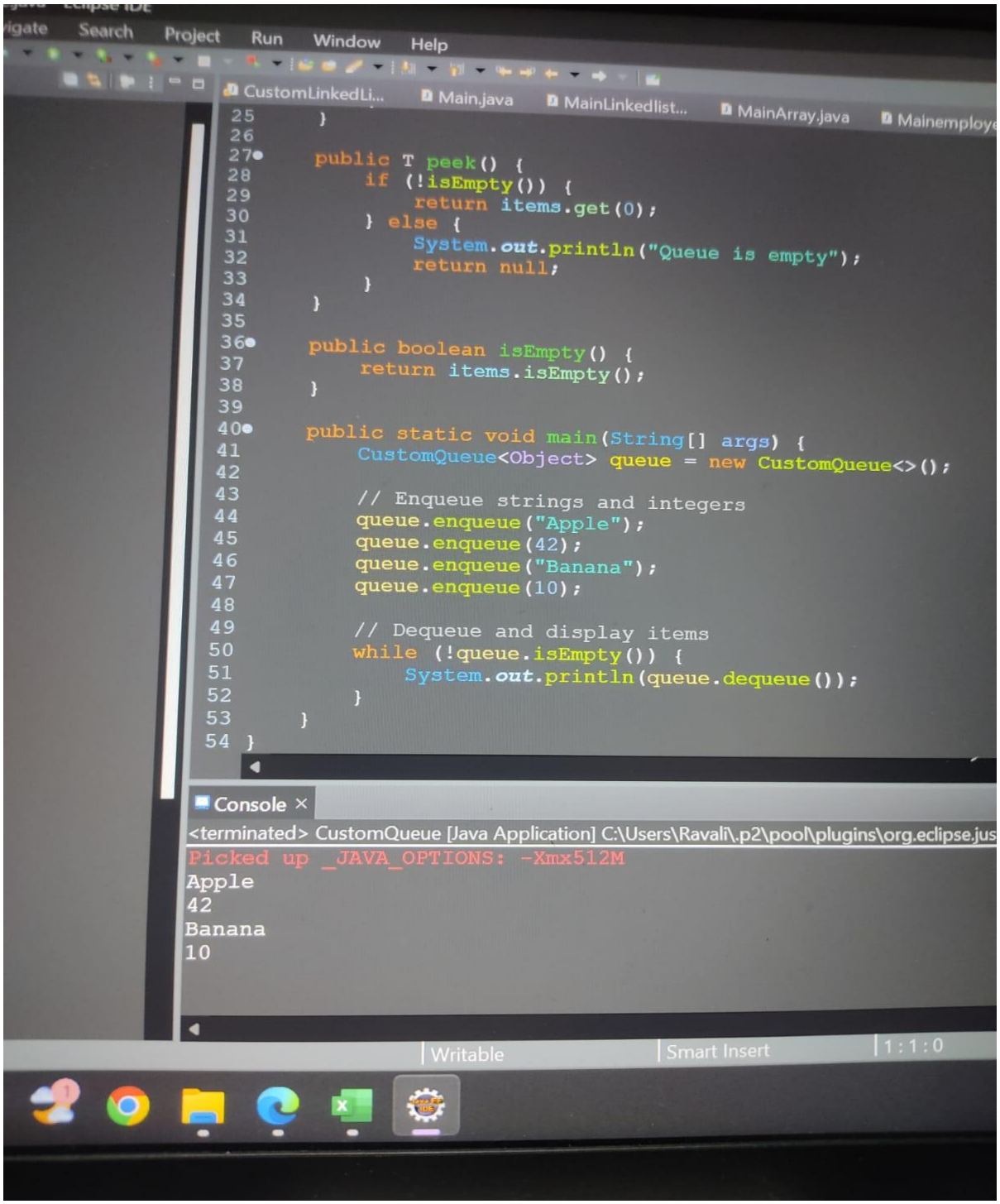
```
Console ×
<terminated> CustomStock [Java Application] C:\Users\Ravali\.p2\pool\plugins\org.eclipse.ju
Picked up _JAVA_OPTIONS: -Xmx512M
3
2
1
```

NAME :- Jangili Ravali
EMAIL :- jangiliravali9@gmail.com

2) **Develop a CustomQueue class with methods for Enqueue, Dequeue, Peek, and IsEmpty. Show how your queue can handle different data types by enqueuing strings and integers, then dequeui ng and displaying them to confirm FIFO order.**

```java
package wiproPrgms;

import java.util.ArrayList;

public class CustomQueue<T> {
    private List<T> items;

    public CustomQueue() {
        items = new ArrayList<>();
    }

    public void enqueue(T item) {
        items.add(item);
    }

    public T dequeue() {
        if (!isEmpty()) {
            return items.remove(0);
        } else {
            System.out.println("Queue is empty");
            return null;
        }
    }

    public T peek() {
        if (!isEmpty()) {
            return items.get(0);
        } else {
            System.out.println("Queue is empty");
            return null;
        }
    }

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

    public static void main(String[] args) {
        CustomQueue<Object> queue = new CustomQueue<>();

        // Enqueue strings and integers
```

NAME :- **Jangili Ravali**
EMAIL :- jangiliravali9@gmail.com



Java code defines a CustomQueue class with methods for enqueue, dequeue, peek, and isEmpty. You can enqueue strings and integers and then dequeue them to confirm FIFO (First In, First Out) order.

NAME :- **Jangili Ravali**
EMAIL :- jangiliravali9@gmail.com

Task 3: Priority Queue Scenario

a) **Implement a priority queue to manage emergency room admissions in a hospital. Patients with higher urgency should be served before those with lower urgency.**

implement a priority queue for managing emergency room admissions in Java:

```java
package wiproPrgms;

import java.util.PriorityQueue;

class Patient implements Comparable<Patient> {
    String name;
    int urgency;

    public Patient(String name, int urgency) {
        this.name = name;
        this.urgency = urgency;
    }

    @Override
    public int compareTo(Patient other) {
        // Patients with higher urgency should be served first
        return Integer.compare(other.urgency, this.urgency);
    }

    @Override
    public String toString() {
        return name + " (Urgency: " + urgency + ")";
    }
}

public class EmergencyRoom {
    PriorityQueue<Patient> patients;

    public EmergencyRoom() {
        patients = new PriorityQueue<>();
    }

    public void admitPatient(String name, int urgency) {
        patients.offer(new Patient(name, urgency));
    }

    public Patient treatPatient() {
        return patients.poll();
    }

    public static void main(String[] args) {
```

NAME :- **Jangili Ravali**
EMAIL :- jangiliravali9@gmail.com

```java
}

public class EmergencyRoom {
    PriorityQueue<Patient> patients;

    public EmergencyRoom() {
        patients = new PriorityQueue<>();
    }

    public void admitPatient(String name, int urgency) {
        patients.offer(new Patient(name, urgency));
    }

    public Patient treatPatient() {
        return patients.poll();
    }

    public static void main(String[] args) {
        EmergencyRoom emergencyRoom = new EmergencyRoom();
        emergencyRoom.admitPatient("John", 3);
        emergencyRoom.admitPatient("Jane", 1);
        emergencyRoom.admitPatient("David", 2);

        System.out.println("Treating patients in order of ur
        while (!emergencyRoom.patients.isEmpty()) {
            System.out.println(emergencyRoom.treatPatient())
}
}
}
```

```
Console ×
<terminated> EmergencyRoom [Java Application] C:\Users\Ravali\.p2\pool\plugins\org.eclipse.ju
Picked up _JAVA_OPTIONS: -Xmx512M
Treating patients in order of urgency:
John (Urgency: 3)
David (Urgency: 2)
Jane (Urgency: 1)
```

Writable | Smart Insert | 28 : 2 : 660

In this implementation, each patient is represented by the Patient class, which implements the Comparable interface to compare patients based on their urgency. The EmergencyRoom class uses a PriorityQueue to manage the patients, where patients with higher urgency values are served first. The main method demonstrates how to admit patients and treat them in order of urgency.

NAME :- **Jangili Ravali**
EMAIL :- jangiliravali9@gmail.com