

**NAME :-** Jangili Ravali

**EMAIL :-** [jangiliravali9@gmail.com](mailto:jangiliravali9@gmail.com)

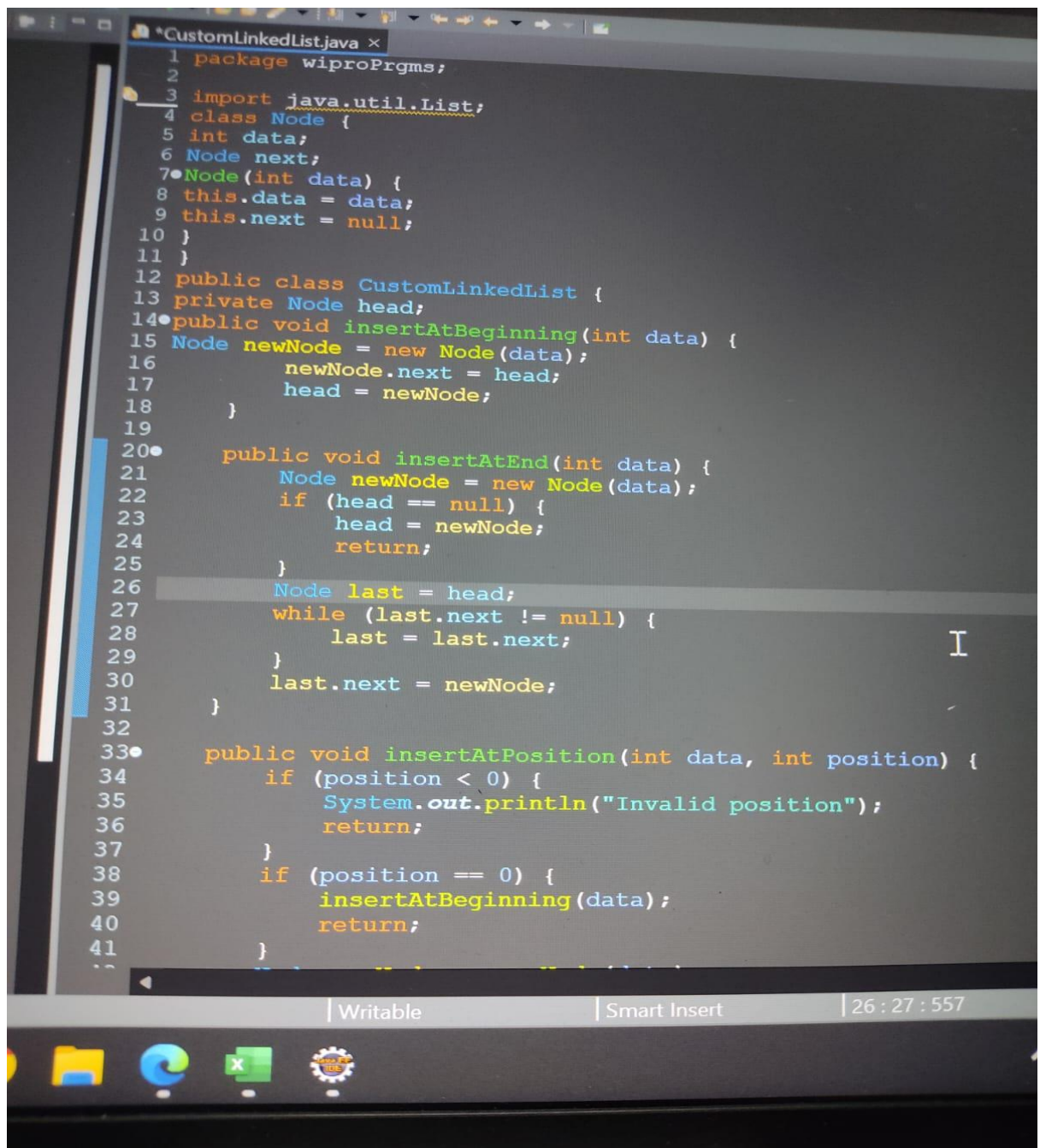
---

**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.



```
1 package wiproPrgms;
2
3 import java.util.List;
4 class Node {
5     int data;
6     Node next;
7     Node(int data) {
8         this.data = data;
9         this.next = null;
10    }
11 }
12 public class CustomLinkedList {
13     private Node head;
14     public void insertAtBeginning(int data) {
15         Node newNode = new Node(data);
16         newNode.next = head;
17         head = newNode;
18     }
19
20     public void insertAtEnd(int data) {
21         Node newNode = new Node(data);
22         if (head == null) {
23             head = newNode;
24             return;
25         }
26         Node last = head;
27         while (last.next != null) {
28             last = last.next;
29         }
30         last.next = newNode;
31     }
32
33     public void insertAtPosition(int data, int position) {
34         if (position < 0) {
35             System.out.println("Invalid position");
36             return;
37         }
38         if (position == 0) {
39             insertAtBeginning(data);
40             return;
41         }
42     }
43 }
```

The screenshot shows a Java IDE with a file named 'CustomLinkedList.java'. The code defines a 'Node' class with an 'int data' field and a 'Node next' reference. It then defines a 'CustomLinkedList' class with a private 'head' field and three public methods: 'insertAtBeginning', 'insertAtEnd', and 'insertAtPosition'. The 'insertAtBeginning' method creates a new node and inserts it at the start of the list. The 'insertAtEnd' method creates a new node and inserts it at the end of the list by traversing to the last node. The 'insertAtPosition' method checks for an invalid position and inserts at the beginning if the position is 0. The IDE interface includes a taskbar at the bottom with icons for File Explorer, Edge, and other applications, and a status bar showing 'Writable', 'Smart Insert', and a timestamp '26:27:557'.

**NAME :-** Jangili Ravali

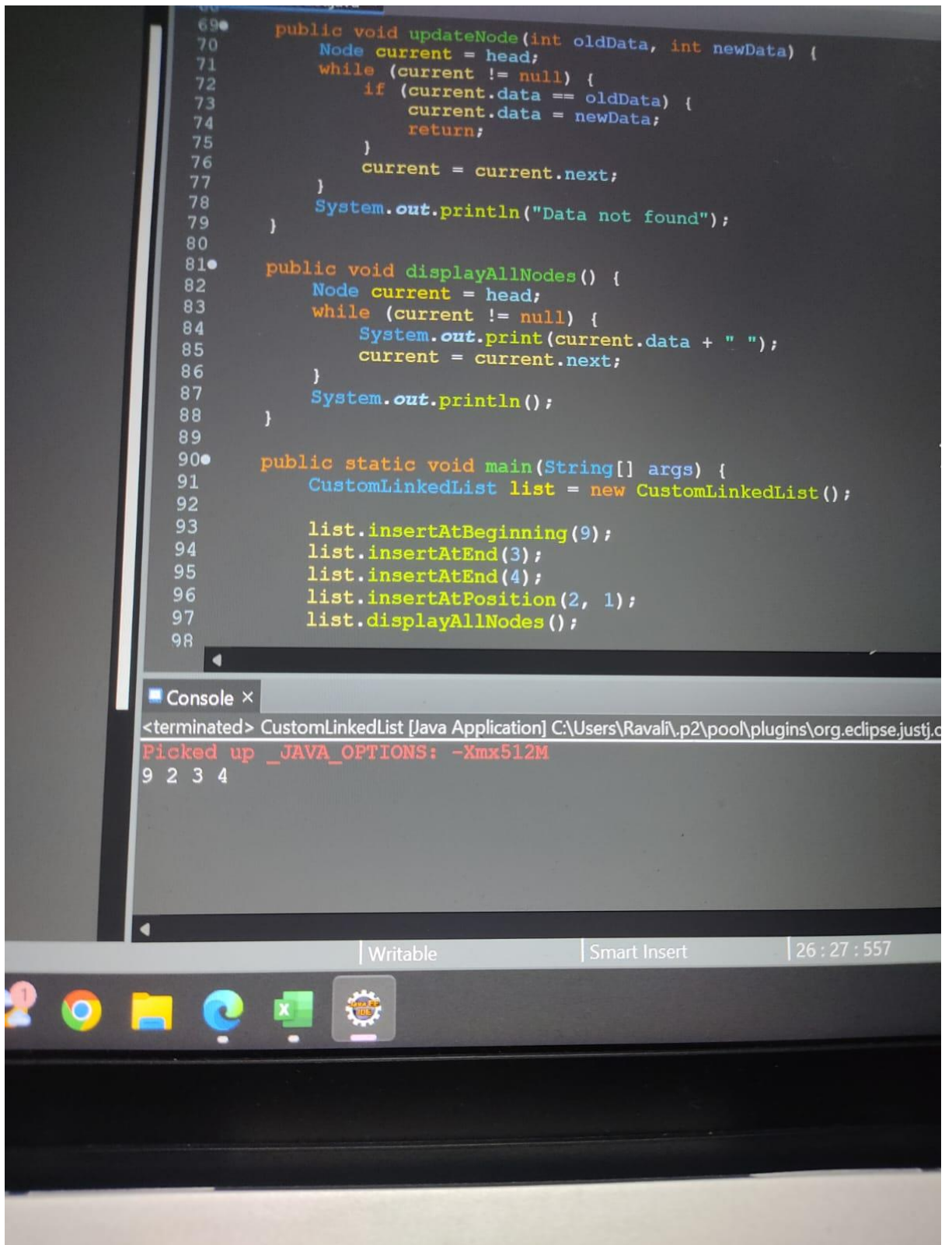
**EMAIL :-** [jangiliravali9@gmail.com](mailto:jangiliravali9@gmail.com)

```
69● public void updateNode(int oldData, int newData) {
70     Node current = head;
71     while (current != null) {
72         if (current.data == oldData) {
73             current.data = newData;
74             return;
75         }
76         current = current.next;
77     }
78     System.out.println("Data not found");
79 }
80
81● public void displayAllNodes() {
82     Node current = head;
83     while (current != null) {
84         System.out.print(current.data + " ");
85         current = current.next;
86     }
87     System.out.println();
88 }
89
90● public static void main(String[] args) {
91     CustomLinkedList list = new CustomLinkedList();
92
93     list.insertAtBeginning(9);
94     list.insertAtEnd(3);
95     list.insertAtEnd(4);
96     list.insertAtPosition(2, 1);
97     list.displayAllNodes();
98 }
```

Console ×

<terminated> CustomLinkedList [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse.justj.c  
Picked up \_JAVA\_OPTIONS: -Xmx512M  
9 2 3 4

Writable Smart Insert 26 : 27 : 557

The image shows a screenshot of an Eclipse IDE. The main editor window displays a Java class named CustomLinkedList. It contains three methods: updateNode, displayAllNodes, and main. The updateNode method iterates through the linked list to find a node with a specific oldData and updates its data to newData. The displayAllNodes method iterates through the list and prints the data of each node. The main method creates a new CustomLinkedList object, inserts nodes at the beginning, end, and a specific position, and then calls displayAllNodes to show the resulting list. The console window at the bottom shows the output of the main method, which is "9 2 3 4". The status bar at the bottom of the IDE indicates "Writable", "Smart Insert", and the time "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.

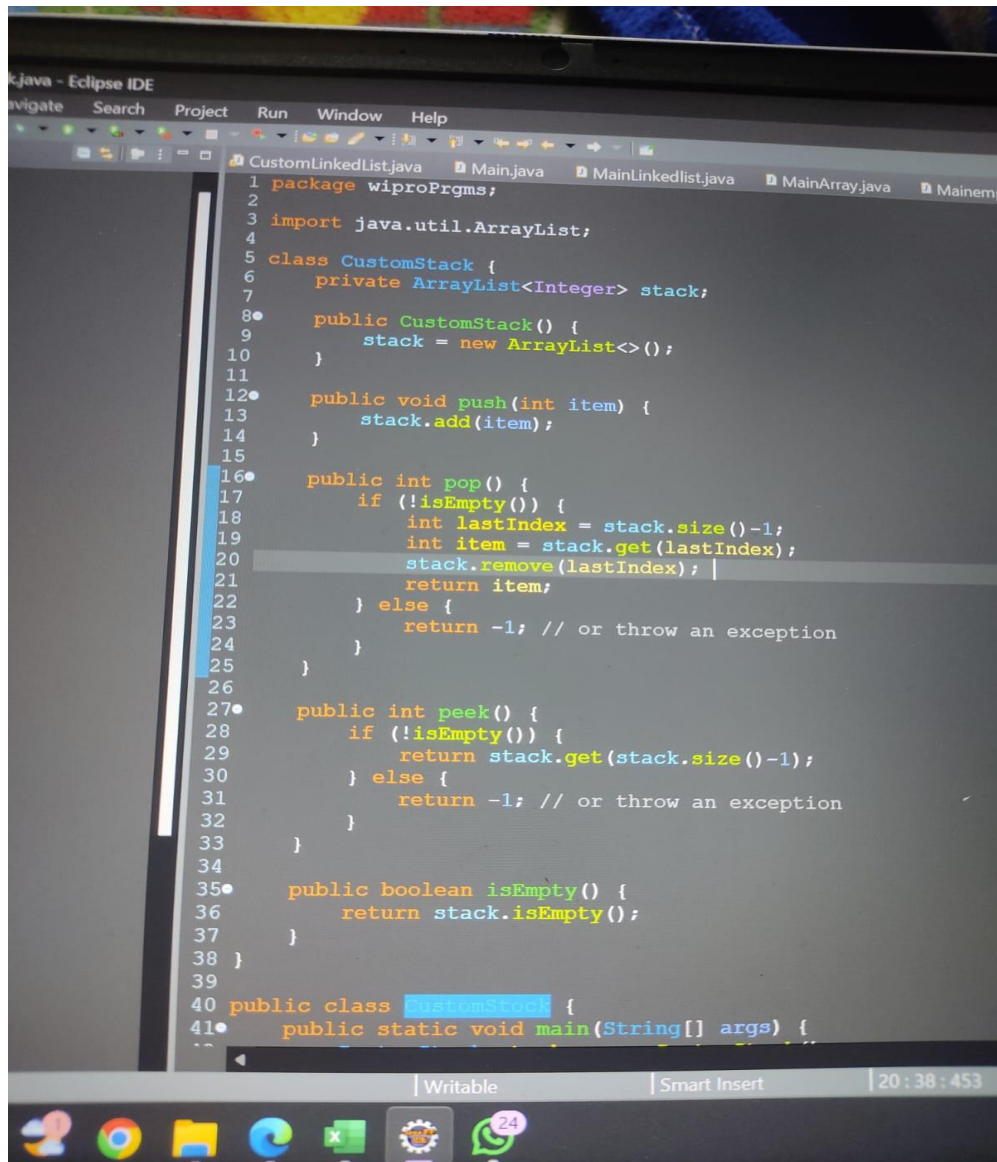
**NAME :-** Jangili Ravali

**EMAIL :-** [jangiliravali9@gmail.com](mailto: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.



```
1 package wiproPrgrms;
2
3 import java.util.ArrayList;
4
5 class CustomStack {
6     private ArrayList<Integer> stack;
7
8     public CustomStack() {
9         stack = new ArrayList<>();
10    }
11
12    public void push(int item) {
13        stack.add(item);
14    }
15
16    public int pop() {
17        if (!isEmpty()) {
18            int lastIndex = stack.size()-1;
19            int item = stack.get(lastIndex);
20            stack.remove(lastIndex);
21            return item;
22        } else {
23            return -1; // or throw an exception
24        }
25    }
26
27    public int peek() {
28        if (!isEmpty()) {
29            return stack.get(stack.size()-1);
30        } else {
31            return -1; // or throw an exception
32        }
33    }
34
35    public boolean isEmpty() {
36        return stack.isEmpty();
37    }
38 }
39
40 public class CustomStack {
41     public static void main(String[] args) {
```



**NAME :-** Jangili Ravali

**EMAIL :-** [jangiliravali9@gmail.com](mailto:jangiliravali9@gmail.com)

```
23         return -1; // or throw an exception
24     }
25 }
26
27 public int peek() {
28     if (!isEmpty()) {
29         return stack.get(stack.size()-1);
30     } else {
31         return -1; // or throw an exception
32     }
33 }
34
35 public boolean isEmpty() {
36     return stack.isEmpty();
37 }
38 }
39
40 public class CustomStack {
41     public static void main(String[] args) {
42         CustomStack stack = new CustomStack();
43         stack.push(1);
44         stack.push(2);
45         stack.push(3);
46
47         while (!stack.isEmpty()) {
48             System.out.println(stack.pop());
49         }
50     }
51 }
52
```

Console ×

<terminated> CustomStock [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse.ju

Picked up \_JAVA\_OPTIONS: -Xmx512M

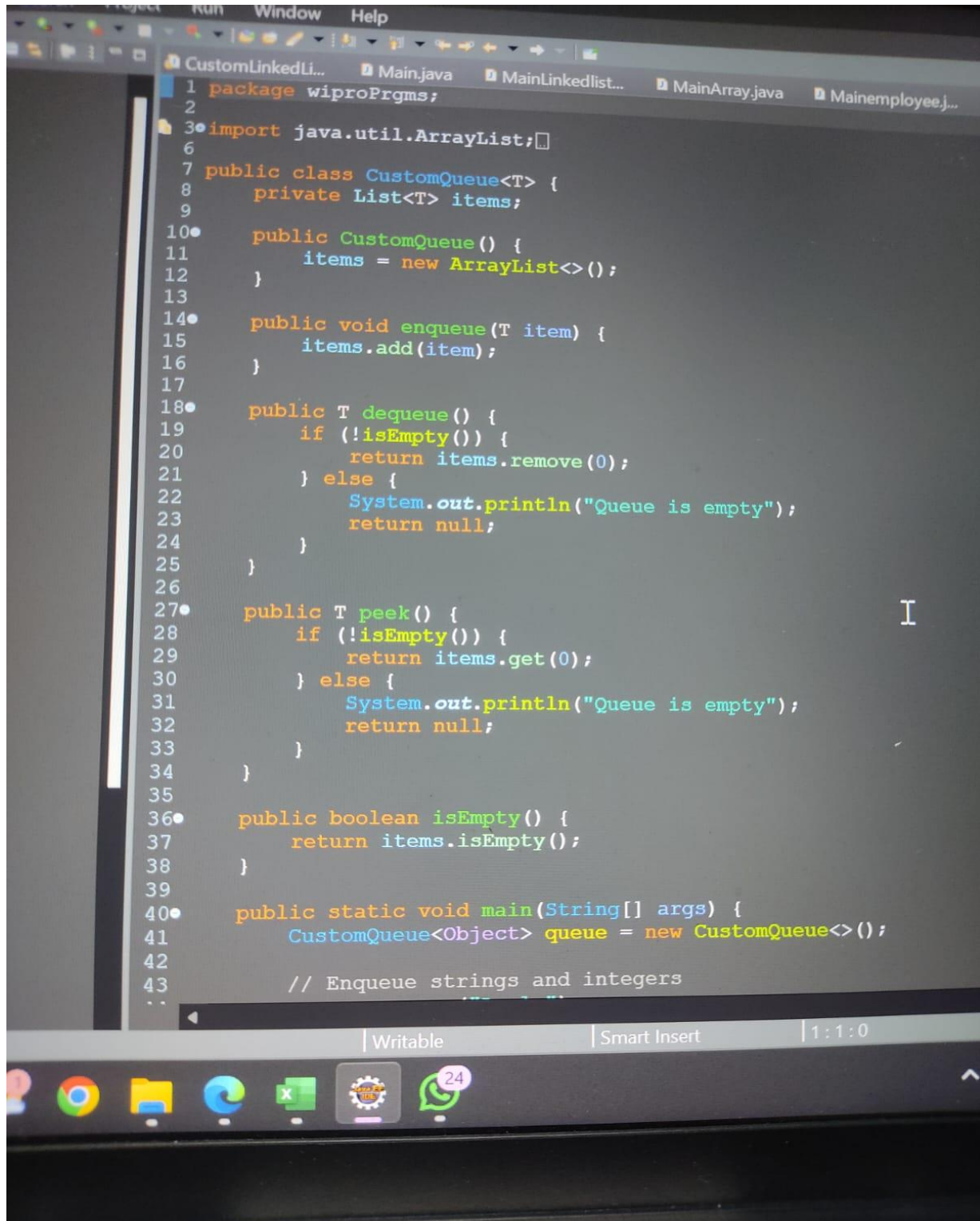
3  
2  
1

Writable | Smart Insert | 20 : 38 : 45

**NAME :-** Jangili Ravali

**EMAIL :-** [jangiliravali9@gmail.com](mailto: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 enqueueing strings and integers, then dequeuing and displaying them to confirm FIFO order.

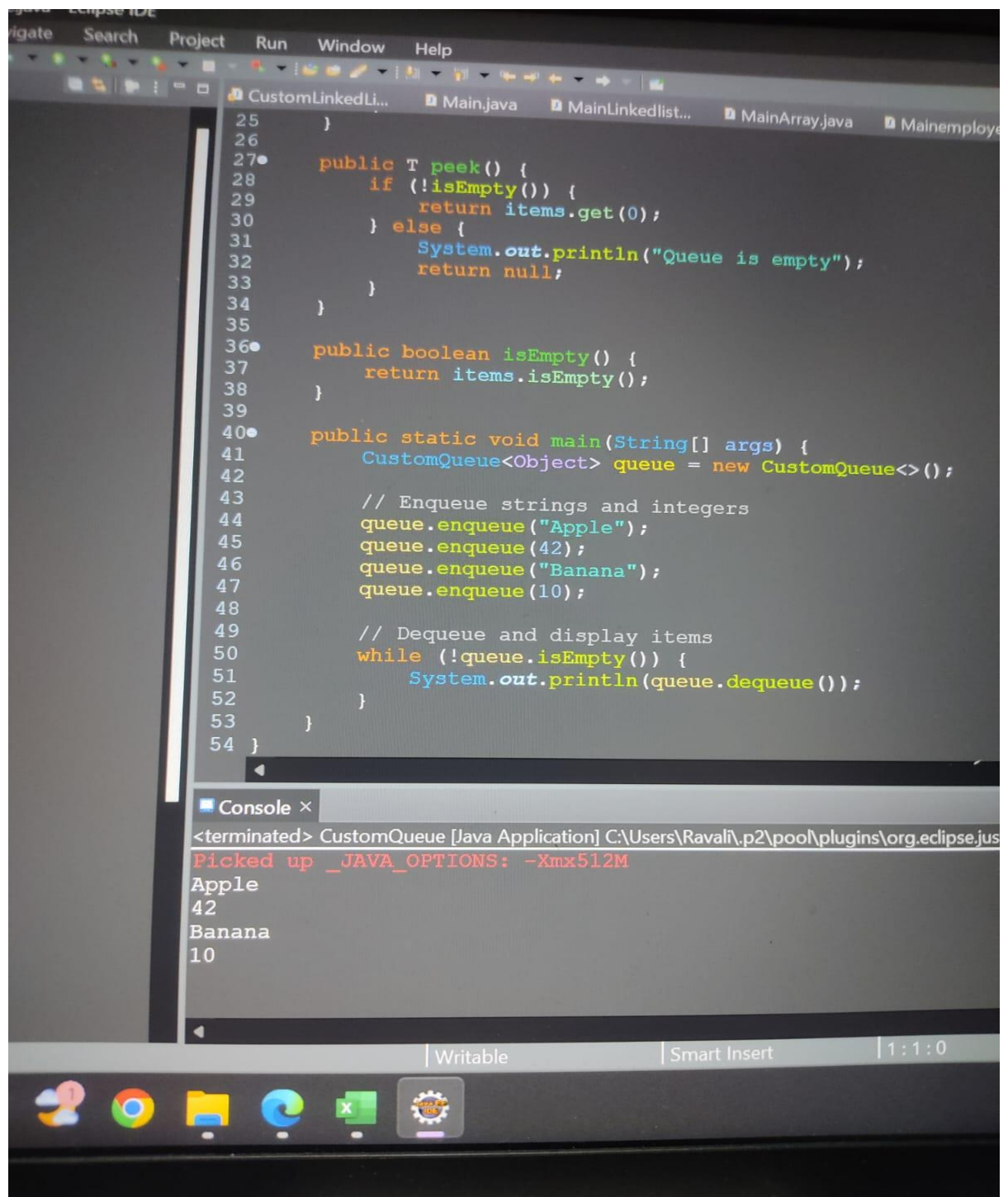


```
1 package wiproPrgrms;
2
3 import java.util.ArrayList;
4
5
6 public class CustomQueue<T> {
7     private List<T> items;
8
9
10    public CustomQueue() {
11        items = new ArrayList<>();
12    }
13
14    public void enqueue(T item) {
15        items.add(item);
16    }
17
18    public T dequeue() {
19        if (!isEmpty()) {
20            return items.remove(0);
21        } else {
22            System.out.println("Queue is empty");
23            return null;
24        }
25    }
26
27    public T peek() {
28        if (!isEmpty()) {
29            return items.get(0);
30        } else {
31            System.out.println("Queue is empty");
32            return null;
33        }
34    }
35
36    public boolean isEmpty() {
37        return items.isEmpty();
38    }
39
40    public static void main(String[] args) {
41        CustomQueue<Object> queue = new CustomQueue<>();
42
43        // Enqueue strings and integers
44    }
```

The screenshot shows an IDE with a Java file named CustomQueue.java. The code implements a CustomQueue class using an ArrayList. It includes methods for enqueue, dequeue, peek, and isEmpty. The main method demonstrates enqueueing strings and integers. The IDE interface includes a menu bar (File, Edit, View, Run, Window, Help), a toolbar, and a status bar at the bottom with icons for various applications and a notification badge.

**NAME :-** Jangili Ravali

**EMAIL :-** [jangiliravali9@gmail.com](mailto:jangiliravali9@gmail.com)



The screenshot shows the Eclipse IDE with a Java project. The editor displays the code for a `CustomQueue` class. The code includes methods for `peek`, `isEmpty`, and a `main` method that enqueues and dequeues elements. The console window at the bottom shows the output of the program, which is the sequence of elements dequeued: Apple, 42, Banana, and 10.

```
25     }
26
27     public T peek() {
28         if (!isEmpty()) {
29             return items.get(0);
30         } else {
31             System.out.println("Queue is empty");
32             return null;
33         }
34     }
35
36     public boolean isEmpty() {
37         return items.isEmpty();
38     }
39
40     public static void main(String[] args) {
41         CustomQueue<Object> queue = new CustomQueue<>();
42
43         // Enqueue strings and integers
44         queue.enqueue("Apple");
45         queue.enqueue(42);
46         queue.enqueue("Banana");
47         queue.enqueue(10);
48
49         // Dequeue and display items
50         while (!queue.isEmpty()) {
51             System.out.println(queue.dequeue());
52         }
53     }
54 }
```

Console Output:

```
<terminated> CustomQueue [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse.jst
Picked up _JAVA_OPTIONS: -Xmx512M
Apple
42
Banana
10
```

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](mailto:jangiliravali9@gmail.com)

### Task 3: Priority Queue Scenario

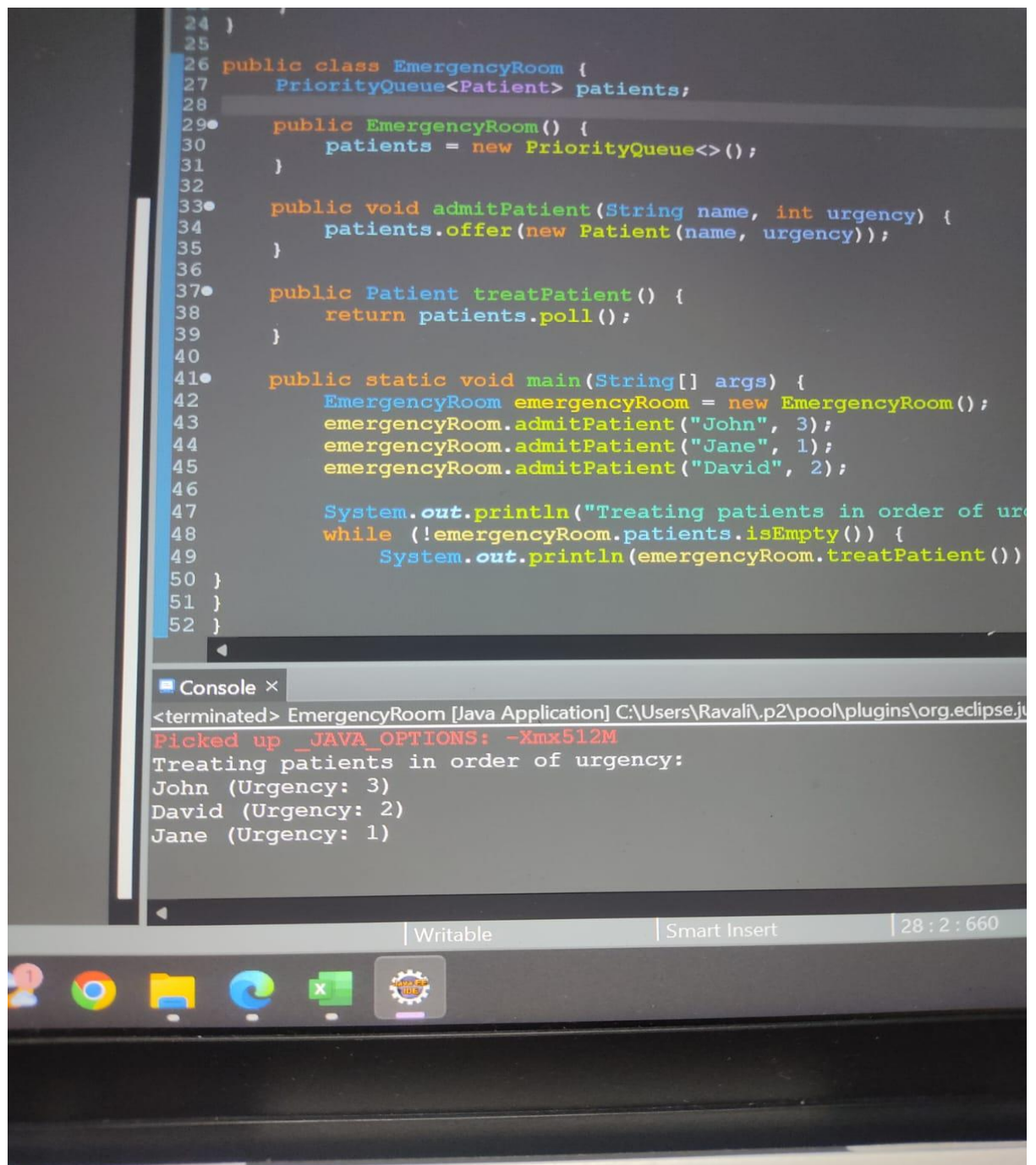
- a) Implement a priority queue to manage emergency room admissions in a hospital. Patients with higher urgency should be served before those with lower urgency.

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

```
1 package wiproPrgrms;
2
3 import java.util.PriorityQueue;
4
5 class Patient implements Comparable<Patient> {
6     String name;
7     int urgency;
8
9     public Patient(String name, int urgency) {
10         this.name = name;
11         this.urgency = urgency;
12     }
13
14     @Override
15     public int compareTo(Patient other) {
16         // Patients with higher urgency should be served first
17         return Integer.compare(other.urgency, this.urgency);
18     }
19
20     @Override
21     public String toString() {
22         return name + " (Urgency: " + urgency + ")";
23     }
24 }
25
26 public class EmergencyRoom {
27     PriorityQueue<Patient> patients;
28
29     public EmergencyRoom() {
30         patients = new PriorityQueue<>();
31     }
32
33     public void admitPatient(String name, int urgency) {
34         patients.offer(new Patient(name, urgency));
35     }
36
37     public Patient treatPatient() {
38         return patients.poll();
39     }
40
41     public static void main(String[] args) {
```

**NAME :-** Jangili Ravali

**EMAIL :-** [jangiliravali9@gmail.com](mailto:jangiliravali9@gmail.com)



```
24 )
25
26 public class EmergencyRoom {
27     PriorityQueue<Patient> patients;
28
29     public EmergencyRoom() {
30         patients = new PriorityQueue<>();
31     }
32
33     public void admitPatient(String name, int urgency) {
34         patients.offer(new Patient(name, urgency));
35     }
36
37     public Patient treatPatient() {
38         return patients.poll();
39     }
40
41     public static void main(String[] args) {
42         EmergencyRoom emergencyRoom = new EmergencyRoom();
43         emergencyRoom.admitPatient("John", 3);
44         emergencyRoom.admitPatient("Jane", 1);
45         emergencyRoom.admitPatient("David", 2);
46
47         System.out.println("Treating patients in order of urgency");
48         while (!emergencyRoom.patients.isEmpty()) {
49             System.out.println(emergencyRoom.treatPatient());
50         }
51     }
52 }
```

Console ×

<terminated> EmergencyRoom [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse.jdt.launcher\org.eclipse.jdt.launcher.exe

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](mailto:jangiliravali9@gmail.com)