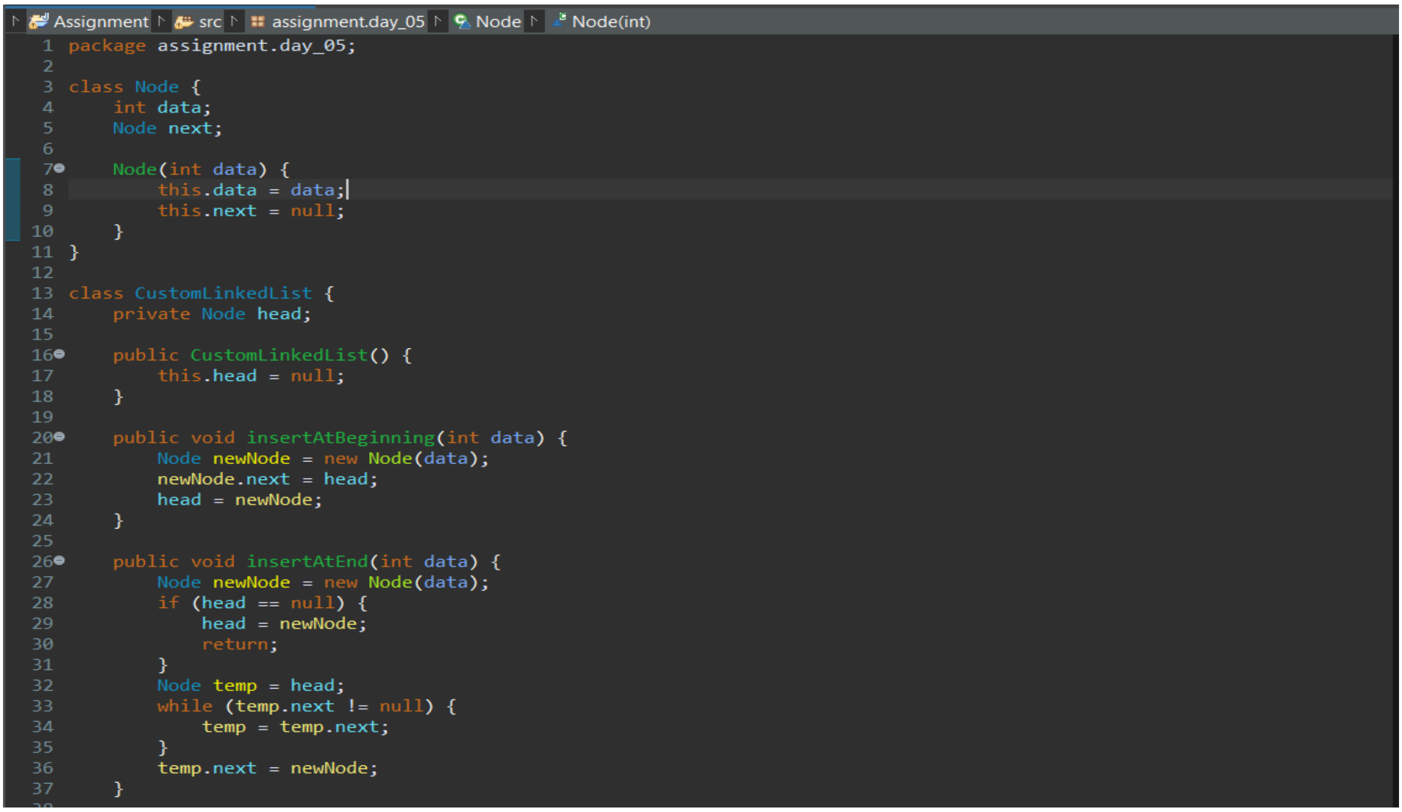
Java Assignment-(Day 5)

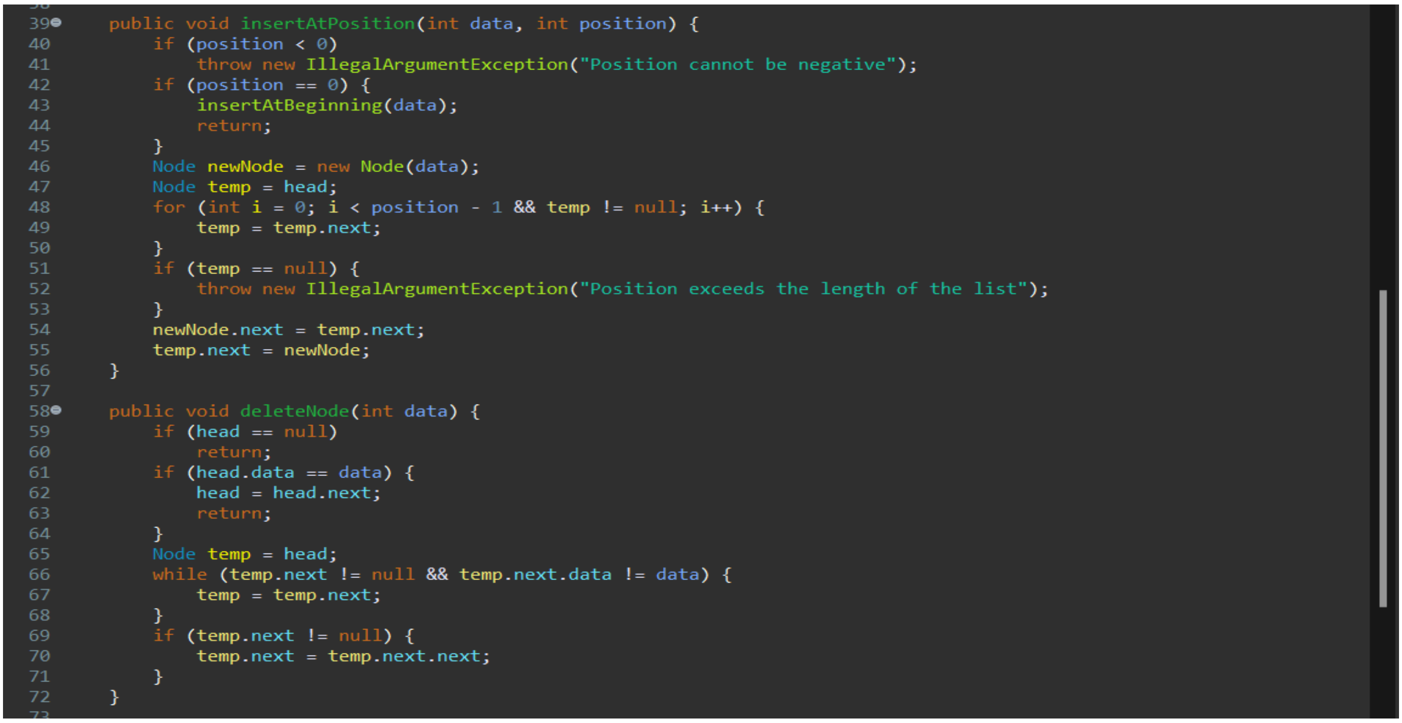
Task 1: Implementing a Linked List

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.

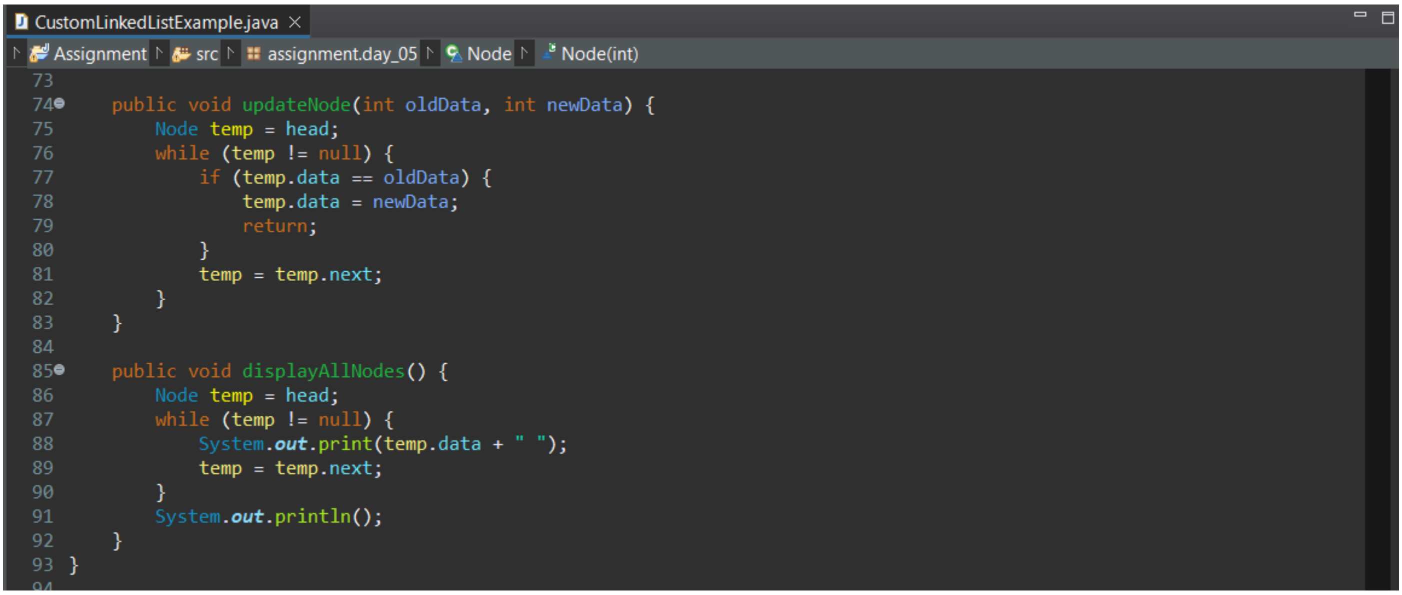
Code:



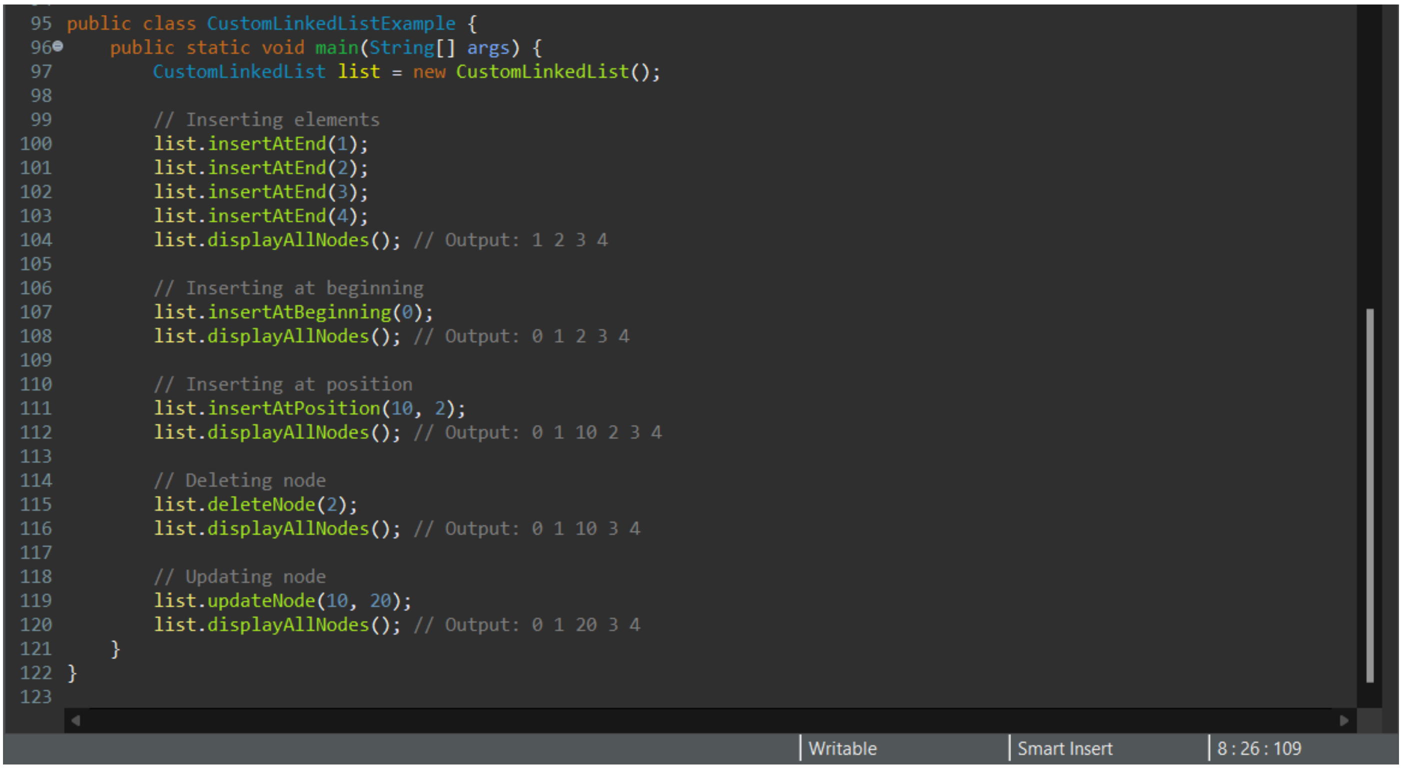
```java
package assignment.day_05;

class Node {
    int data;
    Node next;

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

class CustomLinkedList {
    private Node head;

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

    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 temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
```



```java
    public void insertAtPosition(int data, int position) {
        if (position < 0)
            throw new IllegalArgumentException("Position cannot be negative");
        if (position == 0) {
            insertAtBeginning(data);
            return;
        }
        Node newNode = new Node(data);
        Node temp = head;
        for (int i = 0; i < position - 1 && temp != null; i++) {
            temp = temp.next;
        }
        if (temp == null) {
            throw new IllegalArgumentException("Position exceeds the length of the list");
        }
        newNode.next = temp.next;
        temp.next = newNode;
    }

    public void deleteNode(int data) {
        if (head == null)
            return;
        if (head.data == data) {
            head = head.next;
            return;
        }
        Node temp = head;
        while (temp.next != null && temp.next.data != data) {
            temp = temp.next;
        }
        if (temp.next != null) {
            temp.next = temp.next.next;
        }
    }
```

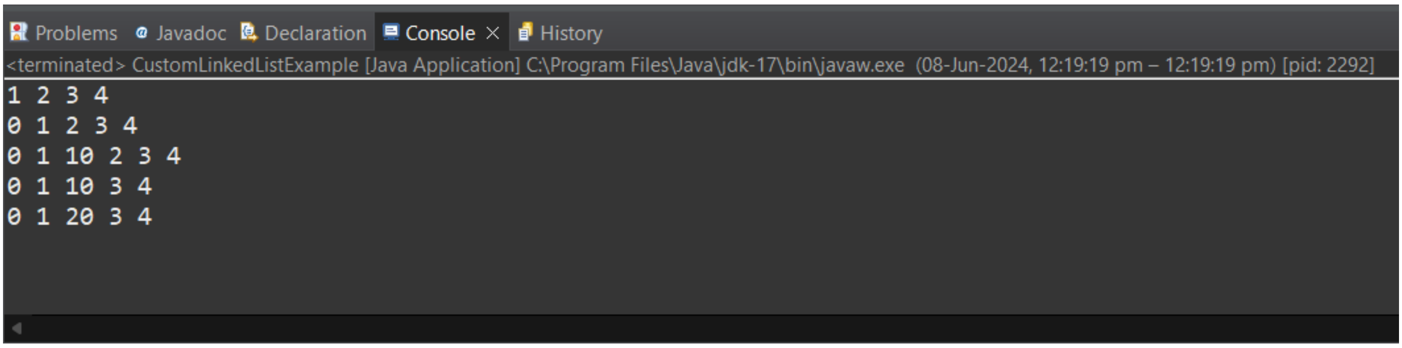


```java

    public void updateNode(int oldData, int newData) {
        Node temp = head;
        while (temp != null) {
            if (temp.data == oldData) {
                temp.data = newData;
                return;
            }
            temp = temp.next;
        }
    }

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



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

        // Inserting elements
        list.insertAtEnd(1);
        list.insertAtEnd(2);
        list.insertAtEnd(3);
        list.insertAtEnd(4);
        list.displayAllNodes(); // Output: 1 2 3 4

        // Inserting at beginning
        list.insertAtBeginning(0);
        list.displayAllNodes(); // Output: 0 1 2 3 4

        // Inserting at position
        list.insertAtPosition(10, 2);
        list.displayAllNodes(); // Output: 0 1 10 2 3 4

        // Deleting node
        list.deleteNode(2);
        list.displayAllNodes(); // Output: 0 1 10 3 4

        // Updating node
        list.updateNode(10, 20);
        list.displayAllNodes(); // Output: 0 1 20 3 4
    }
}

```

Output:

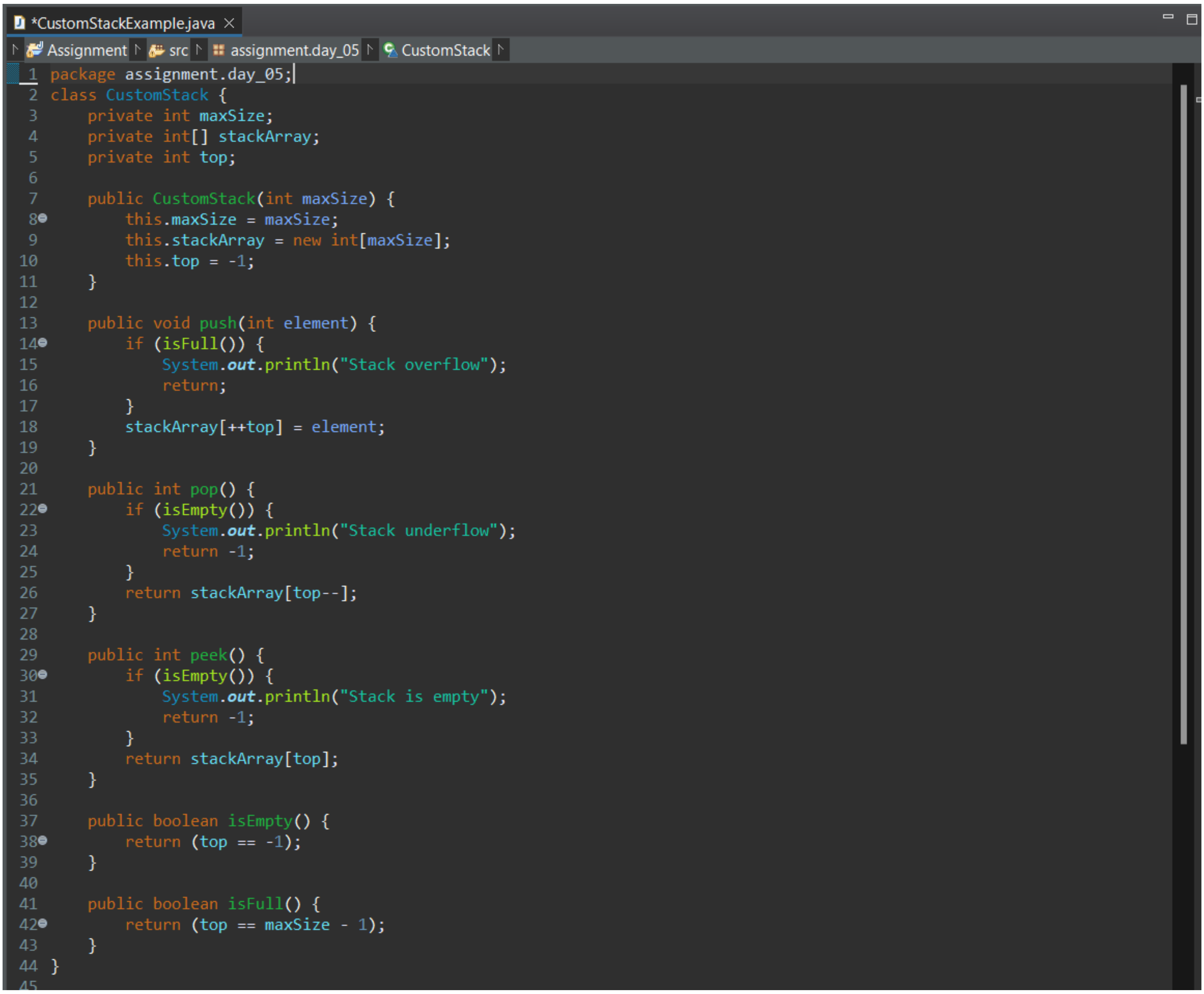


```
<terminated> CustomLinkedListExample [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (08-Jun-2024, 12:19:19 pm – 12:19:19 pm) [pid: 2292]
1 2 3 4
0 1 2 3 4
0 1 10 2 3 4
0 1 10 3 4
0 1 20 3 4
```

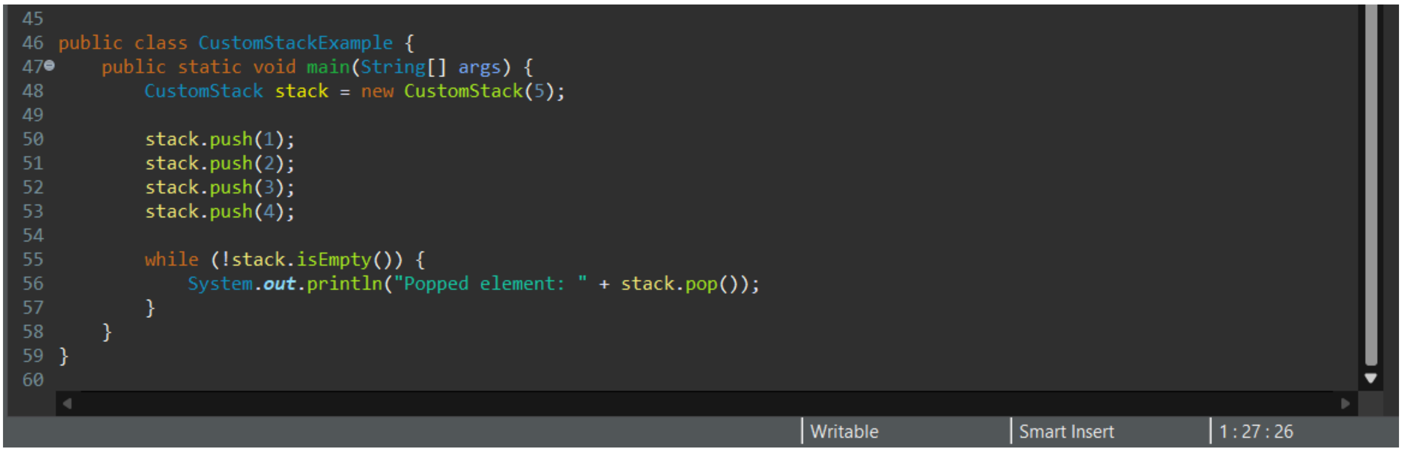
Task 2: Stack and Queue Operations

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.

Code:

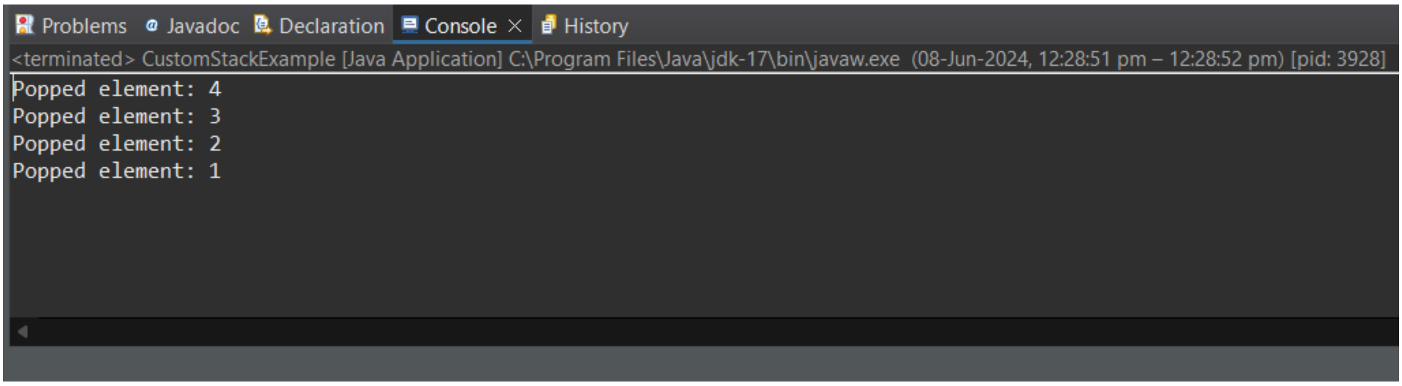


```java
package assignment.day_05;
class CustomStack {
    private int maxSize;
    private int[] stackArray;
    private int top;

    public CustomStack(int maxSize) {
        this.maxSize = maxSize;
        this.stackArray = new int[maxSize];
        this.top = -1;
    }

    public void push(int element) {
        if (isFull()) {
            System.out.println("Stack overflow");
            return;
        }
        stackArray[++top] = element;
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack underflow");
            return -1;
        }
        return stackArray[top--];
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty");
            return -1;
        }
        return stackArray[top];
    }

    public boolean isEmpty() {
        return (top == -1);
    }

    public boolean isFull() {
        return (top == maxSize - 1);
    }
}
```



```java

public class CustomStackExample {
    public static void main(String[] args) {
        CustomStack stack = new CustomStack(5);

        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);

        while (!stack.isEmpty()) {
            System.out.println("Popped element: " + stack.pop());
        }
    }
}

```

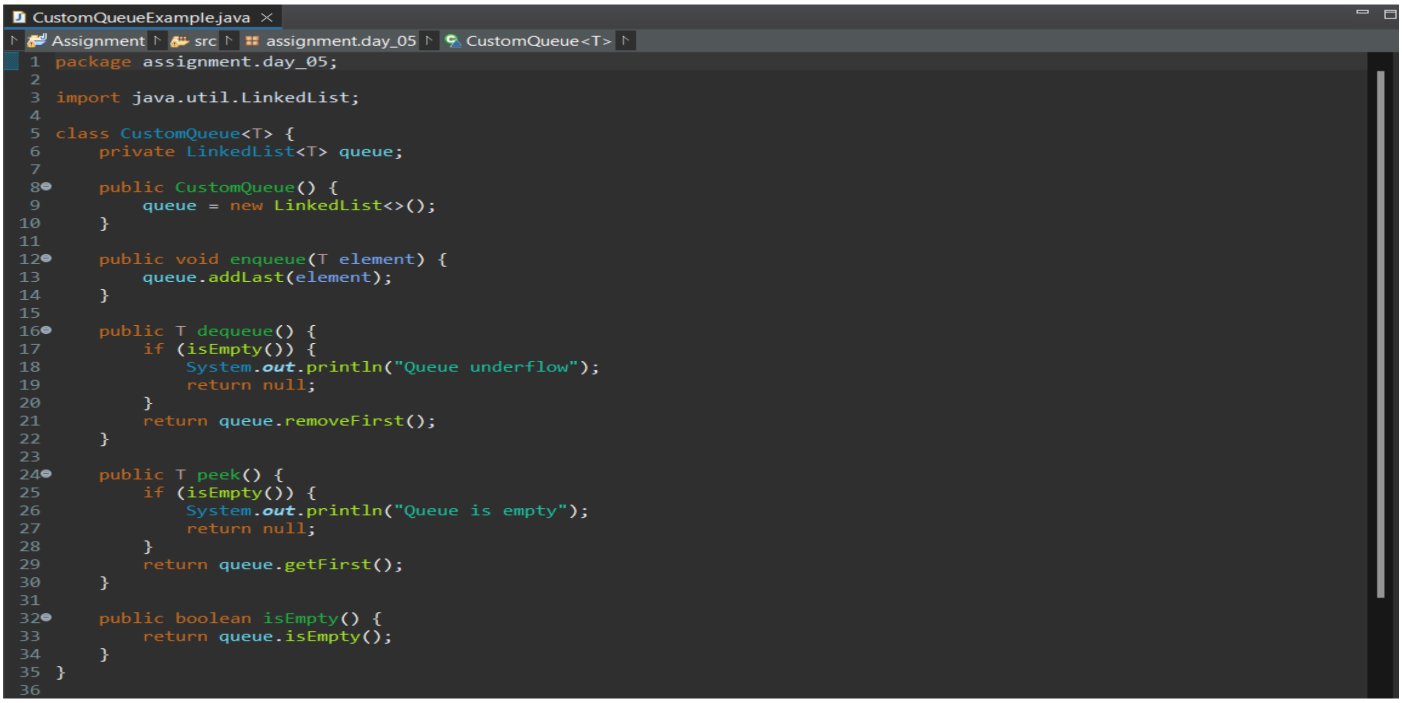
Output:



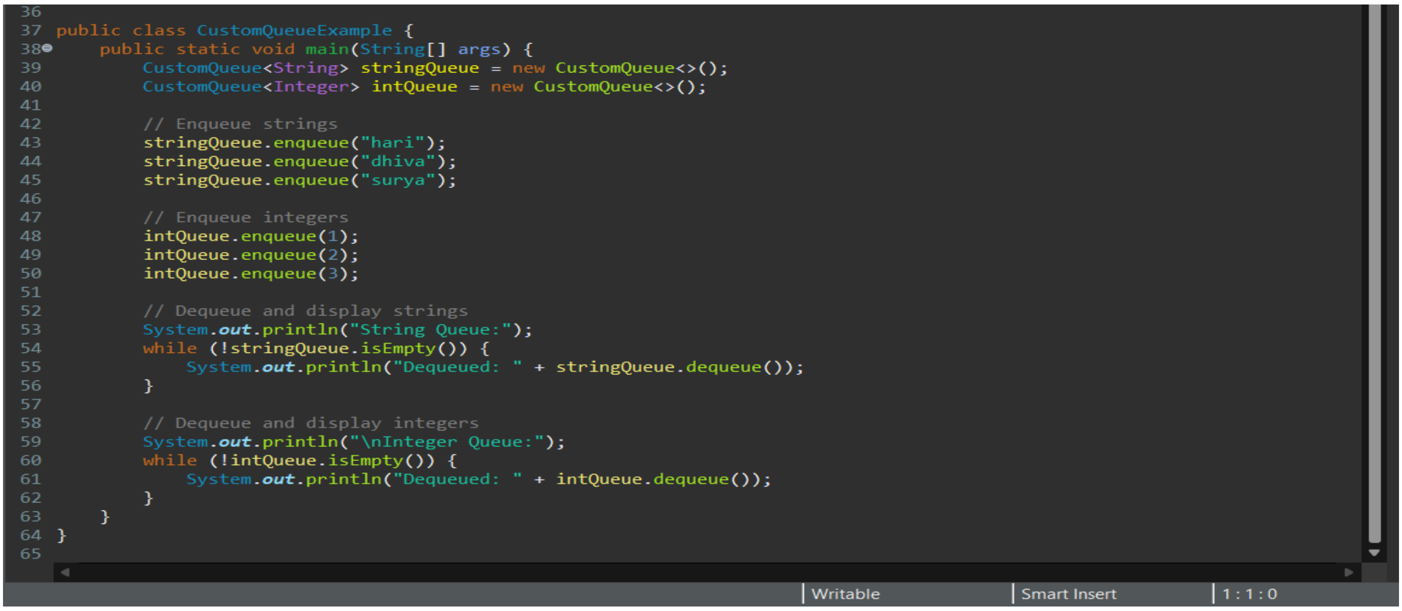
```
<terminated> CustomStackExample [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (08-Jun-2024, 12:28:51 pm – 12:28:52 pm) [pid: 3928]
Popped element: 4
Popped element: 3
Popped element: 2
Popped element: 1
```

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 dequeuing and displaying them to confirm FIFO order.

Code:

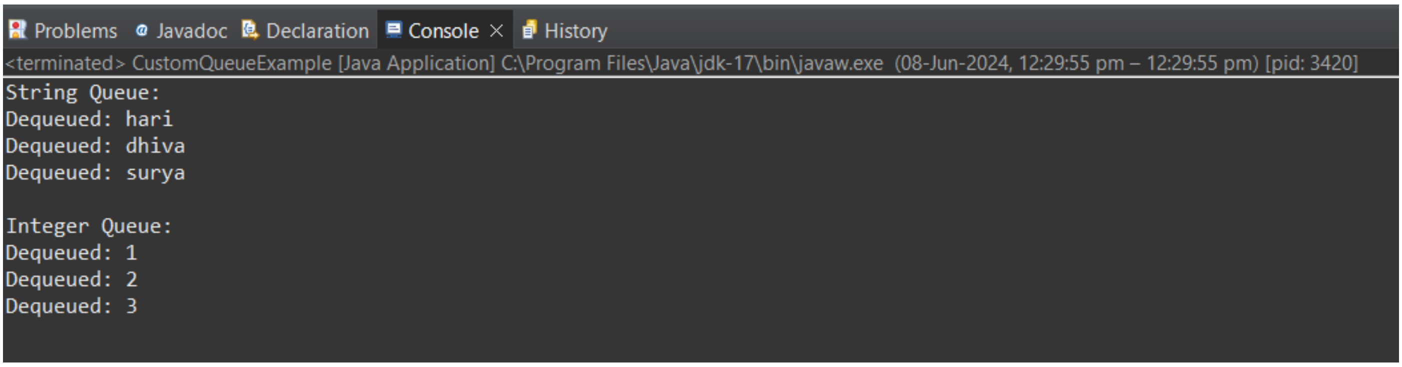


```java
package assignment.day_05;

import java.util.LinkedList;

class CustomQueue<T> {
    private LinkedList<T> queue;

    public CustomQueue() {
        queue = new LinkedList<>();
    }

    public void enqueue(T element) {
        queue.addLast(element);
    }

    public T dequeue() {
        if (isEmpty()) {
            System.out.println("Queue underflow");
            return null;
        }
        return queue.removeFirst();
    }

    public T peek() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return null;
        }
        return queue.getFirst();
    }

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

```



```java

public class CustomQueueExample {
    public static void main(String[] args) {
        CustomQueue<String> stringQueue = new CustomQueue<>();
        CustomQueue<Integer> intQueue = new CustomQueue<>();

        // Enqueue strings
        stringQueue.enqueue("hari");
        stringQueue.enqueue("dhiva");
        stringQueue.enqueue("surya");

        // Enqueue integers
        intQueue.enqueue(1);
        intQueue.enqueue(2);
        intQueue.enqueue(3);

        // Dequeue and display strings
        System.out.println("String Queue:");
        while (!stringQueue.isEmpty()) {
            System.out.println("Dequeued: " + stringQueue.dequeue());
        }

        // Dequeue and display integers
        System.out.println("\nInteger Queue:");
        while (!intQueue.isEmpty()) {
            System.out.println("Dequeued: " + intQueue.dequeue());
        }
    }
}

```

Output:



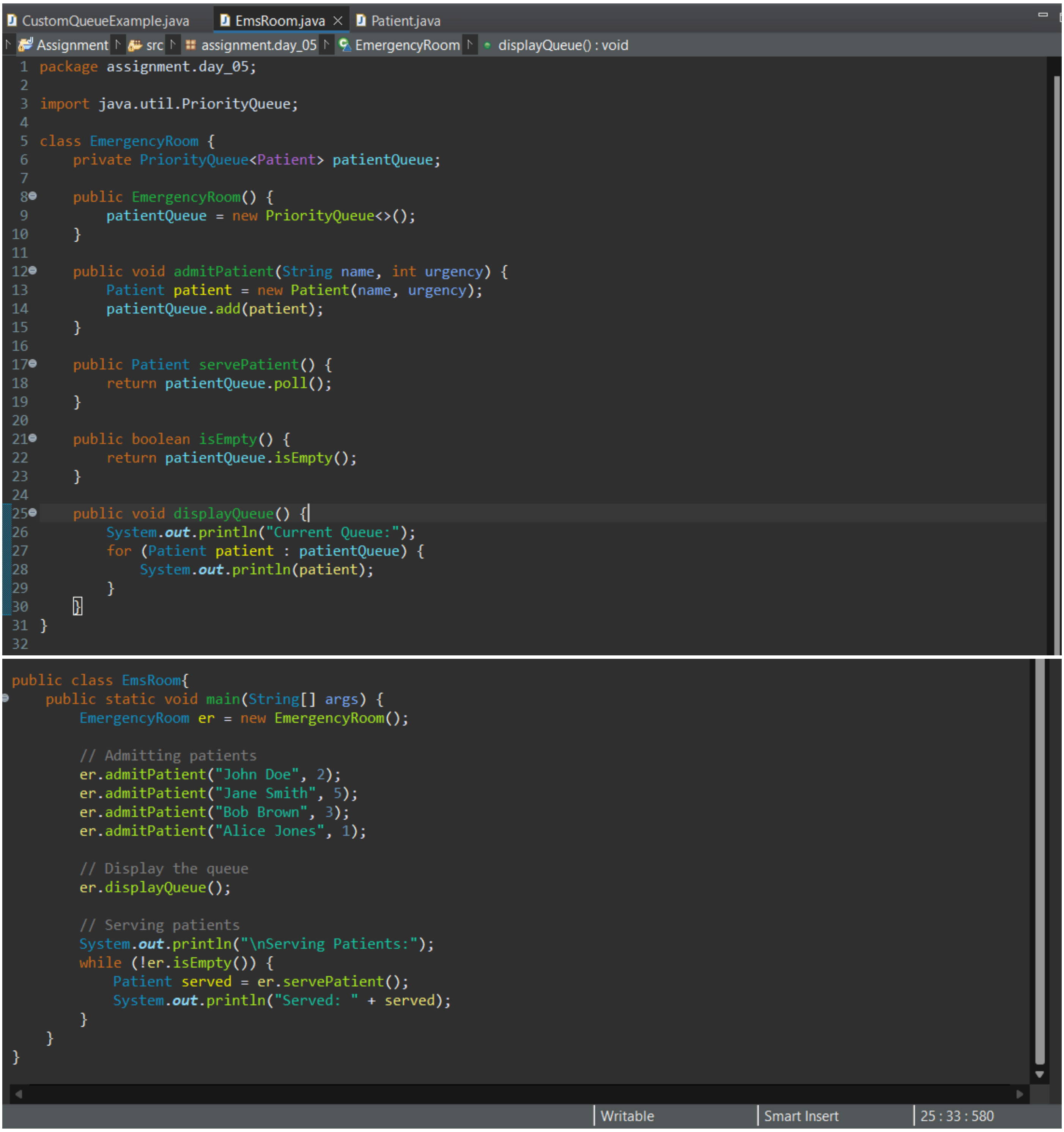
```
<terminated> CustomQueueExample [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (08-Jun-2024, 12:29:55 pm – 12:29:55 pm) [pid: 3420]
String Queue:
Dequeued: hari
Dequeued: dhiva
Dequeued: surya

Integer Queue:
Dequeued: 1
Dequeued: 2
Dequeued: 3
```

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.

Code:

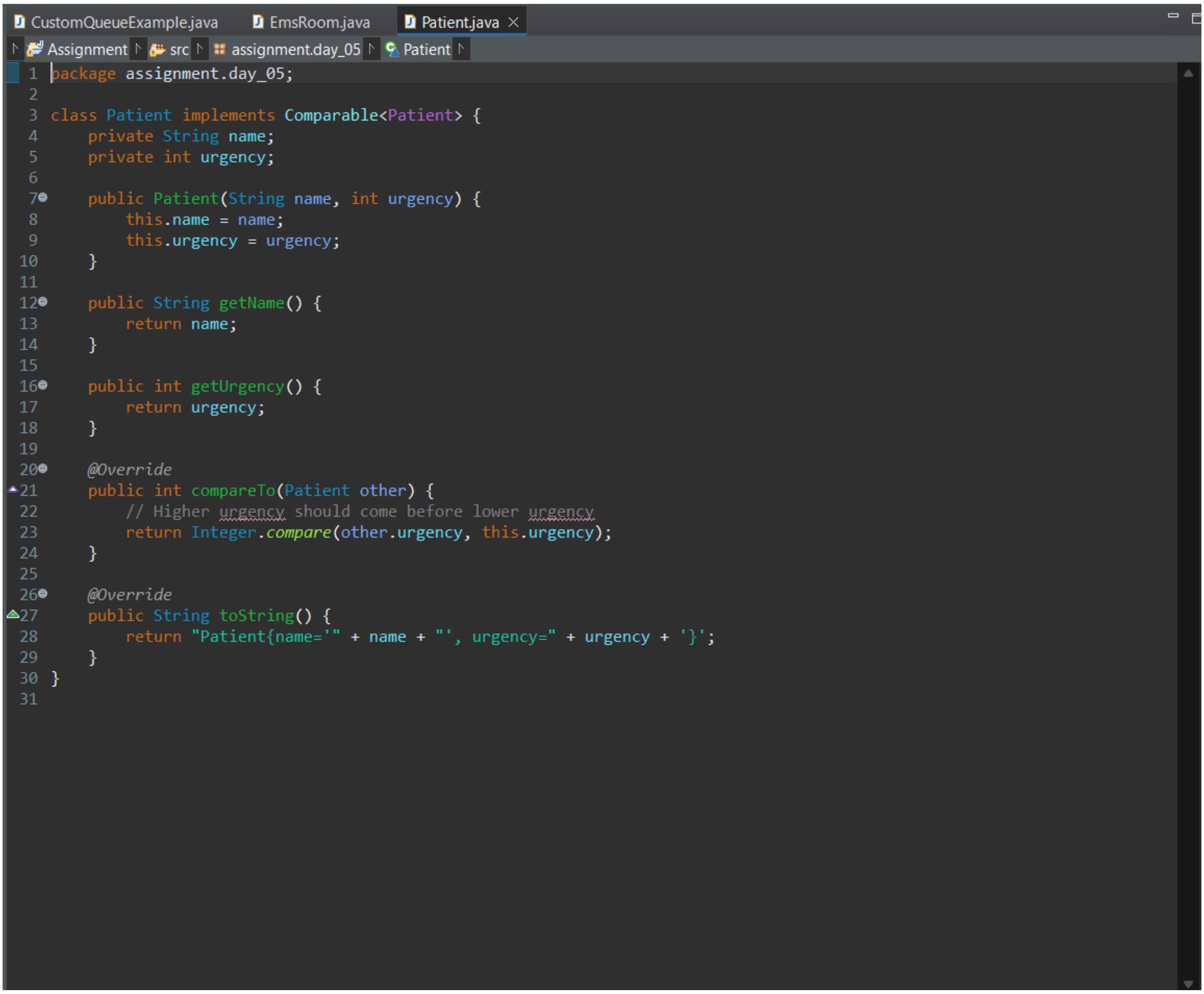
Code for EMS Room:



```java
package assignment.day_05;

import java.util.PriorityQueue;

class EmergencyRoom {
    private PriorityQueue<Patient> patientQueue;

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

    public void admitPatient(String name, int urgency) {
        Patient patient = new Patient(name, urgency);
        patientQueue.add(patient);
    }

    public Patient servePatient() {
        return patientQueue.poll();
    }

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

    public void displayQueue() {
        System.out.println("Current Queue:");
        for (Patient patient : patientQueue) {
            System.out.println(patient);
        }
    }
}

```

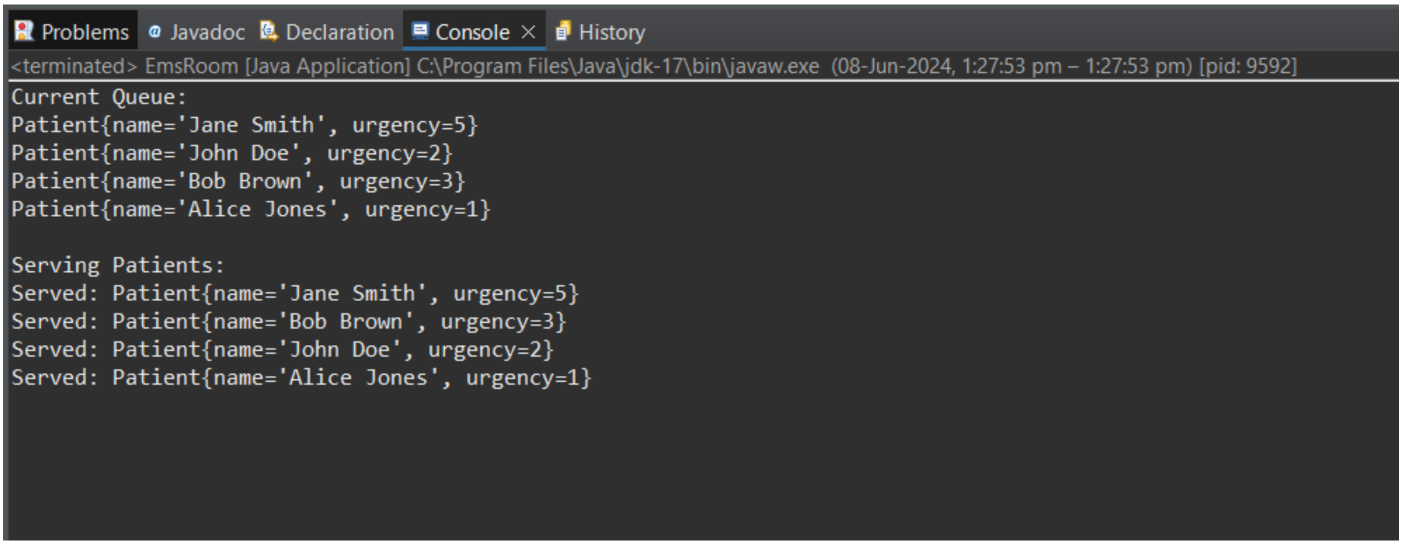
```java
public class EmsRoom{
    public static void main(String[] args) {
        EmergencyRoom er = new EmergencyRoom();

        // Admitting patients
        er.admitPatient("John Doe", 2);
        er.admitPatient("Jane Smith", 5);
        er.admitPatient("Bob Brown", 3);
        er.admitPatient("Alice Jones", 1);

        // Display the queue
        er.displayQueue();

        // Serving patients
        System.out.println("\nServing Patients:");
        while (!er.isEmpty()) {
            Patient served = er.servePatient();
            System.out.println("Served: " + served);
        }
    }
}
```

Code for Patient:



```java
package assignment.day_05;

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

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

    public String getName() {
        return name;
    }

    public int getUrgency() {
        return urgency;
    }

    @Override
    public int compareTo(Patient other) {
        // Higher urgency should come before lower urgency
        return Integer.compare(other.urgency, this.urgency);
    }

    @Override
    public String toString() {
        return "Patient{name='" + name + "', urgency=" + urgency + '}';
    }
}
```

Output:



```
<terminated> EmsRoom [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (08-Jun-2024, 1:27:53 pm – 1:27:53 pm) [pid: 9592]
Current Queue:
Patient{name='Jane Smith', urgency=5}
Patient{name='John Doe', urgency=2}
Patient{name='Bob Brown', urgency=3}
Patient{name='Alice Jones', urgency=1}

Serving Patients:
Served: Patient{name='Jane Smith', urgency=5}
Served: Patient{name='Bob Brown', urgency=3}
Served: Patient{name='John Doe', urgency=2}
Served: Patient{name='Alice Jones', urgency=1}
```