Digvijay Thakare(digvijaythakare2017@gmail.com)

Day 5 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

```
package WiproEP;
    class Node {
        int data;
        Node next;
        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    class CustomLinkedList {
        Node head;
        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 last = head;
            while (last.next != null) {
```

```
                last = last.next;
            }
            last.next = newNode;
        }
        public void insertAtPosition(int position, int data) {
            if (position == 0) {
                insertAtBeginning(data);
                return;
            }
            Node newNode = new Node(data);
            Node current = head;
            for (int i = 0; i < position - 1; i++) {
                if (current == null) {
                    throw new
IndexOutOfBoundsException("Position out of bounds");
                }
                current = current.next;
            }
            if (current == null) {
                throw new IndexOutOfBoundsException("Position
out of bounds");
            }
            newNode.next = current.next;
            current.next = newNode;
        }
        public void deleteNode(int key) {
            Node temp = head, prev = null;
            if (temp != null && temp.data == key) {
                head = temp.next;
                return;
            }
            while (temp != null && temp.data != key) {
                prev = temp;
                temp = temp.next;
            }
            if (temp == null) return;
            prev.next = temp.next;
        }
        public void updateNode(int oldData, int newData) {
            Node current = head;
            while (current != null) {
```

```
                if (current.data == oldData) {
                    current.data = newData;
                    return;
                }
                current = current.next;
            }
            throw new IllegalArgumentException("Node with data
" + oldData + " 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 ll = new CustomLinkedList();
            // Insertions
            ll.insertAtBeginning(10);
            ll.insertAtEnd(20);
            ll.insertAtPosition(1, 15);
            ll.insertAtPosition(0, 5);
            ll.insertAtEnd(25);
            // Display after insertions
            System.out.print("List after insertions: ");
            ll.displayAllNodes();
            // Deletion
            ll.deleteNode(15);
            // Display after deletion
            System.out.print("List after deletion of 15: ");
            ll.displayAllNodes();
            // Update
            ll.updateNode(10, 100);
            // Display after update
            System.out.print("List after updating 10 to 100:
");
            ll.displayAllNodes();
        }
    }
```

Output

```
List after insertions: 5 10 15 20 25
List after deletion of 15: 5 10 20 25
List after updating 10 to 100: 5 100 20 25
```

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 WiproEP;
import java.util.LinkedList;
class CustomStack {
    private LinkedList<Integer> stack;
    public CustomStack() {
        stack = new LinkedList<>();
    }
    public void push(int data) {
        stack.addFirst(data);
    }
    public int pop() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return stack.removeFirst();
    }
    public int peek() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return stack.getFirst();
    }
    public boolean isEmpty() {
        return stack.isEmpty();
```

```
    }
    public void displayStack() {
        System.out.println(stack);
    }
    public static void main(String[] args) {
        CustomStack stack = new CustomStack();
        // Push integers onto the stack
        stack.push(10);
        stack.push(20);
        stack.push(30);
        // Display stack after pushing elements
        System.out.println("Stack after pushing elements:");
        stack.displayStack();
        // Display top element using peek
        System.out.println("Top element (peek): " +
stack.peek());
        // Demonstrate LIFO behavior by popping elements
        System.out.println("Popping stack elements (LIFO
order):");
        while (!stack.isEmpty()) {
            System.out.println(stack.pop());
            // Display stack after each pop
            System.out.println("Stack after popping an
element:");
            stack.displayStack();
        }
    }
}
```

Output

```
Stack after pushing elements:
[30, 20, 10]
Top element (peek): 30
Popping stack elements (LIFO order):
30
Stack after popping an element:
[20, 10]
20
Stack after popping an element:
[10]
```

```
10
Stack after popping an element:
[]
```

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

```
package WiproEP;
import java.util.LinkedList;
class CustomQueue<T> {
    private LinkedList<T> queue;
    public CustomQueue() {
        queue = new LinkedList<>();
    }
    public void enqueue(T data) {
        queue.addLast(data);
    }
    public T dequeue() {
        if (isEmpty()) {
            throw new IllegalStateException("Queue is empty");
        }
        return queue.removeFirst();
    }
    public T peek() {
        if (isEmpty()) {
            throw new IllegalStateException("Queue is empty");
        }
        return queue.getFirst();
    }
    public boolean isEmpty() {
        return queue.isEmpty();
    }
    public void displayQueue() {
        System.out.println(queue);
    }
    public static void main(String[] args) {
        CustomQueue<Object> queue = new CustomQueue<>();
```

```
        // Enqueue different data types
        queue.enqueue(10);
        queue.enqueue("Hello");
        queue.enqueue(20);
        queue.enqueue("World");
        // Display queue after enqueueing elements
        System.out.println("Queue after enqueueing elements:");
        queue.displayQueue();
        // Display front element using peek
        System.out.println("Front element (peek): " +
queue.peek());
        // Demonstrate FIFO behavior by dequeuing elements
        System.out.println("Dequeuing queue elements (FIFO
order):");
        while (!queue.isEmpty()) {
            System.out.println(queue.dequeue());
            // Display queue after each dequeue
            System.out.println("Queue after dequeuing an
element:");
            queue.displayQueue();
        }
    }
}
```

Output

```
Queue after enqueueing elements:
[10, Hello, 20, World]
Front element (peek): 10
Dequeuing queue elements (FIFO order):
10
Queue after dequeuing an element:
[Hello, 20, World]
Hello
Queue after dequeuing an element:
[20, World]
20
Queue after dequeuing an element:
[World]
World
Queue after dequeuing an element:
[]
```

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

```
package WiproEP;
import java.util.Comparator;
import java.util.PriorityQueue;
public class EmergencyRoom {
   private static class Patient {
       private String name;
       private int urgencyLevel;
       public Patient(String name, int urgencyLevel) {
           this.name = name;
           this.urgencyLevel = urgencyLevel;
       }
       public String getName() {
           return name;
       }
       public int getUrgencyLevel() {
           return urgencyLevel;
       }
       @Override
       public String toString() {
           return "Patient{name='" + name + "', urgencyLevel=" +
urgencyLevel + '}';
       }
   }
   private PriorityQueue<Patient> patientQueue;
   public EmergencyRoom() {
```

```
        patientQueue = new
PriorityQueue<>(Comparator.comparingInt(Patient::getUrgencyLevel
).reversed());
    }
    public void admitPatient(String name, int urgencyLevel) {
        patientQueue.offer(new Patient(name, urgencyLevel));
    }
    public Patient serveNextPatient() {
        if (patientQueue.isEmpty()) {
            throw new IllegalStateException("No patients in the
queue");
        }
        return patientQueue.poll();
    }
    public Patient peekNextPatient() {
        if (patientQueue.isEmpty()) {
            throw new IllegalStateException("No patients in the
queue");
        }
        return patientQueue.peek();
    }
    public boolean isEmpty() {
        return patientQueue.isEmpty();
    }
    public static void main(String[] args) {
        EmergencyRoom er = new EmergencyRoom();
        // Admit patients with varying urgency levels
        er.admitPatient("John Doe", 5);
        er.admitPatient("Jane Smith", 10);
        er.admitPatient("Alice Jones", 2);
        er.admitPatient("Bob Brown", 8);
        // Display the next patient to be served
        System.out.println("Next patient to be served: " +
er.peekNextPatient());
        // Serve patients in order of urgency
        System.out.println("Serving patients in order of
urgency:");
        while (!er.isEmpty()) {
            System.out.println("Serving: " +
er.serveNextPatient());
        }
```

```
    }
}
```

Output

```
Next patient to be served: Patient{name='Jane Smith',
urgencyLevel=10}
Serving patients in order of urgency:
Serving: Patient{name='Jane Smith', urgencyLevel=10}
Serving: Patient{name='Bob Brown', urgencyLevel=8}
Serving: Patient{name='John Doe', urgencyLevel=5}
Serving: Patient{name='Alice Jones', urgencyLevel=2}
```