Assignment Day-5

Core Java with DS and Algorithms

Name: Joshnitha Rangolu

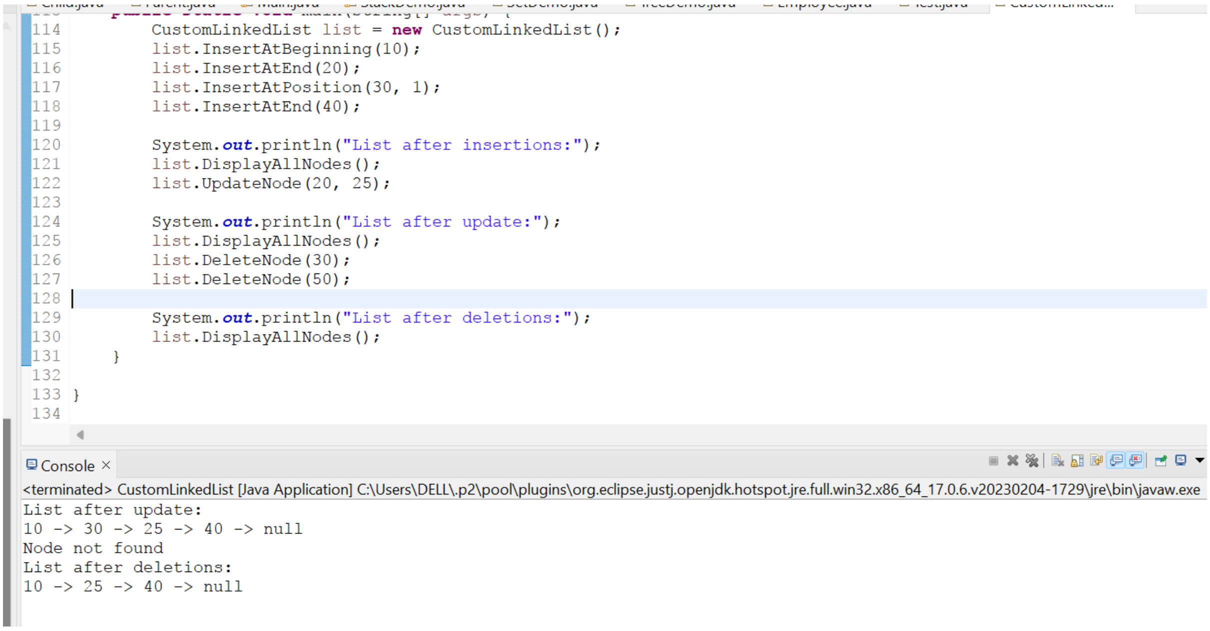
Task 1: Implementing a Linked List

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.**

```
package day_5;

public class CustomLinkedList {

private Node head;

private Node tail;

private static class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

}

}

public void InsertAtBeginning(int data) {

Node newNode = new Node(data);

newNode.next = head;

head = newNode;

if (tail == null) {

tail = head;

}
```

```java
}

public void InsertAtEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = tail = newNode;

return;

}

tail.next = newNode;

tail = newNode;

}

public void InsertAtPosition(int data, int position) {

if (position < 0) {

System.out.println("Invalid position: cannot insert before head");

return;

}

if (position == 0) {

InsertAtBeginning(data);

return;

}

Node newNode = new Node(data);

Node current = head;

int counter = 0;

while (current != null && counter < position - 1) {

current = current.next;

counter++;

}

if (current == null) {

System.out.println("Invalid position: position beyond list size");
return;

}

newNode.next = current.next;

current.next = newNode;

}

public void DeleteNode(int data) {

if (head == null) {

return;

}

if (head.data == data) {

head = head.next;

if (head == null) {

tail = null;

}

return;

}

Node current = head;

while (current.next != null && current.next.data != data) {

current = current.next;

}

if (current.next == null) {

System.out.println("Node not found");

return;

}

current.next = current.next.next;

if (current.next == null) {

tail = current;

}
```

```
}
public void UpdateNode(int oldData, int newData) {
Node current = head;
while (current != null && current.data != oldData) {
current = current.next;
}
if (current == null) {
System.out.println("Node not found");
return;
}
current.data = newData;
}
public void DisplayAllNodes() {
Node current = head;
while (current != null) {
System.out.print(current.data + " -> ");
current = current.next;
}
System.out.println("null");
}
public static void main(String[] args) {
CustomLinkedList list = new CustomLinkedList();
list.InsertAtBeginning(10);
list.InsertAtEnd(20);
list.InsertAtPosition(30, 1);
list.InsertAtEnd(40);
System.out.println("List after insertions:");
list.DisplayAllNodes();
list.UpdateNode(20, 25);

System.out.println("List after update:");

list.DisplayAllNodes();

list.DeleteNode(30);

list.DeleteNode(50);

System.out.println("List after deletions:");

list.DisplayAllNodes();

}

}
```



```
        CustomLinkedList list = new CustomLinkedList();
        list.InsertAtBeginning(10);
        list.InsertAtEnd(20);
        list.InsertAtPosition(30, 1);
        list.InsertAtEnd(40);

        System.out.println("List after insertions:");
        list.DisplayAllNodes();
        list.UpdateNode(20, 25);

        System.out.println("List after update:");
        list.DisplayAllNodes();
        list.DeleteNode(30);
        list.DeleteNode(50);

        System.out.println("List after deletions:");
        list.DisplayAllNodes();
    }

}
```

```
<terminated> CustomLinkedList [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.exe
List after update:
10 -> 30 -> 25 -> 40 -> null
Node not found
List after deletions:
10 -> 25 -> 40 -> null
```

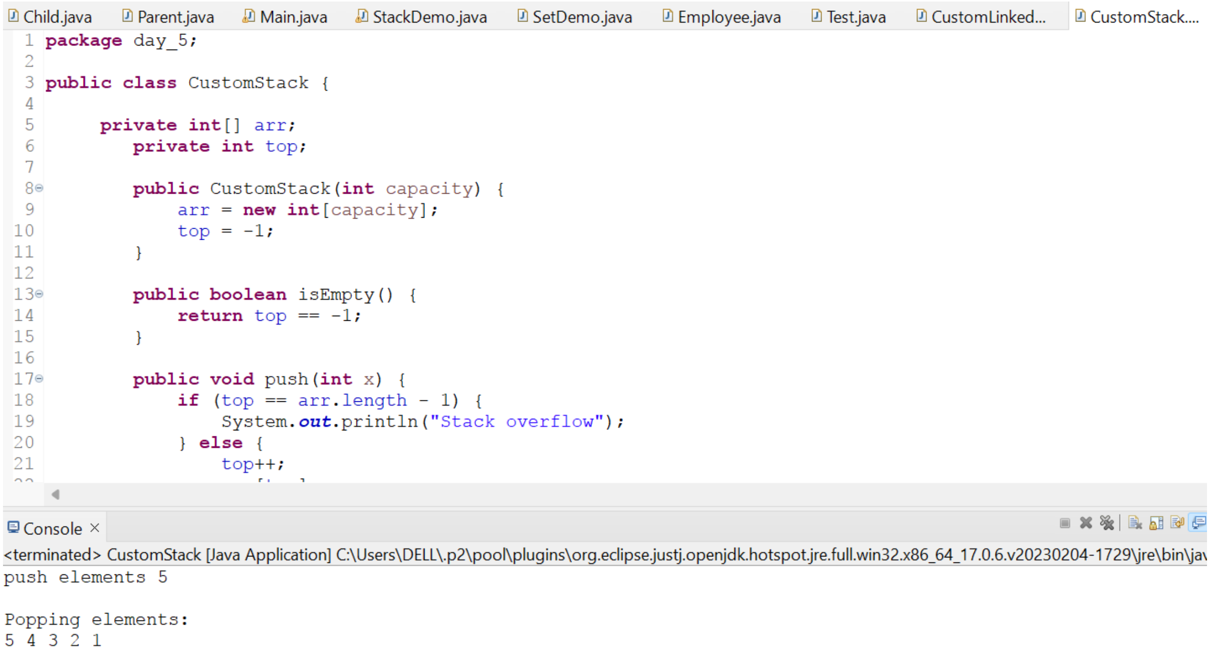
Task 2: Stack and Queue Operations

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.**

```
package day_5;

public class CustomStack {

private int[] arr;

private int top;
```

```
public CustomStack(int capacity) {

arr = new int[capacity];

top = -1;

}

public boolean isEmpty() {

return top == -1;

}

public void push(int x) {

if (top == arr.length - 1) {

System.out.println("Stack overflow");

} else {

top++;

arr[top] = x;

}

}

public int pop() {

if (isEmpty()) {

System.out.println("Stack underflow");

return -1;

} else {

int x = arr[top];

top--;

return x;

}

}

public int peek() {

if (isEmpty()) {

System.out.println("Stack is empty");
```

```java
return -1;
} else {
return arr[top];
}
}
// Driver program to test methods
public static void main(String[] args) {
CustomStack stack = new CustomStack(5);
stack.push(1);
stack.push(2);
stack.push(3);
stack.push(4);
stack.push(5);
System.out.println("push elements "+stack.peek());
System.out.println("\nPopping elements:");
while (!stack.isEmpty()) {
System.out.print(stack.pop() + " ");
}
}
}
```



```
package day_5;

public class CustomStack {

    private int[] arr;
       private int top;

       public CustomStack(int capacity) {
           arr = new int[capacity];
           top = -1;
       }

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

       public void push(int x) {
           if (top == arr.length - 1) {
               System.out.println("Stack overflow");
           } else {
               top++;
```

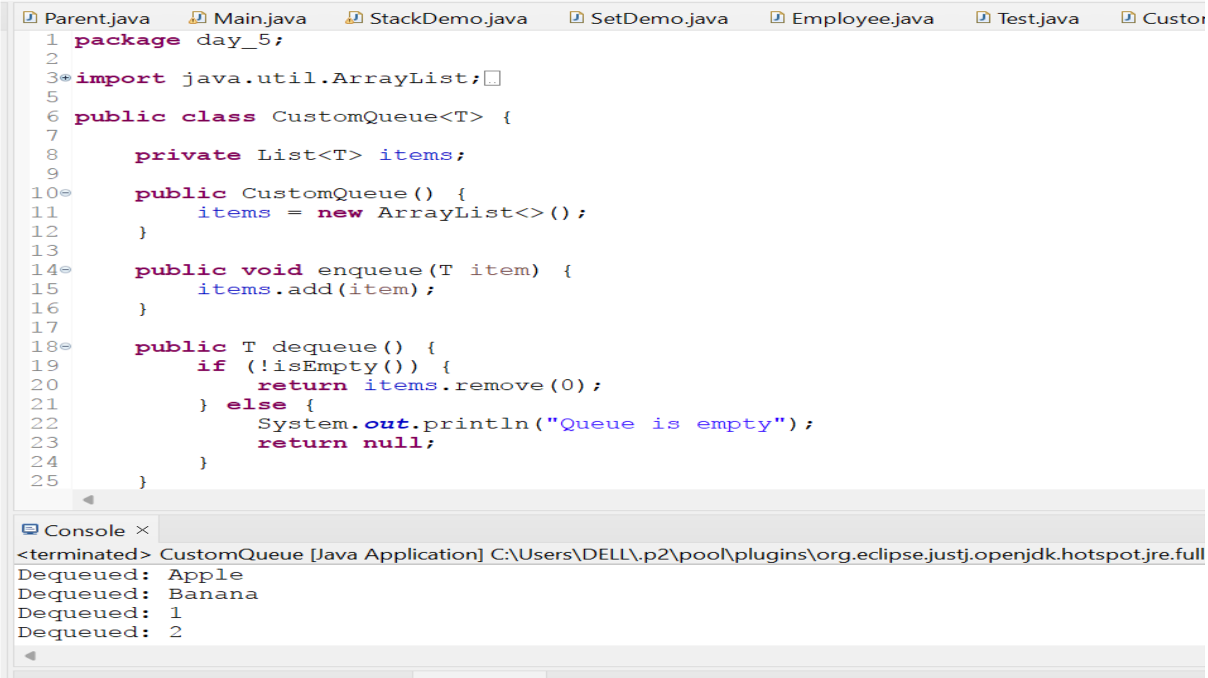
```
<terminated> CustomStack [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\jav
push elements 5

Popping elements:
5 4 3 2 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.

```
package day_5;


import java.util.ArrayList;

import java.util.List;


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();
}

// Example usage
public static void main(String[] args) {
    // Create a queue object
    CustomQueue<Object> queue = new CustomQueue<>();
```

```
        // Enqueue some strings and integers

        queue.enqueue("Apple");

        queue.enqueue("Banana");

        queue.enqueue(1);

        queue.enqueue(2);


        // Dequeue and display items to confirm FIFO order

        while (!queue.isEmpty()) {

            System.out.println("Dequeued: " + queue.dequeue());

        }

    }


}
```



```
package day_5;

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;
        }
    }
```

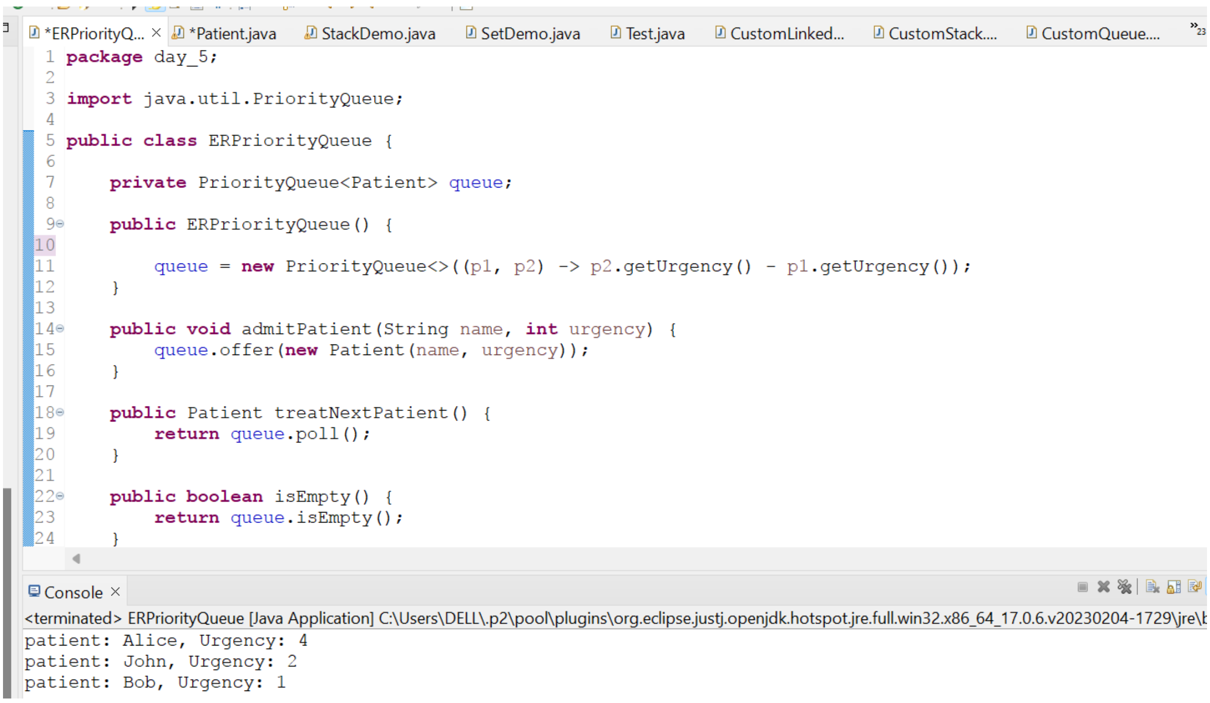
```
<terminated> CustomQueue [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
Dequeued: Apple
Dequeued: Banana
Dequeued: 1
Dequeued: 2
```

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.**

```
package day_5;
import java.util.PriorityQueue;
public class 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;
}
}
public class ERPriorityQueue {
private PriorityQueue<Patient> queue;
public ERPriorityQueue() {
queue = new PriorityQueue<>((p1, p2) -> p2.getUrgency() - p1.getUrgency());
}
public void admitPatient(String name, int urgency) {
queue.offer(new Patient(name, urgency));
}
public Patient treatNextPatient() {
return queue.poll();
}
public boolean isEmpty() {
```

```
return queue.isEmpty();

}

public static void main(String[] args) {

ERPriorityQueue emergencyRoom = new ERPriorityQueue();

emergencyRoom.admitPatient("John", 2);

emergencyRoom.admitPatient("Alice", 4);

emergencyRoom.admitPatient("Bob", 1);

while (!emergencyRoom.isEmpty()) {

Patient nextPatient = emergencyRoom.treatNextPatient();

System.out.println( "patient: " + nextPatient.getName() + ", Urgency: " +
nextPatient.getUrgency());

}

}

}
```



```
package day_5;

import java.util.PriorityQueue;

public class ERPriorityQueue {

    private PriorityQueue<Patient> queue;

    public ERPriorityQueue() {

        queue = new PriorityQueue<>((p1, p2) -> p2.getUrgency() - p1.getUrgency());
    }

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

    public Patient treatNextPatient() {
        return queue.poll();
    }

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

<terminated> ERPriorityQueue [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\b

```
patient: Alice, Urgency: 4
patient: John, Urgency: 2
patient: Bob, Urgency: 1
```