

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;

        }

    }

}
```

```
}

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);

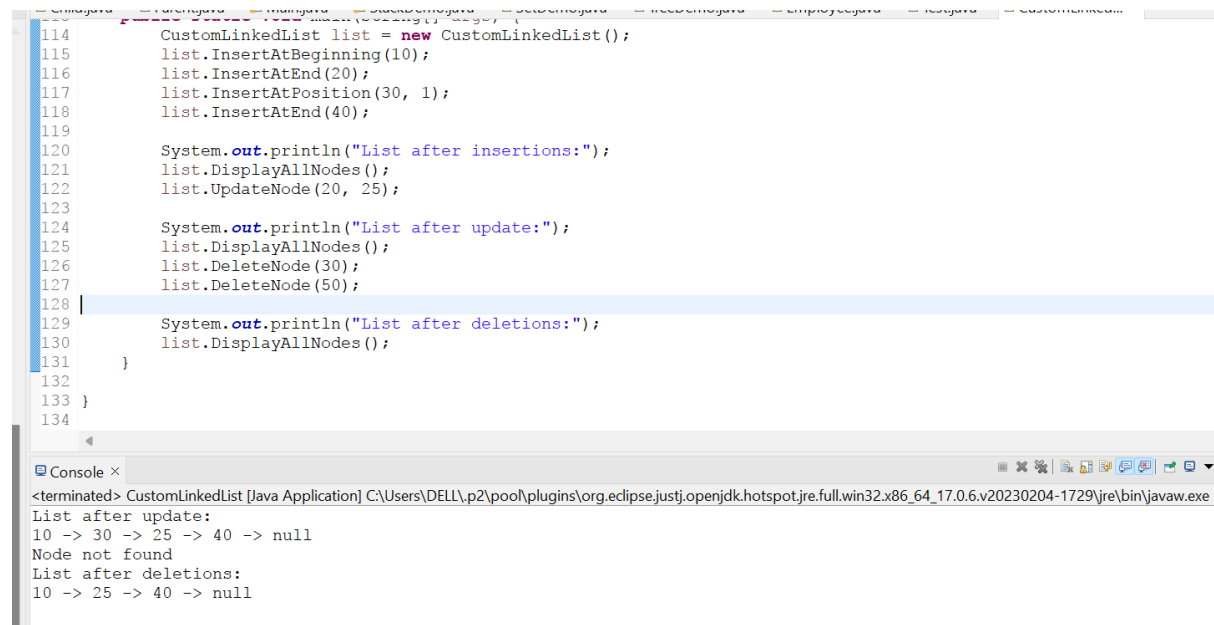
System.out.println("List after deletions:");

list.DisplayAllNodes();

}

}

```



The screenshot shows an Eclipse IDE with a Java file named CustomLinkedList.java. The code implements a linked list with methods for insertion, update, and deletion. The console output shows the state of the list after various operations.

```

114 CustomLinkedList list = new CustomLinkedList();
115 list.InsertAtBeginning(10);
116 list.InsertAtEnd(20);
117 list.InsertAtPosition(30, 1);
118 list.InsertAtEnd(40);
119
120 System.out.println("List after insertions:");
121 list.DisplayAllNodes();
122 list.UpdateNode(20, 25);
123
124 System.out.println("List after update:");
125 list.DisplayAllNodes();
126 list.DeleteNode(30);
127 list.DeleteNode(50);
128
129 System.out.println("List after deletions:");
130 list.DisplayAllNodes();
131 }
132
133 }
134

```

Console Output:

```

<terminated> CustomLinkedList [Java Application] C:\Users\DELL\p2\pool\plugins\org.eclipse.justi.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");

    }

}
```

```
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() + " ");

}

}

}
```

```
Child.java Parent.java Main.java StackDemo.java SetDemo.java Employee.java Test.java CustomLinked... CustomStack....
1 package day_5;
2
3 public class CustomStack {
4
5     private int[] arr;
6     private int top;
7
8     public CustomStack(int capacity) {
9         arr = new int[capacity];
10        top = -1;
11    }
12
13    public boolean isEmpty() {
14        return top == -1;
15    }
16
17    public void push(int x) {
18        if (top == arr.length - 1) {
19            System.out.println("Stack overflow");
20        } else {
21            top++;
22            arr[top] = x;
23        }
24    }
25
26    public void pop() {
27        if (!isEmpty()) {
28            top--;
29        }
30    }
31
32    public int peek() {
33        if (!isEmpty()) {
34            return arr[top];
35        }
36        return -1;
37    }
38
39    public void clear() {
40        top = -1;
41    }
42
43 }
44
45 // Test the CustomStack
46 public class Main {
47     public static void main(String[] args) {
48         CustomStack stack = new CustomStack(5);
49         stack.push(1);
50         stack.push(2);
51         stack.push(3);
52         stack.push(4);
53         stack.push(5);
54         System.out.println("Popping elements:");
55         while (!stack.isEmpty()) {
56             System.out.print(stack.pop() + " ");
57         }
58         System.out.println();
59     }
60 }
```

Console ×

<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 enqueueing 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()) {

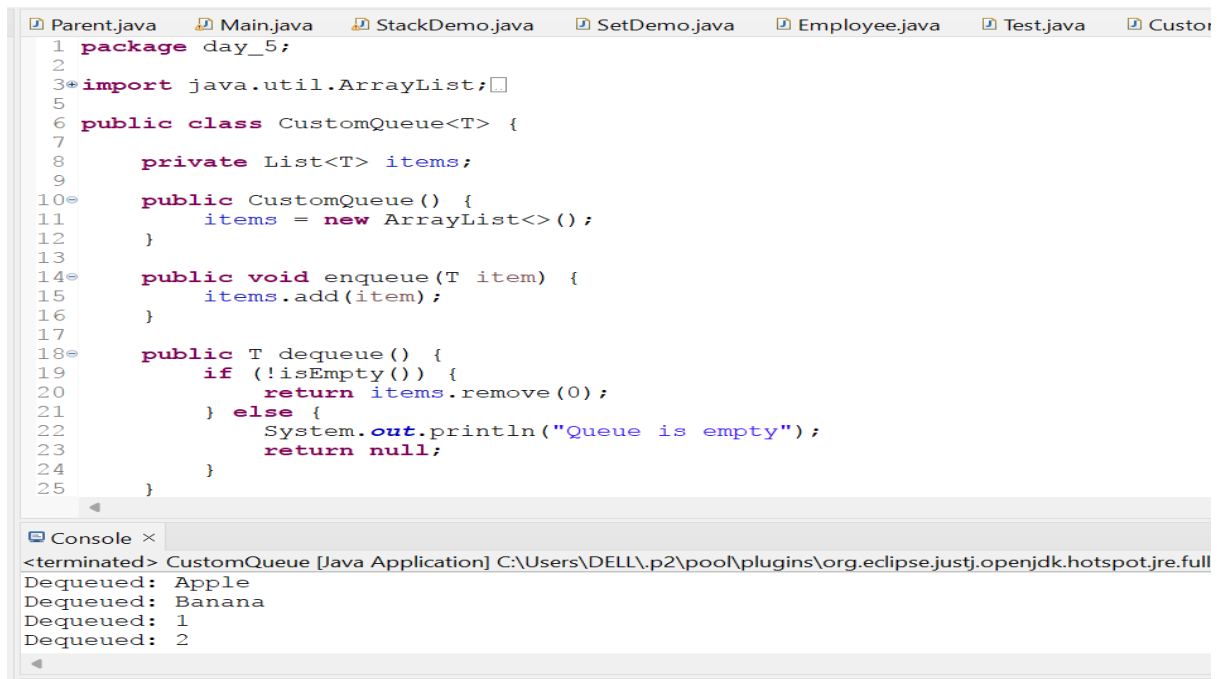
    System.out.println("Dequeued: " + queue.dequeue());

}

}

}

```



The screenshot shows an IDE with a tab for 'CustomQueue.java' selected. The code defines a 'CustomQueue' class using an 'ArrayList' to implement a queue. The 'enqueue' method adds items to the end of the list, and the 'dequeue' method removes items from the front. The console output at the bottom shows the results of running the application: 'Dequeued: Apple', 'Dequeued: Banana', 'Dequeued: 1', and 'Dequeued: 2', confirming the FIFO order.

```

1 package day_5;
2
3 import java.util.ArrayList;
4
5
6 public class CustomQueue<T> {
7
8     private List<T> items;
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 }

```

Console Output:

```

<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();

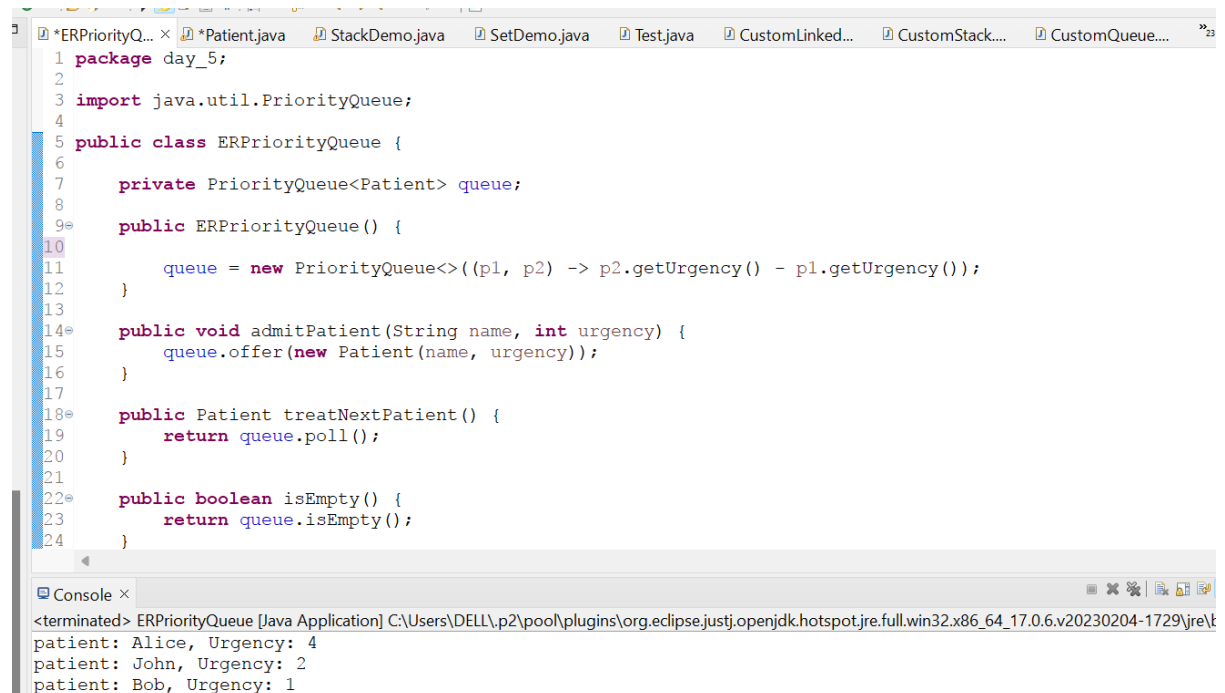
System.out.println( "patient: " + nextPatient.getName() + ", Urgency: " +
nextPatient.getUrgency());

}

}

}

```



The screenshot shows the Eclipse IDE with the `ERPriorityQueue.java` file open. The code defines a class `ERPriorityQueue` that implements a priority queue using `java.util.PriorityQueue`. The `admitPatient` method adds patients to the queue, `treatNextPatient` removes and returns the patient with the highest urgency, and `isEmpty` checks if the queue is empty. The `main` method in `StackDemo.java` (partially visible) demonstrates the usage by admitting patients John, Alice, and Bob with urgencies 2, 4, and 1 respectively, and then treating them in order of urgency.

```

1 package day_5;
2
3 import java.util.PriorityQueue;
4
5 public class ERPriorityQueue {
6
7     private PriorityQueue<Patient> queue;
8
9     public ERPriorityQueue() {
10
11         queue = new PriorityQueue<>((p1, p2) -> p2.getUrgency() - p1.getUrgency());
12     }
13
14     public void admitPatient(String name, int urgency) {
15         queue.offer(new Patient(name, urgency));
16     }
17
18     public Patient treatNextPatient() {
19         return queue.poll();
20     }
21
22     public boolean isEmpty() {
23         return queue.isEmpty();
24     }
25 }

```

The console output shows the execution results:

```

<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\l
patient: Alice, Urgency: 4
patient: John, Urgency: 2
patient: Bob, Urgency: 1

```