

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.

Program:

```
package Assignments.Day5;
```

```
/*
```

```
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.
```

```
*/
```

```
public class Task1 {  
    public static void main(String[] args) {  
        Task1 list = new Task1();  
        insertAtEnd(list, 1);  
        insertAtEnd(list, 2);  
        insertAtBeginning(list, 0);  
        insertAtBeginning(list, -1);  
        insertAtPosition(list, 7, 2);  
        insertAtPosition(list, 3, 5);  
        DisplayAllNodes(list);  
        deleteNode(list, 7);  
        deleteNode(list, -1);  
        updateNode(list, 3, 4);  
        updateNode(list, 2, 3);  
        insertAtEnd(list, 5);  
        insertAtPosition(list, 2, 2);  
        DisplayAllNodes(list);  
    }  
}
```

```
Node head;
```

```
static class Node {
```

```

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

private static void insertAtBeginning(Task1 list, int data){
    Node new_node = new Node(data);
    if(list.head == null){
        list.head = new_node;
    }else{
        Node temp = list.head;
        list.head = new_node;
        new_node.next = temp;
    }
}

private static void insertAtPosition(Task1 list, int data, int position){
    Node new_node = new Node(data);
    Node cur = list.head;
    for (int i = 0; i < position-1; i++) {
        cur = cur.next;
    }
    Node temp = cur.next;
    cur.next = new_node;
    new_node.next = temp;
}

private static void deleteNode(Task1 list, int data) {
    Node cur = list.head;
    if(list.head.data == data){
        list.head = list.head.next;
    }else {
        while (cur.next.data != data) {
            cur = cur.next;
        }
        cur.next = cur.next.next;
    }
}

private static void insertAtEnd(Task1 list, int data){
    Node new_node = new Node(data);

```

```

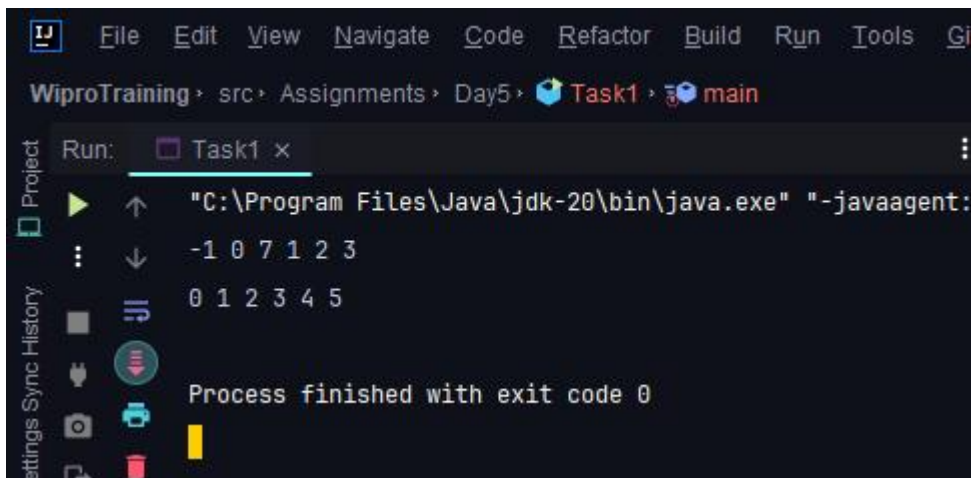
        if(list.head == null){
            list.head = new _node;
        }else {
            Node last = list.head;
            while (last.next != null){
                last = last.next;
            }
            last.next = new _node;
        }
    }
}

private static void updateNode(Task1 list, int data, int newData) {
    Node cur = list.head;
    while (cur.next.data != data ){
        cur = cur.next;
    }
    cur.next.data = newData;
}

private static void DisplayAllNodes(Task1 list){
    Node cur = list.head;
    while (cur != null){
        System.out.print(cur.data+ " ");
        cur = cur.next;
    }
    System.out.println();
}
}
}

```

Output:



```

WiproTraining > src > Assignments > Day5 > Task1 > main
Run: Task1 x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:
-1 0 7 1 2 3
0 1 2 3 4 5
Process finished with exit code 0

```

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.

Program:

```
package Assignments.Day5;

import java.awt.*;
import java.util.LinkedList;
import java.util.Queue;
public class Task2a {

    Queue<Integer> q;
    public Task2a() {
        q = new LinkedList<>();
    }

    private void push(int x) {
        q.add(x);
        int n = q.size();
        for(int i = 1;i<n;i++){
            q.offer(q.poll());
        }
    }

    private void pop() {
        if(q.isEmpty()){
            System.out.println("No elements");
        }else
            q.poll();
    }

    private void top() {
        if(!q.isEmpty()){
            System.out.println(q.peek());

        }else {
            System.out.println("No elements");
        }
    }

    private void DisplayStack(){
        System.out.println(q);
    }
}
```

```

private boolean empty() {
    return q.isEmpty();
}

public static void main(String[] args) {
    Task2a stack = new Task2a();
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);
    stack.push(5);
    stack.top();
    stack.pop();
    stack.DisplayStack();
}
}

```

Output:



The screenshot shows an IDE window titled 'WiproTraining' with the file path 'src > Assignments > Day5 > Task2a'. The 'Run' tab is active, showing the command: `"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\F`. The output console displays the following sequence of operations: `5`, `[4, 3, 2, 1]`, and `Process finished with exit code 0`.

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.

Program:

```

package Assignments.Day5;

import java.util.Stack;

```

```

public class Task2b {
    Stack<Integer> s1;
    Stack <Integer> s2;
    public Task2b() {
        s1 = new Stack<>();
        s2 = new Stack<>();
    }

    public void add(int x) {
        while(!s1.isEmpty()){
            s2.push(s1.pop());
        }
        s1.push(x);
        while(!s2.isEmpty()){
            s1.push(s2.pop());
        }
    }

    public int pop() {
        return s1.pop();
    }

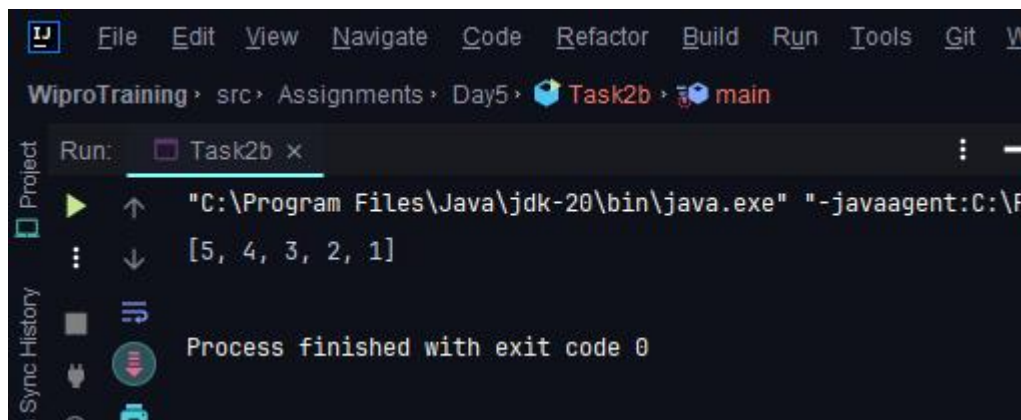
    public int peek() {
        return s1.peek();
    }

    public boolean empty() {
        return s1.isEmpty();
    }
    private void displayQueue(){
        System.out.println(s1);
    }

    public static void main(String[] args) {
        Task2b queue = new Task2b();
        queue.add(1);
        queue.add(2);
        queue.add(3);
        queue.add(4);
        queue.add(5);
        queue.displayQueue();
    }
}

```

Output:



```
WiproTraining > src > Assignments > Day5 > Task2b > main
Run: Task2b x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\F
[5, 4, 3, 2, 1]
Process finished with exit code 0
```

Task 3: Priority Queue Scenario

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

Program:

```
package Assignments.Day5.Task3;

import java.util.PriorityQueue;

public class HospitalTaskScheduling {
    public static void main(String[] args) {
        PriorityQueue<Task> taskQueue = new PriorityQueue<>();

        // Simulate patient arrivals and task scheduling
        Patient patient1 = new Patient("John", 1); // Critical
        Patient patient2 = new Patient("Sarah", 2); // High
        Patient patient3 = new Patient("Mike", 3); // Medium

        taskQueue.add(new Task("Emergency surgery", patient1));
        taskQueue.add(new Task("X-ray and examination", patient2));
        taskQueue.add(new Task("Blood test", patient3));

        // Process tasks in priority order
        while (!taskQueue.isEmpty()) {
            Task task = taskQueue.poll();
            System.out.println("Processing: " + task);
        }
    }
}
```

```

public class Patient {
    private String name;
    private int priority; // 1 for critical, 2 for high, 3 for medium, 4 for low

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

    public int getPriority() {
        return priority;
    }
}

public class Task implements Comparable<Task>{
    private final String description;
    private final Patient patient;

    public Task(String description, Patient patient) {
        this.description = description;
        this.patient = patient;
    }

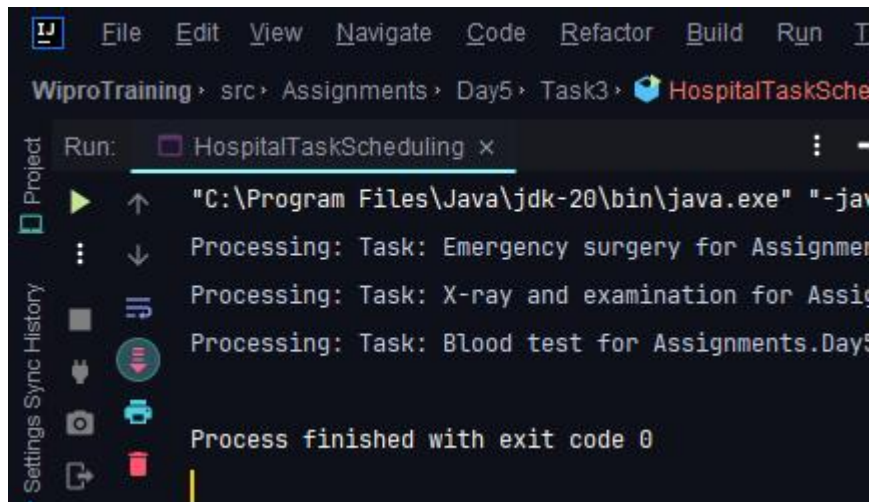
    public Patient getPatient() {
        return patient;
    }

    @Override
    public int compareTo(Task otherTask) {
        // Tasks are compared based on patient priority
        return Integer.compare(this.patient.getPriority(),
otherTask.getPatient().getPriority());
    }

    @Override
    public String toString() {
        return "Task: " + description + " for " + patient;
    }
}

```


Output:



```
WiproTraining > src > Assignments > Day5 > Task3 > HospitalTaskSche

Run: HospitalTaskScheduling x

"C:\Program Files\Java\jdk-20\bin\java.exe" "-jav
Processing: Task: Emergency surgery for Assignmen
Processing: Task: X-ray and examination for Assig
Processing: Task: Blood test for Assignments.Day5
Process finished with exit code 0
```