Object Oriented Analysis and Design using Java
Week 9&10 Lab Assignment

Vanshika Goel	PES1UG20CS484	Section H	Roll No 40

Problem Statement

A Company's Leave Management System has the following features. An Employee (client) can apply for Casual Leave (CL), Sick Leave (SL) and Vacation Leave (VL). The roles in the hierarchy who are responsible for approving or rejecting the leave using the process specified are Director, Project Manager and Tech Lead. The Leave request contains the following details: empName, leaveStatus, approvedBy, requestDate and approvalDate. A CL and SL are for only one day. A VL will have a startDate and endDate. A CL will also need a reason to be specified. The Leave created by the client is assigned a "New" status. If the leave is SL, then it will be processed by Tech Lead, if it is CL, it will be processed by the Project Manager, and if it is VL, will be processed by the Director. The Leave when created is sent to Tech Lead for processing, if it is not SL, the Tech Lead will just pass the request to the next higher level. Similarly, Project Manager will process a CL request or forward the VL request to the next higher level. Once the request is processed, a message should be displayed on the console showing request details and approval details.

Represent the design (using appropriate design patterns) in a UML Class Diagram and implement the same.

Note: Design the application in such a way that extensibility is easy. It should be easy to add new types of Employee and new types of Leave.

Design Patterns Considered

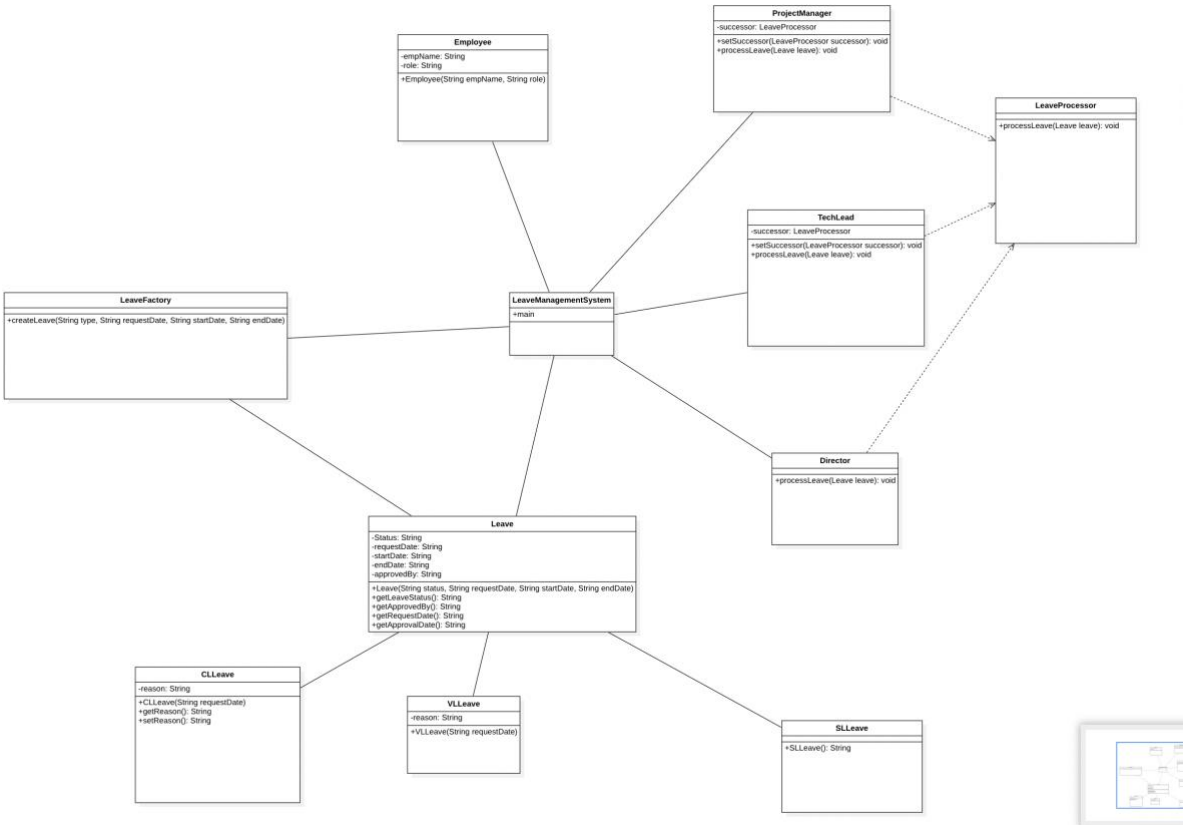
1. Chain of Responsibility Pattern - This pattern can be used to establish a hierarchical chain of objects to process the leave requests. The objects can be the Tech Lead, Project Manager, and Director. The responsibility of processing the leave request can be passed from one object to another based on the type of leave requested.
2. Factory Method Pattern - This pattern can be used to create different types of leaves such as CL, SL, and VL. The Factory Method can create different types of leave objects based on the input given and return the appropriate object.

3. Observer Pattern - This pattern can be used to notify the interested parties about the state changes in the leave request. The Director, Project Manager, and Tech Lead can observe the state changes and take appropriate actions.
4. Template Method Pattern - This pattern can be used to define the basic steps for processing a leave request, while allowing subclasses to override some of the steps to provide custom behavior.

Design Patterns Used

Based on the requirements, the Chain of Responsibility pattern can be used to process the leave requests by passing them through a chain of handlers. The Factory Method pattern can be used to create different types of leave objects.

UML Class Model



Code

```
import java.util.Scanner;

public class LeaveManagementSystem {
   public static void main(String[] args) {
      try (Scanner sc = new Scanner(System.in)) {
         System.out.print("Enter employee name: ");
         String empName = sc.nextLine();

         System.out.print("Enter employee role: ");
         String role = sc.nextLine();

         System.out.print("Enter leave type (CL, SL, VL): ");
         String type = sc.nextLine();

         System.out.print("Enter request date (yyyy-dd-mm): ");
         String requestDate = sc.nextLine();

         System.out.print("Enter start date (yyyy-dd-mm): ");
         String startDate = sc.nextLine();

         System.out.print("Enter end date (yyyy-dd-mm): ");
         String endDate = sc.nextLine();

         Employee employee = new Employee(empName, role);
         Leave leave = LeaveFactory.createLeave(type, requestDate, startDate, endDate);

         // Set up chain of responsibility
         TechLead techLead = new TechLead();
         ProjectManager projectManager = new ProjectManager();
         Director director = new Director();
         techLead.setSuccessor(projectManager);
         projectManager.setSuccessor(director);

         // Process leave
         techLead.processLeave(leave);

         // Display request and approval details
         System.out.println("Leave status: " + leave.getLeaveStatus());
         System.out.println("Approved by: " + leave.getApprovedBy());
         System.out.println("Request date: " + leave.getRequestDate());
         System.out.println("Start date: " + startDate);
         System.out.println("End date: " + endDate);
         System.out.println("Approval date: " + leave.getApprovalDate());
      }
   }
}
```

```
public class CLLeave extends Leave {
    private String reason;

    public CLLeave(String requestDate) {
        super("New", requestDate);
        this.reason = reason;
    }

    public String getReason() {
        return reason;
    }

    public void setReason(String reason) {
        this.reason = reason;
    }
}

public class Director implements LeaveProcessor {
    @Override
    public void processLeave(Leave leave) {
        if (leave instanceof VLLeave) {
            System.out.println("Director processed VLLeave");
            leave.setApprovedBy("Director");
            leave.setLeaveStatus("Approved");
        }
    }
}

public class Employee {
    private String empName;
    private String role;

    public Employee(String empName, String role) {
        this.empName = empName;
        this.role = role;
    }

    public String getEmpName() {
        return empName;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }
}

public class InvalidLeave extends Leave {
```

```
    public InvalidLeave(String requestDate) {
        super("Invalid", requestDate);
    }
}

public class Leave {
    private String leaveStatus;
    private String approvedBy;
    private String requestDate;
    private String startDate;
    private String endDate;
    private String approvalDate;

    public Leave(String leaveStatus, String requestDate) {
        this.leaveStatus = leaveStatus;
        this.requestDate = requestDate;
    }

    public String getLeaveStatus() {
        return leaveStatus;
    }

    public void setLeaveStatus(String leaveStatus) {
        this.leaveStatus = leaveStatus;
    }

    public String getApprovedBy() {
        return approvedBy;
    }

    public void setApprovedBy(String approvedBy) {
        this.approvedBy = approvedBy;
    }

    public String getRequestDate() {
        return requestDate;
    }

    public void setRequestDate(String requestDate) {
        this.requestDate = requestDate;
    }

    public String getStartDate() {
        return startDate;
    }

    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }

    public String getEndDate() {
```

```
        return endDate;
    }

    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }

    public String getApprovalDate() {
        return approvalDate;
    }

    public void setApprovalDate(String approvalDate) {
        this.approvalDate = approvalDate;
    }
}

public class LeaveFactory {
    public static Leave createLeave(String type, String requestDate, String startDate, String endDate) {
        switch (type) {
            case "CL":
                return new CLLeave(requestDate);
            case "SL":
                return new SLLeave(requestDate);
            case "VL":
                return new VLLeave(requestDate, startDate, endDate);
            default:
                return new InvalidLeave(requestDate);
        }
    }
}



public interface LeaveProcessor {
    void processLeave(Leave leave);
}

public class ProjectManager implements LeaveProcessor {
    private LeaveProcessor successor;

    @Override
    public void processLeave(Leave leave) {
        if (leave instanceof CLLeave) {
            System.out.println("ProjectManager processed CLLeave");
            leave.setApprovedBy("ProjectManager");
            leave.setLeaveStatus("Approved");
        } else {
            successor.processLeave(leave);
        }
    }
```

```
    public void setSuccessor(LeaveProcessor successor) {
        this.successor = successor;
    }
}

public class SLLeave extends Leave {
    public SLLeave(String requestDate) {
        super("New", requestDate);
    }
}

public class TechLead implements LeaveProcessor {
    private LeaveProcessor successor;

    @Override
    public void processLeave(Leave leave) {
        if (leave instanceof SLLeave) {
            System.out.println("TechLead processed SLLeave");
            leave.setApprovedBy("TechLead");
            leave.setLeaveStatus("Approved");
        } else {
            successor.processLeave(leave);
        }
    }

    public void setSuccessor(LeaveProcessor successor) {
        this.successor = successor;
    }
}

public class VLLeave extends Leave {
    private String startDate;
    private String endDate;

    public VLLeave(String requestDate, String startDate, String endDate) {
        super("New", requestDate);
        this.startDate = startDate;
        this.endDate = endDate;
    }

    public String getStartDate() {
        return startDate;
    }

    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }

    public String getEndDate() {
        return endDate;
    }
```

```
    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }
}
```

Input and Output Screenshots for all types of leaves