Mubeen Khalid (SP21-BSE-015)

Design pattern

Assignment # 3

# Source Code:

## IEmployee (Component):

public interface IEmployee {

    void giveBonus(int bonusPercentage);

}

## BaseEmployee:

public abstract class BaseEmployee implements IEmployee {

    protected String name;

    protected int salary;

    public BaseEmployee(String name, int salary) {

        this.name = name;

        this.salary = salary;

    }

    public String getName() {

        return name;

    }

    public int getSalary() {

        return salary;

    }

    @Override

    public void giveBonus(int bonusPercentage) {

        this.salary += this.salary \* bonusPercentage / 100;

    }

}

## Teacher (Leaf):

public class Teacher extends BaseEmployee {

    private String qualification;

    private String status;

    public Teacher(String name, int salary, String qualification, String status) {

        super(name, salary);

        this.qualification = qualification;

        this.status = status;

    }

    @Override

    public String toString() {

        return "Teacher: [Name: " + name + ", Salary: " + salary + ", Qualification: " + qualification

                + ", Status: " + status + "]";

    }

}

## DOO (Leaf):

public class DOO extends BaseEmployee {

    private String rank;

    public DOO(String name, int salary, String rank) {

        super(name, salary);

        this.rank = rank;

    }

    @Override

    public String toString() {

        return "DOO: [Name: " + name + ", Salary: " + salary + ", Rank: " + rank + "]";

    }

}

## Director (Composite):

import java.util.ArrayList;

import java.util.List;

public class Director extends BaseEmployee {

    private List<IEmployee> subordinates;

    public Director(String name, int salary) {

        super(name, salary);

        this.subordinates = new ArrayList<>();

    }

    public void addSubordinate(IEmployee employee) {

        subordinates.add(employee);

    }

    public void removeSubordinate(IEmployee employee) {

        subordinates.remove(employee);

    }

    public List<IEmployee> getSubordinates() {

        return subordinates;

    }

    @Override

    public void giveBonus(int bonusPercentage) {

        super.giveBonus(bonusPercentage);

        for (IEmployee employee : subordinates) {

            employee.giveBonus(bonusPercentage);

        }

    }

    @Override

    public String toString() {

        StringBuilder sb = new StringBuilder();

        sb.append("Director [Name: ").append(name).append(", Salary: ").append(salary).append("]\n");

        for (IEmployee employee : subordinates) {

            sb.append("\t").append(employee.toString()).append("\n");

        }

        return sb.toString();

    }

}

## HOD (Composite):

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class HOD extends BaseEmployee {

    private String department;

    private List<IEmployee> subordinates;

    public HOD(String name, int salary, String department) {

        super(name, salary);

        this.department = department;

        this.subordinates = new ArrayList<>();

    }

    public void addSubordinate(IEmployee employee) {

        subordinates.add(employee);

    }

    public void removeSubordinate(IEmployee employee) {

        subordinates.remove(employee);

    }

    public List<IEmployee> getSubordinates() {

        return subordinates;

    }

    @Override

    public void giveBonus(int bonusPercentage) {

        super.giveBonus(bonusPercentage);

        for (IEmployee employee : subordinates) {

            employee.giveBonus(bonusPercentage);

        }

    }

    @Override

    public String toString() {

        StringBuilder builder = new StringBuilder();

        builder.append("HOD: [Name: ").append(name).append(", Salary: ").append(salary).append(", Department: ")

                .append(department).append("]\n");

        for (IEmployee employee : subordinates) {

            builder.append("\t").append(employee.toString()).append("\n");

        }

        return builder.toString();

    }

    public Iterator<IEmployee> iterator() {

        return subordinates.iterator();

    }

}

## Employee (Client):

public class Employee {

    private Director director;

    public void generateOrganogram() {

        director = new Director("Asim Muneer", 16);

        HOD headCS = new HOD("Mazhar Iqbal", 10, "Computer Science");

        HOD headEE = new HOD("Umar Farid", 5, "Electrical Engineering");

        Teacher teacher1 = new Teacher("Mukhtiar Zamin", 15, "PhD", "Permanent");

        Teacher teacher2 = new Teacher("Kashif Bilal", 6, "PhD", "Permanent");

        DOO doo1 = new DOO("Qamar Javed", 7, "DOO");

        DOO doo2 = new DOO("Iqbal", 5, "DOO");

        director.addSubordinate(headCS);

        director.addSubordinate(headEE);

        headCS.addSubordinate(teacher1);

        headCS.addSubordinate(doo1);

        headEE.addSubordinate(teacher2);

        headEE.addSubordinate(doo2);

    }

    public void bonuses(String bonusType, int percentageOfBonus) {

        if (director != null) {

            director.giveBonus(percentageOfBonus);

        }

    }

    public void printDetails() {

        if (director != null) {

            System.out.println(director);

        }

    }

}

## Demo:

public class CompositePatternDemo {

    public static void main(String[] args) {

        Employee controller = new Employee();

        controller.generateOrganogram();

        controller.printDetails();

        System.out.println("\nActivating bonus...\n");

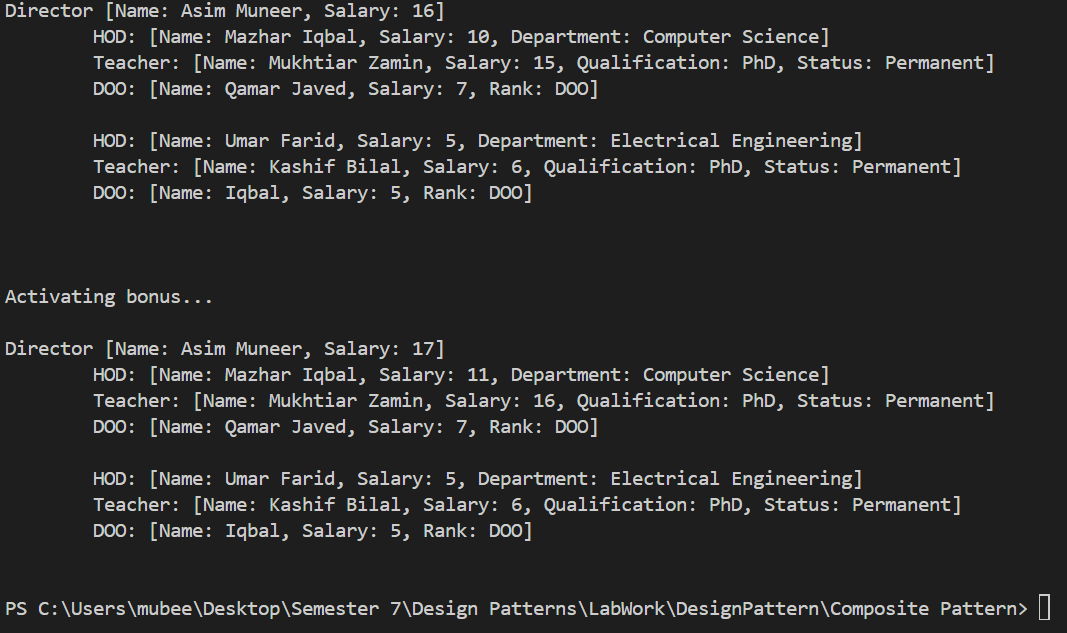
        controller.bonuses("year-end", 10);

        controller.printDetails();

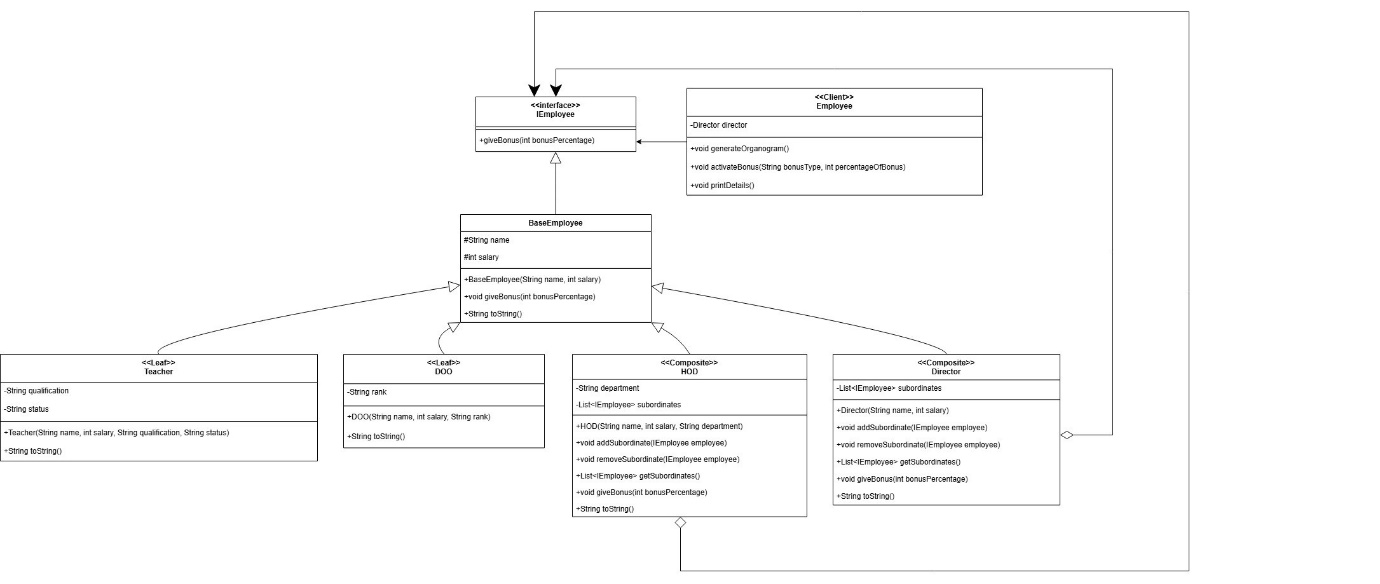
    }

}

## Output:



## Class Diagram:



Github Link:

[DesignPattern/Composite Pattern at main · mubeen070/DesignPattern (github.com)](https://github.com/mubeen070/DesignPattern/tree/main/Composite%20Pattern)