"Heaven's Light is Our Guide"

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Lab Report

Course Title: Software Engineering Sessional

Course Code : CSE 3206

SUBMITTED BY

Ridwanul Islam

Roll: 2003133

Md. Tasreful Islam

Roll: 2003134

Md. Shifat Alam

Roll: 2003135

SUBMITTED TO

Farjana Parvin

Lecturer

Department of Computer Science & Engineering, RUET

Title: Solving problems related to Proxy (Structural Design Pattern) and Chain of Responsibility (Behavioral Design Pattern).

Introduction:

A design pattern in software engineering is a reusable solution to a commonly occurring problem in software design, providing a proven and efficient way to solve design issues.

There are three types of design patterns:

i. Creational Design Patterns
ii. Structural Design Patterns
iii. Behavioral Design Patterns

Among them, behavioral design patterns are concerned with algorithms and the assignment of responsibilities between objects. And the structural design patterns explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.

Proxy is a type of Structural Design Pattern, that lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

An example problem that can solved using Proxy would be:
You are building a database system where certain users must be restricted from querying sensitive data.
Direct access to the real database could compromise security or performance.
Challenges:

- Need to control who can access the database.
- Protect sensitive operations.
- Ensure that the application maintains a clean structure while controlling access.

And, Chain of Responsibility is a Behavioral Design Pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

An example problem that can be solved using Chain of Responsibility is:
In a company, there is a need to implement an approval process for purchase orders based on the purchase amount. The approval process follows the rules:

1. If the purchase amount is below $500, a Junior Associate can approve the purchase.
2. If the purchase amount is above $500 but below $1500, a Senior Associate can approve the purchase.
3. If the purchase amount is above $1500 but below $3000, a Manager needs to approve the purchase.
4. If the purchase amount exceeds the approval limit of the Manager, the request is cancelled.

Code:

For the problem related to Proxy:

```
import java.util.ArrayList;

interface Database {
    void query(String sql);
}

class RealDatabase implements Database {
    public void query(String sql) {
        System.out.println("Executing query: " + sql);
    }
}

class DatabaseProxy implements Database {
    private ArrayList<String> allowedUsers;
    private RealDatabase realDatabase;
    private String currentUser;

    public DatabaseProxy(ArrayList<String> allowedUsers, String currentUser) {
        this.allowedUsers = allowedUsers;
        this.realDatabase = null;
        this.currentUser = currentUser;
    }
```

```
    public void query(String sql) {
        if (isUserAllowedToQuery()) {
            getRealDatabase().query(sql);
        } else {
            System.out.println("Access denied: User does not have permission.");
        }
    }

    private RealDatabase getRealDatabase() {
        if (realDatabase == null) {
            realDatabase = new RealDatabase();
        }
        return realDatabase;
    }

    private boolean isUserAllowedToQuery() {
        return this.allowedUsers.contains(this.currentUser);
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<String> allowedUsers = new ArrayList<>();
        allowedUsers.add("admin");
        allowedUsers.add("user");

        DatabaseProxy databaseProxy1 = new DatabaseProxy(allowedUsers, "user4");

        databaseProxy1.query("SELECT * FROM employees");
        databaseProxy1.query("INSERT INTO employees VALUES ('John', 'Doe', 30)");

        DatabaseProxy databaseProxy2 = new DatabaseProxy(allowedUsers, "user");

        databaseProxy2.query("SELECT * FROM employees");
        databaseProxy2.query("INSERT INTO employees VALUES ('John', 'Doe', 30)");
    }
}
```

For the problem related to Chain of Responsibility:

```
import java.util.ArrayList;

class PurchaseRequest {
    private int amount;
    private boolean approvedStatus = false;
    private String approvedBy;
```

```
    public PurchaseRequest(int amount) {
        this.amount = amount;
    }

    public int getAmount() {
        return amount;
    }

    public void processPR(String approver) {
        this.approvedBy = approver;
        this.approvedStatus = true;
    }


    public void getStatus() {
        System.out.println("---- Purchase Order Summary ----");
        System.out.println("Amount: " + amount);
        System.out.println("Approved Status: " +
                (approvedStatus ? "Approved" : "Sorry, this request cannot be
approved."));
        System.out.println("Approver Name: " + approvedBy);
        System.out.println("-------------------------------");
    }
}

interface Handler {
    Handler setNext(Handler next);
    void handle(PurchaseRequest request);
}

abstract class Employee implements Handler {
    private Handler next;
    protected int approvalLimit;
    protected String name;

    public Handler setNext(Handler next) {
        this.next = next;
        return next;
    }

    public void handle(PurchaseRequest request) {
        if (next != null) {
            next.handle(request);
        }
    }

    protected void approve(PurchaseRequest request) {
        request.processPR(name);
```

```
    }
}

class JuniorAssociate extends Employee {
    public JuniorAssociate() {
        this.name = "Junior";
        this.approvalLimit = 500;
    }

    public void handle(PurchaseRequest request) {
        System.out.print(name + "[ ");
        if (request.getAmount() <= approvalLimit) {
            System.out.print("O ] ");
            approve(request);
        } else {
            System.out.print("X ] ");
            super.handle(request);
        }
    }
}

class SeniorAssociate extends Employee {
    public SeniorAssociate() {
        this.name = "Senior";
        this.approvalLimit = 1500;
    }

    public void handle(PurchaseRequest request) {
        System.out.print(name + "[ ");
        if (request.getAmount() <= approvalLimit) {
            System.out.print("O ] ");
            approve(request);
        } else {
            System.out.print("X ] ");
            super.handle(request);
        }
    }
}

class Manager extends Employee {
    public Manager() {
        this.name = "Manager";
        this.approvalLimit = 3000;
    }

    public void handle(PurchaseRequest request) {
        System.out.print(name + "[ ");
        if (request.getAmount() <= approvalLimit) {
```

```
            System.out.print("O ] ");
            approve(request);
        } else {
            System.out.print("X ] ");
            super.handle(request);
        }
    }
}

public class Main {
    public static void approvalWorkflow(Handler handler) {
        handler.setNext(new SeniorAssociate()).setNext(new Manager());

        ArrayList<PurchaseRequest> PRs = new ArrayList<>();
        PRs.add(new PurchaseRequest(200));
        PRs.add(new PurchaseRequest(600));
        PRs.add(new PurchaseRequest(1600));
        PRs.add(new PurchaseRequest(5000));

        for (PurchaseRequest PR : PRs) {
            System.out.println("\nApproval Flow: ");
            handler.handle(PR);
            System.out.println();
            PR.getStatus();
        }
    }

    public static void main(String[] args) {
        Handler junior = new JuniorAssociate();
        approvalWorkflow(junior);
    }
}
```