Day 25 - 09th Aug 2025

Task01 Bridge Method Structural Design Pattern:

package BridgeMethod;

public interface ExcalidrawAPI {

void drawSquare(int s);

}

package BridgeMethod;

public class DrawingPicture implements ExcalidrawAPI {

@Override

public void drawSquare(int s) {

System.*out*.println("Draw square in Drawing Picture using Excalidraw with side " + s);

}

}

package BridgeMethod;

public class DrawingFrame implements ExcalidrawAPI {

@Override

public void drawSquare(int s) {

System.*out*.println("Draw square in Drawing Frame using Excalidraw with side " + s);

}

}

package BridgeMethod;

// Abstraction

abstract public class Shape {

protected ExcalidrawAPI excalidrawAPI;

protected Shape(ExcalidrawAPI excalidrawAPI) {

this.excalidrawAPI = excalidrawAPI;

}

abstract void draw();

}

package BridgeMethod;

public class Square extends Shape {

private int s;

Square(int s, ExcalidrawAPI excalidrawAPI) {

super(excalidrawAPI);

this.s = s;

}

@Override

void draw() {

excalidrawAPI.drawSquare(s);

}

}

package BridgeMethod;

public class Main {

public static void main(String[] args) {

System.*out*.println("Bridge Method Design Pattern - Structural DP!");

// Draw square using DrawingFrame

Shape square1 = new Square(5, new DrawingFrame());

square1.draw();

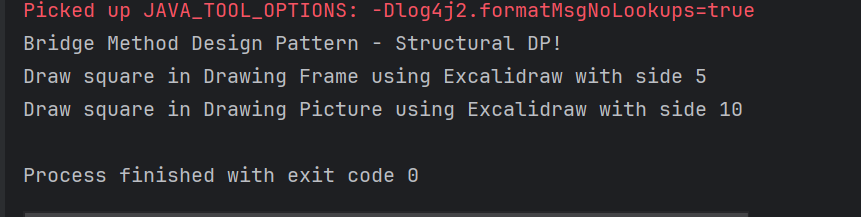
// Draw square using DrawingPicture

Shape square2 = new Square(10, new DrawingPicture());

square2.draw();

}

}



Task02 Composite Method Structural Design Pattern

package CompositeMethod;

// Base Component

public interface Company {

void displayName();

}

package CompositeMethod;

import java.util.ArrayList;

import java.util.List;

// Composite component

public class CompanyHead implements Company {

private int id;

private String name;

private List<Company> subDepartments;

public CompanyHead(int id, String name) {

this.id = id;

this.name = name;

this.subDepartments = new ArrayList<>();

}

@Override

public void displayName() {

System.*out*.println("Department: " + name);

subDepartments.forEach(Company::displayName);

}

public void addDepartment(Company company) {

subDepartments.add(company);

}

public void removeDepartment(Company company) {

subDepartments.remove(company);

}

}

package CompositeMethod;

public class HR implements Company {

private int id;

private String name;

public HR(int id, String name) {

this.id = id;

this.name = name;

}

@Override

public void displayName() {

System.*out*.println("HR Dept: " + name);

}

}

package CompositeMethod;

// Leaf component

public class Software implements Company {

private int id;

private String name;

public Software(int id, String name) {

this.id = id;

this.name = name;

}

@Override

public void displayName() {

System.*out*.println("Software Dept: " + name);

}

// Getters

public int getId() {

return id;

}

public String getName() {

return name;

}

// Setters

public void setId(int id) {

this.id = id;

}

public void setName(String name) {

this.name = name;

}

// Optional: for debugging

@Override

public String toString() {

return "Software{" +

"id=" + id +

", name='" + name + '\'' +

'}';

}

}

package CompositeMethod;

public class Main {

public static void main(String[] args) {

System.*out*.println("Composite Method DP - Structural DP");

// Create leaf departments

Company softwareCompany = new Software(1, "Software Development");

Company hrDepartment = new HR(2, "Human Resources");

// Create composite head

CompanyHead companyHead = new CompanyHead(3, "ABC Company");

// Add leaves to composite

companyHead.addDepartment(softwareCompany);

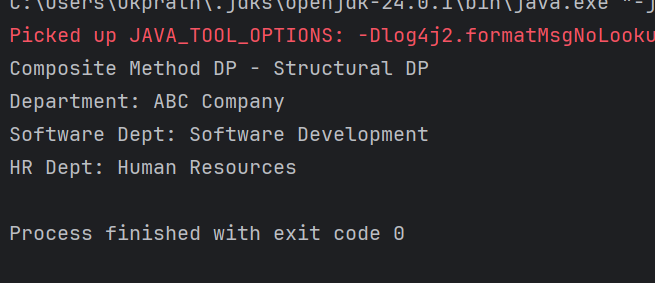
companyHead.addDepartment(hrDepartment);

// Display hierarchy

companyHead.displayName();

}

}



Task 03 Facade Method

package FacadeMethod;

public class BillingCounterService {

public void payBill(String accountId, String billId, double amount) {

System.*out*.println("paying for Mangoes "+ amount +"for billId "+ billId + "from account" +accountId);

}

}

package FacadeMethod;

public class MallFacade {

private StoreStaffService storeStaffService;

private BillingCounterService billingCounterService;

public MallFacade() {

this.storeStaffService = new StoreStaffService();

this.billingCounterService = new BillingCounterService();

}

public void getItems(String items) {

storeStaffService.getItems(items);

}

public void payBill(String accountId, String billId, double amount){

billingCounterService.payBill(accountId,billId, amount );

}

}

package FacadeMethod;

public class StoreStaffService {

public void getItems(String items) {

System.*out*.println("selecting Mangoes");

}

}

package FacadeMethod;

public class Main {

public static void main(String[] args) {

System.*out*.println("Facade Method DP - Structural Design Pattern");

MallFacade mallFacade = new MallFacade();

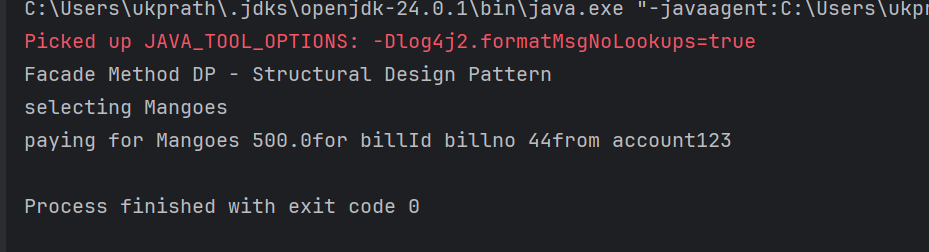
mallFacade.getItems("Fruits");

// mallFacade.transferMoney("123", "7777", 500.0);

mallFacade.payBill("123", "billno 44", 500.0);

}

}



Task 04 Proxy design pattern

package ProxyMethod;

public class DataBase {

void execute(String Query, String desig) {

System.*out*.println("query execution is in process "+ Query);

}

}

package ProxyMethod;

public interface DatabaseExecute {

public abstract void executeQuery(String type);

}

package ProxyMethod;

import java.util.Objects;

public class ProxyDataBase implements DatabaseExecute{

String desig;

DataBase db;

public ProxyDataBase(String desig){

this.desig = desig;

db = new DataBase();

}

@Override

public void executeQuery(String Type){

if(Type.equals("Delete") && (!Objects.*equals*(this.desig, "ADMIN"))) {

System.*out*.println("you don't have permission to delete");

return;

}

db.execute(Type, this.desig);

}

}

package ProxyMethod;

public class Main {

public static void main(String[] args) {

System.*out*.println("Proxy Method Design Pattern - Structural design pattern");

DatabaseExecute emp1 = new ProxyDataBase("ADMIN");

emp1.executeQuery("Delete");

emp1.executeQuery("Write");

emp1.executeQuery("Read");

DatabaseExecute emp2 = new ProxyDataBase("HR");

emp2.executeQuery("Delete");

emp2.executeQuery("Write");

emp2.executeQuery("Read");

DatabaseExecute emp3 = new ProxyDataBase("EMPLOYEE");

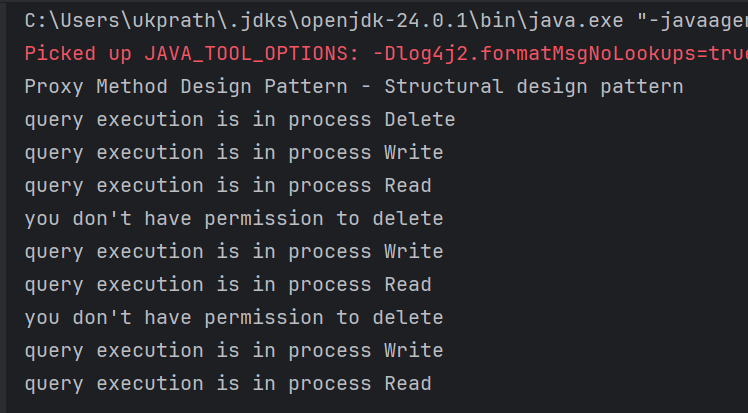
emp3.executeQuery("Delete");

emp3.executeQuery("Write");

emp3.executeQuery("Read");

}

}



Task 05 Command Method Design Pattern

package BehavioralDP.CommandPattern;

public interface Command {

void doIt(); // executing a command

}

package BehavioralDP.CommandPattern;

public class DontDoTask implements Command {

private Task task;

public DontDoTask(Task task) { // ✅ Proper constructor

this.task = task;

}

@Override

public void doIt() {

this.task.dontTask();

}

}

package BehavioralDP.CommandPattern;

public class DoTask implements Command {

private Task task;

public DoTask(Task task) { // ✅ Proper constructor

this.task = task;

}

@Override

public void doIt() {

this.task.doTask();

}

}

package BehavioralDP.CommandPattern;

public class Mom {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void executeCommand() {

this.command.doIt();

}

}

package BehavioralDP.CommandPattern;

public class Task {

public void doTask() {

System.*out*.println("do your home tasks");

}

public void dontTask() {

System.*out*.println("don't do your home tasks");

}

}

package BehavioralDP.CommandPattern;

public class Main {

public static void main(String[] args) {

System.*out*.println("Command Pattern - Behavioural DP");

Task task = new Task();

Mom remote = new Mom();

Command onCommand = new DoTask(task);

Command offCommand = new DontDoTask(task);

remote.setCommand(onCommand);

remote.executeCommand(); // "do your home tasks"

remote.setCommand(offCommand);

remote.executeCommand(); // "don't do your home tasks"

}

}

