***I.meghana(2300030257)***

**AOOP SKILL WEEK-4**

**Problem-1** abstract class DataProcessor { public final void process() { loadData(); processData(); saveData();

}

protected abstract void loadData(); protected abstract void processData(); protected abstract void saveData();

}

class CSVDataProcessor extends DataProcessor { protected void loadData() {

System.out.println("Loading CSV data...");

}

protected void processData() {

System.out.println("Processing CSV data...");

}

protected void saveData() {

System.out.println("Saving processed CSV data...");

}

}

class JSONDataProcessor extends DataProcessor { protected void loadData() { System.out.println("Loading JSON data...");

protected void processData() {

System.out.println("Processing JSON data...");

}

protected void saveData() {

System.out.println("Saving processed JSON data...");

}

}

public class TemplateMethodDemo {

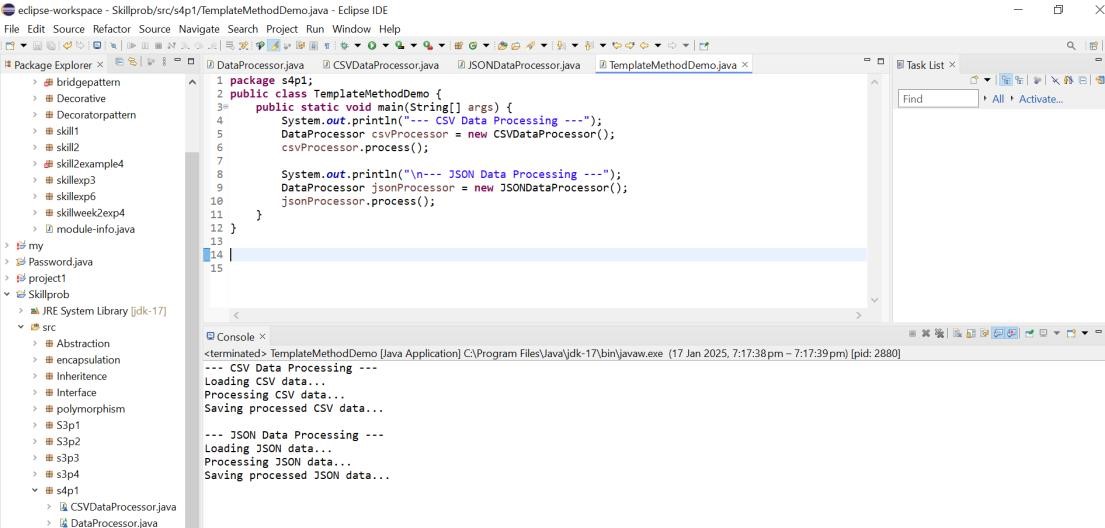
public static void main(String[] args) {

System.out.println("--- CSV Data Processing ---"); DataProcessor csvProcessor = new CSVDataProcessor(); csvProcessor.process();

System.out.println("\n--- JSON Data Processing ---"); DataProcessor jsonProcessor = new JSONDataProcessor(); jsonProcessor.process();

}

}



**Problem-2** package pipeline; public interface Handler { void setNext(Handler nextHandler); void handleRequest(Request request);

}

package pipeline; public class AuthenticationHandler implements Handler { private Handler nextHandler; public void setNext(Handler nextHandler) { this.nextHandler = nextHandler;

}

public void handleRequest(Request request) { if (request.isAuthenticated()) {

System.out.println("Authentication successful."); if (nextHandler != null) { nextHandler.handleRequest(request);

}

} else {

System.out.println("Authentication failed. Request denied.");

}

}

}

package pipeline; public class ValidationHandler implements Handler { private Handler nextHandler; public void setNext(Handler nextHandler) { this.nextHandler = nextHandler;

public void handleRequest(Request request) {

if (request.isValid()) {

System.out.println("Validation successful."); if (nextHandler != null) { nextHandler.handleRequest(request);

} }

else {

System.out.println("Validation failed. Request denied.");

}

}

}

package pipeline; public class BusinessLogicHandler implements Handler { private Handler nextHandler; public void setNext(Handler nextHandler) { this.nextHandler = nextHandler;

}

public void handleRequest(Request request) {

System.out.println("Processing business logic..."); System.out.println("Business logic handled successfully."); if (nextHandler != null) { nextHandler.handleRequest(request);

}

}

}

package pipeline; public class Request { private boolean authenticated; private boolean valid; public Request(boolean authenticated, boolean valid) { this.authenticated = authenticated; this.valid = valid;

}

public boolean isAuthenticated() { return authenticated;

}

public boolean isValid() { return valid;

}

}

package pipeline; public class HandlerChain { private Handler head; public void addHandler(Handler handler) {

if (head == null) { head = handler;

} else {

Handler current = head; while (current instanceof Handler && ((Handler) current).getNext() != null) { current = ((Handler) current).getNext();

}

((Handler) current).setNext(handler);

}

public void handle(Request request) {

if (head != null) { head.handleRequest(request);

}

}

}

package pipeline; public class ChainOfResponsibilityDemo { public static void main(String[] args) {

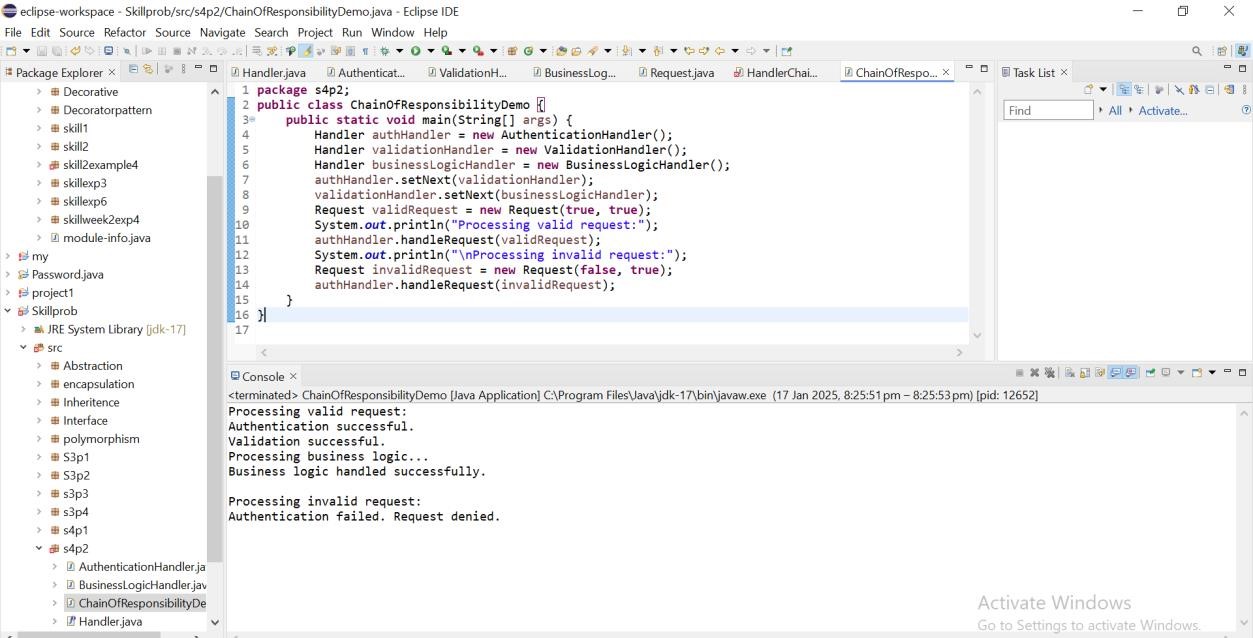
Handler authHandler = new AuthenticationHandler();

Handler validationHandler = new ValidationHandler(); Handler businessLogicHandler = new BusinessLogicHandler(); authHandler.setNext(validationHandler); validationHandler.setNext(businessLogicHandler); Request validRequest = new Request(true, true); System.out.println("Processing valid request:"); authHandler.handleRequest(validRequest);

System.out.println("\nProcessing invalid request:"); Request invalidRequest = new Request(false, true); authHandler.handleRequest(invalidRequest);

}

}



**Problem-3** package texteditor; public interface Command { void execute();

}

package texteditor; public class CopyCommand implements Command {

private TextEditor textEditor; public CopyCommand(TextEditor textEditor) { this.textEditor = textEditor;

}

public void execute() { textEditor.copy();

}

}

package texteditor; public class PasteCommand implements Command {

private TextEditor textEditor;

public PasteCommand(TextEditor textEditor) { this.textEditor = textEditor;

}

public void execute() { textEditor.paste();

}

}

package texteditor; public class UndoCommand implements Command {

private TextEditor textEditor; public UndoCommand(TextEditor textEditor) { this.textEditor = textEditor;

}

public void execute() { textEditor.undo();

}

}

package texteditor; import java.util.Stack; public class TextEditor { private String clipboard; private StringBuilder text; private Stack<Command> history; public TextEditor() { this.clipboard = ""; this.text = new StringBuilder(); this.history = new Stack<>();

}

public void copy() { clipboard = text.toString();

System.out.println("Copied to clipboard: " + clipboard);

}

public void paste() { text.append(clipboard);

System.out.println("Pasted text: " + text.toString()); saveCommand(new PasteCommand(this));

}

public void undo() { if (!history.isEmpty()) { history.pop();

System.out.println("Undo last command. Current text: " + text.toString());

} else {

System.out.println("Nothing to undo.");

}

}

private void saveCommand(Command command) { history.push(command);

}

public void appendText(String newText) { text.append(newText);

System.out.println("Text after append: " + text.toString());

}

public String getText() { return text.toString(); }

}

package texteditor; public class CommandPatternDemo {

public static void main(String[] args) { TextEditor textEditor = new TextEditor(); textEditor.appendText("Hello, world! ");

Command copyCommand = new CopyCommand(textEditor);

Command pasteCommand = new PasteCommand(textEditor);

Command undoCommand = new UndoCommand(textEditor); System.out.println("\nExecuting Copy Command:"); copyCommand.execute();

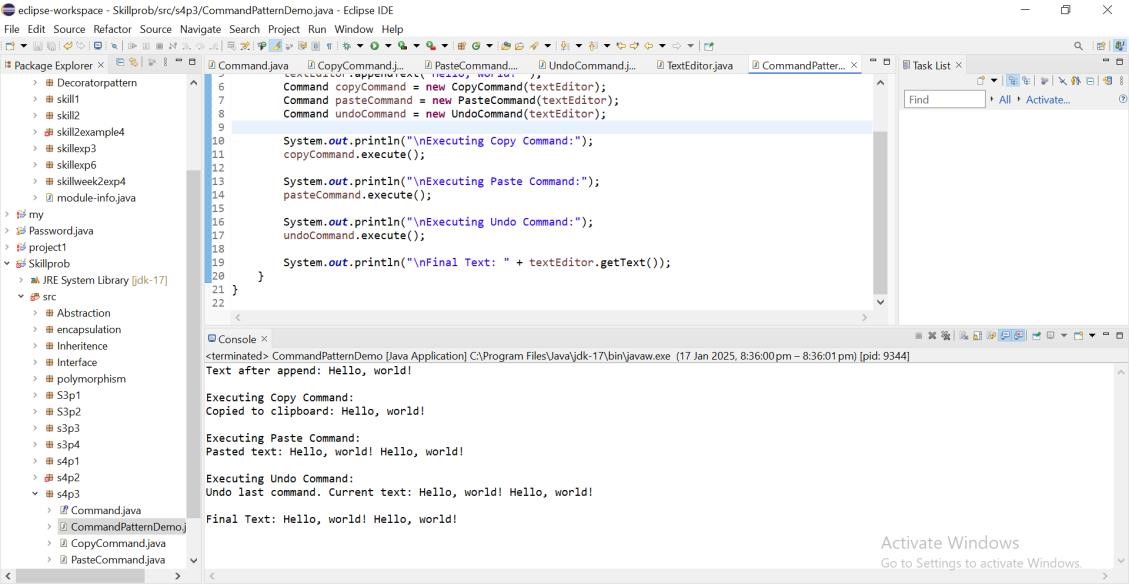
System.out.println("\nExecuting Paste Command:"); pasteCommand.execute();

System.out.println("\nExecuting Undo Command:"); undoCommand.execute();

System.out.println("\nFinal Text: " + textEditor.getText());

}

}



**Problem-4** package supportsystem; public interface SupportHandler { void setNext(SupportHandler nextHandler); void handleRequest(Issue issue);

}

package supportsystem; public class Issue { private String description; private int complexityLevel; public Issue(String description, int complexityLevel) { this.description = description; this.complexityLevel = complexityLevel;

}

public String getDescription() { return description;

}

public int getComplexityLevel() { return complexityLevel;

}

}

package supportsystem; public class Level1Support implements SupportHandler { private SupportHandler nextHandler; public void setNext(SupportHandler nextHandler) { this.nextHandler = nextHandler;

}

public void handleRequest(Issue issue) { if (issue.getComplexityLevel() == 1) {

System.out.println("Level 1 Support: Resolved issue - " + issue.getDescription());

} else {

System.out.println("Level 1 Support: Escalating issue - " + issue.getDescription()); if (nextHandler != null) { nextHandler.handleRequest(issue);

} else {

System.out.println("No further escalation possible. Issue cannot be resolved.");

}

}

}

}

package supportsystem; public class Level2Support implements SupportHandler { private SupportHandler nextHandler; public void setNext(SupportHandler nextHandler) { this.nextHandler = nextHandler;

}

public void handleRequest(Issue issue) { if (issue.getComplexityLevel() == 2) {

System.out.println("Level 2 Support: Resolved issue - " + issue.getDescription());

} else {

System.out.println("Level 2 Support: Escalating issue - " + issue.getDescription()); if (nextHandler != null) { nextHandler.handleRequest(issue);

} else {

System.out.println("No further escalation possible. Issue cannot be resolved.");

}

}

}

}

package supportsystem; public class Level3Support implements SupportHandler { private SupportHandler nextHandler; public void setNext(SupportHandler nextHandler) { this.nextHandler = nextHandler;

}

public void handleRequest(Issue issue) { if (issue.getComplexityLevel() == 3) {

System.out.println("Level 3 Support: Resolved issue - " + issue.getDescription());

} else {

System.out.println("Level 3 Support: Escalating issue - " + issue.getDescription()); if (nextHandler != null) { nextHandler.handleRequest(issue);

} else {

System.out.println("No further escalation possible. Issue cannot be resolved.");

}

}

}

}

package supportsystem; public class SupportSystemDemo { public static void main(String[] args) { SupportHandler level1 = new Level1Support();

SupportHandler level2 = new Level2Support(); SupportHandler level3 = new Level3Support(); level1.setNext(level2); level2.setNext(level3);

Issue issue1 = new Issue("Reset password", 1);

Issue issue2 = new Issue("Install software", 2);

Issue issue3 = new Issue("Debug critical error", 3);

Issue issue4 = new Issue("Unknown issue", 4); System.out.println("Handling Issue 1:"); level1.handleRequest(issue1);

System.out.println("\nHandling Issue 2:"); level1.handleRequest(issue2);

System.out.println("\nHandling Issue 3:"); level1.handleRequest(issue3);

System.out.println("\nHandling Issue 4:"); level1.handleRequest(issue4);

}}

