

Supplementary Exercise on Command Pattern

(a)	<pre> classDiagram class Invoker class Command { +execute() } class Receiver { +action() } class ConcreteCommand { +execute() } Invoker < -- Command ConcreteCommand < -- Command Receiver < -- ConcreteCommand Note over ConcreteCommand: public execute() { receiver.action(); } </pre> <p>The diagram illustrates the Command pattern. It consists of four classes: Invoker, Command, Receiver, and ConcreteCommand. Invoker has a dependency on Command. Command declares an interface with the <code>+execute()</code> method. ConcreteCommand implements this interface and also depends on Command. ConcreteCommand has a dependency on Receiver, which it calls via its <code>+action()</code> method. A note indicates that the <code>execute()</code> implementation in ConcreteCommand calls receiver.action().</p>
(b)	<ul style="list-style-type: none"> The Invoker holds a command and can get the Command to execute a request by calling the <code>execute</code> method. The Command is an object that encapsulates a request to the receiver. Command declares an interface for all commands, providing a simple <code>execute()</code> method which asks the Receiver of the command to carry out an operation. Receiver. The receiver is the component that is acted upon by each request. The ConcreteCommand defines a binding between the action and the receiver. When the Invoker calls <code>execute</code> the ConcreteCommand will run one or more actions on the Receiver.
(c)	<pre> classDiagram class DocumentController class Command { <<Interface>> +execute() : void +undo() : void } class EditCommand { -previousContent : String -doc : Document +EditCommand(Document doc) +execute() +undo() } class ShowCommand { -doc : Document +ShowCommand(Document doc) +execute() +undo() } class Document { -content : String +getContent() +setContent(String content) } DocumentController --> Command EditCommand < -- Command ShowCommand < -- Command Document < -- EditCommand Document < -- ShowCommand </pre> <p>The diagram shows a concrete implementation of the Command pattern. It includes a DocumentController class that interacts with a Command interface. Two concrete command classes, EditCommand and ShowCommand, implement the Command interface. Both EditCommand and ShowCommand depend on a Document object. The Document class has attributes for content and methods to get and set content. A dashed line connects the Command interface to both EditCommand and ShowCommand, indicating they implement the interface.</p>

(d)	<pre> public interface Command { public void execute(); public void undo(); } public class EditCommand implements Command { private Document doc ; private String previousContent; public EditCommand (Document doc){ this.doc = doc; } public void execute() { // save previous content previousContent = doc.getContent(); // get new content Scanner s = new Scanner(System.in); System.out.println("Please input your content"); String value = s.nextLine(); doc.setContent(value); } public void undo() { doc.setContent(previousContent); } } public class ShowCommand implements Command { private Document doc ; public ShowCommand (Document doc){ this.doc = doc; } public void execute() { System.out.println("The current content is "+ doc.getContent()); } public void undo() { } } </pre>
-----	--