Command Pattern

1. Customer Requirement

A customer wants to build a text editor that supports undo and redo functionality. The text editor should be able to encapsulate user actions (such as typing text, deleting text, etc.) into objects so that these actions can be executed, undone, and redone as needed.

2. Design Pattern Explanation

The Command Pattern is used to encapsulate a request as an object, thereby allowing for the parameterization of clients with queues, requests, and operations. It also allows for the support of undoable operations.

Example:

- Text Editor:
  - Command: Each action (typing, deleting) is represented as a command object.
  - Invoker: The text editor that triggers commands.
  - Receiver: The document where the actions (commands) are executed.
  - Undo/Redo: The command objects can be stored in stacks to support undo and redo operations.

3. Java Code Implementation

```java
// Command Interface
interface Command {
    void execute();
    void undo();
}

// Concrete Command: TypeTextCommand
```

```
class TypeTextCommand implements Command {
    private String text;
    private StringBuilder document;

    public TypeTextCommand(String text, StringBuilder document) {
        this.text = text;
        this.document = document;
    }

    @Override
    public void execute() {
        document.append(text);
    }

    @Override
    public void undo() {
        document.delete(document.length() - text.length(), document.length());
    }
}

// Concrete Command: DeleteTextCommand
class DeleteTextCommand implements Command {
    private String deletedText;
    private StringBuilder document;

    public DeleteTextCommand(StringBuilder document) {
        this.document = document;
```

```
    }

    @Override
    public void execute() {
        if (document.length() > 0) {
            deletedText = document.substring(document.length() - 1);
            document.deleteCharAt(document.length() - 1);
        }
    }

    @Override
    public void undo() {
        if (deletedText != null) {
            document.append(deletedText);
        }
    }
}

// Invoker
class TextEditor {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void executeCommand() {
```

```
        command.execute();
    }

    public void undoCommand() {
        command.undo();
    }
}

// Client
public class CommandPatternDemo {
    public static void main(String[] args) {
        StringBuilder document = new StringBuilder();

        TextEditor editor = new TextEditor();

        Command typeHello = new TypeTextCommand("Hello", document);
        Command deleteLastChar = new DeleteTextCommand(document);

        editor.setCommand(typeHello);
        editor.executeCommand();
        System.out.println("After typing: " + document);

        editor.setCommand(deleteLastChar);
        editor.executeCommand();
        System.out.println("After deleting last character: " + document);

        editor.undoCommand();
```

```
        System.out.println("After undoing delete: " + document);


        editor.undoCommand();
        System.out.println("After undoing type: " + document);
    }
}
```

4. Other Important Information

- Encapsulation: Each command encapsulates the information required to perform an action, allowing the action to be executed, undone, and redone as needed.
- Undo/Redo Support: The pattern naturally supports undo and redo operations by maintaining a history of command objects.
- Extensibility: New commands can be easily added without modifying existing code, adhering to the Open/Closed Principle.