# Lab Manual: Design Patterns in a Collaborative Document Editor

Course: Software Design Patterns

Patterns Covered: Creational (Factory Method), Structural (Adapter), Behavioral (Mediator, Memento)

## Scenario Overview

You are building a Collaborative Document Editor. The system starts as a simple document creation tool and gradually evolves into a multi-user, chat-enabled platform with undo functionality. Each stage in the lab manual introduces new requirements that must be solved using a specific design pattern.

## Exercise 1 — Document Creation System

Design Pattern: Factory Method (Creational)

Difficulty: ★☆☆☆☆

🎯 Goal:

Enable users to create different types of documents (Word, PDF) based on their input.

📌 Instructions:

* - Define an abstract Document class with a method open().
* - Implement WordDocument and PDFDocument.
* - Use a DocumentFactory class to generate the correct object based on a string parameter.

💻 Code Template:

abstract class Document {  
 public abstract void open();  
}  
  
class WordDocument extends Document {  
 @Override  
 public void open() {  
 System.out.println("Opening Word document.");  
 }  
}  
  
class PDFDocument extends Document {  
 @Override  
 public void open() {  
 System.out.println("Opening PDF document.");  
 }  
}  
  
class DocumentFactory {  
 public Document createDocument(String type) {  
 // TODO: Return WordDocument or PDFDocument based on type  
 return null;  
 }  
}  
  
public class EditorApp {  
 public static void main(String[] args) {  
 DocumentFactory factory = new DocumentFactory();  
 // TODO: Simulate user input and create a document using the factory  
 Document doc = factory.createDocument("word");  
 doc.open();  
 }  
}

## Exercise 2 — Legacy Format Support

Design Pattern: Adapter (Structural)

Difficulty: ★★☆☆☆

🎯 Goal:

Integrate support for a legacy format (OldPrinter) that cannot be modified.

📌 Instructions:

* - Define a Printable interface.
* - Create an OldPrinter class with an oldPrint() method.
* - Implement an Adapter class to bridge them.

💻 Code Template:

interface Printable {  
 void print(String content);  
}  
  
class OldPrinter {  
 public void oldPrint(String data) {  
 System.out.println("Old printer: " + data);  
 }  
}  
  
class OldPrinterAdapter implements Printable {  
 private OldPrinter oldPrinter;  
  
 public OldPrinterAdapter(OldPrinter oldPrinter) {  
 this.oldPrinter = oldPrinter;  
 }  
  
 @Override  
 public void print(String content) {  
 // TODO: Call oldPrint on the wrapped OldPrinter  
 }  
}

## Exercise 3 — Team Collaboration

Design Pattern: Mediator (Behavioral)

Difficulty: ★★★☆☆

🎯 Goal:

Add real-time chat between users using a ChatRoomMediator.

📌 Instructions:

* - Create a ChatRoomMediator interface and ChatRoom implementation.
* - Define a User class with send(String message) and receive(String message) methods.
* - Register users in the chat room and allow them to communicate.

💻 Code Template:

interface ChatRoomMediator {  
 void sendMessage(String message, User sender);  
 void addUser(User user);  
}  
  
class ChatRoom implements ChatRoomMediator {  
 // TODO: Maintain a list of users and send messages to all except sender  
}  
  
class User {  
 private String name;  
 private ChatRoomMediator chatRoom;  
  
 public User(String name, ChatRoomMediator chatRoom) {  
 this.name = name;  
 this.chatRoom = chatRoom;  
 }  
  
 public void send(String message) {  
 // TODO: Send message via chatRoom  
 }  
  
 public void receive(String message) {  
 System.out.println(name + " received: " + message);  
 }  
  
 public String getName() {  
 return name;  
 }  
}

## Exercise 4 — Undo/Redo in Document Editor

Design Pattern: Memento (Behavioral)

Difficulty: ★★★★☆

🎯 Goal:

Enable the editor to support undo and redo actions by saving snapshots of the document state.

📌 Instructions:

* - Implement a TextEditor (originator) that can write, save, and restore.
* - Create a Memento to hold text state.
* - Use a History class to store and restore states.

💻 Code Template:

class TextEditor {  
 private String text = "";  
  
 public void write(String newText) {  
 text += newText;  
 }  
  
 public String getText() {  
 return text;  
 }  
  
 public EditorMemento save() {  
 // TODO: Return a new memento with the current text  
 return null;  
 }  
  
 public void restore(EditorMemento memento) {  
 // TODO: Restore text from memento  
 }  
}  
  
class EditorMemento {  
 // TODO: Store editor text privately  
}  
  
class History {  
 // TODO: Implement undo/redo stack using List or Stack  
}

## 🏁 Final Integration Challenge — Bring It All Together

Goal:

Design a CollaborativeEditorApp where:  
- Users can create and open documents using the Factory pattern.  
- Users can print with legacy printers using the Adapter.  
- Users can chat using the Mediator.  
- The editor allows undo/redo using the Memento pattern.

💡 Design Tips:

- Use DocumentFactory for creation.  
- Let each User own a TextEditor.  
- Chat between users using ChatRoom.  
- Integrate the History into each TextEditor.