"HEAVENS LIGHT IS OUR GUIDE"

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Lab Report

Course Code: CSE 3206

Course Title: Software Engineering Sessional.

SUBMITTED BY:	**SUBMITTED TO:**
Prothom Antor Banik (Roll: **2003139**) A H Joy (Roll: **2003140**) Md. Wakilul Arifin Akash (Roll: **2003141**) **Department** : CSE **Section** : C	Farjana Parvin Lecturer, Department of CSE, RUET

Title: Problem Solve using design pattern.

Introduction: In software engineering, a design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.

Common design patterns:

1. **Creational patterns:** These patterns are concerned with object creation mechanisms, trying to create objects in a manner suitable to the situation.

2. **Structural patterns:** These patterns deal with class and object composition. That is, they consider ways to create larger structures from smaller ones.

3. **Behavioral patterns:** These patterns are concerned with the assignment of responsibilities between objects

Design Patterns:

Mediator

Objective : Mediator is a behavioral design pattern that reduces chaotic dependencies between objects. The pattern restricts direct communications between objects and forces them to collaborate only via a mediator object.

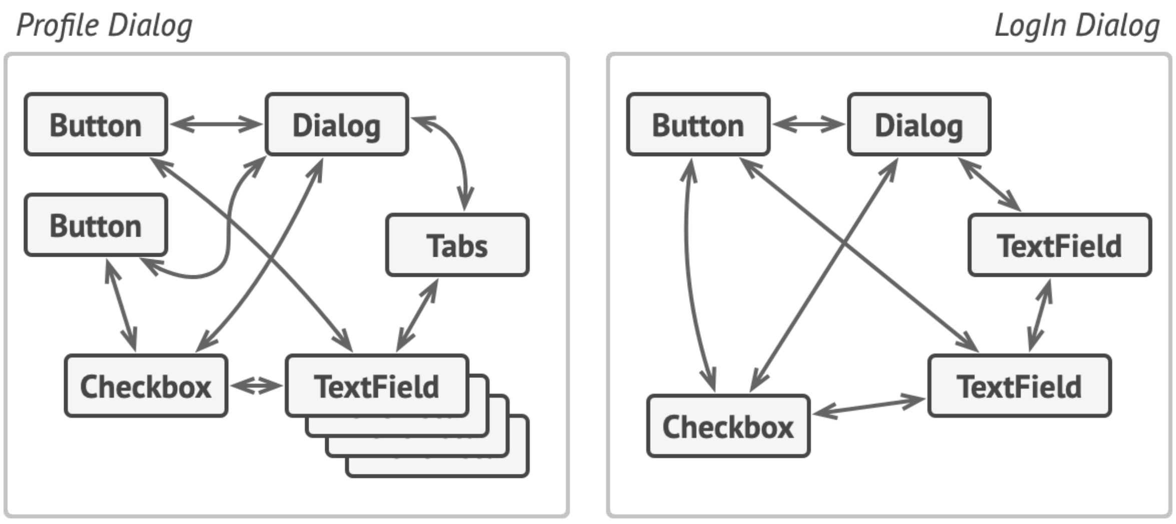

Fig: Chaotic dependency among components.

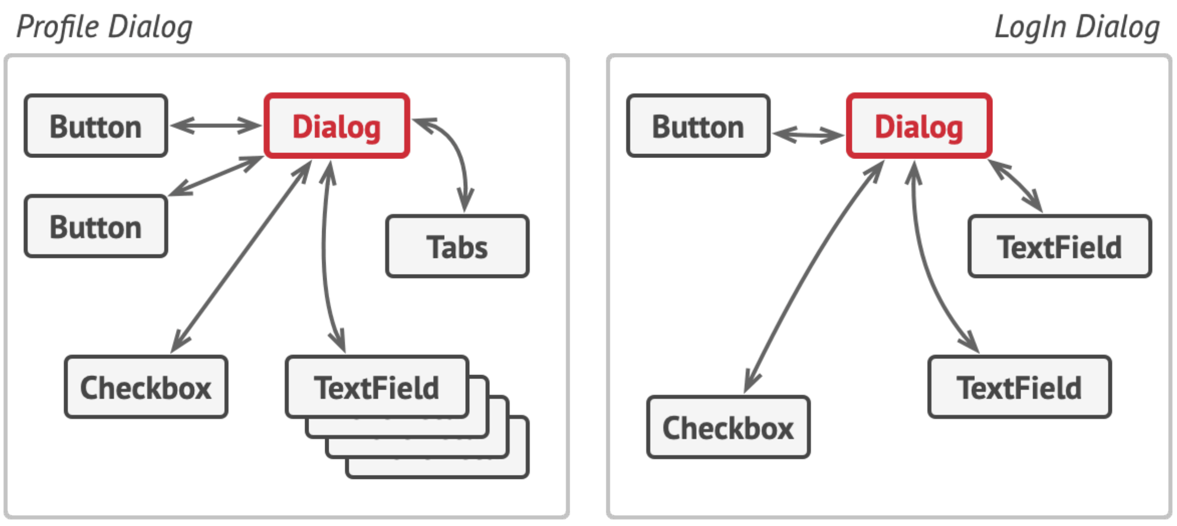

Fig: Objects communicating through the mediator component.

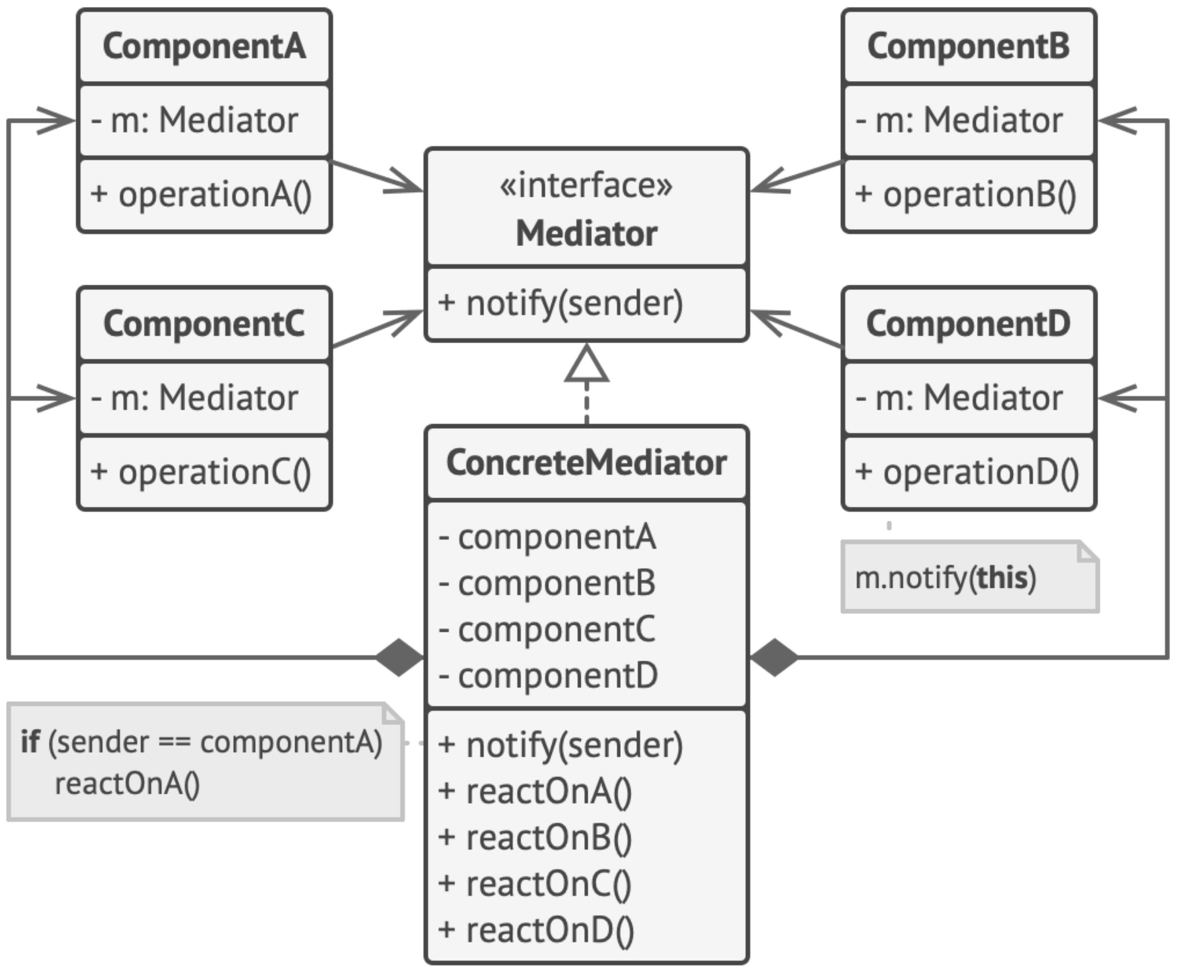

Fig: Example design of a mediator class.

Code:

📁 components

📄 **File: components/Component.java**

```java
package design_pattern.components;

import design_pattern.mediator.Mediator;

public interface Component {
    void setMediator(Mediator mediator);
    String getName();
}
```

📄 **File: components/AddButton.java**

```java
package design_pattern.components;

import design_pattern.mediator.Mediator;
import design_pattern.mediator.Note;
import javax.swing.*;
import java.awt.event.ActionEvent;

public class AddButton extends JButton implements Component {
    private Mediator mediator;
    public AddButton() {
        super("Add");
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }
    @Override
    protected void fireActionPerformed(ActionEvent actionEvent)
{
        mediator.addNewNote(new Note());
    }

    @Override
    public String getName() {
        return "AddButton";
    }
}
```

📄 **File: components/DeleteButton.java**

```java
package design_pattern.components;

import design_pattern.mediator.Mediator;

import javax.swing.*;
import java.awt.event.ActionEvent;

public class DeleteButton extends JButton  implements
Component {
    private Mediator mediator;

    public DeleteButton() {
        super("Del");
    }

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void fireActionPerformed(ActionEvent actionEvent)
{
        mediator.deleteNote();
    }

    @Override
    public String getName() {
        return "DelButton";
    }
}
```

📄 **File: components/TextBox.java**

```
package design_pattern.components;

import design_pattern.mediator.Mediator;

import javax.swing.*;
import java.awt.event.KeyEvent;

public class TextBox extends JTextArea implements Component {
    private Mediator mediator;

    @Override
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    protected void processComponentKeyEvent(KeyEvent keyEvent)
{
        mediator.markNote();
    }
```

```java
    @Override
    public String getName() {
        return "TextBox";
    }
}
```

📁**mediator**

📄 **mediator/Note.java**

```java
package design_pattern.mediator;

public class Note {
    private String name;
    private String text;

    public Note() {
        name = "New note";
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getName() {
        return name;
    }

    public String getText() {
        return text;
    }
    @Override
    public String toString() {
        return name;
    }
}
```

📄 **mediator/Mediator.java**

```java
package design_pattern.mediator;

import design_pattern.components.Component;

import javax.swing.*;

public interface Mediator {
    void addNewNote(Note note);
    void deleteNote();
    void clear();
    void registerComponent(Component component);
    void hideElements(boolean flag);
    void createGUI();
}
```

📄 **mediator/Editor.java**

```java
package design_pattern.mediator;

import design_pattern.components.*;
import design_pattern.components.Component;
import design_pattern.components.List;

import javax.swing.*;
import javax.swing.border.LineBorder;
import java.awt.*;
```

```java
public class Editor implements Mediator {
    private TextBox textBox;
    private AddButton add;
    private DeleteButton del;

    private JLabel titleLabel = new JLabel("Title:");
    private JLabel textLabel = new JLabel("Text:");
    private JLabel label = new JLabel("Add note to proceed.");

    @Override
    public void registerComponent(Component component) {
        component.setMediator(this);
        switch (component.getName()) {
            case "AddButton":
                add = (AddButton)component;
                break;
            case "DelButton":
                del = (DeleteButton)component;
                break;
            case "TextBox":
                textBox = (TextBox)component;
                break;
        }
    }

    @Override
    public void addNewNote(Note note) {
        textBox.setText("");
    }

    @Override
    public void deleteNote() {
        textBox.deleteElement();
    }

    @Override
    public void getInfoFromList(Note note) {
        textBox.setText(note.getText());
    }

    @Override
    public void clear() {
        textBox.setText("");
    }

    @Override
    public void hideElement(boolean flag) {
        textBox.setVisible(!flag);
    }

    @Override
    public void createGUI() {
        JFrame notes = new JFrame("Notes");
        notes.setSize(960, 600);
        notes.setVisible(true);
    }
}
```

📁 /

📄 **Demo.java:** Initialization code

```
package design_pattern;

import design_pattern.components.*;
import design_pattern.mediator.Editor;
import design_pattern.mediator.Mediator;
```

```java
import javax.swing.*;

public class Demo {
    public static void main(String[] args) {
        Mediator mediator = new Editor();

        mediator.registerComponent(new TextBox());
        mediator.registerComponent(new AddButton());
        mediator.registerComponent(new DeleteButton());

        mediator.createGUI();
    }
}
```

Memento

Memento is a behavioral design pattern that saves and restores the previous state of an object without revealing the details of its implementation.

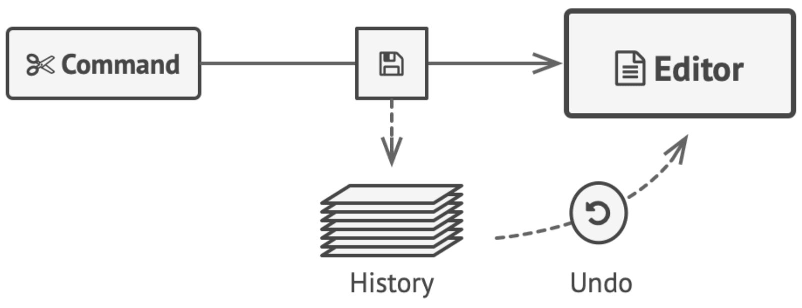

Fig: Application of Memento design pattern.

Code:

📁 memento

📄 **memento/Memento.java**

```java
package design_pattern.memento;

public class Memento {
   private String state;

   public Memento(String state){
       this.state = state;
   }

   public String getState(){
       return state;
   }
}
```

📄 **memento/Originator.java**

```java
package design_pattern.memento;

public class Originator {
   private String state;

   public void setState(String state){
       this.state = state;
   }

   public String getState(){
       return state;
   }

   public Memento saveStateToMemento(){
       return new Memento(state);
   }

   public void getStateFromMemento(Memento memento){
       state = memento.getState();
   }
}
```

📄 **memento/CareTaker.java**

```java
package design_pattern.memento;

import java.util.ArrayList;
import java.util.List;

public class CareTaker {
    private List<Memento> mementoList = new
ArrayList<Memento>();

    public void add(Memento state){
        mementoList.add(state);
    }

    public Memento get(int index){
        return mementoList.get(index);
    }
}
```

📄 **memento/MementoApplication.java**

```java
package design_pattern.memento;

public class MementoApplication {

    public static void main(String[] args) {
        Originator originator = new Originator();
        CareTaker careTaker = new CareTaker();
        originator.setState("State #1");
        originator.setState("State #2");
        careTaker.add(originator.saveStateToMemento());

        originator.setState("State #3");
        careTaker.add(originator.saveStateToMemento());

        originator.setState("State #4");
        System.out.println("Current State: " +
originator.getState());

        originator.getStateFromMemento(careTaker.get(0));
        System.out.println("First saved State: " +
originator.getState());
        originator.getStateFromMemento(careTaker.get(1));
        System.out.println("Second saved State: " +
originator.getState());
    }
}
```