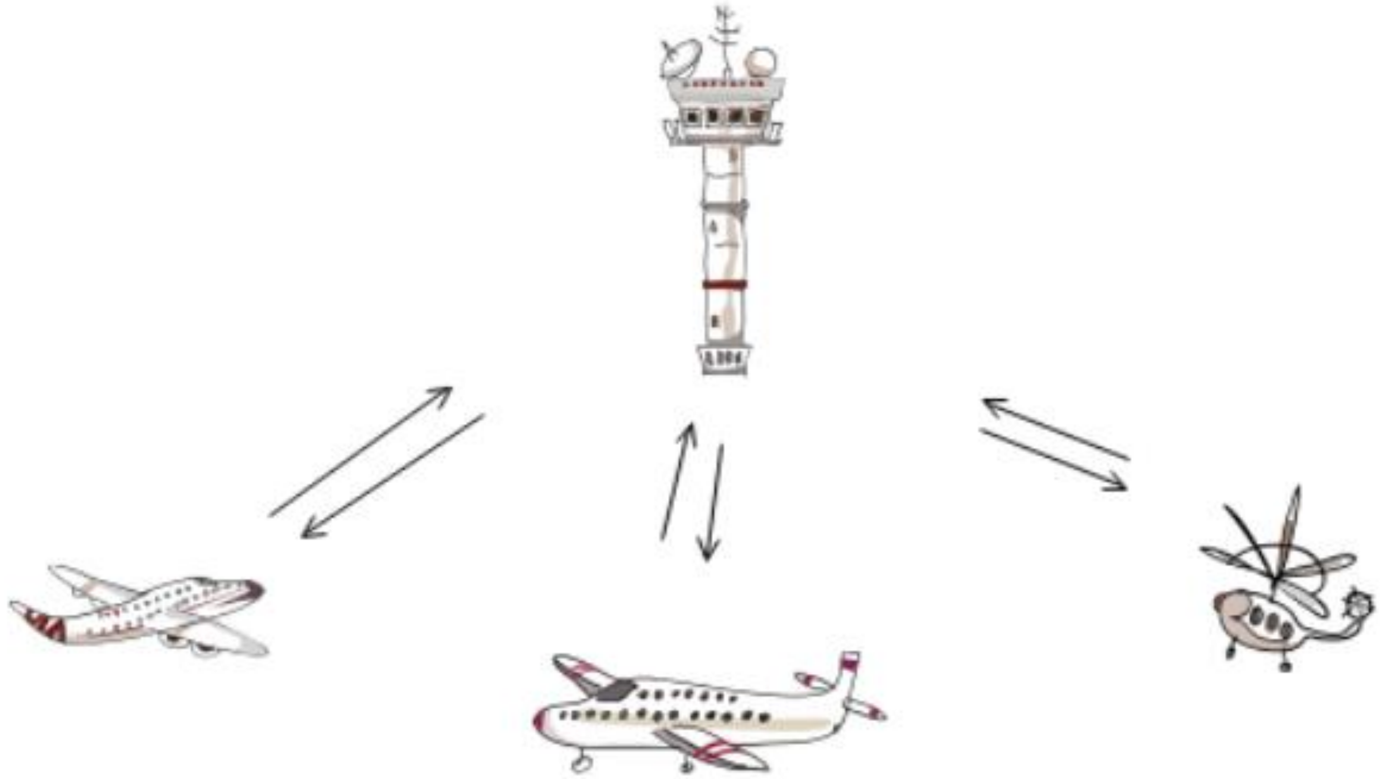
# Mediator Pattern

2022

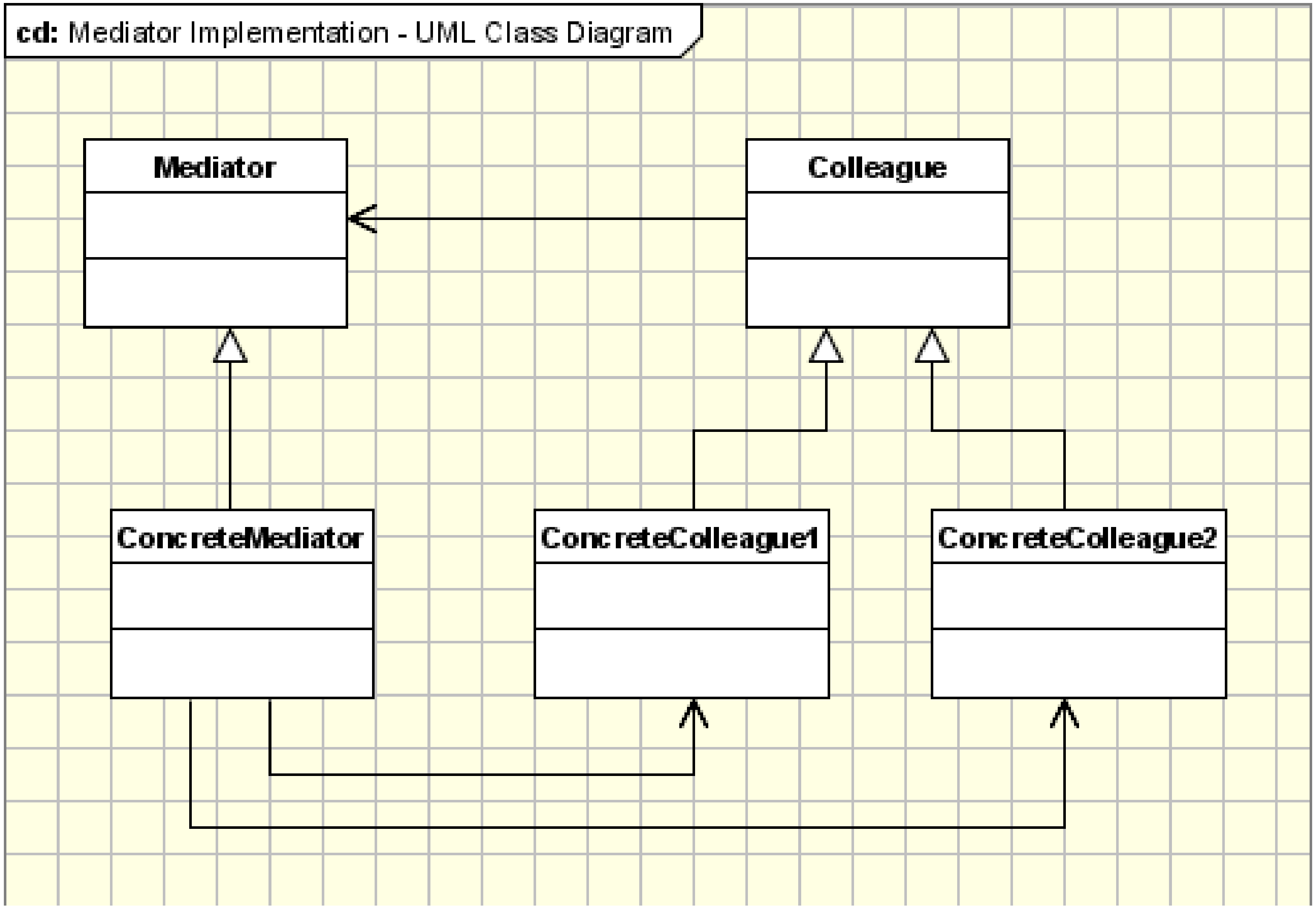
The mediator pattern defines an object that encapsulates how a set of objects interact.

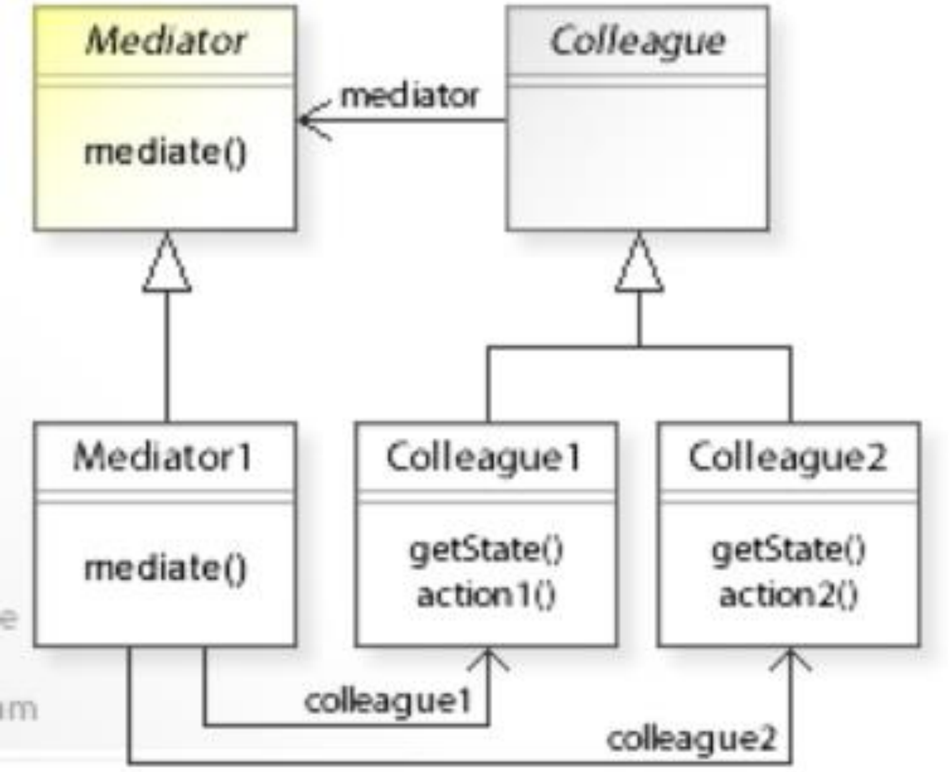
With the mediator pattern, **communication** between objects is encapsulated within a mediator object. Objects no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing coupling.

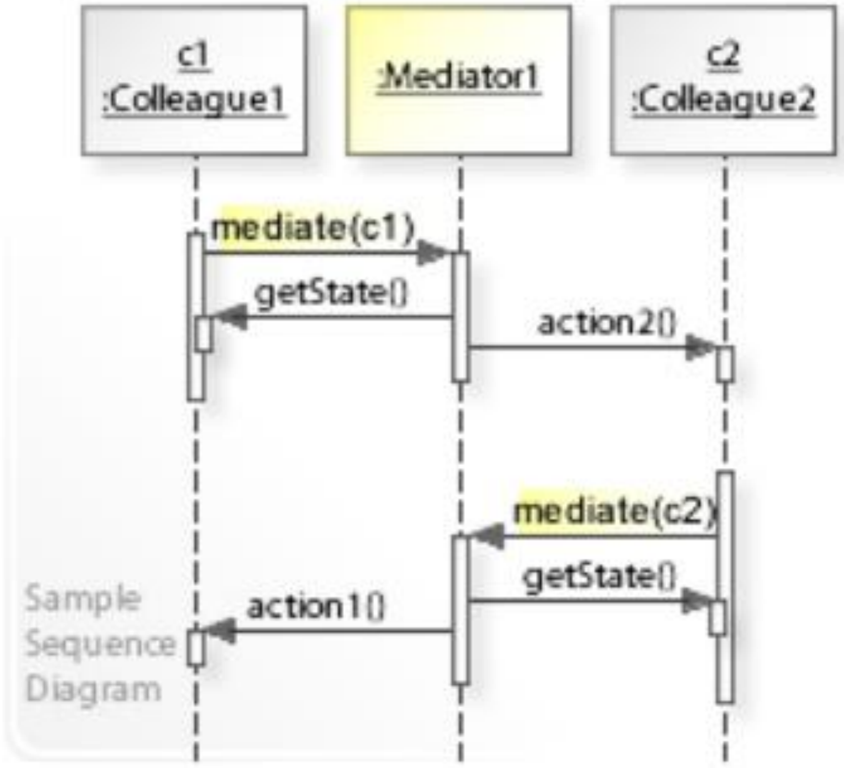
## ATC Mediator



# Mediator Pattern







https://en.wikipedia.org/wiki/Mediator_pattern

# Chatting Mediator

```
public class ChatClient {
    public static void main(String[] args) {
        ChatMediator mediator = new ChatMediatorImpl();
        User user1 = new UserImpl( mediator, "Pankaj");
        User user2 = new UserImpl( mediator, "Lisa");
        User user3 = new UserImpl( mediator, "Saurabh");
        User user4 = new UserImpl( mediator, "David");
        mediator.addUser(user1);
        mediator.addUser(user2);
        mediator.addUser(user3);
        mediator.addUser(user4);
        user1.send("Hi All, this is Panka");
        user2.send("Hi this is Lisa");
    }
}
```

```
import java.util.ArrayList;
import java.util.List;
public class ChatMediatorImpl implements ChatMediator {
    private List<User> users;
    public ChatMediatorImpl(){
        this.users=new ArrayList<>();
    }
    @Override
    public void addUser(User user){
        this.users.add(user);
    }
    @Override
    public void sendMessage(String msg, User user) {
        for(User u : this.users){
            //message should not be received by the user sending it
            if(u != user){
                u.receive(msg);
            }
        }
    }
}
```

```
public interface ChatMediator {
    public void sendMessage(String msg, User user);
    void addUser(User user);
}
```

```
public abstract class User {
    protected ChatMediator mediator;
    protected String name;
    public User( ChatMediator med, String name){
        this.mediator=med;
        this.name=name;
    }
    public abstract void send(String msg);
    public abstract void receive(String msg);
}
```

```
public class UserImpl extends User {
    public UserImpl(ChatMediator med, String name) {
        super(med, name);
    }
    @Override
    public void send(String msg){
        System.out.println(this.name+": Sending Message="+msg);
        mediator.sendMessage(msg, this);
    }
    @Override
    public void receive(String msg) {
        System.out.println(this.name+": Received Message:"+msg);
    }
}
```

```
Pankaj: Sending Message=Hi All, this is Panka
Lisa: Received Message:Hi All, this is Panka
Saurabh: Received Message:Hi All, this is Panka
David: Received Message:Hi All, this is Panka
Lisa: Sending Message=Hi this is Lisa
Pankaj: Received Message:Hi this is Lisa
Saurabh: Received Message:Hi this is Lisa
David: Received Message:Hi this is Lisa
```

# Buffer Mediator

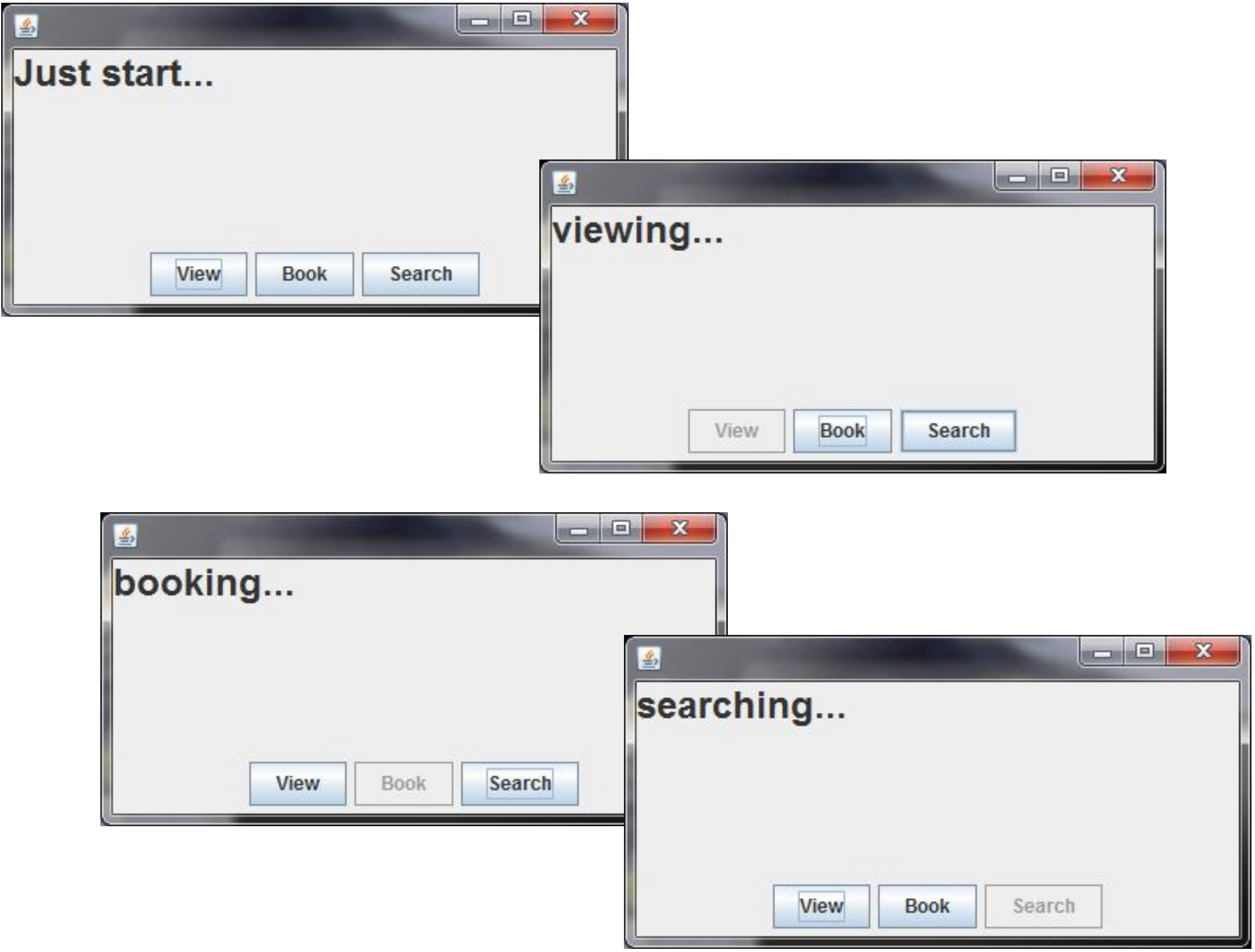
synchronized

```
class Mediator {
private boolean slotFull = false;
private int number;
public synchronized void storeMessage( int num ) {
// no room for another message
   while (slotFull == true) {
     try {
         wait();
       }
      catch (InterruptedException e ) { }
    }
    slotFull = true;
    number = num;
    notifyAll();
 }
public synchronized int retrieveMessage() {
 // no message to retrieve
  while (slotFull == false)
     try {
        wait();
       }
      catch (InterruptedException e ) { }
    slotFull = false;
    notifyAll();
    return number;
   }
}
```

```
class Producer extends Thread {
  // 2. Producers are coupled only to the Mediator
  private Mediator med;
  private int    id;
  private static int num = 1;
  public Producer( Mediator m ) {
    med = m;
    id = num++;
  }
  public void run() {
    int num;
    while (true) {
      med.storeMessage( num = (int)(Math.random()*100) );
      System.out.print( "p" + id + "-" + num + "   " );
    }
  }
}
```

```
public class Consumer extends Thread {
        // 3. Consumers are coupled only to the Mediator
        private Mediator med;
        private int    id;
        private static int num = 1;
        public Consumer( Mediator m ) {
          med = m;
          id = num++;
        }
        public void run() {
          while (true) {
            System.out.print("c" + id + "-" +
                              med.retrieveMessage() + "  ");
          }
        }
      }
```

```
public class MediatorDemo {
public static void main( String[] args ) {
    Mediator mb = new Mediator();
    new Producer( mb ).start();
    new Producer( mb ).start();
    new Consumer( mb ).start();
    new Consumer( mb ).start();
    new Consumer( mb ).start();
    new Consumer( mb ).start();
  }
}
```


Just start...

viewing...

booking...

searching...

## Library Mediator

```
class Mediator {

    BtnView btnView;
    BtnSearch btnSearch;
    BtnBook btnBook;
    LblDisplay show;

    void registerView(BtnView v) {
        btnView = v;
    }
    void registerSearch(BtnSearch s) {
        btnSearch = s;
    }
    void registerBook(BtnBook b) {
        btnBook = b;
    }
    void registerDisplay(LblDisplay d) {
        show = d;
    }
    void book() {
        btnBook.setEnabled(false);
        btnView.setEnabled(true);
        btnSearch.setEnabled(true);
        show.setText("booking...");
    }
    void view() {
        btnView.setEnabled(false);
        btnSearch.setEnabled(true);
        btnBook.setEnabled(true);
        show.setText("viewing...");
    }
    void search() {
        btnSearch.setEnabled(false);
        btnView.setEnabled(true);
        btnBook.setEnabled(true);
        show.setText("searching...");
    }
}
```

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

class LblDisplay extends JLabel {

    Mediator med;

    LblDisplay(Mediator m) {
        super("Just start...");
        med = m;
        med.registerDisplay(this);
        setFont(new Font("Arial", Font.BOLD, 24));
    }

}
```

```
interface Command {
    void execute();
}
```

```
class BtnView extends JButton implements
Command {
    Mediator med;
    BtnView(ActionListener al, Mediator m) {
        super("View");
        addActionListener(al);
        med = m;
        med.registerView(this);
    }
    public void execute() {
        med.view();
    }
}
```

```
interface Command {
    void execute();
}
```

```
class BtnBook extends JButton implements Command {
    Mediator med;
    BtnBook(ActionListener al, Mediator m) {
        super("Book");
        addActionListener(al);
        med = m;
        med.registerBook(this);
    }
    public void execute() {
        med.book();
    }
}
```

```
class MediatorDemo extends JFrame implements ActionListener {
    Mediator med = new Mediator();
    MediatorDemo() {
        JPanel p = new JPanel();
        p.add(new BtnView(this, med));
        p.add(new BtnBook(this, med));
        p.add(new BtnSearch(this, med));
        getContentPane().add(new LblDisplay(med), "North");
        getContentPane().add(p, "South");
        setSize(400, 200);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
        Command comd = (Command) ae.getSource();
        comd.execute();
    }
    public static void main(String[] args) {
        new MediatorDemo();
    }
}
```