Mediator (中介者/调停者, Behavioral Pattern)

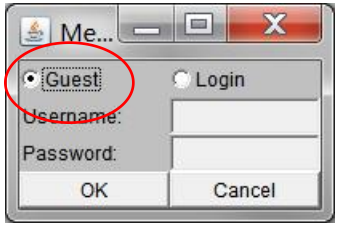
Kai SHI

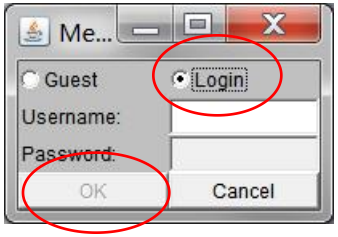
Intent

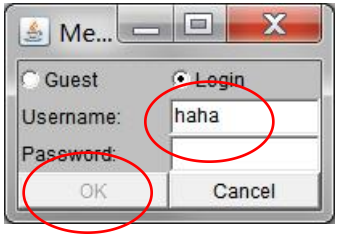
- Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
- (中介者/调停者对象封装了一系列对象相互作用的方式，使得这些对象不必互相明显引用。从而使它们可以较松散地耦合。当这些对象中的某些对象之间的相互作用发生改变时，不会立即影响到其他的一些对象之间的相互作用。从而保证这些相互作用可以彼此独立地变化。)

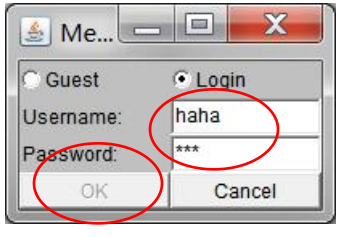
Motivation (1/2)

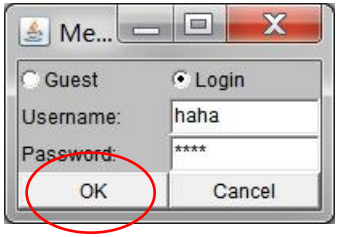
- Object-oriented design encourages the distribution of behavior among objects. Such distribution can result in an object structure with many connections between objects; in the worst case, every object ends up knowing about every other.
- Example Code: mediator.Main



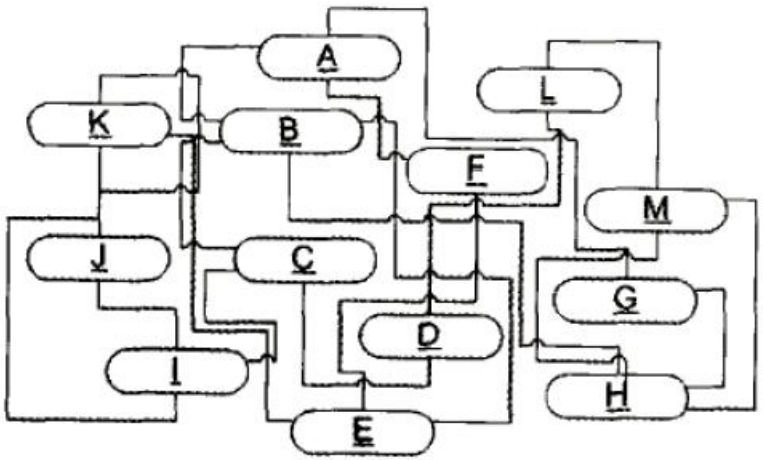


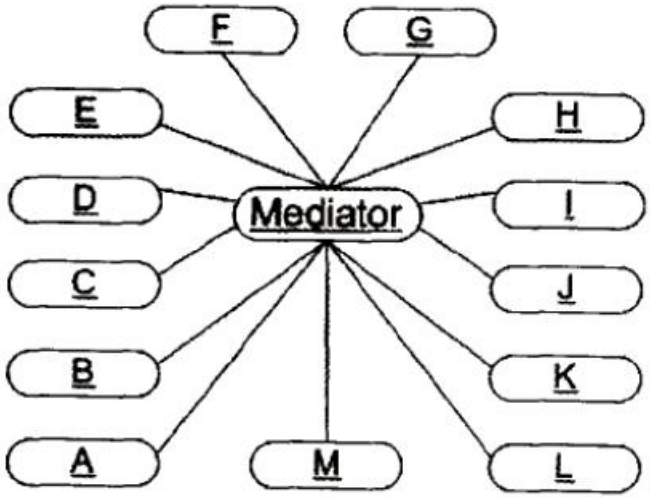






Motivation (2/2)

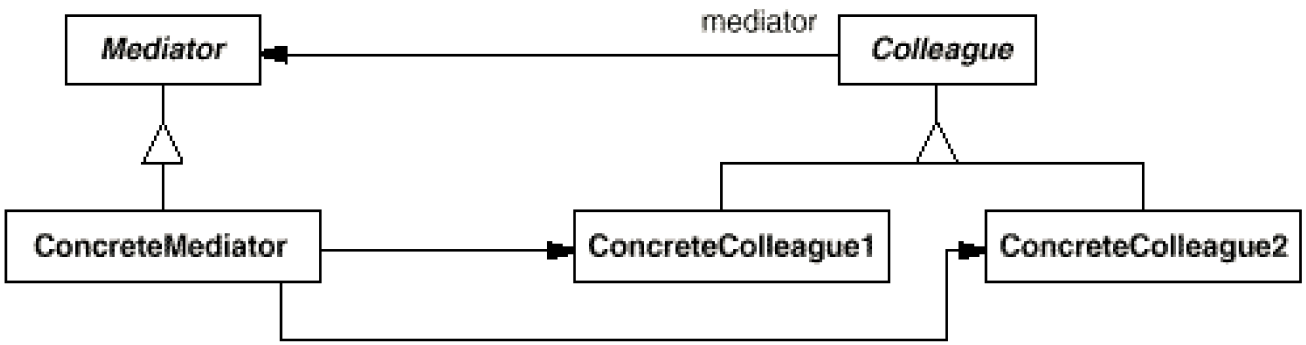




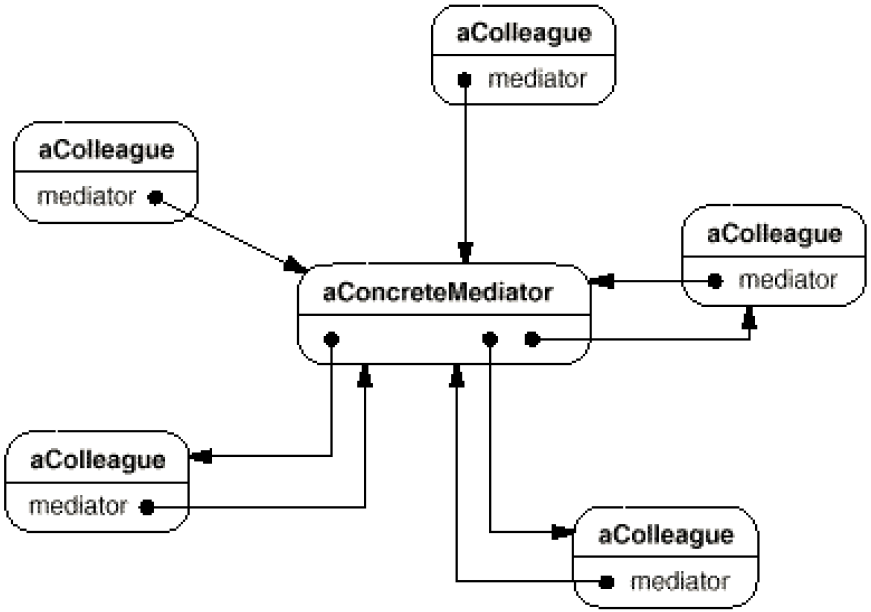
Applicability: Use the Mediator pattern when

- a set of objects communicate in well-defined but complex ways. The resulting interdependencies are unstructured and difficult to understand.
- reusing an object is difficult because it refers to and communicates with many other objects.
- a behavior that's distributed between several classes should be customizable without a lot of subclassing.

Structure



- A typical object structure might look like this:



Participants

- Mediator: Defines an interface for communicating with Colleague objects.
- ConcreteMediator: Implements cooperative behavior by coordinating Colleague objects. knows and maintains its colleagues.
- Colleague classes:
  - Each Colleague class knows its Mediator object.
  - Each Colleague communicates with its mediator whenever it would have otherwise communicated with another colleague. (colleague间的通信一定要通过mediator)

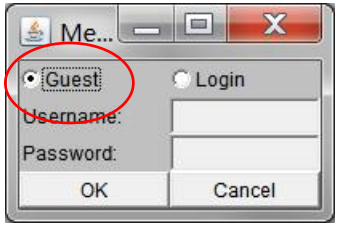
Collaborations

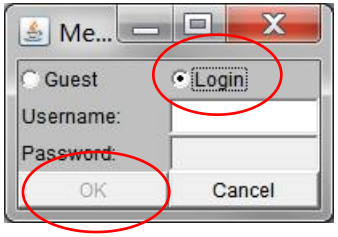
- Colleagues send and receive requests from a Mediator object.
- The mediator implements the cooperative behavior by routing requests between the appropriate colleague(s).

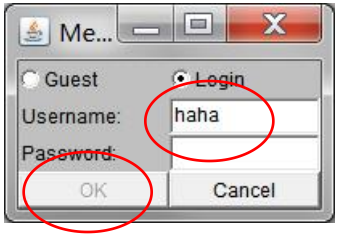
Consequences

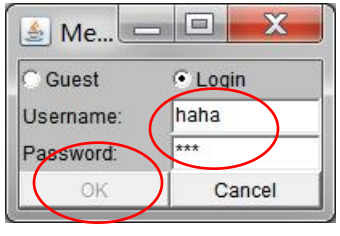
- It decouples colleagues.
- It simplifies object protocols.
- It abstracts how objects cooperate.
- It centralizes control.
  - The Mediator pattern trades complexity of interaction for complexity in the mediator. Because a mediator encapsulates protocols, it can become more complex than any individual colleague. This can make the mediator itself a monolith (巨无霸) that's hard to maintain.

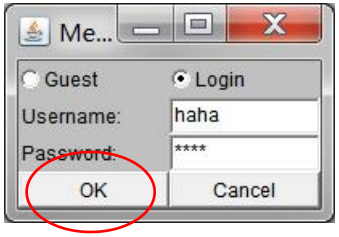
Back to The Problem



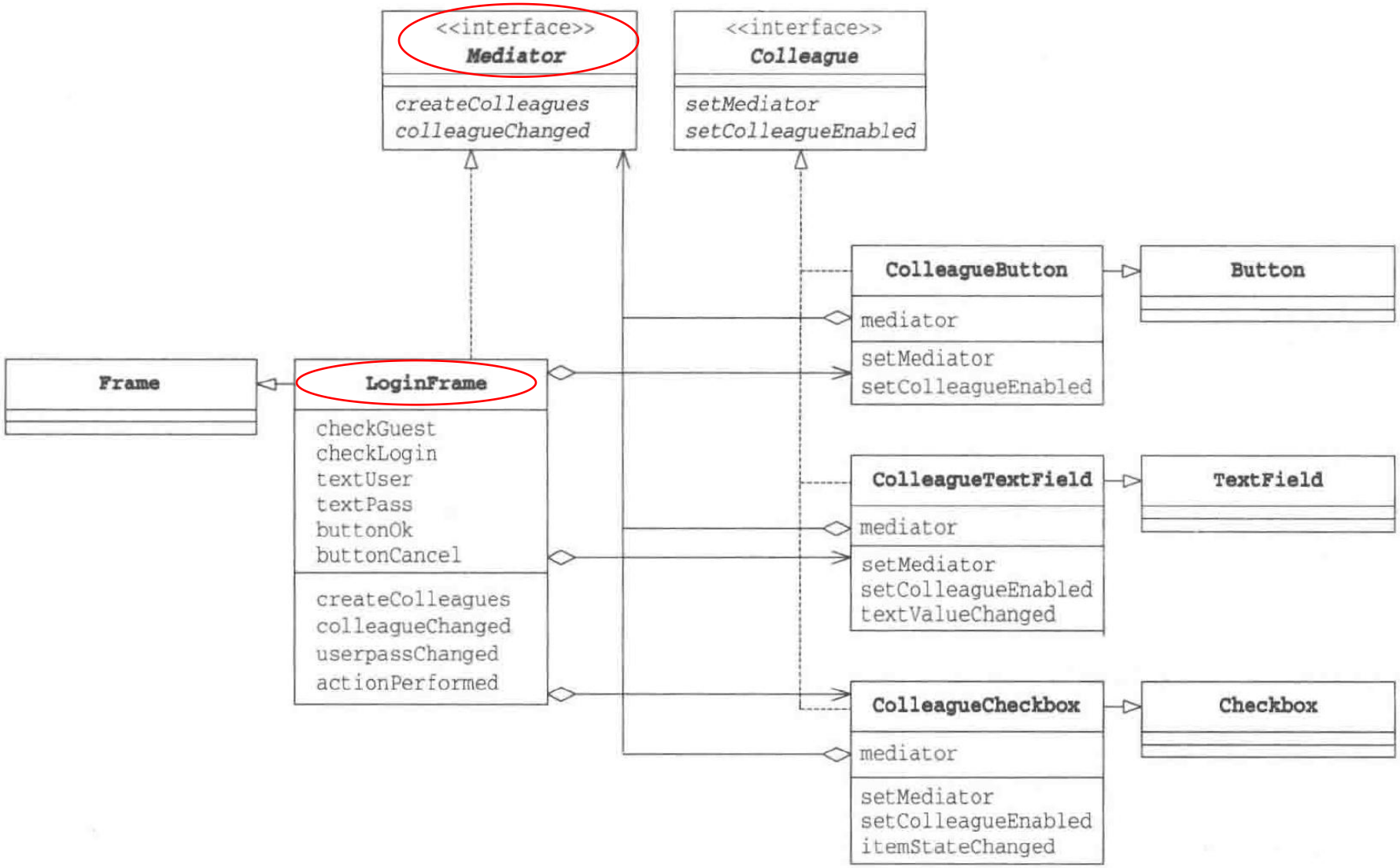




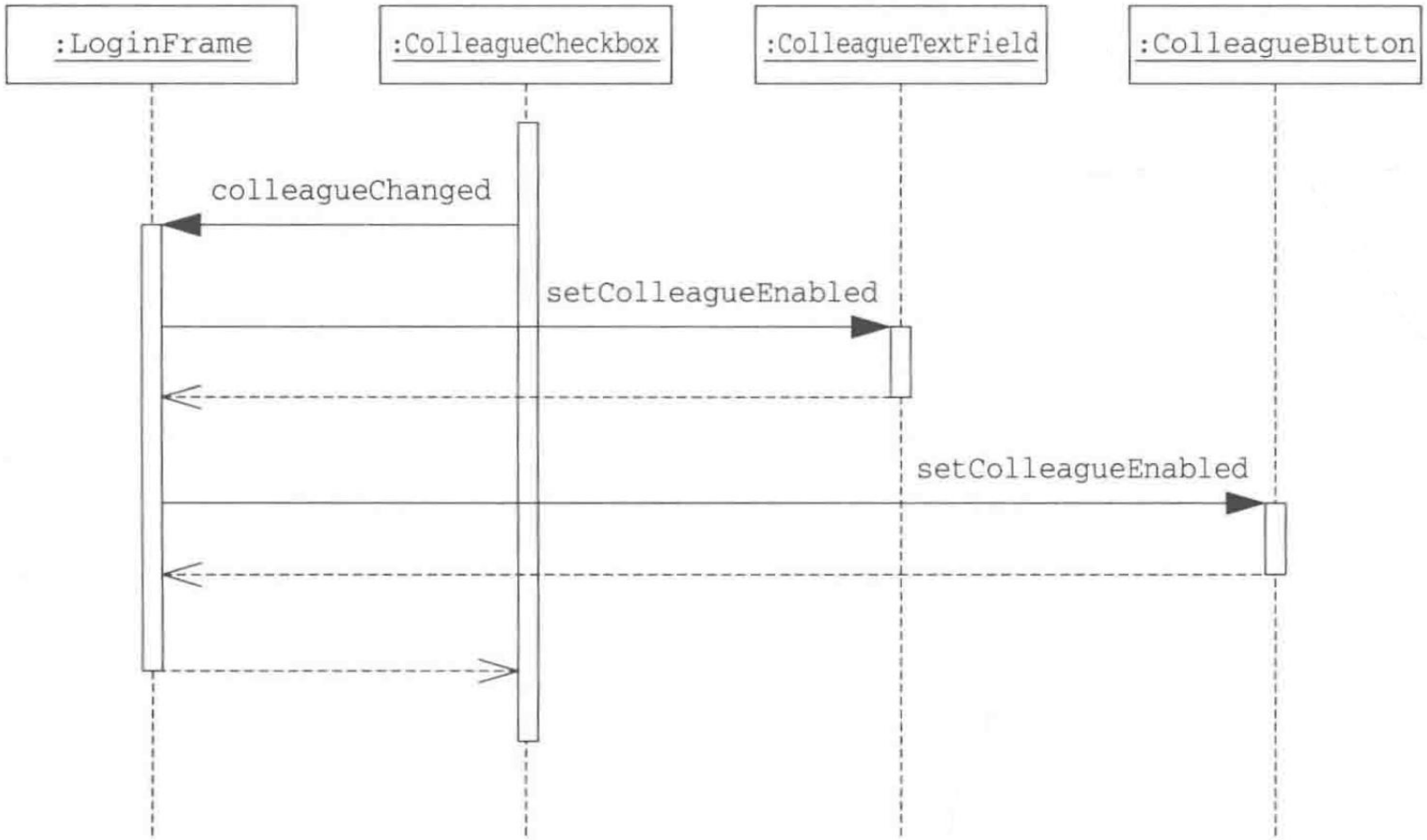




Class Diagram



Sequence Diagram



Interface Mediator

The Mediator creates all Colleagues, which the mediator manages.

```
public interface Mediator {
    public abstract void createColleagues();
    public abstract void colleagueChanged();
}
```

The Colleagues use the method to report to the Mediator.

Interface Colleague

```
public interface Colleague {
    public abstract void setMediator(Mediator mediator);
    public abstract void setColleagueEnabled(boolean enabled);
}
```

The Mediator use the method to give directions to the Colleagure.

Concrete Colleague: ColleagueCheckbox

```
public class ColleagueCheckbox extends Checkbox implements ItemListener, Colleague {
    private Mediator mediator;
    public ColleagueCheckbox(String caption, CheckboxGroup group, boolean state) {  // 构造函数
        super(caption, group, state);
    }
    public void setMediator(Mediator mediator) {            // 保存Mediator
        this.mediator = mediator;
    }
    public void setColleagueEnabled(boolean enabled) {      // Mediator下达启用/禁用指示
        setEnabled(enabled);
    }
    public void itemStateChanged(ItemEvent e) {             // 当状态发生变化时通知Mediator
        mediator.colleagueChanged();
    }
}
```

Concrete Colleague: ColleagueTextField

```
public class ColleagueTextField extends TextField implements TextListener, Colleague {
    private Mediator mediator;
    public ColleagueTextField(String text, int columns) {    // 构造函数
        super(text, columns);
    }
    public void setMediator(Mediator mediator) {             // 保存Mediator
        this.mediator = mediator;
    }
    public void setColleagueEnabled(boolean enabled) {       // Mediator下达启用/禁用的指示
        setEnabled(enabled);
        setBackground(enabled ? Color.white : Color.lightGray);
    }
    public void textValueChanged(TextEvent e) {              // 当文字发生变化时通知Mediator
        mediator.colleagueChanged();
    }
}
```

Concrete Colleague: ColleagueButton

```
public class ColleagueButton extends Button implements Colleague {
    private Mediator mediator;
    public ColleagueButton(String caption) {
        super(caption);
    }
    public void setMediator(Mediator mediator) {            // 保存Mediator
        this.mediator = mediator;
    }
    public void setColleagueEnabled(boolean enabled) {      // Mediator下达启用/禁用的指示
        setEnabled(enabled);
    }
}
```

Concrete Mediator: LoginFrame

```
public class LoginFrame extends Frame implements ActionListener, Mediator {
    private ColleagueCheckbox checkGuest;
    private ColleagueCheckbox checkLogin;
    private ColleagueTextField textUser;
    private ColleagueTextField textPass;
    private ColleagueButton buttonOk;
    private ColleagueButton buttonCancel;

    // 构造函数。
    // 生成并配置各个Colleague后，显示对话框。
    public LoginFrame(String title) {

    // 生成各个Colleague。
    public void createColleagues() {

    // 接收来自于Colleage的通知并判断各Colleage的启用/禁用状态。
    public void colleagueChanged() {
    // 当textUser或是textPass文本输入框中的文字发生变化时。
    // 判断各Colleage的可启用/禁用状态。
    private void userpassChanged() {
    public void actionPerformed(ActionEvent e) {
}
```

Full code: mediator.LoginFrame

Avoid Misusing Mediator

- Mediator pattern is applied to a system for avoiding mess and ugly.
- Mediator pattern should not be applied to a system which **has been** mess and ugly.
  - Such system should be re-designed;
  - Responsibilites of classes should be repartitioned;
  - When the system is going to be mess, firstly, try to clarify the functional dependency;
  - Mediator pattern should be used to avoid the mess system, but not fix it.
  - Mediator pattern is a pattern but not a sliver bullet.