

Lecture 8. Mediator Pattern (Behavioral)

- 行为模式关心算法和对象之间的责任分配。
- 它关心的不仅仅描述对象或类的模式，而是要更加侧重描述它们之间的通信模式。
- 行为模式刻画了很难在运行时跟踪的复杂的控制流。该模式将软件开发者的注意力从控制流转移到对象相互关联的方式方面。

**Professor:
Yushan (Michael) Sun
Fall 2020**

189 0631 7629

Contents of this lecture

1. Introductory example to the mediator design pattern
2. Theory of the Mediator Pattern
3. Example design using the Mediator Pattern
4. Implementation details of the Mediator Pattern



Introductory Example to the Mediator Pattern

Introductory example to the mediator pattern

【例1】：机场指挥塔的功能

a) 指挥起飞：所有的飞机必须接到起飞的命令才能起飞

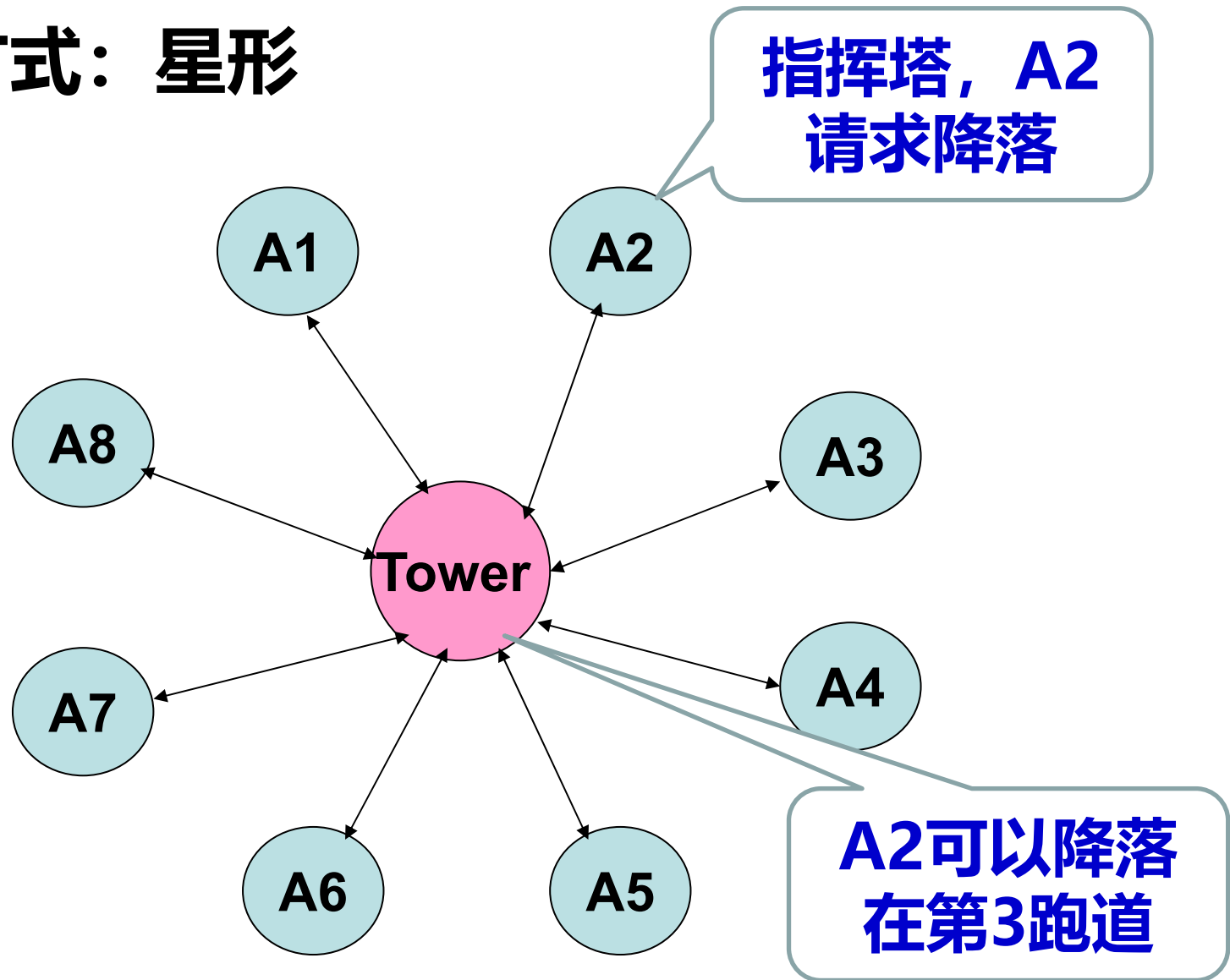
b) 指挥降落：所有的飞机必须接到降落的命令才能降落

c) 通讯：飞机和指挥塔通讯；所有的飞机不能直接通讯



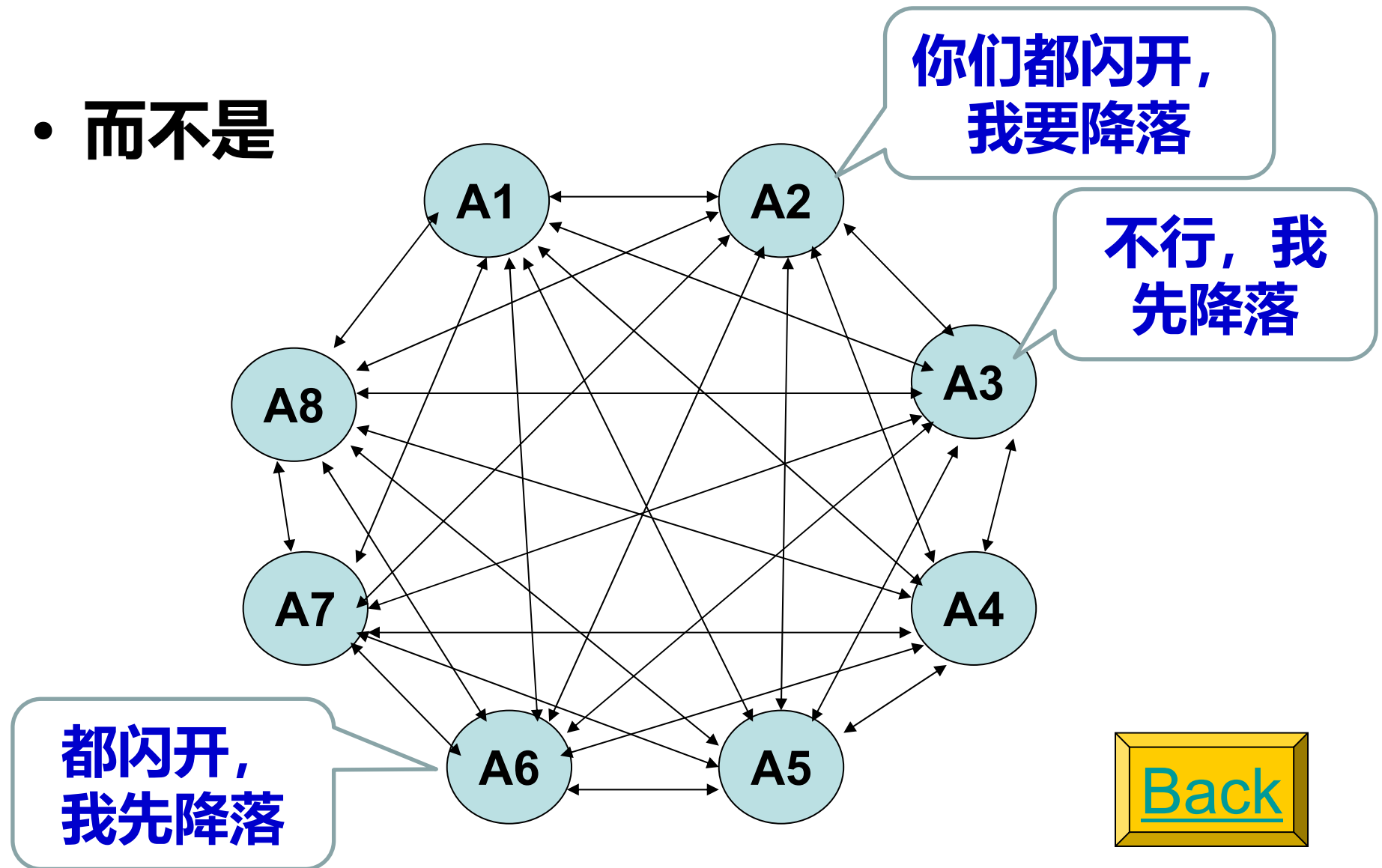
Introductory example to the mediator pattern

- 通讯方式：星形



Introductory example to the mediator pattern

- 而不是



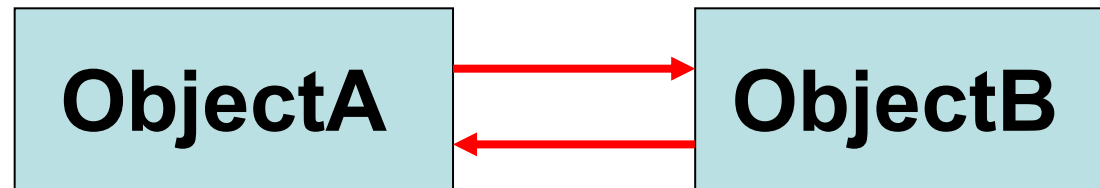
Theory of the Mediator Pattern

Mediator Pattern

DESCRIPTION

- **面向对象应用程序由一些互动的对象组成.**
Generally, OO applications consist of a set of objects that interact with each other for the purpose of providing a service.
- **当参与对象数目较少时，对象之间可以直接交互.**
This interaction can be direct (point-to-point) as long as the number of objects referring to each other directly is very low.

Mediator Pattern

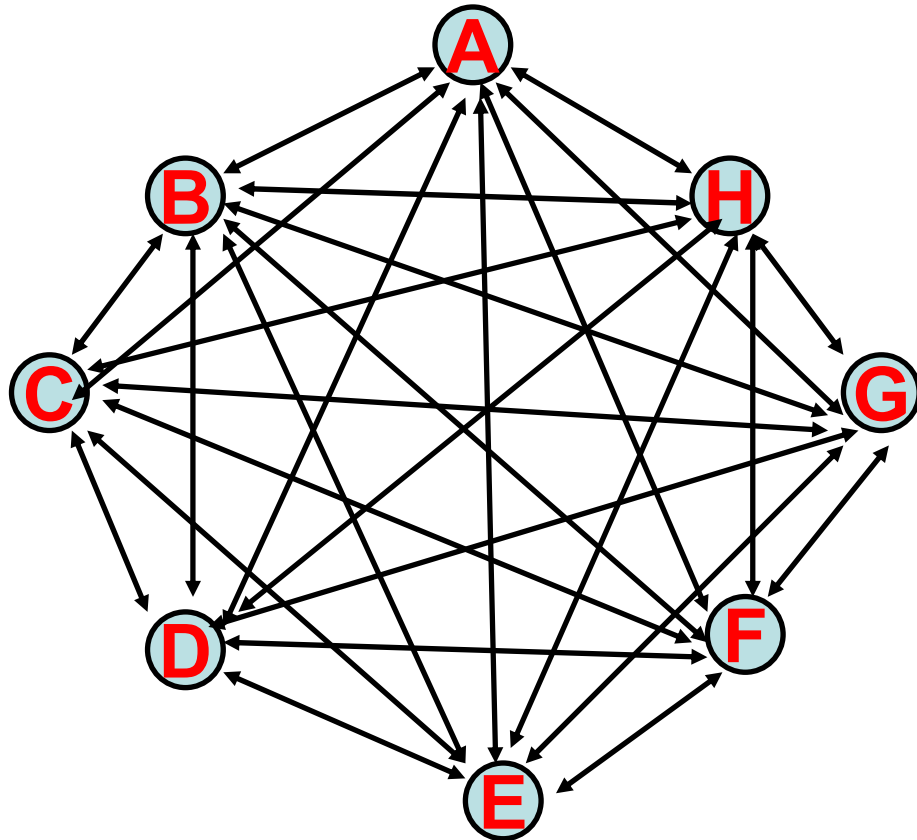


我们经常这样做，例如

- **在策略模式和状态模式里面，Context类的对象与Strategy层次类对象的耦合是双向的**
- **在状态模式里面，Context类的对象与State层次类的对象之间的耦合是双向的**

Mediator Pattern

- 在OO程序中，很多的对象互相直接交互导致混乱
As the number of objects increases, this type of direct interaction can lead to a complex maze of references among objects.



High
coupling

如果要增加一个
新的类，则所有
类的接口均需要
改变

Point to point communication: increased number of objects

Mediator Pattern

Drawbacks of point to point communications

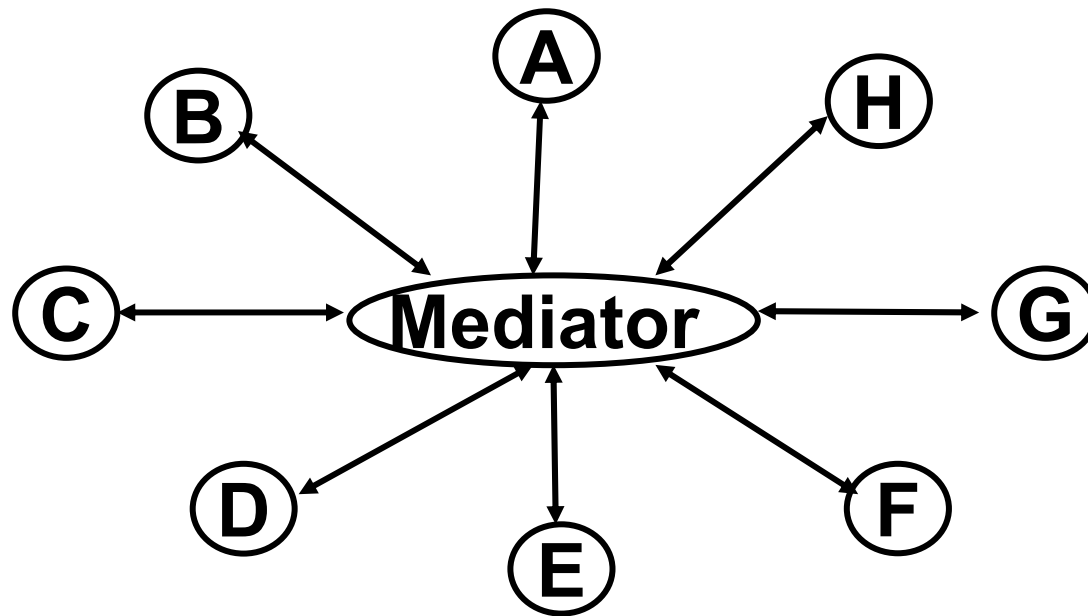
The high coupling of so many objects

- a) Results in complex method calls**
- b) affects the maintainability of the application
(如果你要增加一个新的类，就太麻烦了)**
- c) greatly reduces the scope for reusing these objects because of higher couplings**

Question: How to overcome the above problems?

Mediator Pattern

解决方案: 重新设计为星形 拓扑. To reduce the tight coupling among objects, redesign using star-like form, the mediator design pattern



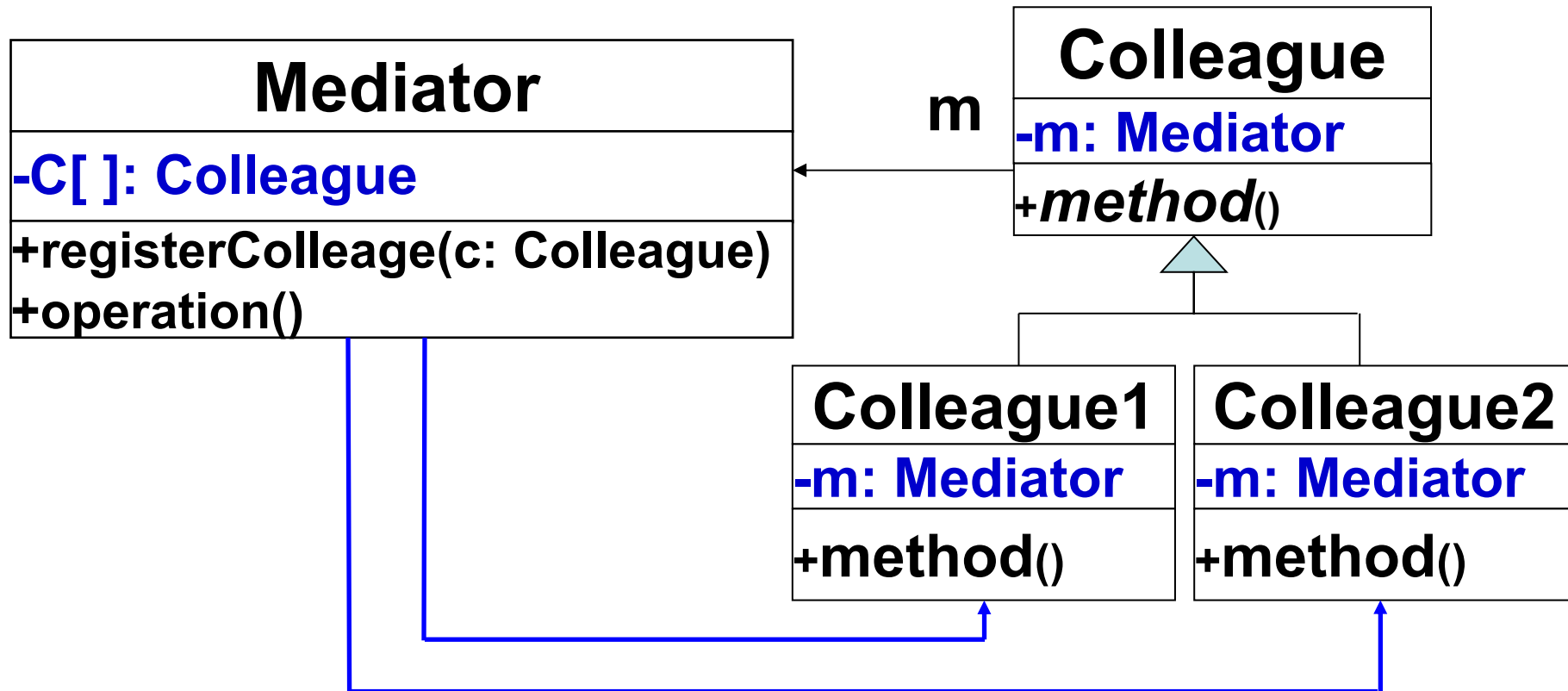
Logical Diagram of the Mediator Design Pattern

Mediator Pattern

- **问题：**怎样设计类图才能实现以上的逻辑图？
- **回答：**
 - a) Mediator类应该与参与者类（对象A, B, C, ... ,H的类）之间有某种形式的关联，
 - b) 而参与者类之间不应该有关联。

Mediator Pattern

The following class diagram represents the above logics.

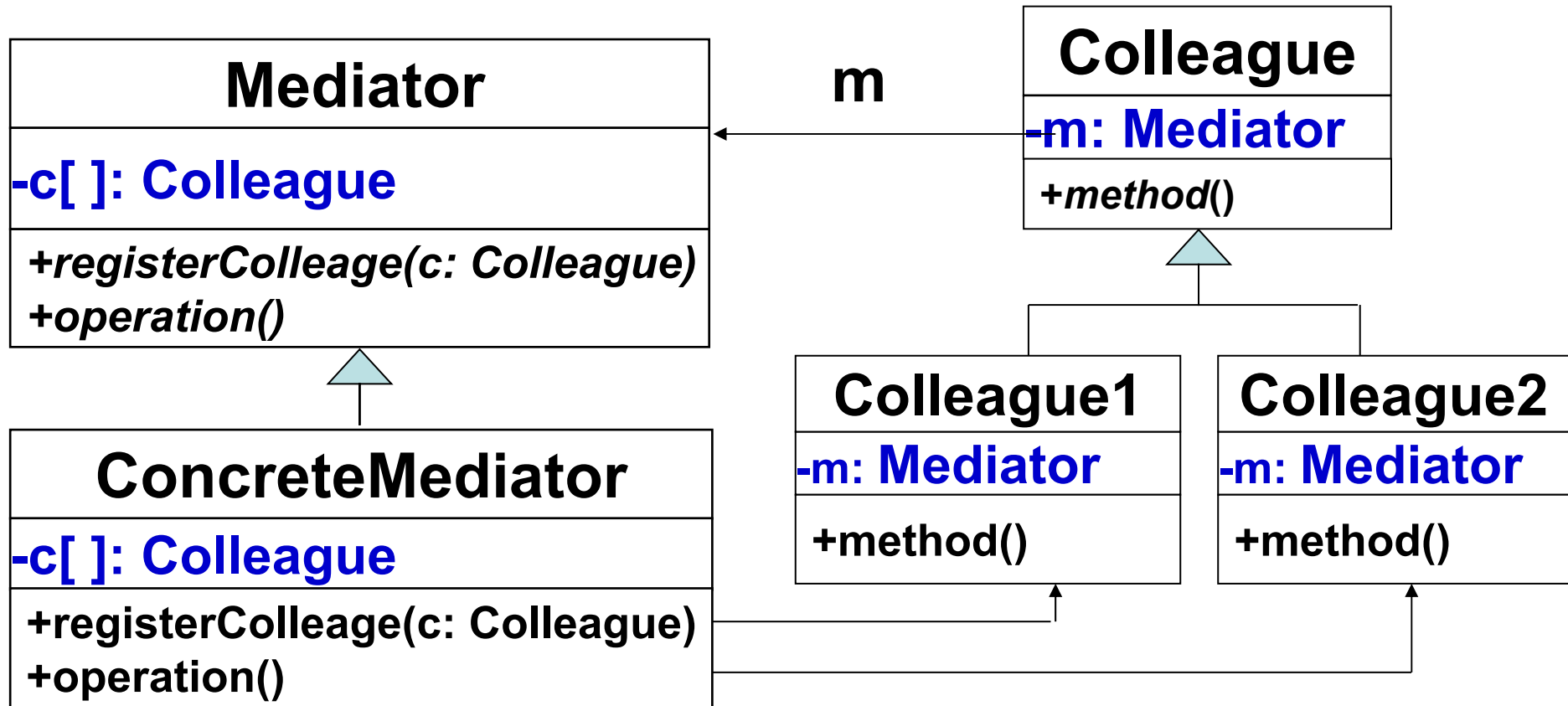


The mediator pattern with only one mediator class

Mediator Pattern

- **为什么中介者类必须拥有register方法?**
- The mediator class must know all the concrete participate colleagues, and so this class needs a register method
- The register method can be used to let the mediator class keep all the participate objects
- **为什么每个Colleague子类都必须保持中介者类的引用?** On the other hand, each participate class must keep a reference of the mediator class, so that it can call the methods in the mediator class to do something.

Mediator Pattern



The mediator Pattern

Mediator Pattern

- 该设计类图由两部分组成，一部分是中介者类，另外一部分是以上的参与者对象。
- 程序的构件说明：
- **Mediator**：中介者的接口。
- **ConcreteMediator**：具体的中介者，可以有多个具体的中介者。
- **Colleague**：参与者对象接口
- **Colleague1**, **Colleague2**：具体的参与者。可以有多个具体的参与者。

Mediator Pattern

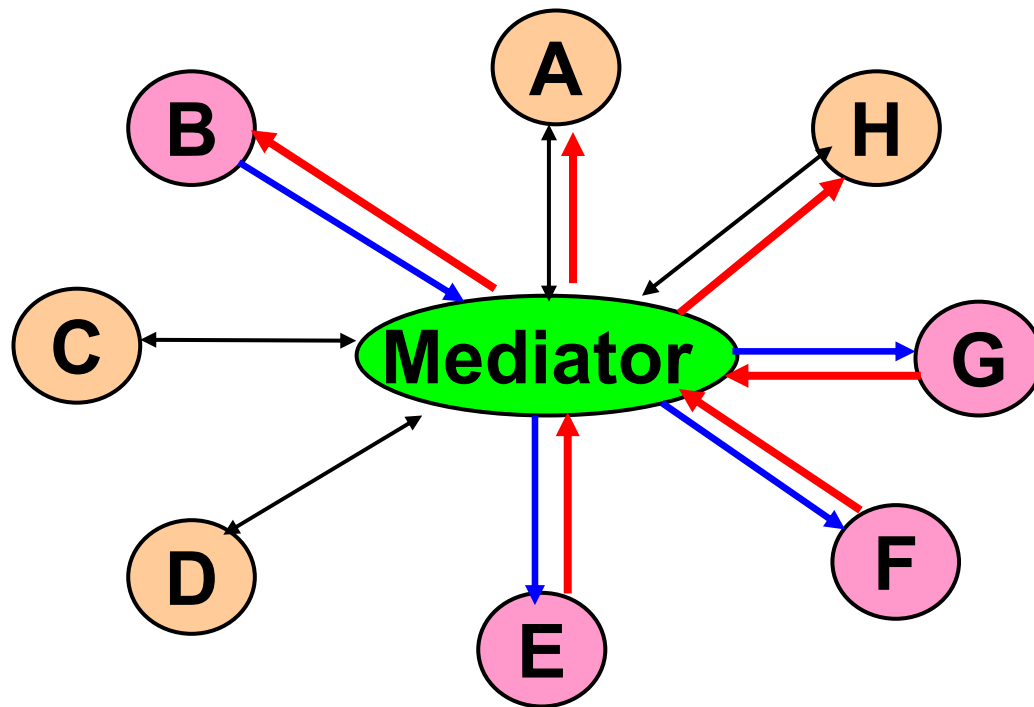
中介者模式的用途：

- We may use the mediator pattern to reduce the direct invocations among objects
- The Mediator pattern suggests abstracting all object interaction details into a separate class, a Mediator **Mediator 类 负责所有对象交互**
 - The **Mediator** will keep a reference of each of the interacting objects **维持每个参与者类的引用**
 - The **Mediator** will provide methods to call the participating objects **提供调用参与对象的方法**

Mediator Pattern

- **中介者模式的典型交互**
- **The interaction between any two different objects is routed through the Mediator class.**
 - **所有的参与者对象都可以发送消息给中介者对象。**
All objects send their messages to the mediator (call methods of Mediator)
 - **中介者对象发送消息给参与者对象实现系统功能（需求）。**
The mediator then sends messages to the appropriate objects to implement the application's requirements (in the mediator then call methods every other objects).

Mediator Pattern



1. Object B may call a method of mediator object.
2. Then mediator object may call methods in objects E, F, G.
3. Objects E, F, G may also call back the methods in Mediator object
4. Mediator object may call object B
5. Mediator object may in turn call other objects

注: Object B may not know which object has been called

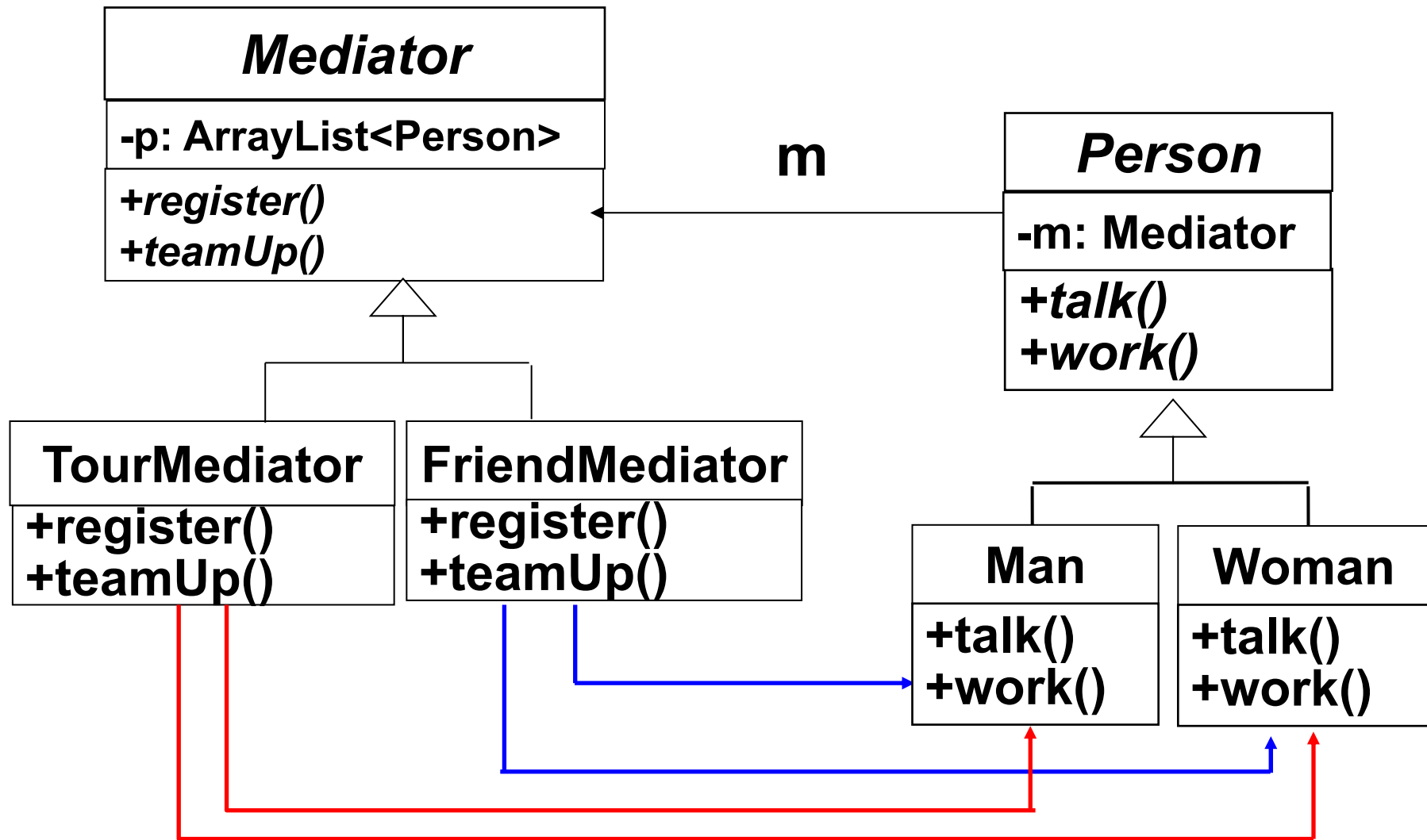
How Objects interact in the Mediator pattern?

Advantages of Mediator Pattern

Advantages of Mediator pattern:

- 1. 参与者类复用性变得更好.** Moving inter object dependencies out of individual objects results in enhanced object reusability.
- 2. 参与者对象之间的关系可以由中介者子类对象调节.** It becomes easier to alter the behavior of object inter relationships, by replacing the mediator with one of its subclasses with extended or altered functionality.

Advantages of Mediator Pattern



具有两个具体的中介者的模式的例子

Advantages of Mediator Pattern

Advantages of Mediator pattern (cont):

3. **有利于参与者类的单元测试。** Objects can be unit tested more easily, because objects do not need to refer to each other directly
4. **有利于参与者类的修改。** The low degree of coupling allows individual classes to be modified without affecting other classes.

- **Disadvantage :**
- **Using a mediator may compromise performance**



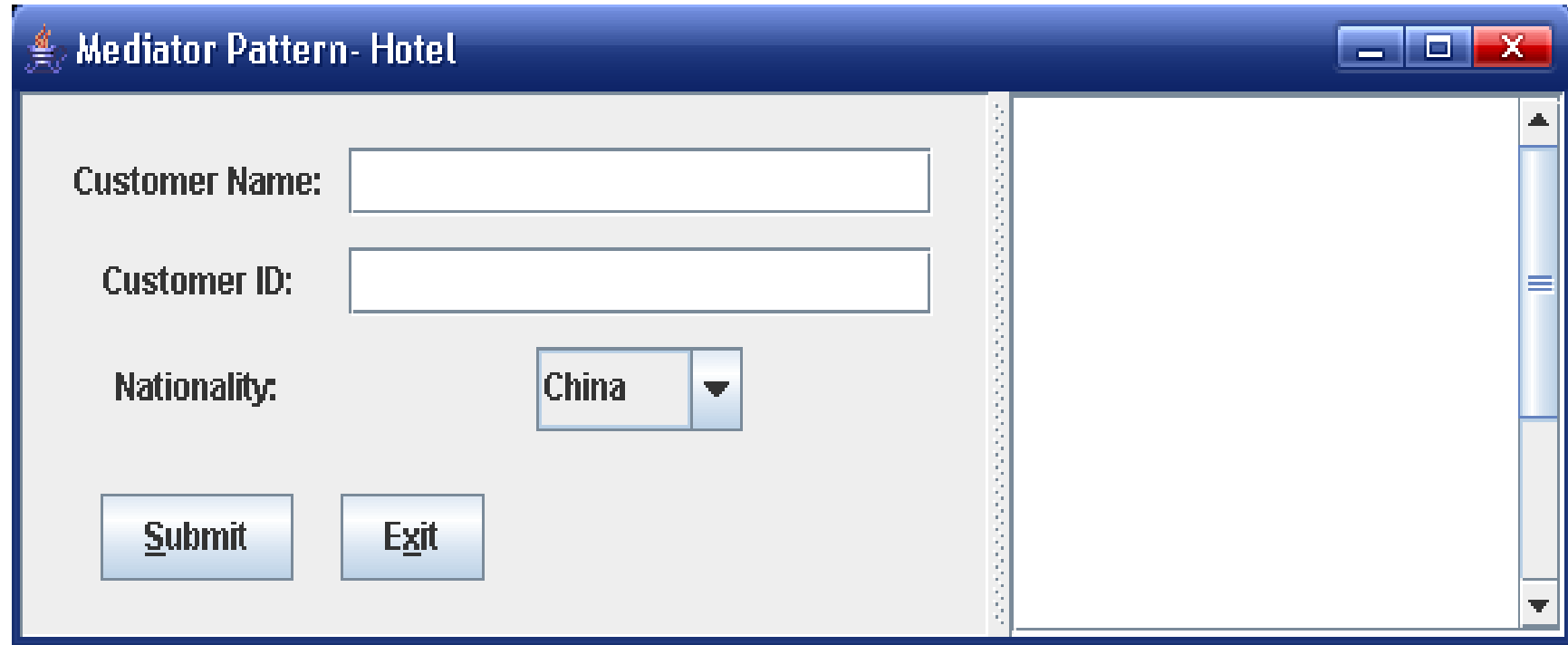


Example Design Using the Mediator Pattern

Example of Mediator Pattern

- **Example: Hotel-Airline-Tour info system**
- **See the following class diagram for the Mediator pattern implementation of collaboration program for**
 - **Hotel,**
 - **Airline and**
 - **Tour.**
- **The prototypes of three user interfaces for the Hotel, Airline and Tour are as below.**

Example of Mediator Pattern



The image shows a Windows-style application window titled "Mediator Pattern- Hotel". The window has a dark blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains three input fields: "Customer Name:" with a text box, "Customer ID:" with a text box, and "Nationality:" with a dropdown menu currently showing "China". Below these fields are two buttons: "Submit" and "Exit". To the right of the input fields is a large, empty white rectangular area, possibly a placeholder for a list or details. The window has a vertical scrollbar on the right side.

Hotel information GUI

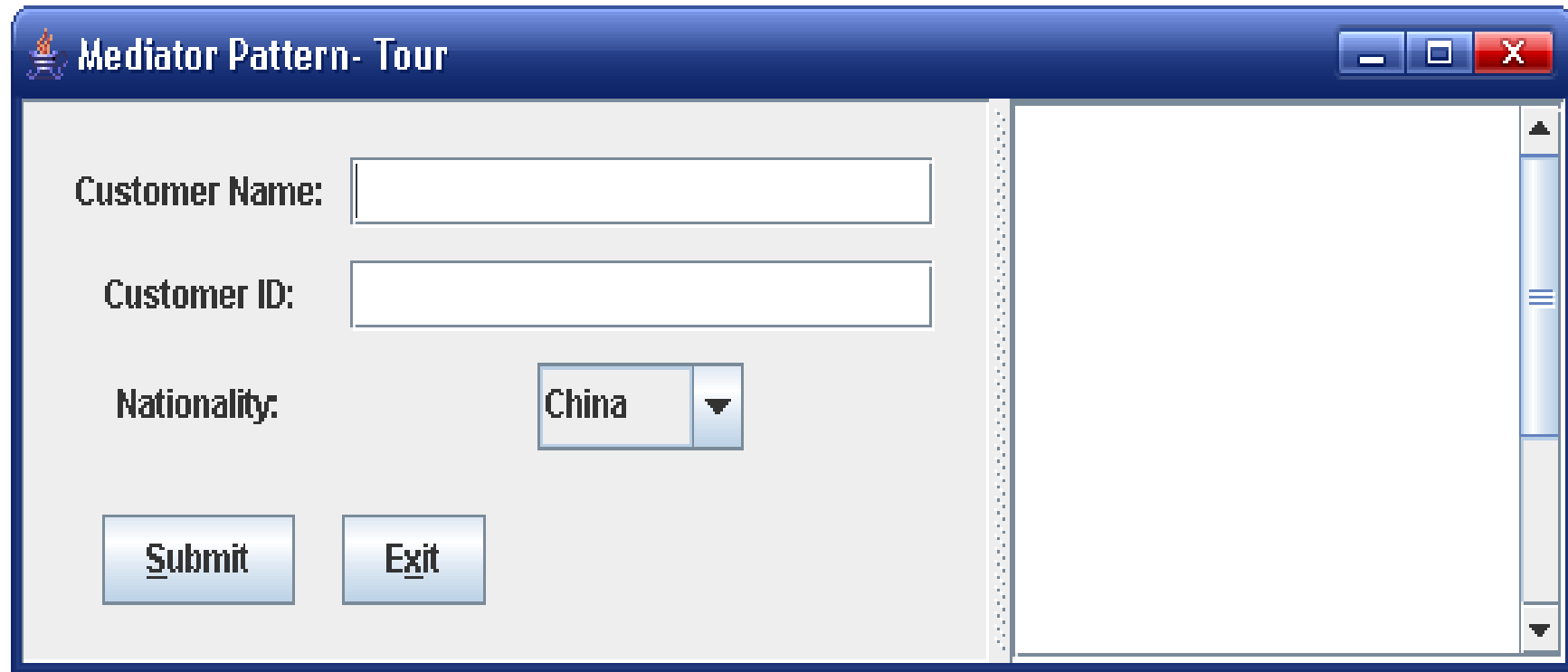
Example of Mediator Pattern



The image shows a Windows-style application window titled "Mediator Pattern- Airline". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is divided into two panes by a vertical splitter. The left pane contains a form with three input fields: "Customer Name:" followed by a text box, "Customer ID:" followed by a text box, and "Nationality:" followed by a dropdown menu currently showing "China". Below these fields are two buttons labeled "Submit" and "Exit". The right pane is empty and has a vertical scrollbar on its right side.

Airline information GUI

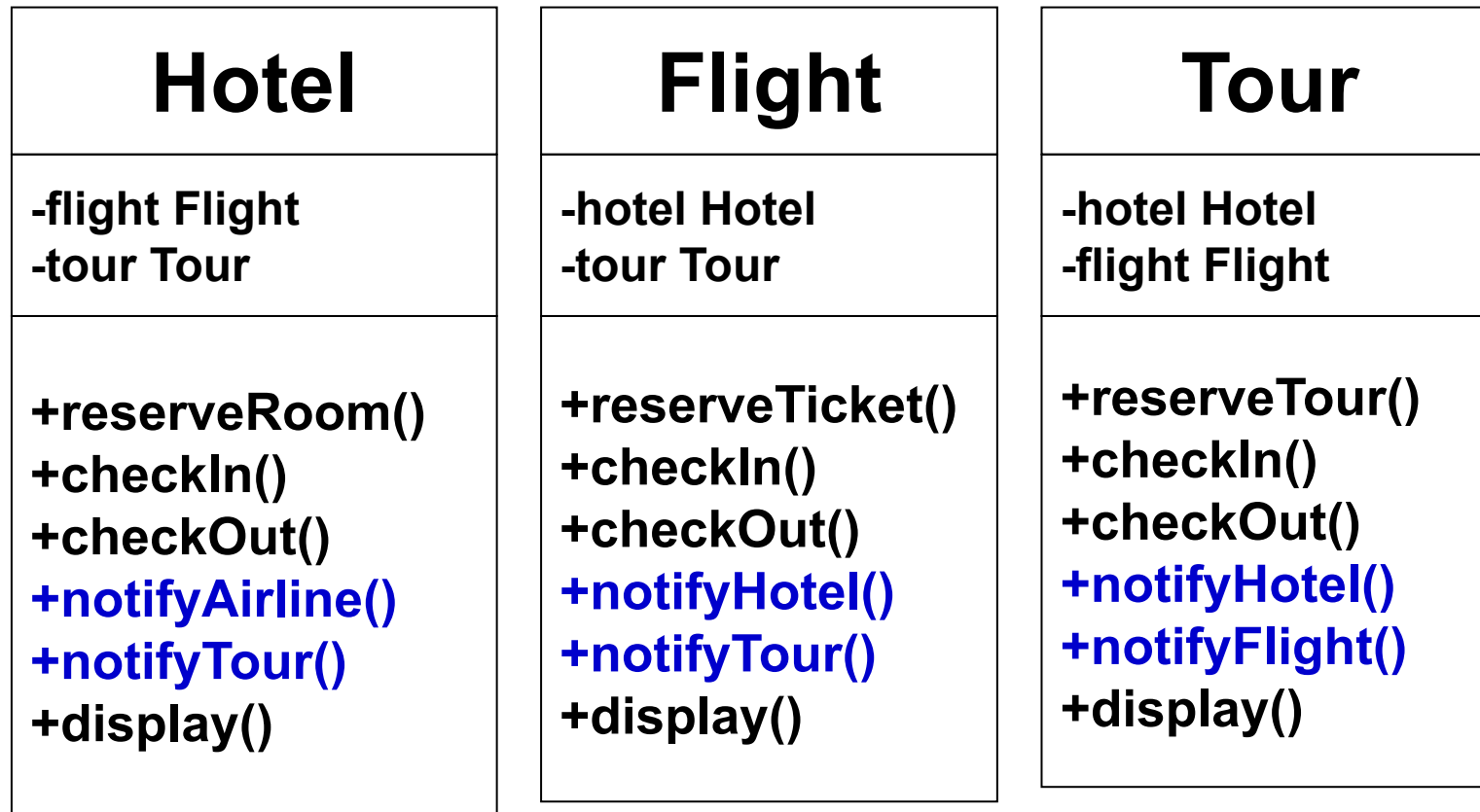
Example of Mediator Pattern



The image shows a Windows-style application window titled "Mediator Pattern- Tour". The window has a dark blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains three input fields: "Customer Name:" with a text box, "Customer ID:" with a text box, and "Nationality:" with a dropdown menu currently showing "China". Below these fields are two buttons: "Submit" and "Exit". To the right of the input fields is a large, empty white rectangular area, possibly a placeholder for a list or details. The window has a vertical scrollbar on the right side.

Tour information GUI

Example of Mediator Pattern



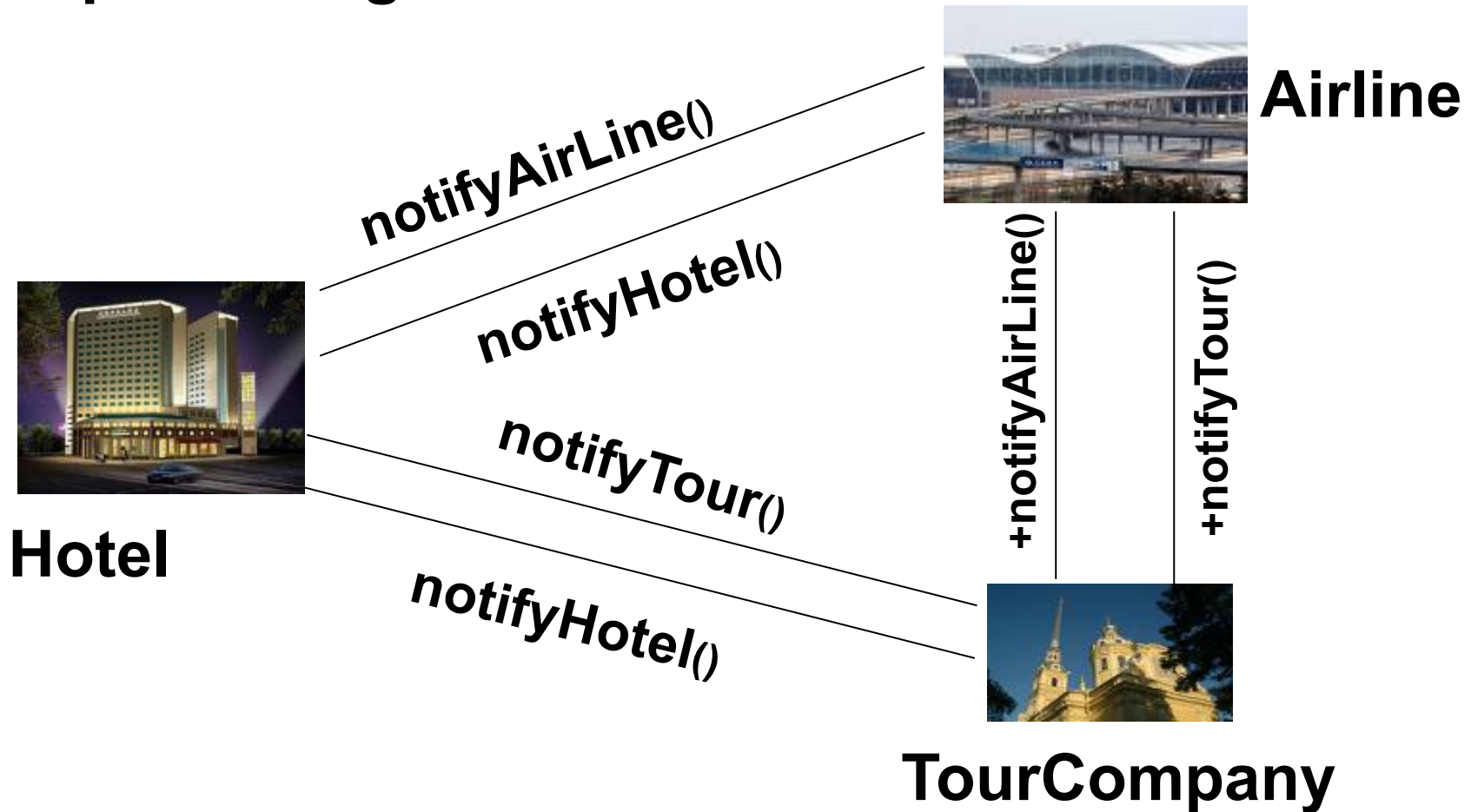
Class diagrams

Example of Mediator Pattern

- **目的：三家企业信息共享**
- **Objective:** When a receptionist inputs customer information into hotel info GUI, then the flight info GUI and the tourist info GUI will show the same information, including
 - **customer's name,**
 - **id number, and**
 - **nationality**
- **So that all the three companies will share information**

Example of Mediator Pattern

In such a way, all the 3 companies are tightly coupled as figure below.

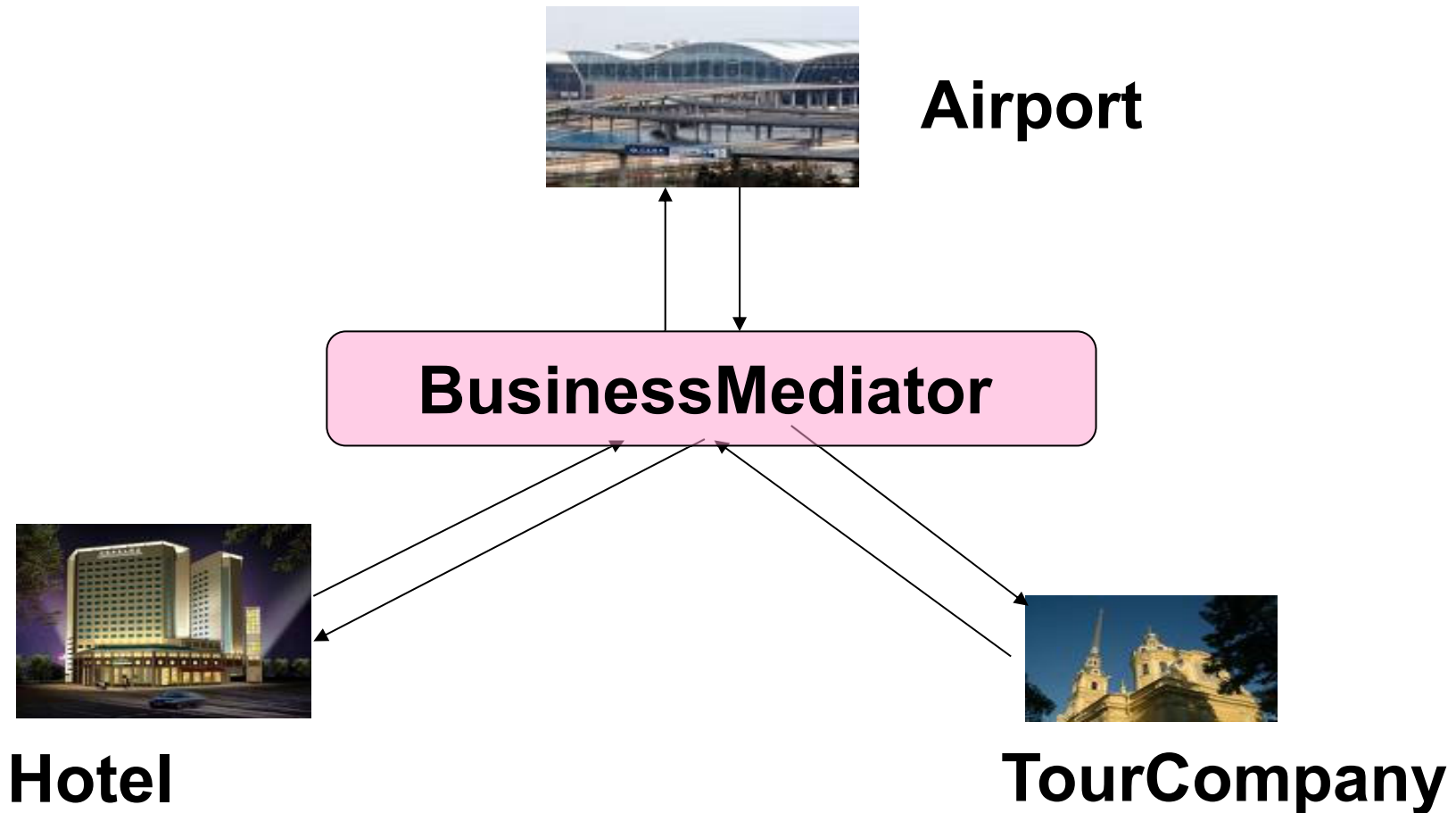


The three objects are tightly coupled

Example of Mediator Pattern

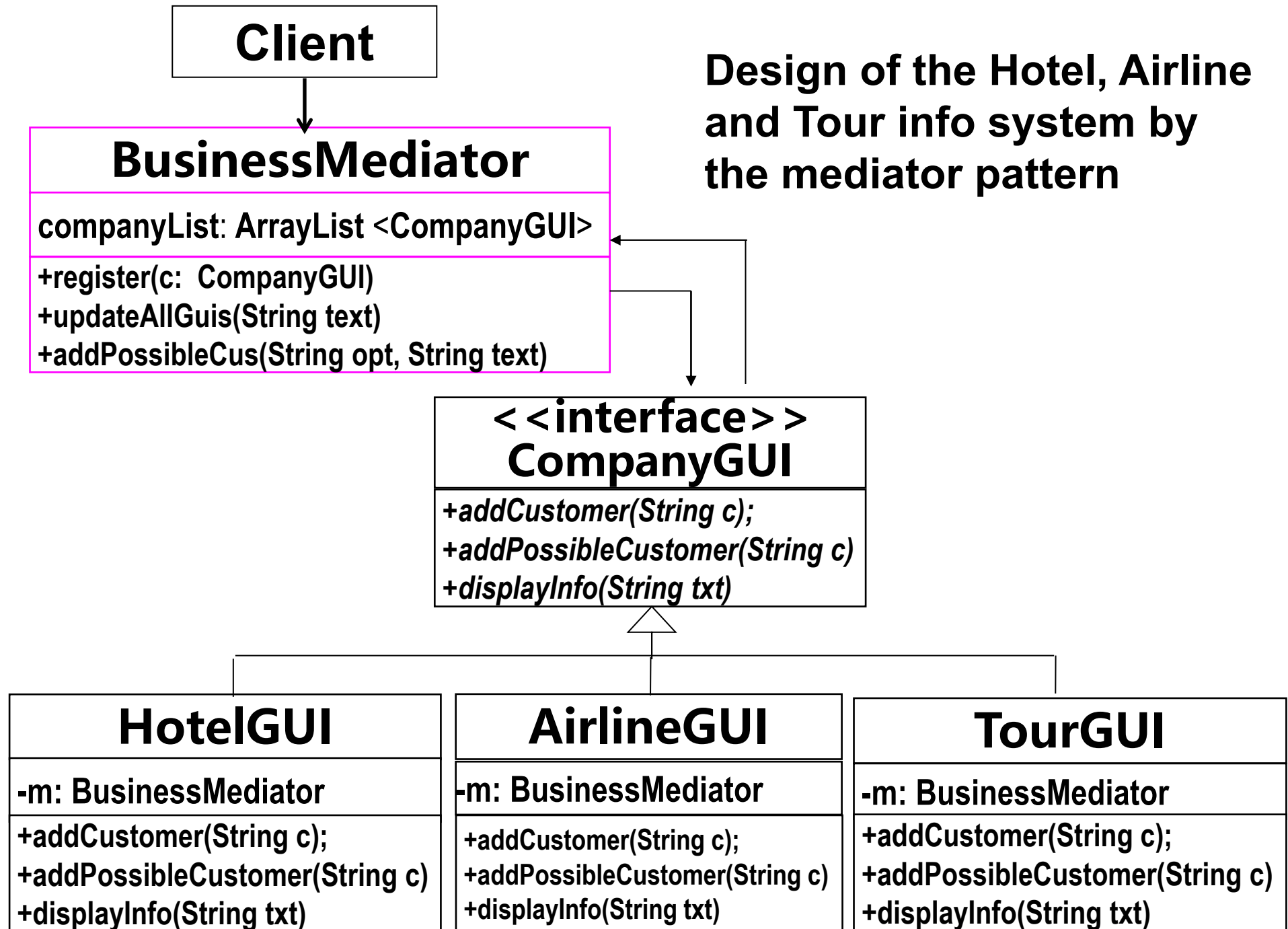
- **设计缺点：各个对象之间直接交互**
 - 调用关系复杂
 - 可扩展性差
 - 等缺点
- **解决方案：使用中介者模式，引入中介者类 BusinessMediator.**

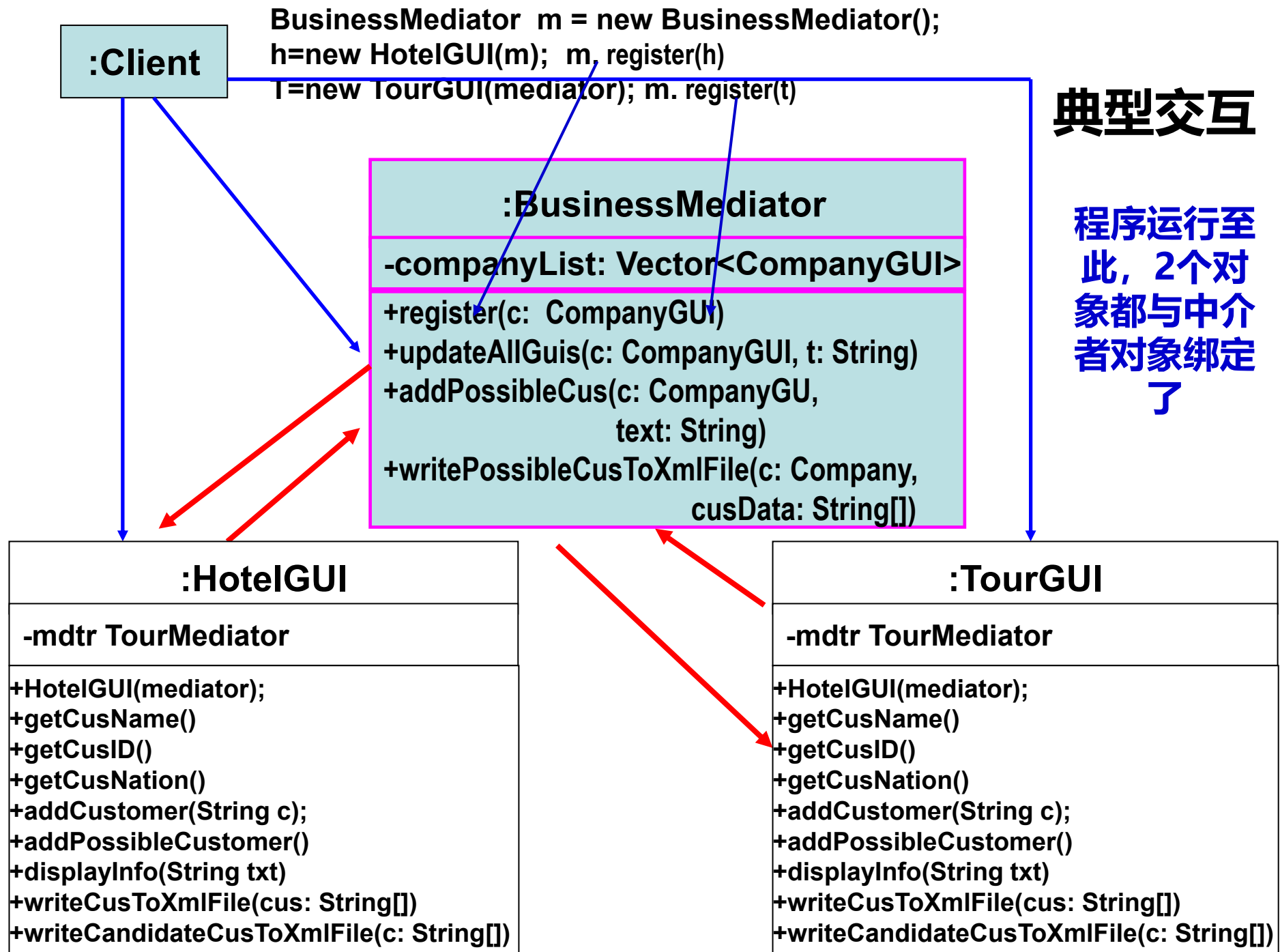
Example of Mediator Pattern



Use mediator to reduce the coupling

**Design of the Hotel, Airline
and Tour info system by
the mediator pattern**





:Client

```
BusinessMediator m = new BusinessMediator();  
h=new HotelGUI(m); m. register(h)  
T=new TourGUI(mediator); m. register(t)
```



:BusinessMediator

-companyList: Vector<CompanyGUI>

+register(c: CompanyGUI)
+updateAllGuis(c: CompanyGUI, t: String)
+addPossibleCus(c: CompanyGU, text: String)
+writePossibleCusToXmlFile(c: Company, cusData: String[])

2个对象都与中介者对象绑定之后，参与者与中介者的交互情况。

:HotelGUI

-mdtr TourMediator

+HotelGUI(mediator);
+getCusName()
+getCusID()
+getCusNation()
+addCustomer(String c);
+addPossibleCustomer(this, t)
+displayInfo(String txt)
+writeCusToXmlFile(cus: String[])
+writeCandidateCusToXmlFile(this, c[])

:TourGUI

-mdtr TourMediator

+HotelGUI(mediator);
+getCusName()
+getCusID()
+getCusNation()
+addCustomer(String c);
+addPossibleCustomer(text)
+displayInfo(String txt)
+writeCusToXmlFile(cus: String[])
+writeCandidateCusToXmlFile(c: String[])

Mike Sun
111668888
China

这个程序设计的可扩展性好。



Implementation Details of the Mediator Pattern

Implementation details of the mediator pattern

1. 创建中介者类(Create a Mediator class)

Write a mediator class, and claim private variables of type ArrayList to hold participating objects, e.g.,

- private ArrayList<ParticipantGUI>
companyList = new ArrayList<ParticipantGUI>();

Implementation details of the mediator pattern

2. 在中介者类里面，写好 注册方法（Write registration method）

In the mediator class, write all registration methods to register all the participating objects, for example,

```
public void register(ParticipantGUI company){  
    companyList.add(company);  
}
```


Implementation details of the mediator pattern

3. 客户类要做的事情

In the client class, create objects of the mediator classes and objects of participating classes.

```
BusinessMediator mediator;
```

```
mediator = new BusinessMediator();
```

```
ParticipantGUI airLine = new AirlineGUI(mediator);
```

```
ParticipantGUI hotel = new HotelGUI(mediator);
```

```
mediator.register(airLine);
```

```
mediator.register(hotel);
```

