

CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2010
Doc 19 Facade, Flyweight, Mediator
15 Apr 2010

Copyright ©, All rights reserved. 2010 SDSU & Roger Whitney, 5500
Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this
document.

References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 185-194, 195-206

This Car Runs on Code, <http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code>

Source lnes of code, http://en.wikipedia.org/wiki/Source_lines_of_code





Size

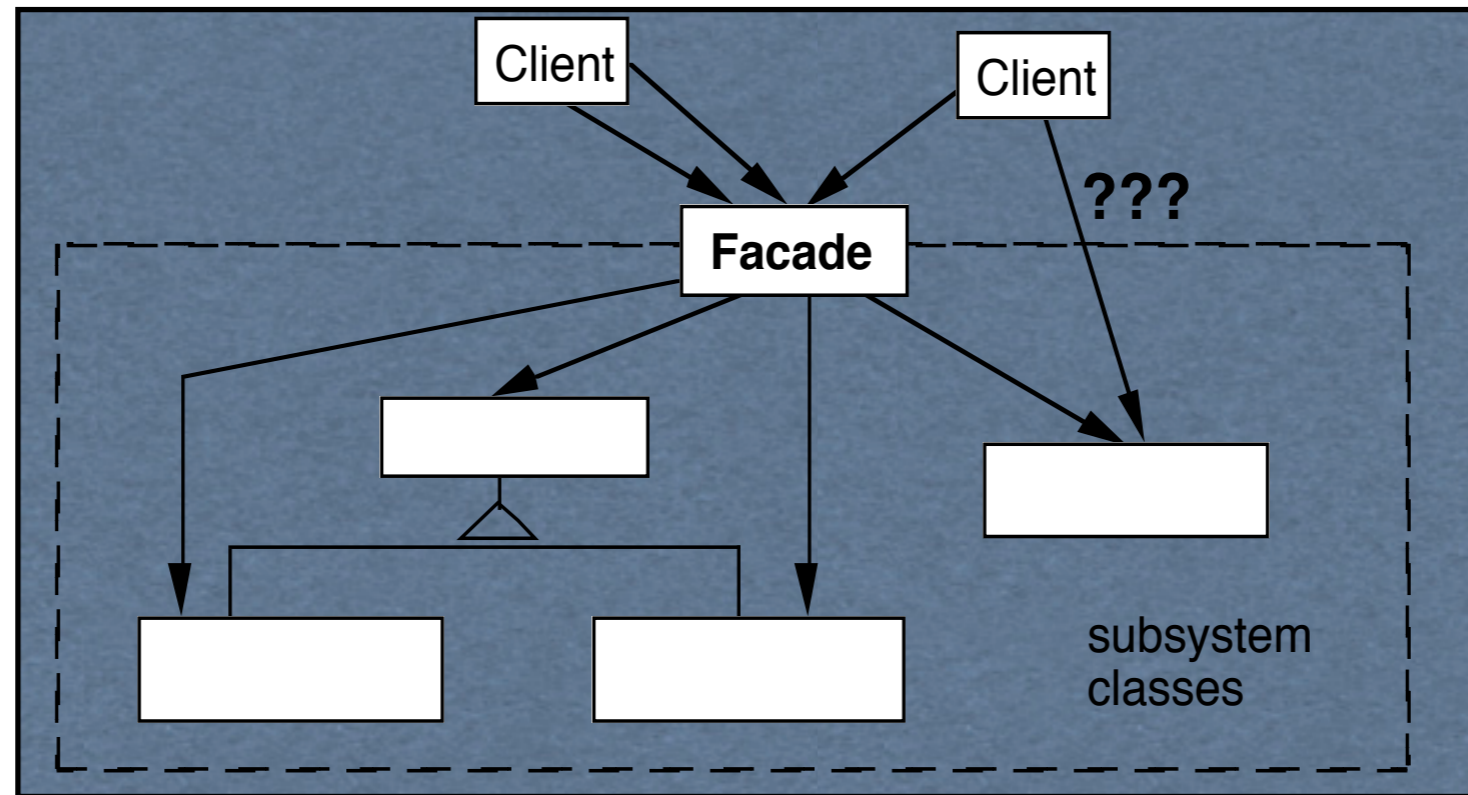
Item	Source Lines of Code (Millions)
F-22 Raptor US jet fighter	1.7
Boeing 787	6.5
S-class Mercedes-Benz radio & navigation system	20
Premium class automobile	~100
Mac OS 10.4	86
Debian 4.0	283

Design Patterns text contains under 8,000 lines

The Facade Pattern

Create a class that is the interface to the subsystem

Clients interface with the Facade class to deal with the subsystem



Consequences of Facade Pattern

It hides the implementation of the subsystem from clients

It promotes weak coupling between the subsystems and its clients

It does not prevent clients from using subsystem classes directly, should it?

Facade does not add new functionality to the subsystem

Public versus Private Subsystem classes

Some classes of a subsystem are

public

facade

private

Compiler Example

The VisualWorks Smalltalk compiler system has 75 classes

Programmers only use Compiler, which uses the other classes

Compiler evaluate: '100 factorial'

```
| method compiler |
```

```
method := 'reset
```

```
  "Resets the counter to zero"
```

```
  count := 0.'
```

```
compiler := Compiler new.
```

```
compiler
```

```
  parse:method
```

```
  in: Counter
```

```
  notifying: nil
```

Flyweight

Use sharing to support large number of fine-grained objects efficiently

Text Example

A document has many instances of the character 'a'

Character has

- Font

- width

- Height

- Ascenders

- Descenders

- Where it is in the document

Most of these are the same for all instances of 'a'

Use one object to represent all instances of 'a'

Java String Example

```
public void testInterned() {  
    String a1 = "catrat";  
    String a2 = "cat";  
    assertFalse(a1 == (a2+ "rat"));  
  
    String a3 = (a2 + "rat").intern();  
    assertTrue(a1 == a3);  
    String a4 = "cat" + "rat";  
    assertTrue(a1 == a4);  
    assertTrue(a3 == a4);  
}
```

public String intern()

Returns a canonical representation for the string object.

A pool of strings, initially empty, is maintained privately by the class String.

Intrinsic State

Information that is independent from the objects context

The information that can be shared among many objects

So can be stored inside of the flyweight

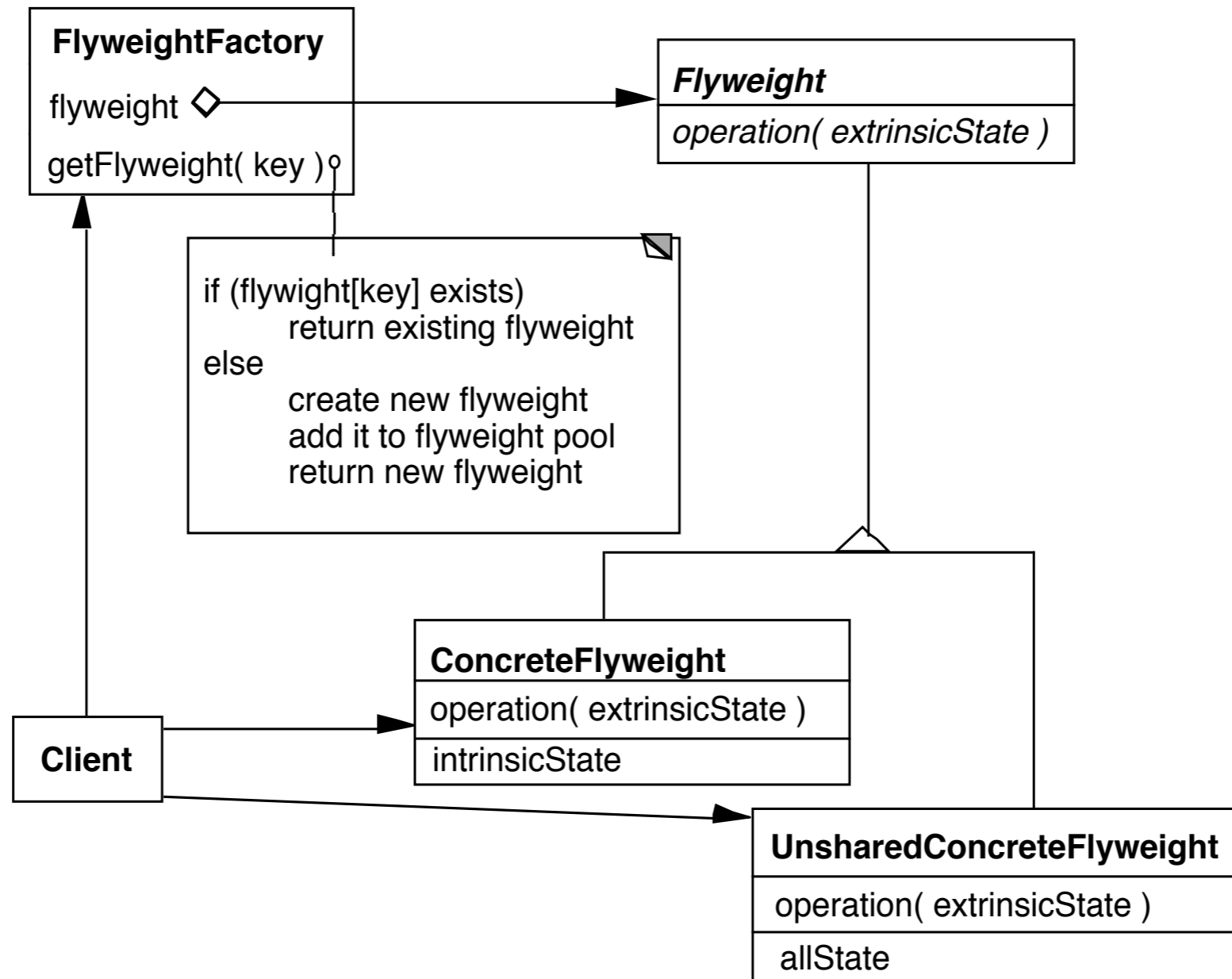
Extrinsic State

Information that is dependent on the objects context

The information that can not be shared among objects

So has to be stored outside of the flyweight

Structure



The Hard Part

Separating state from the flyweight

How easy is it to identify and remove extrinsic state

Will it save space to remove extrinsic state

Example Text

Run Arrays

aaaaabaaaaaaaaaaaaaaaaaaaaa



a b a

5 1 20

Text Example

Lexi Document Editor

Uses character objects with font information
(To support graphic elements)

"A Cat in the hat came **back** the very next day"

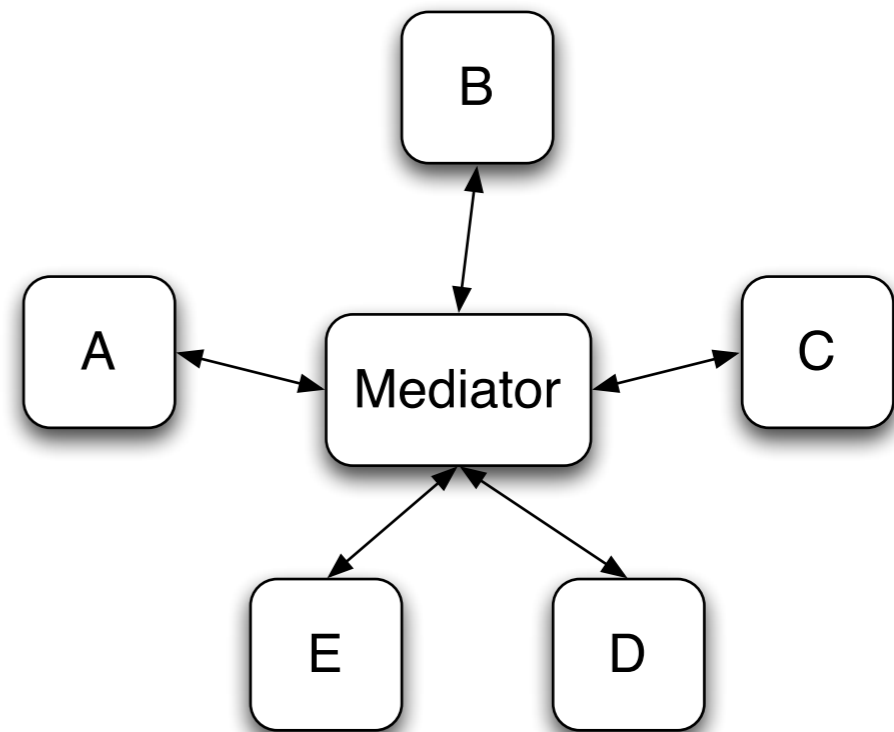
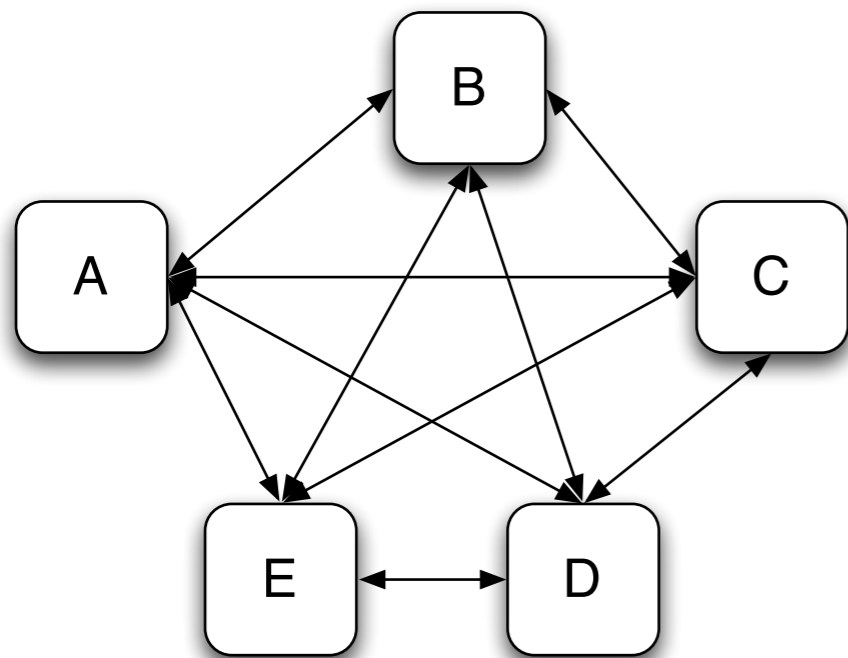
Use run array to store font information (extrinsic state)

Normal Bold Normal

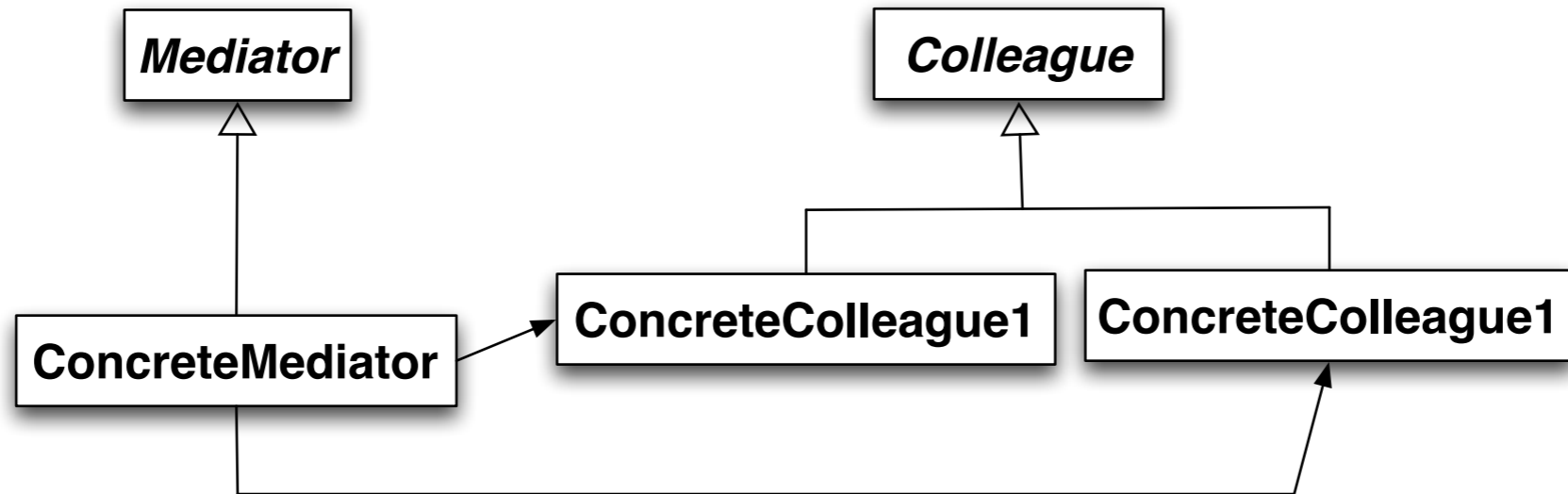
22 4 18

Mediator

A mediator controls and coordinates the interactions of a group of objects



Structure



Participants

Mediator

Defines an interface for communicating with Colleague objects

ConcreteMediator

Implements cooperative behavior by coordinating Colleague objects

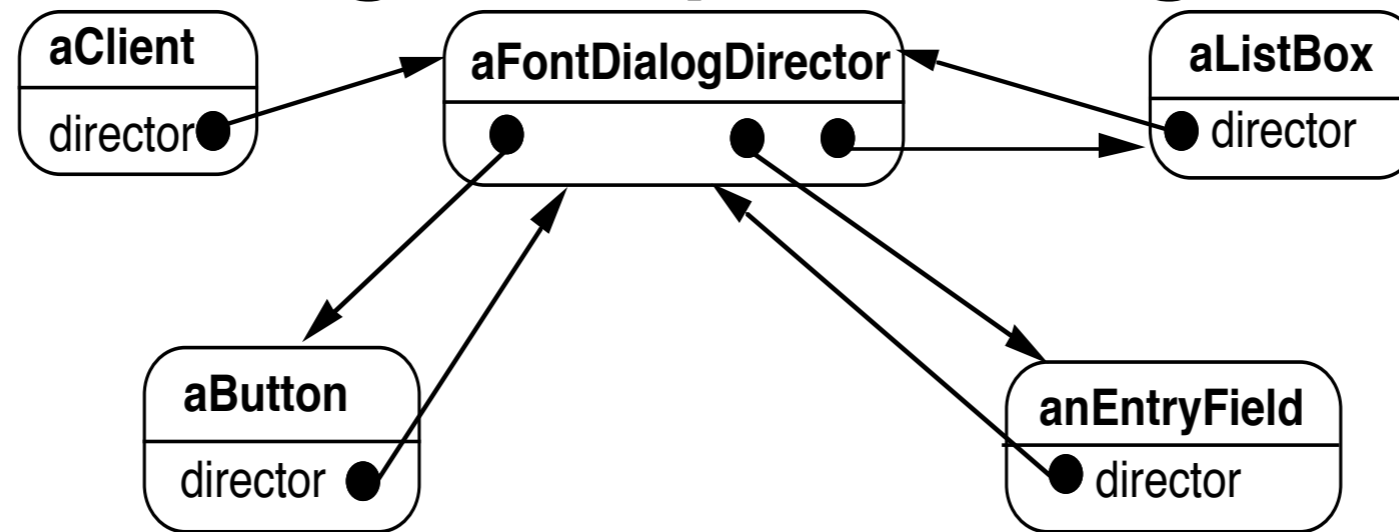
Knows and maintains its colleagues

Colleague classes

Each Colleague class knows its Mediator object

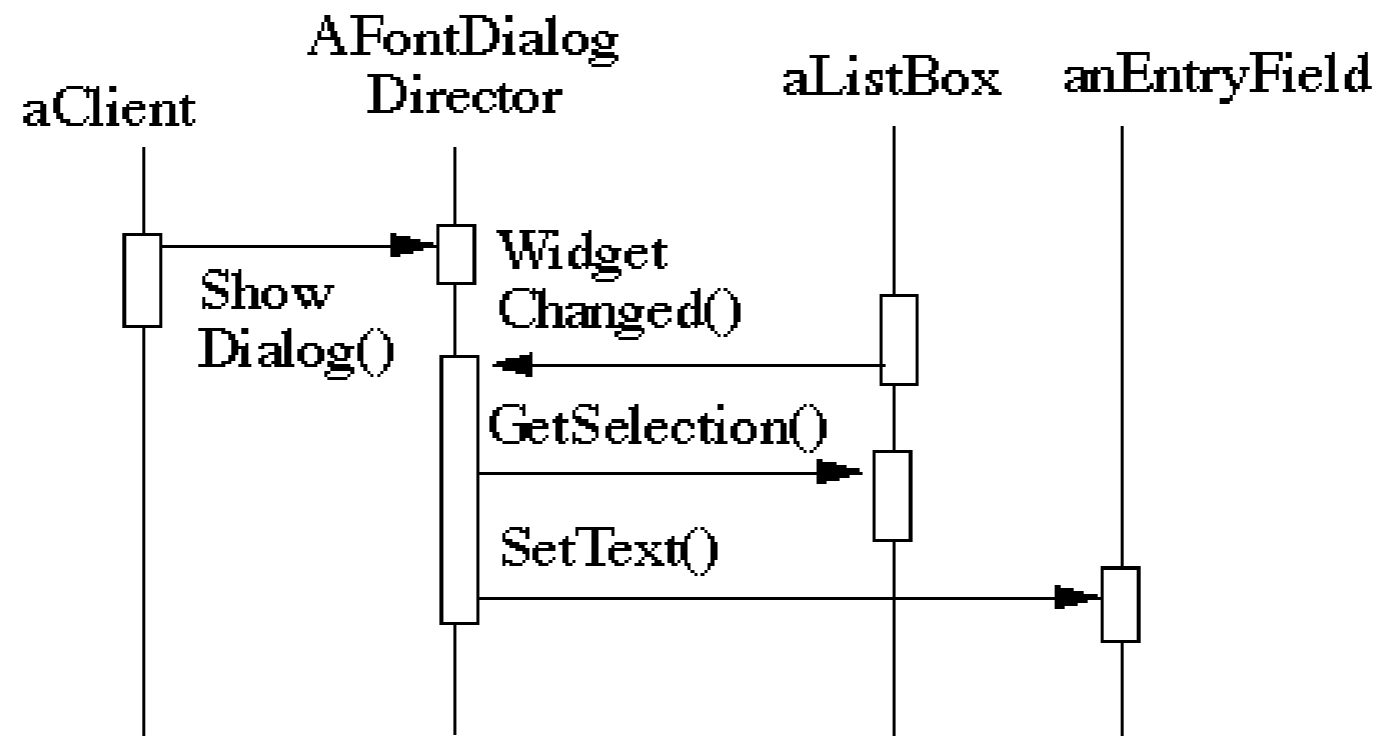
Each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague

Motivating Example - Dialog Boxes



Mediator

Colleagues



How does this differ from a God Class?

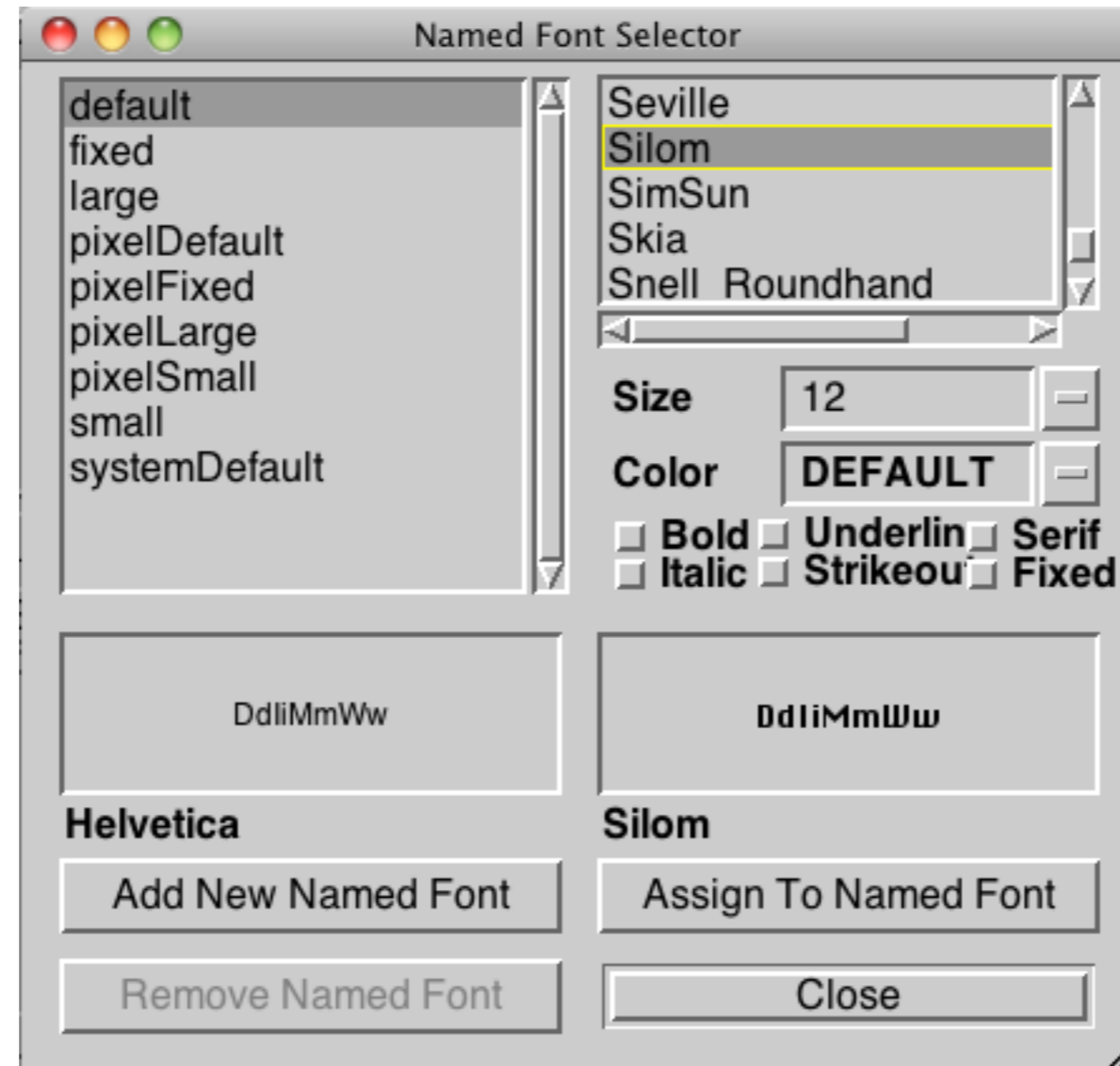
When to use the Mediator Pattern

When a set of objects communicate in a well-defined but complex ways

When reusing an object is difficult because it refers to and communicates with many other objects

When a behavior that's distributed between several classes should be customizable without a lot of subclassing

Classic Mediator Example



Simpler Example



A simple login dialog box titled "Login Dialog". It features a standard window header with three colored buttons (red, yellow, green) on the left. The main area contains two text input fields: "User Name" and "Password". Below the input fields are two buttons: "OK" and "Cancel".

User Name

Password

```
class OKButton extends Button {
```

```
    TextField password;
```

```
    TextField username;
```

```
    Database userData;
```

```
    Model application;
```

```
protected void processEvent(AWTEvent e) {
```

```
    if (!e.isButtonPressed()) return;
```

```
    e.consume();
```

```
    if (password.getText() = "") {  
        notifyUser("Must enter password");  
        return;
```

```
    }
```

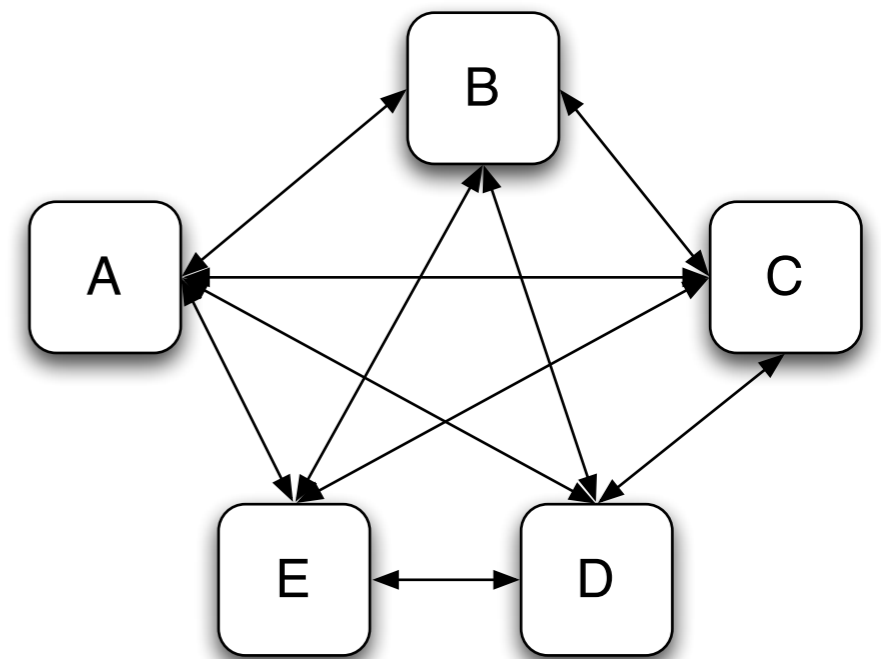
```
    if (username.getText() = "") {  
        notifyUser("Must enter user name");  
        return;
```

```
    }
```

```
    if (!userData.validUser(password.getText(), username.getTest()))  
        notifyUser("Invalid username & password");  
        return;
```

```
    }
```

Non Mediator Solution



Mediator Solution

```
class LoginDialog extends Panel {
```

```
    TextField password;
```

```
    TextField username;
```

```
    Database userData;
```

```
    Button ok, cancel;
```

```
protected void actionPerformed(ActionEvent e) {
```

```
    if (!e.isButtonPressed() or e.getSource() != ok) return;
```

```
    if (password.getText() = "") {
```

```
        notifyUser("Must enter password");
```

```
        return;
```

```
    }
```

```
    if (username.getText() = "") {
```

```
        notifyUser("Must enter user name");
```

```
        return;
```

```
    }
```

```
    if (!userData.validUser(password.getText(), username.getText()))
```

```
        notifyUser("Invalid username & password");
```

```
        return;
```

```
    }
```

What is Different?

Non Mediator Example

Special Button class

OK button coupled to text fields

Mediator Example

No specialButton class

LoginDialog coupled to text fields

Logic moved from button class to LoginDialog

But

Java's event mechanism promotes mediator solution