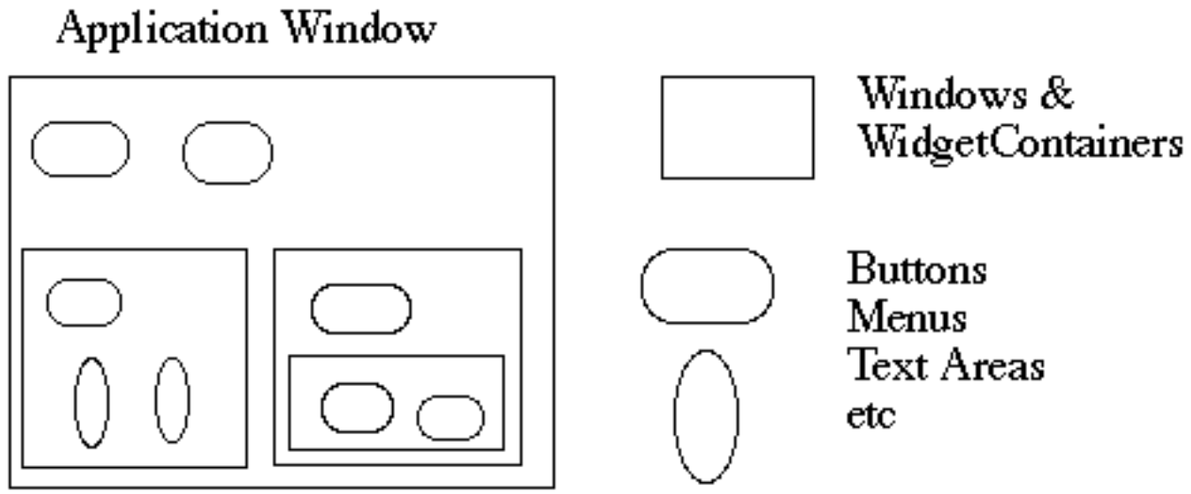
CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2016
Doc 14 Composite, Mediator, Flyweight
April 7, 2016

Composite

Composite Motivation



How does the window hold and deal with the different items it has to manage?

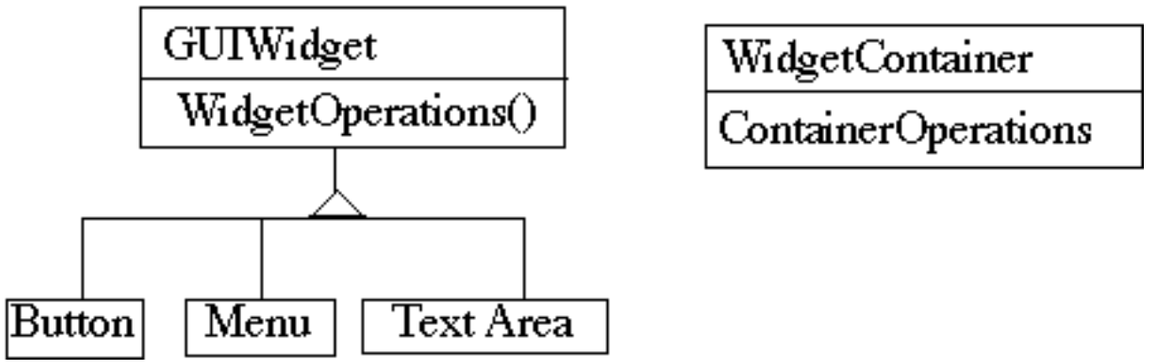
Widgets are different that WidgetContainers

Bad News

```
class Window {
        Buttons[] myButtons;
        Menus[] myMenus;
        TextAreas[] myTextAreas;
        WidgetContainer[] myContainers;

        public void update() {
                if ( myButtons != null )
                        for ( int k = 0; k < myButtons.length(); k++ )
                                myButtons[k].refresh();
                if ( myMenus != null )
                        for ( int k = 0; k < myMenus.length(); k++ )
                                myMenus[k].display();
                if ( myTextAreas != null )
                        for ( int k = 0; k < myButtons.length(); k++ )
                                myTextAreas[k].refresh();
                if ( myContainers != null )
                        for ( int k = 0; k < myContainers.length(); k++ )
                                myContainers[k].updateElements();
                etc.
        }
        public void fooOperation(){
                if (myButtons != null)
                etc.
```

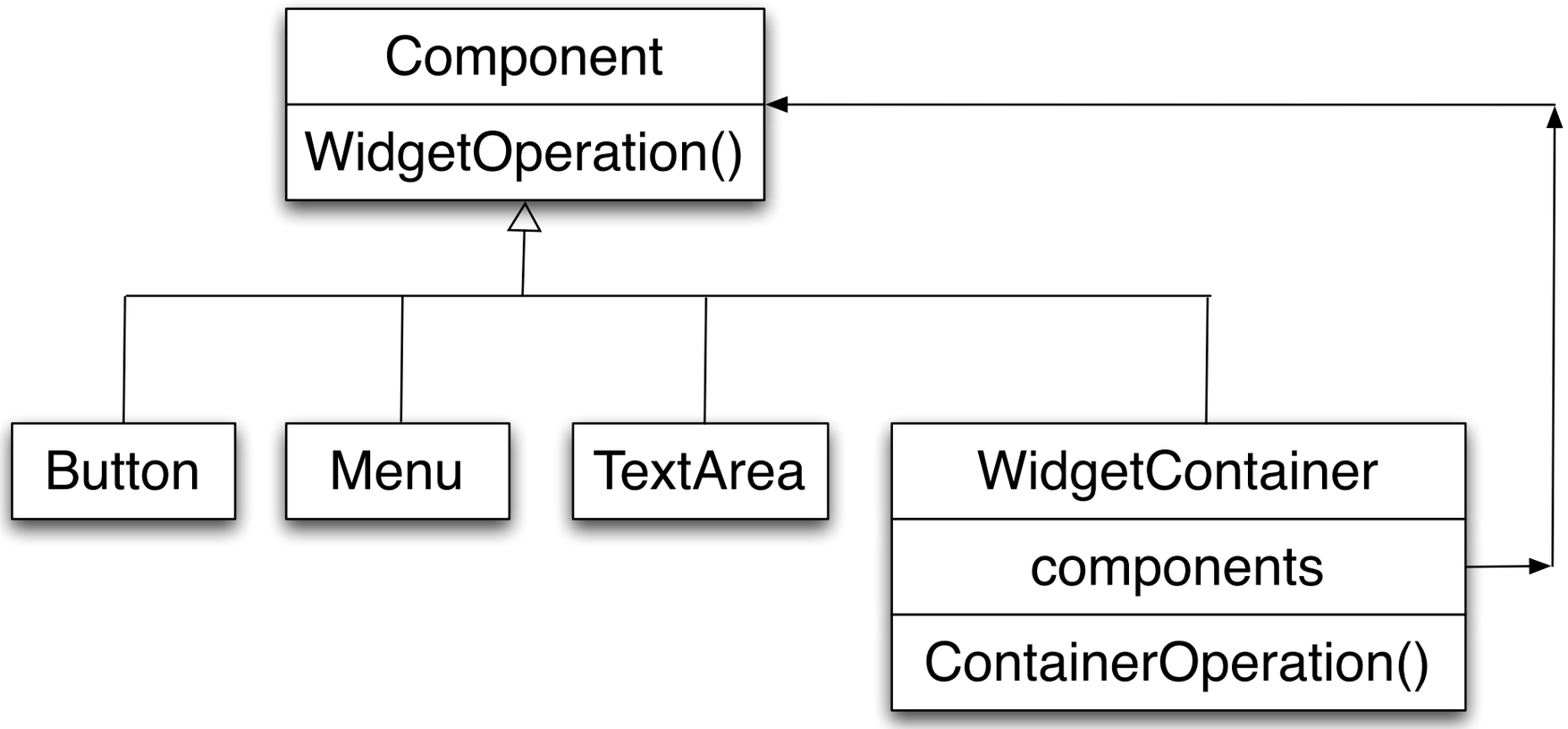
An Improvement




```
class Window {
        GUIWidgets[] myWidgets;
        WidgetContainer[] myContainers;

        public void update(){
                if ( myWidgets != null )
                        for ( int k = 0; k < myWidgets.length(); k++ )
                                myWidgets[k].update();
                if ( myContainers != null )
                        for ( int k = 0; k < myContainers.length(); k++ )
                                myContainers[k].updateElements();
                etc.
        }
}
```

Composite Pattern



Composite Pattern

Component implements default behavior for widgets when possible

Button, Menu, etc overrides Component methods when needed

WidgetContainer will have to overrides all widgetOperations

```
class WidgetContainer {
        Component[] myComponents;

        public void update() {
                if ( myComponents != null )
                        for ( int k = 0; k < myComponents.length(); k++ )
                                myComponents[k].update();
        }
}
```

Issue - WidgetContainer Operations

Should the WidgetContainer operations be declared in Component?

Pro - Transparency

Declaring them in the Component gives all subclasses the same interface

All subclasses can be treated alike. (?)

Con - Safety

Declaring them in WidgetContainer is safer

Adding or removing widgets to non-WidgetContainers is an error

One out is to check the type of the object before using a WidgetContainer operation

Issue - Parent References

```
class WidgetContainer
        {
        Component[] myComponents;

        public void update() {
                if ( myComponents != null )
                        for ( int k = 0; k < myComponents.length(); k++ )
                                myComponents[k].update();
        }

        public add( Component aComponent ) {
                myComponents.append( aComponent );
                aComponent.setParent( this );
        }
}

class Button extends Component {
        private Component parent;
        public void setParent( Component myParent) {
                parent = myParent;
        }

        etc.
```

More Issues

Should Component implement a list of Components?

The button etc. will have a useless data member

Child ordering is important in some cases

Who should delete components?

Applicability

Use Composite pattern when you want

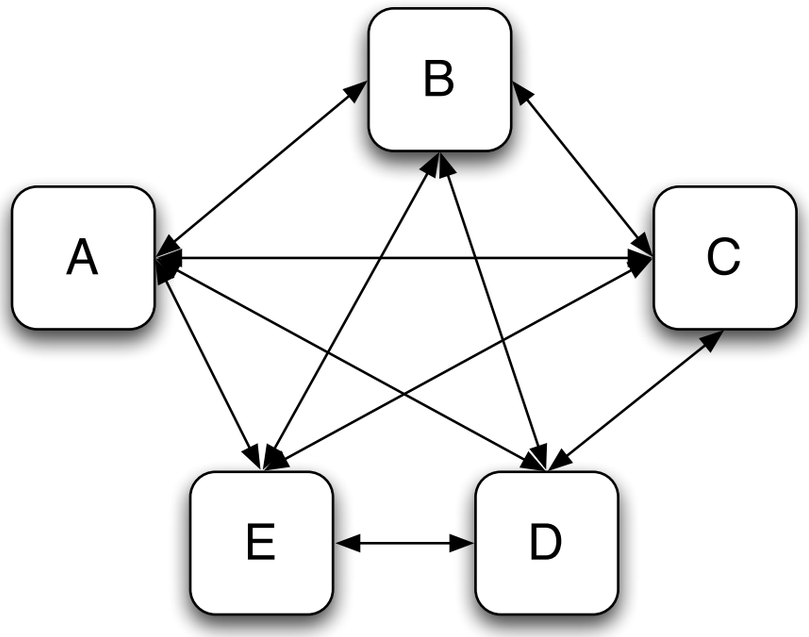
To represent part-whole hierarchies of objects

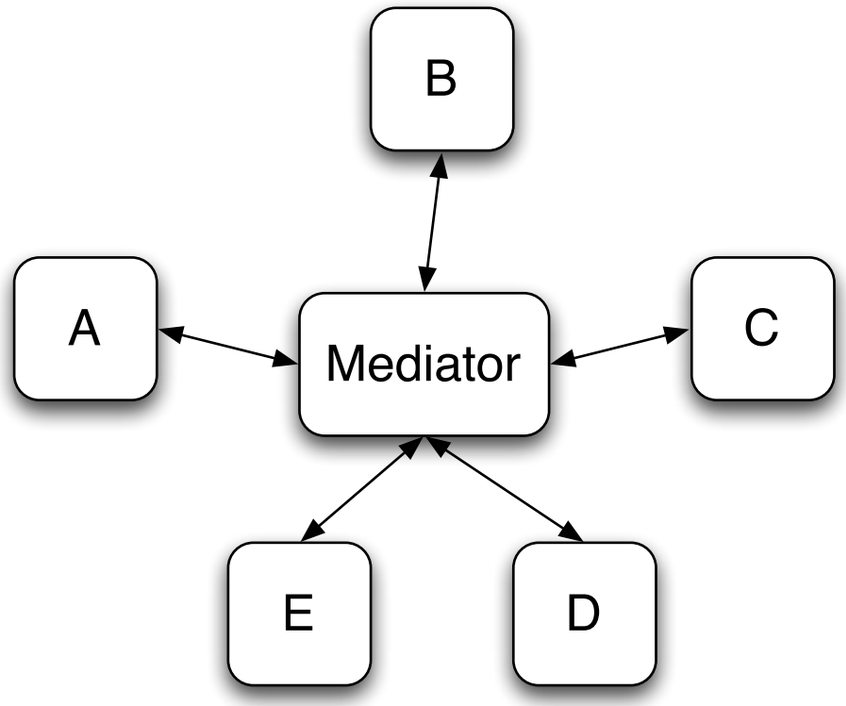
Clients to be able to ignore the difference between compositions of objects and individual objects

Mediator

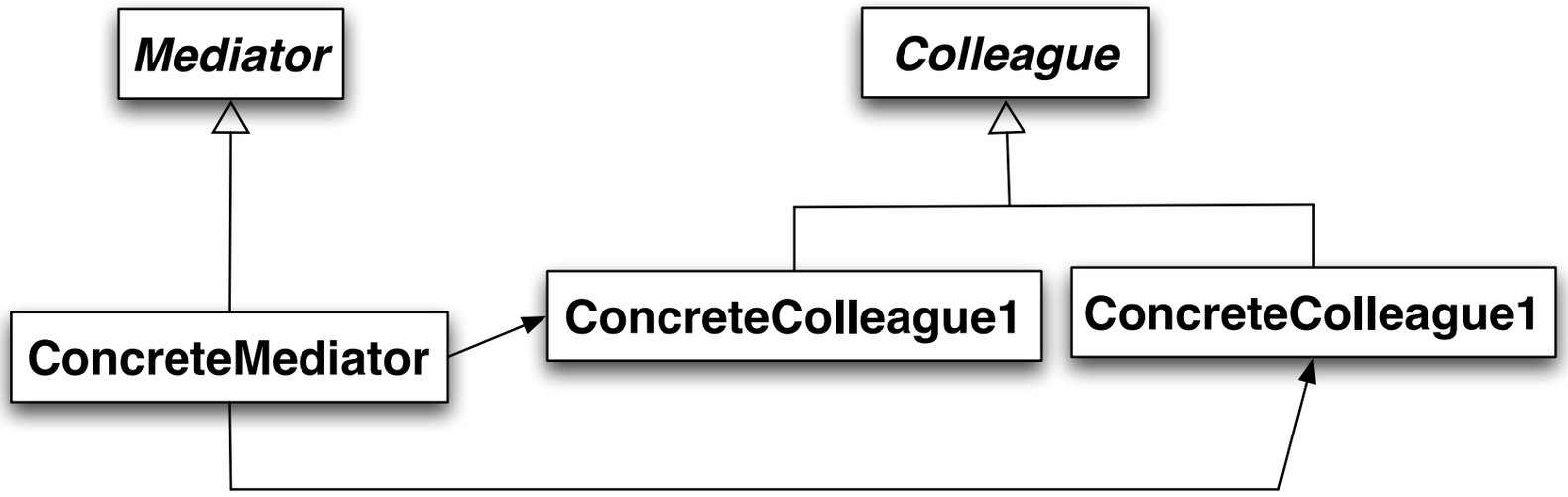
Mediator

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





Structure


Mediator
Colleague
ConcreteMediator
ConcreteColleague1
ConcreteColleague1

Participants

Mediator

Defines an interface for communicating with Colleague objects

ConcreteMediator

Implements cooperative behavior by coordinating Colleague objects

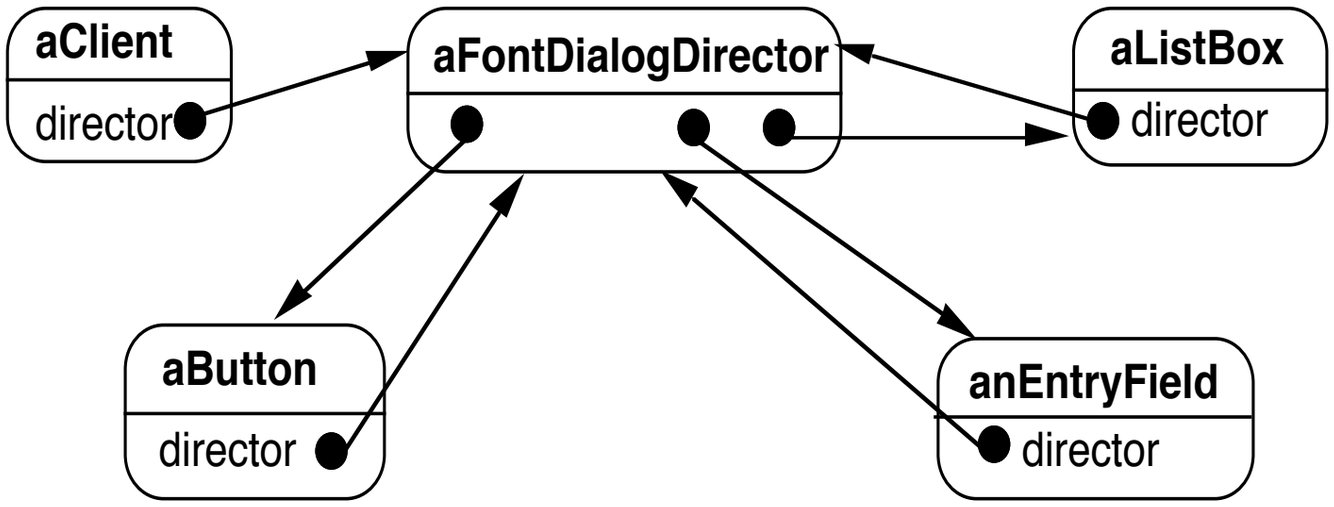
Knows and maintains its colleagues

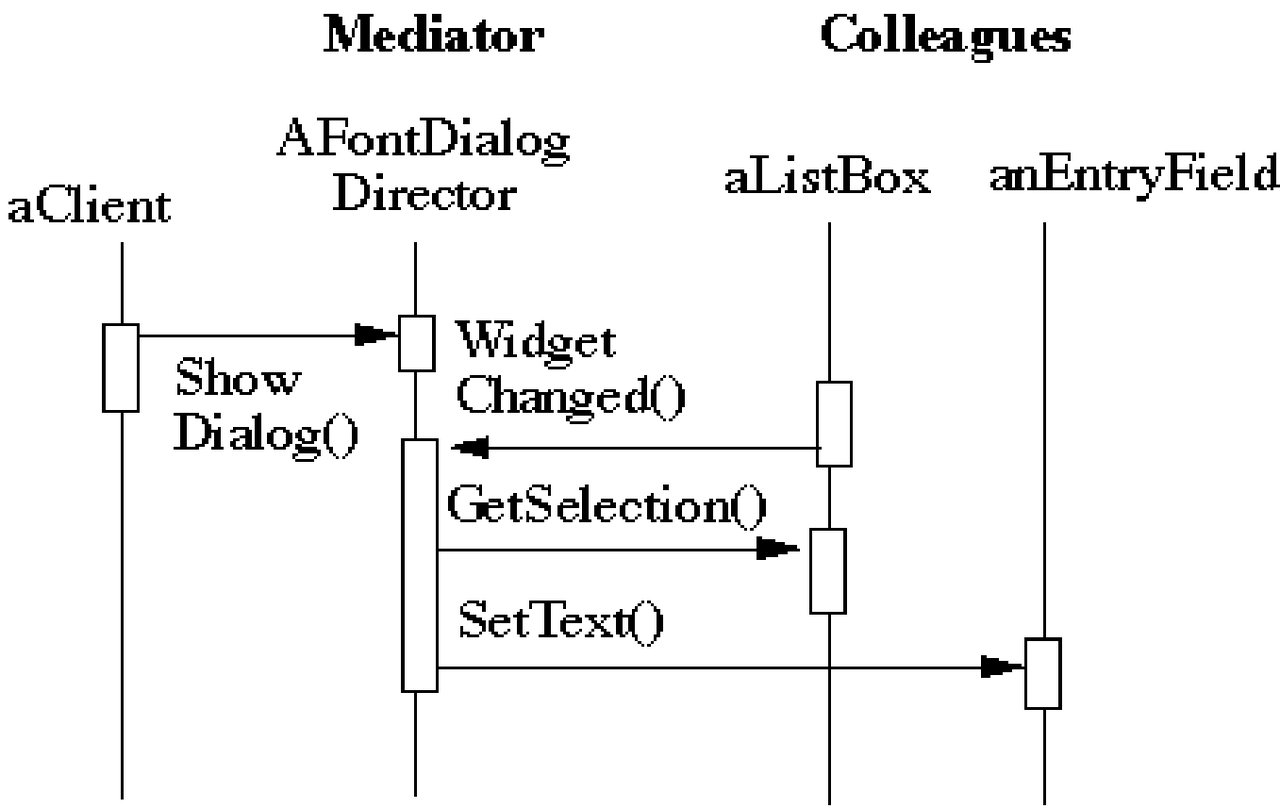
Colleague classes

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





How does this differ from a God Class?

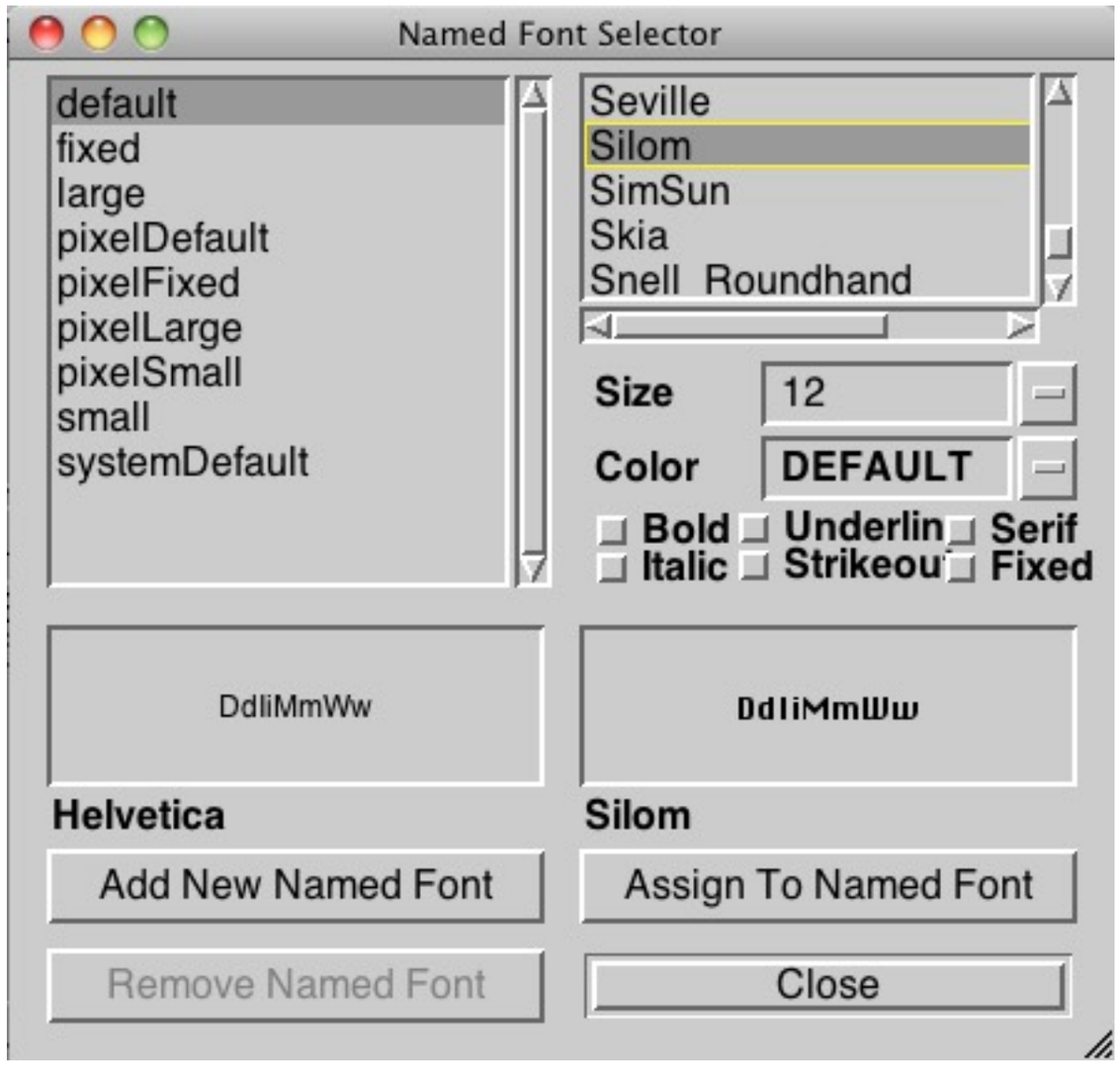
When to use the Mediator Pattern

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

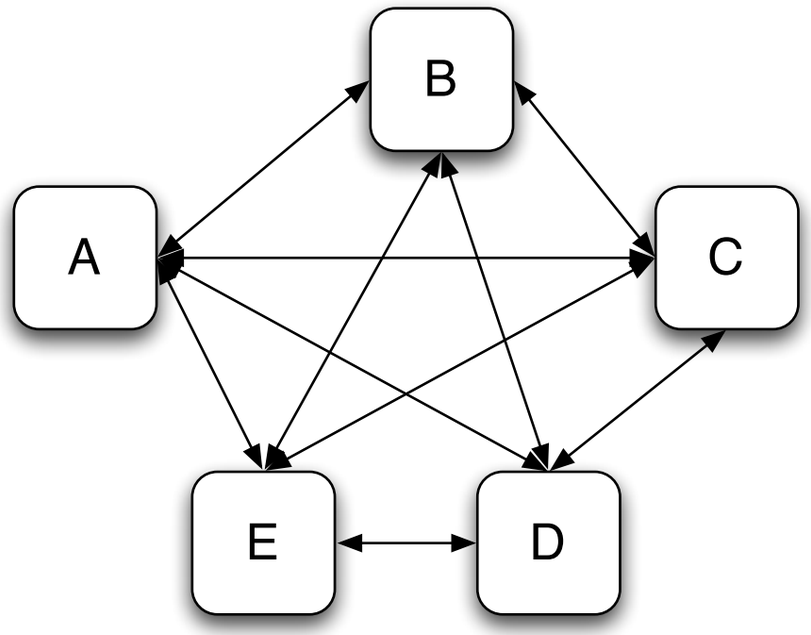


Simpler Example

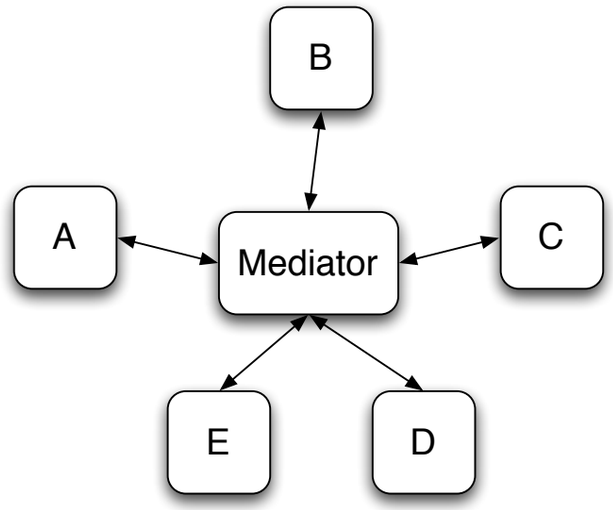
Non Mediator Solution

```
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;
            }
```



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.getTest()))
                  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

Flyweight

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 object's context

The information that can be shared among many objects

So can be stored inside of the flyweight

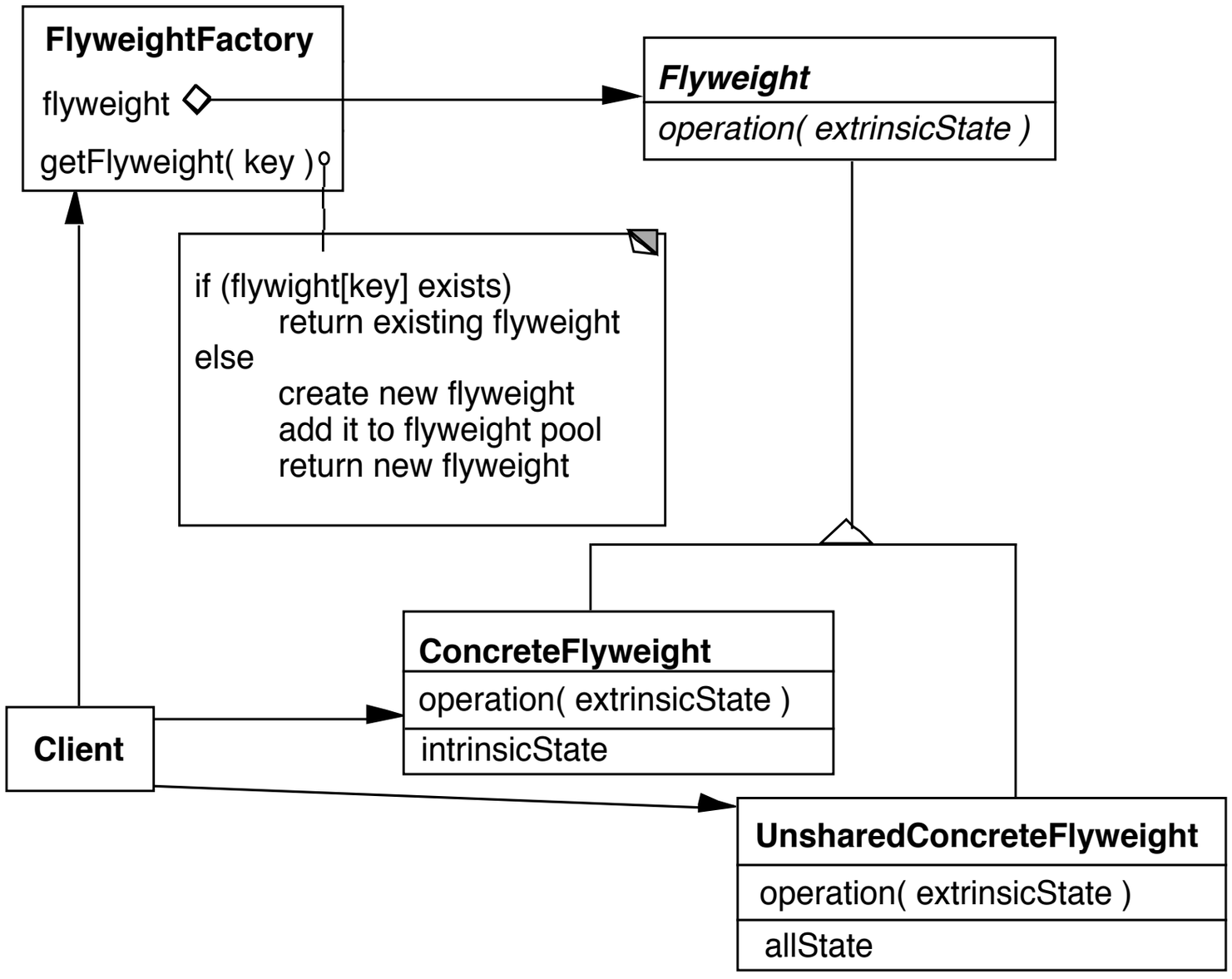
Extrinsic State

Information that is dependent on the object's context

The information that can not be shared among objects

So has to be stored outside of the flyweight

Structure


FlyweightFactory
flyweight
getFlyweight(key)
Flyweight
operation(extrinsicState)
if (flywight[key] exists)
return existing flyweight
else
create new flyweight
add it to flyweight pool
return new flyweight
ConcreteFlyweight
operation(extrinsicState)
intrinsicState
Client
UnsharedConcreteFlyweight
operation(extrinsicState)
allState

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

aaaaabaaaaaaaaaaaaaaaaaaa

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