# CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2018
Doc 18 Composite, Mediator, Flyweight
Nov 13, 2018

# Builder

# Builder

Separate construction of a complex object from its representation

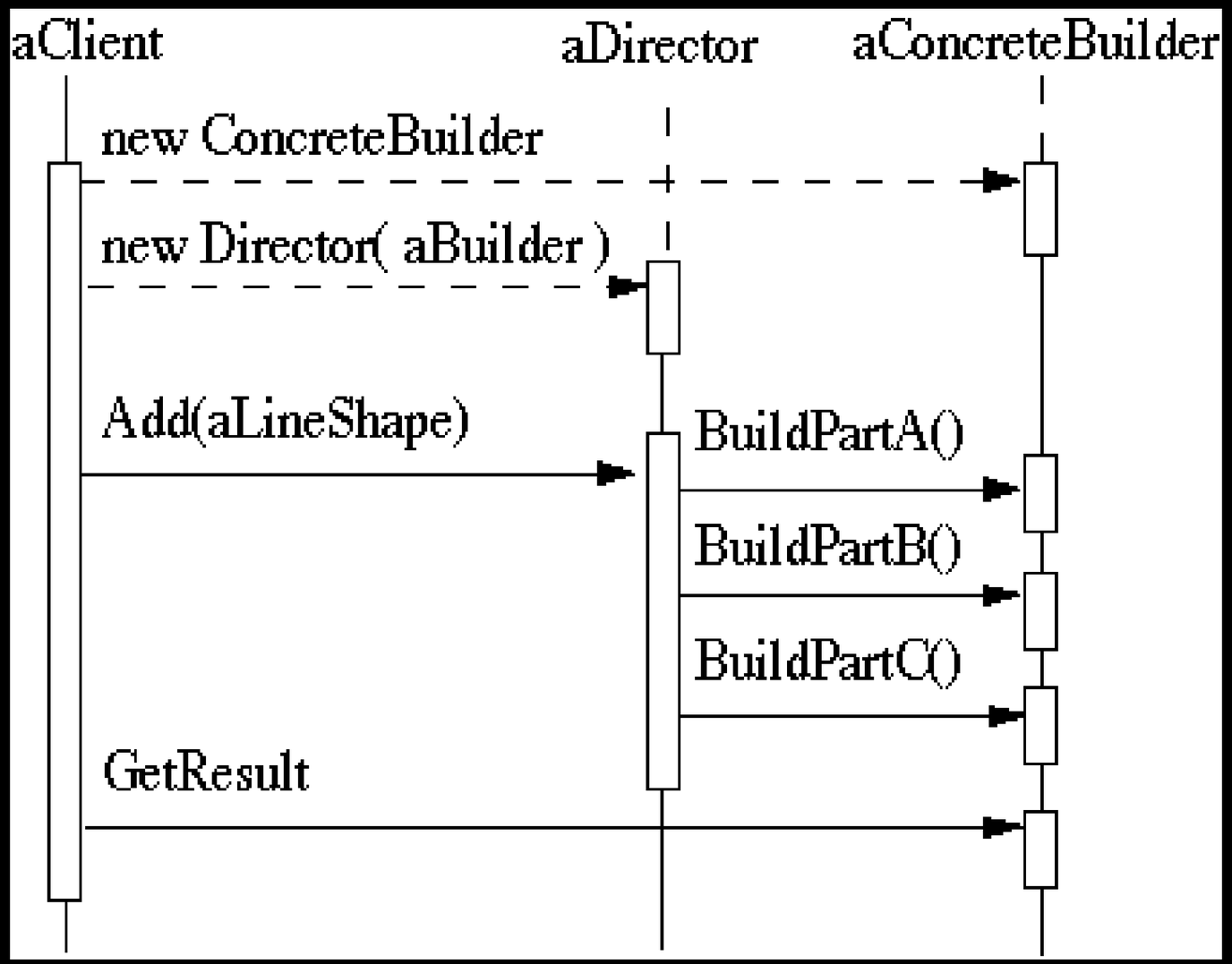
So same construction process can create different representations

# Builder

Client

Director

Builder



# RTF Converter

A word processing document has complex structure

How to convert Rich Text Format (RTF) to

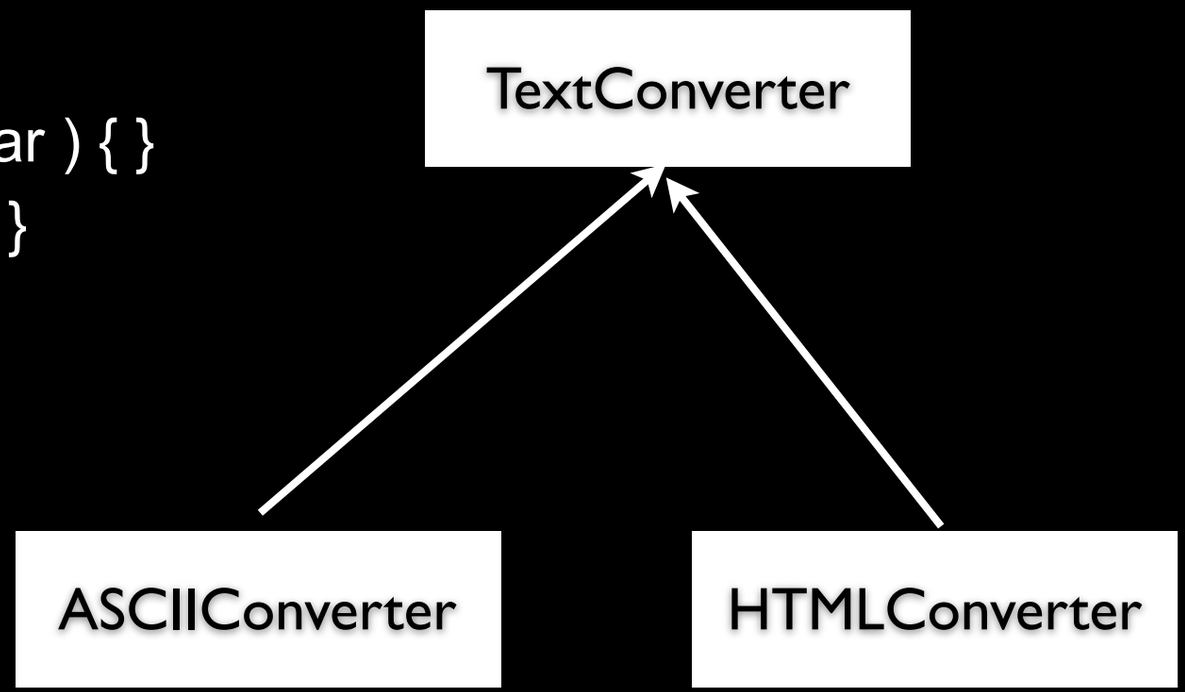
- TeX
- html
- PDF
- etc.

# Pseudo Solution

```
class RTF_Reader {
   TextConverter builder;
   String RTF_Text;

   public RTF_Reader( TextConverter aBuilder, String RTFtoConvert ){
      builder = aBuilder;
      RTF_Text = RTFtoConvert;
      }

public void parseRTF(){
   RTFTokenizer rtf = new RTFTokenizer( RTF_Text );

   while ( rtf.hasMoreTokens() ){
      RTFToken next = rtf.nextToken();

      switch ( next.type() ){
         case CHAR:    builder.character( next.char() ); break;
         case FONT:    builder.font( next.font() ); break;
         case PARA:    builder.newParagraph( ); break;
         etc.
      }
```

# Builder Classes

```
abstract class TextConverter {
   public void character( char nextChar ) { }
   public void font( Font newFont )   { }
   public void newParagraph() {}
}
```



# Sample Program

```
main(){
   ASCII_Converter simplerText = new ASCII_Converter();
   String rtfText;

   // read a file of rtf into rtfText

   RTF_Reader myReader =
      new RTF_Reader( simplerText, rtfText );

   myReader.parseRTF();

   String myProduct = simplerText.getText();
}
```

# The Hard Part

The builder interface

# XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<RootElement param="value">
    <FirstElement>
        Some Text
    </FirstElement>
    <SecondElement param2="something">
        Pre-Text <Inline>Inlined text</Inline> Post-text.
    </SecondElement>
</RootElement>
```

# SAX - Builder Pattern

Director

XMLReader

Builder

ContentHandler

# ContentHandler Interface

void characters(char[] ch, int start, int length)
Receive notification of character data.

void endDocument()
Receive notification of the end of a document.

void endElement(java.lang.String uri, java.lang.String localName, java.lang.String qName)
Receive notification of the end of an element.

void endPrefixMapping(java.lang.String prefix)
End the scope of a prefix-URI mapping.

void ignorableWhitespace(char[] ch, int start, int length)
Receive notification of ignorable whitespace in element content.

void processingInstruction(java.lang.String target, java.lang.String data)
Receive notification of a processing instruction.

void setDocumentLocator(Locator locator)
Receive an object for locating the origin of SAX document events.

void skippedEntity(java.lang.String name)
Receive notification of a skipped entity.

void startDocument()
Receive notification of the beginning of a document.

void startElement(java.lang.String uri, java.lang.String localName, java.lang.String qName, Attributes att
Receive notification of the beginning of an element.

void startPrefixMapping(java.lang.String prefix, java.lang.String uri)
Begin the scope of a prefix-URI Namespace mapping.

# Simple API XML (SAX)

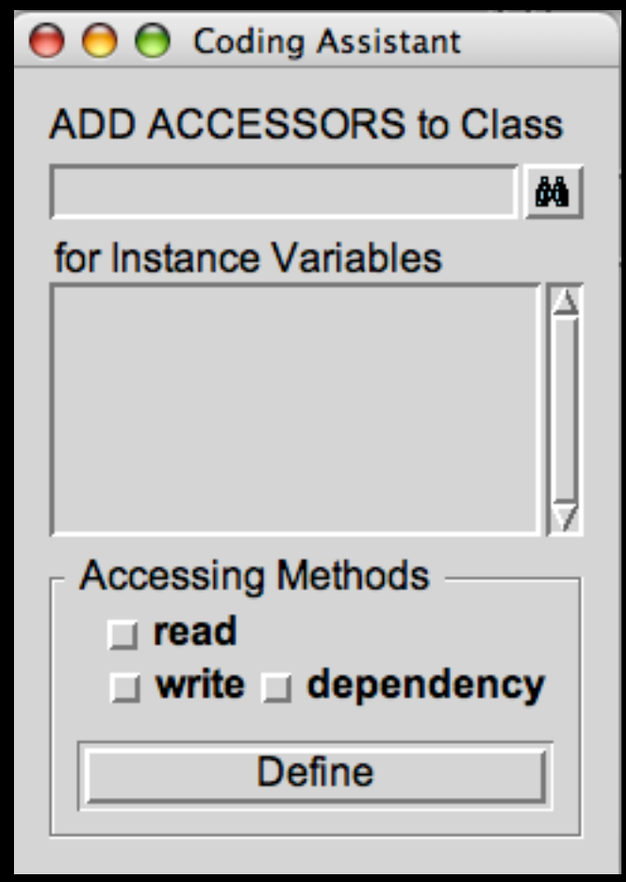
```
public static void main (String args[]) throws Exception {
  XMLReader director = XMLReaderFactory.createXMLReader();
  ContentHandler builder = new MySAXApp();
  director.setContentHandler(builder);
  director.setErrorHandler(builder);

  FileReader source = new FileReader("Foo.xml");
  director.parse(new InputSource(source));
  handler.getResult();
  }
```

# Examples - VW Smalltalk

ClassBuilder
MenuBuilder
UIBuilder

# UIBuilder

```
#(#{UI.FullSpec}
    #window:
    #(#{UI.WindowSpec}
      #label: #(#{Kernel.UserMessage} #key: #CodingAssistant
            #defaultString: 'Coding Assistant' #catalogID: #UIPainter)
      #min: #(#{Core.Point} 242 320 )
      #max: #(#{Core.Point} 242 320 )
      #bounds: #(#{Graphics.Rectangle} 279 140 521 460 ) )
    #component:
    #(#{UI.SpecCollection}
      #collection: #(
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 14 0 12 0 )
          #label: #(#{Kernel.UserMessage} #key: #ADDACCESSORSToClass
            #defaultString: 'ADD ACCESSORS to Class' #catalogID: #UIPainter) )
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 16 0 65 0 )
          #label: #(#{Kernel.UserMessage} #key: #forInstanceVariables
              #defaultString: 'for Instance Variables' #catalogID: #UIPainter) )
```



# Simplified Builder Pattern

More common the standard Pattern

Used to set multiple fields

Replaces using constructor with many parameters

# Person Example

```
public class Person
{
   private final String lastName;
   private final String firstName;
   private final String middleName;
   private final String salutation;
   private final String suffix;
   private final String streetAddress;
   ...
   private final boolean isEmployed;
```

# PersonBuilder

```
public class PersonBuilder
{
   private String newLastName;
   private String newFirstName;
   private String newMiddleName;
   private String newSalutation;
   private String newSuffix;
   private String newStreetAddress;
...
private boolean newIsEmployed;

public PersonBuilder setLastName(String newLastName) {
   this.newLastName = newLastName;
   return this;
}

public PersonBuilder setFirstName(String newFirstName) {
   this.newFirstName = newFirstName;
   return this;
}
```

# PersonBuilder - Continued

```
public PersonBuilder setMiddleName(String newMiddleName) {
      this.newMiddleName = newMiddleName;
      return this;
   }
```

The rest of the set methods

```
public Person createPerson() {
      return new Person(newLastName, newFirstName, newMiddleName,
newSalutation, newSuffix, newStreetAddress, newCity, newState, newIsFemale,
newIsEmployed, newIsHomeOwner);
   }
```

# Building a Person

```
Person test = new PersonBuilder().
        setLastName("Whitney").
        setFirstName("Roger").
        ...
        setIsEmployed(true).
        createPerson();
```

# Improvements

Make Builder an inner class (Java)

Group fields into separate classes
- Name Class
  - firstName
  - lastName
  - middleName
  - salutation
  - suffix

# Android Example

Building a Notification

```
Notification note = new Notification.Builder(mContext)
        .setContentTitle("New mail from " + sender.toString())
.setContentText(subject)
.setSmallIcon(R.drawable.new_mail)
.setLargeIcon(aBitmap)
.build();
```

Strategy
vs
Builder

# Prototype

# Prototype

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

**Applicability**

Use the Prototype pattern when

A system should be independent of how its products are created, composed, and represented; and

When the classes to instantiate are specified at run-time; or

To avoid building a class hierarchy of factories that parallels the class hierarchy of products; or

When instances of a class can have one of only a few different combinations of state.

# Insurance Example

Insurance agents start with a standard policy and customize it

Two basic strategies:

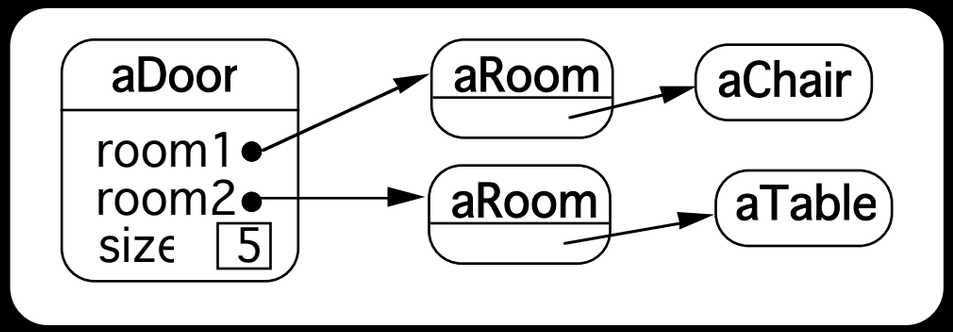
Copy the original and edit the copy

Store only the differences between original and the customize version in a decorator

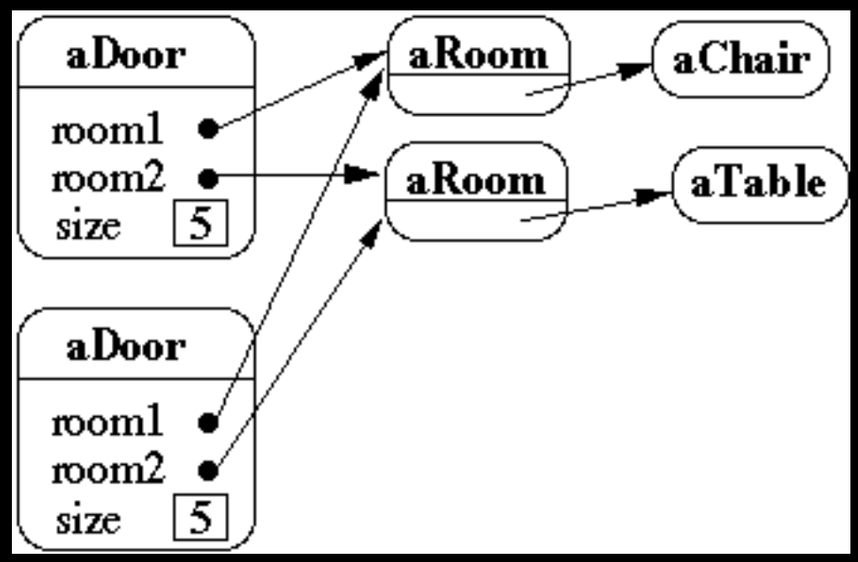
# Copying Issues

Shallow Copy Verse Deep Copy

Original Objects

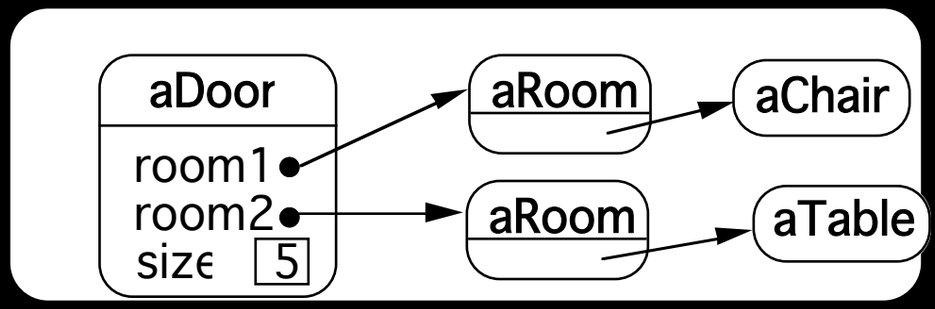



Shallow Copy

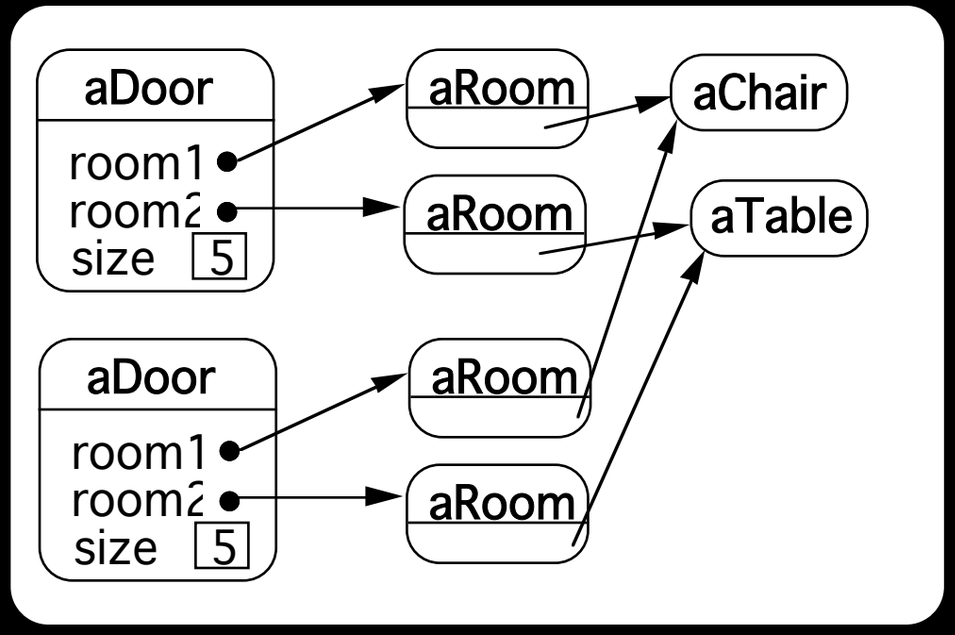


# Shallow Copy Verse Deep Copy

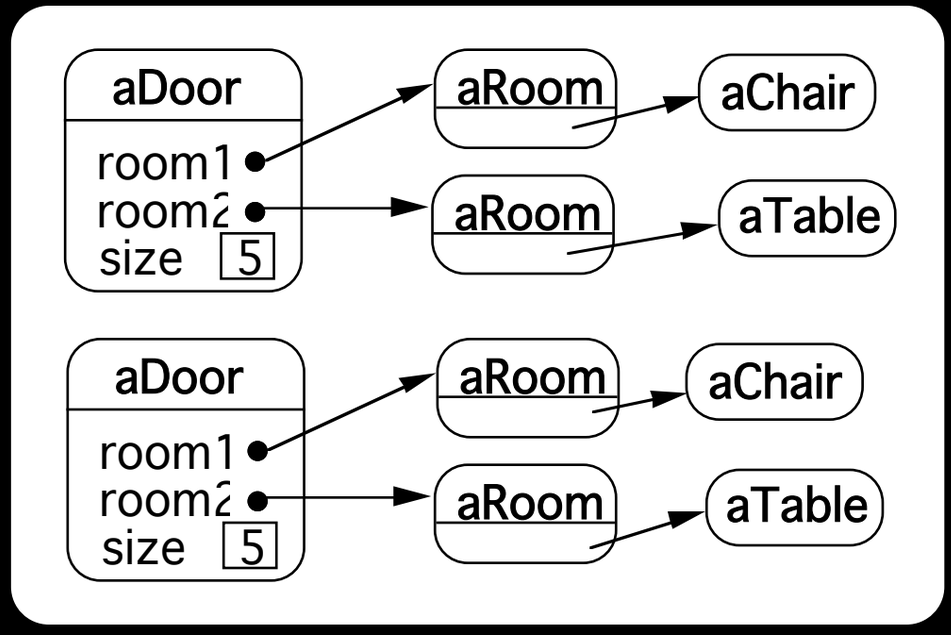
Original Objects




Deep Copy




Deeper Copy



# Prototype-based Languages

No classes

Behavior reuse (inheritance)
  Cloning existing objects which serve as prototypes

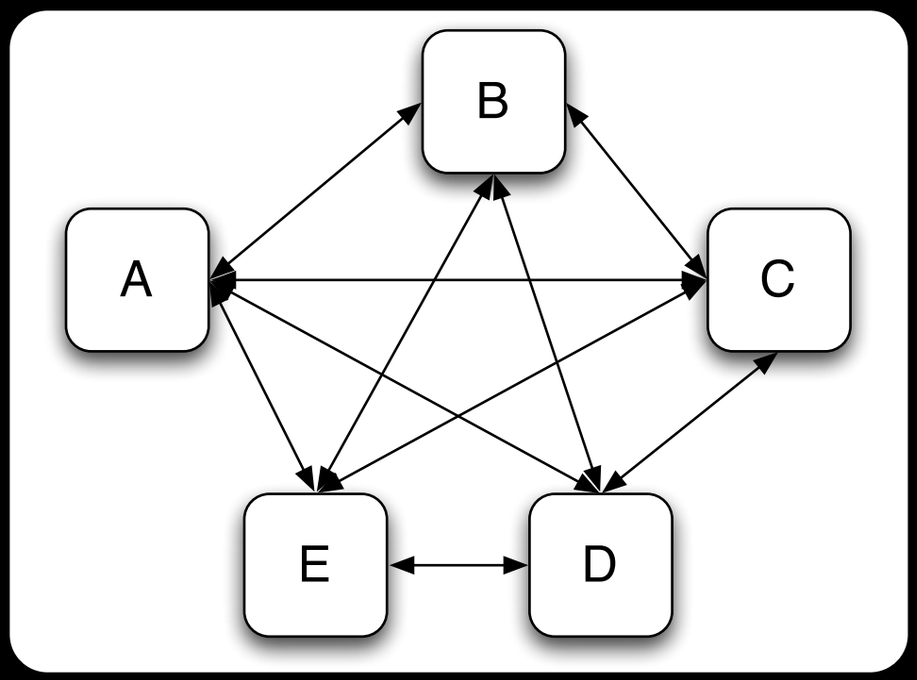
Some Prototype-based languages

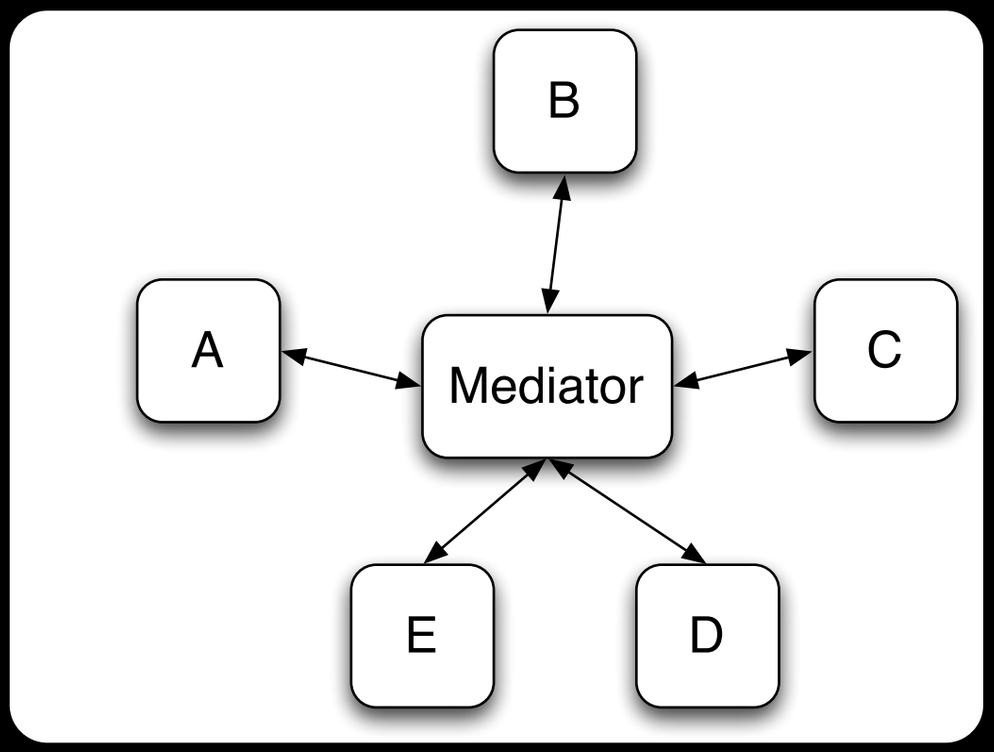
  Self
  JavaScript
  Squeak (eToys)
  Perl with Class::Prototyped module

# Mediator

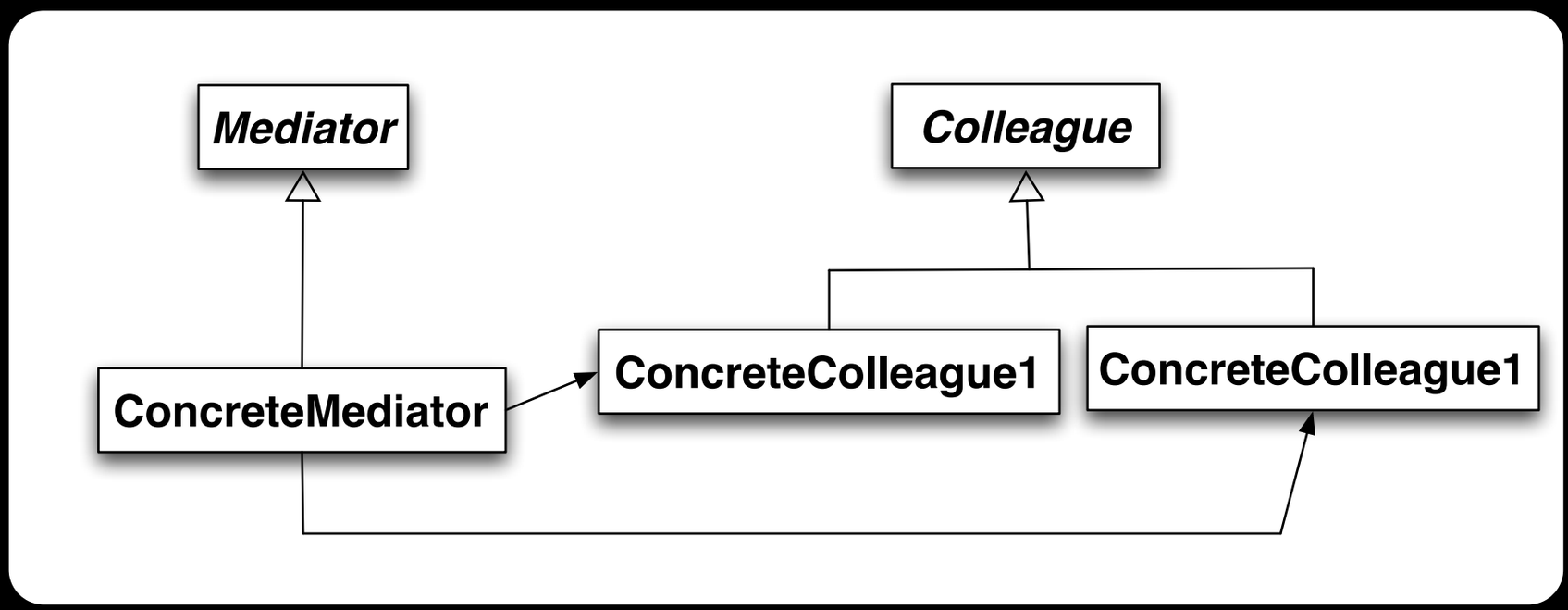
# Mediator

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

ConcreteMediator

Implements cooperative behavior by coordinating Colleague objects

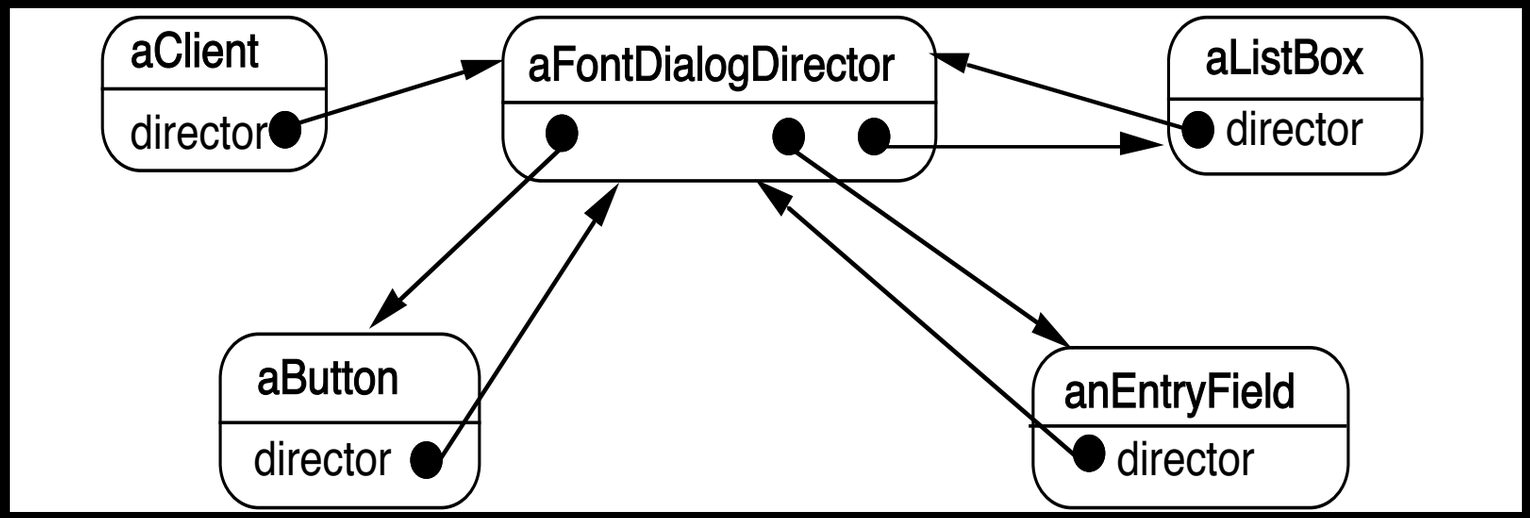
Knows and maintains its colleagues

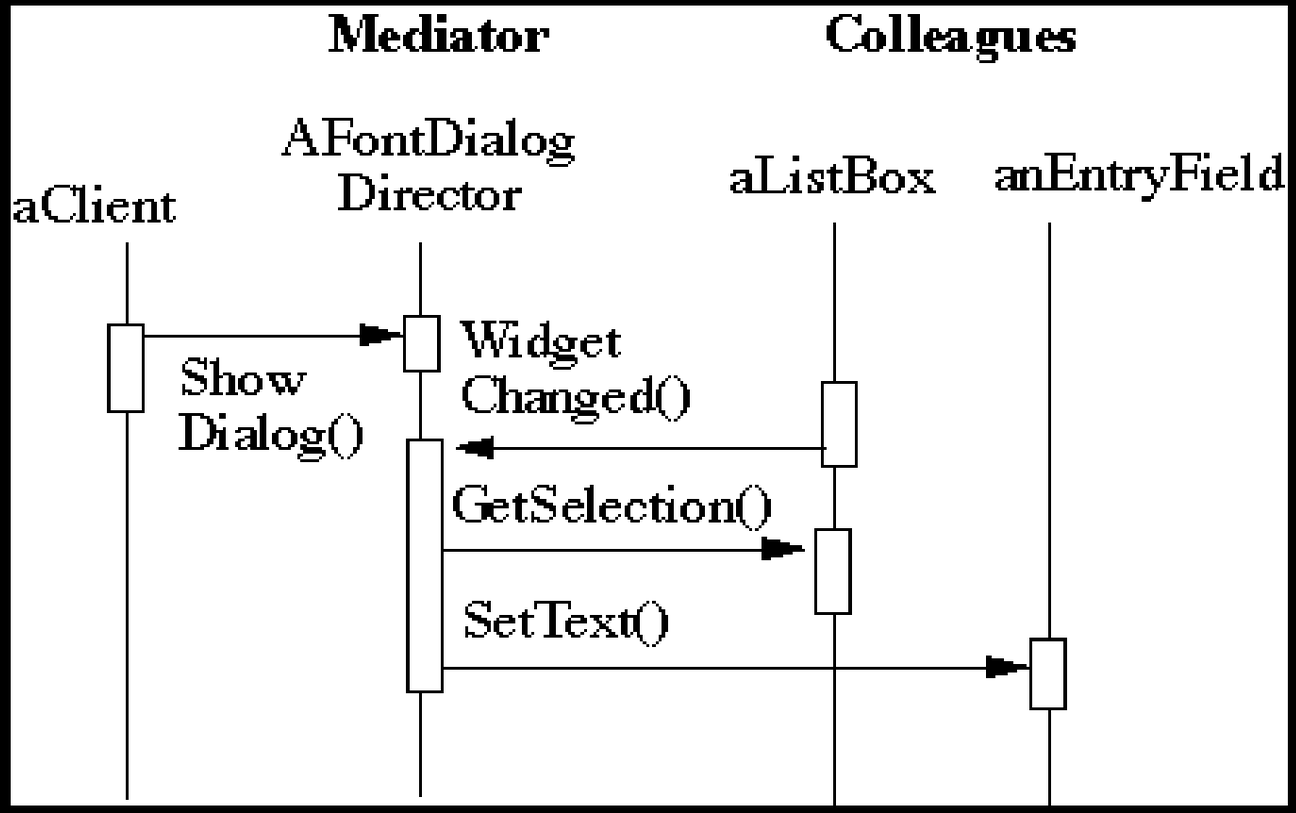
Colleague classes

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?

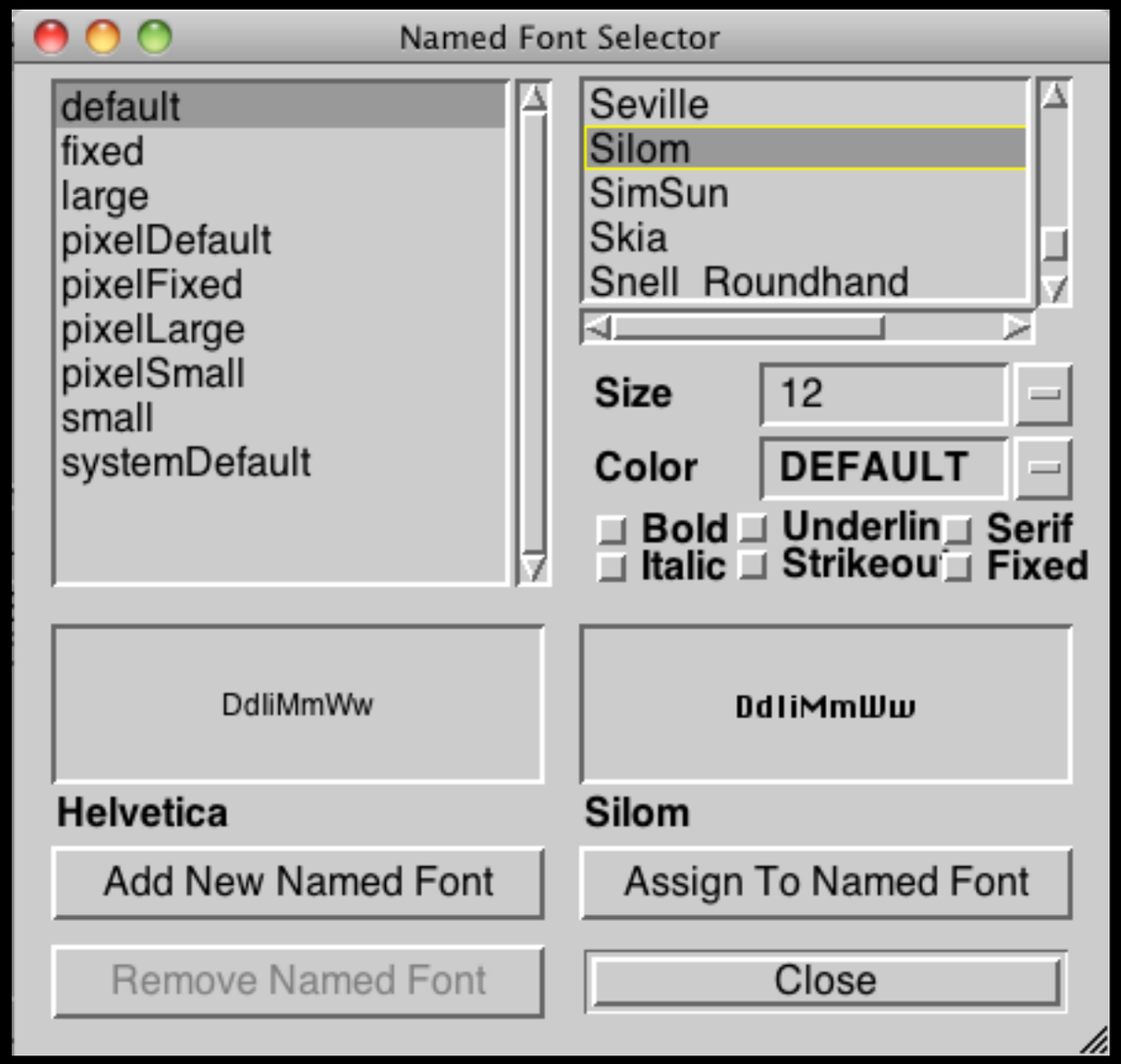
# When to use the Mediator Pattern

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

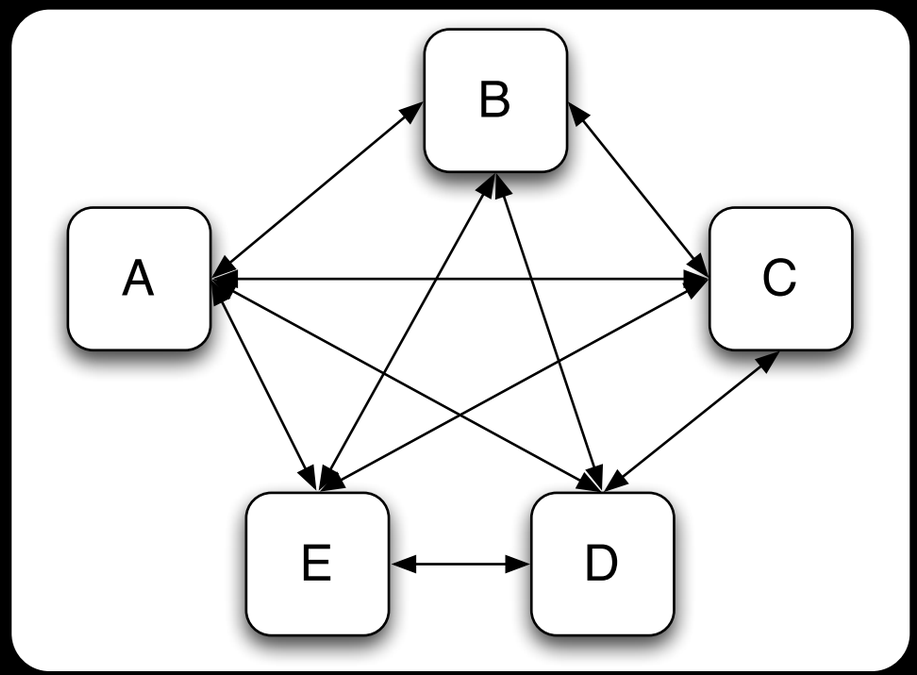
# Classic Mediator Example



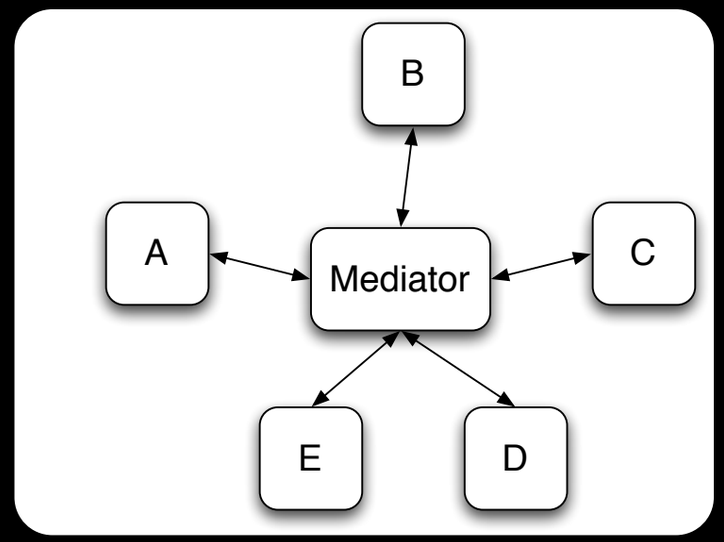
# Simpler Example

# 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

# ReactiveX

In some cases ReactiveX reduces mediator so setting up streams

# 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

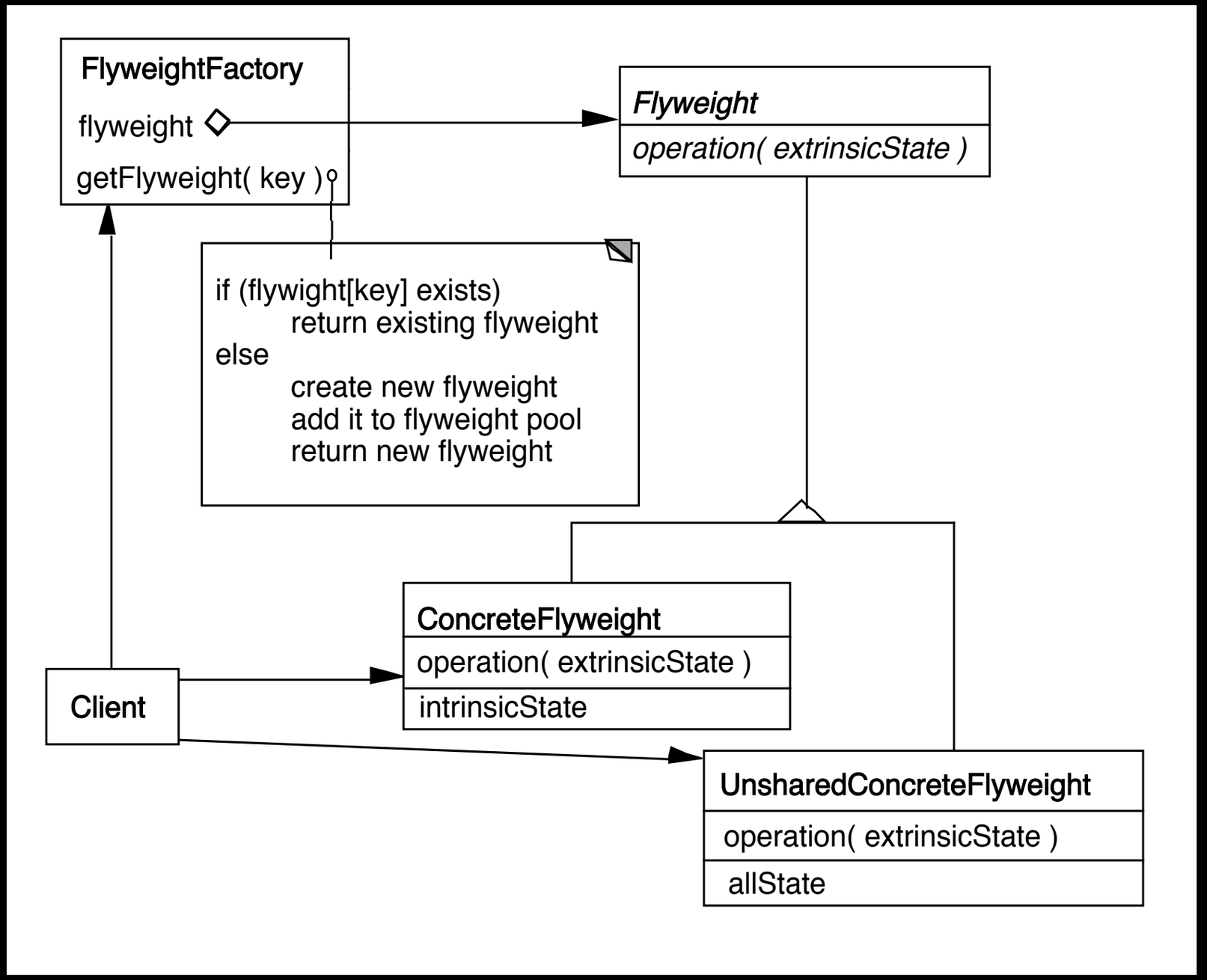
# Extrinsic State

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

aaaaabaaaaaaaaaaaaaaaaaa

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 |