CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2014
Doc 18 Flyweight, Builder, Composite
Apr 22, 2014

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

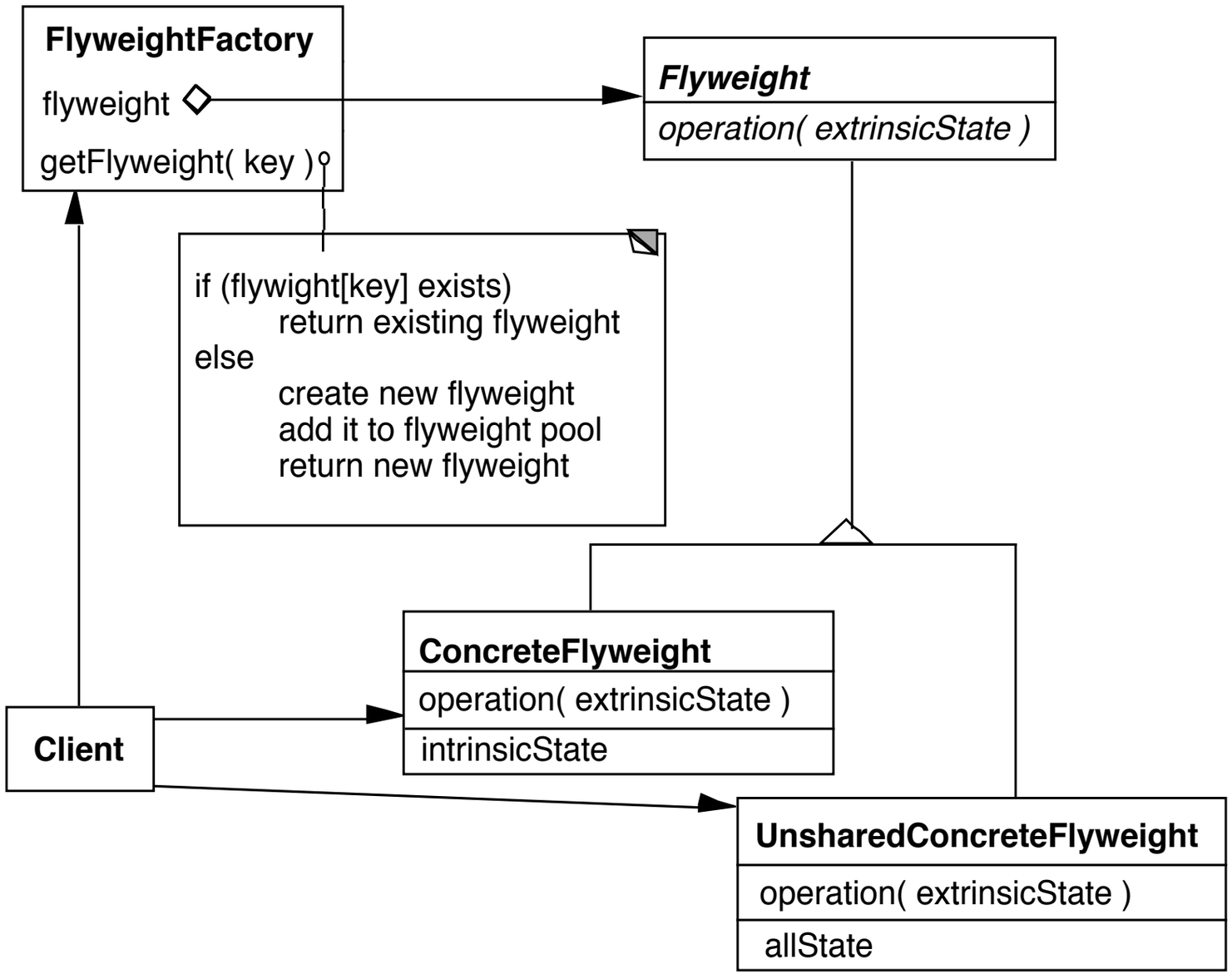
Extrinsic State

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

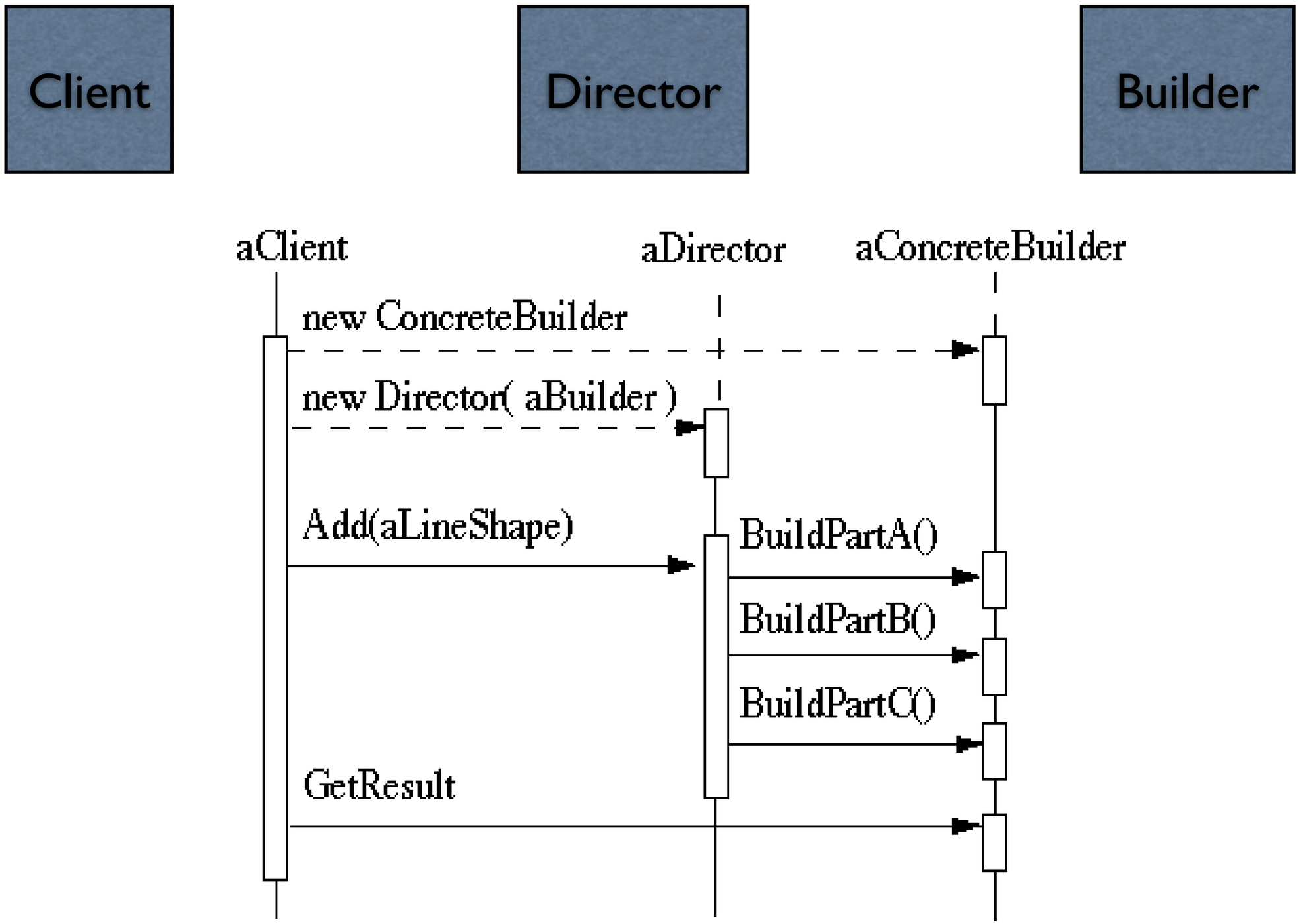
Builder

Builder

Separate construction of a complex object from its representation

So same construction process can create different representations

Builder


Client
Director
Builder
aClient
aDirector
aConcreteBuilder
new ConcreteBuilder
new Director(aBuilder)
Add(aLineShape)
BuildPartA()
BuildPartB()
BuildPartC()
GetResult

RTF Converter

A word processing document has complex structure

How to convert Rich Text Format (RTF) to

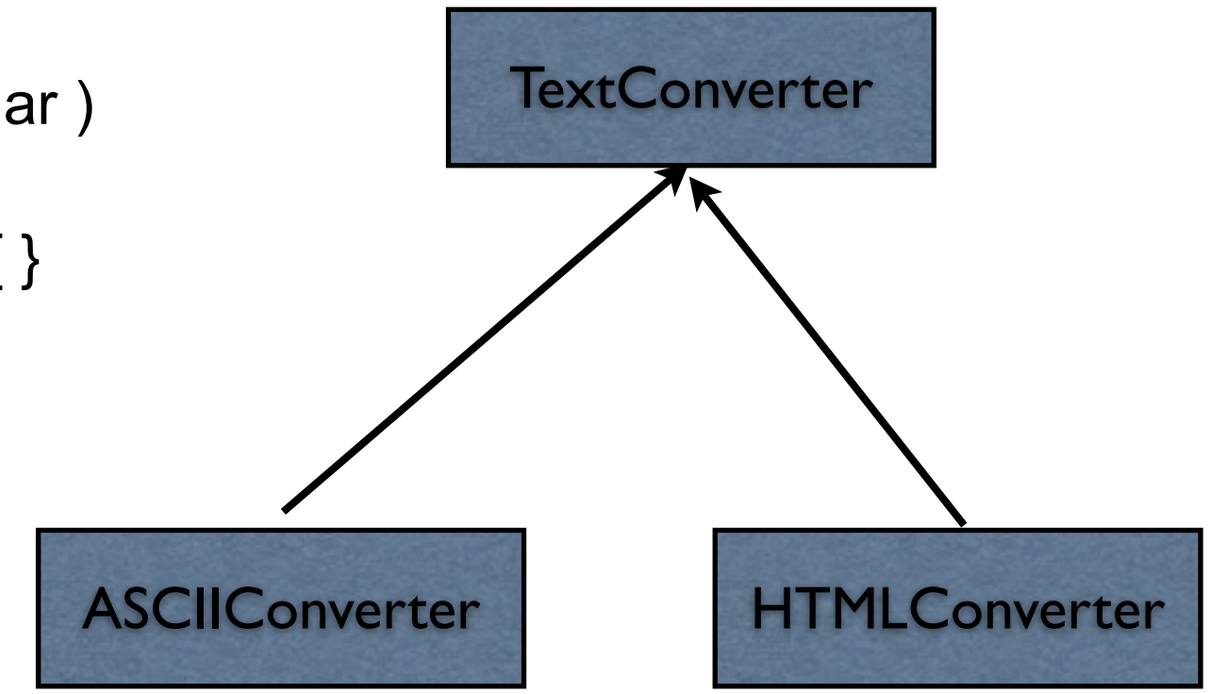
- TeX
- html
- PDF
- etc.

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 atts)
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)

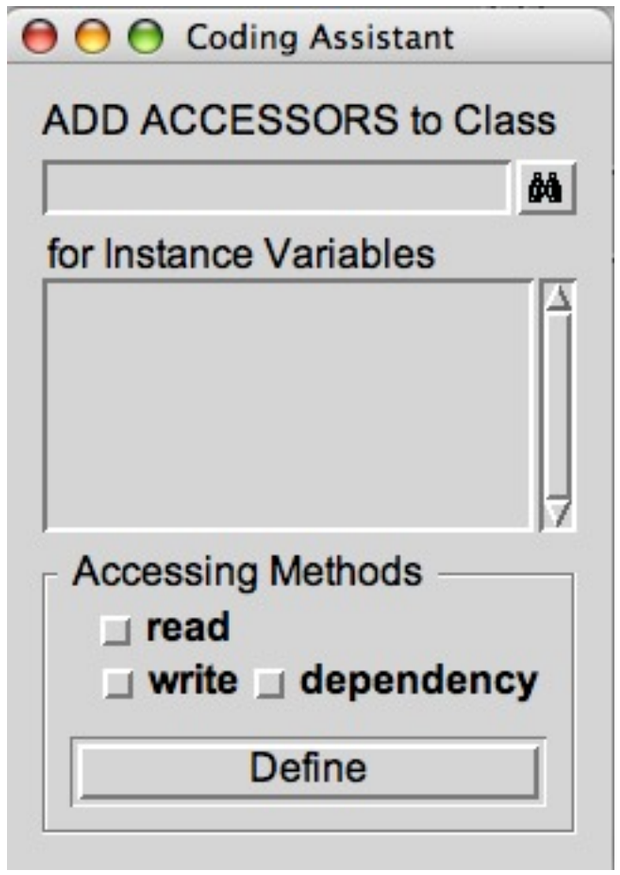
```
public static void main (String args[]) throws Exception {
  XMLReader director = XMLReaderFactory.createXMLReader();
  ContentHandler builder = new MySAXApp();
  director.setContentHandler(builder);
  director.setErrorHandler(builder);

  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;
```

```
public Person(
      final String newLastName,
      final String newFirstName,
      final String newMiddleName,
      final String newSalutation,
      final String newSuffix,
      final String newStreetAddress,
      ...
      final boolean newIsEmployed) {
            this.lastName = newLastName;
         this.firstName = newFirstName;
            ...
}
```


http://www.javaworld.com/article/2074938/core-java/too-many-parameters-in-java-methods-part-3-builder-pattern.html

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)

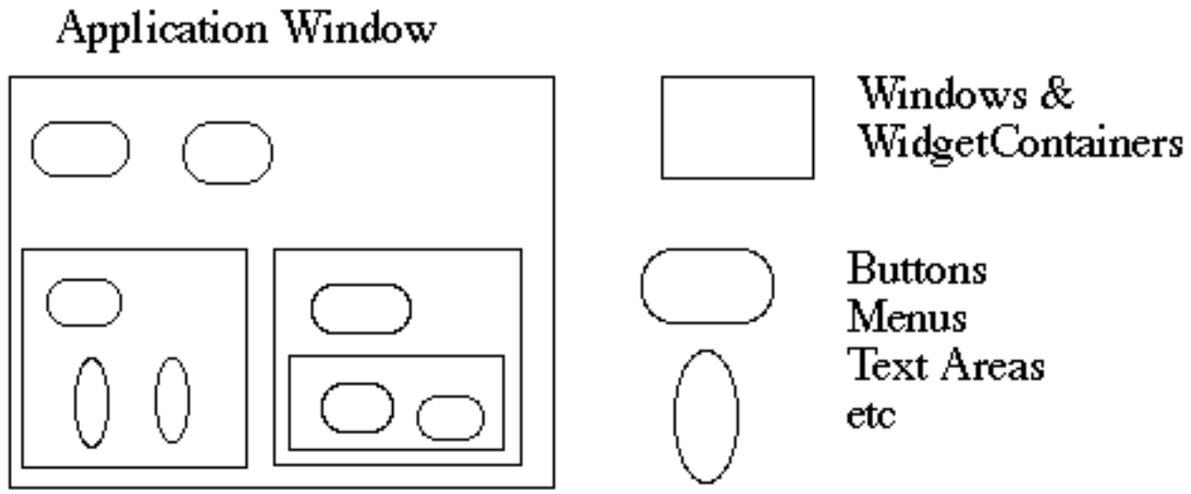
Group fields into separate classes

- Name Class
  - firstName
  - lastName
  - middleName
  - salutation
  - suffix

Strategy
vs
Builder

Composite

Composite Motivation



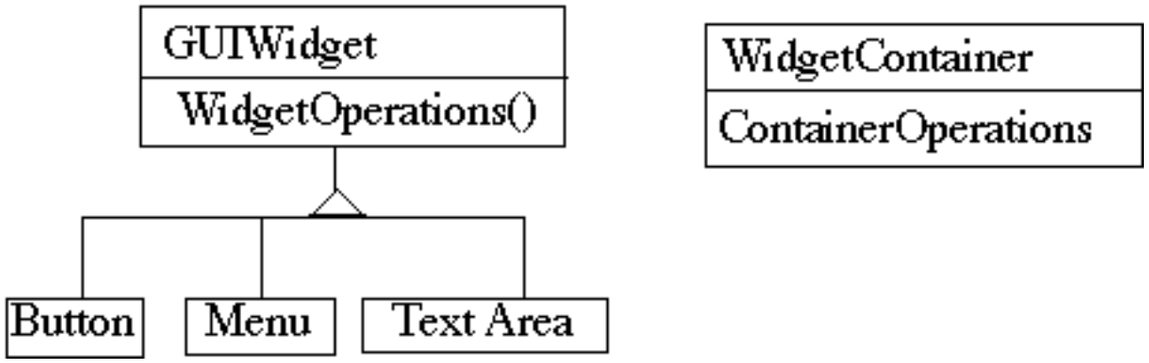
How does the window hold and deal with the different items it has to manage?

Widgets are different that WidgetContainers

Bad News

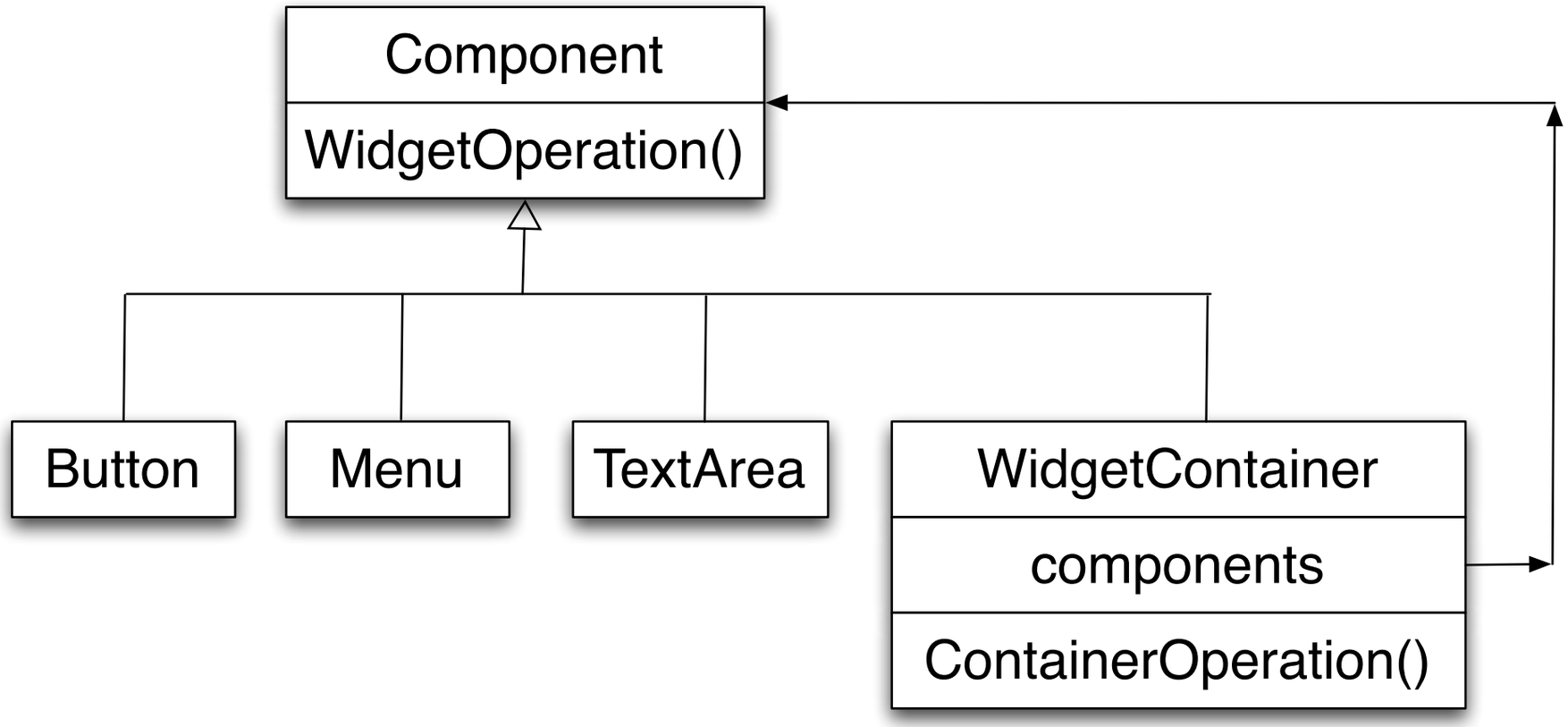
```
class Window {
        Buttons[] myButtons;
        Menus[] myMenus;
        TextAreas[] myTextAreas;
        WidgetContainer[] myContainers;

        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