CS 635 Advanced Object-Oriented Design & Programming

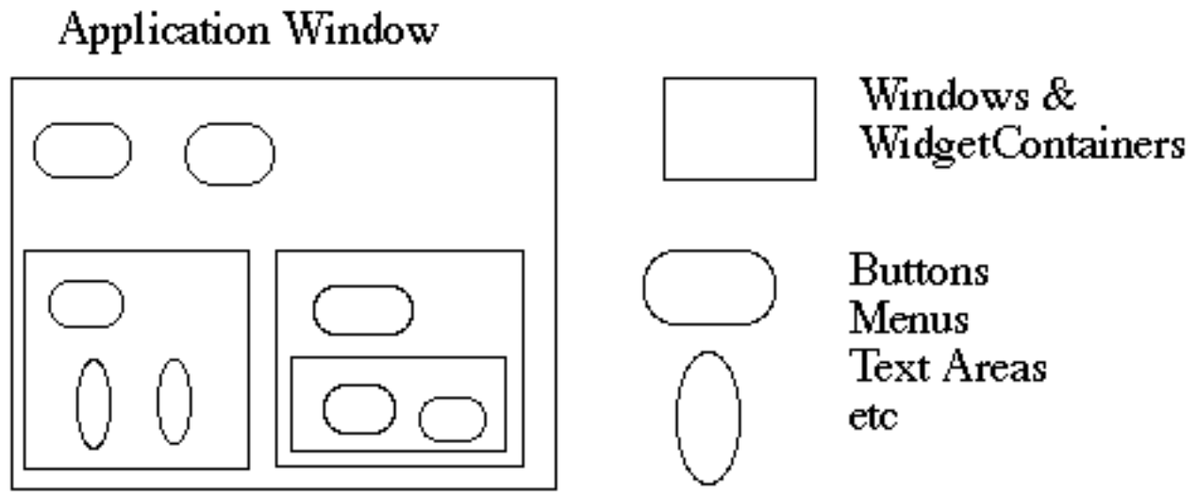
Spring Semester, 2006
Doc 5 Composite
Feb 7, 2006

Copyright ©, All rights reserved. 2006 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 Resuable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995, pp. 163-174

Motivation



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

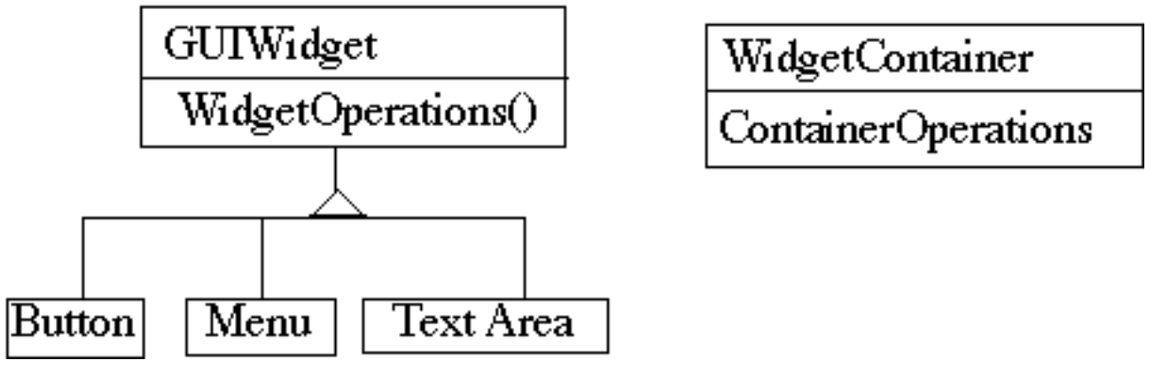
Widgets are different that WidgetContainers

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.
```

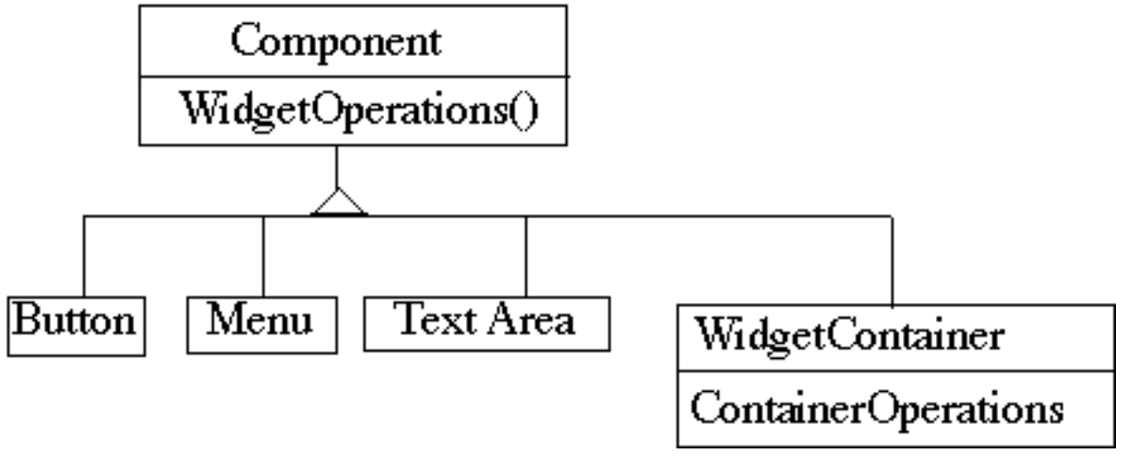
An Improvement




```
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



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 Opeartions

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

Java Example

