Composite Pattern

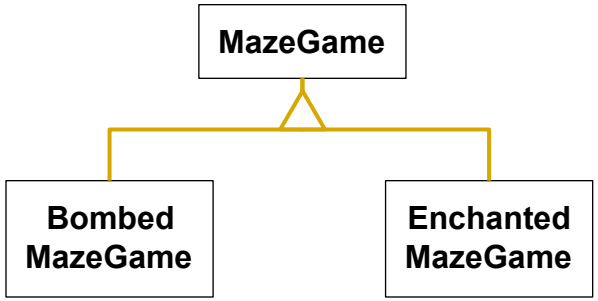
CS356 Object-Oriented Design and Programming
http://cs356.yusun.io
November 3, 2014

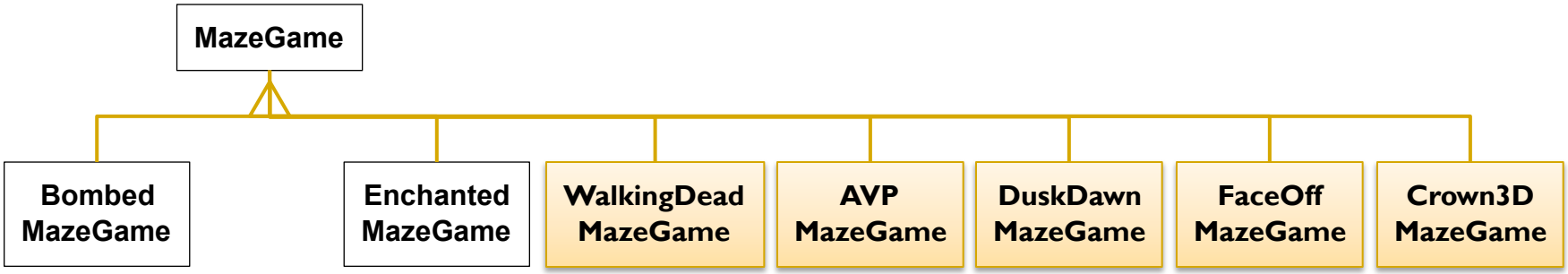
Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

CAL POLY POMONA

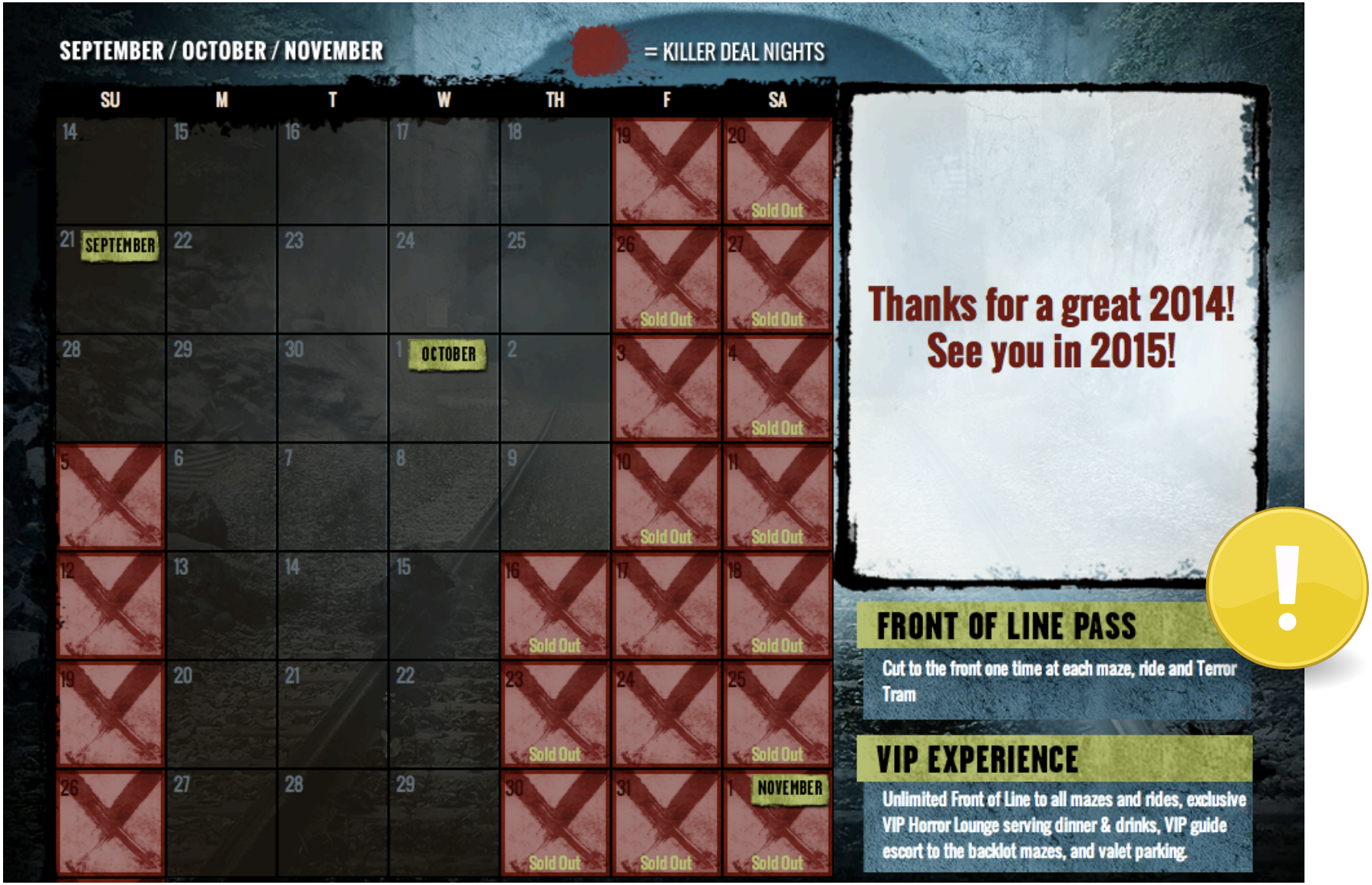
Horror Nights is Great

Design Pattern?





Suggestion

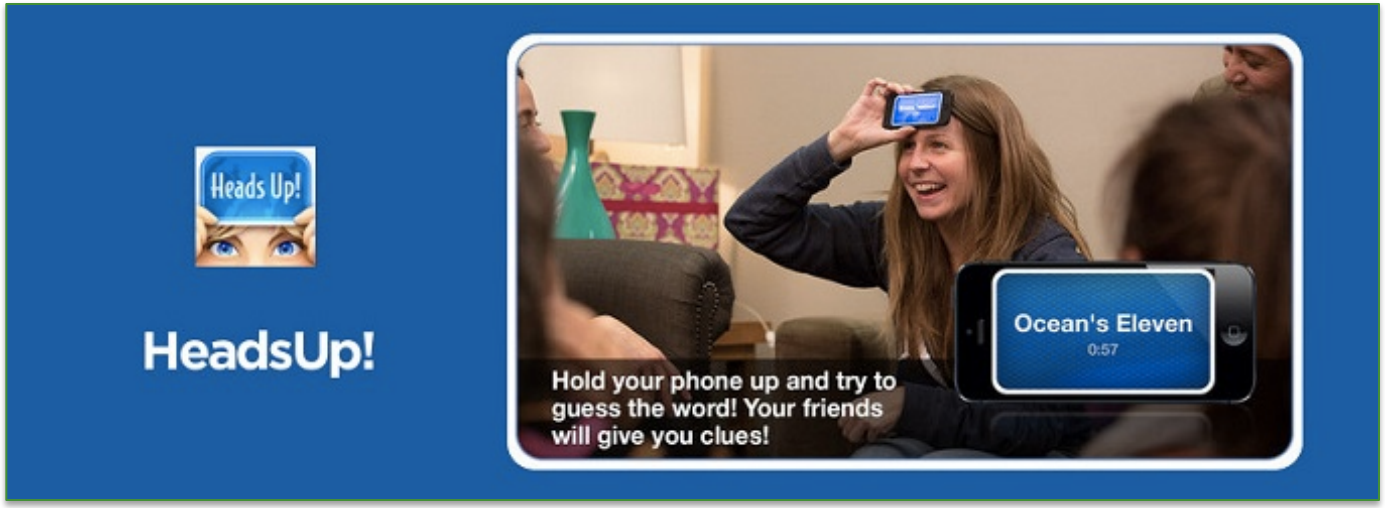

SEPTEMBER / OCTOBER / NOVEMBER
= KILLER DEAL NIGHTS
SU M T W TH F SA
14 15 16 17 18 19 20 Sold Out
21 SEPTEMBER 22 23 24 25 26 Sold Out 27 Sold Out
28 29 30 1 OCTOBER 2 3 4 Sold Out
5 6 7 8 9 10 Sold Out 11 Sold Out
12 13 14 15 16 Sold Out 17 18 Sold Out
19 20 21 22 23 Sold Out 24 25 Sold Out
26 27 28 29 30 Sold Out 31 Sold Out 1 NOVEMBER Sold Out
Thanks for a great 2014!
See you in 2015!
FRONT OF LINE PASS
Cut to the front one time at each maze, ride and Terror Tram
VIP EXPERIENCE
Unlimited Front of Line to all mazes and rides, exclusive VIP Horror Lounge serving dinner & drinks, VIP guide escort to the backlot mazes, and valet parking.

Problem


GRINGOTTS
WAIT TIME
450
MINUTES
WARNING!
© Twitter.com/InsideTheMagic


STUDIO DIRECTORY
LAST TERROR TRAM 11:45PM
NEXT PERFORMANCE
LIVE SHOWS
RIDES & ATTRACTIONS
WAIT TIME
120 min
150 min
70 min
75 min
30 min
30 min
45 min
TERROR TRAM
JURASSIC PARK IN THE DARK
MUMMY
SIMPSONS RIDE
DESPICABLE ME
CHECK WAIT TIMES
HHNWaitTimes.com

Solution 1: Kill Time in the Line



- Senior Project:
  - Apps to help kill the time
  - Context-aware
  - Group involvement
  - Social
  - Move and exercise

Solution 2: Be Smart on Wait Times

- Not very useful inside the park
- Historical data might be more useful
- Senior Project:
  - Crawling and saving the data
  - Wait Time Prediction
  - Trip planner

Composite Pattern

CS356 Object-Oriented Design and Programming
http://cs356.yusun.io
November 3, 2014

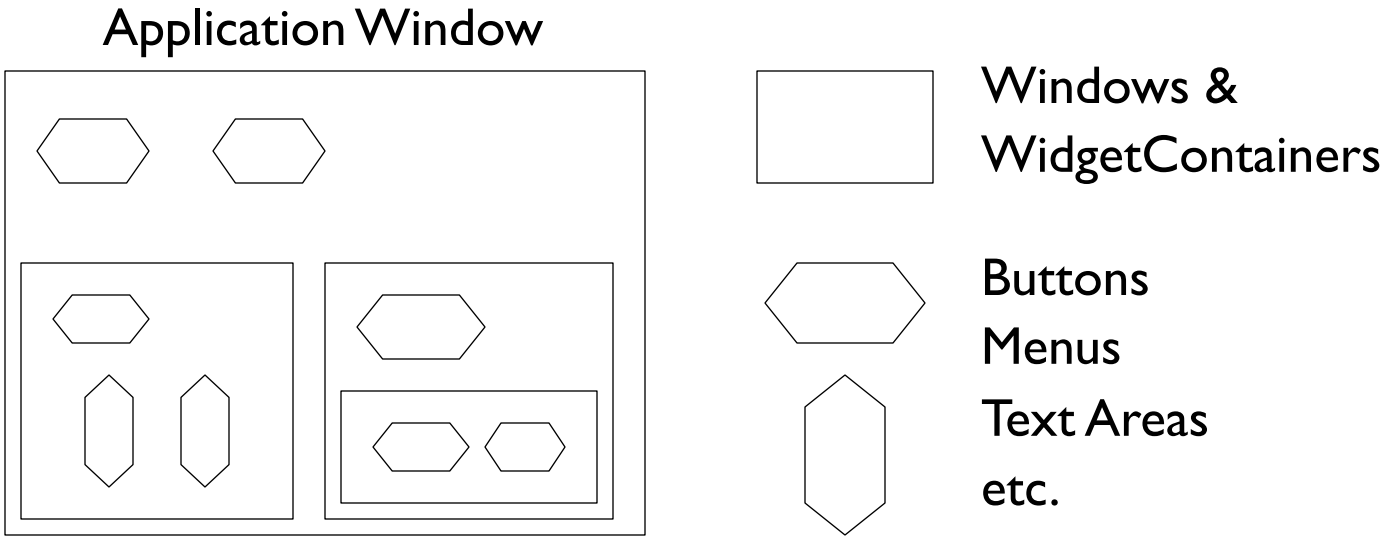
Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

CAL POLY POMONA

Composite

- *Intent*
  - Compose objects into tree structures to represent part-whole hierarchies
  - Individual and compositions of objects treated uniformly
- *Applicability*
  - To represent whole-part hierarchies of objects
  - To allow clients to ignore the difference between composition and individual objects

Motivation



- GUI Windows and GUI elements
  - How does the window manage with the different items?
  - Widgets are different than WidgetContainers

Nightmare Implementation

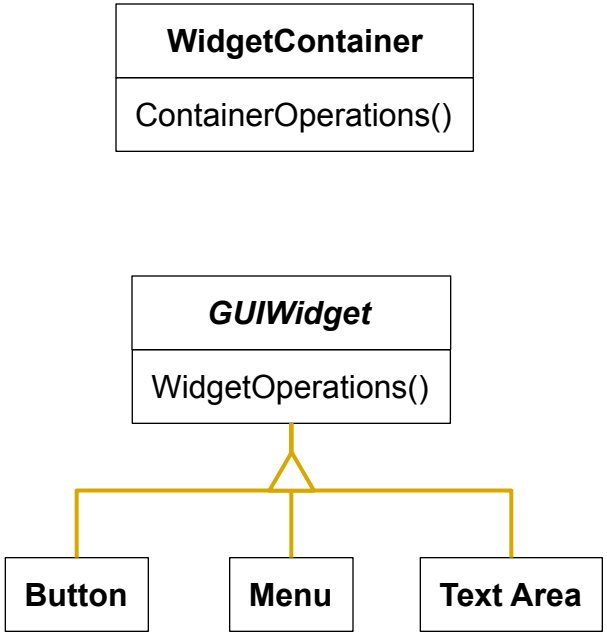
- Deal with each category of objects' operations individually

```
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 < myTextAreas.length(); k++ )
        myTextAreas[k].refresh();
    if ( myContainers != null )
      for (int k = 0; k < myContainers.length();k++)
        myContainers[k].updateElements();
    ...
  }
  ...
```

Program to Interface

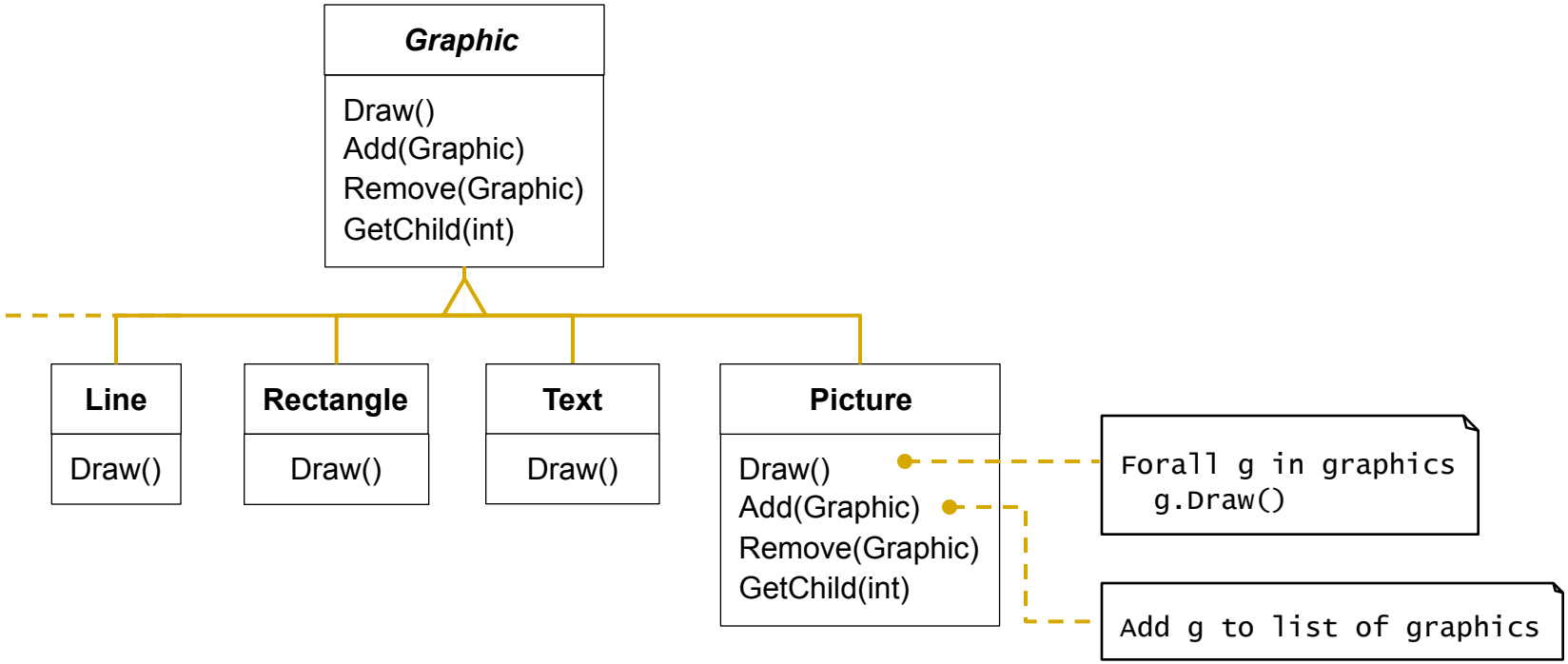
- Uniform dealing with widget operations
- But still containers are treated different
- Still not sufficient

```
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();
    ...
  }
}
```

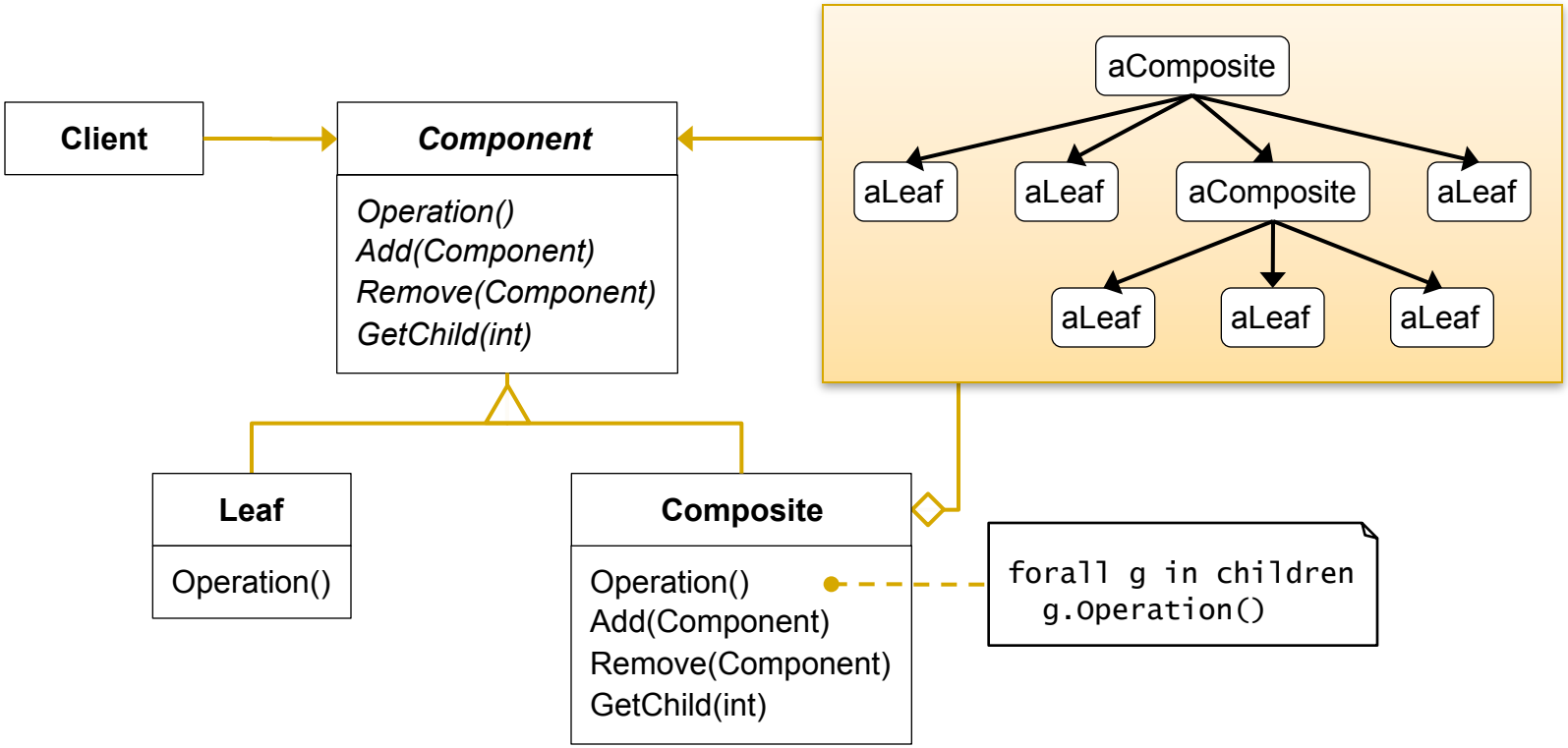


Composite Pattern Solution

- Encapsulate composite and simple objects behind a common interface



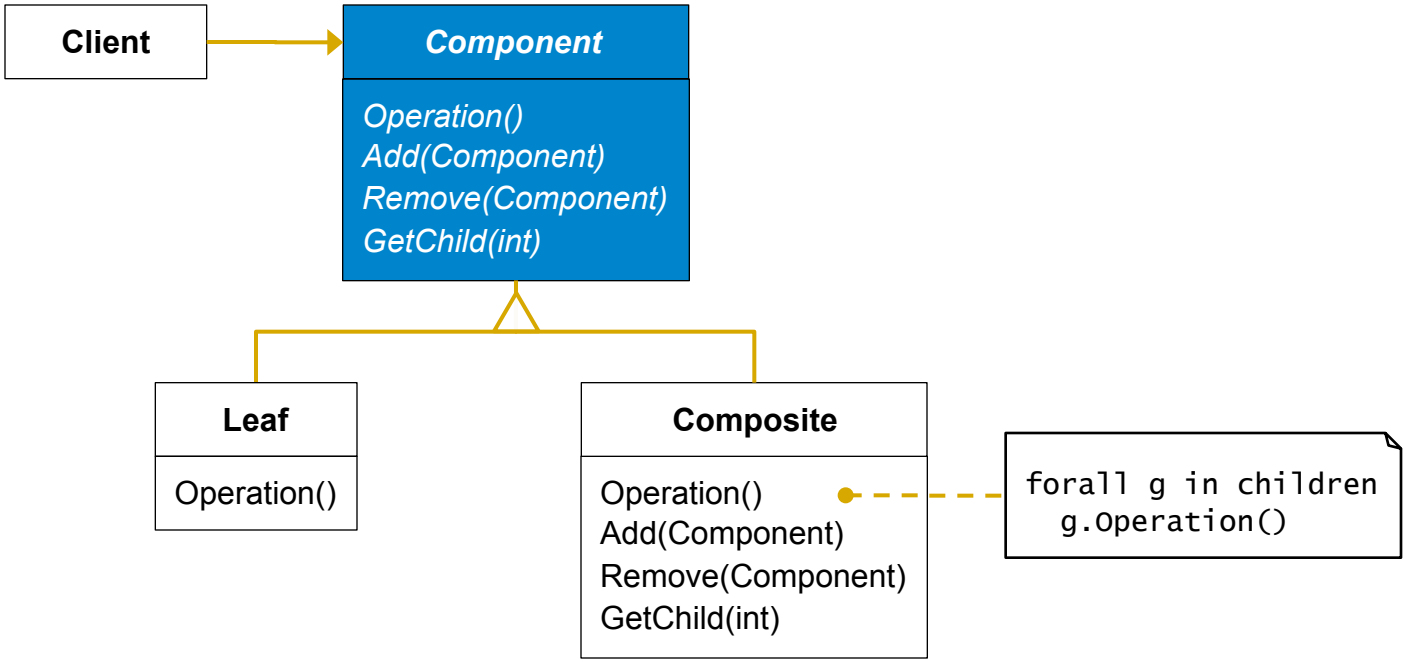
Structure



- Component class role gives a consistent interface
  - *Leaf class* – Components without further sub-structure
  - *Composite class* – Components with multiple parts

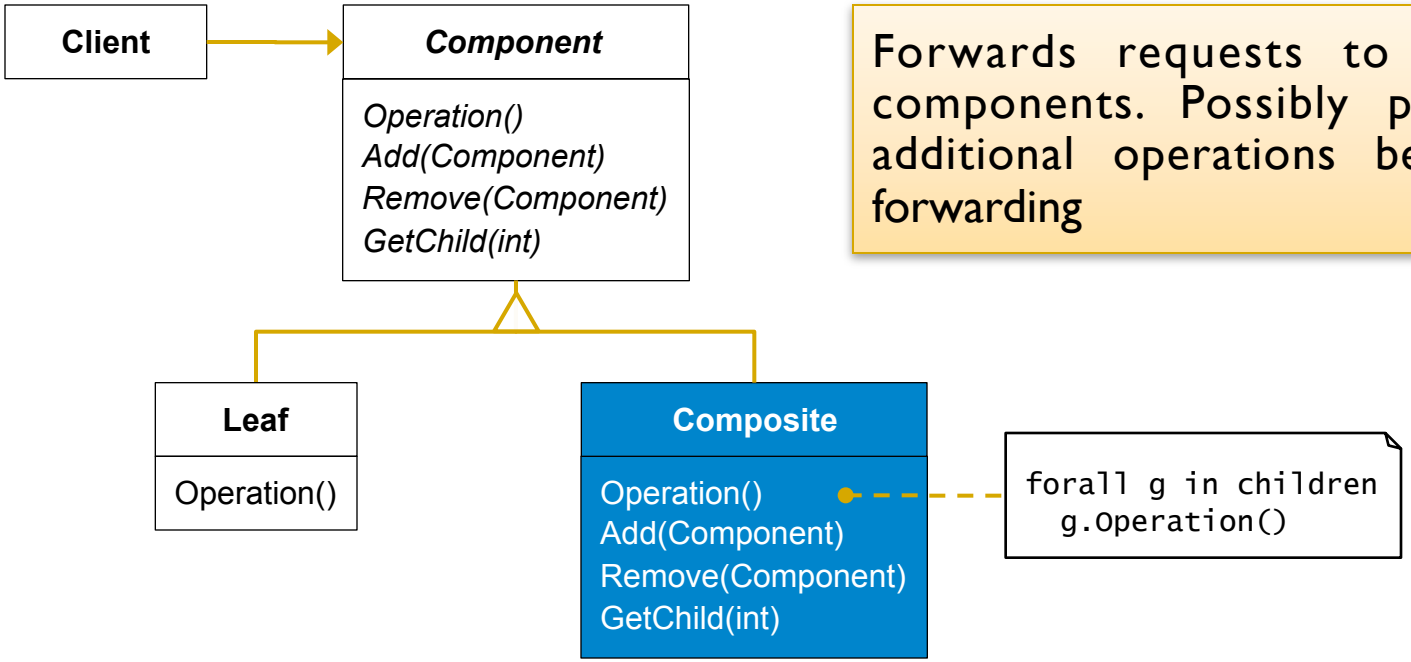
Component

- Declares interface for objects in the composition
- Implements default behavior for components when possible



Composite

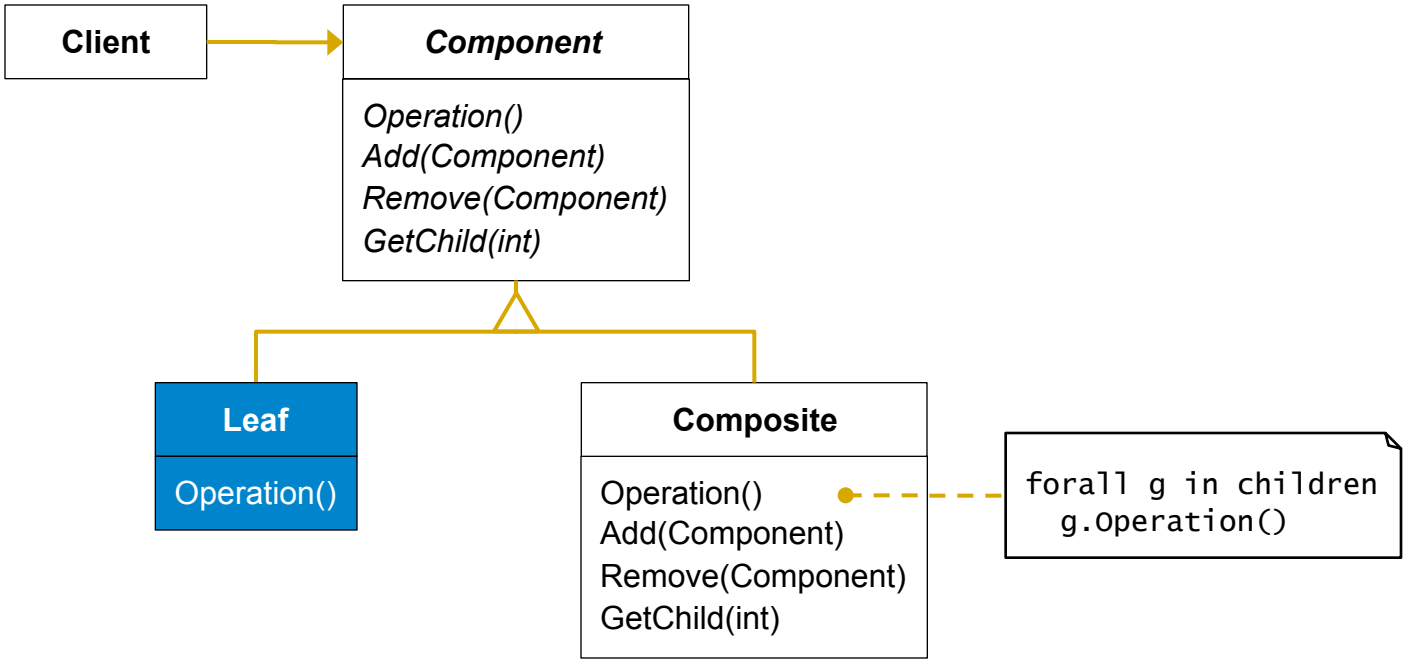
- Defines behavior for components having children
- Stores child components
- Implement child-specific operations



Forwards requests to its child components. Possibly performing additional operations before/after forwarding

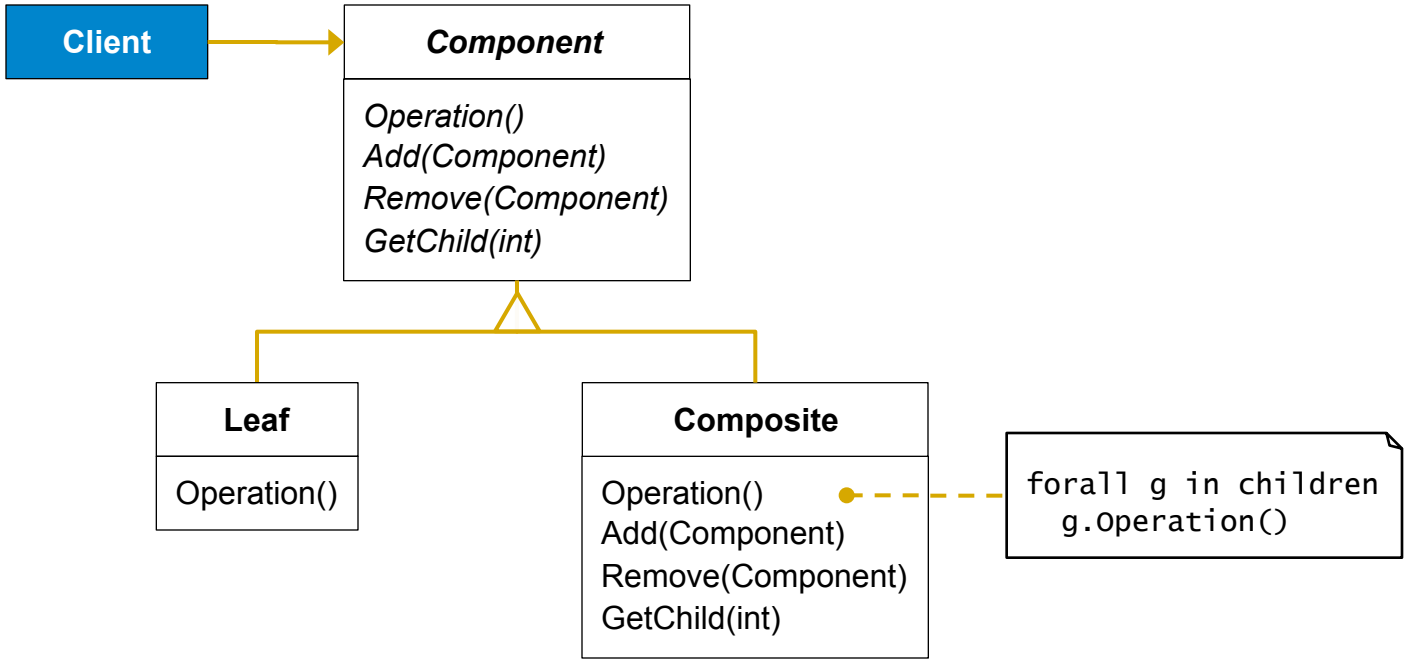
Leaf

- Defines behavior for primitive objects in the composition



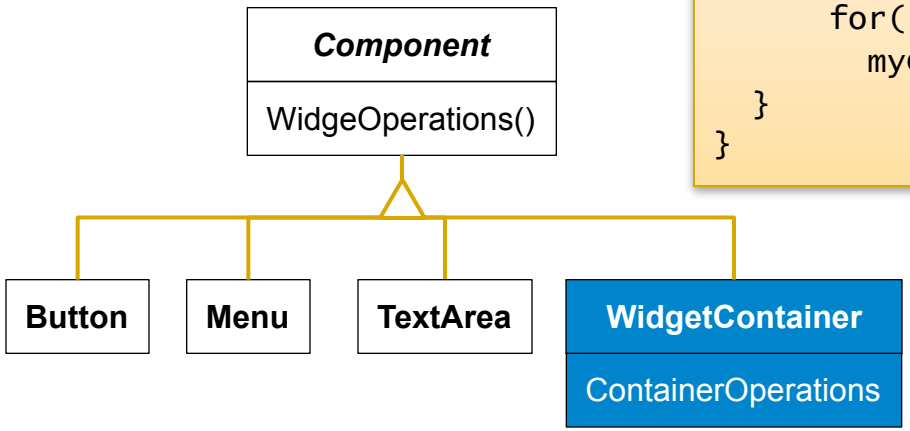
Client

- Manipulates objects in the composition through the Component interface



Applying Composite to Widget Problem

```
class Window {
  Component[] myComponents;

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

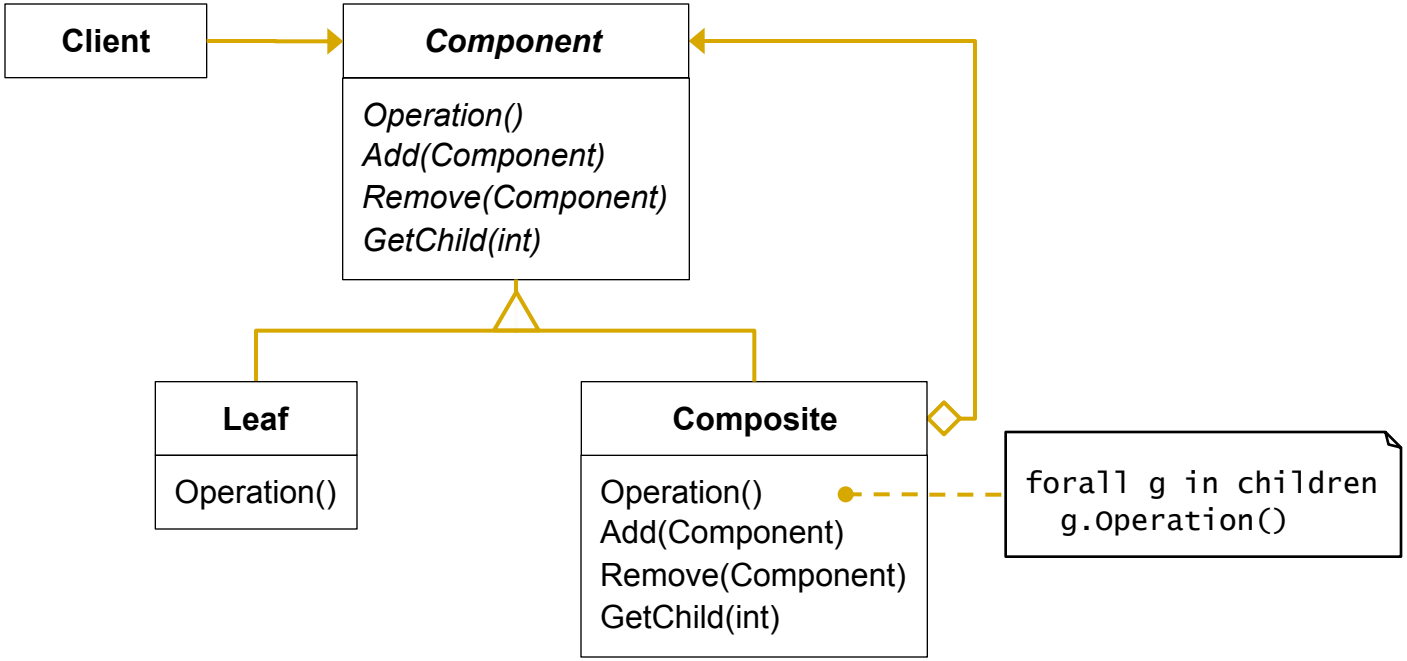


- Component implements default behavior when possible
  - Button, Menu, etc. override Component methods when needed
- WidgetContainer will have to override all widget operations

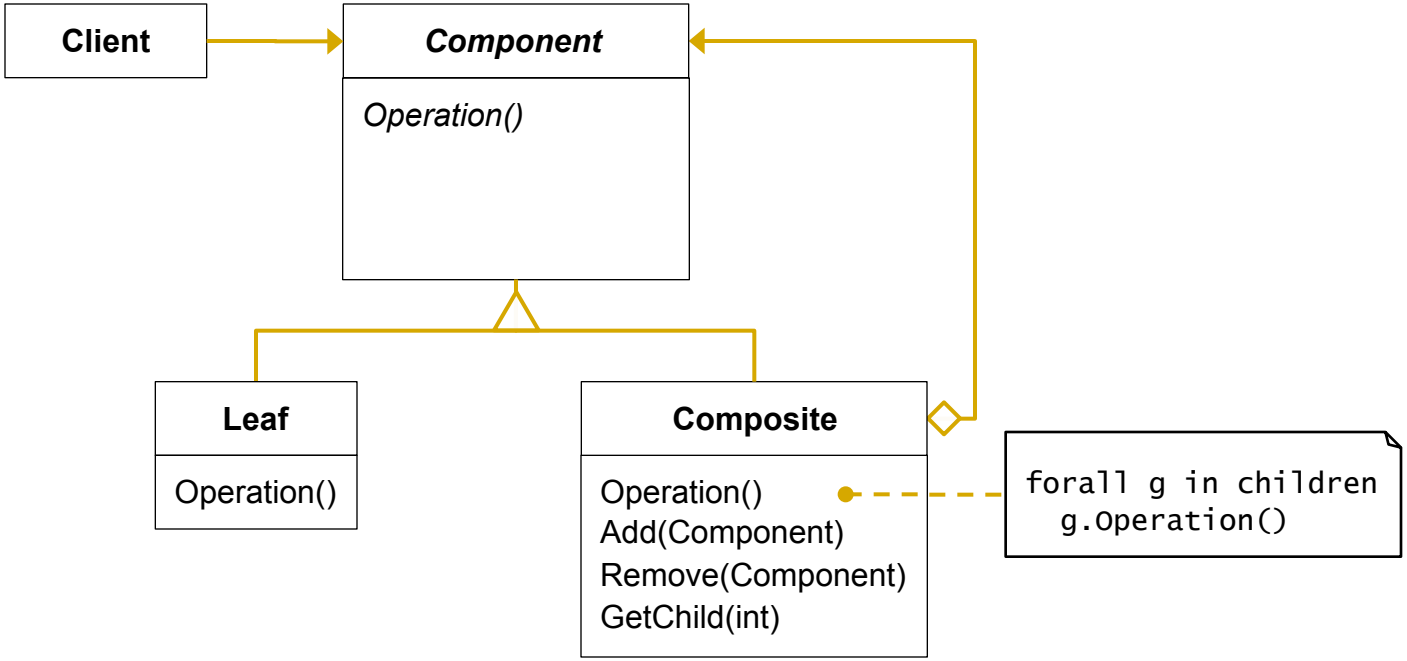
Issue: Where to Place Container Operations?

- Adding, deleting, managing components in composite
  - Should they be placed in Component or in Composite?

Approach 1


Client
Component
Operation()
Add(Component)
Remove(Component)
GetChild(int)
Leaf
Operation()
Composite
Operation()
Add(Component)
Remove(Component)
GetChild(int)
forall g in children
g.Operation()

Approach 2


Client
Component
Operation()
Leaf
Operation()
Composite
Operation()
Add(Component)
Remove(Component)
GetChild(int)
forall g in children
g.Operation()

Issue: Where to Place Container Operations?

- Adding, deleting, managing components in composite
  - Should they be placed in Component or in Composite?
- Pro-Transparency Approach
  - Declaring in the Component gives all subclasses the same interface
    - All subclasses can be treated alike
  - Safety problem
    - Clients may do stupid things like adding objects to leaves
    - What should be the response to adding a TextArea to a button?
      - Throw an exception?
- Pro-Safety Approach
  - Declaring them in WidgetContainer is safer
    - Adding or removing widgets to non-WidgetContainers is an error

Consequences

- Defines uniform class hierarchies
  - Recursive composition of objects
- Make clients simple
  - Don't know whether dealing with a leaf or a composite
  - Avoids dealing in a different manner with each class
- Easier to extend
  - Easy to add new Composite or Leaf classes
  - Awesome example of Open-Closed principle

- Overly general design
  - Harder to restrict what type can be added