CSC207H: Software Design
Lecture 12

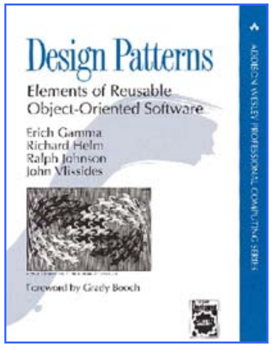
Wael Aboelsaadat

wael@cs.toronto.edu
http://ccnet.utoronto.ca/20075/csc207h1y/
Office: BA 4261
Office hours: R 5-7

Acknowledgement: These slides are based on material by Prof. Karen Reid (University of Toronto), Prof. Bits & Sundar (Purdue Univ.)

Design Patterns

Design Pattern Space

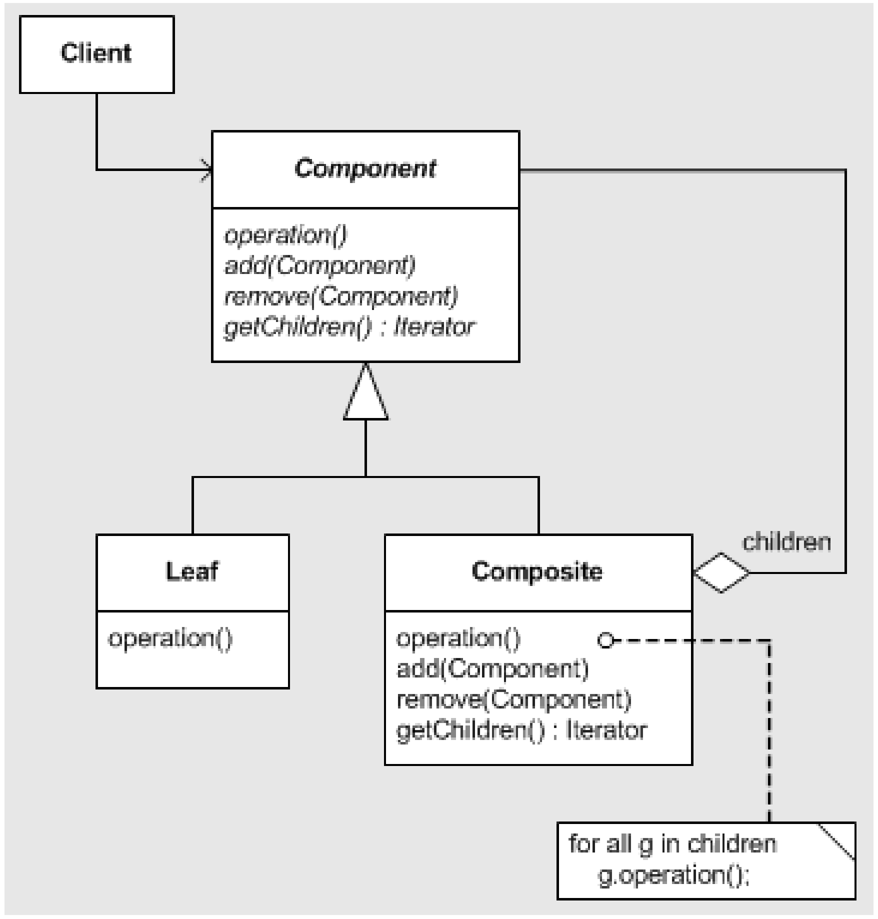

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Creational	**Structural**	**Behavioral**
Factory Method	Adapter	Interpreter
Abstract Factory	Bridge	Template Method
Builder	Composite	Chain of Responsibility
Prototype	Decorator	Command
Singleton	Flyweight	Iterator
	Facade	Mediator
	Proxy	Memento
		Observer
		State
		Strategy
		Visitor

Composite Pattern

- Facilitates the composition of objects into tree structures that represent part-whole hierarchies.
- These hierarchies consist of both primitive and composite objects.

Composite Design Pattern



Composite Pattern – Participants

- Component
  - Declares interface for objects and for accessing children
  - Implements default behavior
- Leaf
  - No children; defines behavior for primitive objects
- Composite
  - Defines behavior for components with children
  - Stores children and implements children-related operations
- Client
  - Manipulates objects in the composition thru' Component interface.

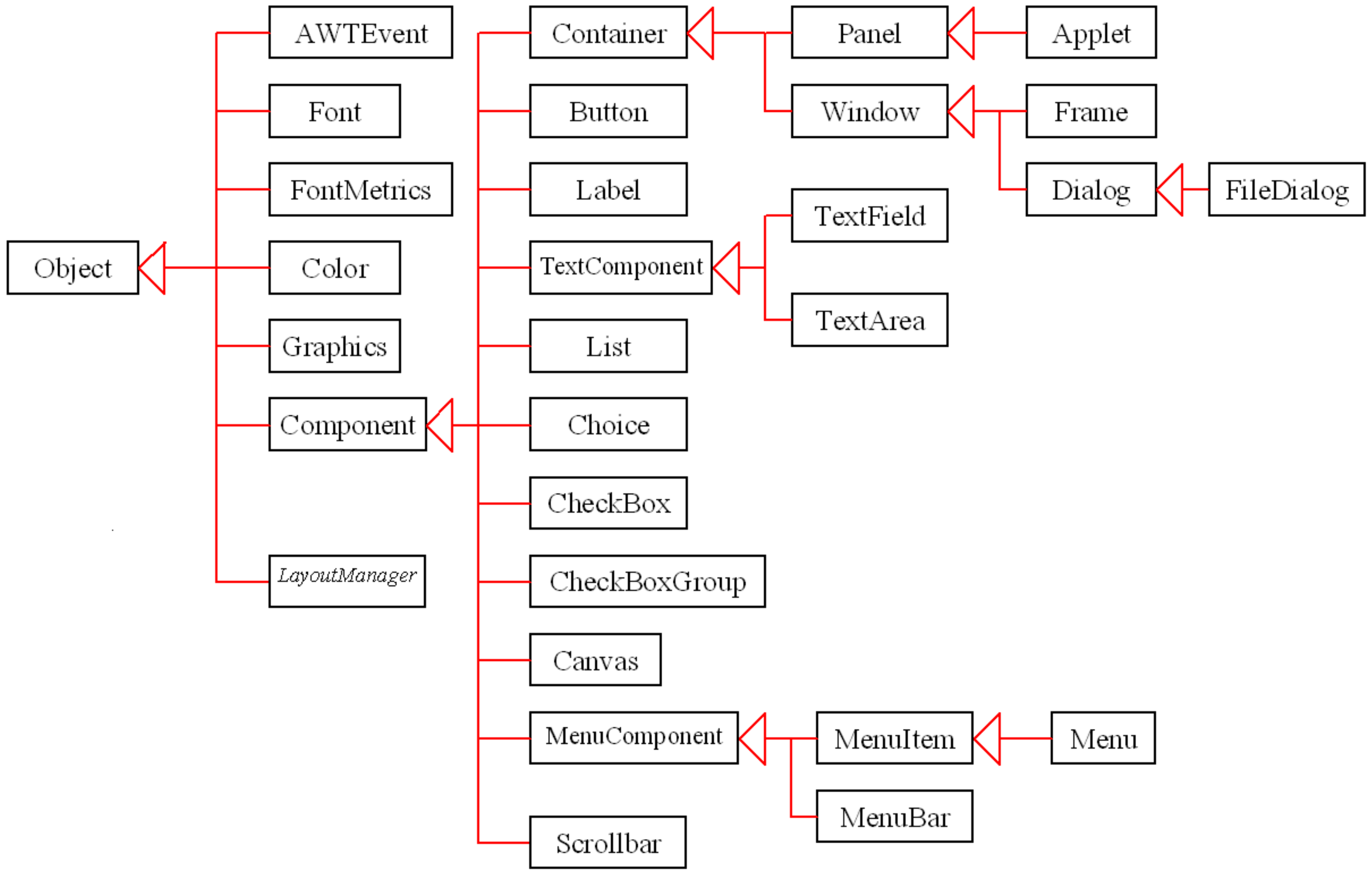
Composite Pattern - Consequences

- Defines Class hierarchies for recursive composition.
- Makes clients simple (can treat composite structures and individual objects uniformly)
- Makes it easy to add new components (no code needed for components or for clients)
- Can make your design overly general – Harder to restrict the components of a composite.

Example 1

four-door sedan

Example 2: AWT Class Hierarchy

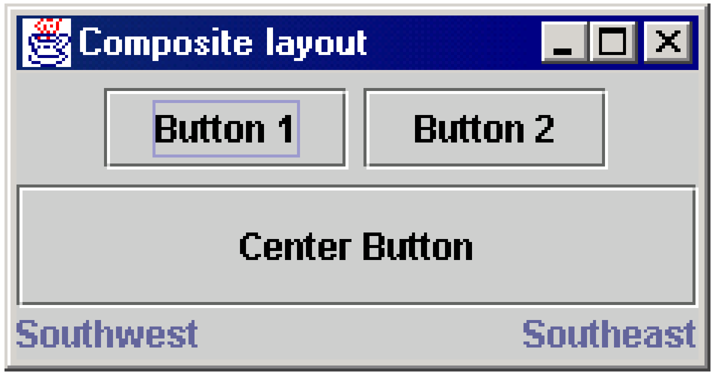


Composite example: layout

```
Container north = new JPanel(new FlowLayout());
north.add(new JButton("Button 1"));
north.add(new JButton("Button 2"));

Container south = new JPanel(new BorderLayout());
south.add(new JLabel("Southwest"), BorderLayout.WEST);
south.add(new JLabel("Southeast"), BorderLayout.EAST);

Container cp = getContentPane();
cp.add(north, BorderLayout.NORTH);
cp.add(new JButton("Center Button"),
   BorderLayout.CENTER);
cp.add(south, BorderLayout.SOUTH);
```



In a Utopian Object Oriented World!

- Every class must know how to paint itself (if it is a visual component)

- Every class must know how to load and save itself from file (xml or otherwise)

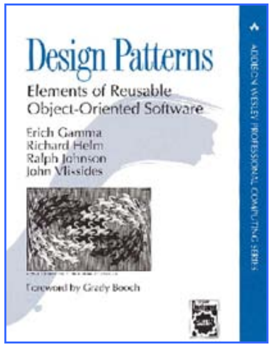
- Every class must know how to add/subtract itself

from other objects who are of the same type (i.e. operator support).

- Every class which has children (a composite) must provide the client with an iterator interface to access children

Class definition (for a given problem domain) must be as complete as possible

Design Pattern Space


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON WESLEY PROFESSIONAL COMPUTING SERIES

Creational	**Structural**	**Behavioral**
Factory Method	Adapter	Interpreter
Abstract Factory	Bridge	Template Method
Builder	Composite	Chain of Responsibility
Prototype	Decorator	Command
Singleton	Flyweight	Iterator
	Facade	Mediator
	Proxy	Memento
		Observer
		State
		Strategy
		Visitor