Design Patterns & Software Architecture
Composite

dr. Joost Schalken-Pinkster

University of Applied Science Utrecht

The Netherlands

The contents of these course slides are based on:
Chris Loftus, *Course on Design Patterns & Software Architecture for NEU*. Aberystwyth University, 2013.
Gemma et al.(1995). Design Patterns: Elements of Reusable Object-Oriented Software,Addison-Wesley,1995

Session overview

- Composite

Composite design pattern

Composite design pattern: Let’s start with an example

You have been asked to develop a simple graphical library. The initial requirements are:

1. The library must be able to handle graphical primitives such as lines and circles.
2. The library must allow for recursive composites of composites and graphical primitives so that a drawing can be made up of drawings and primitives.
3. Operations need to be provided that allow for the adding, removal, display and navigation of graphical components (composites and primitives).

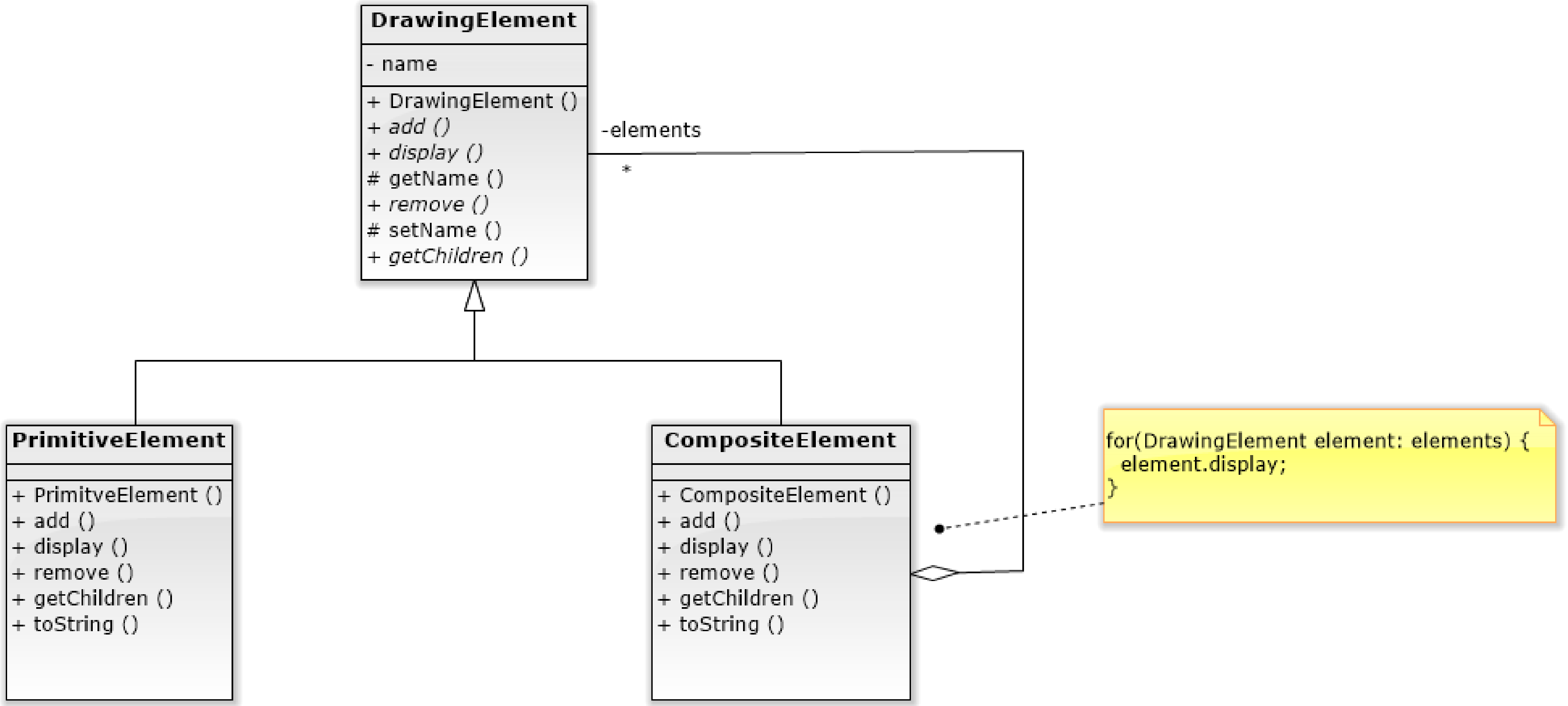
Let’s find a design pattern

Will now present, on the board,
and using Eclipse,
a solution that utilises
the composite design pattern…

Composite pattern definition

- **Intent**: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

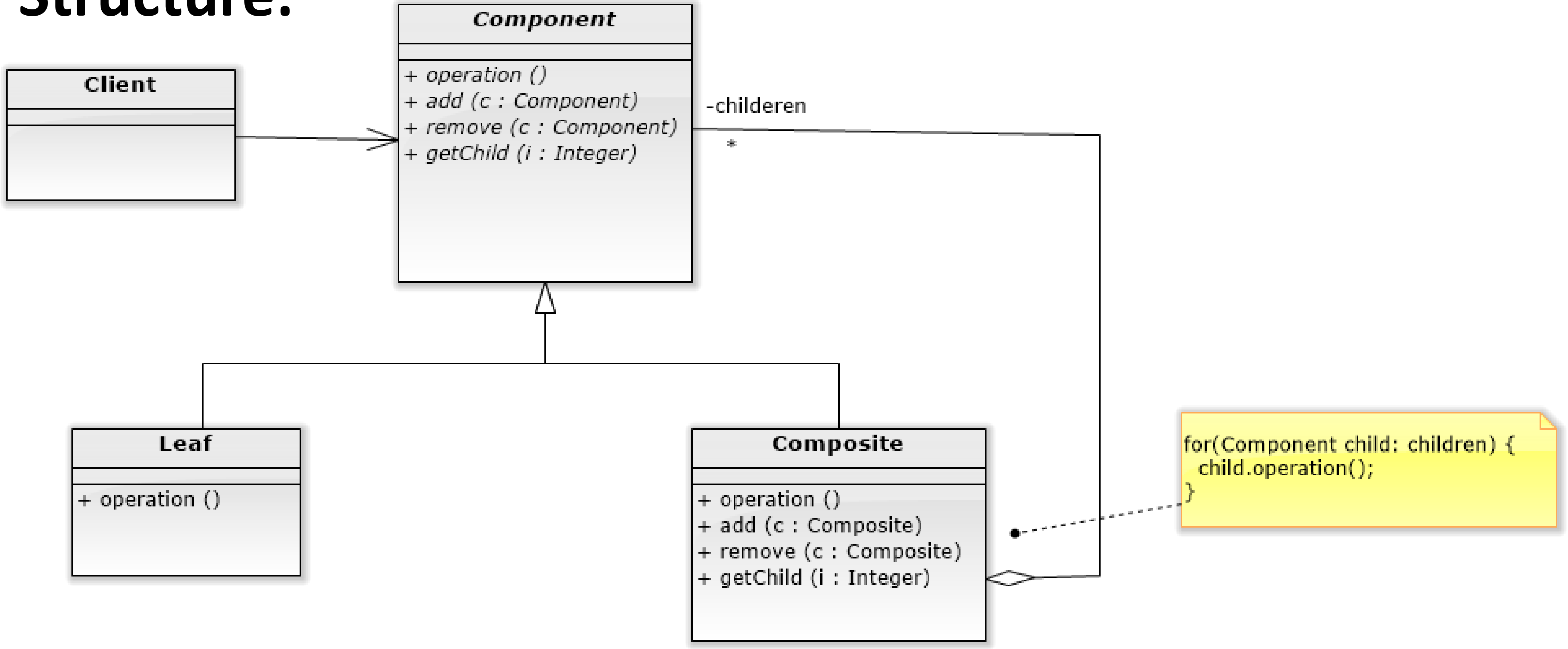
- **Motivation**: The graphics application…



```
for(DrawingElement element: elements) {
  element.display;
}
```

- **Applicability**: Use Composite when
  - You want to represent whole-part hierarchies of objects
  - You want clients to be able to ignore the difference between composite objects and individual objects.

Structure:



```
for(Component child: children) {
  child.operation();
}
```

- **Participants**:
  - **Component**
    - declares the interface for objects in the composition.
    - implements default behaviour for the interface common to all classes, as appropriate.
    - declares an interface for accessing and managing its child components
    - (optional) defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.
  - **Leaf**
    - represents leaf objects in the composition. A leaf has no children.
    - defines behaviour for primitive objects in the composition.
  - **Composite**
    - defines behaviour for components having children.
    - stores child components.
    - implements child-related operations in the Component interface.
  - **Client**
    - manipulates objects in the composition through the Component interface.

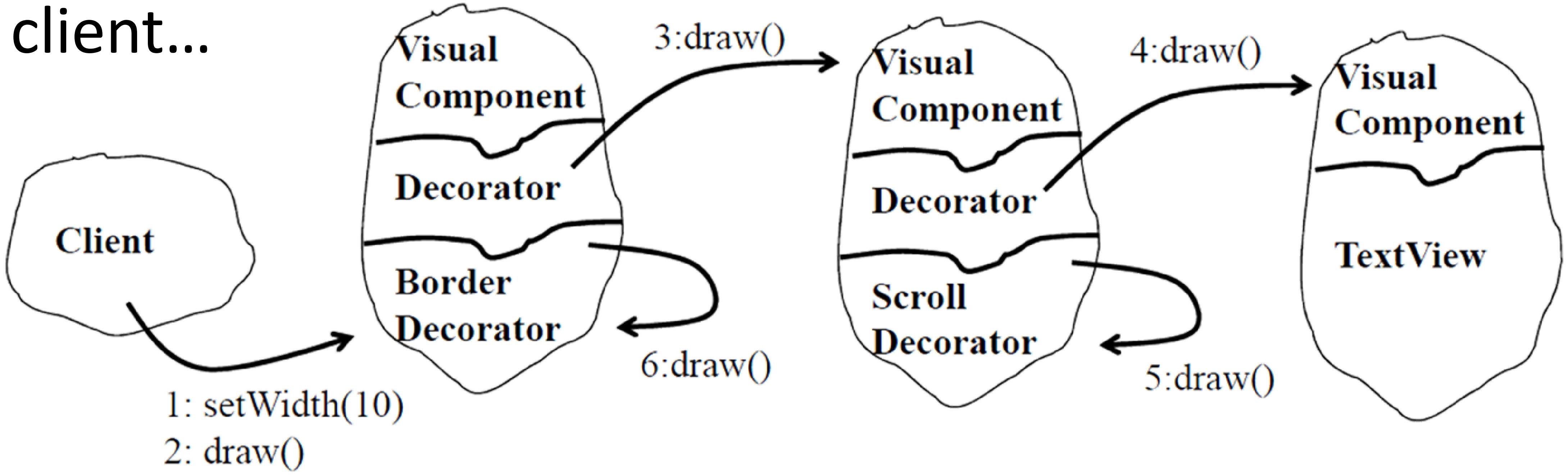
- **Consequences**: The composite pattern
  - Defines class hierarchies of primitive and composite objects…
  - Makes the client simple…
  - Makes it easy to add new kinds of components…
  - Can make your design too general…

Implementation:

- Explicit parent reference…
- Sharing components…
- Maximizing the Component (e.g. DrawingElement) interface…
- Should Component (e.g. DrawingElement) implement a list of Components?...
- Child ordering…
- Caching to improve performance…
- Consider data structure efficiency issues…
- Who should delete components?...

How does Composite differ from Decorator?

- Decorator is composite with only one component at each level. Decorator does not do object aggregation.
- Also Decorator tends to expose new behavior to client…



Reading

For tomorrow please read:

- Chapter 9 (Well managed collections) of Head First Design Patterns.
- Chapter 13 (Patterns in the Real World) of Head First Design Patterns
- Chapter Chain of Responsibility of Design Patterns: Elements of Reusable Object-Oriented Software, pp 251-262.