Dr. Michael Eichberg
Software Engineering
Department of Computer Science
Technische Universität Darmstadt

Software Engineering

The Composite Design Pattern

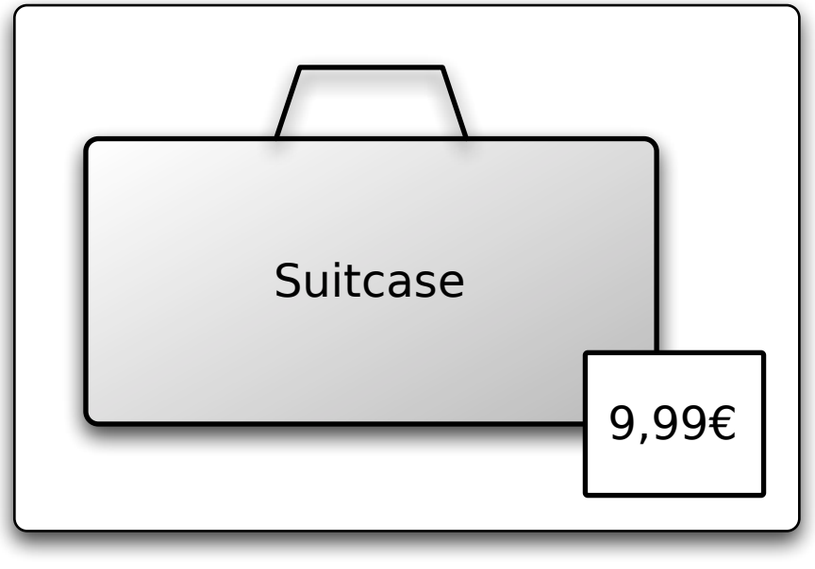
For details see Gamma et al. in “Design Patterns”

TECHNISCHE UNIVERSITÄT DARMSTADT

The Composite Design Pattern

Motivation

- Imagine a drawing editor where complex diagrams are build out of simple components and where the user wants to treat classes uniformly most of the time whether they represent primitives or components
- Example
  - Picture contains elements
  - Elements can be grouped
  - Groups can contain other groups



The Composite Design Pattern

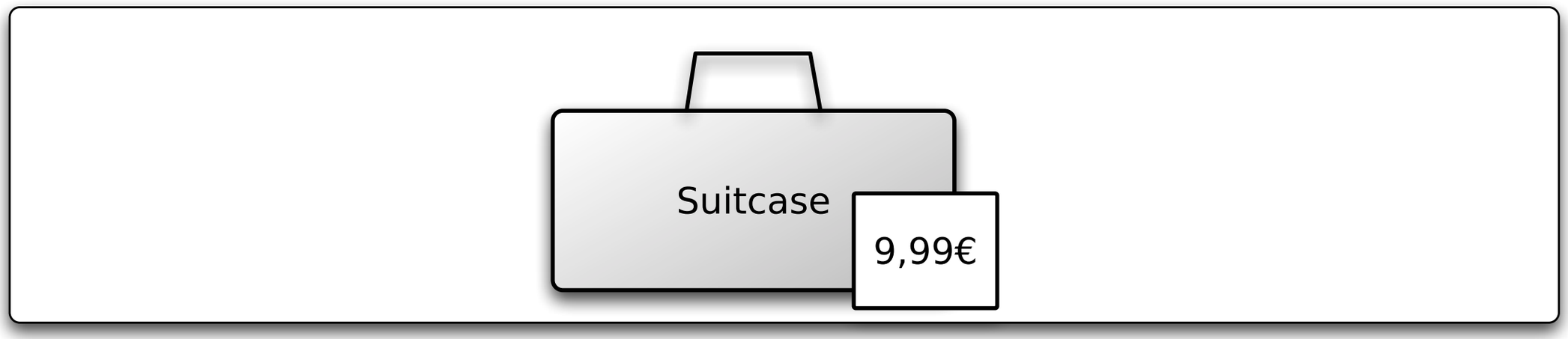
Intent

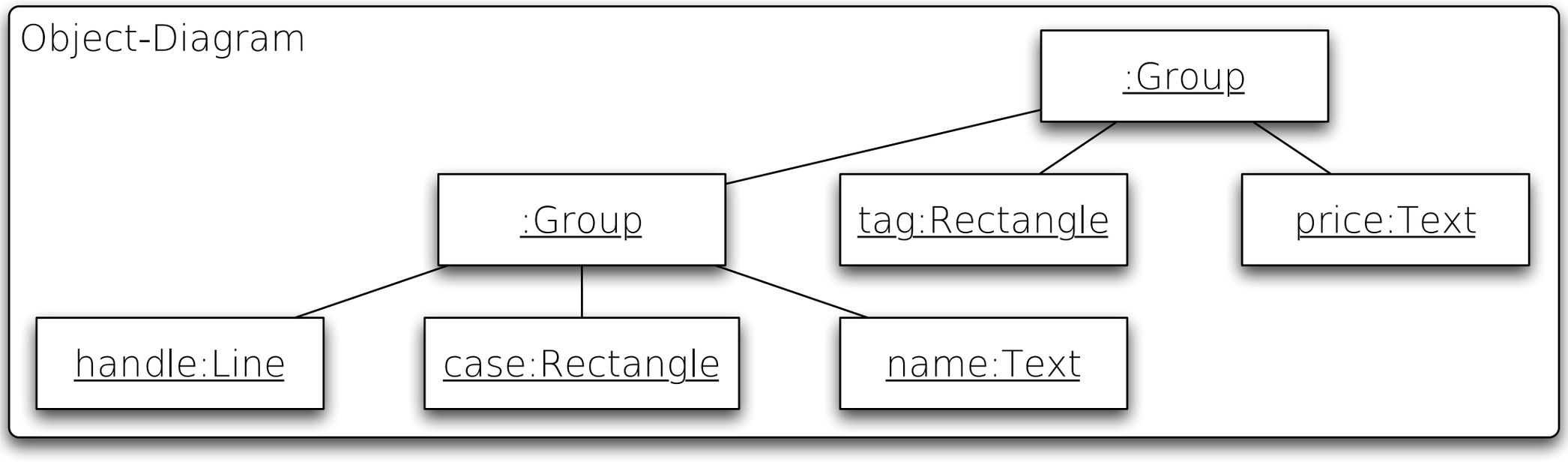
- Compose objects into tree structures to represent part-whole hierarchies
- The composite design pattern lets clients treat individual objects and compositions of objects uniformly

The Composite Design Pattern

Example

Drawing

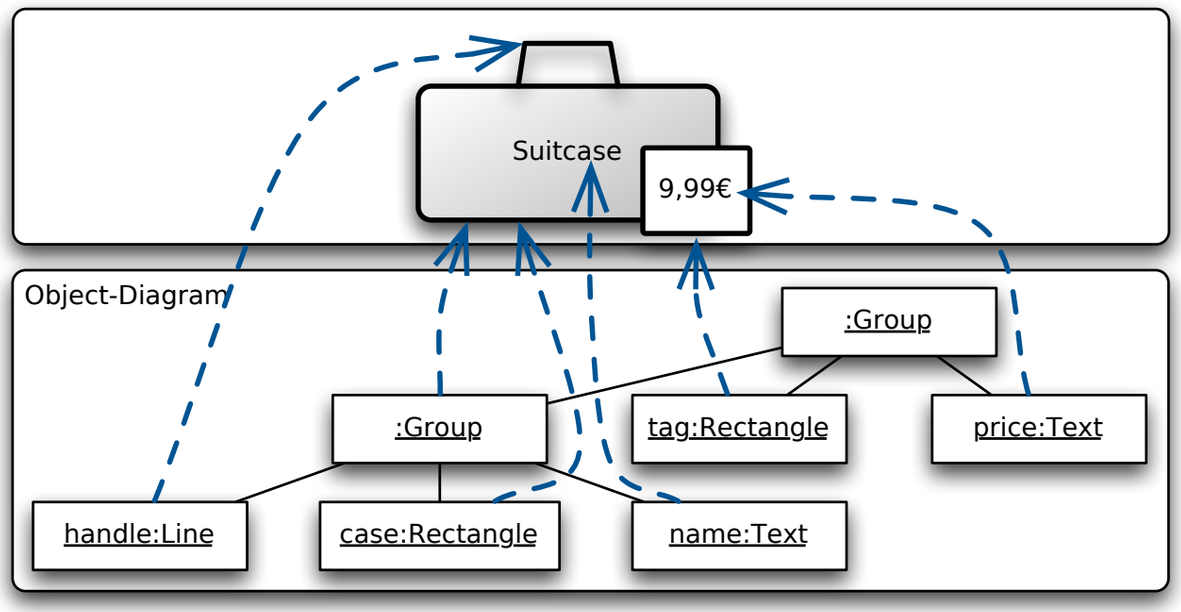


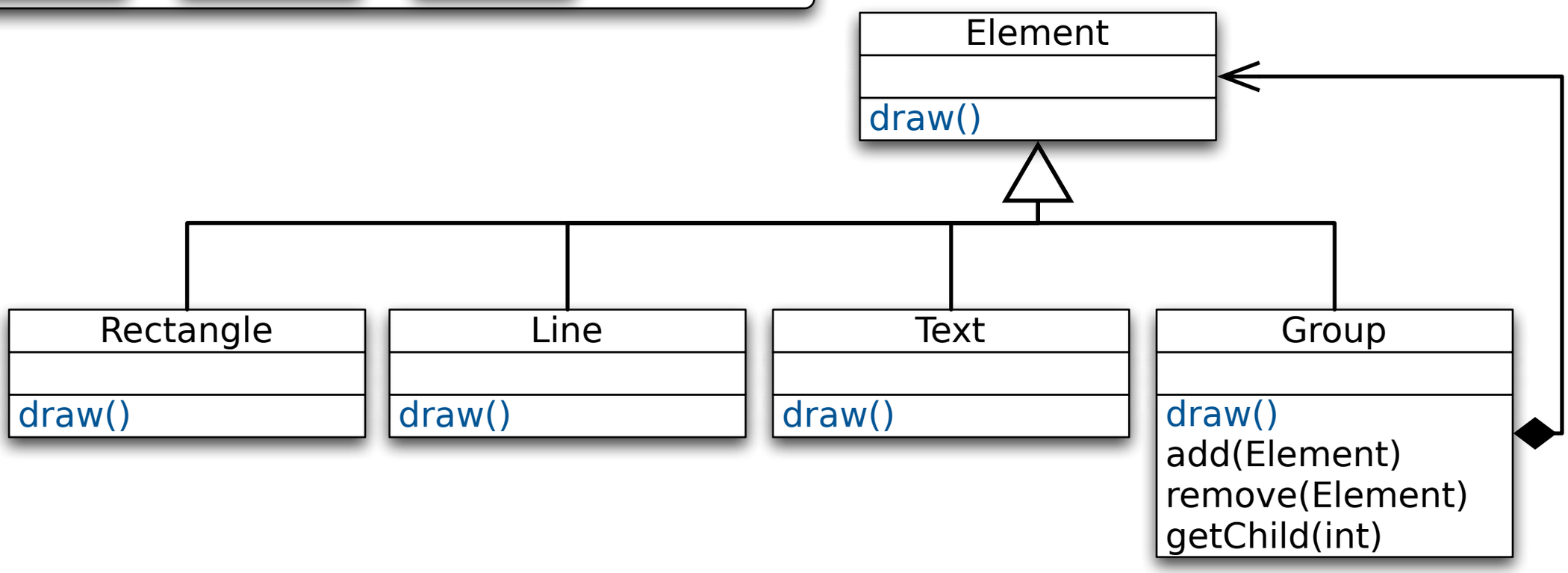


Corresponding Object Diagram

The Composite Design Pattern

Example





The Composite Design Pattern

Applicability

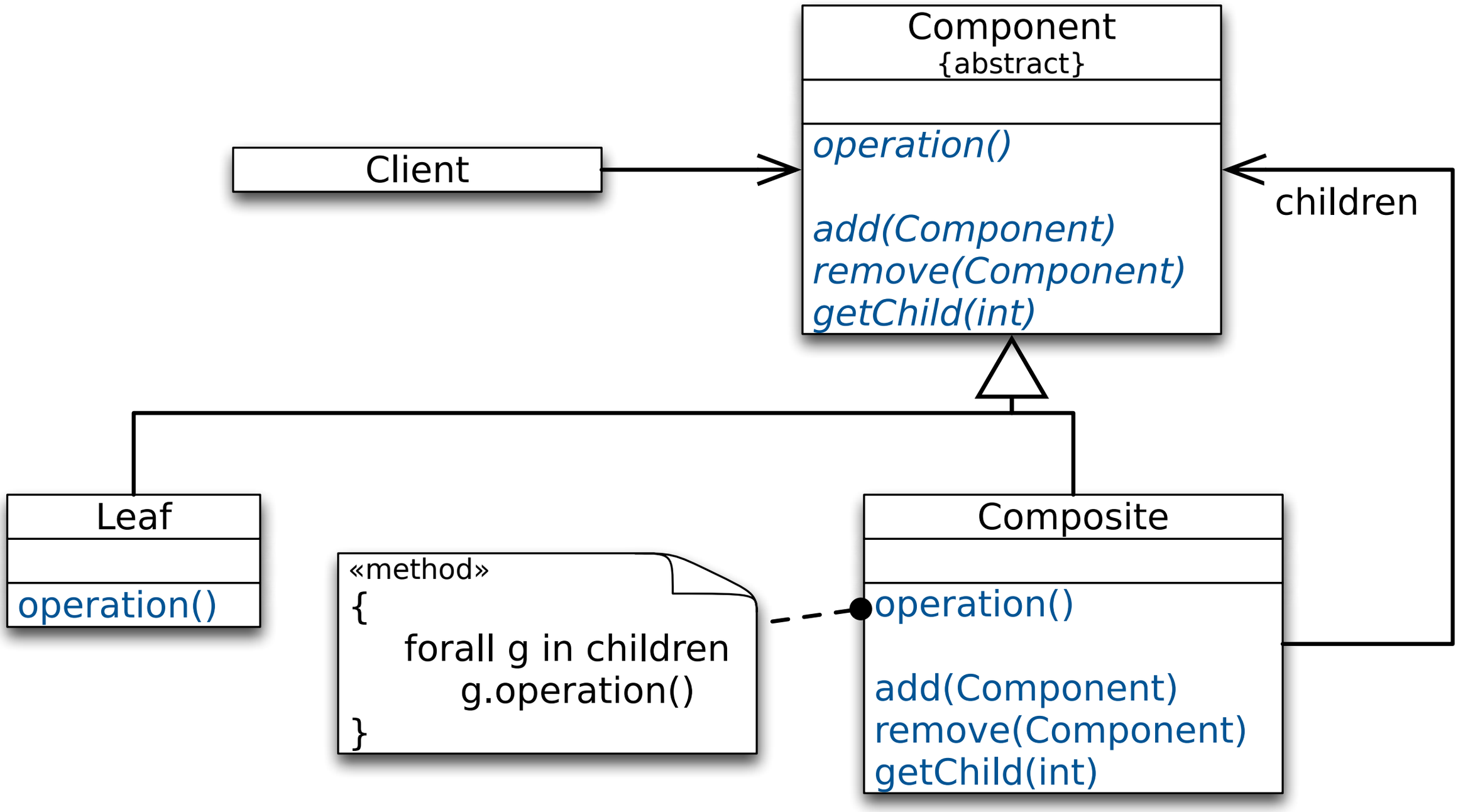
Use composite when...

- you want to represent part-whole hierarchies of objects
- you want clients to be able to ignore the difference between individual and composed objects

(Clients will treat all objects in the composite structure uniformly.)

The Composite Design Pattern

Structure



The Composite Design Pattern

Participants

- `Component`
  - Declares the interface for objects in the composition
  - Implements the default behavior as appropriate
  - (Often) declares an interface for accessing and managing child components
- `Leaf`
  Represents leaf objects in the composition; defines the primitive behavior
- `Composite`
  Stores children / composite behavior
- `Client`
  Accesses objects in the composition via Component interface

The Composite Design Pattern

Collaborations

- Clients interact with objects through the Component interface
- Leaf recipients react directly
- Composites forward requests to their children, possibly adding before/after operations

Excursion: A pattern is a collaboration
Object diagram for the context. Which roles are involved?
Sequence diagram for interactions (Interaction diagram for context & interaction.) What is the order of method calls?

The Composite Design Pattern

Consequences

- Primitive objects can be recursively composed ✓
- Clients can treat composites and primitives uniformly ✓
  (Clients do not have to write tag-and-case statement-style functions.)
- New components can easily be added ✓
- Design may become overly general ✗
  (You can't always rely on the type system to enforce certain constraints; e.g. that a composite has only certain components.)

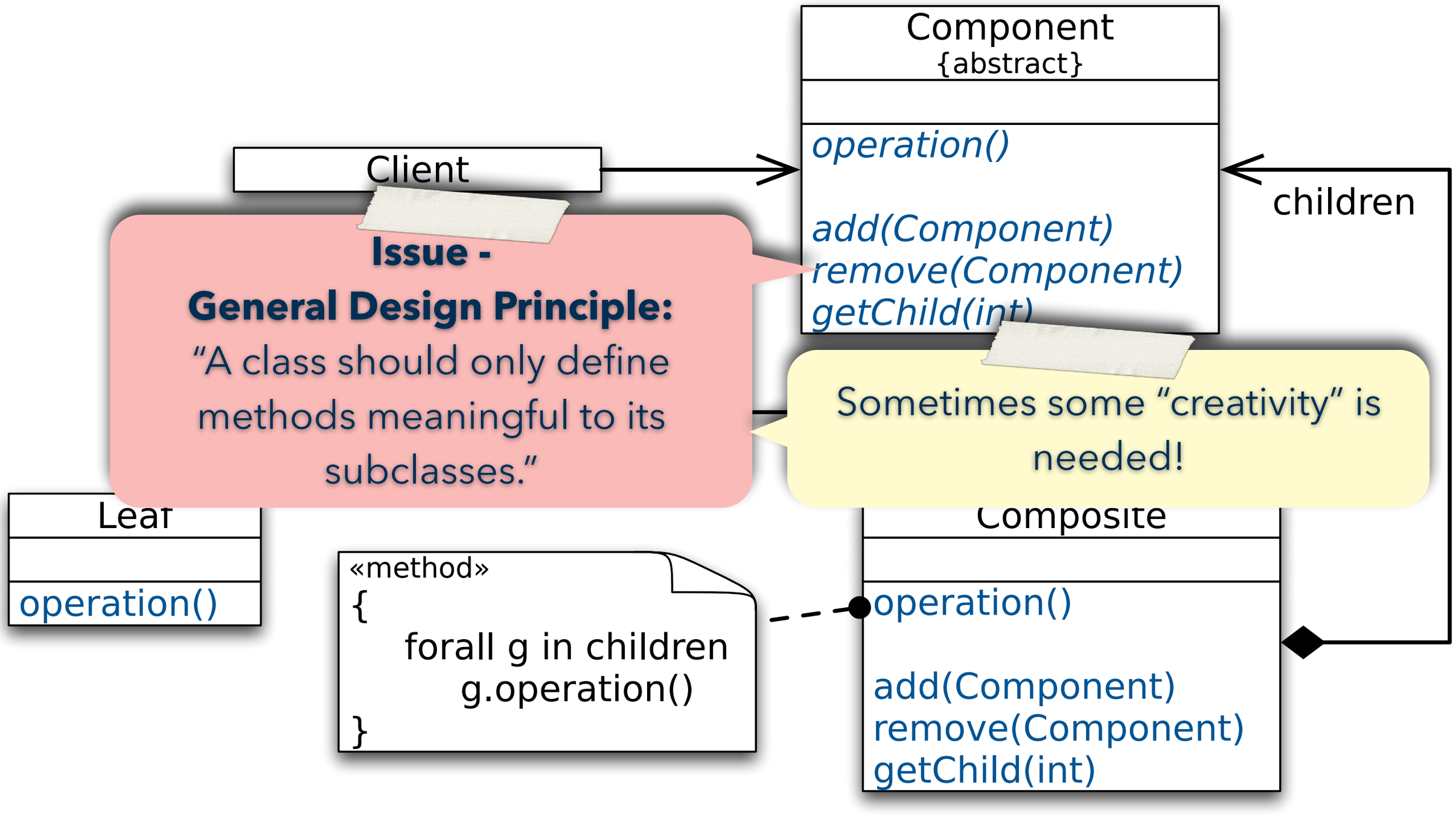
The Composite Design Pattern

Implementation

- **Explicit parent references**
  May facilitate traversal and management of a composite structure; often defined in the component class. Need to be maintained.
- **Sharing components**
  E.g. to reduce storage requirements it is often useful to share components. (→Flyweight Pattern)
- **Size of the component interface**
  To make clients unaware of the specific Leaf or Composite classes the Component class should define as many operations for Composite and Leaf as possible.
  (May require a little “creativity”…)

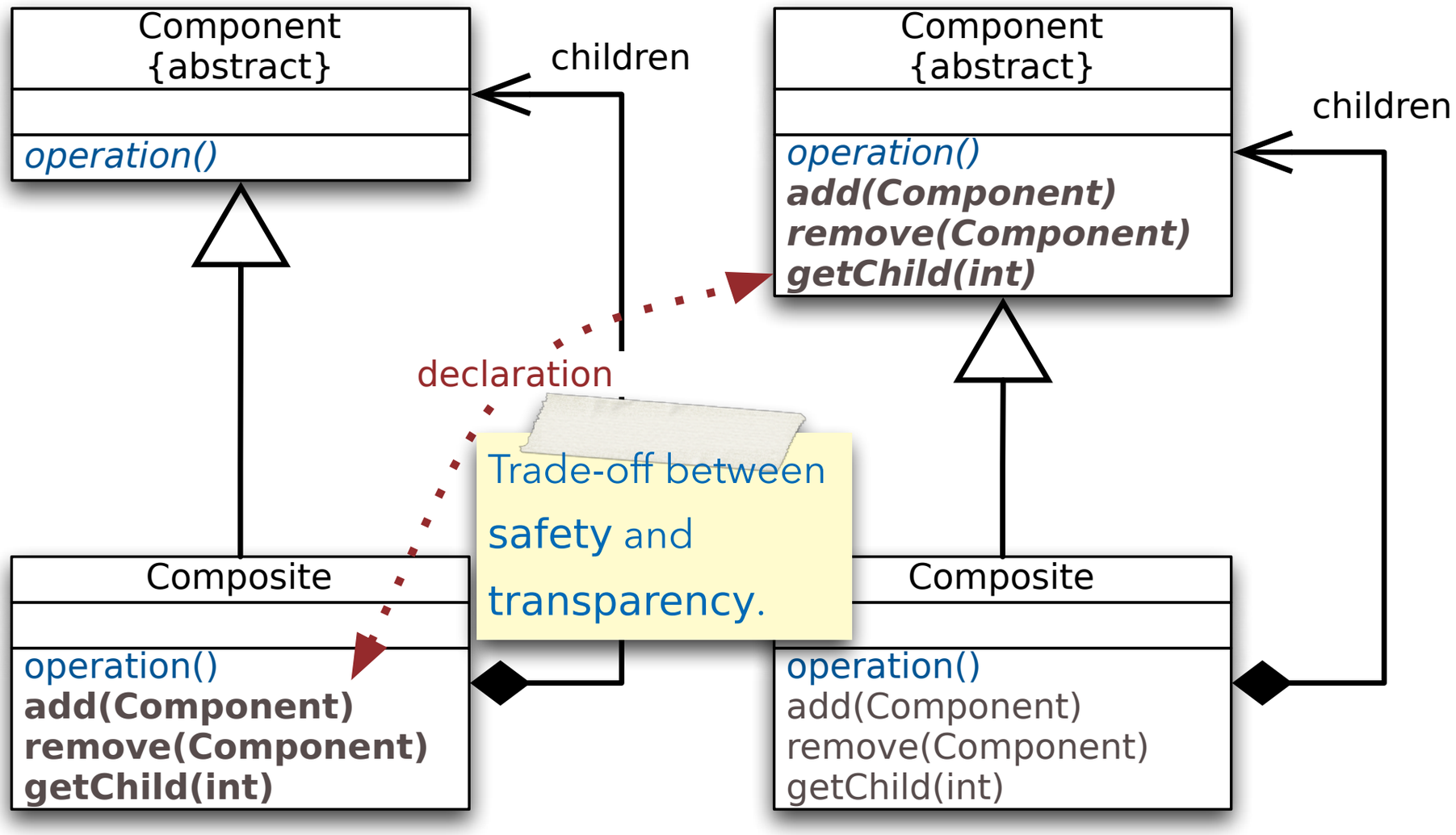
The Composite Design Pattern

Structure



The Composite Design Pattern - Implementation

- **Placing child management operations - who declares them?**
  - at the root (Component) is convenient, but less safe because clients may try to do meaningless things
  - in Composite is safe



The Composite Design Pattern
Example - Component Class

- Computer equipment contains:
  - drives,
  - graphic cards in the PCIe slots,
  - memory,
  - and more.
- Such a part-whole structure can be modeled naturally with the Composite pattern.

The Composite Design Pattern

Example - Component Class


```
public abstract class Equipment {
   private String name;
   public String name() { return name; }

   public abstract int price();
   // more methods, e.g., for power consumption etc.

   // Child management
   public abstract void add(Equipment eq);
   public abstract void remove(Equipment eq);
   public Iterator<Equipment> iterator(){
      return NULL_ITERATOR;
   };
}
```

The Composite Design Pattern

Example - Leaf Class


```
public class HardDisk extends Equipment {
   public int price() {
      return 50;
   }

   ...
}
```

The Composite Design Pattern

Example - Composite Class


```
public class CompositeEquipment extends Equipment {

   ...

   public int price() {
      int total = 0;
      for (int i=0; i < equipment.length; i++)
         total += equipment[i].price();
      return total;
   }

   public void add(Equipment eq) {...};
   public void remove(Equipment eq) {...};

   public Iterator<Equipment> iterator() {...};
}
```

The Composite Design Pattern

Example - Demo Usage


```
public class Chassis extends CompositeEquipment{…}
public class Bus extends CompositeEquipment{…}
public class Card extends Equipment{…}
public class Mainboard extends CompositeEquipment{…}
```

} Further Definitions

DEMOCODE

```
Chassis chassis = new Chassis();
Mainboard mainboard = new Mainboard(“Hypermulticore”);
Bus bus = new Bus("PCIe Bus");

chassis.add(new HardDisk("Personal 2Tb Drive"));
chassis.add(new HardDisk(“512GB PCIe – SSD”));
chasses.add(mainboard);
mainboard.add(bus);
bus.add(new Card("Graphics Card"));
bus.add(new HardDisk("YetAnotherDisk")); // checks required...?
System.out.println("Total price: " + chassis.price() );
```

The Composite Design Pattern

Known Uses

- View class of Model/View/Controller
- Application frameworks & toolkits
  - ET++, 1988
  - Graphics, 1988
  - Glyphs, 1990
  - InterViews, 1992
  - Java (AWT, Swing, SWT, JavaFX, Files, …)

The Composite Design Pattern

Related Patterns

- **Iterator**
  Traverse composite
- **Visitor**
  To localize operations that are otherwise distributed across Composite and Leaf classes
- **Chain of Responsibility**
  Use components hierarchy for task solving
- **Flyweight**
  For sharing components

Will be discussed later (as part of more advanced lectures.)

The Composite Design Pattern

Summary

The Composite Design Pattern facilitates to compose objects into tree structures to represent part-whole hierarchies.

Apply the composite pattern if clients can treat individual objects and compositions of objects uniformly.