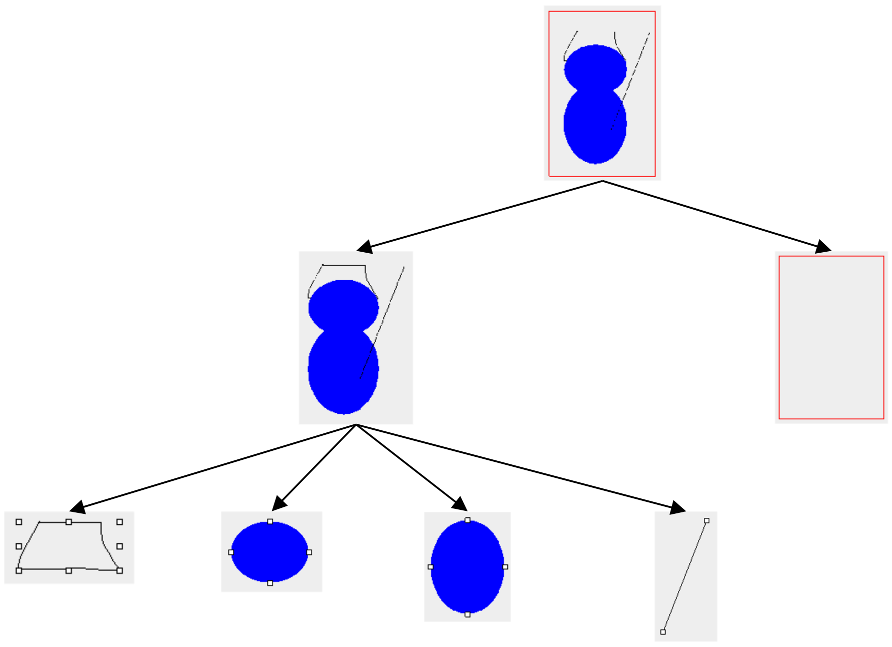
Composite Pattern

Goal:

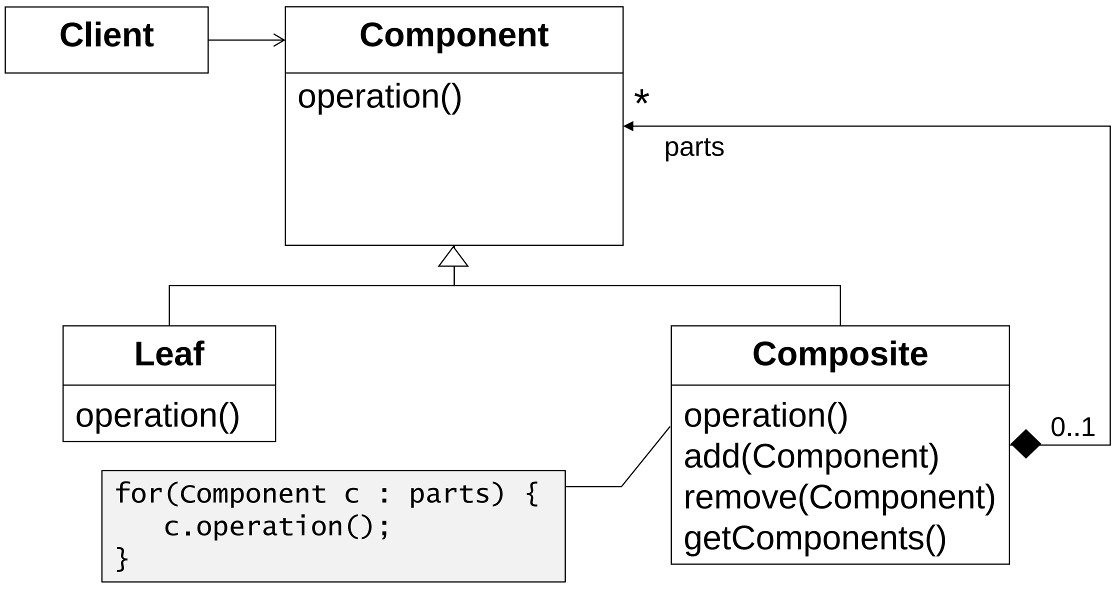
Representation of recursive part-whole hierarchies (part-of relations).

Motivation:



Goal: Uniform access to operations which are defined both for composites and for single objects.

Structure:

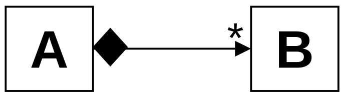


```
for(Component c : parts) {
   c.operation();
}
```

Remarks:

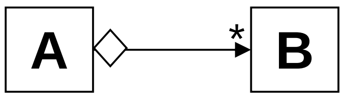
Composition vs. Aggregation

Composition
B is a fixed part of A



- Part B may be part of at most one whole A.
- If the whole A is deleted, then also all its parts.
- The whole A acts substitutional for its parts, i.e. the operations defined on the whole A are propagated to its parts.
- Usage: for concrete things (Car, Airplane, etc.)
  → Data structure: Tree

Aggregation
B is a variable part of A



- Part B may be part of several wholes A.
- Part B may exist individually.
- Usage: for abstract things (e.g. units of organizations)
  → Data structure: Directed acyclic graph (DAG)

Child Management

The methods for the management of the parts of a composite (e.g. `addChild`, `removeChild`, `getChildAt`, etc.) can be defined either in the composite interface or in the component interface.

Composite: This approach favors type safety and clarity
+ Clear separation between component and composite, i.e. the composite specific methods are defined where they belong to.
+ Simple and easy to understand interfaces.
- Clients must distinguish between component and composite (and must apply type tests).

Example:
- AWT-Components: both `java.awt.Button` and `java.awt.Container` are derived from base class `java.awt.Component` which does not define any composite-specific methods
- FX Scene-Graphs: `javafx.scene.shape.Shape` and `javafx.scene.Parent` (this is the composite class) are both derived from the base class `javajavafx.scene.Node`.

Component: This approach favors transparency and uniformity
+ Uniform interface, there is no need to distinguish between components and composites.
+ Simplified handling, i.e. no need to use type tests (`instanceof`) and/or type casts
- Leads to more complex interfaces for all components which are less intuitive
- Composite specific method must be provided by all components (in conformance to their specification)

Example:
- Swing-Components: class `javax.swing.JButton` is an extension of class `JComponent` which defines composite specific methods and may contain other components
- FX Scene-Graphs: `javafx.scene.control.Control` is an extension of the container class `javafx.scene.Parent.`