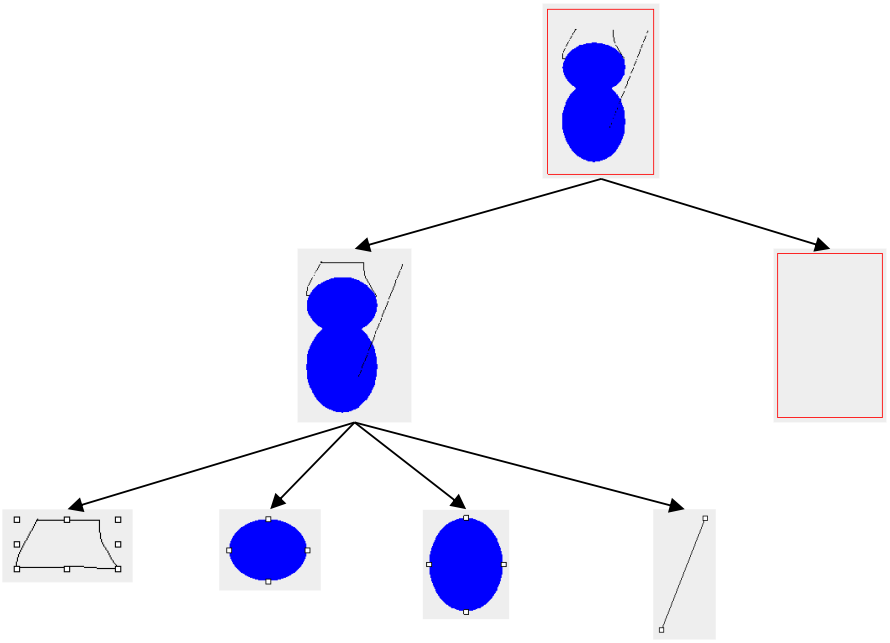
Composite Pattern

Ziel:

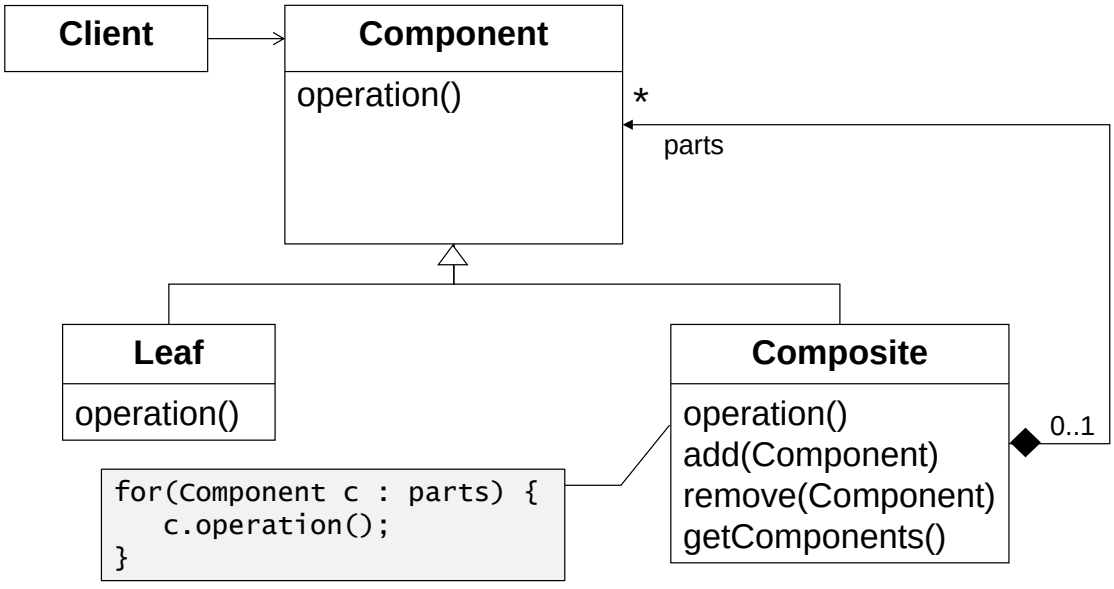
Repräsentation von rekursiven Teile-Ganzes-Beziehungen (part-of).

Motivation:



Ziel: Auf Operationen, die für zusammengesetzte und atomare Objekte gelten, soll gleichartig zugegriffen werden können (uniformity).

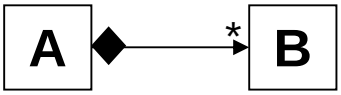
Struktur:



Bemerkungen:

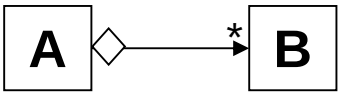
Komposition versus Aggregation

Komposition
B ist fester Bestandteil von A



- Teil B darf Teil höchstens eines Ganzen A sein
- Wird Ganzes A gelöscht, so auch all seine Teile B
- Ganzes A handelt stellvertretend für seine Teile, Operationen werden an Teile propagiert
- Für konkrete Dinge (physikalisch, z.B. Auto, Hubschrauber)
  → Datentyp: Baum

Aggregation
B ist variabler Bestandteil von A



- Teil B kann zu mehreren Ganzen A gehören
- Teil B kann unabhängig existieren
- Für abstrakte Dinge (z.B. Organisationseinheiten)
  → Datentyp: Gerichteter, azyklischer Graph (DAG)

Child Management

Die Methoden für die Verwaltung der Teile (z.B. `addChild`, `removeChild`, `getChildAt`, etc) können in folgenden Klassen definiert werden:

Composite: Diese Variante favorisiert **Typsicherheit und Klarheit**
+ Klare Aufgabentrennung zwischen Component und Composite, denn Methoden sind nur dort definiert, wo sie nötig sind
+ Einfach verständliche Schnittstellen
- Benutzer muss unterscheiden zwischen Component und Composite

Beispiel:
- AWT-Komponenten: `java.awt.Button` und `java.awt.Container` sind beide von `java.awt.Component` abgeleitet, welches keine Composite-spezifischen Methoden enthält.
- FX Scene-Graphs: `javafx.scene.shape.Shape` und `javafx.scene.Parent` (das ist die Container-Klasse) sind beide von `javafx.scene.Node` abgeleitet.

Component: Diese Variante favorisiert **Transparenz und Einheitlichkeit**
+ Einheitliche Schnittstelle, es muss nicht unterschieden werden zwischen Components und Composites
+ Vereinfacht Umgang, da weniger Typtests (`instanceof`) und/oder Type-Casts nötig sind
- Führt zu komplexeren Schnittstellen für alle Components
- Manchmal wenig intuitive Schnittstellen

Beispiel:
- Swing-Komponenten: `javax.swing.JButton` ist von JComponent abgeleitet, welches Composite-spezifische Methoden enthält und daher wiederum andere Komponenten enthalten kann.
- FX Scene-Graphs: `javafx.scene.control.Control` ist von `javafx.scene.Parent` (der Container-Klasse) abgeleitet.