WIKIPEDIA

Kompositum (Entwurfsmuster)

Das **Kompositum** (englisch *composite* oder *whole-part*) ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung, das zur Kategorie der Strukturmuster (englisch *structural patterns*) gehört. Es ist ein so genanntes GoF-Entwurfsmuster. Das Kompositionsmuster (*composite pattern*) wird angewendet, um Teil-Ganzes-Hierarchien zu repräsentieren, indem Objekte zu Baumstrukturen zusammengefügt werden.[1] Die Grundidee des Kompositionsmusters ist, in einer abstrakten Klasse sowohl primitive Objekte als auch ihre Behälter zu repräsentieren. Somit können sowohl einzelne Objekte als auch ihre Kompositionen einheitlich behandelt werden.

Inhaltsverzeichnis

Verwendung

UML-Diagramme
- Klassendiagramm
- Objektdiagramm

Bestandteile

Vorteile

Nachteile

Beispiel
- C++
- Java

Verwendung in der Analyse

Verwandte Entwurfsmuster

Weblinks

Einzelnachweise

Verwendung

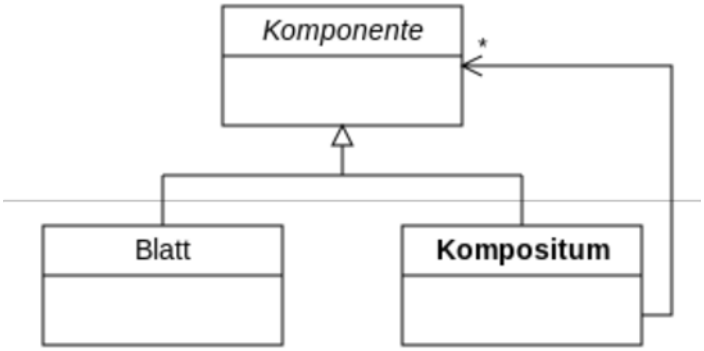
- Implementierung von Teil-Ganzes-Hierarchien.
- Verbergen der Unterschiede zwischen einzelnen und zusammengesetzten Objekten.

Ein typisches Beispiel für ein Kompositum sind hierarchische Dateisysteme, insbesondere ihre Repräsentation innerhalb von Dateimanagern oder Datei-Browsern als Verzeichnisse und Dateien.

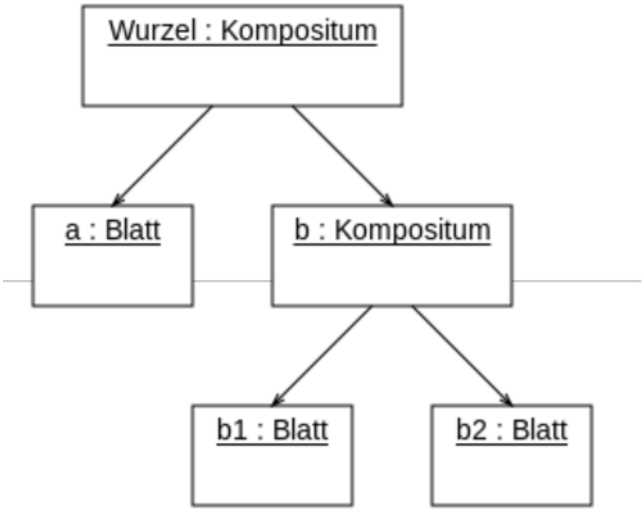
Ein anderes Beispiel sind die Klassendefinitionen der grafischen Benutzeroberfläche von Java. Alle Elemente wie Schaltflächen und Textfelder sind Spezialisierungen der Klasse `Component`. Die Behälter für diese Elemente sind aber ebenfalls Spezialisierungen derselben Klasse. Mit anderen Worten: Alle Standardelemente werden wesentlich durch eine einzige (Kompositum-)Klasse definiert.

UML-Diagramme

Klassendiagramm



Objektdiagramm



Bestandteile

Die `Komponente` definiert als Basisklasse das gemeinsame Verhalten aller Teilnehmer. Sie ist im Allgemeinen abstrakt und zum Beispiel ein Verzeichniseintrag.

Das `Blatt` repräsentiert ein Einzelobjekt, es besitzt keine Kindobjekte und ist zum Beispiel in einem Dateiverzeichnis eine Datei.

Das `Kompositum` enthält Komponenten, also weitere Komposita oder auch Blätter, als Kindobjekte und repräsentiert zum Beispiel ein Verzeichnis.

Vorteile

- einheitliche Behandlung von Primitiven und Kompositionen
- leichte Erweiterbarkeit um neue Blatt- oder Container-Klassen

Nachteile

Ein zu allgemeiner Entwurf erschwert es, Kompositionen auf bestimmte Klassen (und damit zumeist Typen) zu beschränken. Das Typsystem der Programmiersprache bietet dann keine Hilfe mehr, so dass Typüberprüfungen zur Laufzeit nötig werden.

Beispiel

Javas AWT-Klassen sind nach dem Kompositum-Muster gebaut. Da alle von Container erben, können sie jeweils selbst wieder Elemente aufnehmen.

Das folgende Beispiel besteht aus einer Grafik-Klasse; eine Grafik kann eine Ellipse oder eine Komposition von vielen Grafiken sein. Jede Grafik implementiert eine Methode zum Ausdrucken.

Es könnten noch weitere Figuren (Rechteck etc.) oder weitere Methoden (etwa „Rotiere“) implementiert werden.

C++

```
class Grafik {
public:
    virtual void print() const = 0;
    virtual ~Grafik() {} // Abstrakte Klassen, von denen geerbt wird, sollten immer einen virtuellen
Destruktor haben
};

class GrafikKompositum : public Grafik {
    std::set<Grafik const*> children;
    typedef std::set<Grafik const*>::const_iterator grIter;
public:
    void print() const {
        for (grIter it = children.begin(); it != children.end(); it++) (*it)->print();
    }

    void add(Grafik const* component) {
        children.insert(component);
    }

    void remove(Grafik const* component) {
        children.erase(component);
    }
};

class Ellipse: public Grafik {
public:
    void print() const {
        std::cout << "Ellipse" << std::endl;
    }
};

int main() {
    Ellipse ellipse1, ellipse2, ellipse3, ellipse4;

    GrafikKompositum grafik1, grafik2, grafikGesamt;

    grafik1.add(&ellipse1);
    grafik1.add(&ellipse2);
    grafik1.add(&ellipse3);
    grafik2.add(&ellipse4);

    grafikGesamt.add(&grafik1);
    grafikGesamt.add(&grafik2);

    grafikGesamt.print();
}
```

Java

```
/** "Komponente" */
interface Graphic {

    //Prints the graphic.
    public void print();
}

/** "Komposition" */
class CompositeGraphic implements Graphic {
```

```
    //Collection of child graphics.
    private List<Graphic> childGraphics = new ArrayList<Graphic>();

    //Prints the graphic.
    public void print() {
        for (Graphic graphic : childGraphics) {
            graphic.print();
        }
    }

    //Adds the graphic to the composition.
    public void add(Graphic graphic) {
        childGraphics.add(graphic);
    }

    //Removes the graphic from the composition.
    public void remove(Graphic graphic) {
        childGraphics.remove(graphic);
    }
}

/** "Leaf" */
class Ellipse implements Graphic {

    //Prints the graphic.
    public void print() {
        System.out.println("Ellipse");
    }
}

/** Client */
public class Program {

    public static void main(String[] args) {
        //Initialize four ellipses
        Ellipse ellipse1 = new Ellipse();
        Ellipse ellipse2 = new Ellipse();
        Ellipse ellipse3 = new Ellipse();
        Ellipse ellipse4 = new Ellipse();

        //Initialize three composite graphics
        CompositeGraphic graphic = new CompositeGraphic();
        CompositeGraphic graphic1 = new CompositeGraphic();
        CompositeGraphic graphic2 = new CompositeGraphic();

        //Composes the graphics
        graphic1.add(ellipse1);
        graphic1.add(ellipse2);
        graphic1.add(ellipse3);

        graphic2.add(ellipse4);

        graphic.add(graphic1);
        graphic.add(graphic2);

        //Prints the complete graphic (four times the string "Ellipse").
        graphic.print();
    }
}
```

Verwendung in der Analyse

Ein Kompositum ist auch als reines Daten-Muster interessant, ohne dass Operationen in den Klassen definiert werden, da es zur Repräsentation allgemeiner Baumstrukturen verwendet werden kann. Daher ist dieses Muster auch in der Analyse sinnvoll einsetzbar, z. B. zur Darstellung verschachtelter Aufträge oder Auftragnehmer (mit Unteraufträgen/Unterauftragnehmern), verschachtelter Abläufe, hierarchischer Gruppen von Dingen (Benutzergruppen, E-Mail-Listen, Artikelgruppen, organisatorische Verbünde) usw. Es muss aber darauf geachtet werden, ob solche Hierarchien tatsächlich gleichförmig sind, oder ob die inneren Ebenen verschiedene fachliche Bedeutung haben. Letzteres drückt sich z. B. darin aus, dass Begriffe wie „Gruppe“ und „Untergruppe“ fachlich unterschieden werden.

Verwandte Entwurfsmuster

- Visitor
- Decorator
- Ein Design auf Basis des Kommando-Musters kann oft sinnvollerweise auch zusammengesetzte Kommandos enthalten, die nach dem Kompositum-Muster aufgebaut sind.

Weblinks

- Einsteigerfreundliche Einführung in das Composite Design Pattern (http://www.philipphauer.de/study/se/design-pattern/composite.php)

Einzelnachweise

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Entwurfsmuster*. 5. Auflage. Addison-Wesley, 1996, ISBN 3-8273-1862-9, S. 239.