

5 Entwurfsmuster (Design Patterns)

Entwurfsmuster

Generalisierte Lösungsideen zu immer wiederkehrenden Entwurfsproblemen werden Entwurfsmuster genannt. Sie sind keine fertig codierten Lösungen, sondern beschreiben den Lösungsansatz

- ✍ Grundidee geht auf einen Architekten (*Christopher Alexander*) zurück
- ✍ C. Alexander definierte Muster als eine dreigeteilte Relation aus
 - der Beziehung zu einem gewissen Kontext
 - der Problemstellung
 - der Lösung
- ✍ C. Alexander stellte Muster für den Anwendungsbereich der Architektur zusammen
- ✍ Später wurde dieser Ansatz auf andere Gebiete erweitert
- ✍ Für den Bereich der Software-Spezifikation waren aktiv
 - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Entwurfsmuster, Addison-Wesley, 1996)

5.1 Was ist ein Entwurfsmuster?

Ein Entwurfsmuster

- ✍ beschreibt abstrakt eine bewährte Lösung für ein bestimmtes und häufig wiederkehrendes Problem des objektorientierten Software-entwurfs
- ✍ entsteht durch die Analyse und Überarbeitung vorhandener Designlösungen, setzt also Entwurfserfahrung voraus (Entwurfsmuster werden nicht “erfunden”, sondern entdeckt !)
- ✍ kann in seiner Struktur verstanden werden als eine Menge von Klassen, die festgelegte Verantwortlichkeiten haben und in einer definierten Vererbungs- bzw. Benutzungsbeziehungen zueinander stehen
- ✍ kann in seiner Dynamik verstanden werden als eine Menge von Objekten, die nach einem beschreibbaren Prinzip interagieren bzw. erzeugt werden
- ✍ kann immer nur zusammen mit dem Entwurfsproblem beschrieben werden, das es lösen soll.

5.2 Wozu Entwurfsmuster?

Entwurfsmuster

- ✍ helfen, existierende Softwareentwürfe zu analysieren und zu reorganisieren
- ✍ erleichtern die Einarbeitung in Software-Architekturen (z.B. Klassenbibliotheken, Rahmenwerke), solange sie auf der Basis von bekannten Entwurfsmustern dokumentiert sind
- ✍ sind “Mikroarchitekturen”, die sich von erfahrenen Entwicklern als Bausteine innerhalb größerer Software-Architekturen wiederverwenden lassen (Wiederverwendung von Design-Lösungen statt Wiederverwendung von Code).
- ✍ stellen uns die Elemente einer Sprache, in der wir über Software-Architekturen nachdenken und kommunizieren können.
- ✍ sollen die softwaretechnische Qualität von Entwürfen erhöhen (z.B. ihre Wiederverwendbarkeit und Erweiterbarkeit).

5.3 Beschreibung von Entwurfsmustern

Eine Musterbeschreibung besteht meist aus folgenden Teilen:

1. **Name:** Treffender Name für die Essenz des Musters
2. **Kontext :** In welchem Kontext ist das Muster überhaupt relevant und einsetzbar
3. **Problem:** Welches Problem kann im gegebenen Kontext auftauchen und bedarf einer Lösung?
4. **Lösung:** Wie kann das Problem gelöst werden? Welche Bestandteile sind nötig und welche Interaktionen finden zwischen diesen statt? Häufig begleitet von Klassendiagrammen und/oder Sequenzdiagrammen
5. **Folgerungen:** Diskussion der Vor- und Nachteile. Effizienz, Wiederverwendbarkeit, Flexibilität, Erweiterbarkeit
6. **Beispiele:** Ein Beispiel, bei dem Kontext und Problem gut erkennbar sind und die Lösung aufgezeigt wird
7. **Varianten:** Gibt es Varianten der Lösung oder der Problemstellung und wie sehen diese aus?

5.4 Arten von Mustern

Software-Muster

- Architekturmuster
- Entwurfsmuster
- Idiome

Prozess-Muster

- Softwareentwicklung
- Projektmanagement
- Gesamtsystem- und Betriebsebene

5.5 Klassifikation von Mustern

- ✍ **Erzeugungsmuster (creational patterns)**
 - Helfen, ein System unabhängig davon zu machen, wie seine Objekte erzeugt, zusammengesetzt und repräsentiert werden
- ✍ **Strukturmuster (structural patterns)**
 - Befassen sich mit der Zusammensetzung von Klassen und Objekten zu größeren Strukturen
- ✍ **Verhaltensmuster (behavioural patterns)**
 - Befassen sich mit der Interaktion zwischen Objekten und Klassen
 - Beschreiben komplexe Kontrollflüsse, die zur Laufzeit schwer nachvollziehbar sind
- ✍ **Klassenbasierte Muster**
 - Behandeln Beziehungen zwischen Klassen
 - Ausgedrückt durch Vererbungsstrukturen
 - Festgelegt zur Übersetzungszeit
- ✍ **Objektbasierte Muster**
 - Beschreiben Beziehungen zwischen Objekten, die zur Laufzeit geändert werden können
 - Benutzen auch bis zu einem gewissen Grad die Vererbung

5.6 Entwurfsmuster-Katalog (nach Gamma et. al.)

Erzeugungsmuster

- Factory Method
- Abstract Factory
- Builder
- Prototype
- Singleton

Strukturmuster

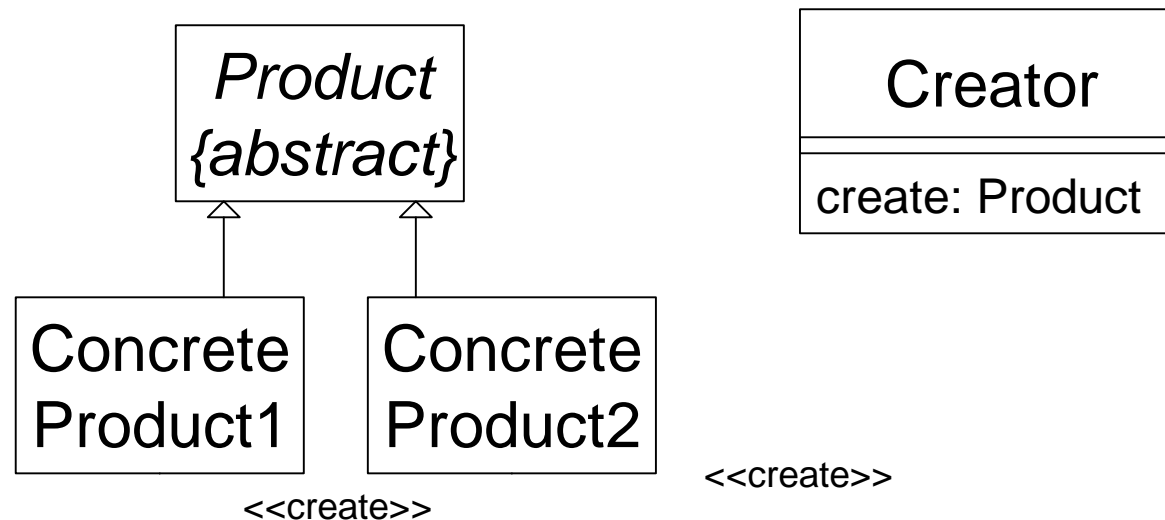
- Adapter
- Bridge
- Composite
- Facade
- Flyweight
- Proxy

Verhaltensmuster

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor


5.7.1 Erzeugungsmuster „*Factory Method*“

- ✍ **Name:** Factory Method (dt. Fabrikmethode oder virtueller Konstruktor)
- ✍ **Problem:** Bei der Erzeugung von Objekten soll zwischen Varianten gewählt werden; dies soll automatisch zum Zeitpunkt der Erzeugung geschehen.
- ✍ **Lösung:**



5.7.2 Erzeugungsmuster „Singleton“

 **Name:** Singleton

 **Problem:** Sicherstellen, dass es immer nur eine Instanz einer Klasse gibt.

 **Kontext:** Oft darf es nicht mehrere Instanzen einer Klasse geben:

- Printer Spooler
- Timer

 **Lösung:**

Singleton
static Instance() SingletonOperation() GetSingletonData()
static uniqueInstance() singletonOperation()

5.7.2 Erzeugungsmuster „*Singleton*“ (2)

Folgerungen:

- Kontrollierter Zugriff
- Sauberer Namensraum
- Verfeinerung durch Vererbung

Hintergrund: Klassenvariablen und Methoden

- Definition durch das Keyword „static“
- Klassenvariablen
 - Gemeinsamer Zustand aller Objekte dieser Klasse
 - Existieren exakt ein Mal (wie globale Variablen)
 - Müssen explizit definiert und initialisiert werden
- Klassenmethoden
 - Zugriff nur über Klassennamen

Singleton* s = Singleton::Instance();

 - Zugriff nur von Klassenmethoden

5.7.2 Erzeugungsmuster „*Singleton*“ (3)

Header-Datei

```
class Singleton {  
    public:  
        // Zugriff auf das Singleton  
        static Singleton* Instance();  
    protected:  
        // Verhindere explizite Konstruktion  
        Singleton();  
    private:  
        // Verweis auf das einzige Singleton-Objekt  
        static Singleton* instance;  
};
```

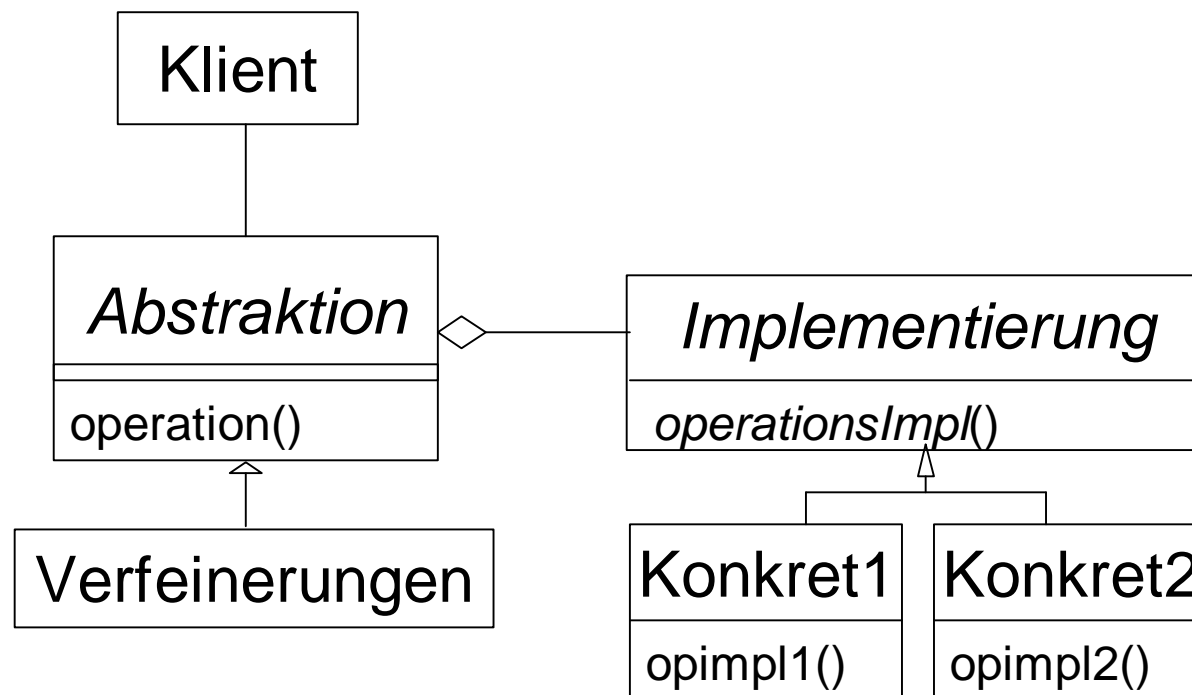
5.7.2 Erzeugungsmuster „*Singleton*“ (4)

Source-Datei

```
// Initialisierung der Klassenvariable
Singleton* Singleton::instance= NULL;
// Zugriffsfunktion
Singleton* Singleton::Instance()
{
    if (instance == 0)
        instance= new Singleton;
    return instance;
}
```

5.7.3 Strukturmuster „*Bridge*“

- ✍ **Name:** Bridge
- ✍ **Problem:** Eine Abstraktion von Ihrer Implementierung entkoppeln, so dass beide unabhängig variierbar sind.
- ✍ **Lösung:**



5.8 Zusammenfassung Entwurfsmuster

Entwurfsmuster bieten

- Ein gemeinsames Entwurfsvokabular („Lass uns ein Singleton einstzen“)
- Dokumentation und Lernhilfe
- Eine Ergänzung zu bestehenden Methoden (Entwurfsmuster sind keine Notation sondern fassen die Erfahrung von Experten zusammen)

“The best designs will use many design patterns that dovetail and intertwine to produce a greater whole.”