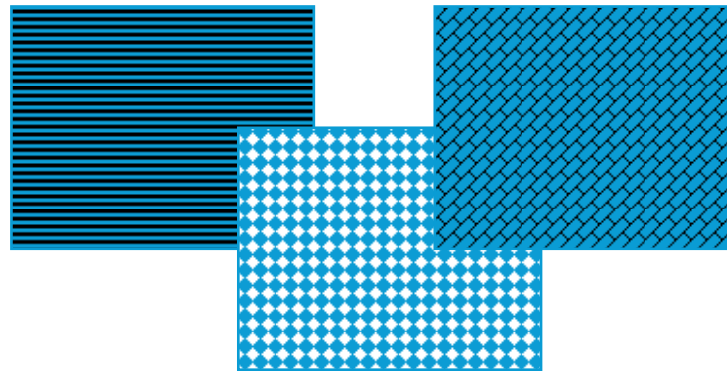




LÖSUNGSSTRATEGIEN IN DER SOFTWARE ENTWICKLUNG

Einführung in Software Design Patterns nach GoF

Danielle Klatt, Triona GmbH
18. Juli 2008



GLIEDERUNG

- Einführung
 - Wer oder was ist GoF?
 - Pattern – wozu braucht ein Software-Entwickler das?
- Design Patterns nach GoF
 - Erzeugung von Objekten – Creational Patterns
 - Struktur von Objekten – Structural Patterns
 - Verhalten von Objekten – Behavioural Patterns
- Anti-Patterns

GOF – GANG OF FOUR

- Erich Gamma
 - Schweizer Informatiker
 - Mitentwickler von JUnit für Java
- Richard Helm
- Ralph Johnson
- John Vlissides

PATTERN – WAS GENAU IST DAS?

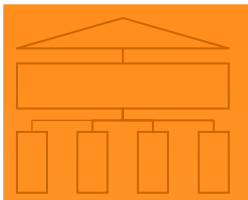
- Pattern (engl.: Muster, Modell) = Vorlage, Modell, wonach etwas hergestellt wird
- Design Pattern = allgemeines, wiederverwendbares *Lösungsmuster* für ein wiederkehrendes Problem im Software Design
- PRO:
 - Beschleunigung des Entwicklungsprozesses
 - Erhöhung der Lesbarkeit im Code
 - Erleichterung der Kommunikation



DESIGN PATTERNS NACH GOF



Creational Patterns: Erzeugung von Objekten



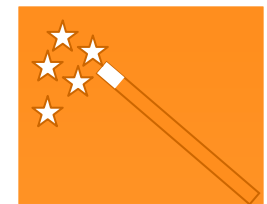
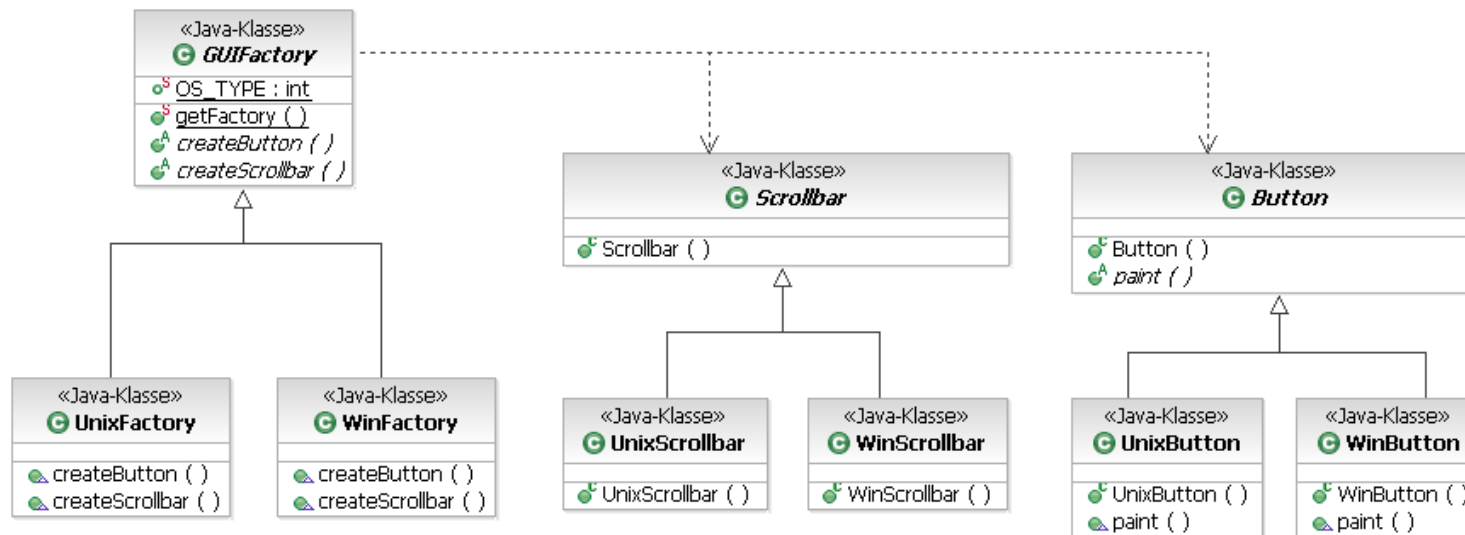
Structural Patterns: Struktur von Objekten



Behavioural Patterns: Verhalten von Objekten

ABSTRACT FACTORY

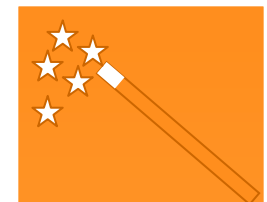
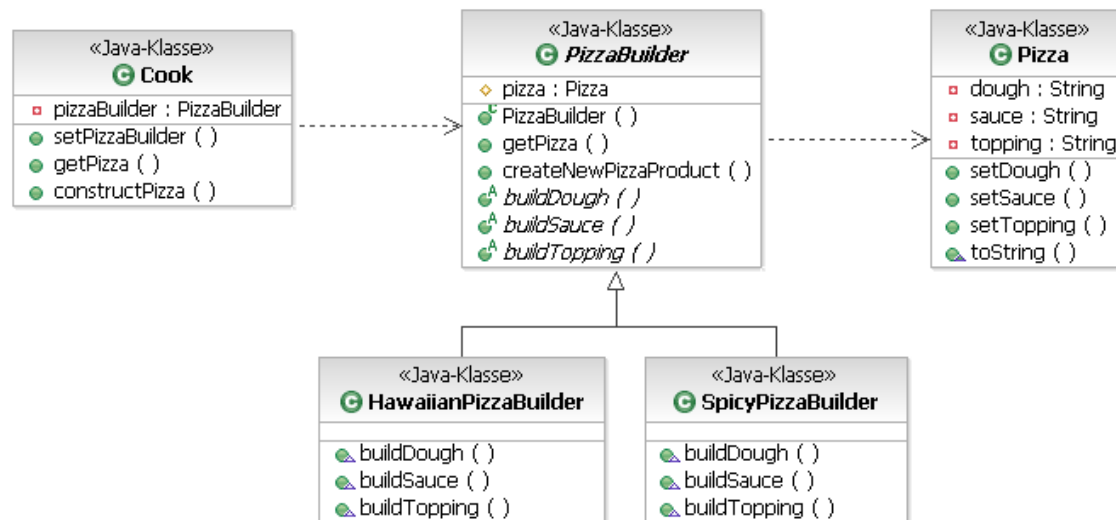
- „Provide an interface for creating families of related or dependent objects without specifying their concrete class“ ([GOF2005] p. 87)
- **Problemstellung:** Eine Anwendung soll auf unterschiedliche Plattformen portabel sein.
Gefahr: Eine Fülle von if-/else-Abfragen für alle unterstützten Plattformen im Code verteilt.
- **Lösung:** Hinzufügen einer Abstraktionsebene zur Erstellung einer Gruppe von verwandten oder abhängigen Objekten.



Creational

BUILDER

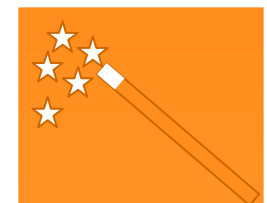
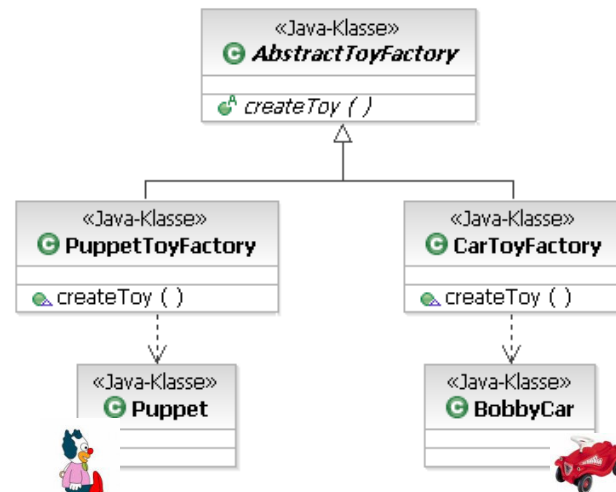
- „Separate the construction of a complex object from its representation so that the same construction process can create different representations“ ([GOF2005] p. 97)
- **Problem:** Eine Anwendung muss komplexe Objekte erzeugen.
- **Lösung:** Gleicher Konstruktionsprozess erzeugt unterschiedliche Repräsentationen ==> Schritt-für-Schritt-Erzeugung des Produkts (Objekts)



Creational

FACTORY METHOD

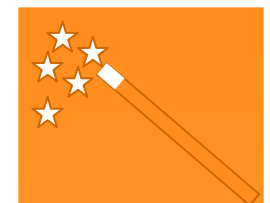
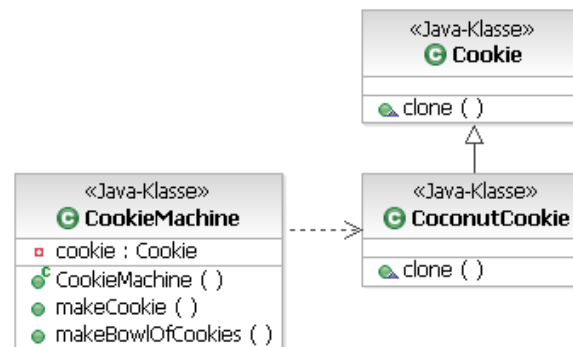
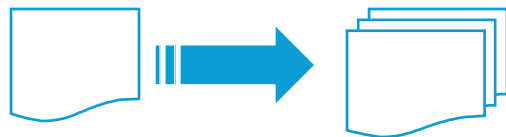
- „Define an interface for creating an object, but let the subclasses decide which class to instantiate. Factor Method lets a class defer instantiation to subclasses.“ ([GOF2005] p. 107)
- **Problem:** Ein Framework soll das Architekturmodell für eine Reihe von Applikationen standardisieren. Die einzelnen Applikationen sollen jedoch selbst Ihre eigenen Objekte definieren und instanziiieren.
- **Lösung:** Definition eines virtuellen Konstruktors : Abstrakte Superklasse spezifiziert das Standardverhalten (durch Platzhalter) und delegiert die Details an seine Unterklassen.



Creational

PROTOTYPE

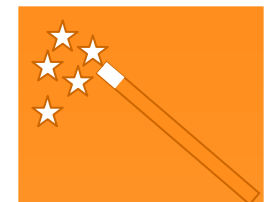
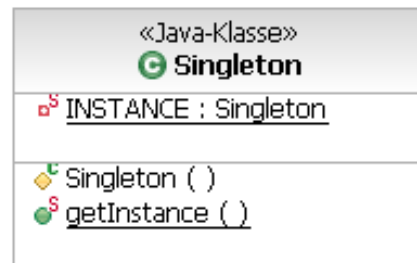
- „Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.“ ([GOF2005] p. 117)
- **Problem:** Erzeugung einer Menge von gleichen Objekten
- **Lösung:** Nutzung einer Instanz als Basis für alle weiteren Instanzen mit Hilfe einer clone()-Methode.



Creational

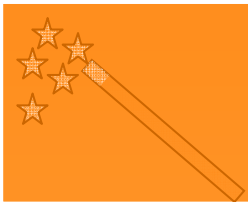
SINGLETON

- „Ensure a class only has one instance, and provide a global point of access to it.“ ([GOF2005] p. 127)
- **Problem:** Eine Applikation benötigt eine einzige Instanz eines Objekts.
- **Lösung:** Die einzige Instanz wird als privates, statisches Feld im Objekt selbst gehalten ==> „Just-in-Time-Initialisierung“ bzw. „Initialisierung bei erster Nutzung“



Creational

DESIGN PATTERNS NACH GOF



Creational Patterns: Erzeugung von Objekten



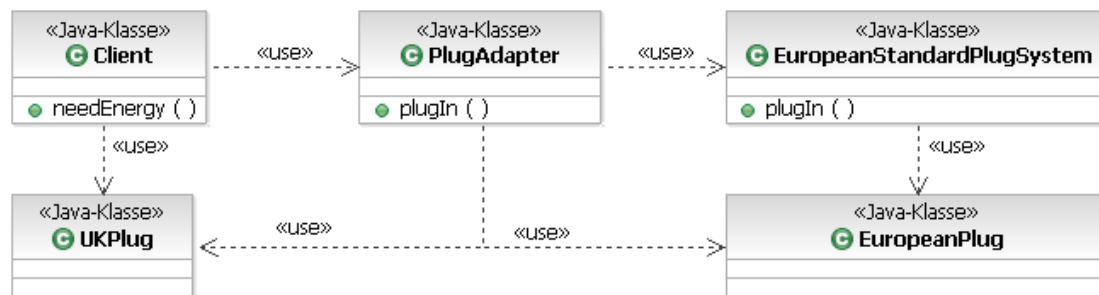
Structural Patterns: Struktur von Objekten



Behavioural Patterns: Verhalten von Objekten

ADAPTER (WRAPPER)

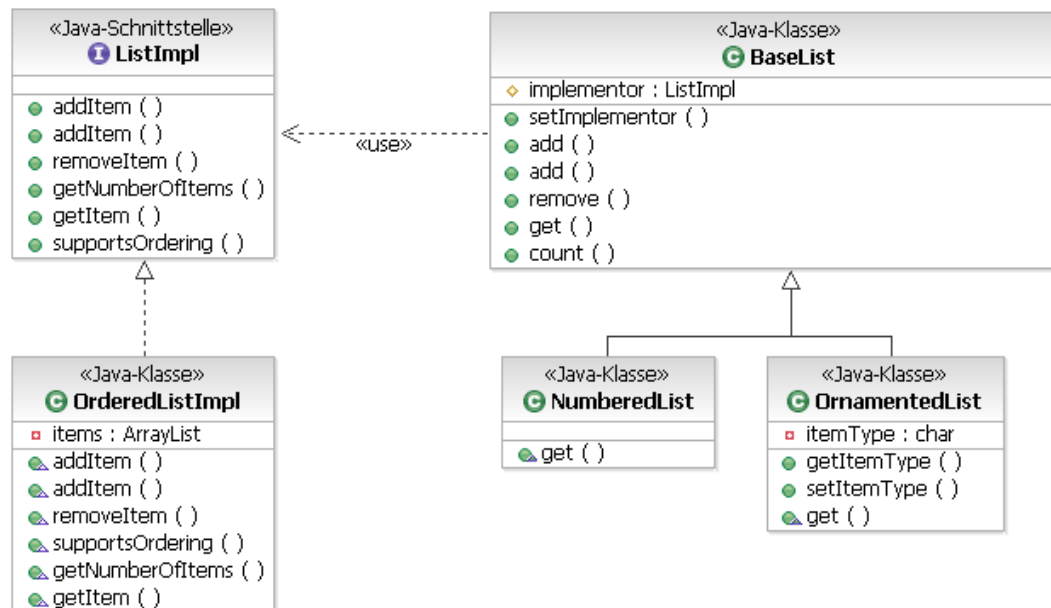
- „Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.“ ([GOF2005] p. 139)
- **Problem:** Schnittstelle eines alten Systems soll wiederverwendet werden, ist jedoch inkompatibel mit dem neuen System
- **Lösung:** Einführung einer Abstraktionsebene zur Übersetzung der alten Schnittstelle für das neue System



Structural

BRIDGE

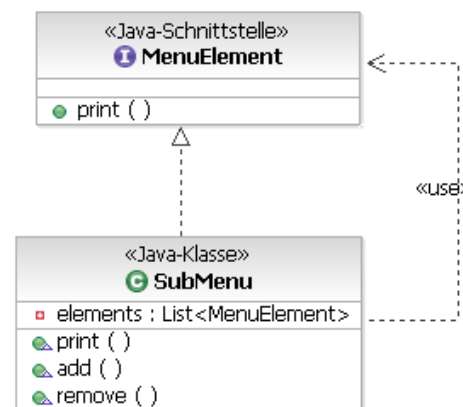
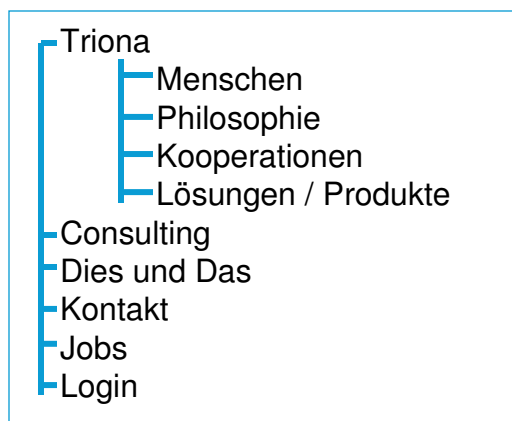
- „Decouple an abstraction from its implementation so that the two can vary independently.“ ([GOF2005] p. 151)
- **Problem:** Rasches Anwachsen von Klassen aufgrund gekoppelter Interfaces und einer großen Anzahl von Implementierungen
- **Lösung:** Veröffentlichung des Interfaces in einer Vererbungshierarchie und Haltung der Implementierung in einer eignen



Structural

COMPOSITE

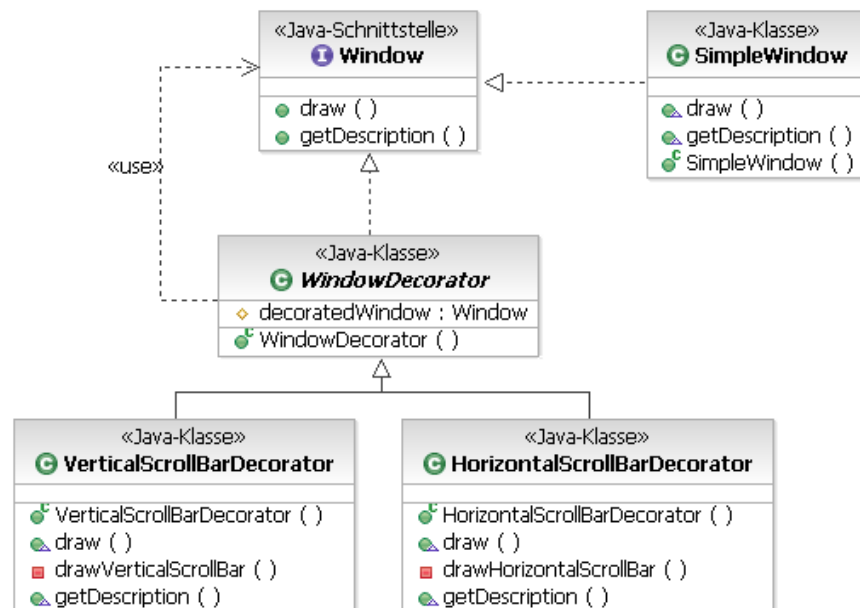
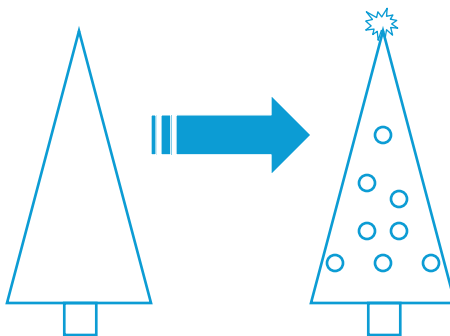
- „Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.“ ([GOF2005] p. 163)
- **Problem:** Manipulation einer hierarchischen Sammlung von einfachen und komplexen Typen in der gleichen Art und Weise. Unterschiede zwischen den einzelnen Objekten sind nebensächlich.
- **Lösung:** Abstrakte Basisklasse spezifiziert das Verhalten für alle einfachen und komplexen Elemente. ==> Verzeichnisse enthalten Elemente, die wiederum Verzeichnisse sein können.



Structural

DECORATOR

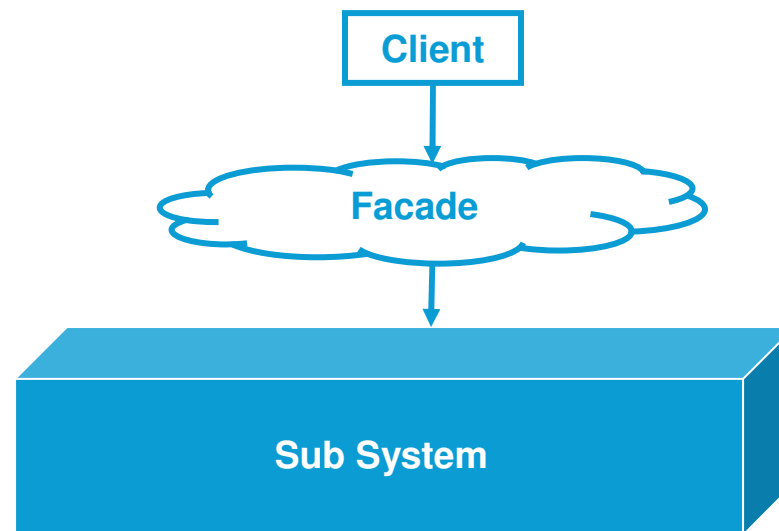
- „Attach additional responsibilities to an object dynamically . Decorators provide a flexible alternative to subclassing for extending functionality.“ ([GOF2005] p. 175)
- **Problem:** Hinzufügen von Status oder Verhalten zu einem Objekt zur Laufzeit.
- **Lösung:** Kapselung des Original-Objekts in einem abstrakten Wrapper (Decorator). Sowohl der Decorator als auch die Basisklasse erben von den selben Interface



Structural

FACADE

- „Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.“ ([GOF2005] p. 185)
- **Problem:** Ein Client benötigt einen vereinfachten Zugriff auf ein komplexes Subsystem
- **Lösung:** Verstecken der Komplexität eines Teilsystems hinter einer vereinfachten Schnittstelle.



Structural

FLYWEIGHT

- „Use sharing to support large numbers of fine-grained objects efficiently.“ ([GOF2005] p. 195)
- **Problem:** Objekte mit feiner Granularität ermöglichen eine optimale Flexibilität, können sich jedoch negativ auf Performance und Speichernutzung auswirken.
- **Lösung:** Teilung der Objekte in einen statischen und einen dynamischen Teil.
Dynamischer Teil = Flyweight-Objekt und wird wiederverwendet; statischer Teil wird zwischengespeichert (z.B. vom Client) und bei Bedarf rekonstruiert.

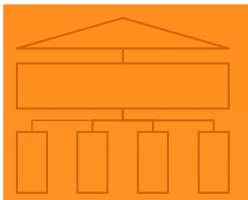


Structural

DESIGN PATTERNS NACH GOF



Creational Patterns: Erzeugung von Objekten



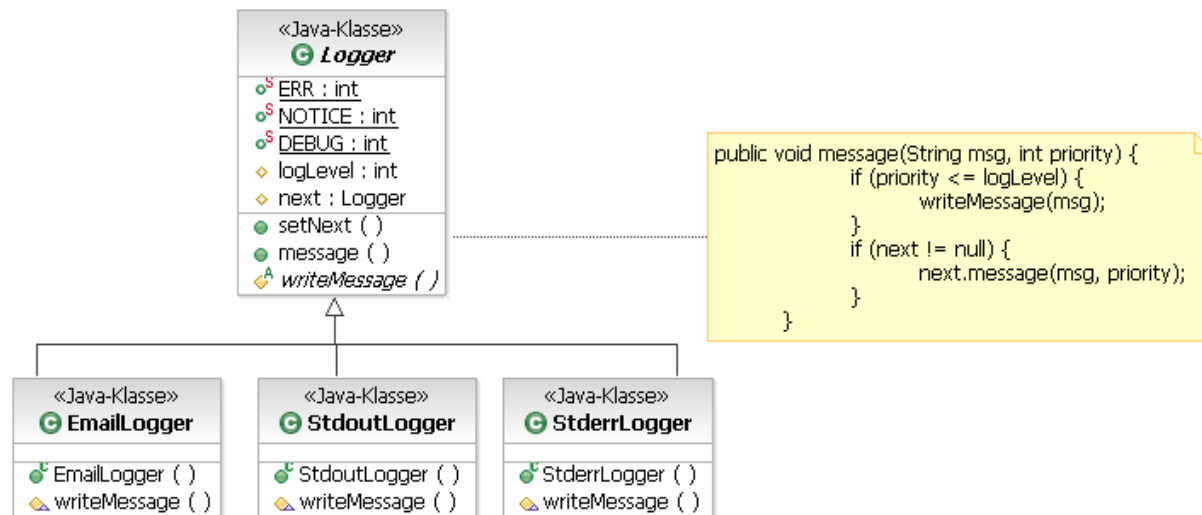
Structural Patterns: Struktur von Objekten



Behavioural Patterns: Verhalten von Objekten

CHAIN OF RESPONSIBILITY

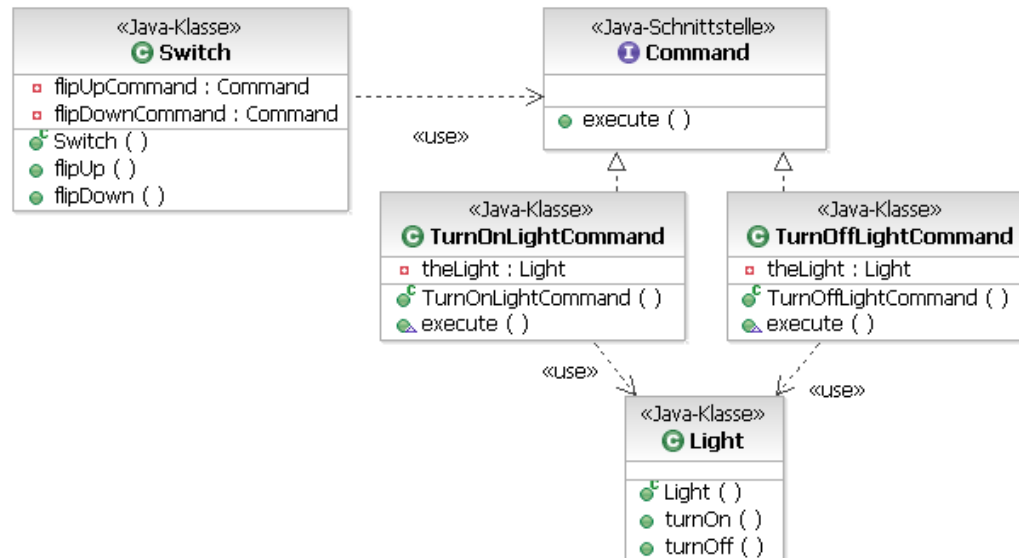
- „Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.“ ([GOF2005] p. 223)
- **Problem:** Es gibt eine Reihe möglicher Verarbeitungselemente und ein Fluss von Anfragen muss bearbeitet werden.
- **Lösung:** Verknüpfung der Verarbeitungselemente in einer Verarbeitungskette. Eine Anfrage wird die Kette entlang gereicht, bis das zuständige Element diese Anfrage bearbeitet.



Behavioural

COMMAND

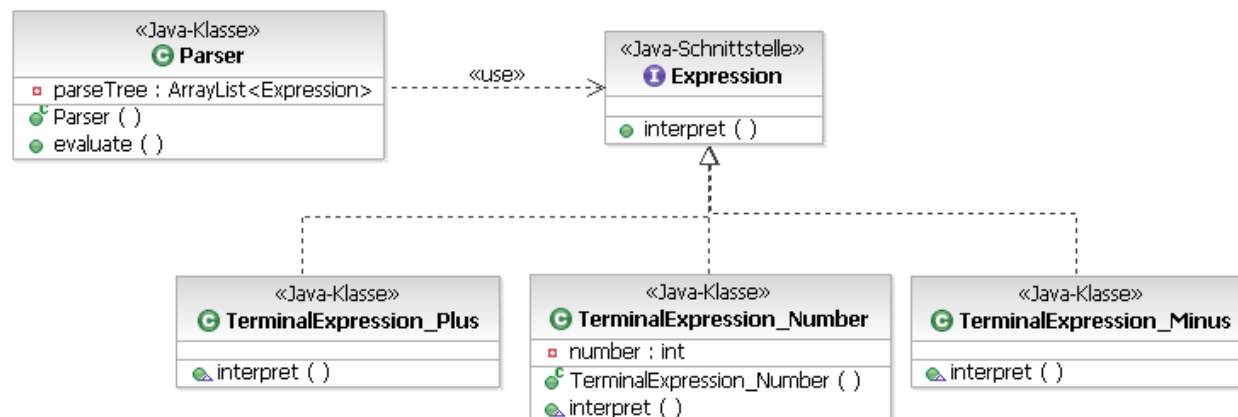
- „Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.“ ([GOF2005] p. 233)
- **Problem:** Anfragen an Objekte sollen gestellt werden, ohne etwas über die benötigte Methode und den Empfänger der Anfrage zu kennen.
- **Lösung:** Einführung einer abstrakten Basisklasse, welche den Empfänger (das Objekt) mit einer Aktion koppelt. Die execute()-Methode ruft die Aktion beim Empfänger auf.



Behavioural

INTERPRETER

- „Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.“ ([GOF2005] p. 243)
- **Problem:** Art von Problemen tritt wiederholt in einer definierten Domäne auf. Wenn sich die Domäne als Sprache charakterisieren lässt, könnten diese Probleme mit Hilfe eines Übersetzers gelöst werden.
- **Lösung:** Abbildung einer Domäne auf eine Sprache, die Sprache auf eine Grammatik und die Grammatik auf ein objekt-orientiertes Design.



Behavioural

ITERATOR

- „Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.“ ([GOF2005] p. 257)
- **Problem:** Eine abstrakte Navigation durch durchmischte Datenstrukturen ist erforderlich.
- **Lösung:** Eine Liste ermöglicht Zugriff auf seine Elemente ohne seine innere Struktur preis zu geben. Zusätzlich können unterschiedliche Arten der Navigation möglich sein. Verantwortung für Zugriff auf und Navigation durch die Liste wird an ein Iterator-Objekt übertragen.

```

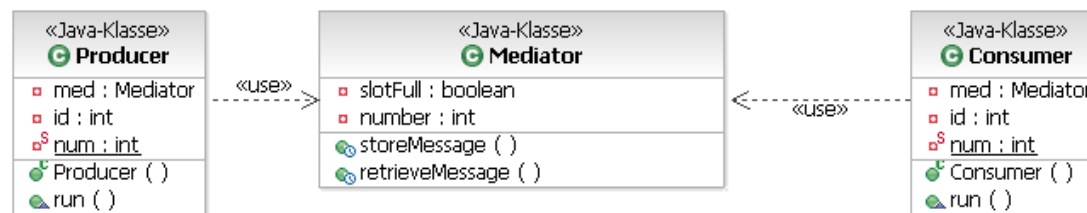
«Java-Schnittstelle»
+ Set
+ size ( )
+ isEmpty ( )
+ contains ( )
+ iterator ( )
+ toArray ( )
+ toArray ( )
+ add ( )
+ remove ( )
+ containsAll ( )
+ addAll ( )
+ retainAll ( )
+ removeAll ( )
+ clear ( )
+ equals ( )
+ hashCode ( )
    
```



Behavioural

MEDIATOR

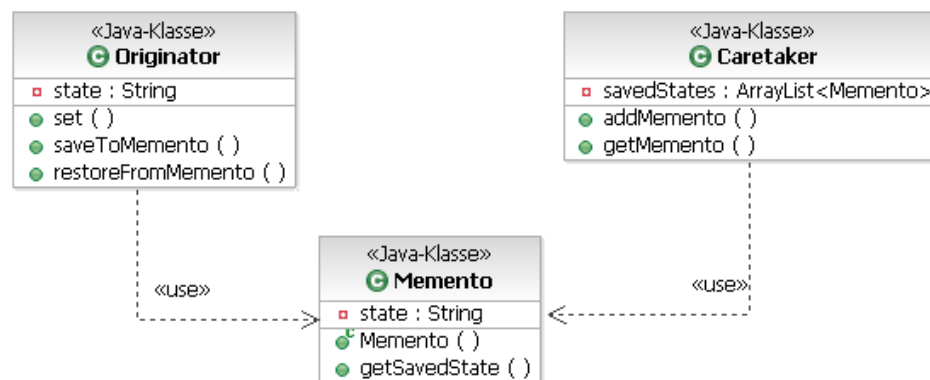
- „Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.“ ([GOF2005] p. 273)
- **Problem:** Abhängigkeiten zwischen Objekten führen zu hoher Komplexität (Spaghetti-Code)
- **Lösung:** Einführung einer Abstraktionsschicht, welche die Interaktion zwischen den unterschiedlichen Objekten steuert.



Behavioural

MEMENTO (TOKEN)

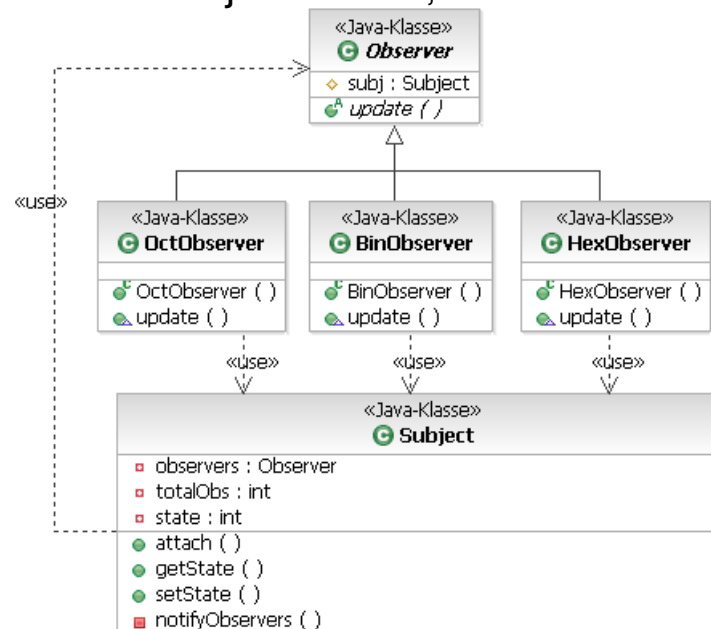
- „Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.“ ([GOF2005] p. 283)
- **Problem:** Ein Objekt muss auf einen früheren Status zurückgesetzt werden (Undo).
- **Lösung:** Der Status eines Objekts (Originator) wird zu einem bestimmten Zeitpunkt festgehalten (Memento) und von einem anderen Objekt verwaltet (Caretaker). Wenn das Objekt auf diesen Zustand zurückgesetzt werden soll, wird der entsprechende Zustand (Memento) beim Caretaker angefragt.



Behavioural

OBSERVER

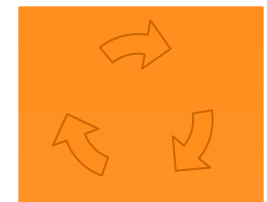
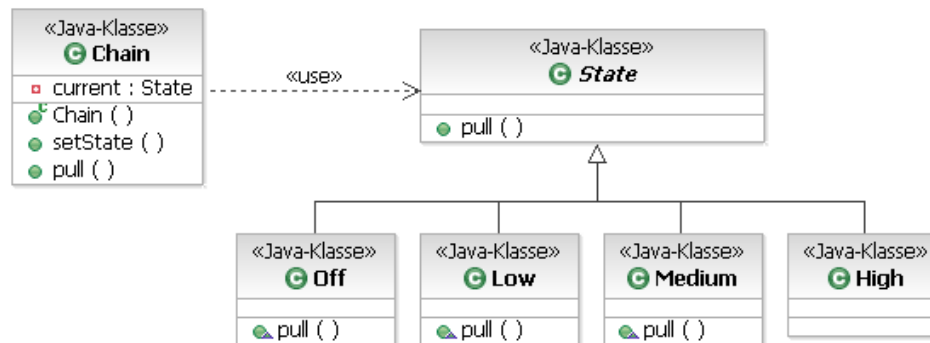
- „Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.“ ([GOF2005] p. 293)
- **Problem:** Ein Ereignis soll überwacht werden. Sobald ein Ereignis auftritt, soll das System darauf reagieren.
- **Lösung:** Beobachter (Observer) registrieren sich an einem Subjekt, wenn sie erzeugt werden. Sobald sich das Subjekt ändert, werden alle registrierten Beobachter informiert.



Behavioural

STATE

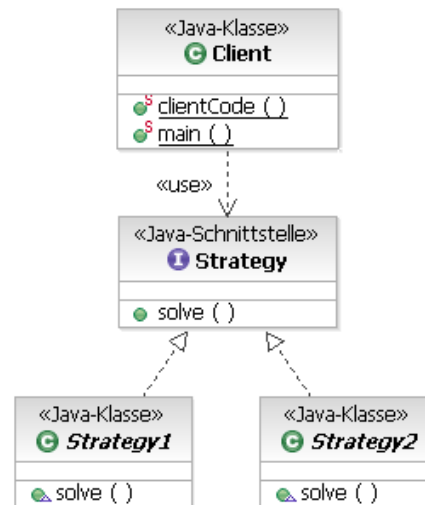
- „Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.“ ([GOF2005] p. 305)
- **Problem:** Ein Objekt muss zur Laufzeit sein Verhalten ändern, sobald sich sein Zustand verändert.
- **Lösung:** Ein Kontext bildet die Schnittstelle nach außen und verwaltet die aktuelle Statusausprägung. Eine abstrakte Klasse bildet den Status ab. Jede Ausprägung des Status wird durch eine Subklasse des abstrakten Status abgebildet ==> Objekt-orientierte „State Machine“



Behavioural

STRATEGY

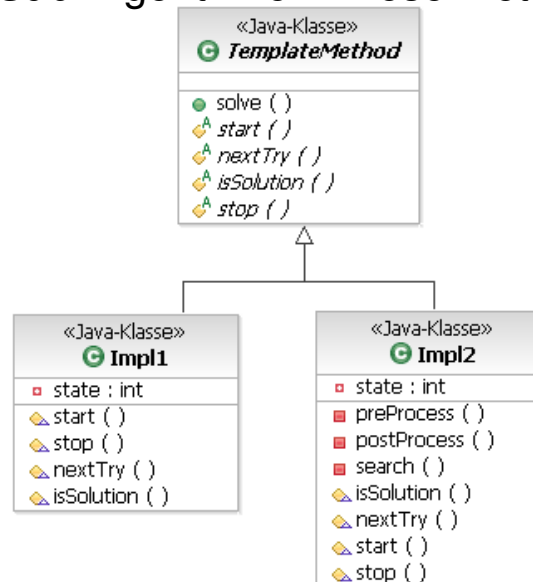
- „Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.“ ([GOF2005] p. 315)
- **Problem:** Clients sollen von der eigentlichen Implementierung des Algorithmus entkoppelt werden.
- **Lösung:** Abstraktion eines Algorithmus als Interface und die Implementierung unterschiedlicher Algorithmen in den abgeleiteten Klassen.



Behavioural

TEMPLATE METHOD

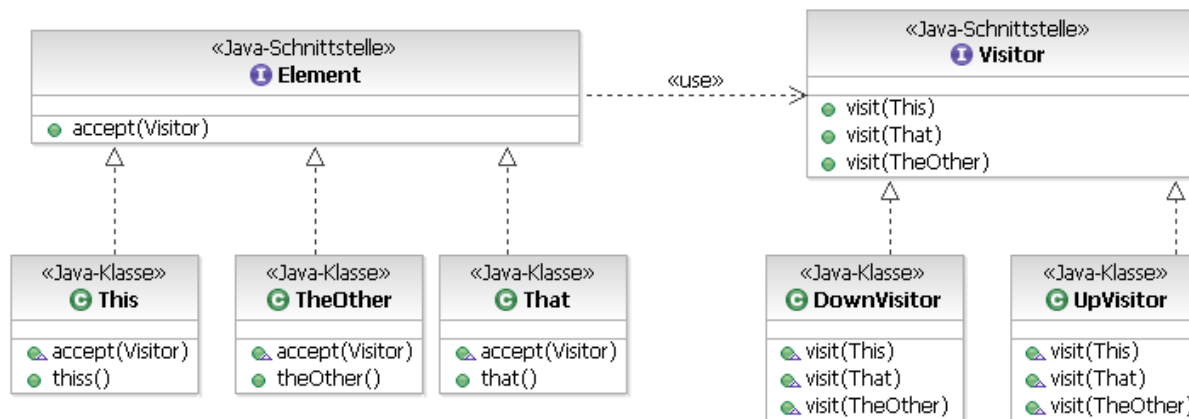
- „Define a skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.“ ([GOF2005] p. 325)
- **Problem:** Komponenten haben signifikante Gemeinsamkeiten (gleicher Ablauf) und können doch nicht durch ein gemeinsames Interface abgebildet werden. Gefahr: Redundanz
- **Lösung:** Definition des Standard-Algorithmus in einer abstrakten Basisklasse mit Platzhalter-Methoden für Sub-Algorithmen. Diese Methoden werden in den Subklassen implementiert.



Behavioural

VISITOR

- „Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.“ ([GOF2005] p. 223)
- **Problem:** Viele unterschiedliche Operationen müssen an Objekten in einer heterogenen komplexen Struktur durchgeführt werden. Die Objekte sollen jedoch nicht um diese Operationen erweitert werden.
- **Lösung:** Abstrakte Basisklasse Visitor mit „visit()“-Methode = Basis für Knotenobjekte. Elemente, auf denen Methoden ausgeführt werden sollen: accept()-Methode, um die Visitor-Klasse entgegenzunehmen.



Behavioural

ANTI-PATTERNS

- Anti = (griech: „gegen“)
- Allgemeine Lösung eines Problems mit negativen Auswirkungen
- Falsche Anwendung von Design Patterns (falscher Kontext)
- Fehlende Design-Erfahrung (Copy & Paste, Spaghetti-Code,..)
- Beispiel: Strategy vs. Template Method

REFERENZEN

- [GOF2005] - Design Patterns: Elements of Reusable Object-Oriented Software, E. Gamma, R. Helm, R. Johnson, J. Vlissides; Addison-Wesley, 1995, 32nd Printing, April 2005
- [MEY2007] - <http://lexikon.meyers.de/>
- [WIK2008] - http://en.wikipedia.org/wiki/Design_pattern_%28computer_sience%29
- [JAV2008] - <http://www.java2s.com/Code/Java/Design-Pattern/CatalogDesign-Pattern.htm>
- [SOU2008] - http://sourcemaking.com/design_patterns

FRAGEN / DISKUSSION