Softwaretechnik

Entwurfsmuster

Burkhardt Renz

Fachbereich MNI
Technische Hochschule Mittelhessen

Sommersemester 2012

Inhalt

Entwurfsmuster

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

– Christopher Alexander

Design Patterns: descriptions of communicating objects and classes that are customized to solve a general design problems in a particular context.

– GoF

Literatur zu Entwurfsmustern


Craig Larman
Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process
Prentice-Hall


Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley


John Vlissides
Pattern Hatching, Addison Wesley (dt. unter dem Titel: Entwurfsmuster anwenden)


Eric Freeman, Elisabeth Freeman
Entwurfsmuster von Kopf bis Fuß, O'Reilly

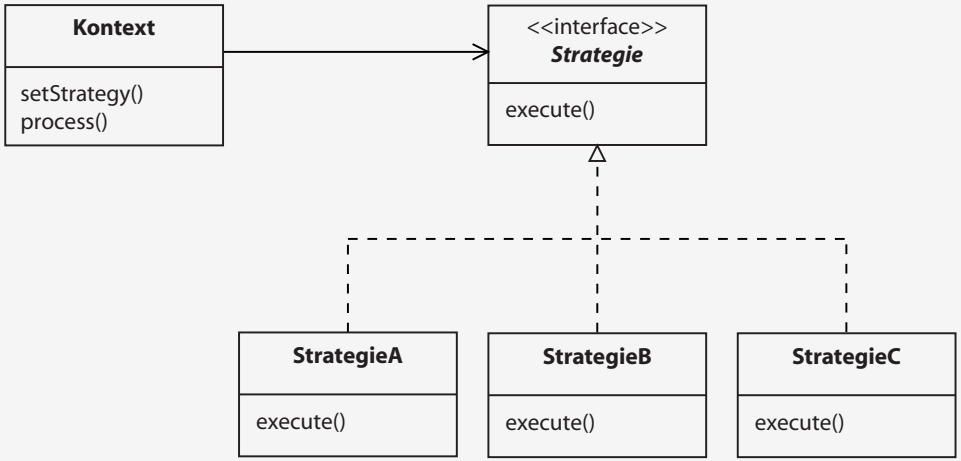
Inhalt

Strategy

Zweck

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. [GoF]

Struktur

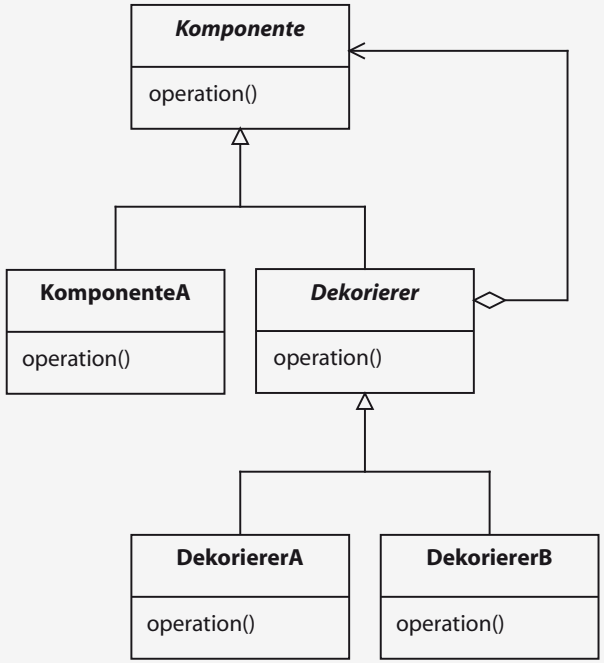


Decorator

Zweck

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. [GoF]

Struktur

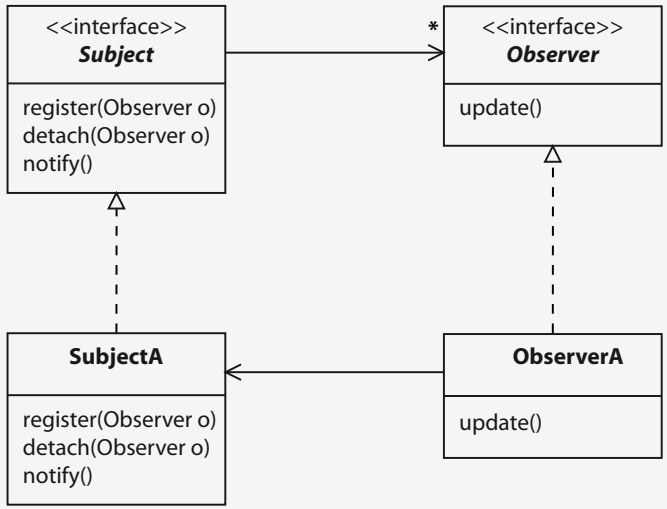


Observer

Zweck

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. [GoF]

Struktur



Inhalt

Zusammenspiel von Entwurfsmustern

Wir konstruieren eine API zu einem hierarchischen Dateisystem und verwenden dabei die folgenden Entwurfsmuster:

- Composite
- Proxy
- Visitor
- Singleton
- Mediator

Schritt 1

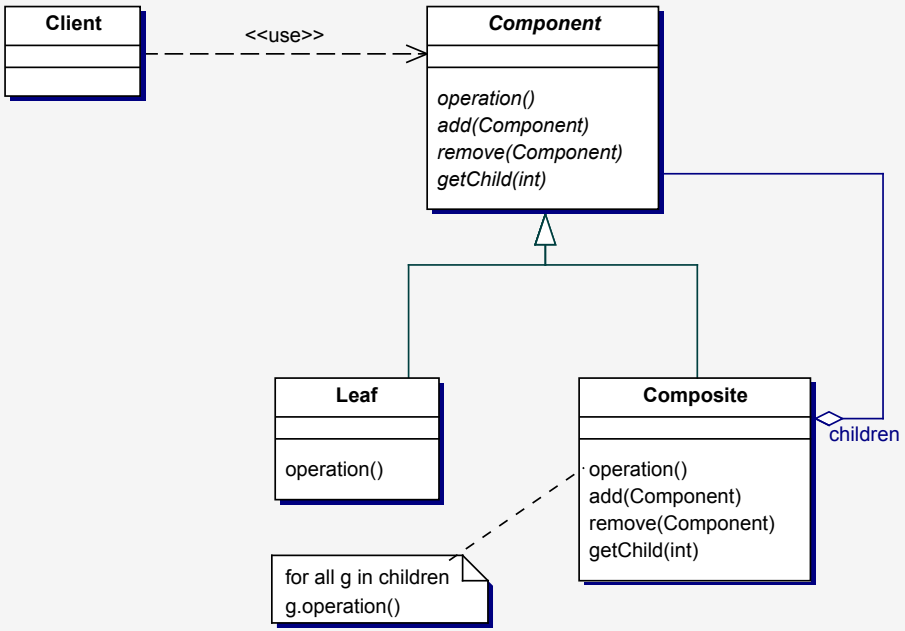
- Ein hierarchisches Dateisystem ist ein Baum von Verzeichnissen und Dateien.
- Viele Kommandos, wie etwa ls (list files and directories), können auf Verzeichnisse und Dateien angewandt werden.
- Wie sieht eine Struktur aus, in der es möglich ist, Verzeichnisse und Dateien gleich zu behandeln?
- Entwurfsmuster Composite.

Composite

Zweck

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. [GoF]

Struktur



Composite II

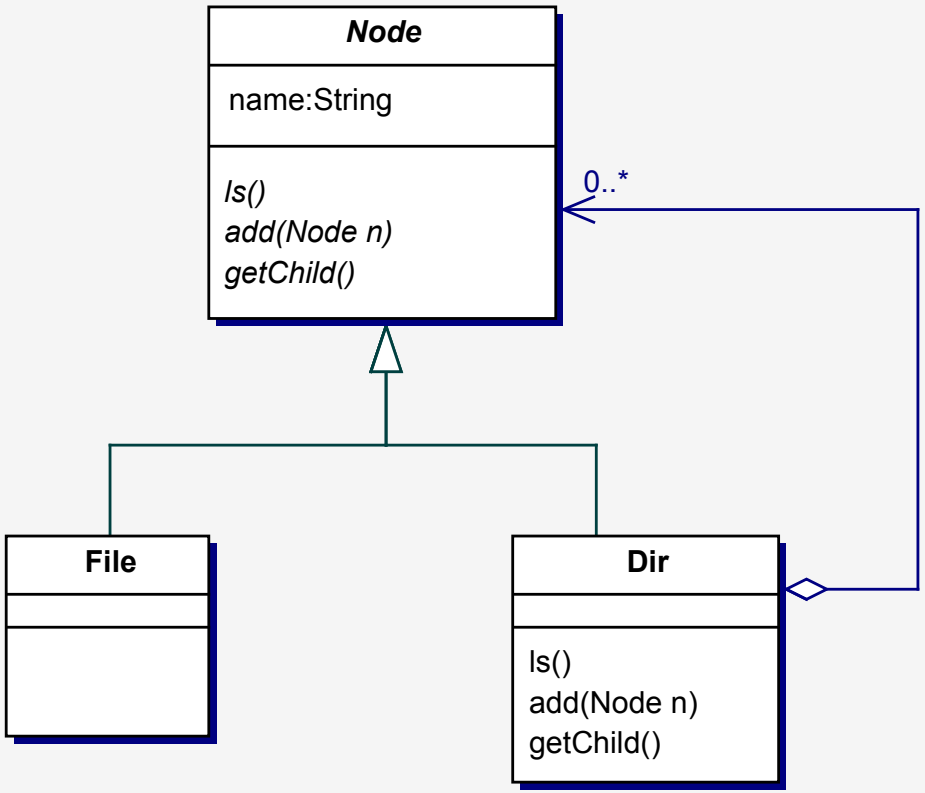
Eigenschaften

- Entwurf von Hierarchien von Aggregationen (Bäume)
- Clients können individuelle Elemente und Kompositionen gleichartig handhaben (siehe `ls()`)
- Neue Arten von Elementen können leicht hinzugefügt werden (siehe Schritt 2)
- Restriktion auf bestimmte Arten von Elementen jedoch schwierig

Beispiele

- View-Klasse in SmallTalks MVC-Architektur
- Viele Frameworks für graphische Benutzeroberflächen verwenden das Muster

Struktur nach Schritt 1



Code von ls

Wie arbeitet die Methode ls()?

Klasse Node

```
public void ls() {
  System.out.println( name );
}
```

Klasse Dir

```
@Override
public void ls() {
  super.ls();
  for ( Node n: nodes ) {
    n.ls();
  }
}
```

Schritt 2

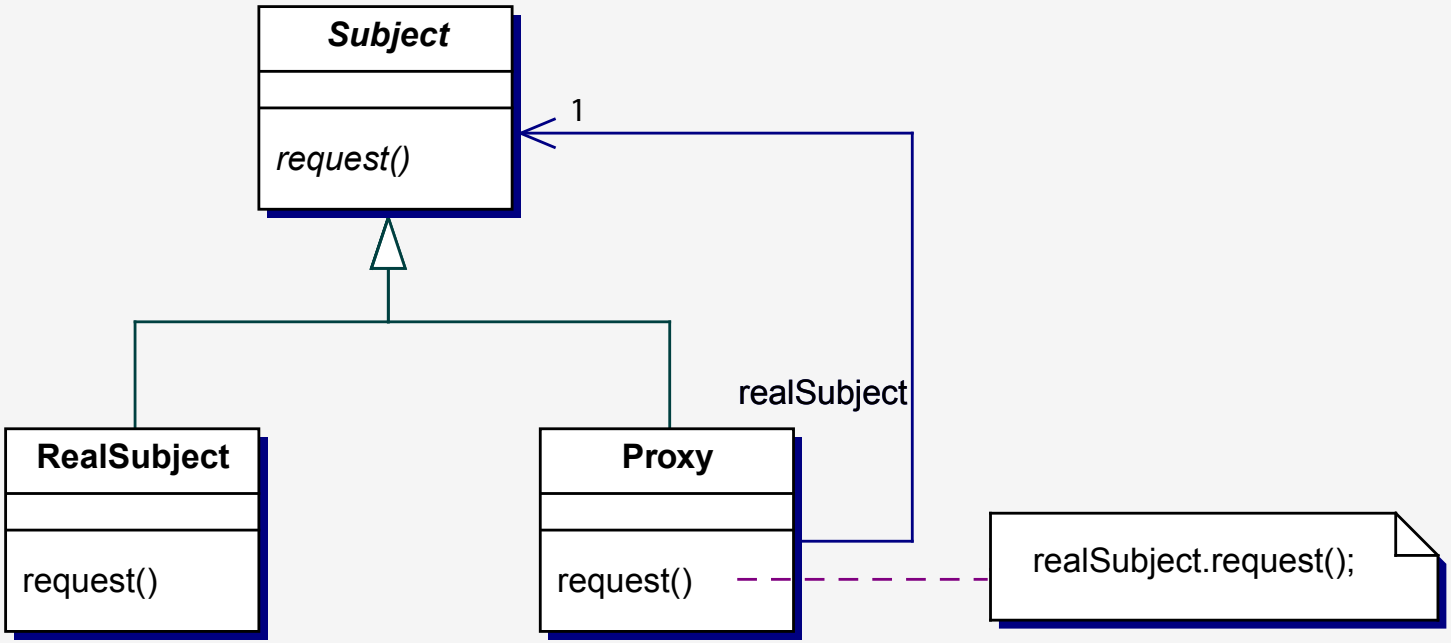
- Im zweiten Schritt wollen wir einen *symbolischen Link* einführen
- Es soll (wie etwa unter UNIX) möglich sein, dass eine Datei in mehreren Verzeichnissen eingetragen ist, als Link also.
- Wir wollen unsere Struktur erweitern, ohne sie stark zu ändern. Insbesondere soll das Entwurfsmuster Composite beibehalten werden.
- Das geht gut, denn Composite erlaubt die Erweiterung durch weitere Typen von Elementen in der Struktur.
- Für den Link selbst verwenden wir das Muster Proxy.

Proxy

Zweck

Provide a surrogate or placeholder for another object to control access to it. [GoF]

Struktur



Proxy II

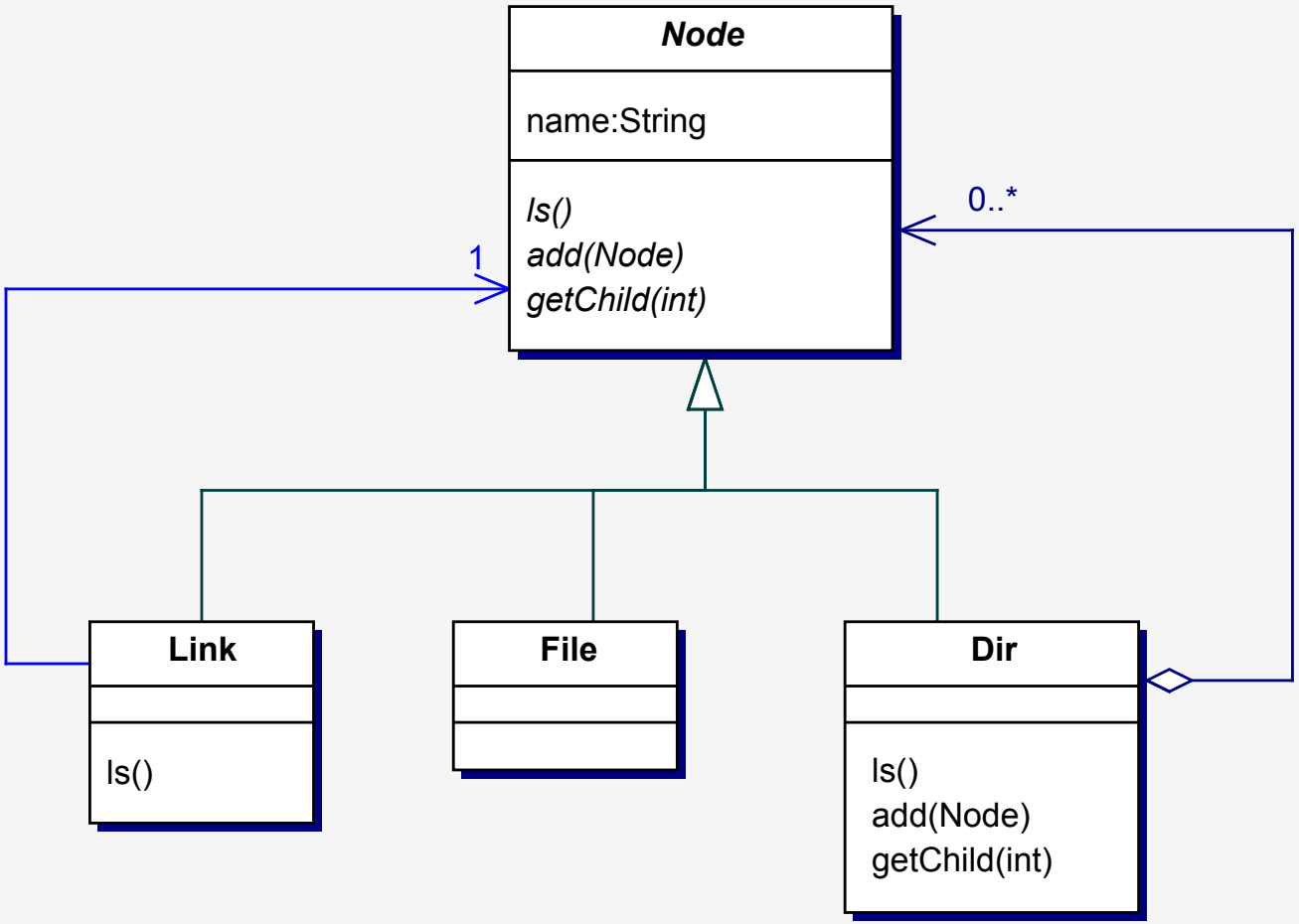
Eigenschaften

- Indirekter Zugriff auf ein Objekt durch das Proxy
- Variante: Remote Proxy = Stellvertreter für ein Objekt in einem anderen Adressraum
- Variante: Virtuelles Proxy = Stellvertreter für ein Objekt, das nur bei Bedarf wirklich erzeugt wird

Beispiele

- Remote Proxy wird in der Broker Architektur (CORBA) verwendet
- Virtuelles Proxy kann man beim Zugriff auf Datenbanken verwenden

Struktur nach Schritt 2



Code der Klasse Link

```
public class Link extends Node {

  private Node subject;

  public Link( Node node) {
    super( node.getName() );
    subject = node;
  }

  @Override
  public void ls() {
      subject.ls();
  }
}
```

Schritt 3

- Es sind in einem Dateisystem viele weitere Funktionen denkbar, die wie `ls()` den Baum traversieren und je nach Typ des Knotens eine Aufgabe erfüllen.
  Z.B. die Methode getSize(), die die Größe von Dateien und Verzeichnissen ausgibt.
- Wenn wir bei unserer bisherigen Technik bleiben, müssen wir für jede dieser Funktionen neue Methoden in unsere Klassen aufnehmen.
  Binnen kurzem haben wir eine Menge von Methoden. Geht das auch besser?
- Idee: Trennen des Traversierens des Baums von der eigentlichen Funktion. Diese Idee führt zum Muster des Visitors.

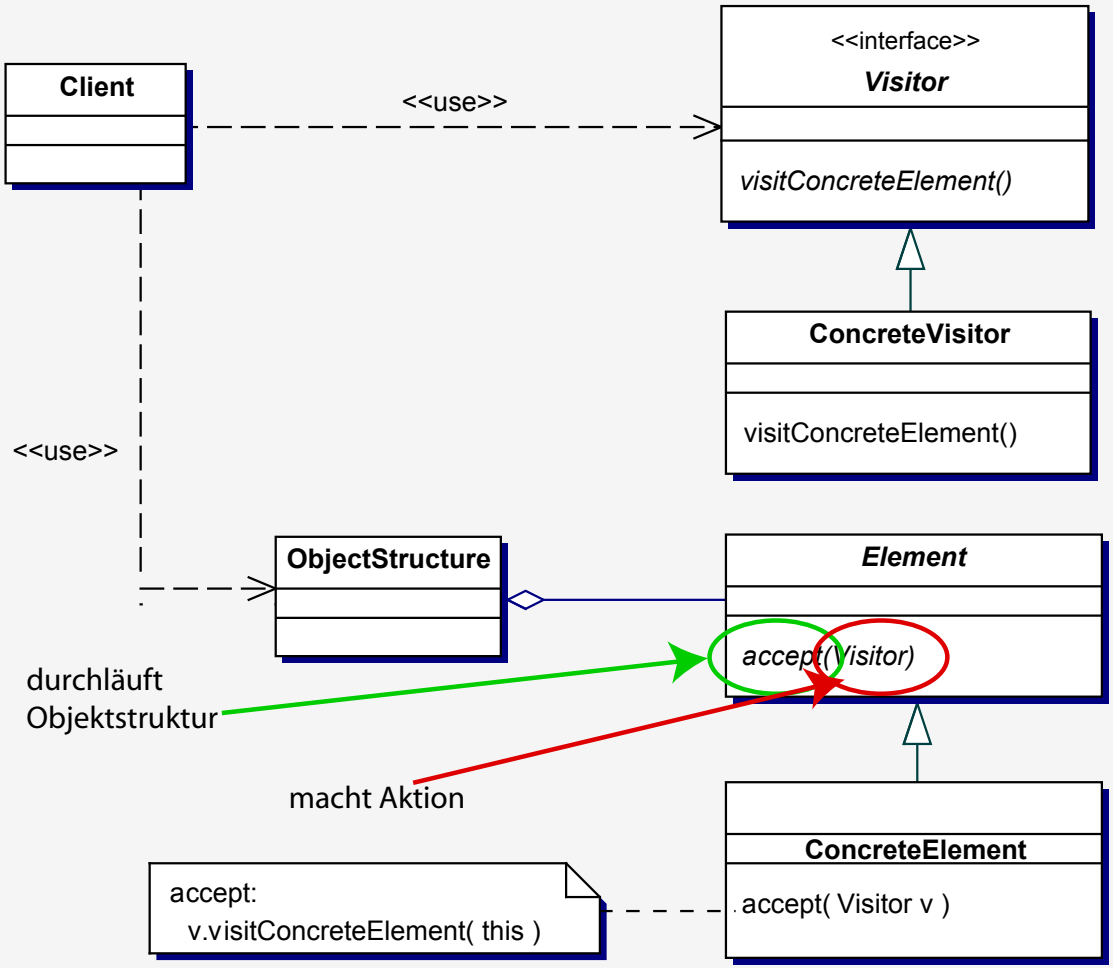
Visitor

Zweck

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. [GoF]

Visitor

Struktur



Visitor II

Eigenschaften

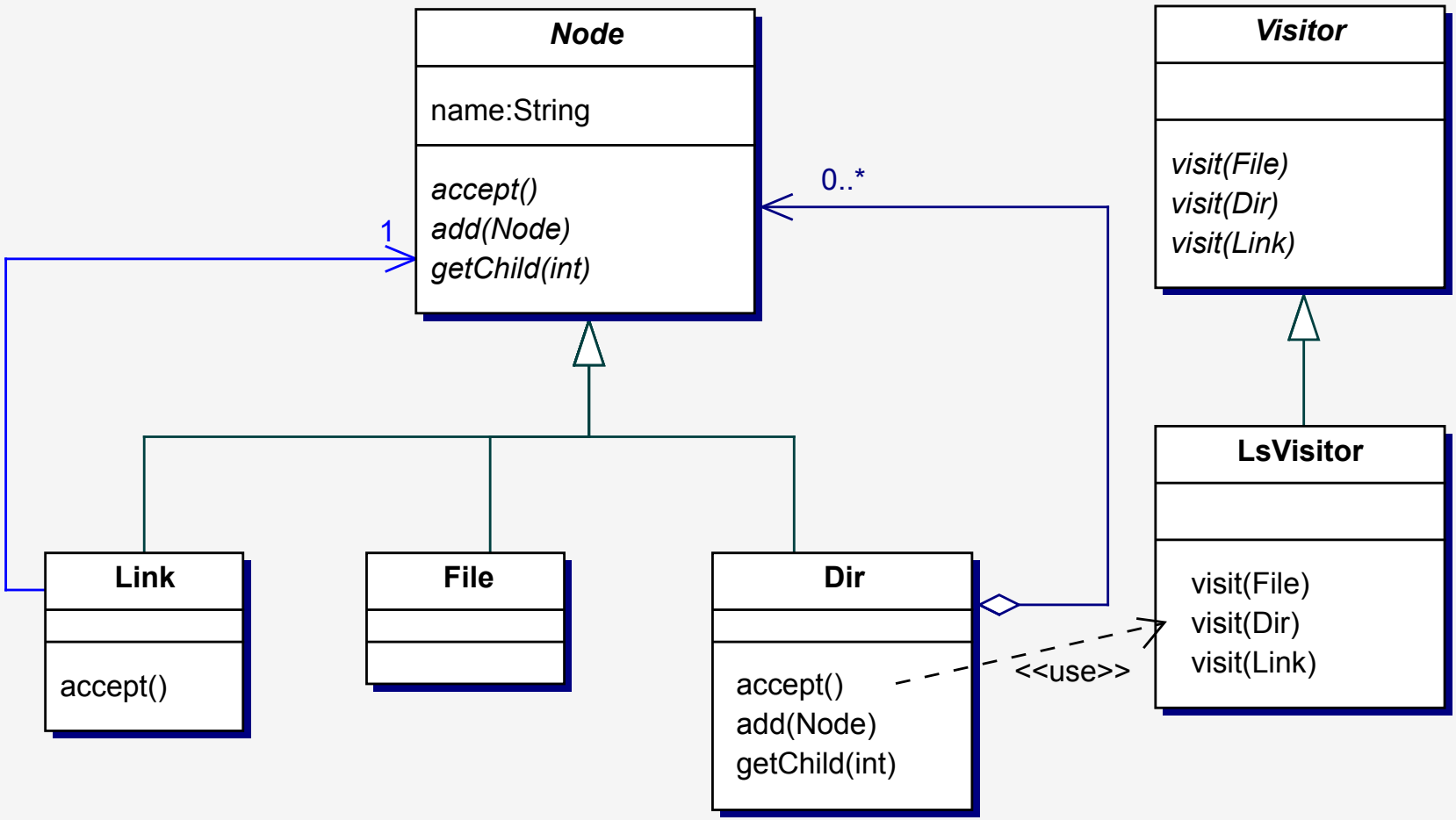
- Es ist leicht möglich, neue Operationen auf den Elementen der Objektstruktur hinzuzufügen
- Verhalten ist im Visitor konzentriert, Trennung von Struktur und Verhalten
- Iteratoren können verwendet werden, um den präzisen Aufbau der Objektstruktur vor dem Visitor zu cachieren
- Aber: das Hinzufügen neuer Klassen von Elementen ist aufwändig, weil alle Visitors angepasst werden müssen.

Visitor III

Beispiele

- Bei der Konstruktion von Compilern kann man den Visitor einsetzen, die Objektstruktur ist dann der „Parse-Baum“
- Verschiedene Toolkits (siehe [GoF]) verwenden Visitor
- Dateisystem wird für Operationen erweiterbar

Struktur nach Schritt 3



Implementierung des Visitors

Kombination von `accept` und `visit`:
in Klasse Dir

```
public void accept( Visitor v ) {
  v.visit( this );
  for ( Node n: nodes ) {
    n.accept( v );
  }
}
```

im Visitor:

```
public void visit( Dir dir ) {
  System.out.println( dir.getName() + "/" );
}
```

Schritt 4

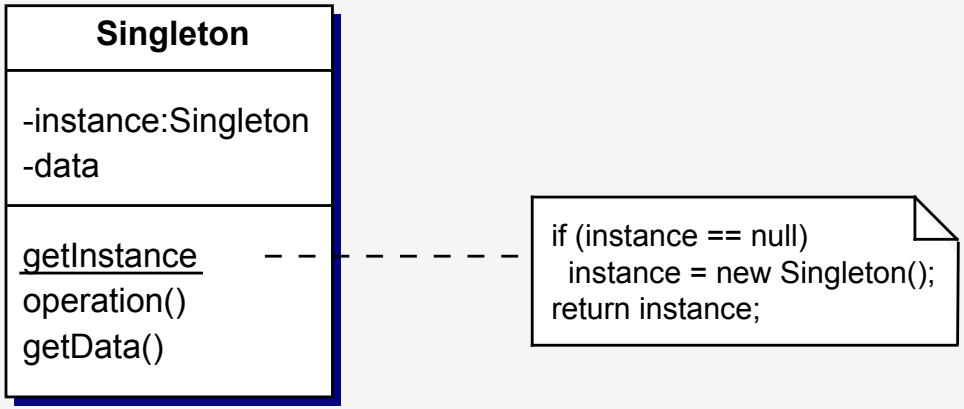
- Unser Dateisystem soll auch Benutzerberechtigungen unterstützen
- UNIX kennt `user`, `group` und `other`. Zuerst muss das System also wissen, wie ein User heißt und es muss ihn authentifizieren, damit es seine Rechte berücksichtigen kann.
- Wie nun den User modellieren? Natürlich als Objekt. Also brauchen wir eine Klasse `User`.
- Nun soll es aber zu einem User (Login) immer nur ein Objekt der Klasse `User` geben — dasjenige, das diesen User repräsentiert.
- Dafür eignet sich eine Variante des Musters Singleton.

Singleton

Zweck

Ensure a class only has one instance, and provide a global point of access to it. [GoF]

Struktur



Singleton II

Eigenschaften

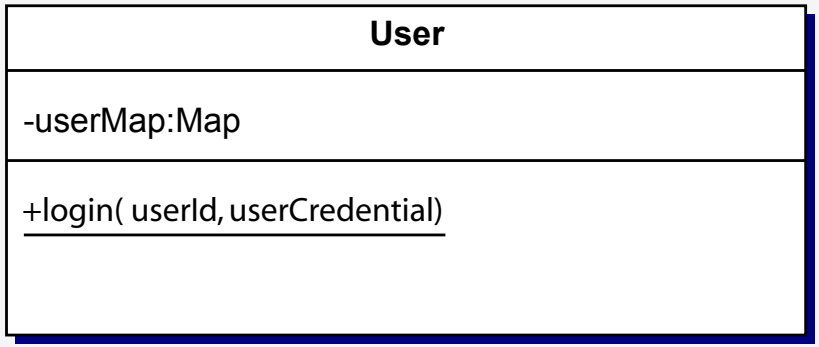
- Kontrollierter Zugriff auf genau ein Objekt der Klasse
- Variante: Zugriff auf eine bestimmte Zahl von Objekten (wie in unserem Beispiel)
- Geeignete Technik, um globale Informationen geschickt zu verwalten
- Aber: Vorsicht, wenn ein Singleton Resourcen benötigt, die wieder freigegeben werden müssen

Beispiele

- Informationen zur Konfiguration eines Programms
- Benutzerspezifische Informationen einer Anwendung (so wie in unserem Beispiel)

Struktur nach Schritt 4

Zusätzlich



Die Methode `login`

1. erzeugt ein neues User-Objekt, authentifiziert es,
2. registriert dieses Objekt in der `Map`.
3. und gibt es zurück.

Ist dies einmal passiert, wird nur noch das registrierte Objekt zurückgegeben.

Schritt 5

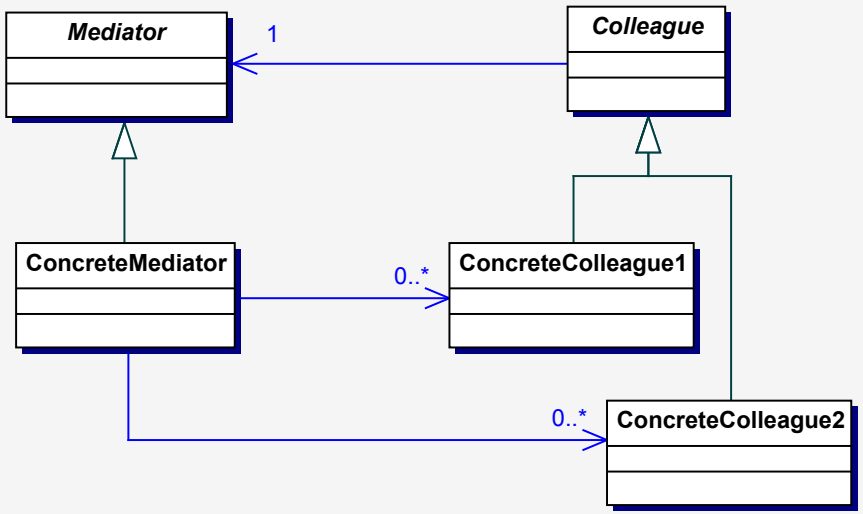
- Nun wollen wir auch noch Gruppen erlauben. Dabei gilt
  - Ein User kann Mitglied mehrerer Gruppen sein,
  - Eine Gruppe kann mehrere User haben.
- Man könnte ein gegenseitiges Mapping durch Container in User und in Group entwerfen.
  Nachteil: solche doppelte Abbildung ist schwer zu ändern.
  Stets sind beide Klassen betroffen.
- Lösung: Man zentralisiere die Verantwortung für die Zuordnung von Usern zu Gruppen in eine eigene Klasse: den Mediator.

Mediator

Zweck

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. [GoF]

Struktur



Mediator II

Eigenschaften

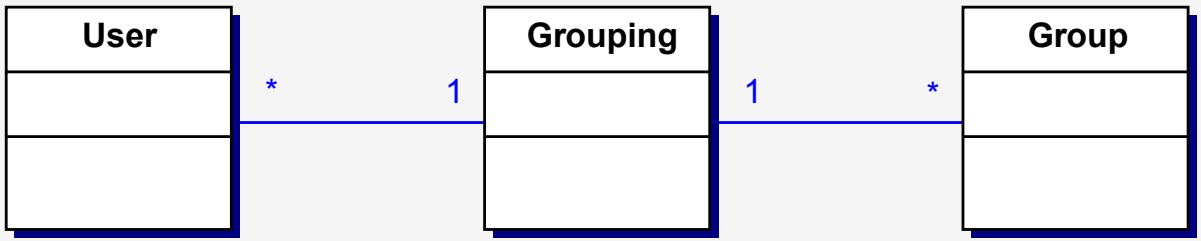
- Mediator entkoppelt Colleagues und macht sie dadurch leichter änder- und erweiterbar.
- Mediator vereinfacht Protokolle. Jeder Colleague braucht nur mit dem Mediator Informationen auszutauschen, nicht mit allen anderen Colleagues
- Mediator regelt, wie Objekte miteinander kommunizieren
- Mediator zentralisiert die Steuerung, weil alle Colleagues über ihn kommunizieren

Beispiel

- In graphischen Benutzeroberflächen: ein Mediator verwaltet alle Element eines Dialogs oder Formulars

Struktur nach Schritt 5

Zusätzlich



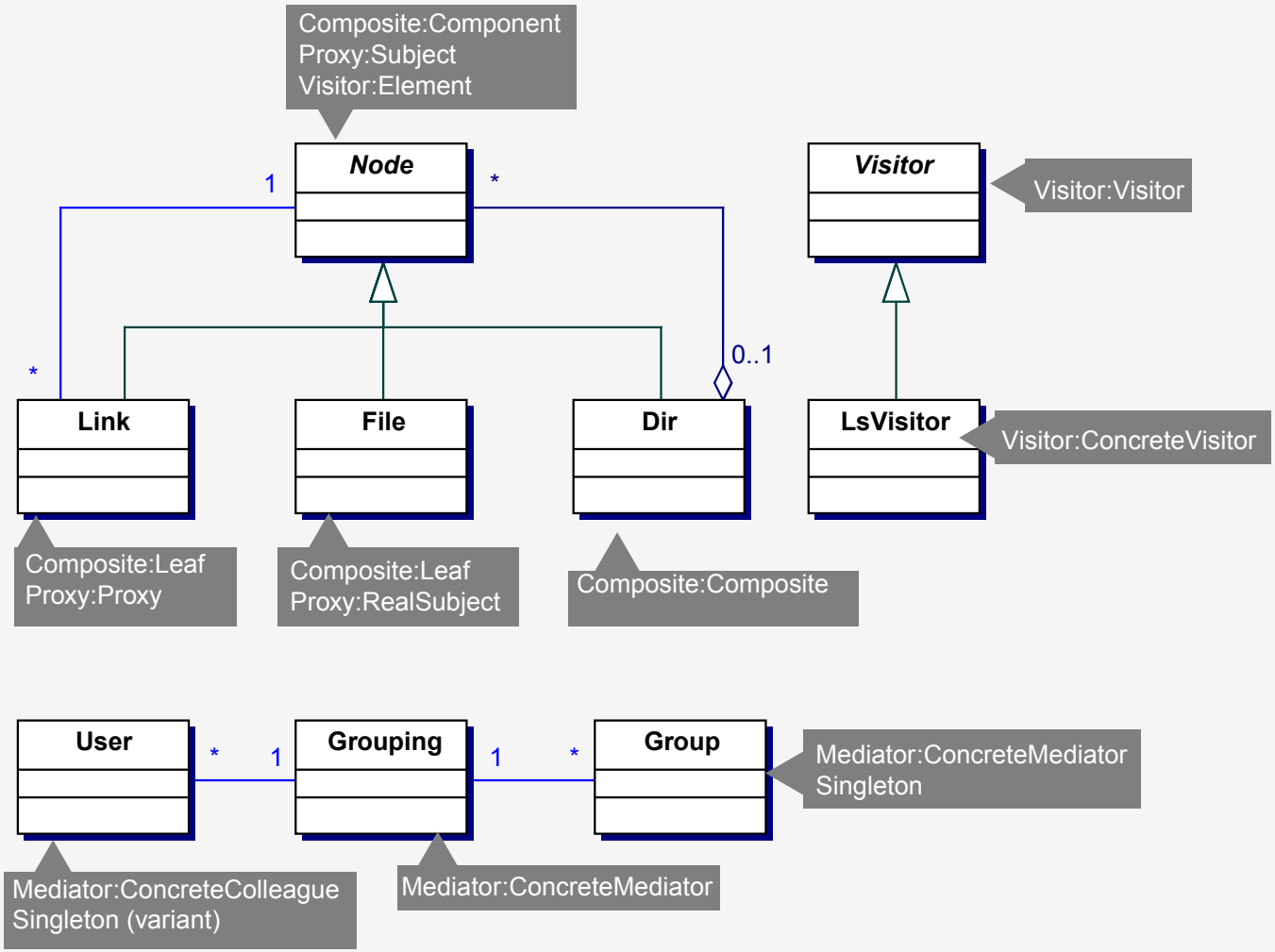
Implementierung

Die Schnittstelle von Grouping könnte so aussehen:

```
class Grouping
{
  static Grouping getGrouping();
  void register( User usr, Group grp );
  void unregister( User usr, Group grp );
  List<Group> getGroups( User usr );
  List<User> getUsers( Group grp );
}
```

Bemerkt? `Grouping` ist auch ein *Singleton*.

Das Ergebnis im Überblick



Zusammenfassung

Entwurfsmuster, erprobte Lösung typischer Probleme, im analogen Kontext intelligent angewandt, können den Entwurf stark verbessern.
Insbesondere die Qualitäten Erweiterbarkeit und Wiederverwendbarkeit profitieren vom Einsatz von Entwurfsmuster.
Wer Entwurfsmuster kennt, kann den Softwareentwurf besser beurteilen. Wir haben einige wenige Entwurfsmuster kennengelernt.
In einer Reihe von Veranstaltungen werden die Themen der Veranstaltung vertieft:

- Softwarearchitektur und Anwendungsentwicklung
- Wahlpflichtveranstaltungen
- Seminare zum Thema
- Praktikum