Übung 04

Design Patterns

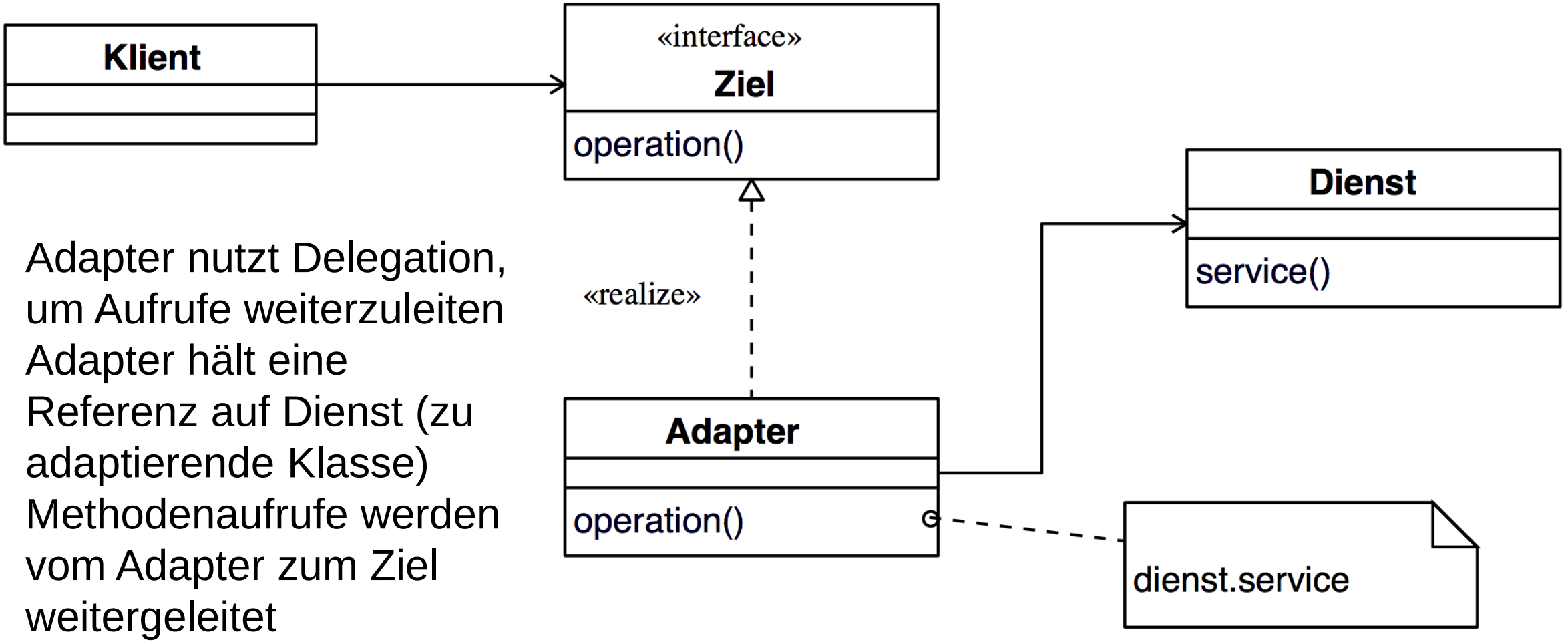
0. Organisatorisches

- 08.12.16
  - Gastvorlesung: Prof. Dr.-Ing. Thomas Leich (FH Harz, METOP AG): Global SE + SE in Praxis
- 09.12.16
  - Reguläre Übung
  - **KEIN** neues Übungsblatt
- 15.12.16
  - **KEINE** Abgabe eines Übungsblattes
- 16.12.16
  - **KEINE** Übung!
  - Übungsblatt über Weihnachten steht noch nicht fest

1. Adapter Pattern

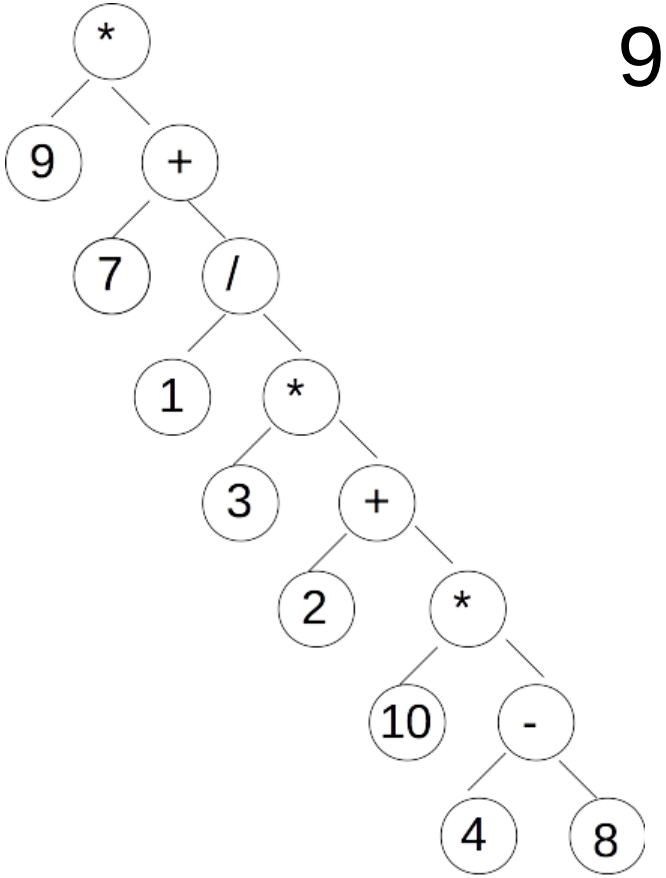
http://giphy.com/gifs/3o7TKCJwXWVFGelgw0

1. Adapter Pattern



- Adapter nutzt Delegation, um Aufrufe weiterzuleiten
- Adapter hält eine Referenz auf Dienst (zu adaptierende Klasse)
- Methodenaufrufe werden vom Adapter zum Ziel weitergeleitet

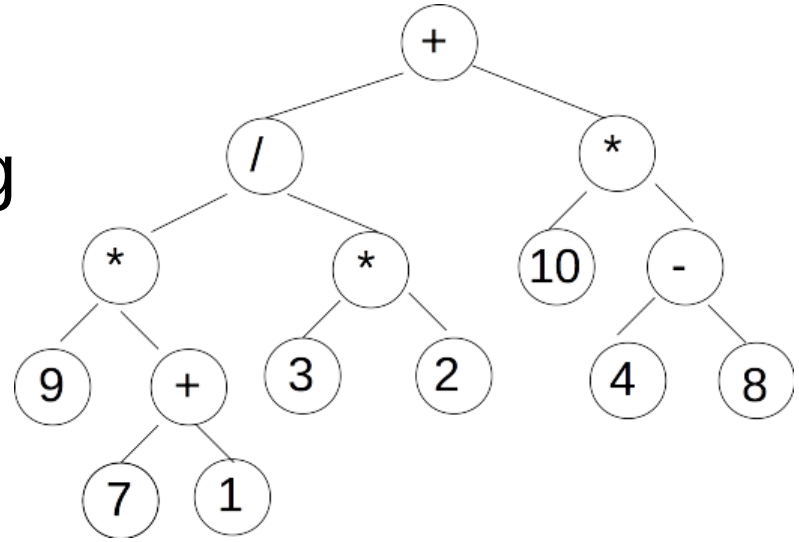
2. Visitor Pattern



9 * 7 + 1 / 3 * 2 + 10 * 4 – 8

Ist das ein Baum?

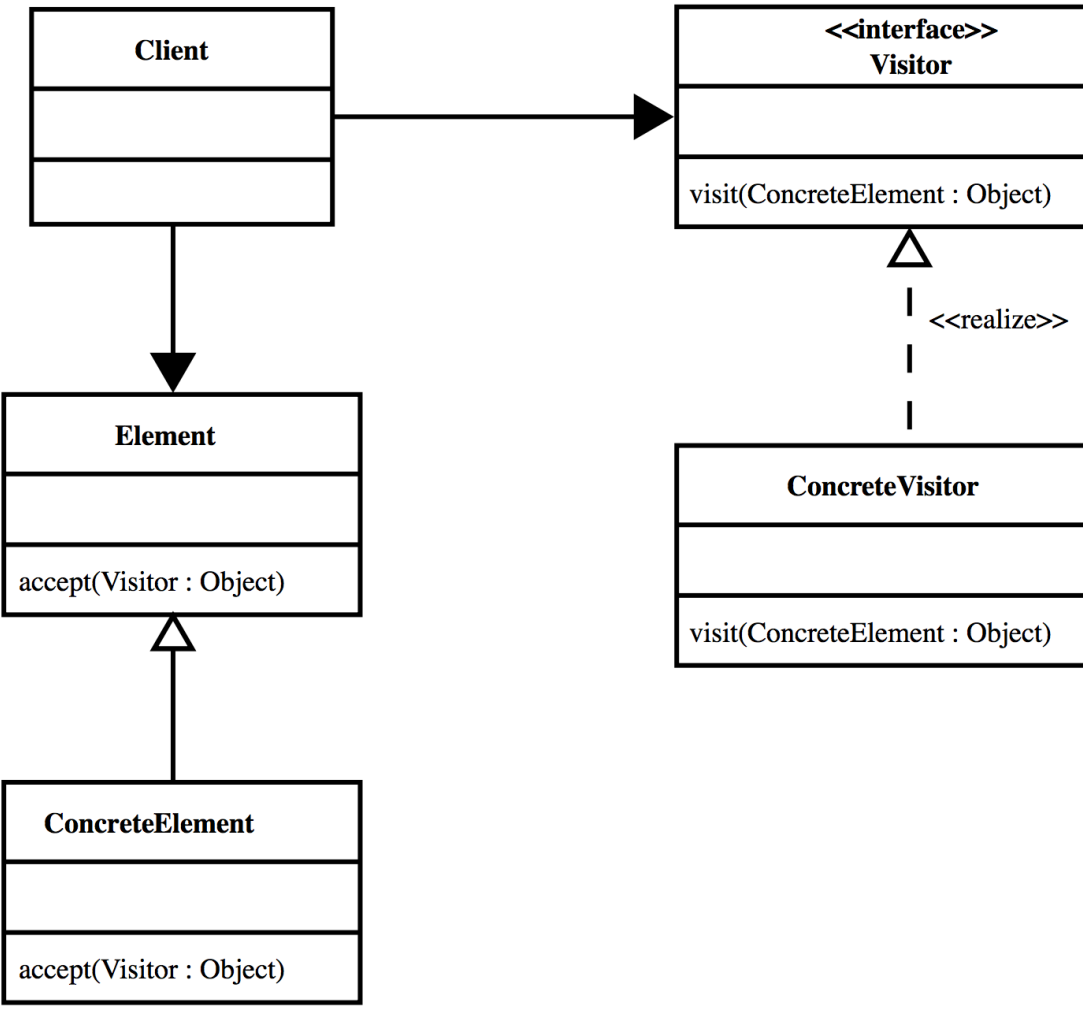
Ja, aber mehrdeutig




```
(9*(7+(1/(3*(2+(10*(4-8)))))))=56
```

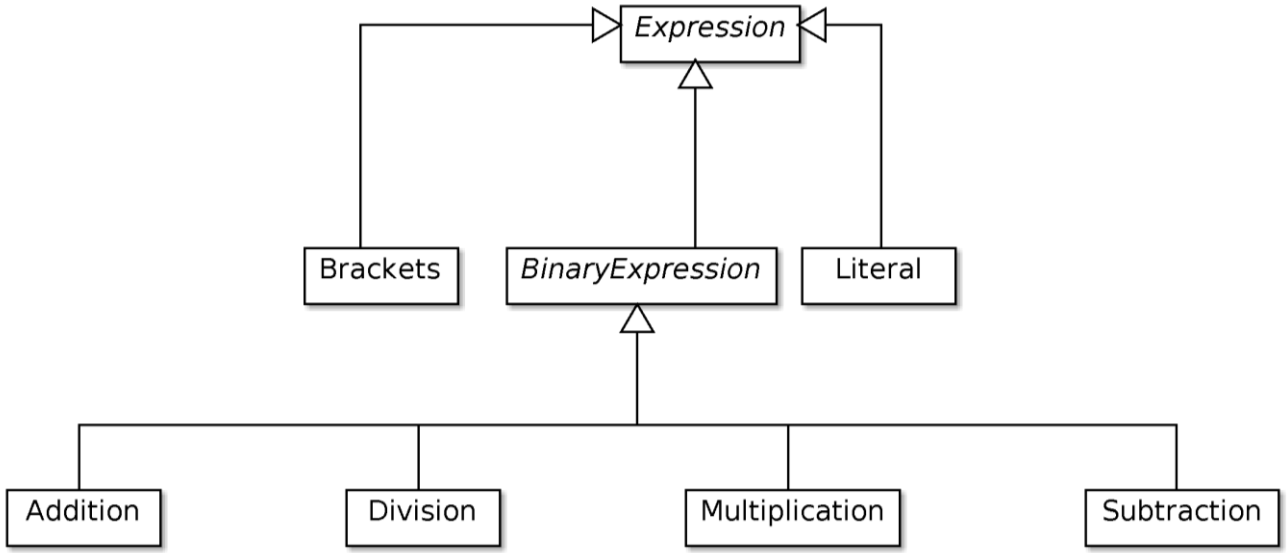
```
(((9*(7+1))/(3*2))+(10*(4-8)))=-28
```

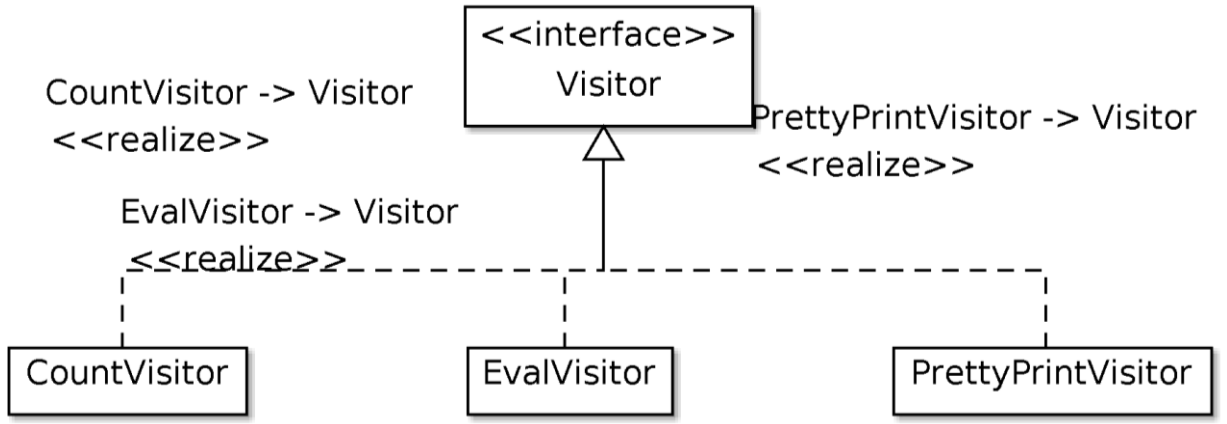
2. Visitor Pattern



- Visitor (Visitor)
  - **deklariert** eine *visit()* Methode für jedes ConcreteElement der Objekt-Struktur
  - die Methodensignatur bestimmt die Klasse die den visit-Request schickt und der Visitor kann mit dem Objekt über das Interface der Klasse kommunizieren
- ConcreteVisitor (EvalVisitor)
  - **implementiert** jede *visit()* Methode die der Visitor deklariert
  - jede Methode implementiert einen Teil des Algorithmus, eben jenen Teil der für das jeweilige Objekt der Objektstruktur vorgesehen ist
  - ConcreteVisitor kann auch Zustandinformationen halten
- Element (Node)
  - **deklariert** eine *accept()* Methode die einen Visitor als Argument nimmt
- ConcreteElement (Expression, Addition, Literal)
  - **implementiert** eine *accept()* Methode die einen Visitor als Argument nimmt

2. Visitor Pattern





✓ Visitor erlaubt einfaches Hinzufügen neuer Operationen

- neue Visitor Klasse vs. Anpassen einer Klassenstruktur

✓ Visitor separiert verwandte Operationen von nicht-verwandten

- Sowohl Visitor als auch Klassenstruktur werden einfacher verständlich

✓ Visitor kann über Klassenhierarchien hinweg operieren

- `void Visitor::visit(String s);`
- `void Visitor::visit(Interger i);`

✓ Visitor kann eigenen Zustand verwalten

- z.B. Laufzeit-Informationen, welches Objekt zuletzt besucht wurde

✗ Hinzufügen neuer ConcreteElement Klassen ist unvorteilhaft

- Jedes neue ConcreteElement benötigt i.d.R. eine neue abstrakte Methode und insbesondere eine Implementierung in **jedem** ConcreteVisitor

✗ verletzt Kapselung

- ConcreteElement muss *public* Methoden bereitstellen, sodass Visitor Zugriff hat

http://www.hwk-chemnitz.de/fileadmin/user_upload/ausbildung/Pruefungen/Gesellenpruefungen/Fragezeichen.JPG