CCD Roter Dan

III|III

“Because inheritance exposes a subclass to details of its parent's implementation, it's often said that inheritance breaks encapsulation.”

Gang of Four

- Prinzip
  - ~~Don’t Repeat Yourself~~
  - ~~Keep It Simple Stupid~~
  - ~~Vorsicht vor Optimierungen~~
  - Favour Composition over Inheritance
- Praktik
  - ~~Die Pfadfinderregel beachten~~
  - Root Cause Analysis
  - ~~Versionskontrollsystem~~
  - ~~Einfache Refaktorisierung~~
  - ~~Täglich reflektieren~~

Favour Composition over Inheritance (FCoI)

Warum?

Komposition fördert die lose Kopplung und die Testbarkeit eines Systems und ist oft flexibler.

Favour Composition over Inheritance (FCoI)

2 Konzepte in der OOP

1. *Whitebox-Reuse (Vererbung)*
2. *Blackbox-Reuse (Komposition)*

Favour Composition over Inheritance (FCoI)

Whitebox-Reuse (Vererbung):

Subklasse abhängig von Elternklasse

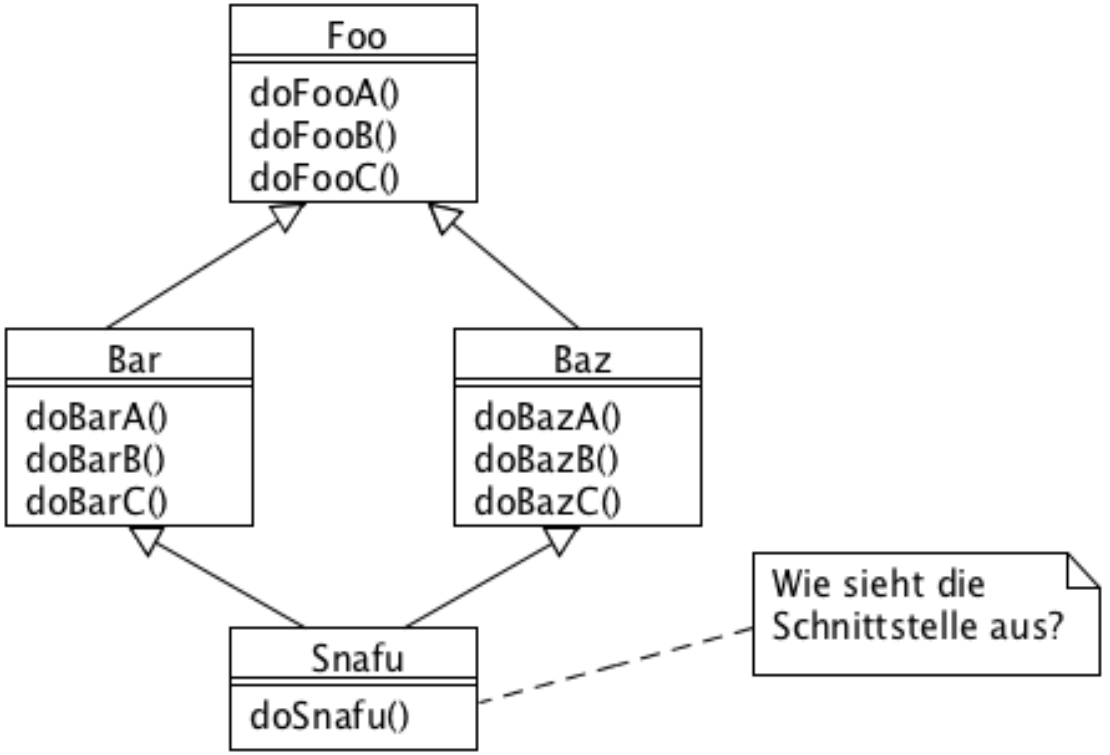
→ unnötige Komplexität (große Hierarchien, Mehrfachvererbung)

→ schlecht testbar (großer Scope, Dependencies)

→ statisch, Implementierung nicht zur Laufzeit tauschbar.

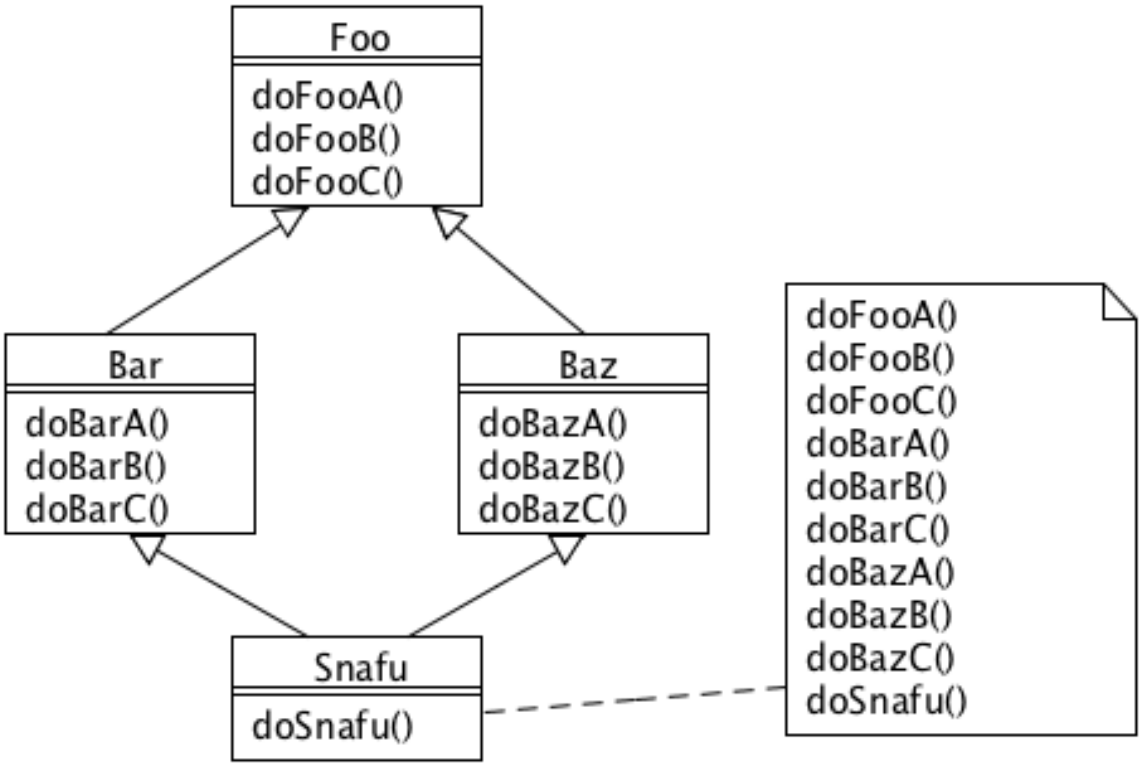
Favour Composition over Inheritance (FCoI)

Whitebox-Reuse (Vererbung):



Favour Composition over Inheritance (FCoI)

Whitebox-Reuse (Vererbung):



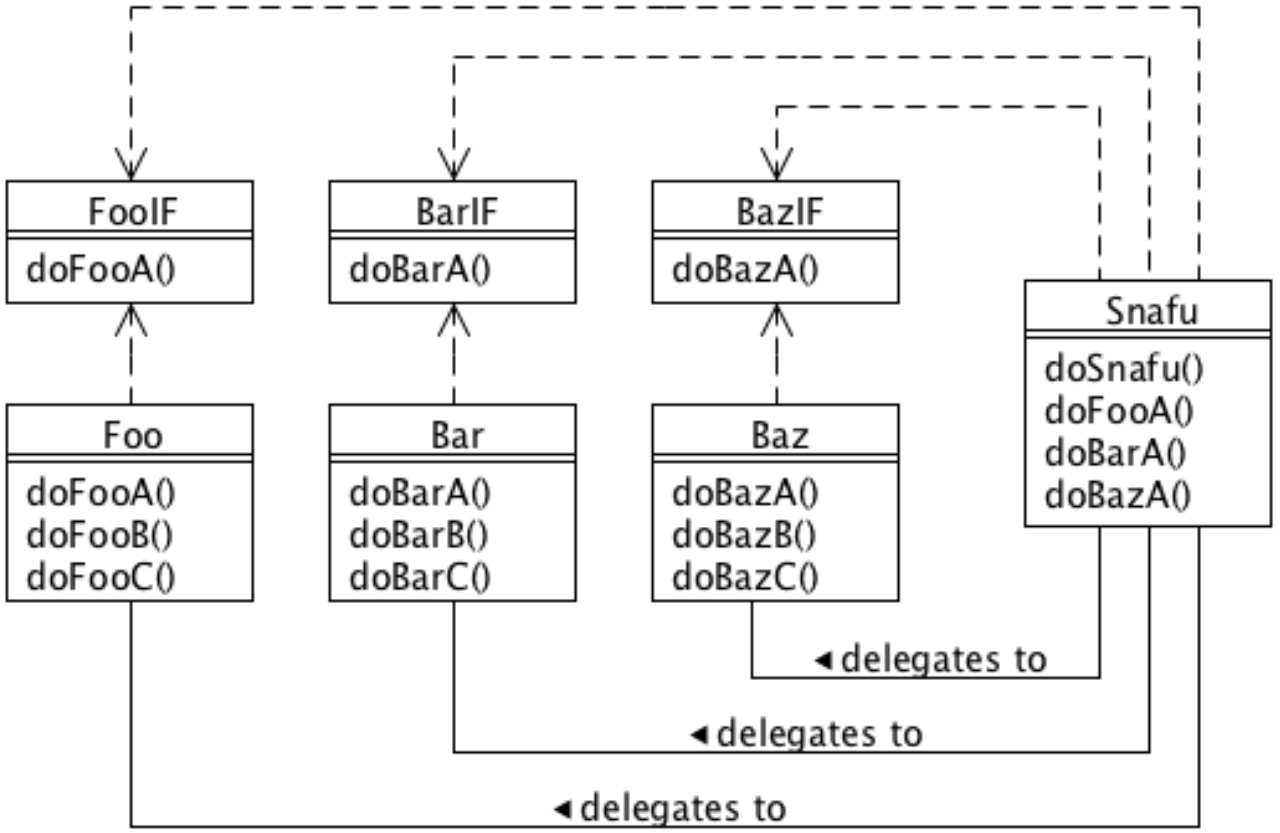
Favour Composition over Inheritance (FCoI)

Blackbox-Reuse (Komposition):

Fördert die Entkopplung, wenn man geeignete Interfaces benutzt.

Favour Composition over Inheritance (FCoI)

Blackbox-Reuse (Komposition):



Next?

Root Cause Analysis

Warum?

Symptome behandeln bringt vielleicht schnell eine Linderung – langfristig kostet es aber mehr Aufwand.

Wer stattdessen unter die Oberfläche von Problemen schaut, arbeitet am Ende effizenter.

Root Cause Analysis

Beispiel: Die Sortierung von Daten im Speicher ist zu langsam.

Oberflächlich: punktuelle Optimierungen, unsafe Code, Parallelisierung, …

Root Cause: falscher Algorithmus

Root Cause Analysis

Auch als „Five Why's“ bekannt

Stammt aus dem
Toyota Produktions System (TPS).

Die Grundidee: frage mindestens
fünf mal „Warum?“

Siehe auch Ishikawa diagram.

Root Cause Analysis

Auch Hilfsmittel in Scrum
um Grund für Impediments zu finden.

http://www.clean-code-developer.de/