Einführung in die Softwaretechnik

Vorlesung, Teil 2

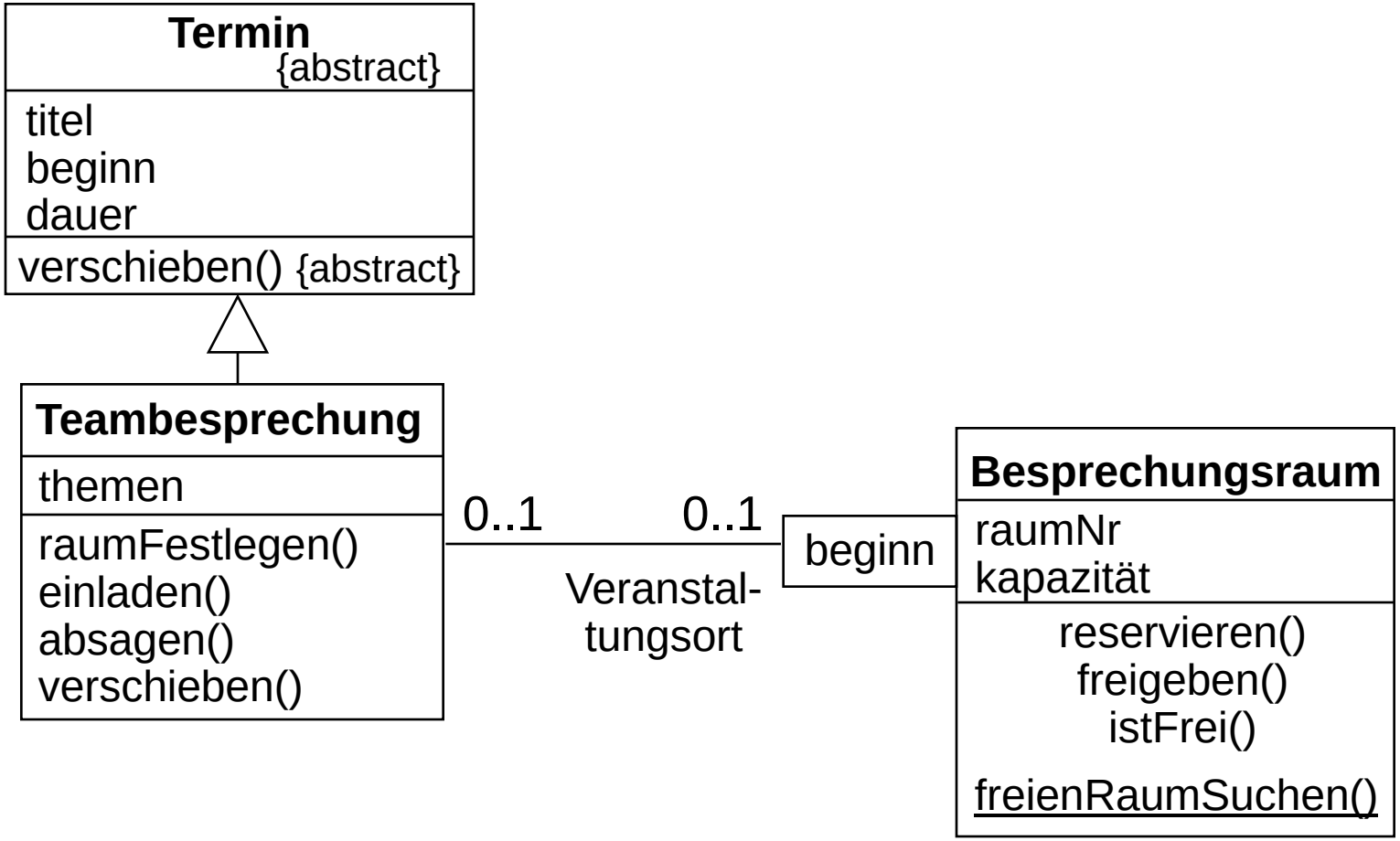
Kapitel 6. Software- und Systementwurf
-- Ergänzung: Warum sind statische Anteile schlecht?

Prof. Dr. Bernhard Rumpe
Lehrstuhl Informatik 3 (Software Engineering)
Rheinisch-Westfälische Technische Hochschule Aachen

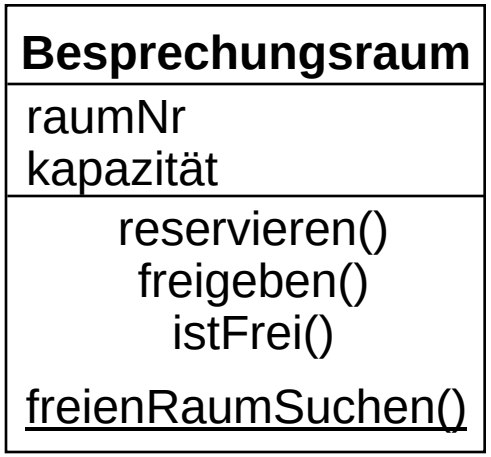
http://www.se-rwth.de/

- Frisch publiziert:
  - **Top 25 Most Dangerous Programming Errors**
  - http://cwe.mitre.org/top25/
  - Die ersten drei:
  - CWE-20: Improper Input Validation
    CWE-116: Improper Encoding or Escaping of Output
    CWE-89: Failure to Preserve SQL Query Structure (aka 'SQL Injection')

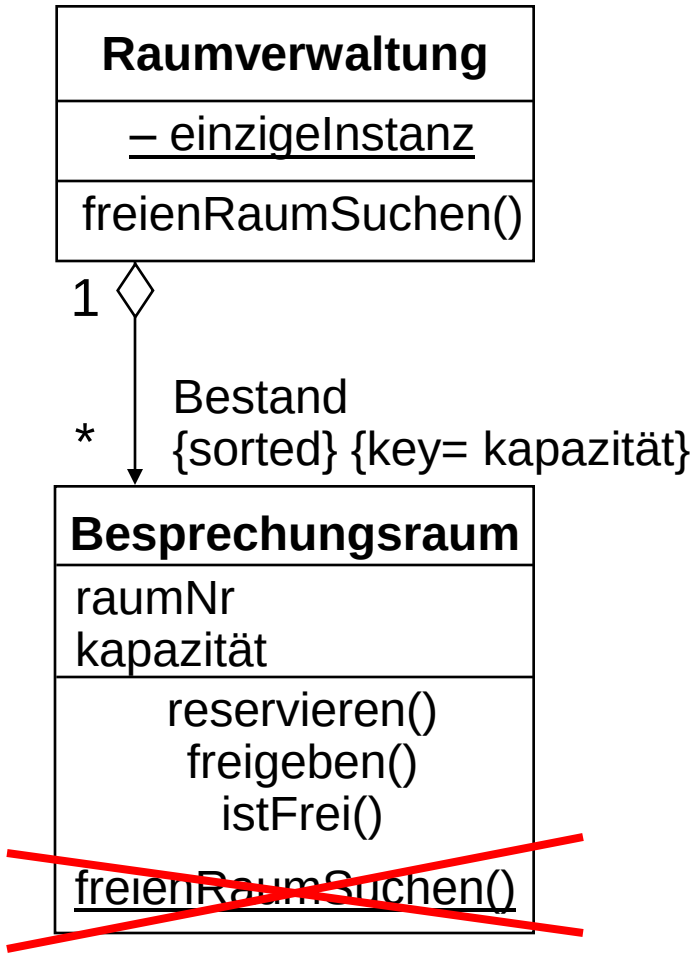
Verwaltungsklassen



Entwurfsentscheidung: statische Methode bleibt. Realisierung?



Entwurfsentscheidung: Verwalterklasse.

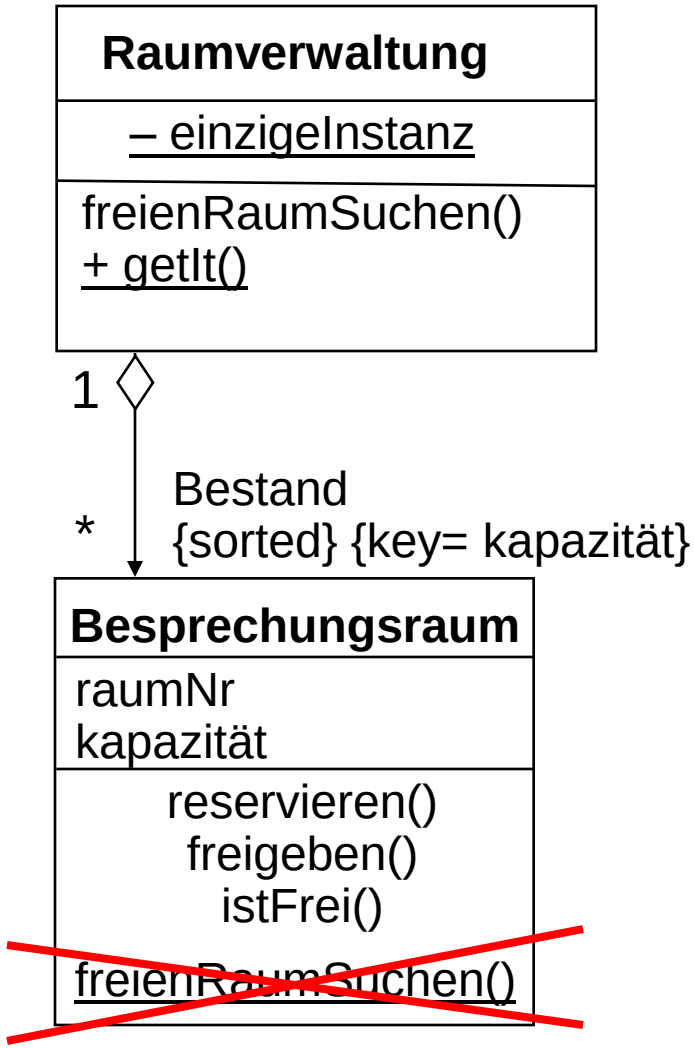


Entwurfsentscheidung: Verwalterklasse. Noch etwas genauer …

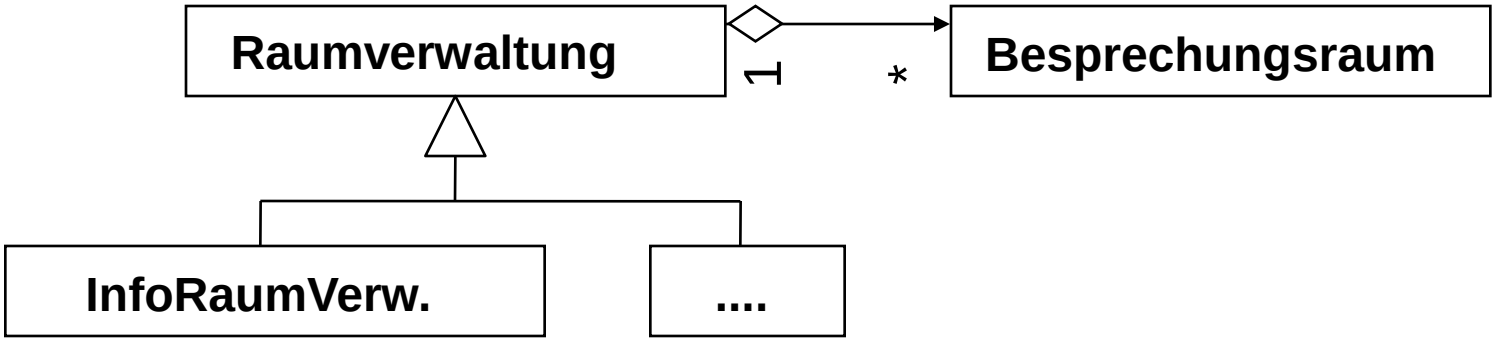
```
getIt() {
    return einzigeInstanz;
}

<<constructor>>
Raumverwaltung() {
    einzigeInstanz = this;
}

<<constructor>>
Raumverwaltung(Raumverwaltung r) {
    einzigeInstanz = r;
}
```

Aufruf:

```
getIt().freienRaumSuchen(…)
```

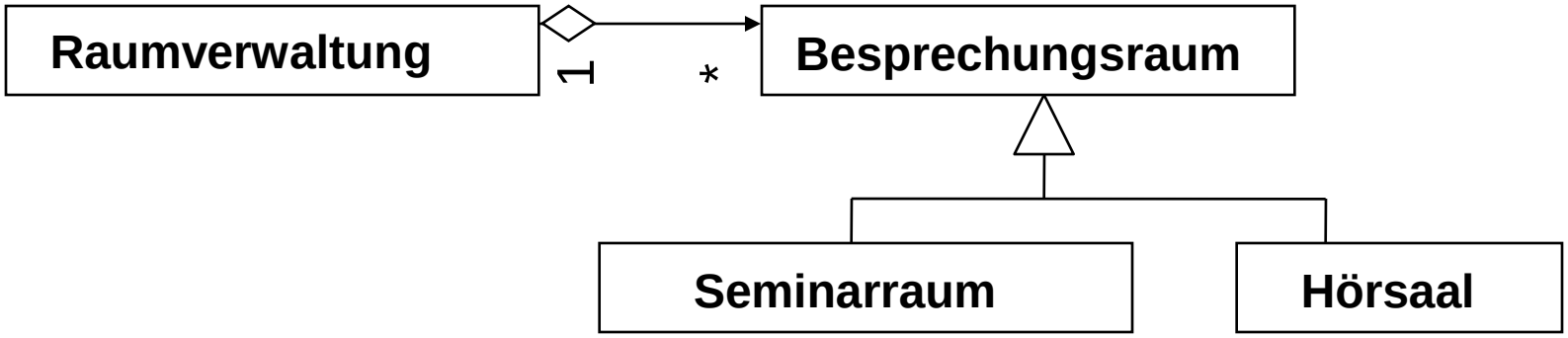


Vorteil 1: Flexibilität bei der Raumverwaltung




```
<<constructor>>
InfoRaumVerw() {
    super(this);
}

// eigene Realisierung von freienRaumSuchen()
```

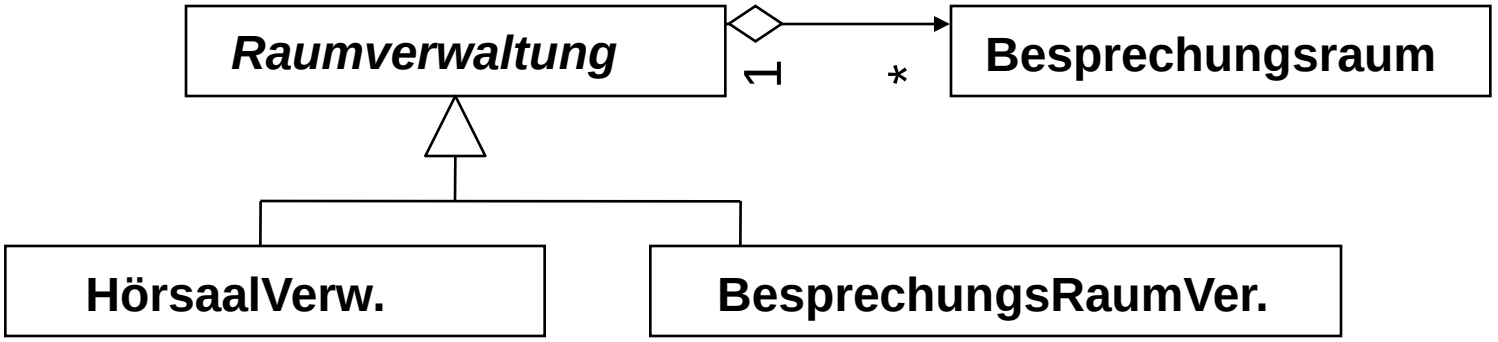
Vorteil 2: Flexibilität bei den Räumen



Separation of Concerns:

Neue Arten von Besprechungsräumen sind leichter hinzuzufügen

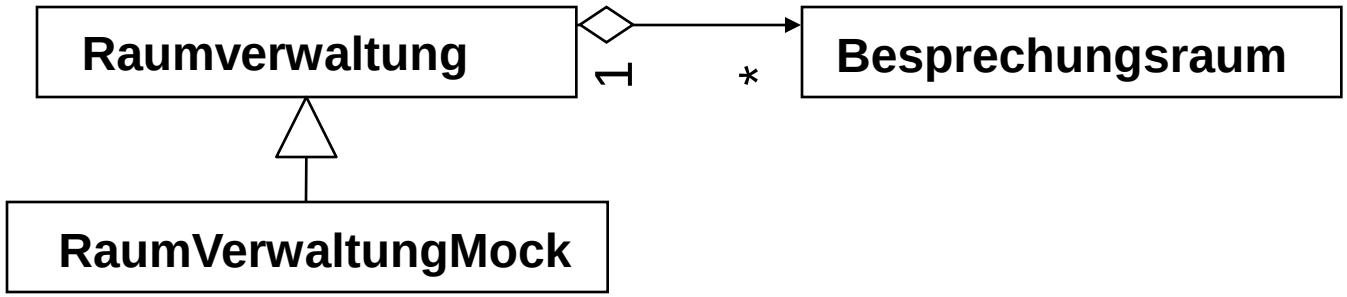
Vorteil 3: Parallelität (Threads) leichter hanhabbar



Annahme: pro Gebäude eine Raumverwaltung, die technisch in je einem Thread realisiert wird.
→Wir geben den „einzige Instanz“-Ansatz auf

- Erlaubt mehrere Subklassen von Raumverwaltung
- ggf. mehrere Instanzen jeder Subklasse
- Abschottung paralleler Vorgänge in jeweils eigenen Instanzen (kein zentrales „Bottleneck“ einer einzelnen Instanz, aber Aufwand jeweils die richtige Raumverwaltung zu kennen.)
- Bei hoher Rechnerbelastung auch leicht auf mehrere Prozessoren verteilbar.

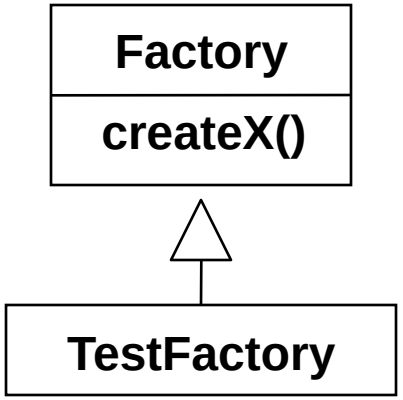
Vorteil 4: testfähiges System



Annahme: wir wollen den Nutzer der Raumverwaltung testen.
Raumverwaltung hat aber Seiteneffekte (z.B. auf Datenbank)
→Mock-Objekt durch Subklasse

```
<<constructor>>
RaumverwaltungMock() {
    super(this);
}

freienRaumSuchen(…) {
    // geeigneter Testcode ohne Nebenwirkungen
}
```

Wie werden jeweils die richtigen Objekte erzeugt?

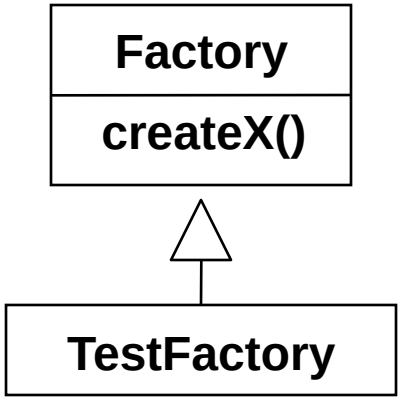


Objekterzeugung nie durch „new Constructor()“ sondern in Factory auslagern:

createX() { return new X(); }
oder z.B. wenn Singleton:
createX() { if(einzigesX==null) { einzigesX = new X(); return einzigesX; }

und in der MockFactory:
createX() { return new XMock(); }

Wie werden jeweils die richtigen Objekte erzeugt?



Objekterzeugung nie durch „new Constructor()“ sondern in Factory auslagern:

createX() { return new X(); }
oder z.B. wenn Singleton:
createX() { if(einzigesX==null) { einzigesX = new X(); return einzigesX; }

und in der MockFactory:
createX() { return new XMock(); }

→ Bleibt das Erzeugen der richtigen Factory beim Start
(also der main()-Methode oder beim Test-Setup.)

Factories und Singletons sind effektive Entwurfsmuster (siehe dort)