Universität Bremen

Praktische Informatik 1
Klassenentwurf

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

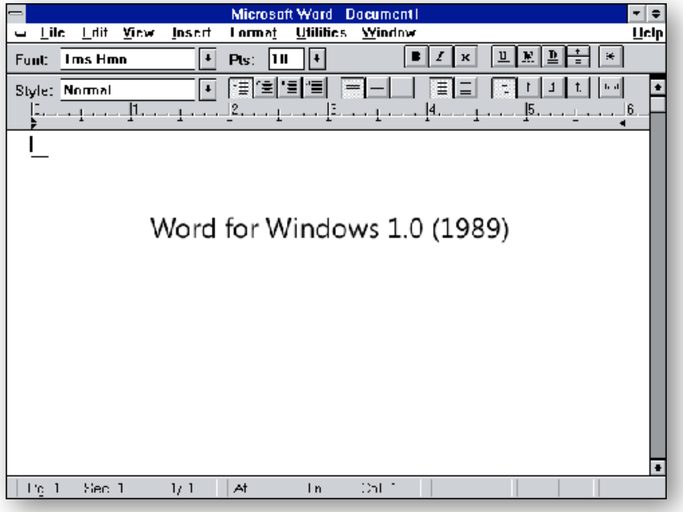
Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

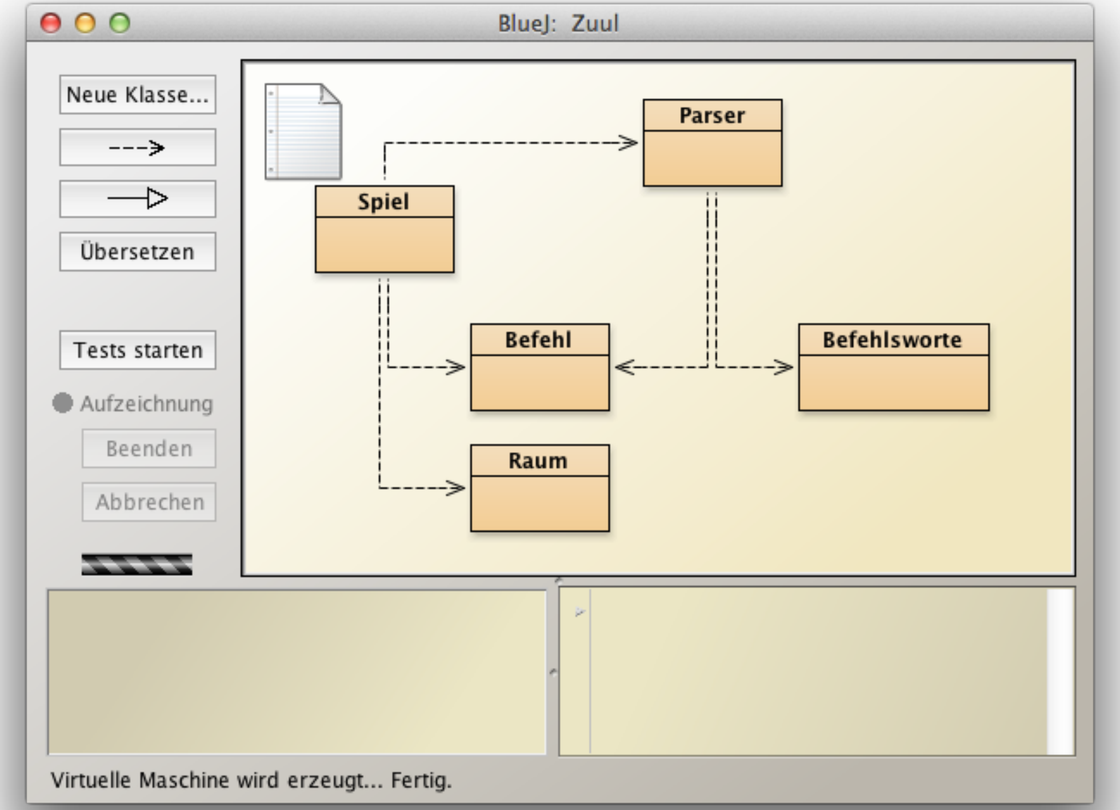
Motivation

- Eine Anwendung kann trotz schlechtem Klassenentwurf implementiert werden und ihre Aufgabe erfüllen
- Probleme können auftreten, wenn nachträglich Änderungen oder Erweiterungen gemacht werden müssen
- Bei großen Anwendungen auch schon bei der Erstimplementierung
- Software wird oft lange weiterentwickelt und gepflegt
  - TeX (1977), Word for Windows (1989), Java (1995)



Universität Bremen

Die Welt von Zuul: Demo



Universität Bremen

Kopplung und Kohäsion

- **Kopplung**: Grad der Abhängigkeiten zwischen Klassen
  - Sollte möglichst lose sein
  - Klassen kommunizieren nur mit wenigen anderen Klassen über möglichst schmale, wohl definierte Schnittstellen
- **Kohäsion**: Wie gut bildet eine Programmeinheit (Methode, Klasse, Paket) eine logische Aufgabe oder Einheit ab?
  - Bei hoher Kohäsion ist eine Programmeinheit für genau eine wohl definierte Aufgabe oder Einheit zuständig

Universität Bremen

Code-Duplizierung

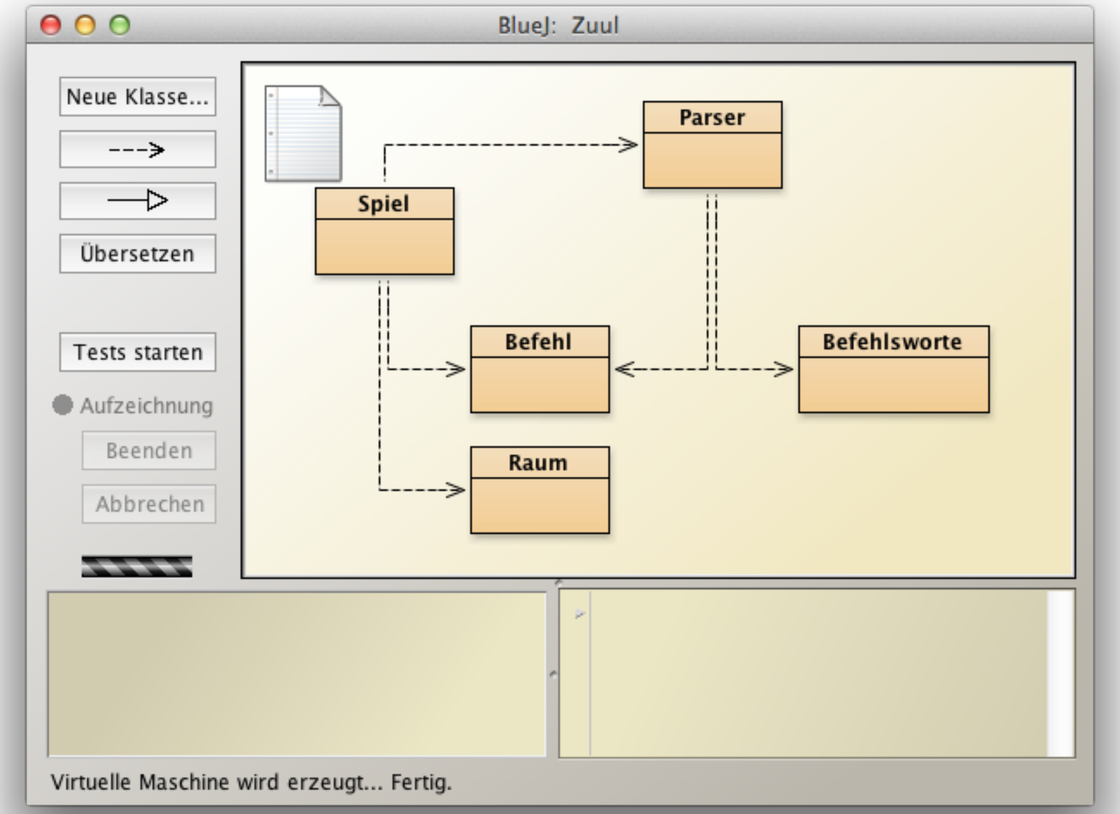
- Ein Quelltextabschnitt erscheint mehr als einmal in einer Anwendung
- Alle zukünftigen Änderungen müssen an allen Kopien durchgeführt werden
  - Insbesondere auch Fehlerkorrekturen

```
private void willkommenstextAusgeben()
{
                    :
    System.out.print("Ausgänge: ");
    if (aktuellerRaum.nordausgang != null) {
        System.out.print("north ");
    }
                    :
```

```
private void wechselRaum(Befehl befehl)
{
                    :
    System.out.print("Ausgänge: ");
    if (aktuellerRaum.nordausgang != null) {
        System.out.print("north ");
    }
                    :
```

Universität Bremen

Code-Duplizierung entfernen: Demo



Universität Bremen

Kapselung

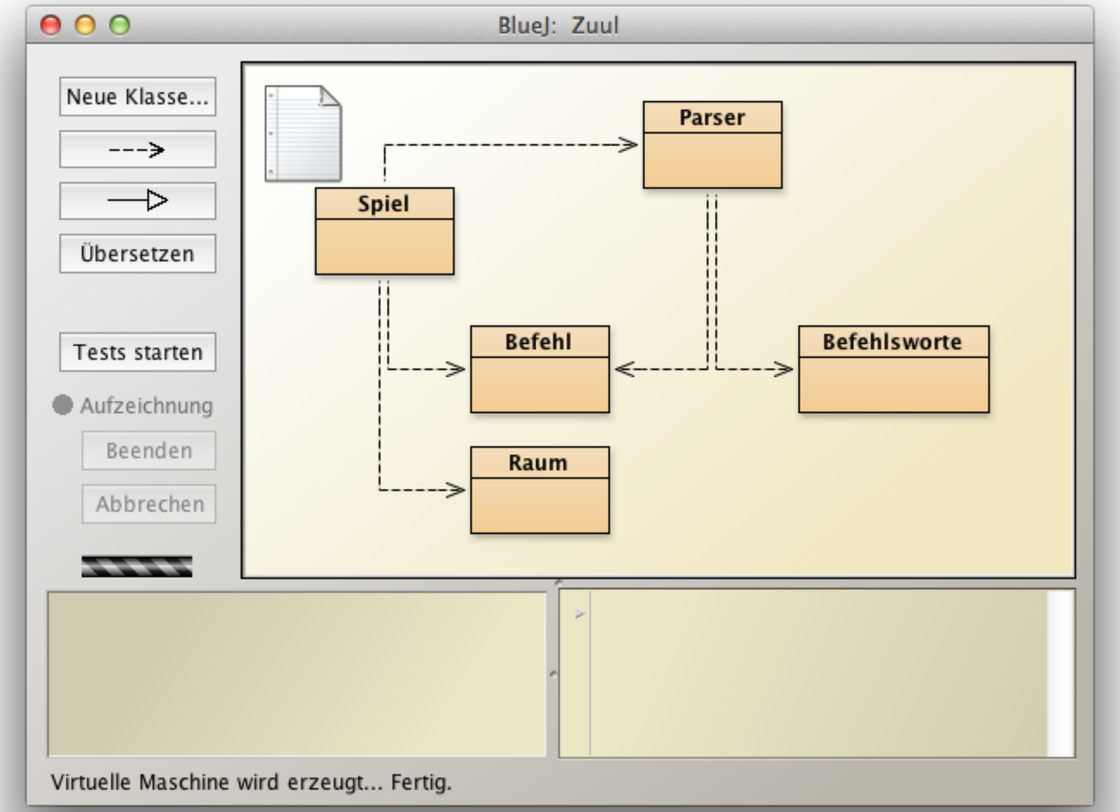
- **Kapselung**: Mache das **Was** öffentlich (Funktionalität), aber nicht das **Wie** (Realisierung)
- Saubere **Kapselung** reduziert die **Kopplung**
- **Vorteil**: Implementierung einer Klasse kann geändert werden, ohne dass andere Klassen beeinträchtigt werden (**Änderungen lokal halten**)
- Attribute sollten **private** sein

```
private Raum nordausgang;
private Raum südausgang;
private Raum ostausgang;
private Raum westausgang;
```

```
private final Map<String, Raum> ausgänge;
```

Universität Bremen

Kapselung verbessern: Demo



Universität Bremen

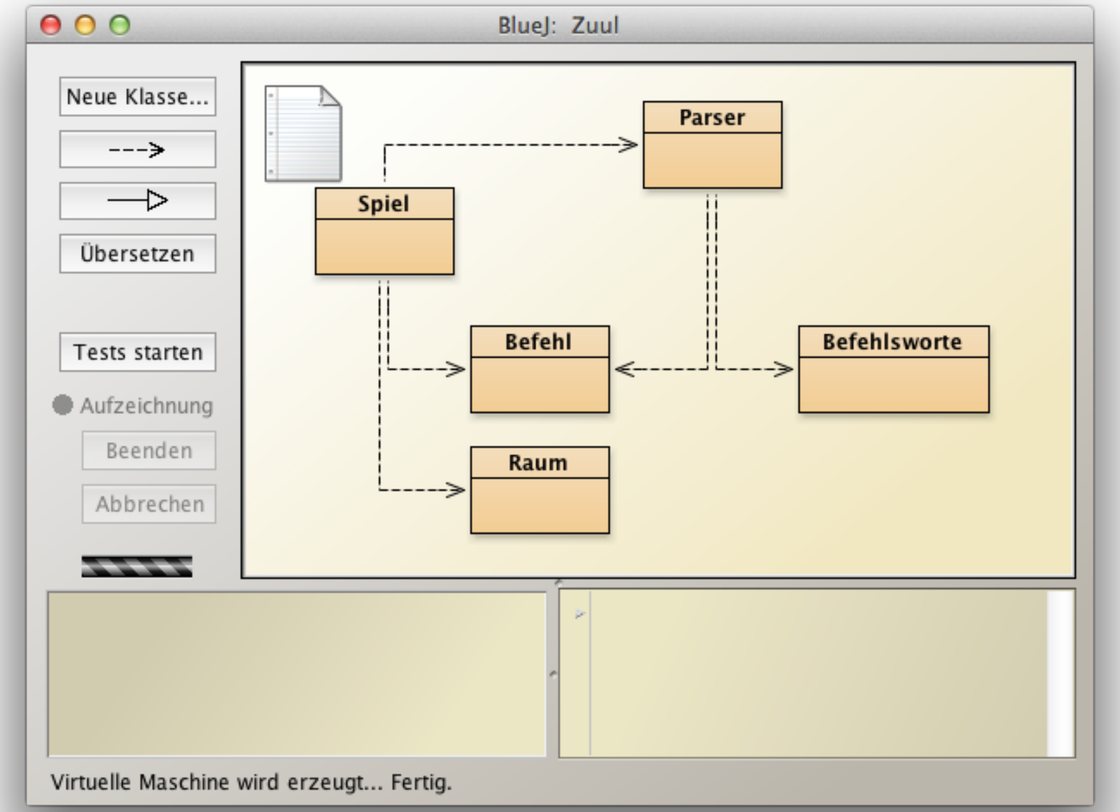
Entwurf nach Zuständigkeiten

- Entwurfsprozess, bei dem jeder Klasse eine klare Verantwortung zugewiesen wird
- Welche Klasse ist für welche Funktionen der Anwendung zuständig?

```
class Spiel
{
                    :
    private void rauminfoAusgeben()
                    :
```

```
class Raum
{
                    :
    String gibLangeBeschreibung()
    {
        return "Sie sind " + beschreibung +
                ".\n" + gibAusgängeAlsString();
    }
    private String gibAusgängeAlsString()
                    :
```

Kapselung nutzen : Demo



Universität Bremen

Implizite Kopplung

- Eine Klasse ist von Implementierungsdetails einer anderen Klasse abhängig, ohne dass dies offensichtlich ist

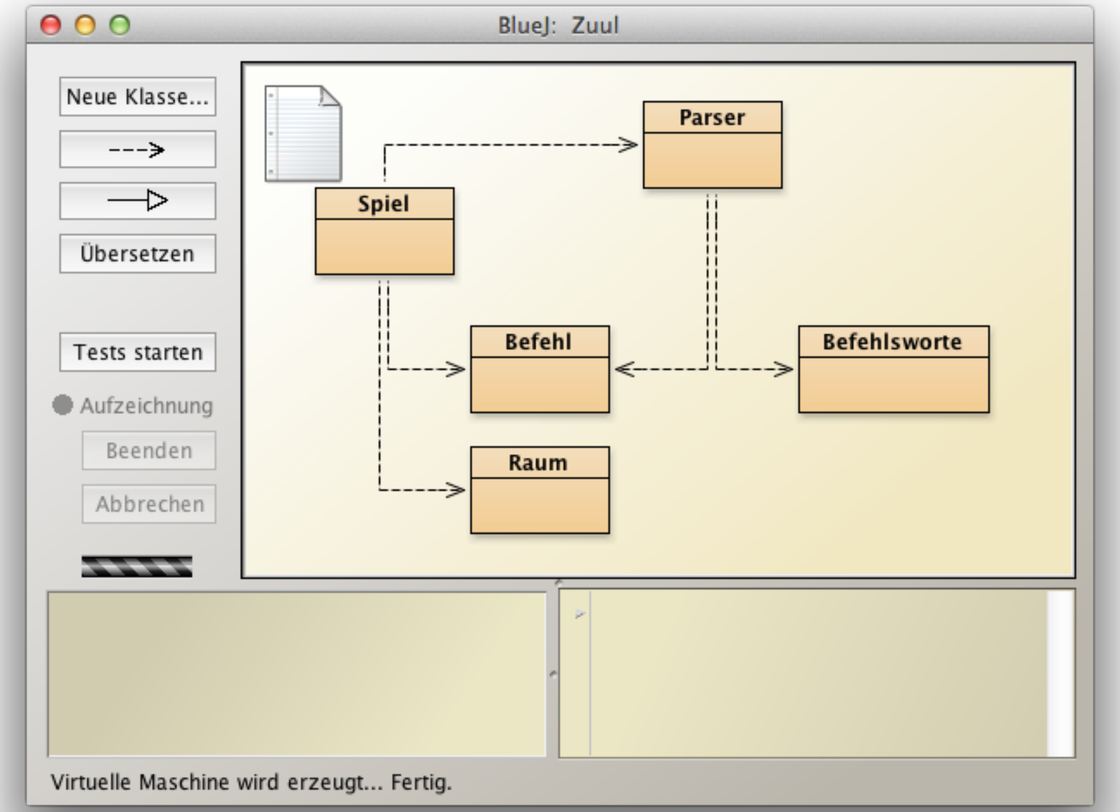
- Klasse kann geändert werden, ohne dass der Compiler auf notwendige Anpassungen an anderen Klassen hinweist

```
class Spiel
{
    private void hilfstextAusgeben()
    {
                        :
        System.out.println("    go quit help");
                        :
```

```
class Befehlsworte
{
    private static String gültigeBefehle[ ] = {
        "go", "quit", "help"
    };
```

Universität Bremen

Refactoring: Demo



Universität Bremen

Kohäsion

- **Von Methoden**: Methode von hohe Kohäsion ist verantwortlich für genau eine wohl definierte Aufgabe
- **Von Klassen**: Klasse hoher Kohäsion repräsentiert genau eine wohl definierte Einheit
- **Für bessere Lesbarkeit**: Methoden und Klassen hoher Kohäsion bündeln Funktionalität an einer Stelle, wodurch das Programm besser lesbar ist
- **Für Wiederverwendung**: Methoden und Klassen hoher Kohäsion können eher an mehreren Stellen eingesetzt werden

Universität Bremen

Vorausdenken

- Beim Entwurf bereits bedenken
  - Was könnte sich ändern?
  - Welche Teile werden vermutlich während der Lebenszeit des Programms unverändert bleiben?
- Z.B.: Trennung der Programmlogik von der Benutzerinteraktion

```
void alleAusgeben()
{
    for (final String befehl : gültigeBefehle) {
        System.out.print(befehl + "  ");
    }
    System.out.println();
}
```

vs.

```
String gibBefehlsliste()
{
    String liste = "";
    for (final String befehl : gültigeBefehle) {
        liste += befehl + "  ";
    }
    return liste.trim();
}
```

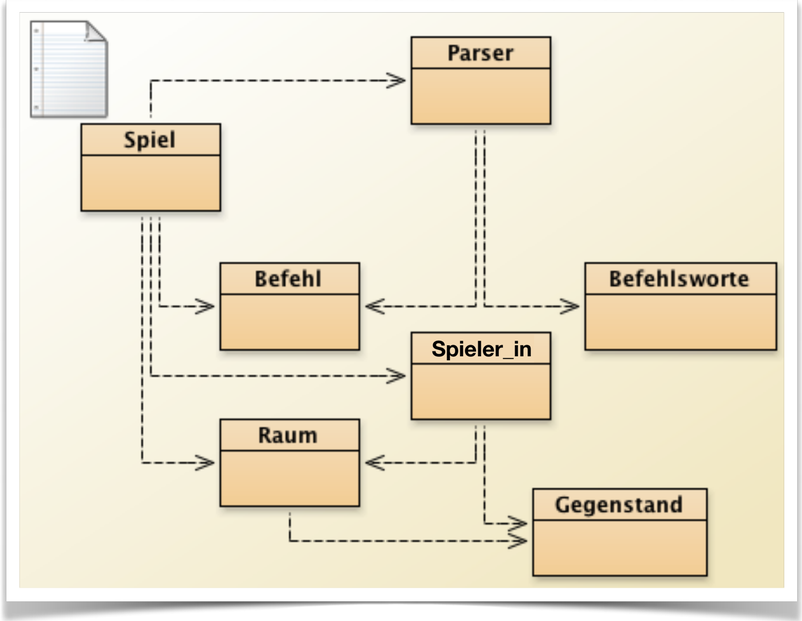
Refactoring

- Restrukturierung des bestehenden Entwurfs, um die Qualität des Klassenentwurfs in Hinsicht auf vorzunehmende Änderungen und Erweiterungen zu erhalten
- Zerlegen (häufig) oder Zusammenfassen (seltener) von Klassen oder Methoden
- Umstrukturierungen vornehmen
- Überprüfen, dass Funktionalität immer noch identisch
  → **Regressionstests**
- Erst dann Erweiterungen vornehmen

```
Raum nordausgang;
Raum südausgang;
Raum ostausgang;
Raum westausgang;
```

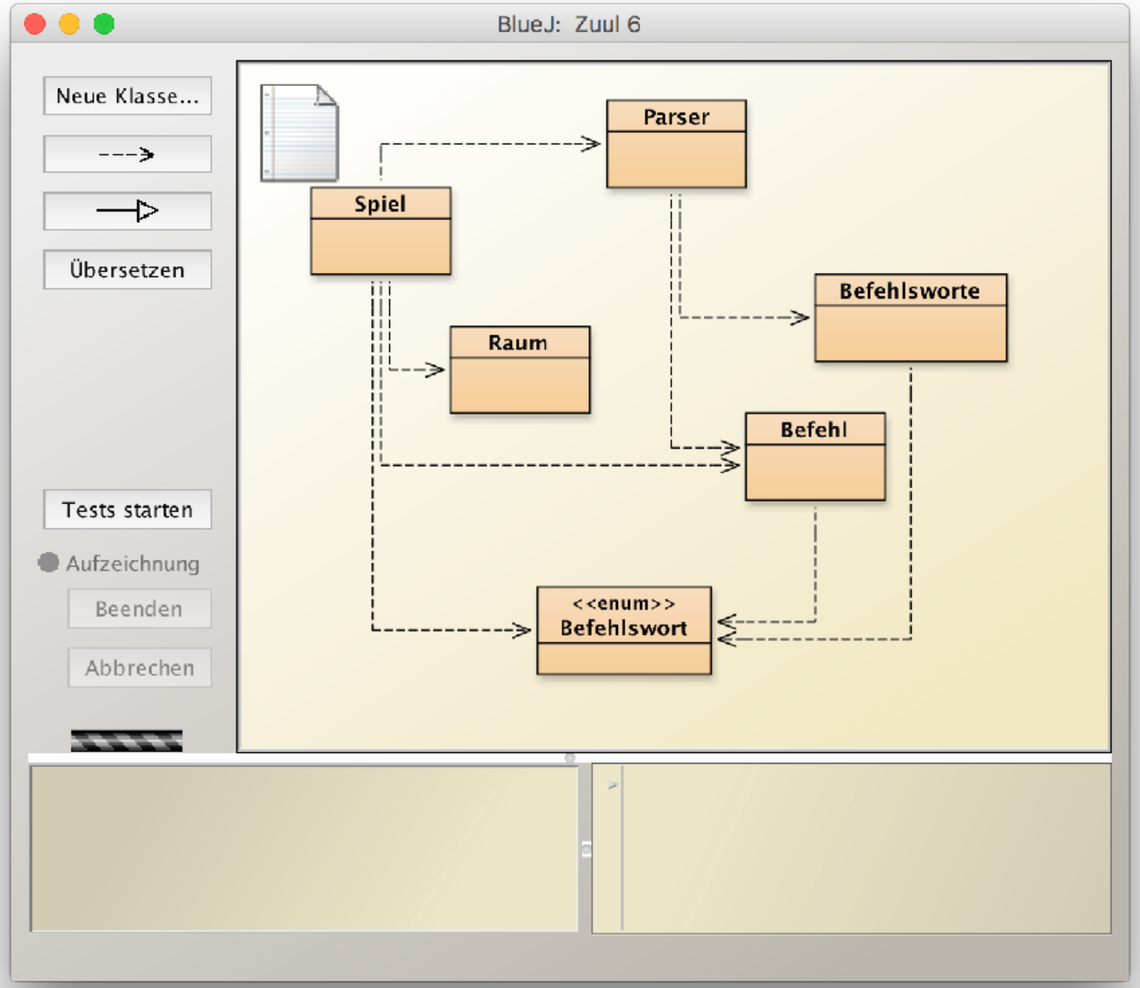
```
private final Map<String, Raum> ausgänge;
```

Refactoring: Beispiel



- Eine Spieler:in kann Gegenstände im aktuellen Raum aufheben
- Eine Spieler:in kann beliebig viele Gegenstände tragen, aber nur bis zu einem Maximalgewicht
- Einige Gegenstände können nicht aufgehoben werden
- Eine Spieler:in kann Gegenstände im aktuellen Raum ablegen

Aufzählungstypen: Demo



Universität Bremen

Aufzählungstypen

- Definieren einen Satz benannter Konstanten
- Ihre Werte sind nur zu Werten desselben Typs kompatibel
- Können Attribute, Konstruktoren und Methoden enthalten
- Die Konstanten sind eigentlich statische Referenzen auf Aufzählungstyp-Instanzen

```
enum Befehlswort {GO, QUIT, HELP, UNKNOWN}
```

```
enum Befehlswort
{
    GO("go"), QUIT("quit"),
    HELP("help"), UNKNOWN("?");

    private final String name;

    private Befehlswort(final String name)
    {
        this.name = name;
    }

    public String toString()
    {
        return name;
    }
}
```

Universität Bremen

Aufzählungstypen: Vordefinierte Methoden

- **E.values()** liefert ein Array mit allen Werten des Aufzählungstypen **E** zurück
- **E.valueOf(String s)** liefert Wert des Aufzählungstyps **E** mit dem Namen **s** zurück
- **toString()** liefert Namen des Werts zurück
- **ordinal()** liefert Index des Werts zurück, in der Reihenfolge der Deklaration

```
void alleAusgeben()
{
    for (final Befehlswort b : Befehlswort.values()) {
        if (b != Befehlswort.UNKNOWN) {
            System.out.print(b + " ");
        }
    }
    System.out.println();
}
```

Zusammenfassung der Konzepte

- (**Implizite**) **Kopplung** und **Kohäsion**
- **Code-Duplizierung**
- **Kapselung**
- **Entwurf nach Zuständigkeiten**
- **Vorausdenken** und **Refactoring**
- **Aufzählungstyp**