Universität Bremen

Praktische Informatik 1
Fehler vermeiden 2

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

Zusicherungen im Code

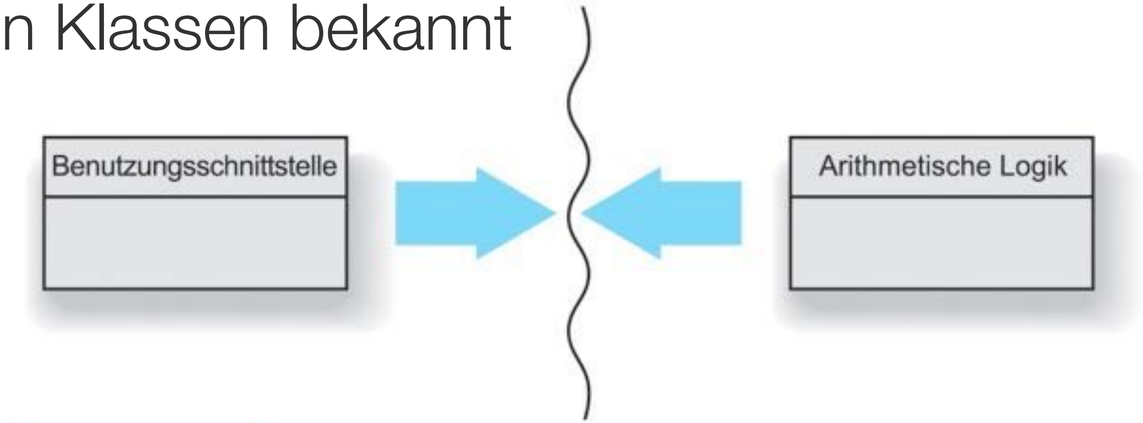
- Programmierer:in kann ihre Annahmen explizit formulieren
- **assert** erzeugt einen **AssertionError**, wenn Bedingung nicht erfüllt ist (mit optionalem Text als Nachricht)
- Müssen in der JVM explizit aktiviert werden
  (Parameter **-ea** wird von BlueJ automatisch gesetzt)
- Dürfen den Zustand des Programms nicht beeinflussen

```
assert bedingung;
assert bedingung : "Text";
```

```
final int zeitpunkt = TAGESBEGINN + stunde;
assert zeitpunkt <= LETZTER_PLANBARER_TERMIN;
```

Modularisierung und Schnittstellen

- **Modularisierung** ist zwingend notwendig, wenn mehrere Entwickler:innen an einem Projekt arbeiten (aber auch sonst eine gute Idee)
- Damit mehrere Komponenten zusammenarbeiten können, muss die **Schnittstelle** zwischen ihnen klar definiert sein
- **Schnittstelle**: Teile der Klasse, die anderen Klassen bekannt sind
  - Wird in der (JavaDoc) Dokumentation beschrieben



Universität Bremen

Schnittstelle der arithmetisch-logischen Einheit

```
/**
 * Liefere den Wert, der aktuell in
 * der Anzeige gezeigt wird.
 */
int gibAnzeigewert()

/**
 * Eine Zifferntaste wurde getippt.
 * @param ziffer Die einzelne Ziffer.
 */
void zifferGetippt(int ziffer)

/**
 * Die '+'-Taste wurde getippt.
 */
void plus()
```

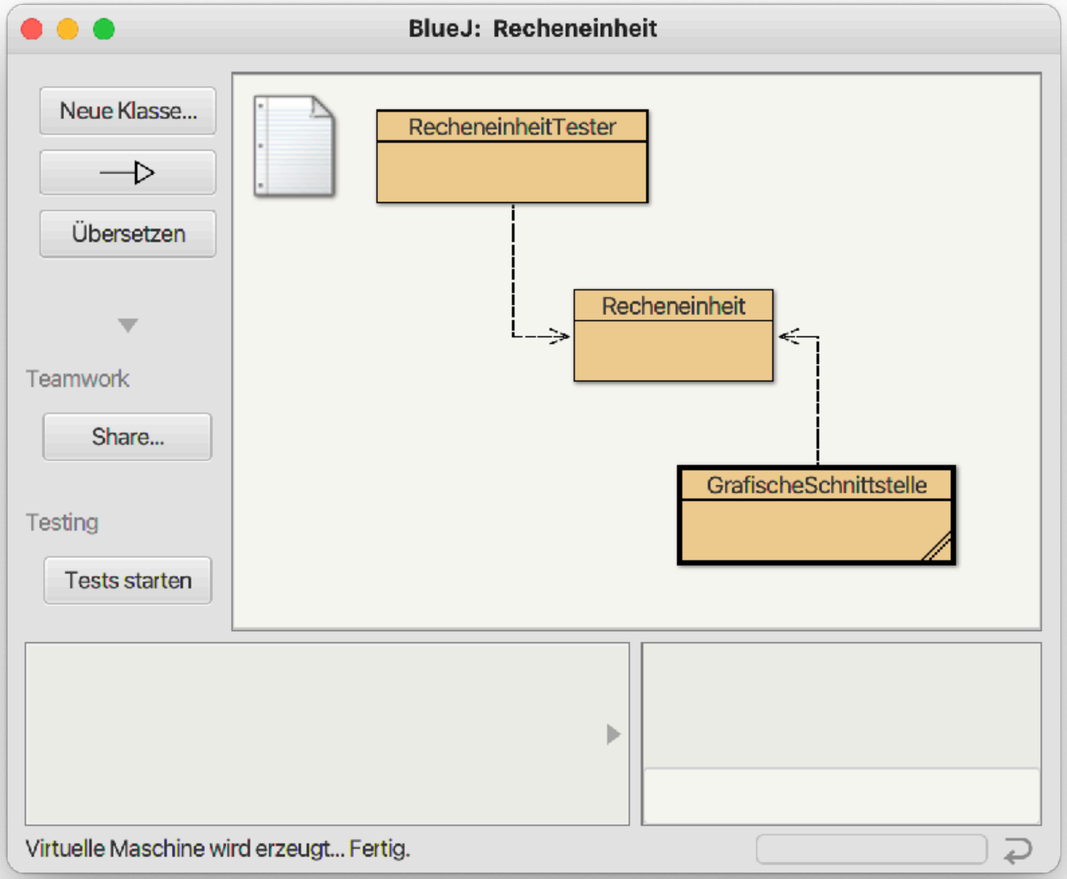
```
/**
 * Die '-'-Taste wurde getippt.
 */
void minus()

/**
 * Die Gleich-Taste wurde getippt.
 */
void gleich()

/**
 * Die C-Taste (für 'Clear') wurde
 * getippt.
 */
void clear()
```

Universität Bremen

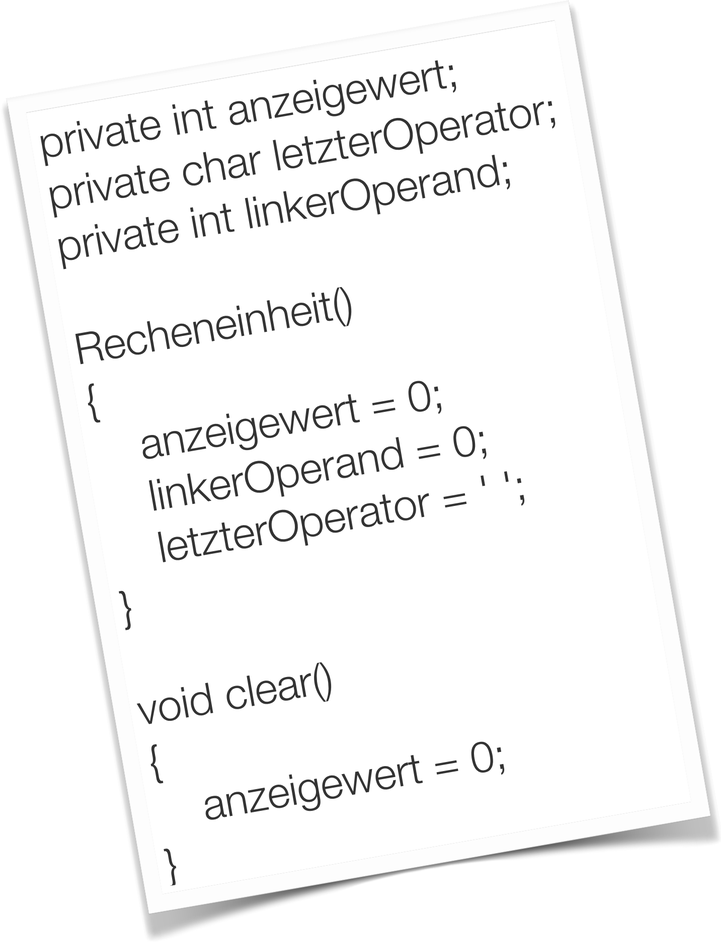
Recheneinheit: Demo



Universität Bremen

Manuelle Ausführung (Walkthrough)

- Zeilenweises Durchgehen eines Quelltextabschnitts, bei dem **Zustandsänderungen** und **Verhalten** einer Anwendung beobachtet werden
- Ausführung mit **Papier** und **Bleistift**
- Eignet sich eher für lokal begrenzte Probleme
- Andere Ansätze zur Fehlerfindung
  - Ortswechsel
  - Längere Pausen, z.B. „Eine Nacht darüber schlafen“



```
private int anzeigewert;
private char letzterOperator;
private int linkerOperand;

Recheneinheit()
{
    anzeigewert = 0;
    linkerOperand = 0;
    letzterOperator = ' ';
}

void clear()
{
    anzeigewert = 0;
}
```

Zustand kontrollieren

- Durchgehen des Quelltextes Methodenaufruf für Methodenaufruf
- Notieren der Zustände der Attribute in einer Tabelle, z.B. nach jedem Methodenaufruf

C 3

```
private int anzeigewert;
private int linkerOperand;
private char letzterOperator;
Recheneinheit()
{
    anzeigewert = 0;
    linkerOperand = 0;
    letzterOperator = ' ';
}
void clear()
{
    anzeigewert = 0;
}
void zifferGetippt(final int ziffer)
{
    anzeigewert = anzeigewert * 10 + ziffer;
}
```

Aufgerufene Methode	anzeigewert	linkerOperand	letzterOperator
Recheneinheit()	0	0	' '
clear()	0	0	' '
zifferGetippt(3)	3	0	' '

Universität Bremen

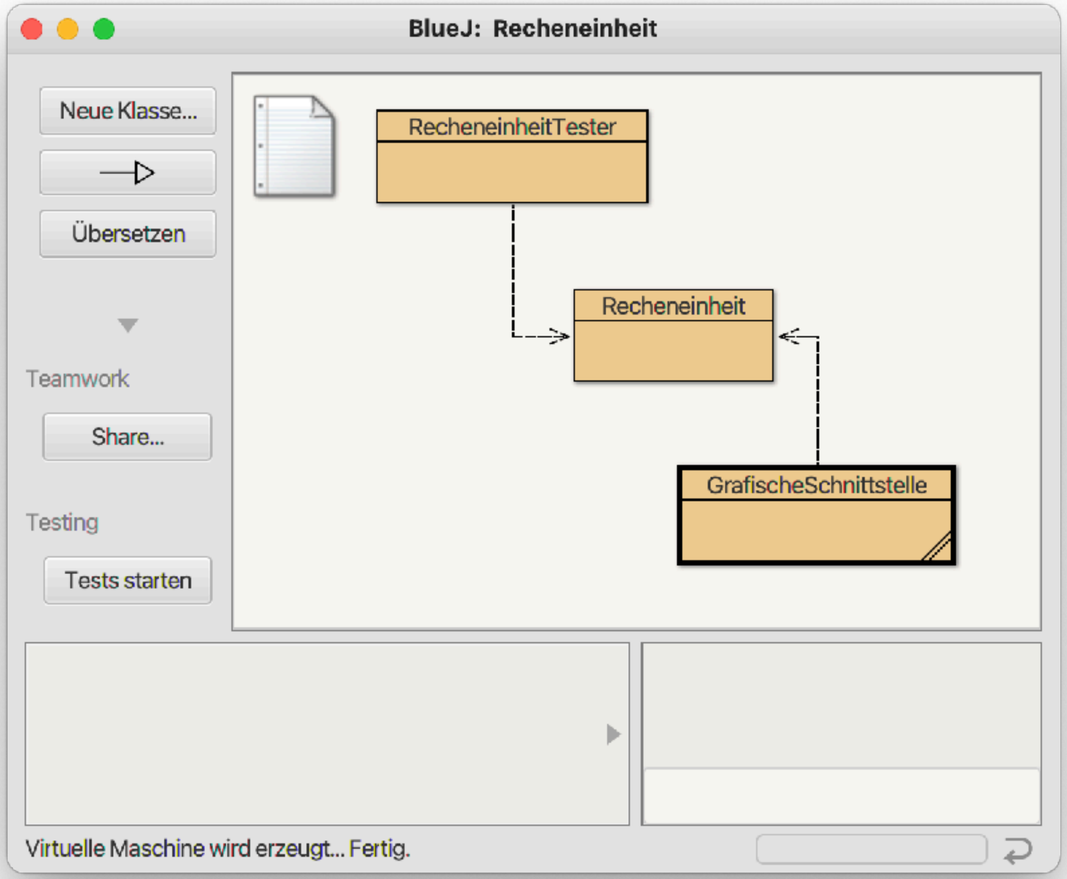
```
private int anzeigewert;
private int linkerOperand;
private char letzterOperator;
Recheneinheit() {
    anzeigewert = 0;
    linkerOperand = 0;
    letzterOperator = ' ';
}
void clear() {
    anzeigewert = 0;
}
void zifferGetippt(final int ziffer) {
    anzeigewert = anzeigewert * 10 + ziffer;
}
void plus() {
    letztenOperatorAnwenden();
    letzterOperator = '+';
    anzeigewert = 0;
}
```

```
void minus() {
    letztenOperatorAnwenden();
    letzterOperator = '-';
    anzeigewert = 0;
}
void gleich() {
    if (letzterOperator == '+') {
        anzeigewert = linkerOperand + anzeigewert;
    } else {
        anzeigewert = linkerOperand - anzeigewert;
    }
    linkerOperand = 0;
}
private void letztenOperatorAnwenden() {
    if (letzterOperator == '+') {
        linkerOperand += anzeigewert;
    } else if (letzterOperator == '-') {
        linkerOperand -= anzeigewert;
    } else { // Es gibt keinen letzten Operator.
        linkerOperand = anzeigewert;
    }
}
```

C 9 - 4 =
C 3 + 4 =

	anzeigewert	linkerOperand	letzterOperator
Rech()	0	0	' '
clear()	0	0	' '
ziffer(9)	9	0	' '
leOA()	9	9	' '
minus()	0	9	'-'
ziffer(4)	4	9	'-'
gleich()	5	0	'-'
clear()	0	0	'-'
ziffer(3)	3	0	'-'
leOA()	3	-3	'-'
plus()	0	-3	'+'
ziffer(4)	4	-3	'+'
gleich()	1	0	'+'

Recheneinheit mit Ausgabeanweisungen: Demo



Universität Bremen

Ausgabeanweisungen

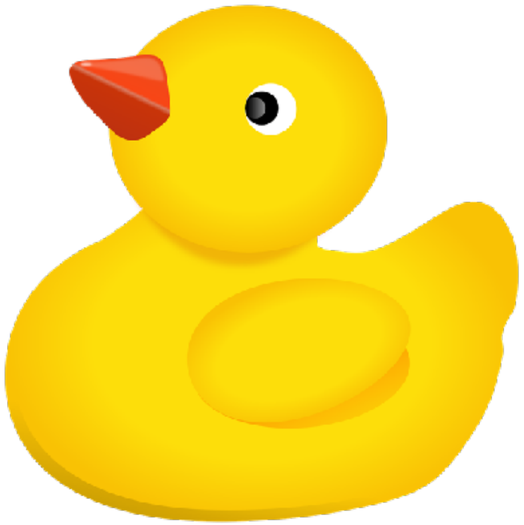
- Textausgabe während der Programmausführung
  - Welche Methodenaufrufe?
  - Mit welchen aktuellen Parametern?
  - In welcher Reihenfolge?
  - Werte von Attributen und lokalen Variablen an wichtigen Stellen
- Ausgabeanweisungen können abschaltbar implementiert werden
- Für Interessierte: Paket **java.util.logging**

```
private boolean testAusgaben = true;
void zifferGetippt(final int ziffer)
{
    testAusgabe("ziffer " + ziffer);
    anzeigewert = anzeigewert * 10 + ziffer;
    testAusgaben();
}
private void testAusgaben()
{
    testAusgabe("anzeigewert = " + anzeigewert +
                ", linkerOperand = " + linkerOperand +
                ", letzterOperator = " + letzterOperator);
}
private void testAusgabe(final String info)
{
    if (testAusgaben) {
        System.out.println(info);
    }
}
```

Universität Bremen

Mündliche Ausführung

- Einer anderen Person die Arbeitsweise einer Klasse oder Methode erklären
- Mögliche Ergebnisse
  - Zuhörer:in entdeckt den Fehler
  - Durch die Verbalisierung fällt einem selbst der Fehler auf
- Zuhörer:in muss also nicht mit dem Quelltext vertraut sein
  - Deshalb kann der Code z.B. auch einer Gummiente erklärt werden



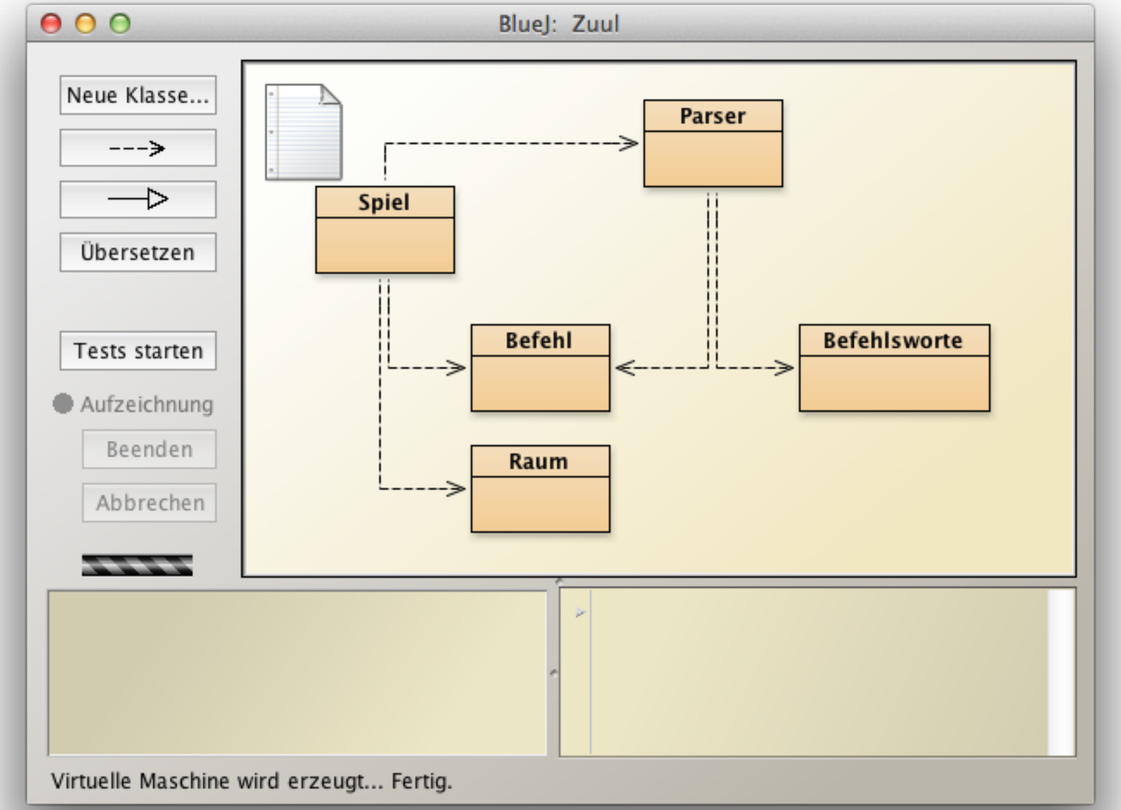
Universität Bremen

Zusammenfassung der Konzepte

- **assert**
- **Modularisierung** und **Schnittstellen**
- **Manuelle Ausführung** und **mündliche Ausführung**
- **Ausgabeanweisungen**

Hausaufgabe bis Dienstag: Die Welt von Zuul

- Was tut die Anwendung?
- Welche Befehle akzeptiert das Spiel?
- Was bewirken die Befehle?
- Welche Räume gibt es (Karte)?
- **Was tun die einzelnen Klassen?**



Universität Bremen

Quiz

```
Gclass Counter
{
    private final int value;

    void Counter()
    {
        value = 0;
    }

    int getValue()
    {
        return va1lue;
    }
```

```
    int void count()
    {
        if (value == 1000) {
            value = 0;
        }
        else {
            value = value + 1;
        }
    }
}
```