Universität Bremen

Praktische Informatik 1
Klassendefinitionen 1

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

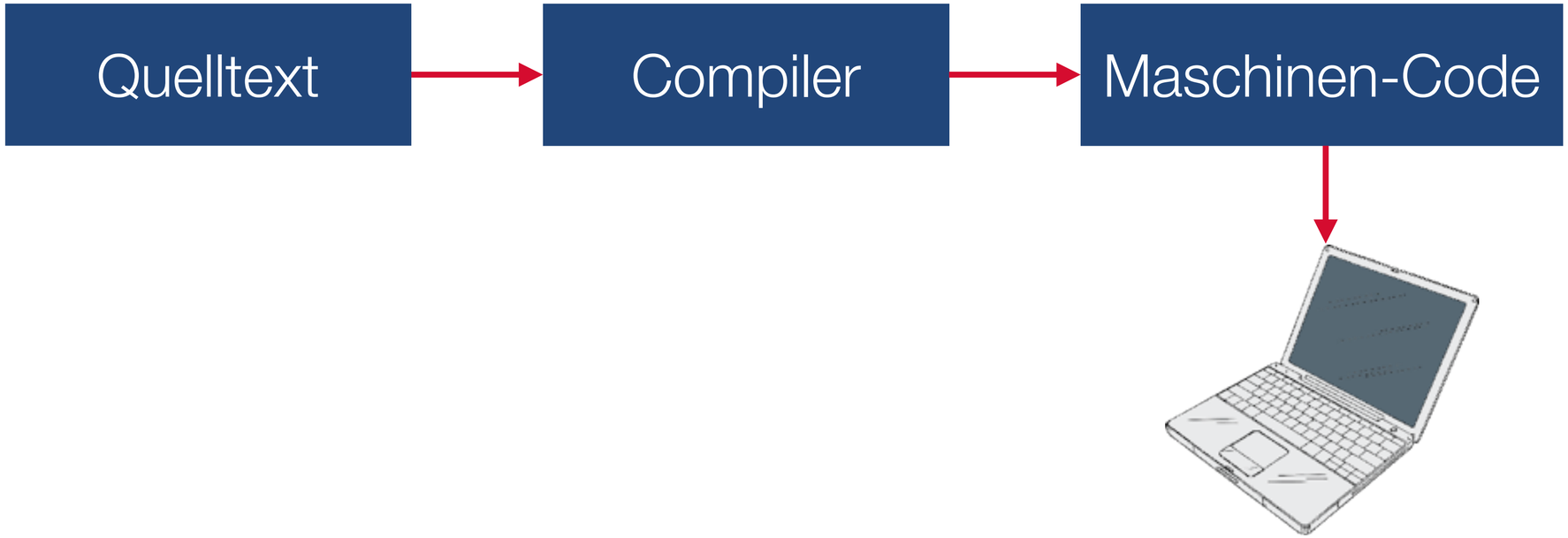
Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

Übersetzung

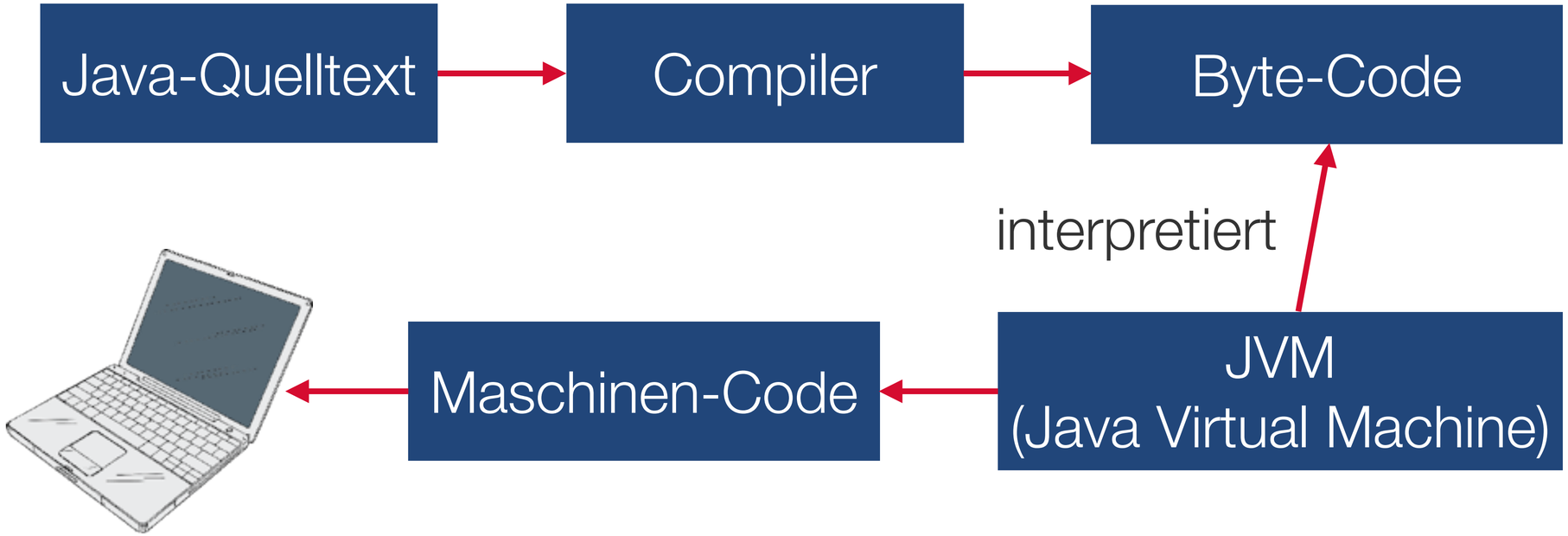
- Computer verstehen kein Java
- Darum muss der Menschen-lesbare Quelltext in Maschinen-ausführbare Form übersetzt werden



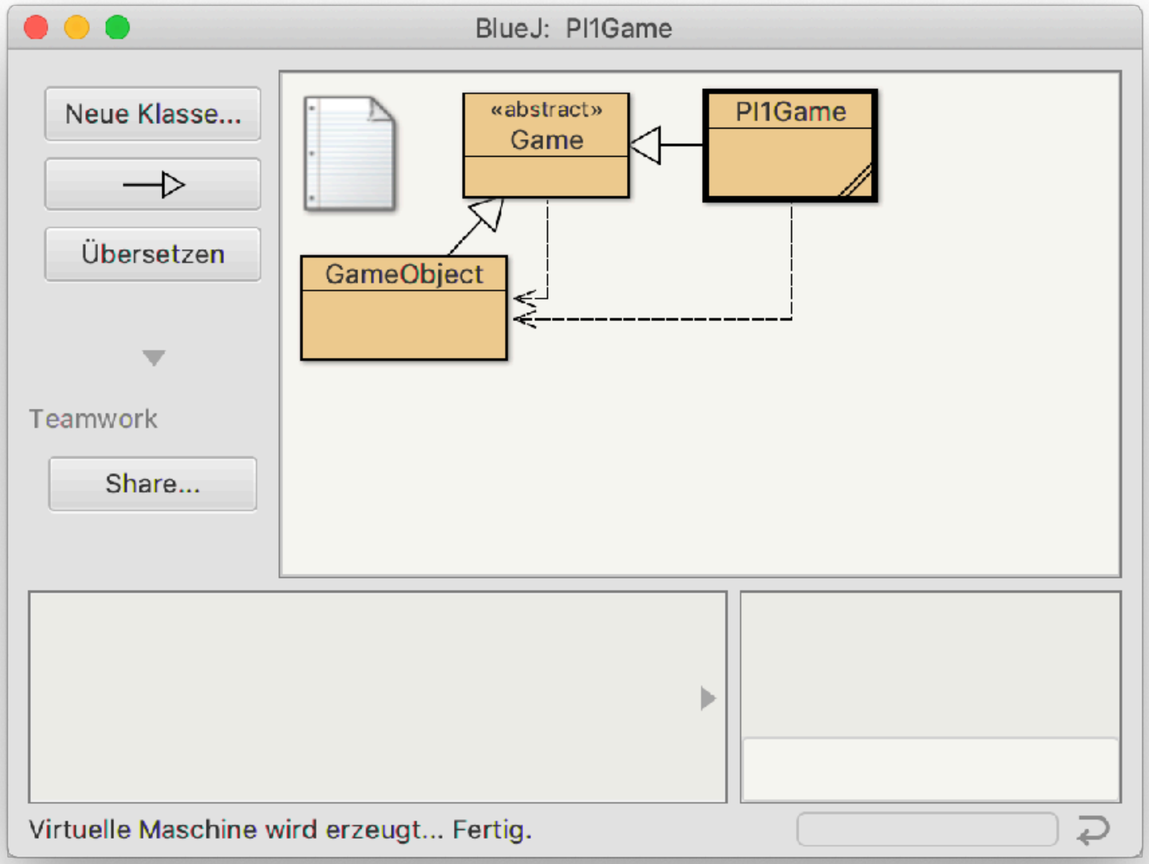
Übersetzung und Ausführung bei Java

- Computer verstehen kein Java

- Darum muss der Menschen-lesbare Quelltext in Maschinen-ausführbare Form übersetzt werden



Erstes Programm: Demo



Universität Bremen

Methoden

- **Methodenkopf**
  - Enthält die **Signatur** der Methode
  - **static**: Methode gehört nicht zu einem Objekt

```
static void main()
{
    playSound("step");
    sleep(1000);
}
```

- **Methodenrumpf**
  - Enthält die **Anweisungen**, die ausgeführt werden, wenn diese Methode **aufgerufen** wird
  - Anweisungen werden **nacheinander ausgeführt**, in der Reihenfolge, in der sie aufgeschrieben sind
  - Sie werden jeweils mit einem **;** beendet

```
static void main()
{
    playSound("step");
    sleep(1000);
}
```

Universität Bremen

Objekte

- Ein neues Objekt (d.h. eine **Instanz**) wird durch **new** gefolgt vom Aufruf des **Konstruktors** erzeugt

```
new GameObject(0, 1, 2, "laila");
```

- Ein Konstruktor hat **denselben Namen** wie die Klasse, zu der er gehört

- Um auf Objekte zugreifen zu können, müssen sie z.B. an **Konstante zugewiesen** werden

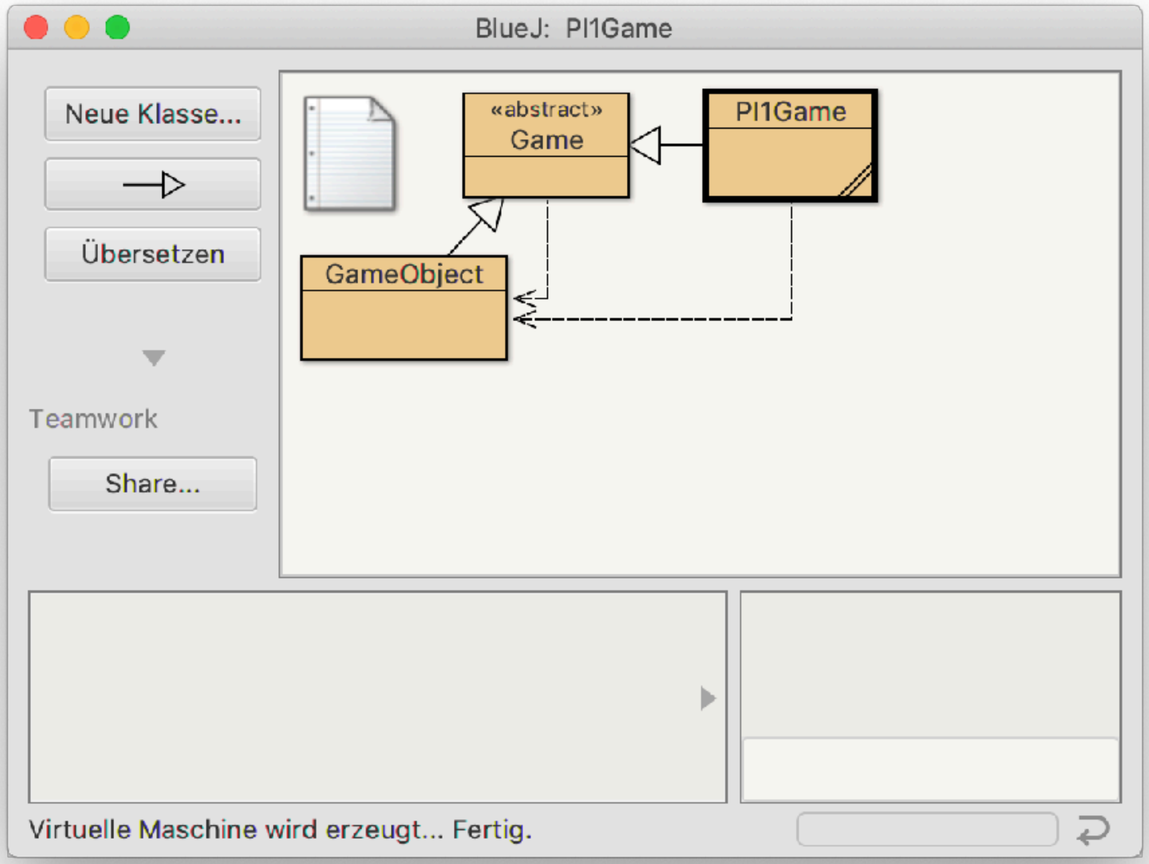
```
final GameObject player = new GameObject(0, 1, 2, "laila");
```

- Mit dem Punkt (**.**) können dann **Objektmethoden** aufgerufen werden

```
player.setRotation(0);
```

- Schema: *wer* **.** *tut-was* **(** *womit* **)**

Bedingte Ausführung: Demo



Universität Bremen

Lokale Konstanten und Variablen

- Speicherung von Werten während der Ausführung einer Methode
- Existieren ab ihrer Definition bis zum Ende des sie einschließenden Blocks

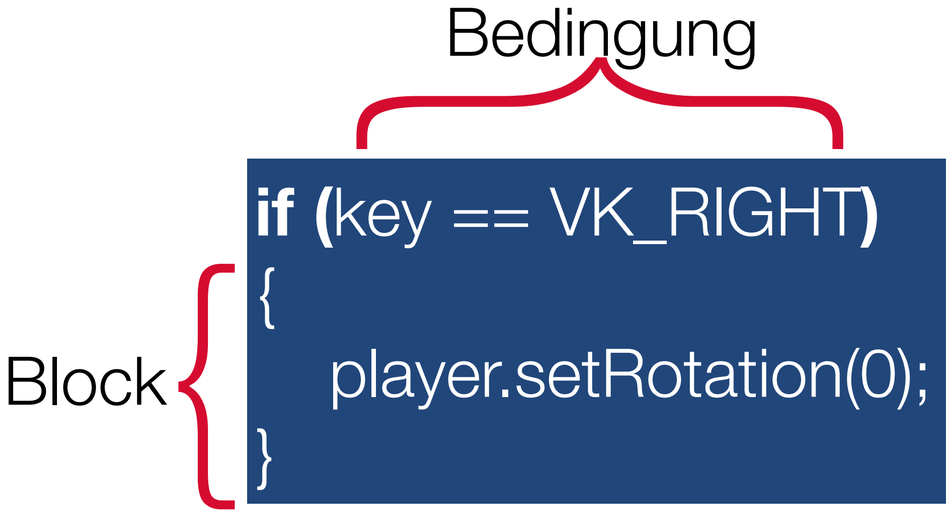
```
{
    int key = getNextKey();
                  :
}
// Kein key mehr
```

- Müssen vor dem ersten Auslesen initialisiert werden, z.B. bei ihrer Definition
- Konstanten (**final**) behalten denselben Wert während ihrer Lebenszeit, Variablen können hingegen geändert werden
- Wenn möglich, Konstanten verwenden

```
int coord = player.getX();
coord = player.getY();
final int key = getNextKey();
key = getNextKey(); // Fehler!
```

Bedingte Ausführung

- Die **if**-Anweisung macht die Ausführung eines **Blocks** von Anweisungen abhängig von einer **Bedingung**



- Die Bedingung ist ein **boolescher Ausdruck**, der **true** oder **false** liefert

- Der Anweisungsblock wird nur ausgeführt, wenn die Bedingung **true** ist

```
if ( Bedingung ) {
    Befehl1;
    Befehl2;
        :
}
```

Universität Bremen

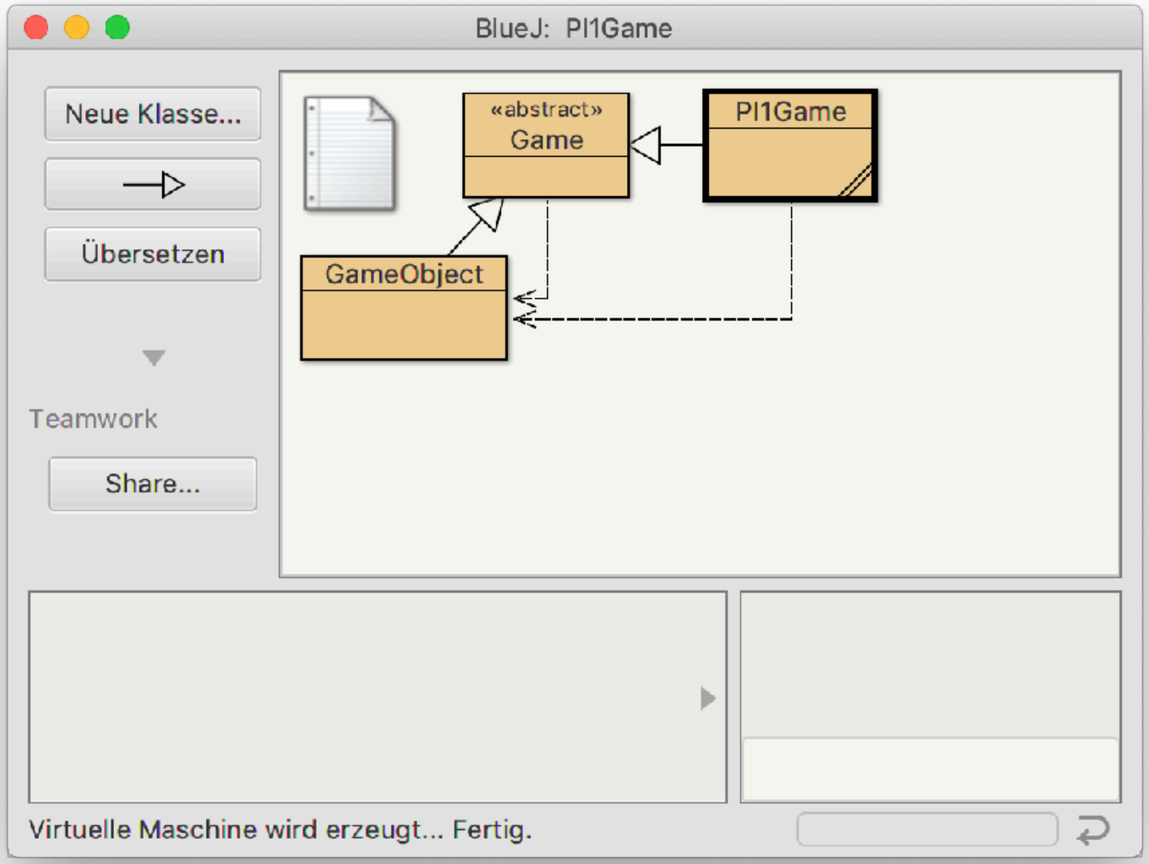
Vergleichsoperatoren

- Vergleichsoperatoren vergleichen ihren linken **Operanden** mit dem rechten und liefern **true** oder **false** zurück
- Haben beide Operanden einen **kompatiblen Typ**, können sie auf (Un)Gleichheit getestet werden
- Sind beide von einem **Zahlentyp**, kann ihre Größe verglichen werden
- Frage: Was ist das Gegenteil von **kleiner als**?

Operator	Bedeutung	true	false
<	Kleiner als	4 < 5	4 < 4
>	Größer als	5 > 4	4 > 4
<=	Kleiner gleich	4 <= 4	5 <= 4
>=	Größer gleich	4 >= 4	4 >= 5
==	Gleich	4 == 4	4 == 5
!=	Ungleich	4 != 5	4 != 4

Universität Bremen

Bedingte Verzweigung: Demo



Universität Bremen

Bedingte Verzweigung

- Die **if**-**else**-Anweisung macht die Ausführung **zweier** Blöcke von Anweisungen abhängig von **einer** Bedingung

- Der **erste** Anweisungsblock wird nur ausgeführt, wenn die Bedingung **true** ist

- Der **zweite** Anweisungsblock wird nur ausgeführt, wenn die Bedingung **false** ist

```
if (key == VK_RIGHT) {
    player.setRotation(0);
}
else {
    playSound("error");
}
```

```
if ( Bedingung ) {
    Wenn true
}
else {
    Wenn false
}
```

Universität Bremen

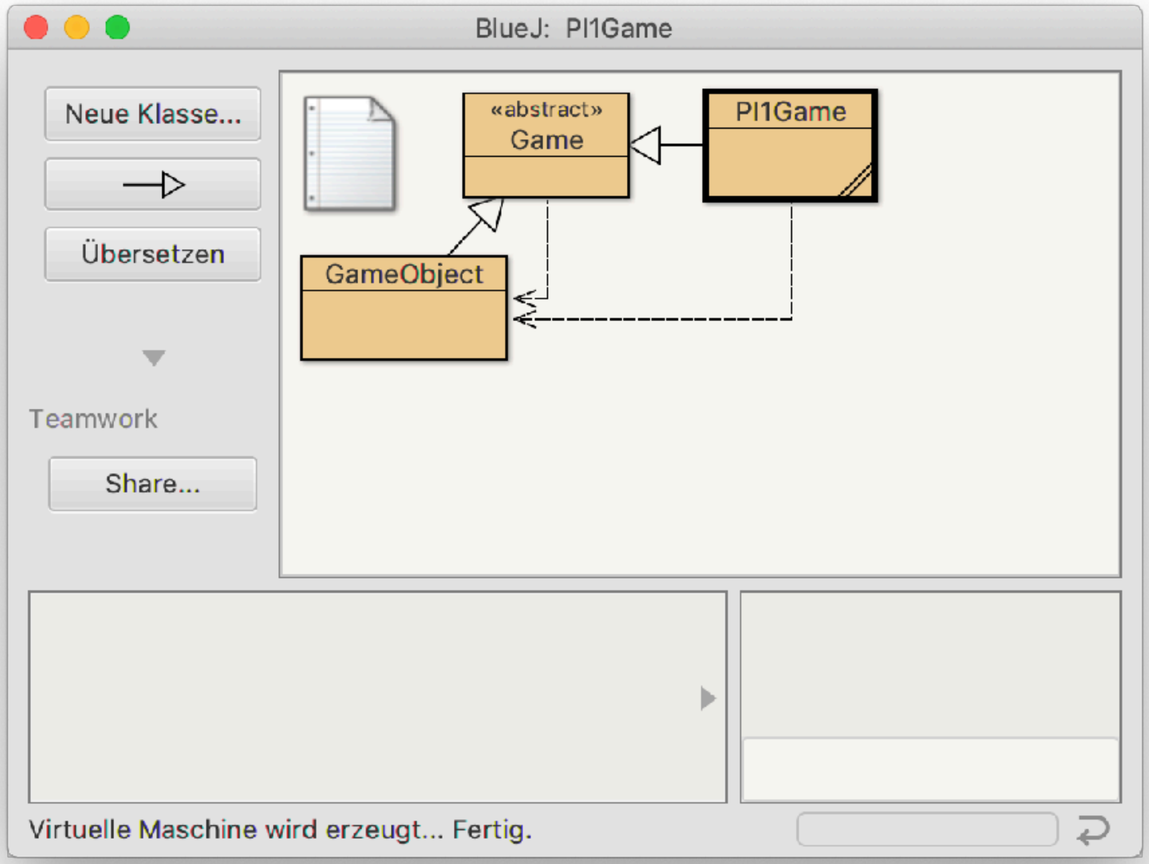
Mehrfach-Verzweigung

- **else**-Blöcke können wieder eine Verzweigung enthalten (**if**-Blöcke auch), so dass nacheinander mehrere Fälle getestet werden können
  - Das kann sich beliebig oft wiederholen
- Wenn der **else**-Block jeweils nur eine weitere **if**-Anweisung enthält, gibt es hierfür auch eine Kurzschreibweise
- Fortgeschrittene können auch einen Blick auf die (etwas spezialisierte) **switch**-Anweisung werfen

```
if (key == VK_RIGHT) {
    player.setRotation(0);
}
else {
    if (key == VK_LEFT) {
        player.setRotation(2);
    }
    else {
        playSound("error");
    }
}
```

```
if (key == VK_RIGHT) {
    player.setRotation(1);
}
else if (key == VK_LEFT) {
    player.setRotation(2);
}
else {
    playSound("error");
}
```

Bedingte Verzweigung mit Schleife: Demo



Universität Bremen

Coding-Style

```
final int key = getNextKey();
if (key == VK_SPACE) {
    playSound("error");
}
```

- Programmier- und Formatierungskonventionen
  - Programmierkonvention: Z.B. „Alles, was seinen Wert nicht ändert, wird als **final** deklariert“
  - Formatierungskonvention: Z.B. „Schließende, geschweifte Klammern stehen allein in einer Zeile“
- Machen Quelltexte leichter lesbar, insbesondere, wenn mehrere Personen zusammen programmieren
- Alle Quelltexte dieser Veranstaltung folgen einer Formatierungskonvention
  - Einfach mal darauf achten!

Universität Bremen

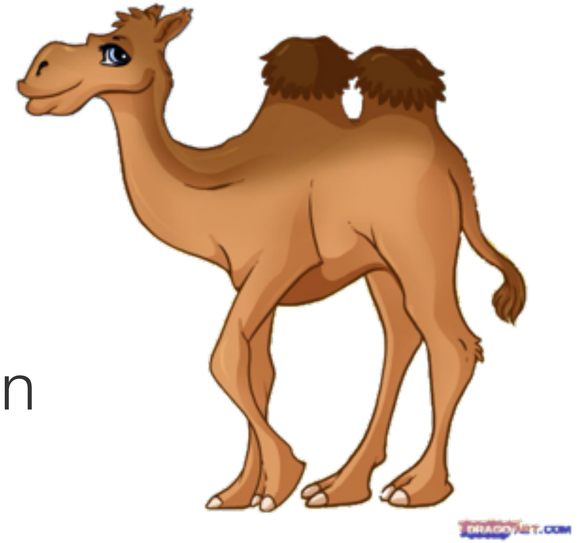
Coding-Style: Einrückung

- Die Struktur eines Programms wird durch **Einrückungen** verdeutlicht
- Jeder Beginn eines Blocks erhöht die **Einrücktiefe**, jedes Ende verringert sie
- BlueJ verwendet **Einrückstufen** von 4 Zeichen, z.B. mit →| bzw. **Umsch+→|**
- BlueJ kann mit „Bearbeiten|Auto-Layout“ den Quelltext automatisch einrücken
  - Mehrzeilige Anweisungen werden aber ungünstig eingerückt

```
class Buzzer extends Game
{
    static void main()
    {
        final int key = getNextKey();
        if (key == VK_SPACE) {
            playSound("error");
        }
    }
}
```

Coding-Style: Konventionen für Namen (Bezeichner)

- CamelCase
  - Klassen beginnen mit Großbuchstaben
  - Methoden, Variablen und Kurzzeitkonstanten (**final**) beginnen mit Kleinbuchstaben



- Dauerkonstanten (**static final**) werden nur groß geschrieben und verwenden _ als **WORT_TRENNER**
- Namen sollten prägnant beschreiben, welche Aufgabe die Klasse, die Methode, das Attribut usw. hat

Universität Bremen

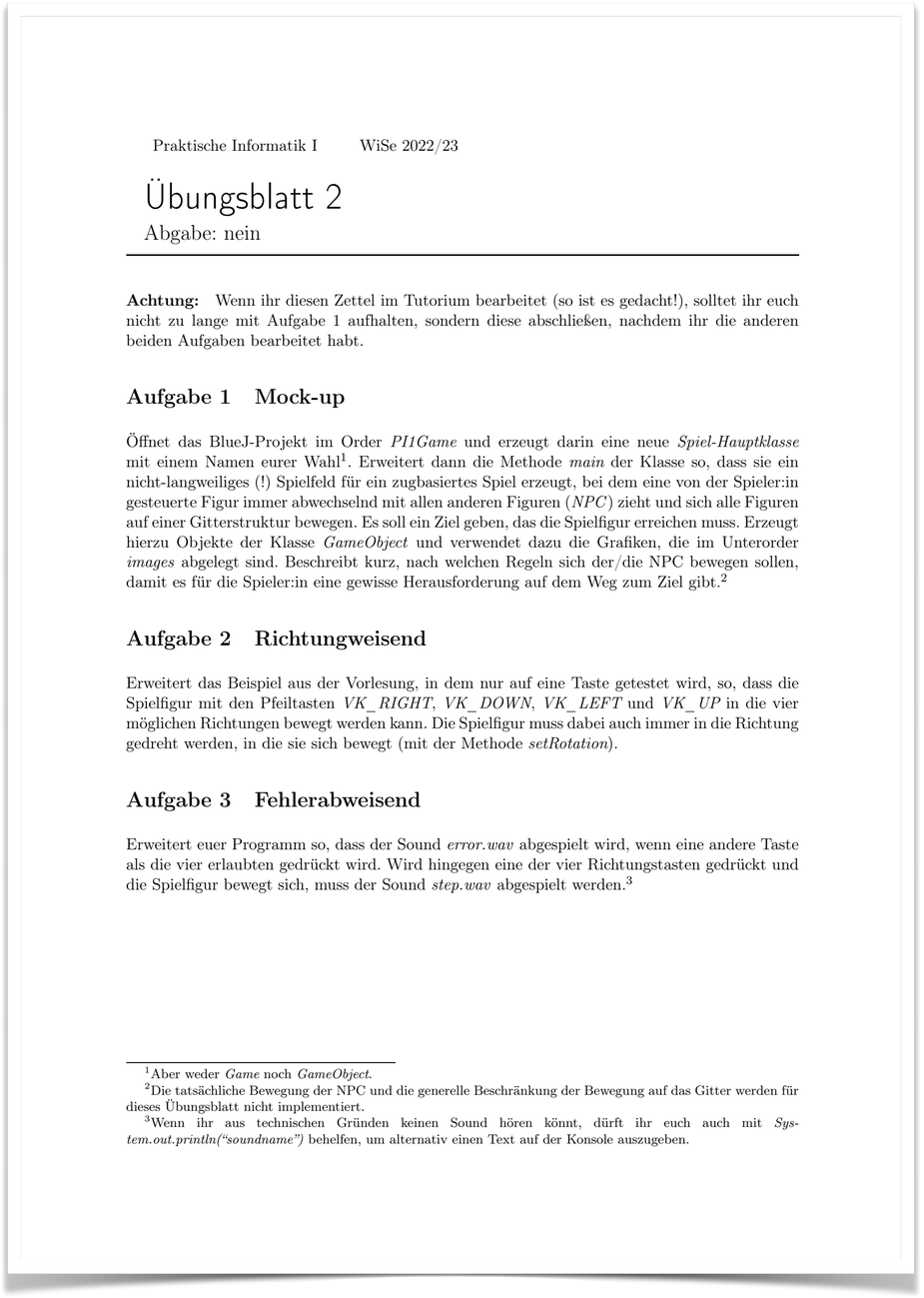
Zusammenfassung der Konzepte

- **Methodenkopf** und **Methodenrumpf**
- **Quelltext** und **Übersetzung**
- **Lokale Konstanten** und **Variablen**
- **Bedingte Ausführung**/**Verzweigung**
- **Vergleichsoperatoren**
- **Coding-Style**, insbesondere **Einrückung**

Universität Bremen

Übungsblatt 2

- Aufgabe 1: Szene erzeugen
  - Besser fertigstellen, wenn alles andere erledigt ist
  - Profitipp: Wer eigene Grafiken einsetzen will, kann am Anfang von **main()** von Hand die Zeichenfläche mit passenden Abmessungen erzeugen mit **new GameObject.Canvas(**<fensterbreite>**,** <fensterhöhe>**,** <kachelbreite>**,** <kachelhöhe>**)**
- Aufgabe 2: Bedingte Ausführung
- Aufgabe 3: Bedingte Mehrfachverzweigung



Praktische Informatik I WiSe 2022/23

Übungsblatt 2

Abgabe: nein

Achtung: Wenn ihr diesen Zettel im Tutorium bearbeitet (so ist es gedacht!), solltet ihr euch nicht zu lange mit Aufgabe 1 aufhalten, sondern diese abschließen, nachdem ihr die anderen beiden Aufgaben bearbeitet habt.

Aufgabe 1 Mock-up

Öffnet das BlueJ-Projekt im Order *PI1Game* und erzeugt darin eine neue *Spiel-Hauptklasse* mit einem Namen eurer Wahl[1]. Erweitert dann die Methode *main* der Klasse so, dass sie ein nicht-langweiliges (!) Spielfeld für ein zugbasiertes Spiel erzeugt, bei dem eine von der Spieler:in gesteuerte Figur immer abwechselnd mit allen anderen Figuren (*NPC*) zieht und sich alle Figuren auf einer Gitterstruktur bewegen. Es soll ein Ziel geben, das die Spielfigur erreichen muss. Erzeugt hierzu Objekte der Klasse *GameObject* und verwendet dazu die Grafiken, die im Unterorder *images* abgelegt sind. Beschreibt kurz, nach welchen Regeln sich der/die NPC bewegen sollen, damit es für die Spieler:in eine gewisse Herausforderung auf dem Weg zum Ziel gibt.[2]

Aufgabe 2 Richtungweisend

Erweitert das Beispiel aus der Vorlesung, in dem nur auf eine Taste getestet wird, so, dass die Spielfigur mit den Pfeiltasten *VK_RIGHT*, *VK_DOWN*, *VK_LEFT* und *VK_UP* in die vier möglichen Richtungen bewegt werden kann. Die Spielfigur muss dabei auch immer in die Richtung gedreht werden, in die sie sich bewegt (mit der Methode *setRotation*).

Aufgabe 3 Fehlerabweisend

Erweitert euer Programm so, dass der Sound *error.wav* abgespielt wird, wenn eine andere Taste als die vier erlaubten gedrückt wird. Wird hingegen eine der vier Richtungstasten gedrückt und die Spielfigur bewegt sich, muss der Sound *step.wav* abgespielt werden.[3]

[1] Aber weder *Game* noch *GameObject*.
[2] Die tatsächliche Bewegung der NPC und die generelle Beschränkung der Bewegung auf das Gitter werden für dieses Übungsblatt nicht implementiert.
[3] Wenn ihr aus technischen Gründen keinen Sound hören könnt, dürft ihr euch auch mit *System.out.println("soundname")* behelfen, um alternativ einen Text auf der Konsole auszugeben.