Universität Bremen

Praktische Informatik 1
Klassendefinitionen 2

Thomas Röfer

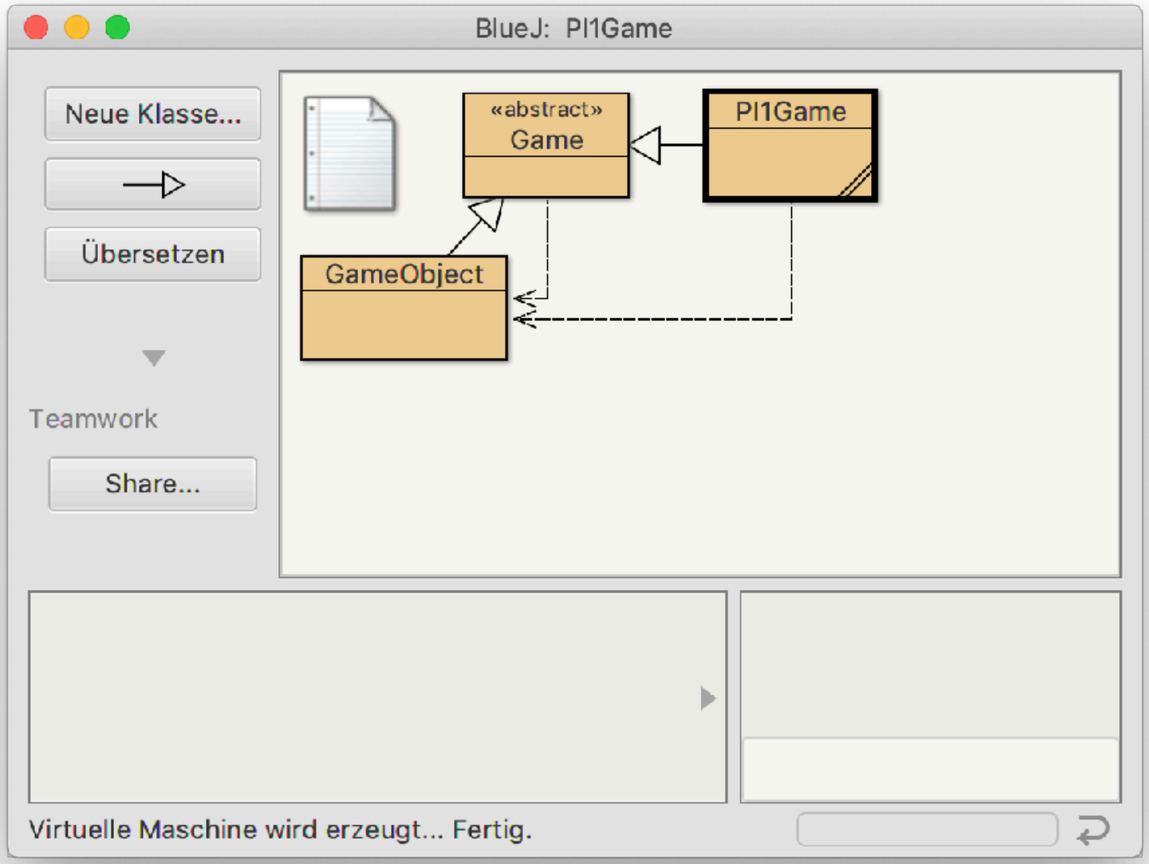
Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

Schleifen: Demo



Universität Bremen

Abweisende Schleife

- Solange die Bedingung wahr ist, wird der Schleifenrumpf ausgeführt
- Falls die Bedingung bereits zu Anfang nicht erfüllt ist, wird der Schleifenrumpf **niemals** ausgeführt
- Der Schleifenrumpf sollte die in der Bedingung verwendeten Variablen verändern, sonst terminiert die Schleife nicht (außer durch expliziten Abbruch im Schleifenrumpf)
- **while (true)**: Endlosschleife

```
int key = getNextKey();
while (key != VK_ESCAPE) {
    if (key == VK_RIGHT) {
        // usw.
    }
    key = getNextKey();
}
```

```
while ( bedingung ) {
    anweisungen
}
```

Universität Bremen

Annehmende Schleife

- Führe den Schleifenrumpf solange aus, bis die Bedingung falsch ist
- Falls die Bedingung bereits zu Anfang falsch ist, wird der Schleifenrumpf **dennoch** ausgeführt
- Wird eher selten verwendet

```
int key;
do {
    key = getNextKey();
    if (key == VK_RIGHT) {
        // usw.
    }
} while (key != VK_ESCAPE);
```

```
do {
    anweisungen
} while ( bedingung );
```

Schleifen verlassen oder vorzeitig fortsetzen

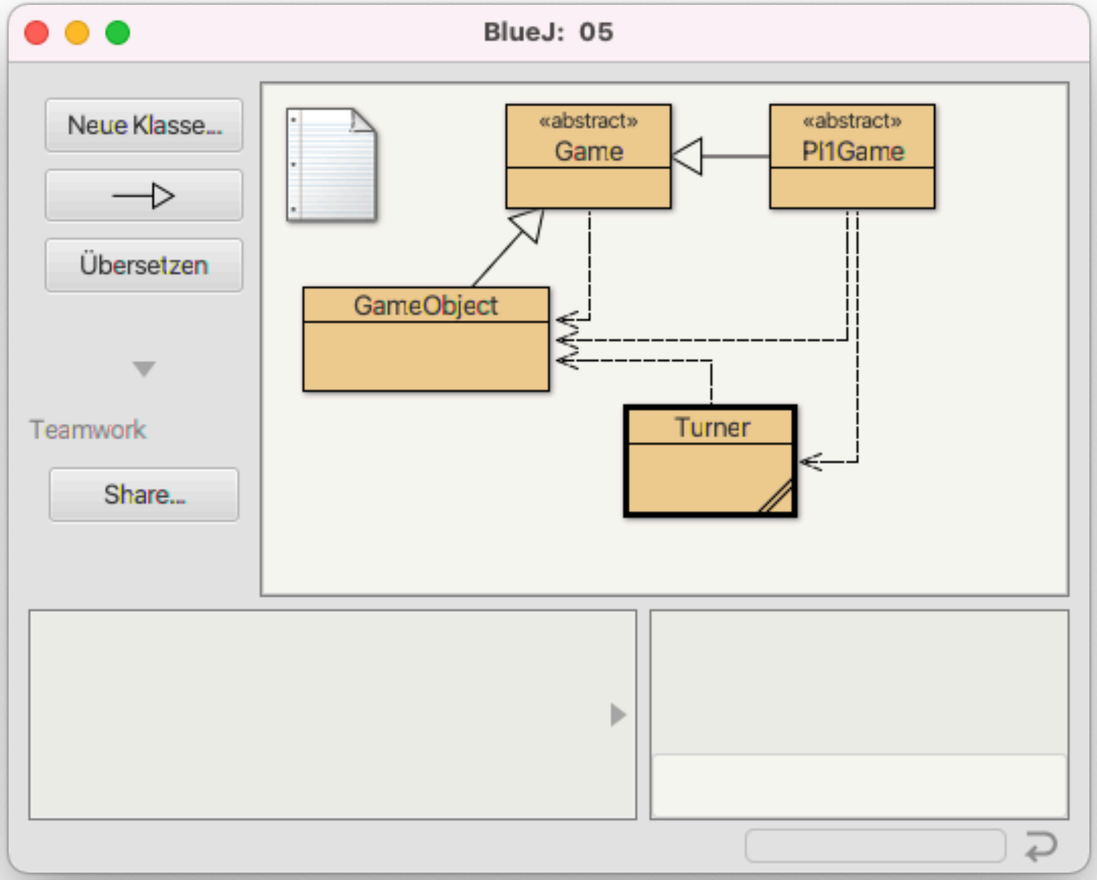
- **break**: Verlassen der aktuellen Schleife
  - Die Ausführung wird hinter der Schleife fortgesetzt
- **continue**: Der aktuelle Durchlauf der Schleife wird beendet
  - Die Ausführung wird direkt vor Ende des Schleifenblocks fortgesetzt
  - Der Schleifenblock wird wieder ausgeführt, wenn die Schleifenbedingung noch erfüllt ist

```
while (true) {
    final int key = getNextKey();
    if (key == VK_ESCAPE) {
        break;
    }
    // usw.
}
```

```
while (true) {
    final int key = getNextKey();
    if (key == VK_RIGHT) {
        player.setRotation(0);
    }
    // else if ...
    else {
        continue;
    }
    playSound("step");
}
```

Universität Bremen

Eigene Klasse erzeugen: Demo



Universität Bremen

Klassendefinition

- **Klassenkopf**
  - **Name** der Klasse
  - Optional weitere Angaben, zu denen wir später kommen
- **Klassenrumpf**
  - **Attribute**: Die Datenfelder, aus denen Instanzen der Klasse bestehen
  - **Konstruktoren**: Initialisieren Instanzen der Klasse
  - **Methoden**: Implementieren die Funktionalität der Klasse

```
class Klassenname
{
    Attribute
    Konstruktoren
    Methoden
}
```

Universität Bremen

Attribute

- Der Zustand von Objekten wird in **Attributen** (Datenfeldern, Objektvariablen, Zustandsfeldern) gespeichert
- Ihre Deklaration und Definition besteht aus
  - **Zugriffsmodifizierer**: meistens **private**
  - Optional **final** (wenn Attribut konstant ist)
  - **Typ**
  - **Name**
  - Optional eine **Initialisierung**

```
class Turner
{
    private final GameObject avatar;
    private int counter = 0;
```

Universität Bremen

Konstruktoren

- Ein **Konstruktor** initialisiert eine Instanz einer Klasse, also ihre Attribute
  - Er hat **denselben Namen** wie seine Klasse
  - Er hat **keinen** Rückgabetyp
- Konstruktoren können **überladen** sein, d.h. es kann mehrere mit unterschiedlichen **Signaturen** geben
  - Ein Konstruktor kann einen anderen als erste Anweisung aufrufen
- Ohne Konstruktor generiert Java automatisch einen ohne Parameter (**Standard-Konstruktor**)
- Java initialisiert variable Attribute standardmäßig mit **0**/**false**/**null**, konstante nicht

```
Turner(final GameObject avatar,
       final int maxCounter)
{
    this.avatar = avatar;
    this.maxCounter = maxCounter;
}
```

```
Turner(final GameObject avatar)
{
    this(avatar, 2);
}
```

Universität Bremen

Methodenkopf

- Enthält die **Signatur** der Methode
  - **Zugriffsmodifizierer**: Kann die Methode von außerhalb der Klasse aufrufen werden?
    nichts (*package private*): ja, **private**: nein
  - **Rückgabetyp**, **Name**, **Parameter**
- Methoden können **überladen** sein, d.h. es kann mehrere mit unterschiedlichen **Signaturen** geben
- Methoden in **verschiedenen Klassen** können **dieselbe Signatur** verwenden

```
void act()
{
    turn(1);
}

private void turn(
        final int direction)
{
    avatar.setRotation(
            avatar.getRotation()
            + direction);
}
```

Universität Bremen

Verwendung von Parametern

- Anweisungen können die **formalen Parameter** der Methode im **Methodenrumpf** verwenden
- Bei einem **Methodenaufruf** werden an den Stellen der Nennung der **formalen Parameter** die **aktuellen Parameterwerte** verwendet
- Parameter sind auch lokale Variablen bzw. Konstanten und können genauso verwendet werden

```
private void turn(
        final int direction)
{
    avatar.setRotation(
            avatar.getRotation()
            + direction);
}
```

```
turn(1);
```



```
{
    avatar.setRotation(
            avatar.getRotation()
            + 1);
}
```

Universität Bremen

Geheimnisprinzip

- Klassen können eine korrekte Ausführung nur garantieren, wenn sie die Hoheit über den Zustand ihrer Objekte behalten
- Daher werden Änderungen am Zustand nur über Methodenaufrufe erlaubt, d.h. alle Attribute sind **private**
- Konstruktoren und Methodenaufrufe prüfen die **Gültigkeit** der an sie übergebenen aktuellen Parameter und lehnen ungültige ab (wie? → später)
  - Dadurch kann der Objektzustand niemals ungültig werden
- Die **öffentliche Schnittstelle** einer Klasse (Menge der von außen verwendbaren Methoden) soll möglichst klein sein, d.h. auch Hilfsmethoden sind **private**

Universität Bremen

Eigenes Objekt und andere Objekte

- **Interner Methodenaufruf**: Aufruf einer Methode für eigenes Objekt

```
turn(1);
```

- **Externer Methodenaufruf**

```
otherTurner.turn(1);
```

  - Aufruf einer Methode eines anderen Objekts
  - Das Objekt wird vor dem Methodenaufruf genannt, gefolgt von einem **.**
  - Interne Methodenaufrufe sind eigentlich externe über Objekt **this**

```
this.turn(1);
```

- **Attributzugriff**: Gleiches gilt für Attribute

```
counter = counter + 1;
```

  - Da Attribute oft **private** sind, geht dies meistens nur für Attribute der eigenen Klasse

```
otherTurner.counter = otherTurner.counter + 1;
```

Universität Bremen

Zuweisungen

- Wertet den Ausdruck rechts vom Gleichheitszeichen aus und weist das Ergebnis der Variablen links zu
- Der vorherige Wert der Variablen geht verloren, d.h. sie ändert ihren Zustand
- Die linke Seite muss zu einer **Speicherstelle** ausgewertet werden können (**L-Wert**), für die rechte reicht auch ein Wert (**R-Wert**), sie kann also z.B. auch ein Literal sein
- Der **Typ** der rechten Seite muss zum Typ der linken Seite passen

```
class Turner
{
    private final GameObject avatar;
    private final int maxCounter;
    private int counter = 0;

    Turner(final GameObject avatar,
           final int maxCounter)
    {
        this.avatar = avatar;
        this.maxCounter = maxCounter;
    }

    void act()
    {
        counter = counter + 1;
        // ...
    }
}
```

Universität Bremen

Zusammenfassung der Konzepte

- **Abweisende** und **annehmende Schleife**
- **break** und **continue**
- **Klassendefinition**: **Attribut**, **Konstruktor** und **Methode**
- Nutzung von **Parametern**
- **Interner** und **externer Methodenaufruf**
- **Zuweisung**: **L-Wert** und **R-Wert**