EML Research

http://www.eml-research.de/

# Java - Part 3

Katja Wegner

katja.wegner@eml-r.villa-bosch.de

# Überblick

⇒ Objektorientierte Programmierung → Klassen

- Attribute
- Methoden
- Konstruktoren
- Modifikatoren

# Objektorientierte Programmierung

⇒ **Ziel:** Modellierung der realen Welt als Menge von Objekten

⇒ **dazu:** Abstraktion ihrer Eigenschaften als Attribute

und ihrer Fähigkeiten als Methoden

⇒ **Klasse:** Beschreibung von Objekten der gleichen Struktur

- Gleiche Attribute
- Gleiche Methoden

# Klassen

⇒ **Klassendefinition**

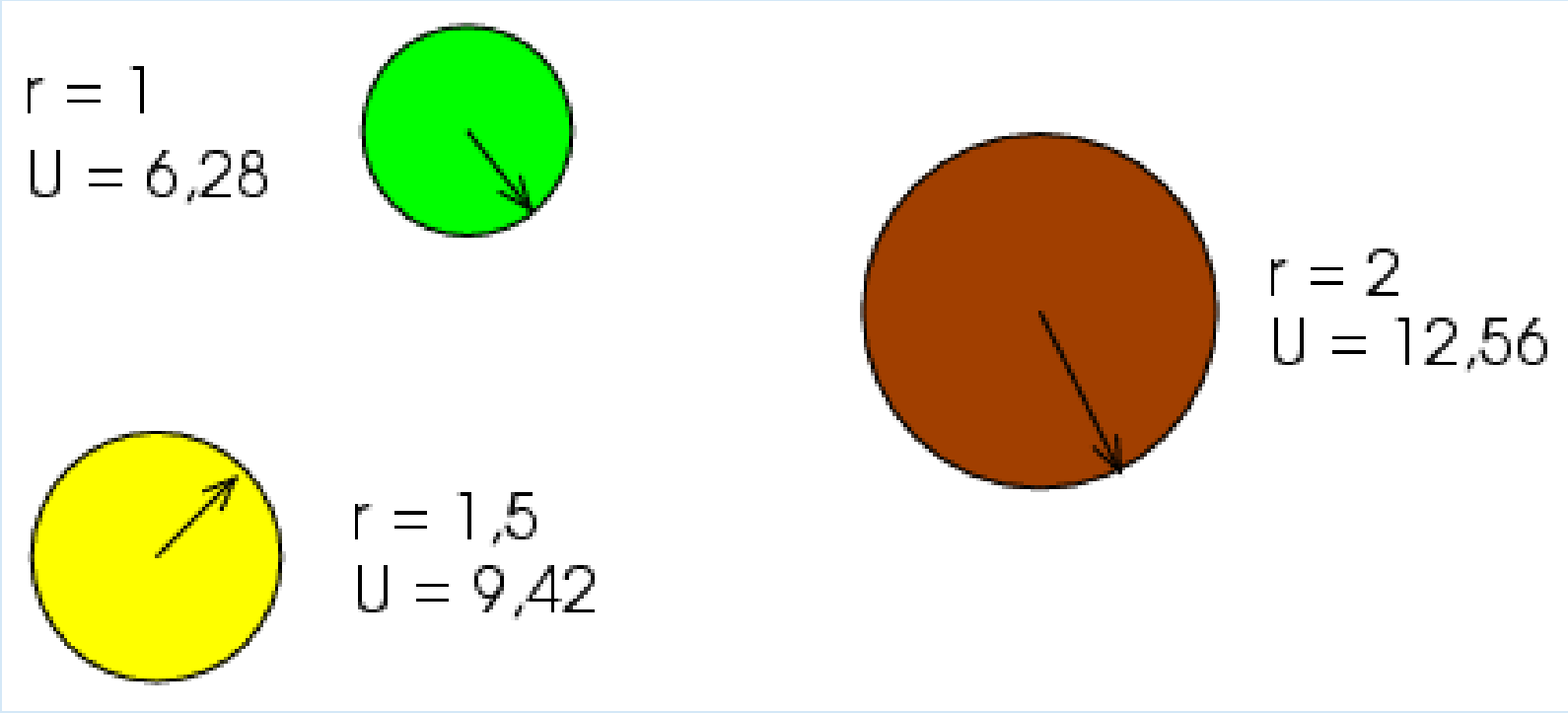
- **Attribute**
- **Methoden** (Verhaltensweise)
- **Konstruktoren** (Instantiierung, z.B. kreieren von Objekten)

```
class ClassName {
        attribut_1;
        ...
        attribut_k;

        konstruktor_1;
        ...
        konstruktor_m;

        methode_1;
        ...
        methode_n;
}
```

# Klassen - Beispiel

⇒ ein Kreis:

- besitzt einen Radius → Attribut
- mit dem Radius kann Umfang ($U = 2\Pi r$) berechnet werden → Methode



# Klasse Kreis

```
public class Kreis {
        public static final double PI =  3.14159;      // Klassenattribut

        public static double radInGrad(double rads) { // Klassenmethode
                return rads * 180 / PI;
        }

        public double radius;                                  // Instanzattribut
        public double flaeche() {                              // Instanzmethode
                return PI * radius * radius;
        }
        public double umfang() {                               // Instanzmethode
                return 2 * PI * radius;
        }
}
```

# Modifikatoren für Datenelemente

⇒ spezifizieren spezielle Eigenschaften von Feldern und Variablen, die in Methoden benutzt werden

- **final:** konstante Variable (Wert nicht veränderbar)
- **static:** Klassenattribut, gehört zur Klasse, in der es definiert ist und nicht zu einem bestimmten Objekt der Klasse (existiert nur einmal pro Klasse, nicht pro Objekt)

# Methoden

⇒ definieren das Verhalten einer Klasse

⇒ **Aufruf:** $objektName.methodenName(Parameter)$

⇒ **Definition:**

| <Modifikator> | <Rückgabetyp> | <methodenName> | (Typ param1,...,Typ paramN) |
|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ |
| zusätzliche Eigenschaften (z.B. Zugriff auf die Methode ) | Datentyp des Wertes, den die Methode zurück gibt (**void:** wenn sie nichts zurückgibt) | Name, mit dem die Methode aufgerufen wird | Paramater, die an die Methode übergeben werden |

# Modifikatoren für Methoden

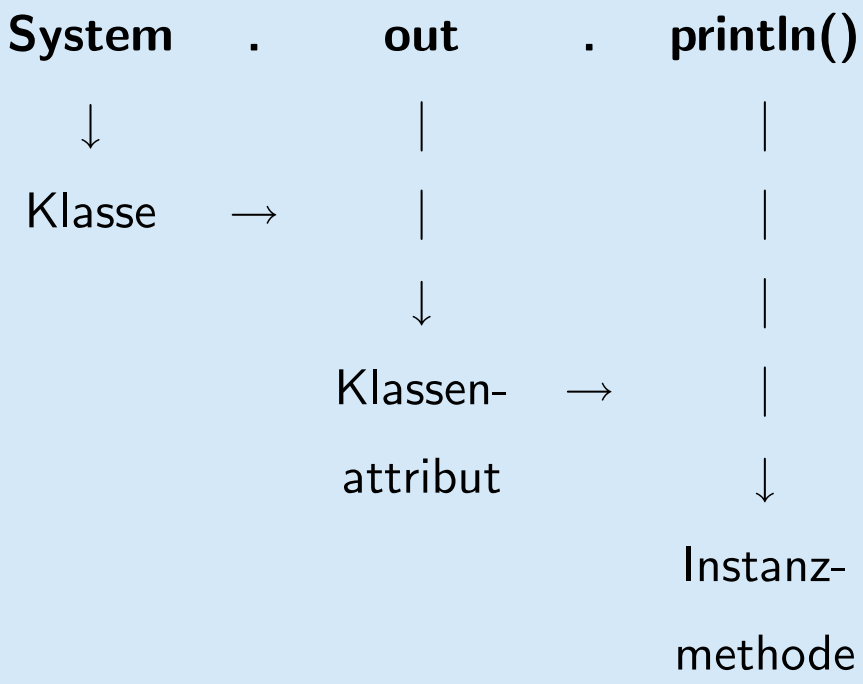
$\Rightarrow$ spezifizieren die Eigenschaften der Methode

- **final:** Methode kann nicht überschrieben werden
- **static:** Klassenmethode kann alle Klassenattribute und Klassenmethoden verwenden, aber keine Instanzattribute oder Instanzmethoden
  (**Aufruf:** $Klassenname.methodenName$)

---

- **native:** kennzeichnet eine Methode, die in einer anderen Progammiersprache geschrieben ist

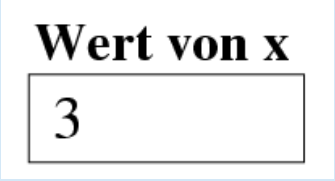
# Auflösung eines Geheimnisses

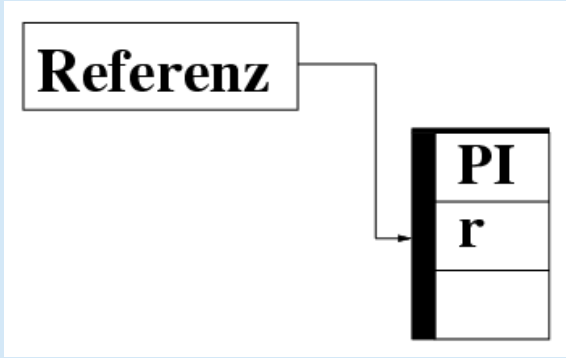



Weitere Infos **API:** java.lang.System

# Parameterübergabe (Wiederholung)

| **call-by-value** | **call-by-reference** |
|---|---|
| Wert/Inhalt der Variable an Methode übergeben | Referenz/Speicherplatzposition übergeben |
| primitive Datentypen | Objekte |
| **Beispiel:** | |
| int x; | Kreis k; |



# Konstruktoren

⇒ Objekte einer Klasse erzeugen und initialisieren

$$<modifier>\ ClassName(Typ\ param1,\ ...,\ Typ2\ paramN)$$

⇒ standardmäßig "leerer" Konstruktor definiert

$$public\ ClassName()\ \{\}$$

⇒ ABER: Wenn irgendein Konstruktor definiert wurde, ist der "leere" Konstruktor nicht mehr automatisch definiert.

⇒ Konstruktoren sind nicht vererbbar

⇒ ABER: Aufruf der Konstruktoren der Oberklasse in Kontruktoren der Unterklasse.

**Expliziter Aufruf:** $super(...)$

# Konstruktoren - Beispiel

```
Kreis() {                              // eigener leerer Konstruktor
        this.radius = 0;
}

Kreis(double radius) {                  // Konstruktor (call-by-value)
        this.radius = radius;
}

Kreis(Kreis kreis2) {                  // Konstruktor (call-by-reference)
        this(kreis2.radius);
}
```

→ **POLYMORPHISMUS - Überladen von Methoden**

# “this”

⇒ spricht Attribute und Konstruktoren innerhalb der Klasse an

⇒ wird immer automatisch durch Compiler benutzt

⇒ muss explizit in folgenden Fällen angewendet werden:

- Aufruf eines Konstruktors innerhalb eines anderen Konstruktors

```
Kreis() {
        this(0); // ruft den Konstruktor Kreis(double radius) a
}
```

- Ein Parametername entspricht einem Attributnamen in der Klasse

```
Kreis(double radius) {
        this.radius = radius;
}
```

# Erzeugen neuer Objekte

⇒ **new:** new <ClassName> (parameter1, ..., parameterN)

⇒ **Beispiele:**

```
Kreis kreis1 = new Kreis();

Kreis kreis2 = new Kreis(12.0);

Kreis kreis3 = new Kreis(kreis2);
```

# Beispiel - Geometrische Punkte

```
class Punkt {
    int x,y;
    Punkt(int x, int y) {  // Konstruktor
        this.x = x;
        this.y = y;
    }
    void verschiebe(int dx, int dy) {
        x += dx;
        y += dy;
    }
    void drucke(String s) {
        System.out.println("Punkt "+s+"("+x+", "+y+")");
    }
    boolean equals(Punkt p) {
        return ((x == p.x) && (y == p.y));
    } }
```

# Beispiel - Geometrische Punkte (2)

```
class PunktDemo {
    public static void main(String[] args) {
        Punkt p1 = new Punkt(3, 65);
        Punkt p2 = new Punkt(19, 68);

        p1.drucke("p1");
        p2.drucke("p2");

        System.out.println("p1==p2? "+(p1==p2));

        p1.verschiebe(16, 3);
        p1.drucke("p1");
        p2.drucke("p2");
        System.out.println("p1==p2? "+(p1==p2));
        System.out.println("p1.equals(p2)? "+p1.equals(p2));
    } }
```

# 3.1. Übung

Benutzen Sie die Klasse Punkt und definieren Sie eine Klasse Rechteck, die aus zwei Punkten und den folgenden Methoden besteht:

- einen Konstrukter mit den zwei Punkten als Parametern
- einen leeren Konstrukter, der den oberen aufruft (alle Parameter = 0)
- eine Methode zum Verschieben des Rechtecks
- eine Methode zum Überprüfen, ob ein Punkt innerhalb des Rechtecks liegt
- eine Methode zum Berechnen und Zurückgeben der Schnittmenge dieses Rechtecks mit einem zweiten (Schnittmenge ist wieder ein Rechteck)
- eine Methode toString(), die die Punkte des Rechtecks als String zurückgibt