Objekt-orientierte Programmierung - Vererbung und Polymorphie

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de, Raum 149B
Version: 1.0

Folien: https://delors.github.io/prog-java-oo-inheritance/folien.de.rst.html

https://delors.github.io/prog-java-oo-inheritance/folien.de.rst.html.pdf

Fehler melden:

https://github.com/Delors/delors.github.io/issues

1. Fortgeschrittene Objekt-orientierte Programmierung mit Java

Generalisierung und Spezialisierung

Implementierung einer Raumverwaltung

Ein Vorlesungsraum an der DHBW

```
public class Vorlesungsraum {
    private int zustand; // 0: nutzbar,
                         // 1: zu renovieren
    private String gebaeude;
    private int ebene;
    private int raumNummer;
    private int kapazitaet;
    private boolean klimatisiert;
    private Ausstattung[] ausstattung;

    public void setzeZustand(int zustand)...
    public void setzeEbene(int ebene)...
    public void setzeRaum(int raumNr)...
    public void generiereBeschreibung()...

    public void reserviere()...
}
```

Eine Teeküche an der DHBW

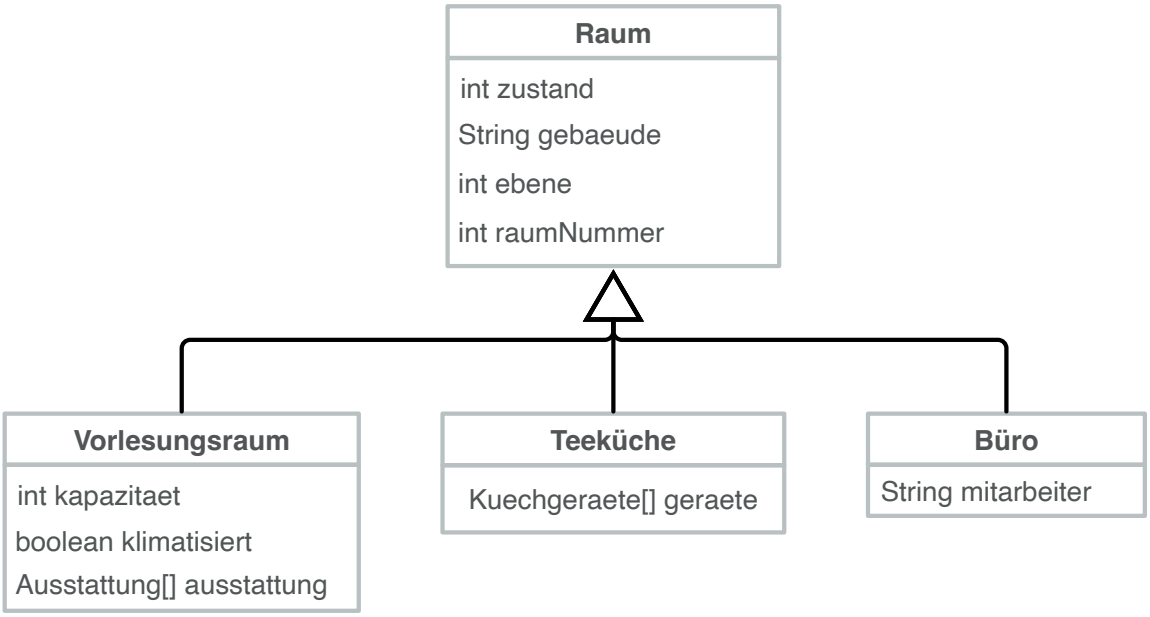
```
public class Teekueche {
    private int zustand; // 0: nutzbar,
                         // 1: zu renovieren
    private String gebaeude;
    private int ebene;
    private int raumNummer;
    private Kuechgeraete[] geraete;



    public void setzeZustand(int zustand)...
    public void setzeEbene(int ebene)...
    public void setzeRaum(int raumNr)...
    public void generiereBeschreibung()...

    public void setzeSchliessberechtigung()...
}
```

Identifikation der Gemeinsamkeiten und Modellierung einer allgemeinen Klasse



Klassen können durch eine Vererbungshierachie in Oberklassen (Superklassen) (hier: `Raum`) und Unterklassen (Subklassen) (hier: `Vorlesungsraum`, `Buero`, `Teekueche`, ...) eingeteilt werden.

Unterklassen *spezialisieren* eine Oberklasse: Die Oberklasse definiert gemeinsame Attribute und Methoden. Eine Unterklasse kann neue Attribute und Methoden hinzufügen bzw. überschreiben. Dabei ist darauf zu achten, dass die Unterklasse sich

Abstraktion (Abstraction)

Definition:

Abstraktion bedeutet, die wesentlichen Eigenschaften und Funktionen eines Objekts hervorzuheben und Details zu verstecken, die für die Nutzung des Objekts nicht relevant sind.

Ziel: Details und Komplexität verstecken; d. h. wir möchten von unnötigen Details abstrahieren.

Beispiel: Eine *Form*-Klasse, die über verschiedene Unterklassen wie *Kreis*, *Quadrat* und *Dreieck* abstrahiert. Alle Formen bieten eine Möglichkeit zur Berechnung der Fläche obwohl diese ggf. sehr verschieden berechnet wird.

```
abstract class Form {
    abstract double berechneFlaeche();
}
```

```
class Kreis extends Form {
    double r = 0.0;
    double berechneFlaeche() {
        return Math.PI * r * r;
    }
}
```

```
class Quadrat extends Form {
    double seite = 0.0;
    double berechneFlaeche() {
        return seite * seite;
    }
}
```

Vererbung (🇺🇸 *Inheritance*)

Definition:

Erlaubt es, eine Klasse von einer anderen abzuleiten und deren Eigenschaften und Methoden zu erben.

Vorteile:

- Wiederverwendbarkeit des Codes
- Hierarchische Strukturierung

Beispiel: *Auto* als Basisklasse und *Elektroauto* als abgeleitete Klasse

```java
class Auto {
    String marke;

    void fahren() {
        System.out.println("Das Auto fährt.");
    }
}

class Elektroauto extends Auto {
    int batteriestand;
```

Polymorphie (Polymorphism)

Definition:
Fähigkeit von Objekten, verschiedene Formen anzunehmen.

Typen:
- **Überladen** von Methoden (🇺🇸 *Compile-Time Polymorphism*)
- **Überschreiben** von Methoden (🇺🇸 *Runtime Polymorphism`*)

Vorteil: Ermöglicht flexiblen und dynamischen Code

Beispiel: Methode *fahren* wird in verschiedenen Klassen unterschiedlich implementiert.

```
class Auto {
    void fahren() {
        System.out.println("Das Auto fährt.");
    }
}

class Elektroauto extends Auto {
    void fahren() { // Überschreiben der Methode
        System.out.println("Das Elektroauto fährt leise.");
```

Zusammenfassung und Vorteile von Objekt-orientierter Programmierung[1]

Kapselung:

Schützt die Daten und kontrolliert den Zugriff.

Abstraktion:

Vereinfacht die Komplexität des Codes.

Vererbung:

Ermöglicht Code-Wiederverwendung und Hierarchien.

Polymorphie:

Erlaubt flexiblen Code durch unterschiedliche Implementierungen.

[1] Diese Vorteile gelten im Wesentlichen für alle objektorientierten Programmiersprachen.

Übung

Meine Erste Klassenhierarchie

Erstelle eine einfache *Tier*-Klasse mit einer Methode *lautGeben()*. Erstelle dann die Klassen *Hund* und *Katze*, die *Tier* erweitern, und überschreibe die Methode *lautGeben()* mit unterschiedlichen Ausgaben.

Meine Erste Klassenhierarchie

Erstelle eine einfache *Tier*-Klasse mit einer Methode *lautGeben()*. Erstelle dann die Klassen *Hund* und *Katze*, die *Tier* erweitern, und überschreibe die Methode *lautGeben()* mit unterschiedlichen Ausgaben.

Kapselung (🇺🇸 *Encapsulation*)[#]

Ziel: Daten eines Objekts vor direktem Zugriff von außen schützen.

Zugriff auf Daten erfolgt über **Getter** und **Setter**.

Vorteile:
- Schutz der Datenintegrität
- Kontrollierter Zugriff auf die Daten; fördert die Wartbarkeit

```
class Auto {
    private int geschwindigkeit;

    public int getGeschwindigkeit() {
        return geschwindigkeit;
    }

    public void setGeschwindigkeit(int geschwindigkeit) {
        if (geschwindigkeit >= 0) {
            this.geschwindigkeit = geschwindigkeit;
        }
    }
```

[2] Kapselung dient vor allem dem Programming-in-the-Large. Sprachen wie zum Beispiel Python bieten diesbezüglich zum Beispiel deutlich weniger Konzepte.

Fehlerbehandlung