Karlsruher Institut für Technologie

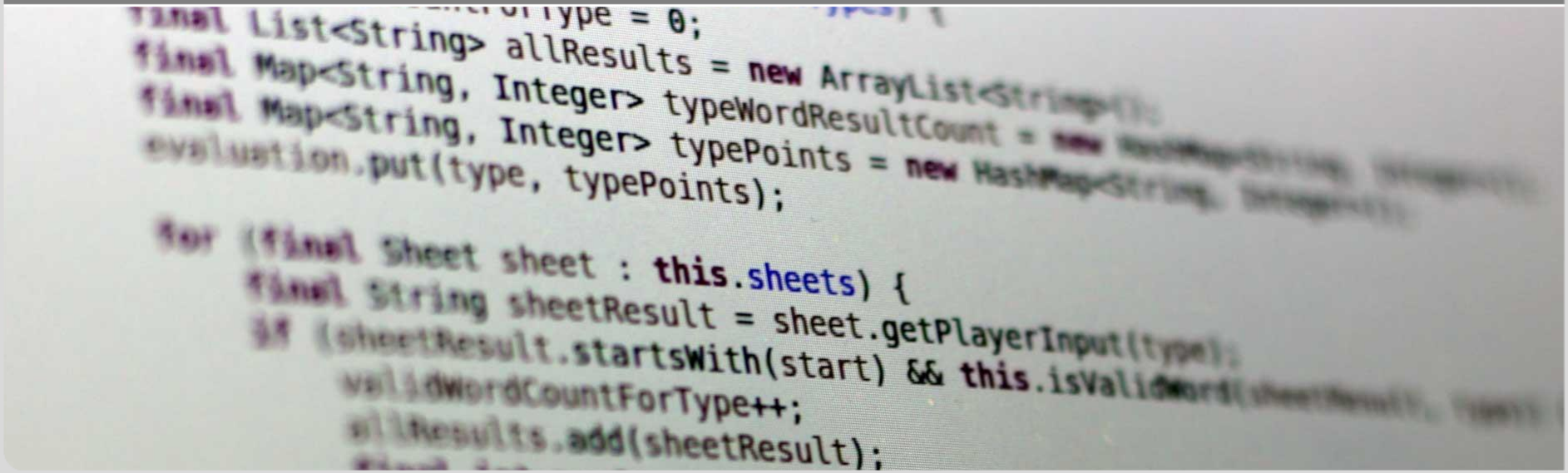
DHBW
Duale Hochschule
Baden-Württemberg

Programmieren I

Überblick „Objektorientierung“

Heusch 12
Ratz 7

Institut für Automation und angewandte Informatik



Objektorientierte Softwarenentwicklung

- Was ist das grundlegende Konzept der Objektorientierung?
  - Ein Programm wird als eine *Ansammlung diskreter Objekte* betrachten, die sowohl Daten als auch Verhalten in sich vereinen.
  - Dies steht im Gegensatz zur konventionellen Programmierung, bei der Datenstruktur und Verhalten nur lose miteinander verbunden sind.
  - Zentrale Eigenschaften des objektorientierten Ansatzes:
    - Klassifikation
    - Identität
    - Kapselung
    - Vererbung/Generalisierung
    - Polymorphismus (auch: Polymorphie; dt. „Vielgestaltigkeit“)

Eigenschaften von Objekten: Klassifikation

- Objekte mit den gleichen Daten (***Attributen***) und dem gleichen Verhalten (***Operationen***) werden zu Objekten einer ***Klasse*** gruppiert.
- Die ***Klasse*** selbst ist eine Abstraktion, welche die Eigenschaften beschreibt, die für eine Anwendung wichtig sind, und den Rest ignoriert. Die Wahl von Klassen ist immer beliebig und hängt von der Anwendung ab.
- Jede ***Klasse*** beschreibt eine möglicherweise unendliche Menge individueller Objekte. Jedes Objekt wird als eine ***Instanz*** seiner Klasse bezeichnet.
- Jede ***Instanz*** der Klasse besitzt eigene Werte für alle ihre Attribute, während sie die Attributnamen und Operationen mit anderen Instanzen der Klasse teilt.

Klasse in Java: Grundlegendes

- Eine Klasse besteht aus:
  - Attributen (Daten)
  - Methoden
- Die Attribute beschreiben die Daten (Eigenschaften) eines Objektes der Klasse.
- Die Methoden beschreiben das Verhalten von Objekten der Klasse.
- Eine Klasse (class) in Java ist ein benutzerdefinierter (Daten-)Typ.
- Eine Variable, deren Typ eine Klasse ist, heißt Objektvariable oder Instanzvariable.

Klasse: Beispiel

Deklaration der Klasse

```
class Cat {
    // Attribute = Daten
    int age;

    // Methoden = Verhalten
    int getAge() {
        return age;
    }

    void setAge(int a) {
        age = a;
    }

    void meow() {
        System.out.println("Meow!");
    }
}
```

Deklaration von Objektvariablen und Erzeugen von Objekten (Instanzen) der Klasse

```
public static void main(String args[]) {
    Cat frisky = new Cat();
    frisky.setAge(5);
    frisky.meow();

    Cat tony;
    tony = new Cat();
    tony.setAge(6);
    tony.meow();
}
```

Klasse: Konstruktoren

```java
class Cat {
    // Attribute = Daten
    int age;

    // Konstruktoren
    Cat() {
    }

    Cat(int a) {
        age = a;
    }

    // Methoden = Verhalten
    int getAge() { return age; }
    void setAge(int a) { age = a; }

    void meow() {
        System.out.println("Meow!");
    }
}
```

```java
public static void main(String args[]) {
    Cat frisky = new Cat();
    // via setter-Methode
    frisky.setAge(5);
    frisky.meow();

    Cat tony;
    // via Konstruktor
    tony = new Cat(6);
    tony.meow();
}
```

Eigenschaften von Objekten: Identität

- **Identität** heißt, dass Daten diskreten, unterscheidbaren Entitäten - Objekten - zugeordnet werden. Beispiele:
  - Absatz in einem Dokument
  - Fenster auf einer Workstation
  - Figur in einem Schachspiel
- Jedes Objekt besitzt eine eigene **inhärente** ("ihm innewohnende") **Identität**.
- Mit anderen Worten, zwei Objekte sind klar voneinander unterscheidbar, selbst wenn alle ihre Attributwerte (wie Name oder Größe) identisch sind.

Identität: Beispiel

Deklaration von Objekten der Klasse `Cat`

```
public static void main(String args[]) {
    Cat frisky = new Cat();
    frisky.setAge(5);
    frisky.meow();

    Cat tony;
    tony = new Cat(6);
    tony.meow();

    Cat lulu = new Cat();
    lulu.setAge(5);
    lulu.meow();
}
```

Eigenschaften von Objekten: Kapselung

- Unter **Kapselung** versteht man das Verbergen von Daten oder Informationen vor dem Zugriff von außen.
- Der direkte Zugriff auf interne Daten wird unterbunden und erfolgt statt dessen über definierte Schnittstellen (auch bekannt als „Black-Box-Modell“).
- Mit der Kapselung ist in Java eng die Verwendung von *Modifikatoren* verbunden.

Heusch 12.7
Ratz 7.3.3

Kapselung: Beispiel (1)

```
public class Fraction {

    public int numerator = 0;       // public --> Variablen nach
    public int denominator = 1;     // außen sichtbar

}
```

Kapselung: Beispiel (2)

```
public class Fraction {

    private int numerator = 0;   // private --> Variablen nicht
    private int denominator = 1; // nach außen sichtbar

    /**
     * Setzen des Nenners (divisors). Bei Übergabe
     * von divisor==0 wird der Nenner nicht gesetzt
     */
    public void setDenominator(int divisor) {
        if (divisor != 0) {      // Fehlervermeidung
            denominator = divisor;
        }
    }

    public int getDenominator() {
        return denominator;
    }

}
```

Klasse: Eigenschaften und Verhalten

Attribute (Eigenschaften)

```
class Cat {
    int age;

    Cat() {
    }

    Cat(int a) {
        age = a;
    }

    int getAge() { return age; }
    void setAge(int a) { age = a; }

    void meow() {
        System.out.println("Meow!");
    }
}
```

```
class Dog {
    int age;

    Dog() {
    }

    Dog(int a) {
        age = a;
    }

    int getAge() { return age; }
    void setAge(int a) { age = a; }

    void bark() {
        System.out.println("Woof!");
    }
}
```

Methoden (Verhalten)

Eigenschaften von Objekten: Vererbung

- Es können sehr allgemeine Klassen definiert werden und diese dann in immer detailliertere Unterklassen verfeinert werden.
- Jede Unterklasse übernimmt oder erbt alle Attribute (Eigenschaften) und Methoden ihrer Oberklasse
- Daneben fügt sie ihre eigenen individuellen Attribute und Methoden hinzu.

- Aber: Konstruktoren werden nicht vererbt.

Beispiel: Vererbung (1) - Klassen-Deklaration

Oberklasse

```
class Animal {
    int age;

    Animal() { }
    Animal(int a) { age = a; }

    int getAge() { return age; }
    void setAge(int a) { age = a; }
}
```

Unterklassen

```
class Cat extends Animal {
    void meow() {
        System.out.println("Meow!");
    }
}
```

```
class Dog extends Animal {
    void bark() {
        System.out.println("Woof!");
    }
}
```

Beispiel: Vererbung (2) - Objekte

Deklaration von Objektvariablen und Erzeugen von Objekten der Klassen `Animal`, `Cat` und `Dog`

```
public static void main(String args[]){
    Cat frisky = new Cat();
    frisky.setAge(5);
    frisky.meow();

    Dog pluto = new Dog();
    pluto.setAge(5);
    pluto.bark();

    Animal hugo = new Animal();
    hugo.setAge(4);
}
```

Das geht nicht:

```
public static void main(String args[]){
    Cat frisky = new Cat(5);
    Dog pluto;
    pluto = new Dog(4);
    Animal hugo = new Animal(4);
    hugo.bark();
}
```

Eigenschaften von Objekten: Polymorphismus

- Polymorphismus ist ein weiteres Konzept der objektorientierten Software-Technologie.
  - Möglichkeit Methoden mit gleichen Namen für unterschiedliche Objekte zu implementieren.
  - Das objektorientierte Softwaresystem sucht sich zur Laufzeit, abhängig vom Typ des Objektes, die richtige Methode aus.
- Überladung ist kein Polymorphismus.
- Überladung:
  - Der Typ der Argumente entscheidet über die aufgerufene Funktion/Methode. Überladung wird immer zur Übersetzungszeit aufgelöst.
- Polymorphismus:
  - Der Typ des Objekts entscheidet
    - zur Laufzeit - über die aufgerufene Methode.

Heusch 12.3.4
Ratz 7.3.4

Beispiel für Überladung

```
class Bird {

    void move() {
        System.out.println("I fly up in the air: flutter flutter!");
    }

    void move(String parameter) {
        System.out.println("I fly up in the air: " + parameter);
    }

    public static void main(String[] args) {
        Bird b = new Bird();
        b.move();
        b.move("glide glide!");
    }
}
```

```
> java Bird
I fly up in the air: flutter flutter!
I fly up in the air: glide glide!
```

Beispiel Polymorphismus (1)

```
class Bird {
    void move(){
        System.out.println("I fly up in the air: flutter flutter!");
    }
}
```

```
class Songbird extends Bird {

}
```

```
class Penguin extends Bird {
    void move() {
        System.out.println("I swim in the water: float float!");
    }
}
```

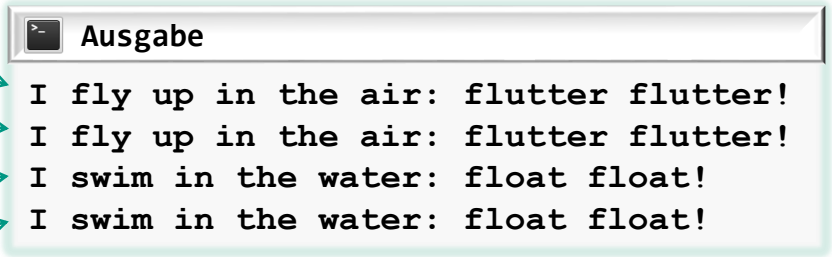
Beispiel Polymorphismus (2)

```
public static void main(String[] args) {
    Bird b = new Bird();
    b.move();

    Songbird s = new Songbird();
    s.move();

    Penguin p = new Penguin();
    p.move();

    Bird b2;
    b2 = p;          // zulässig
    b2.move();       // Polymorphismus
}
```



Mehrfachvererbung

- In Java gibt es keine echte Mehrfachvererbung
- Über Schnittstellen (Interfaces) ist eine Art Mehrfachvererbung möglich
- Das folgende Beispiel aus C++ zeigt eine mögliche Mehrfachvererbung

Mehrfachvererbung in C++

```
class Bird
{
  public:
    void move()
    {
        cout<<"I fly up in the air: flutter!";
    };
};
```

```
class Cat
{
   private:
     int age;
   public:
     int getAge();
     void setAge(int a);
     void meow();
};
```

```
class FlyingCat: public Cat, public Bird
{
};
```

```
int main()
{
    FlyingCat superCat;
    superCat.setAge(5);
    superCat.meow();
    superCat.move();
    return 0;
};
```

Objekt wird implizit bei der Deklaration erzeugt

Zusammenfassung

- Erläuterung der Begriffe:
  - Klassifikation
  - Identität
  - Kapselung
  - Vererbung/Generalisierung
  - Polymorphismus
- Grober Überblick über den Klassenbegriff in Java