



Objektorientierung 3: Vererbung

Vorkurs Informatik



Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

(nach Folien von L. Vettin, V. Weidler, W. Kessler)  CC Attribution-NonCommercial-ShareAlike 4.0 Licence

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Rückblick: Gelernte Konzepte

- ▶ Klasse, Objekt
- ▶ Attribut
- ▶ Methode
- ▶ Rückgabetyp, Rückgabewert
- ▶ Parameter
- ▶ Konstruktor
- ▶ Getter/Setter
- ▶ Klassendokumentation
- ▶ Statische Methode

Rückblick: Gelernte Konzepte

- ▶ Eine **Klasse** ist ein Bauplan/eine Schablone für eine bestimmte Art von Objekten.
- ▶ Ein **Objekt** ist ein konkretes Exemplar einer bestimmten Klasse.
- ▶ **Attribute** dienen zum Speichern von Informationen über ein Objekt. Sie haben einen Namen und einen Datentyp.
- ▶ Der **Konstruktor** einer Klasse wird jedes Mal ausgeführt, wenn ein Objekt dieser Klasse erzeugt wird.
- ▶ Objekte haben **Methoden**, bei deren Aufruf eine Aktion ausgeführt wird.
- ▶ Die **Methodensignatur** gibt Auskunft über Namen, Rückgabetyp, und Parameter einer Methode.

Rückblick

- Machen Sie sich klar, was bei dem nachfolgenden Programm passiert.

Rückblick

Datei Raccoon.java:

```
public class Raccoon {  
  
    public Color color;  
  
    public Raccoon() {  
        this.color = Color.GRAY;  
    }  
  
    public Raccoon(Color color) {  
        this.color = color;  
    }  
  
    public void sleep() {  
        System.out.println("I'm sleeping!");  
    }  
  
    public Color getColor() {  
        return this.color;  
    }  
}
```

Datei Color.java:

```
public enum Color {  
    GRAY, ORANGE, BLACK;  
}
```

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        Raccoon fluffy =  
            new Raccoon();  
        Raccoon rocket =  
            new Raccoon(Color.ORANGE);  
  
        fluffy.sleep();  
        System.out.println  
            (rocket.getColor());  
    }  
}
```

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Vererbung

- ▶ Klassen in der echten Welt besitzen eine Vielzahl an Beziehungen. Eine wichtige Beziehung ist die „ist eine“ Beziehung. Diese würden wir gerne in unserer Software modellieren, um sie verwenden zu können.

Vererbung

- Klassen in der echten Welt besitzen eine Vielzahl an Beziehungen. Eine wichtige Beziehung ist die „ist eine“ Beziehung. Diese würden wir gerne in unserer Software modellieren, um sie verwenden zu können. Beispiel:



Säugetier >



Hamster



- Man spricht von Abstraktion und Spezialisierung. Diese werden durch das Konzept der Vererbung umgesetzt.

https://de.wikipedia.org/wiki/Datei:Animal_diversity.png

https://de.wikipedia.org/wiki/Datei:Mammal_Diversity_2011.png

<https://pixabay.com/de/photos/nagetier-hamster-zwerghamster-mini-3625950/>

Vererbung

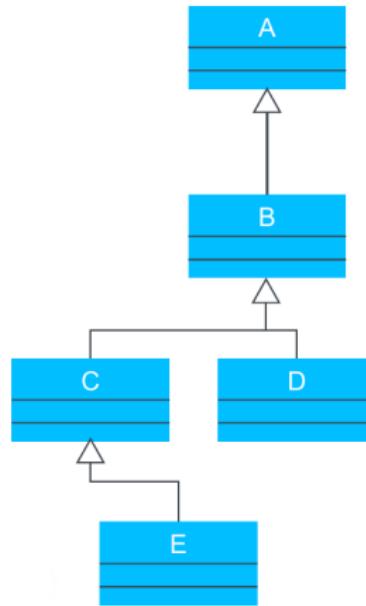
Vererbung

Vererbung erlaubt es Programmierern, Klassen zu erstellen, die auf existierenden Klassen basieren, eine neue Realisierung zu spezifizieren, während das Verhalten beibehalten wird, um Code wiederzuverwenden.

- ▶ Andere Beispiele, bei denen Vererbung sinnvoll sein könnte:
 - ▶ Fahrzeuge: Autos, Busse, Fahrräder, ...
 - ▶ Möbel: Schrank, Sofa, Regal, ...
 - ▶ ...

Terminologie

- ▶ A ist eine Elternklasse / Superklasse von B
- ▶ B ist eine Kindklasse / Subklasse von A



Terminologie

- Erbt man von einer anderen Klasse, benötigt man in Java das Stichwort **extends**.

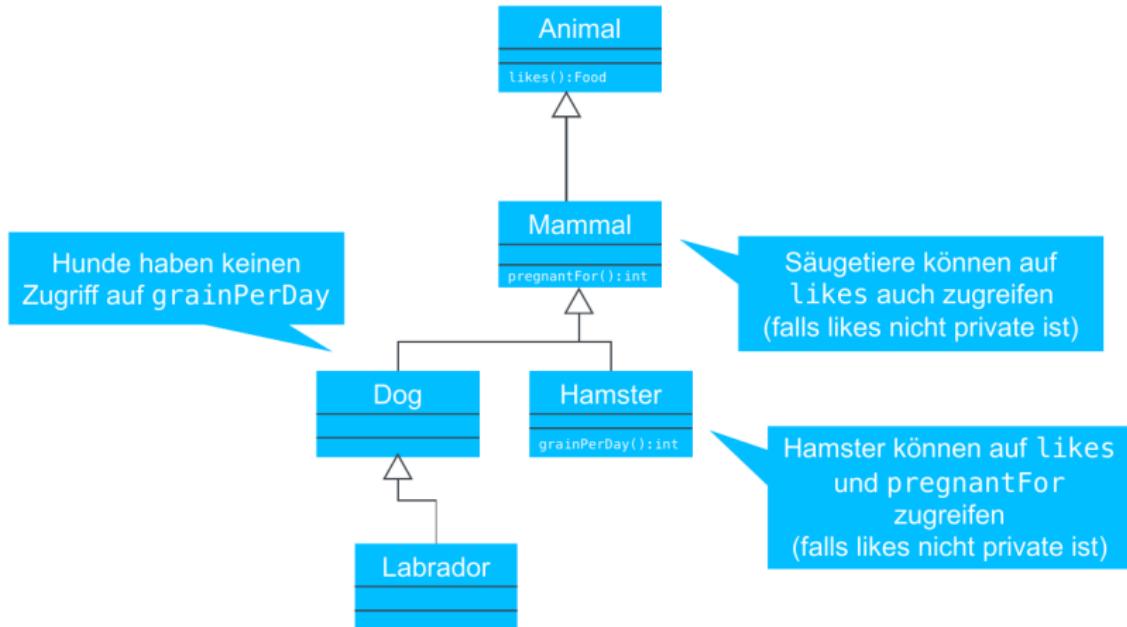
```
public class Animal {  
    ...  
}
```

```
public class Mammal extends Animal {  
    ...  
}
```

```
public class Hamster extends Mammal {  
    ...  
}
```

Klassisches Beispiel - Teil 1

Wir wollen die Hierarchie der Tiere durch eine Hierarchie von Klassen darstellen:



Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Abstraktion

Problem:

- ▶ Manche Klassen können keine Implementierung ihrer Operationen anbieten.
- ▶ Beispiel: `likes()` hängt vollständig von der Art des Tieres ab.
Beim Hamster hätte man Körner, beim Hund Würstchen.

Abstraktion

Problem:

- ▶ Manche Klassen können keine Implementierung ihrer Operationen anbieten.
- ▶ Beispiel: `likes()` hängt vollständig von der Art des Tieres ab.
Beim Hamster hätte man Körner, beim Hund Würstchen.

Lösung:

- ▶ Deklariere eine Operation, die nur aus der Spezifikation besteht.
- ▶ Man spricht hierbei von einer abstrakten Operation.

Abstraktion

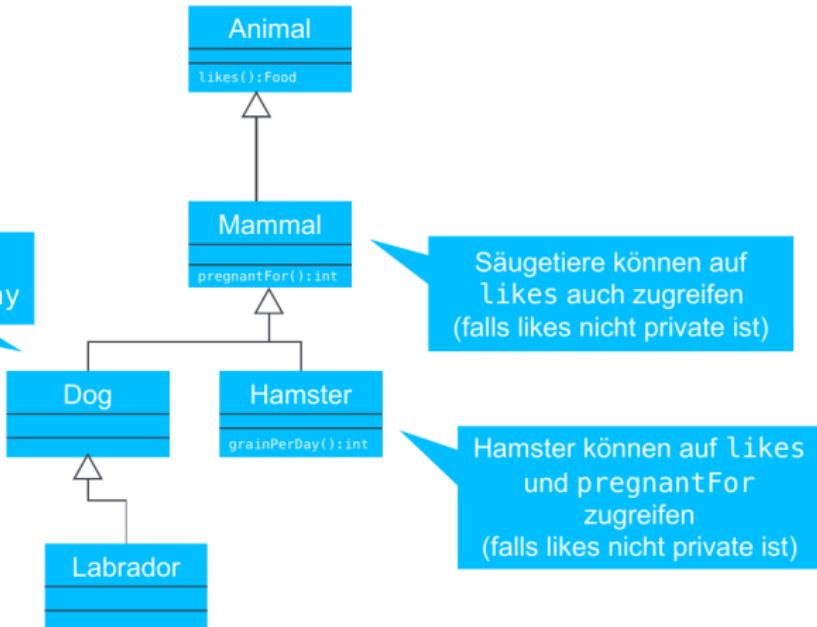
- ▶ In Java verwendet man das Schlüsselwort `abstract` als Operationsmodifizierer, um solch eine Operation zu deklarieren.
- ▶ Beispiel: `public abstract Food likes();`
- ▶ Abstrakte Operationen besitzen keinen Operationsrumpf und dürfen nicht `private` sein.

Abstraktion

- ▶ In Java verwendet man das Schlüsselwort `abstract` als Operationsmodifizierer, um solch eine Operation zu deklarieren.
 - ▶ Beispiel: `public abstract Food likes();`
 - ▶ Abstrakte Operationen besitzen keinen Operationsrumpf und dürfen nicht `private` sein.
-
- ▶ Von einer Klasse mit abstrakten Operationen können keine Objekte erstellt werden.
 - ▶ Die Klasse muss deshalb ebenfalls als abstrakt deklariert werden.
 - ▶ Beispiel: `abstract class Animal`
 - ▶ In den Unterklassen müssen dann die Operationen konkret implementiert werden.
 - ▶ Dann fügt man in der Unterklasse die Annotation `@Override` hinzu.

Klassisches Beispiel - Teil 2

Wir wollen folgende Klassen implementieren und ergänzen ein Attribut String name:



Klassisches Beispiel - Teil 2

```
public abstract class Animal {  
  
    private String name;  
  
    public Animal(String name) {  
        this.name = name;  
    }  
  
    public abstract Food likes();  
}
```

```
public abstract class Mammal  
extends Animal {  
  
    public Mammal(String name) {  
        super(name);  
    }  
  
    public abstract int pregnantFor();  
}
```

```
public class Dog  
extends Mammal {  
  
    public Dog(String name) {  
        super(name);  
    }  
  
    @Override  
    public Food likes() {  
        return Food.MEAT;  
    }  
  
    @Override  
    public int pregnantFor() {  
        return 60;  
    }  
}
```

Klassisches Beispiel - Teil 2

- ▶ Konstruktoren werden nicht vererbt.
- ▶ Um das geerbte Attribut initialisieren zu können, muss der Konstruktor der Kindklasse den Konstruktor der Elternklasse mittels `super` aufrufen.
- ▶ Jede Kindklasse kann zusätzlich noch eigene Attribute besitzen und Operationen implementieren.

Klassisches Beispiel - Teil 2

Klasse Hamster:

```
public class Hamster extends Mammal {  
  
    public Hamster(String name) {  
        super(name);  
    }  
  
    @Override  
    public Food likes() {  
        return Food.GRAIN;  
    }  
  
    @Override  
    public int pregnantFor() {  
        return 21;  
    }  
  
    public int grainPerDay() {  
        return 80;  
    }  
}
```

Klassisches Beispiel - Teil 2

Objekte instanziieren:

```
public class Test {  
    public static void main(String[] args) {  
  
        Animal barni = new Animal("barni"); // illegal!  
  
        Dog bella = new Dog("bella");  
        Hamster paula = new Hamster("paula");  
  
        System.out.println(bella.pregnantFor());  
        System.out.println(paula.pregnantFor());  
    }  
}
```

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Konstruktoren

- ▶ Konstruktoren werden grundsätzlich nicht an die Kindklasse vererbt. Sie können jedoch – und manchmal müssen sie sogar – aufgerufen werden.
- ▶ Man verwendet `super(arg1, ..., argn)`, um einen Konstruktor der Elternklasse aufzurufen.

Konstruktoren

```
public abstract class Animal {  
  
    private String name;  
  
    public Animal(String name) {  
        this.name = name;  
    }  
  
    public abstract Food likes();  
}
```

```
public class Dog extends Mammal {  
  
    public Color color;  
  
    public Dog(String name, Color color) {  
        super(name);  
        this.color = color;  
    }  
  
    ...  
}
```

Konstruktoren

- ▶ Mit `this(arg1, ..., argn)` lässt sich von einem Konstruktor innerhalb einer Klasse ein anderer Konstruktor aufrufen.
- ▶ Falls kein Standardkonstruktor in der Elternklasse vorhanden ist, muss der Aufruf explizit erfolgen.

```
public class Dog extends Mammal {  
  
    public Color color;  
  
    public Dog(String name, Color color) {  
        super(name);  
        this.color = color;  
    }  
  
    public Dog(String name) {  
        this(name,Color.BLACK);  
    }  
  
    ...  
}
```

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Überschreiben von Operationen

- Manchmal existiert eine Implementierung einer Operation der Elternklasse, aber sie könnte in der Kindklasse zweckmäßiger implementiert werden.

Überschreiben von Operationen

Beispiel:

- ▶ In Java erweitern alle Klassen die Klasse `Object` direkt oder indirekt, auch wenn dies nicht durch `extends` spezifiziert wird.
- ▶ Durch `Object` werden einige Operationen angeboten, z.B.
 - ▶ `public String toString()`
„Returns a string representation of the object. In general, the `toString` method returns a String that textually represents this object.“
 - ▶ `public boolean equals(Object obj)`
„Indicates whether some other object is equal to this one. The `equals` method implements an equivalence relation on non-null object references.“

Überschreiben von Operationen

- ▶ Standardimplementierungen:

```
public class Object {  
    [...]  
  
    public String toString() {  
        return getClass().getName() + "@" +  
            Integer.toHexString(hashCode());  
    }  
  
    public boolean equals(Object obj) {  
        return (this == obj);  
    }  
    [...]  
}
```

Beispiel

Nachteile:

- ▶ Die Methode `toString` liefert nur einen Hashcode, mit dem der Programmierer in der Regel nichts anfangen kann.
- ▶ Bspw. liefert `System.out.println(paula.toString())` die Ausgabe
`mintpse.animals.Hamster@33909752.`

Beispiel

Nachteile:

- ▶ Die Methode `toString` liefert nur einen Hashcode, mit dem der Programmierer in der Regel nichts anfangen kann.
 - ▶ Bspw. liefert `System.out.println(paula.toString())` die Ausgabe `mintpse.animals.Hamster@33909752`.
- ▶ Die Methode `equals` überprüft lediglich, ob die Objekte identisch sind. Alles andere kann die Methode nicht behandeln.
 - ▶ Erst wenn in den Unterklassen klar ist, wie die Objekte konkret aussehen, kann auch vernünftig implementiert werden, was verglichen werden soll.

Überschreiben

Lösung:

- ▶ Die Standardimplementierungen können in den Unterklassen nach Bedarf überschrieben werden.
- ▶ Für den Kompiler fügt man dabei die Annotation `@Override` hinzu.
- ▶ Falls `@Override` hinzugefügt wurde und die zu überschreibende Operation in keiner Oberklasse existiert, erhält man eine Fehlermeldung.

Überschreiben

- Beispiel:

```
public class Hamster {  
  
    [...]  
  
    @Override  
    public String toString() {  
        return "Hamster [pregnantFor()=" + pregnantFor() +  
               ", likes()=" + likes() + "]";  
    }  
  
}
```

Überschreiben

Voraussetzungen für Überschreiben

- ▶ Namensgleicher Methodenname.
- ▶ Eingabeparameter von Anzahl, Reihenfolge und Typ identisch.
- ▶ Rückgabeparameter identisch.
- ▶ Zugriffsrechte der Methode der Oberklasse werden nicht eingeschränkt.

Überschreiben

- ▶ Manchmal wollen wir den Code aus überschriebenen Operationen wiederverwenden.
- ▶ Wir können auf überschriebene Operationen zugreifen mit

`super .<Methodename>`

Zugriff auf überschriebene Operationen

- Beispiel:

```
public class Hamster {  
  
    [...]  
  
    @Override  
    public String toString() {  
        String result = super.toString();  
        result += "Hamster [pregnantFor()=" + pregnantFor() +  
                  ", likes()=" + likes() + "]";  
        return result;  
    }  
  
}
```

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Überladen

Problem:

- ▶ Häufig hätte man gerne mehrere Operationen mit unterschiedlichen Argumenttypen und Argumentanzahlen, die alle dasselbe Verhalten realisieren.
- ▶ Diese sollten entsprechend auch denselben Namen haben.

Überladen

Problem:

- ▶ Häufig hätte man gerne mehrere Operationen mit unterschiedlichen Argumenttypen und Argumentanzahlen, die alle dasselbe Verhalten realisieren.
- ▶ Diese sollten entsprechend auch denselben Namen haben.

Lösung:

- ▶ In Java gibt es die Möglichkeit, Operationen zu überladen: Operationen oder Konstruktoren mit dem gleichen Namen, aber unterschiedlicher Anzahl oder Typ, werden vom Compiler als unterschiedliche Operationen behandelt.

Beispiel

► Beispiel:

```
public class Dog extends Mammal {  
  
    public Dog() {  
        super("");  
    }  
  
    public Dog(String name) {  
        super(name);  
    }  
  
    ...  
}
```

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Zugriffsmodifikatoren

Zugriffsmodifikatoren

Zugriffsmodifikatoren für Klassen und Methoden geben an, wo etwas sichtbar ist und verwendet werden kann.

public Überall sichtbar.

private Nur innerhalb der eigenen Klasse sichtbar. Kann nicht aus anderen Klassen heraus verwendet werden.

protected Für alle Klassen im selben Paket sichtbar sowie für Unterklassen in anderen Paketen.

Outline

- 1 Rückblick
- 2 Vererbung
- 3 Abstraktion
- 4 Konstruktoren
- 5 Überschreiben von Operationen
- 6 Überladen von Operationen
- 7 Zugriffsmodifikatoren
- 8 Zusammenfassung

Zusammenfassung: Objektorientierung

- ▶ Das Konzept der **Vererbung** erlaubt es, Abstraktion und Spezialisierung umzusetzen.
- ▶ Von einer **abstrakten Klasse** können keine Objekte erzeugt werden, sie dienen nur als Oberklasse für andere Klassen.
- ▶ **Überschreiben** bedeutet, eine Methode gleicher Signatur in der Unterklasse zu erstellen.
- ▶ **Überladen** bedeutet, den gleichen Namen für zwei verschiedene Methoden oder Konstruktoren zu verwenden.
- ▶ Das Schlüsselwort **this** bezieht sich auf das aktuelle Objekt.
- ▶ Das Schlüsselwort **super** ist eine Referenz auf die Oberklasse.
- ▶ **Zugriffsmodifikatoren** geben an, wo etwas verwendet werden kann (**public** = überall, **private** = nur in der eigenen Klasse).



Vielen Dank!



Frank Schweiner

E-Mail frank.schweiner@mint-kolleg.de
Telefon +49 (0) 711 685-84326
Fax —

Universität Stuttgart
MINT-Kolleg Baden-Württemberg
Azenbergstr. 12
70174 Stuttgart