



Universität Stuttgart
MINT-Kolleg Baden-Württemberg



Objektorientierung 2: Attribute, Konstruktoren

Vorkurs Informatik



Universität Stuttgart



Karlsruher Institut für Technologie

F. Schweiner



Getting started ...

Java 20 - Predictions

```
import whocares;
```

whocares keyword was introduced to replace all imports. The compiler decides what to do.

```
#demo #class #instagood
```

Comments were replaced by hashtags.

```
class Java20Class {
```

Access modifiers were removed. Everything is now PUBLIC as imposed by NSA. No further comments allowed.



This is just syntax sugar..



```
(String args[]) {
```

public static void main was replaced by an emoji.

```
helloworld;
```

helloworld keyword was introduced as a shortcut to System.out.println ("Hello World!")

```
}
```

Latin characters were removed after Donald Trump reelection.

```
}
```



I miss the private objects...

Daniel Stori {turnoff.us}

©Daniel Stori, <http://turnoff.us/geek/java20-predictions/>

Outline

- 1 Rückblick
- 2 Attribute und Konstruktoren
- 3 Enumerations
- 4 Klassenmethoden
- 5 Zusammenfassung

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

(und *Programmierung und Softwareentwicklung*, Prof. Dr.-Ing. Steffen Becker und Sandro Speth,
Institute of Software Engineering, Universität Stuttgart)

Outline

- 1 Rückblick
- 2 Attribute und Konstruktoren
- 3 Enumerations
- 4 Klassenmethoden
- 5 Zusammenfassung

Klassen

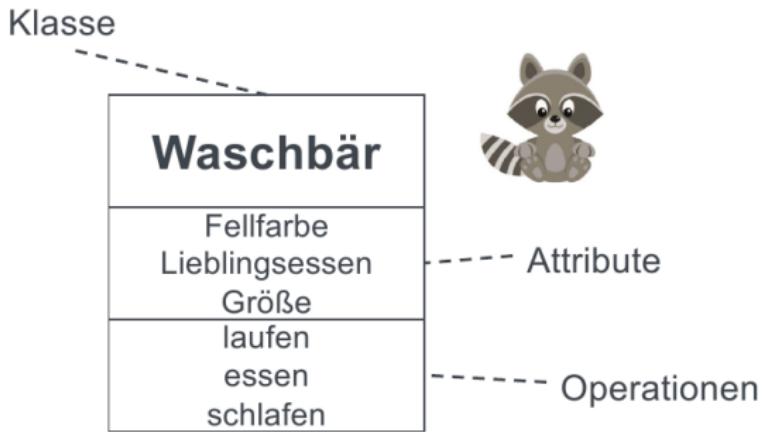
Eine **Klasse** ist eine verallgemeinerte Sammlung von Eigenschaften.

- Sie dient nur als Bauplan von Objekten.

Klassen

Eine **Klasse** ist eine verallgemeinerte Sammlung von Eigenschaften.

- ▶ Sie dient nur als Bauplan von Objekten.
- ▶ Sie definiert für die Objekte u.a.
 - ▶ die Eigenschaften (**Attribute**) und
 - ▶ die Fähigkeiten (**Operationen**)



Vereinfachtes Klassendiagramm

Rückblick

Datei Raccoon.java:

```
public class Raccoon {  
  
    public String color = "gray";  
  
    public void sleep() {  
        System.out.println("I'm sleeping!");  
    }  
  
    public void eat(int times) {  
        for (int i=0; i<times; i++) {  
            System.out.println("I'm eating!");  
        }  
    }  
  
    public String getColor() {  
        return this.color;  
    }  
}
```

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        Raccoon fluffy = new Raccoon();  
  
        fluffy.sleep();  
        fluffy.eat(5);  
        String color =  
            fluffy.getColor();  
    }  
}
```

Outline

- 1 Rückblick
- 2 Attribute und Konstruktoren
- 3 Enumerations
- 4 Klassenmethoden
- 5 Zusammenfassung

Informationen über ein Objekt speichern

Attribute, (Zustands-)Felder, Objektvariablen

Attribute dienen zum Speichern von Informationen über ein Objekt.
Informationen können konkrete Werte oder selbst Objekte sein.

Informationen über ein Objekt speichern

Attribute, (Zustands-)Felder, Objektvariablen

Attribute dienen zum Speichern von Informationen über ein Objekt. Informationen können konkrete Werte oder selbst Objekte sein.

Konstruktor

Der Konstruktor einer Klasse ist eine besondere Art Methode, die automatisch jedes Mal ausgeführt wird, wenn ein Objekt dieser Klasse erzeugt wird. Er kann zum Initialisieren der Attribute verwendet werden.

Informationen über ein Objekt speichern

- ▶ Welche Informationen könnte man über einen Waschbär speichern?
 - ▶ Fellfarbe
 - ▶ Lieblingsessen
 - ▶ Größe
 - ▶ ...
- ▶ Die Werte unterscheiden sich aber evtl. bei verschiedenen Waschbüren!
- ▶ Wir müssen diese Werte individuell festlegen können, wenn wir mit **new** einen neuen Waschbär erzeugen.
- ▶ Hierfür dient der Konstruktor.

Informationen über ein Objekt speichern

- ▶ Welche Informationen könnte man über einen Waschbär speichern?
 - ▶ Fellfarbe
 - ▶ Lieblingsessen
 - ▶ Größe
 - ▶ ...
- ▶ Die Werte unterscheiden sich aber evtl. bei verschiedenen Waschbüren!
- ▶ Wir müssen diese Werte individuell festlegen können, wenn wir mit **new** einen neuen Waschbär erzeugen.
- ▶ Hierfür dient der Konstruktor.
- ▶ Schauen wir uns mal ein vollständiges Beispiel an, bei dem wir nur die Fellfarbe betrachten.
- ▶ Die Details im Programm werden wir dann auf den folgenden Folien klären. Vielleicht sehen Sie direkt, was wo passiert?

Beispiel

Datei Raccoon.java:

```
public class Raccoon {  
  
    public String color;  
  
    // eigener Konstruktor  
    public Raccoon() {  
        this.color = "gray";  
    }  
    // eigener Konstruktor  
    public Raccoon(String color) {  
        this.color = color;  
    }  
  
    public void sleep() {  
        System.out.println("I'm sleeping!");  
    }  
  
    public String getColor() {  
        return this.color;  
    }  
}
```

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        // Hier wird der  
        // Konstruktor  
        // aufgerufen  
        Raccoon fluffy =  
            new Raccoon();  
        Raccoon rocket =  
            new Raccoon("orange");  
  
        fluffy.sleep();  
        System.out.println  
            (rocket.getColor());  
    }  
}
```

Informationen über ein Objekt speichern

Attribute, (Zustands-)Felder, Objektvariablen

Attribute dienen zum Speichern von Informationen über ein Objekt.
Informationen können konkrete Werte oder selbst Objekte sein.

Informationen über ein Objekt speichern

Attribute, (Zustands-)Felder, Objektvariablen

Attribute dienen zum Speichern von Informationen über ein Objekt. Informationen können konkrete Werte oder selbst Objekte sein.

- ▶ Attribute haben einen Namen und einen Datentyp.
- ▶ Attribute können in allen Methoden der Klasse wie Variablen verwendet werden.
- ▶ Verschiedene Objekte können verschiedene Werte in den Attributen haben.

Attribute deklarieren

- ▶ Attribute werden mit Datentyp und Namen deklariert.
- ▶ Konvention: Um Attributdeklarationen schnell finden zu können, werden sie an den Anfang einer Klasse geschrieben.
- ▶ Konvention: Attributnamen werden kleingeschrieben.
- ▶ Syntax: <Modifikator> <Datentyp> <Name> = <Wert>;
oder <Modifikator> <Datentyp> <Name>;

```
public class Raccoon {  
  
    public String color;  
    ...  
}
```

Attribute verwenden

- ▶ Attribute können innerhalb einer Klasse wie Variablen verwendet werden.
- ▶ Davor schreibt man i.d.R. noch `this`:

```
public class Raccoon {  
  
    public String color;  
  
    public void hello() {  
        System.out.println("Hi, my color is " + this.color + ".");  
    }  
}
```

Attribute verwenden

- ▶ Attribute können innerhalb einer Klasse wie Variablen verwendet werden.
- ▶ Davor schreibt man i.d.R. noch `this`:

```
public class Raccoon {  
  
    public String color;  
  
    public void hello() {  
        System.out.println("Hi, my color is " + this.color + ".");  
    }  
}
```

- ▶ Syntax für Zugriff auf ein Attribut von außen:

`<Objektname>. <Attributname>`

```
Raccoon rocket = new Raccoon("orange");  
System.out.println(rocket.color);
```

Zwei Variablen mit demselben Namen

- ▶ Aufpassen: Lokale Variablen oder Parameter dürfen heißen wie Attribute. Verwendungen des Namens beziehen sich dann auf die lokale Variable/den Parameter. Sie "überdecken" das Attribut:

```
public String color;  
  
public Raccoon (String color) {  
    color = color; // ???  
}
```

- Das ist eine sehr schlechte Idee!
- Um Attribute von Parametern zu unterscheiden verwende **this**

```
public String color;  
  
public Raccoon (String color) {  
    this.color = color; // ok  
}
```

Getter und Setter

- ▶ Um falsche Zugriffe und falsches Setzen von Attributen zu vermeiden, empfiehlt es sich, Attribute **private** zu deklarieren.
- ▶ Soll das Lesen eines Attributes erlaubt sein, legt man eine **get**-Methode für dieses Attribut an.
- ▶ Soll das Schreiben erlaubt sein, legt man eine **set**-Methode an, in der bei Bedarf auch z. B. geprüft werden kann, ob ein gültiger Wert zugewiesen wurde.
- ▶ Typisches Beispiel ist ein Bankkonto, bei dem man sich überlegen sollte, was man mit dem Kontostand alles darf:

```
public class Account {  
  
    private int balance; // Falschen Zugriff verhindern  
  
    public int getBalance() { ... } // Kontostand abfragen  
    public void setBalance(int balance) { ... } // sinnvoll?  
}
```

Beispiel Getter und Setter

```
public class Account {  
  
    private int balance;  
  
    public int getBalance() {  
        return this.balance;  
    }  
  
    public void setBalance(int balance) {  
        if (balance >= 0) {  
            this.balance = balance;  
        }  
    }  
}
```

Quiz: Attribute

Welche Aussagen sind korrekt?

- ▶ Attribute können `private` oder `public` sein.
- ▶ `private name;` ist eine gültige Deklaration eines Attributs.
- ▶ Alle Objekte der gleichen Klasse haben für ein Attribut immer denselben Wert.
- ▶ Der Wert eines Attributs kann nie ein Objekt sein.
- ▶ Attribute können in allen Methoden der Klasse verwendet werden.
- ▶ Attribute und Methoden werden von außen gleich aufgerufen.

Quiz: Attribute

Welche Aussagen sind korrekt?

- ▶ Attribute können `private` oder `public` sein. (korrekt)
- ▶ `private name;` ist eine gültige Deklaration eines Attributs. (falsch)
- ▶ Alle Objekte der gleichen Klasse haben für ein Attribut immer denselben Wert. (falsch)
- ▶ Der Wert eines Attributs kann nie ein Objekt sein. (falsch)
- ▶ Attribute können in allen Methoden der Klasse verwendet werden. (korrekt)
- ▶ Attribute und Methoden werden von außen gleich aufgerufen. (falsch)

Konstruktoren

Konstruktor

Der Konstruktor einer Klasse ist eine besondere Art Methode, die automatisch jedes Mal ausgeführt wird, wenn ein Objekt dieser Klasse erzeugt wird.

Konstruktor

Datei Raccoon.java:

```
public class Raccoon {  
  
    public void sleep() {  
        System.out.println("I'm sleeping!");  
    }  
  
}
```

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        // Hier wird der  
        // Konstruktor  
        // aufgerufen  
        Raccoon fluffy = new Raccoon();  
  
        fluffy.sleep();  
    }  
}
```

Konstruktoren

- ▶ Wenn für eine Klasse kein Konstruktor definiert wird, stellt Java einen Standardkonstruktor zur Verfügung, der beim Erzeugen eines Objekts mit `new Raccoon()` aufgerufen wird.
- ▶ Man kann auch selbst Konstruktoren definieren:
 - ▶ Ein Konstruktor hat keinen Rückgabetyp (auch nicht `void`).
 - ▶ Ein Konstruktor heißt exakt genauso wie die Klasse.
 - ▶ Syntax:
`<Modifikator> <Klassenname>(<Parameter>*) {...}`
- ▶ Ein Konstruktor wird nur ein Mal pro Objekt ausgeführt.

Konstruktor

Datei Raccoon.java:

```
public class Raccoon {  
  
    // eigener Konstruktor  
    public Raccoon() {  
    }  
  
    public void sleep() {  
        System.out.println("I'm sleeping!");  
    }  
}
```

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        // Hier wird der  
        // Konstruktor  
        // aufgerufen  
        Raccoon fluffy = new Raccoon();  
  
        fluffy.sleep();  
    }  
}
```

Konstruktoren definieren

- ▶ Konstruktoren können Parameter haben
- ▶ Ein Konstruktor wird oft verwendet, um Attribute des Objekts zu initialisieren
- ▶ Besonders nützlich sind Konstruktoren, wenn Attributwerte vom Aufrufer gesetzt werden sollen
- ▶ Es kann mehrere Konstruktoren für eine Klasse geben (mit unterschiedlichen Parametern).

Rückblick

Datei Raccoon.java:

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

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        // Hier wird der  
        // Konstruktor  
        // aufgerufen  
        Raccoon rocket =  
            new Raccoon("orange");  
  
        rocket.sleep();  
        System.out.println  
            (rocket.getColor());  
    }  
}
```

Konstruktoren definieren

- ▶ Es kann mehrere Konstruktoren für eine Klasse geben (mit unterschiedlichen Parametern).
- ▶ Die Argumente beim Erzeugen eines Objektes bestimmen, welche Konstruktor verwendet wird

Rückblick

Datei Raccoon.java:

```
public class Raccoon {  
  
    public String color;  
  
    // eigener Konstruktor  
    public Raccoon() {  
        this.color = "gray";  
    }  
    // eigener Konstruktor  
    public Raccoon(String color) {  
        this.color = color;  
    }  
  
    public void sleep() {  
        System.out.println("I'm sleeping!");  
    }  
  
    public String getColor() {  
        return this.color;  
    }  
}
```

Datei Test.java:

```
public class Test {  
  
    public static void main  
        (String[] args) {  
  
        // Hier wird der  
        // Konstruktor  
        // aufgerufen  
        Raccoon fluffy =  
            new Raccoon();  
        Raccoon rocket =  
            new Raccoon("orange");  
  
        fluffy.sleep();  
        System.out.println  
            (rocket.getColor());  
    }  
}
```

Quiz: Konstruktoren

Welche Aussagen sind korrekt?

- ▶ Ein Konstruktor kann Parameter haben.
- ▶ Es kann mehrere Konstruktoren für eine Klasse geben, so lange sie sich in den Parametern unterscheiden.
- ▶ Ein Konstruktor kann anders heißen als die Klasse.
- ▶ Ein Konstruktor hat den Rückgabetyp `void`.
- ▶ Ein Konstruktor wird immer dann aufgerufen, wenn ein Objekt der Klasse erzeugt wird.
- ▶ Ein Konstruktor kann mehrmals aufgerufen werden für ein Objekt.

Quiz: Konstruktoren

Welche Aussagen sind korrekt?

- ▶ Ein Konstruktor kann Parameter haben. (korrekt)
- ▶ Es kann mehrere Konstruktoren für eine Klasse geben, so lange sie sich in den Parametern unterscheiden. (korrekt)
- ▶ Ein Konstruktor kann anders heißen als die Klasse. (falsch)
- ▶ Ein Konstruktor hat den Rückgabetyp `void`. (falsch)
- ▶ Ein Konstruktor wird immer dann aufgerufen, wenn ein Objekt der Klasse erzeugt wird. (korrekt)
- ▶ Ein Konstruktor kann mehrmals aufgerufen werden für ein Objekt. (falsch)

Outline

- 1 Rückblick
- 2 Attribute und Konstruktoren
- 3 Enumerations
- 4 Klassenmethoden
- 5 Zusammenfassung

Enumerations

- ▶ Enumerations sind Aufzählungsdatentypen.
- ▶ Sinnvoll, wenn Werte, die der Datentyp annehmen kann, eine endliche, abgeschlossene, aufzählbare Menge von Konstanten ist.

Enumerations

- ▶ Enumerations sind Aufzählungsdatentypen.
- ▶ Sinnvoll, wenn Werte, die der Datentyp annehmen kann, eine endliche, abgeschlossene, aufzählbare Menge von Konstanten ist.
- ▶ Beispiele:
 - ▶ Wochentage,
 - ▶ Planeten,
 - ▶ Himmelsrichtungen,
 - ▶ Farben,
 - ▶ ...

Enumerations

- ▶ Anlegen einer Enumeration:

```
enum Weekday {  
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY  
}
```

- ▶ Variable auf einen der vordefinierten Werte setzen:

```
Weekday day = Weekday.MONDAY;
```

Anwendung

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 Attribute und Konstruktoren
- 3 Enumerations
- 4 Klassenmethoden
- 5 Zusammenfassung

Klassenmethoden und Klassenattribute

- ▶ Wir haben gelernt: Objekte (nicht Klassen) haben Methoden.
- ▶ Stimmt das immer? Was ist mit `Math.sqrt(16)` ?

Klassenmethoden und Klassenattribute

- Wir haben gelernt: Objekte (nicht Klassen) haben Methoden.
- Stimmt das immer? Was ist mit `Math.sqrt(16)` ?

Statische Methoden, Klassenmethoden

Methoden, die zu Klassen (und nicht zu Objekten) gehören, werden in ihrer Signatur durch das Schlüsselwort `static` gekennzeichnet. Sie werden auch als Klassenmethoden bezeichnet.

Umgehen mit `static`

- ▶ Eine Klassenmethode kann aufgerufen werden, auch wenn kein Objekt dieser Klasse existiert.
- ▶ Dazu wird der Klassenname statt einem Objektnamen verwendet. Syntax: `<Klassenname>.<Methodenname>(<Parameter>*)`
- ▶ Informationen aus Objekten können in Klassenmethoden nicht verwendet werden, d. h. man kann keine nicht-`static` Methoden aufrufen oder nicht-`static` Attribute verwenden.

Ein Blick zurück

Zur Erinnerung unser allererstes Programm:

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hallo Welt!");  
        System.out.println("Wie geht's?");  
    }  
}
```

- ▶ Die Methode `main` ist `static`. Sie wird aufgerufen, bevor ein Objekt der Klasse `HelloWorld` existiert (ein solches Objekt wird in diesem Beispiel auch nie erzeugt).

Outline

- 1 Rückblick
- 2 Attribute und Konstruktoren
- 3 Enumerations
- 4 Klassenmethoden
- 5 Zusammenfassung

Zusammenfassung: Attribute und Konstruktoren

- ▶ **Attribute** dienen zum Speichern von Informationen über ein Objekt. Sie haben einen Namen und einen Datentyp.

- ▶ Attribute definieren:

<Modifikator> <Datentyp> <Name>;

- ▶ Zugriff auf Objektattribut:

<Objektname>.<Attributname>

- ▶ Der **Konstruktor** einer Klasse wird jedes Mal ausgeführt, wenn ein Objekt dieser Klasse erzeugt wird.

- ▶ Konstruktor definieren:

<Modifikator> <Klassenname>(<Parameter>*) { ... }

- ▶ Konstruktor aufrufen = Objekt erzeugen:

new <Klassenname>(<Parameter>*)



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