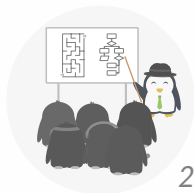


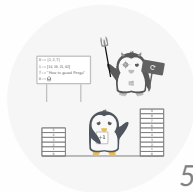
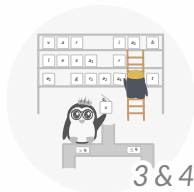
# Klasse globale, potente Entwürfe

## *Tutorium fünf*

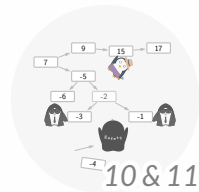
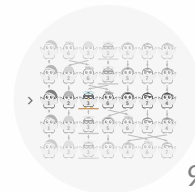
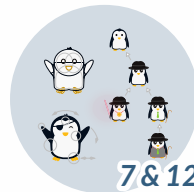
## Theorie



## Grundlagen



## Vertiefungen



- Bei Arrays *muss* Java („die erste Ebene“) initialisieren!
  - Alles wird „Null“ (`0`, `0.0`, `'\0'`, `false`, `null`)
  - `new float[2][ ]` → `{null, null}`, `new int[2][1]` → `{{0}, {0}}`, ...
- Methoden werden in Java durch ihre Signatur unterschieden
  - Dies ist der Name und die Parametertypliste
  - Im Beispiel ist dies `mega(int, char[ ])`:

```
public static double mega(int i, char[] r) {  
    return (double) i + r.length;  
}
```
- Unterprogramme sind ein wichtiger Abstraktions-Mechanismus
- Seiteneffekte sind ein Problem und sollten (wo möglich) vermieden werden!

*„Not all OOP systems are based around classes. Prototype-based OOP doesn't have classes. [...] The reason we have objects, [...] is that object (sic) can be experts about a problem space.“*

*Curtis „Ovid“ Poe — Why is Object-Oriented Programming Bad?*

# Präsenzaufgabe

1

Show me ya' potency

1. Implementieren Sie eine Klasse Potenz für Potenzen. Die Klasse soll zwei private Attribute `basis` und `potenz` besitzen, sowie einen Konstruktor mit zwei Argumenten definieren, der den Attributen Anfangswerte zuweist. Zusätzlich soll es getter und setter für die Attribute geben. Die Klasse soll eine öffentliche Methode besitzen, die die Attribute der Instanz auf die Konsole ausgibt.
2. Legen Sie (im selben Ordner) eine zweite Java Datei namens `PotenzMain.java` an, die als Programmeinstiegspunkt dienen soll, d.h. hier ist die `main` Methode implementiert. Instanziiieren Sie innerhalb der `main` Methode ein Objekt der Klasse `Potenz` und lassen Sie sich die Attribute des Objekts anzeigen.

```
javac Potenz.java PotenzMain.java
java PotenzMain
```

# Ich bin Herbert Klassenzüchter!

Implementieren Sie eine Klasse *Potenz* für Potenzen. Die Klasse soll zwei private Attribute *basis* und *potenz* besitzen, sowie einen Konstruktor mit zwei Argumenten definieren, der den Attributen Anfangswerte zuweist.

Zusätzlich soll es *getter* und *setter* für die Attribute geben. Die Klasse soll eine öffentliche Methode besitzen, die die Attribute der Instanz auf die Konsole ausgibt.

```
public class Potenz {  
    private double basis;  
    private int potenz;  
  
    public Potenz(double b, int e) {  
        this.basis = b;  
        this.potenz = e;  
    }  
    public double getBasis() { return this.basis; }  
    public int getPotenz() { return this.potenz; }  
    public void setBasis(double b) { this.basis = b; }  
    public void setPotenz(int e) { this.potenz = e; }  
  
    public void print() {  
        System.out.println(basis + "^(" + potenz + ")");  
    }  
}
```

Die Typen von *basis* und *potenz* sind hier sinnvoll, aber frei gewählt.

Potenz.java



Legen Sie eine zweite Java Datei namens `PotenzMain.java` an, die als ProgrammEinstiegspunkt dienen soll, d.h. hier ist die `main` Methode implementiert.

Instanziiieren Sie innerhalb der `main` Methode ein Objekt der Klasse `Potenz` und lassen Sie sich die Attribute des Objekts anzeigen.

 `Potenz.java`

```
public class Potenz {  
    public Potenz(double b, int e) { ... }  
    ...  
    public void print() { ... }  
}
```

 `PotenzMain.java`

```
public class PotenzMain {  
    ▷ display in browser  
    public static void main(String[] args) {  
        Potenz p = new Potenz(2.0, 3);  
        p.print(); // → 2.0^(3)  
    }  
}
```

PotenzMain.java

# Übungsblatt 5



# Aufgabe 1: Methoden mit einer variablen Parameterzahl

Legen Sie eine Java Datei namens *Quersummen.java* und implementieren Sie die folgende Aufgabe innerhalb dieser Datei als eine öffentliche statische Methode namens *quersummeVonQuersummen*. Die Methode soll eine beliebige Anzahl an Parametern vom Typ *int* erwarten und für jede dieser Zahlen die Quersumme berechnen und diese aufsummieren. Zusätzlich soll eine weitere boolean Parameter angeben, ob aus der Summe wiederum die Quersumme berechnet werden soll. Testen Sie Ihre Implementierung mit den angehenden Beispielen.

## Beispiele:

Ja, 123, 92, 57, 30  $\rightarrow 6 + 11 + 12 + 3 = 32 \rightarrow 5$

Nein, 12, 9, 4  $\rightarrow 3 + 9 + 4 = 16$

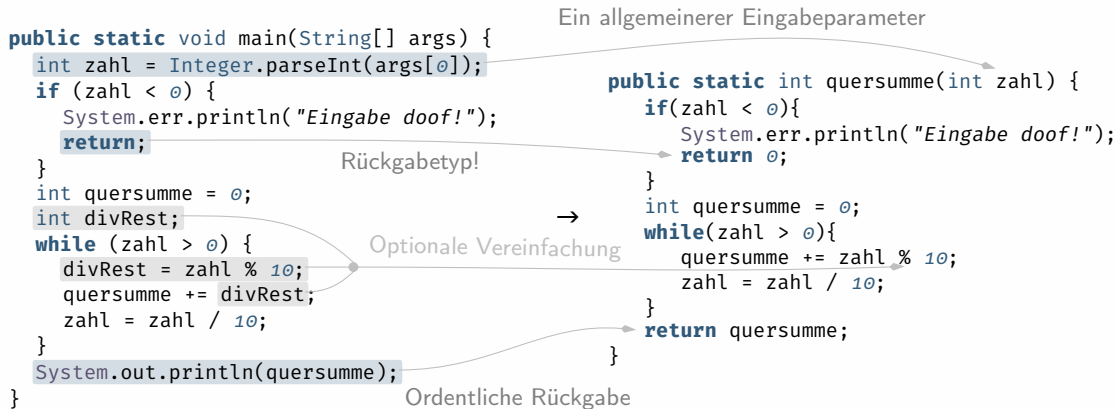


```
public class Quersumme {  
    public static void main(String[] args) {  
        int zahl = Integer.parseInt(args[0]);  
  
        if (zahl < 0) {  
            System.err.println("Eingabe ungültig!");  
            return;  
        }  
        int quersumme = 0;  
        int divRest;  
        while (zahl > 0) {  
            divRest = zahl % 10;  
            quersumme = quersumme + divRest;  
            zahl = zahl / 10;  
        }  
        System.out.println(quersumme);  
    }  
}
```

Blatt 3, Aufgabe 3

# Meta-Quersummen

- Wir sollen mehrere Quersummen berechnen. Wir haben schon ein Programm für eine:



It is always a great time to be a good boy or girl scout!

# Beliebige Quersummen brummen

- › Die beliebige Anzahl ints schaffen wir mit varargs:

```
public class Quersummen {  
    public static int quersumme(int zahl) { ... }  
  
    public static int quersummeVonQuersummen(boolean sum2, int... zahlen) {  
        int quersummenSumme = 0;  
        for(int zahl : zahlen)  
            quersummenSumme += quersumme(zahl);  
  
        return sum2 ? quersumme(quersummenSumme) : quersummenSumme;  
    }  
}
```

*Hier nur ein kurzer Name, damit es auf die Folie passt*

*Signatur?*

# Ich bin die Biene!

› Jetzt fehlt noch die Methode zum Testen:

```
public class Quersummen {  
    public static int quersumme(int zahl) { ... }  
    public static int quersummeVonQuersummen(boolean sum2, int... zahlen)  
        { ... }  
  
    public static void main(String[] args) {  
        System.out.println(quersummeVonQuersummen(true, 123, 92, 57, 30));  
        System.out.println(quersummeVonQuersummen(false, 12, 9, 4));  
    }  
}
```

*Grüße aus einem summderbaren letzten Semester*

Quersummen.java



## Aufgabe 2: Globale Variablen

Legen Sie eine Java Datei namens `CharRotation.java` an. Innerhalb dieser Datei sollen Sie die folgende Aufgabe implementieren.

Implementieren Sie eine Methode namens `rotiereCharacterArray`, die ein `char` Array als Parameter erwartet und innerhalb dieses Arrays (in place) alle Klein- sowie Großbuchstaben um  $n$  Stellen zyklisch und alphabetisch verschiebt. Legen Sie  $n$  als statische globale Konstante an. Dabei sollen Klein- und Großbuchstaben erhalten bleiben. Testen Sie Ihre Implementierung mit mindestens einem Beispiel.

**Beispiel:**

$n = 3, \{ 'a', 'Z' \} \rightarrow \{ 'd', 'C' \}$

Implizite Typkovertierung und Unterprogramme können hier sehr hilfreich sein.



- › Wir reduzieren das Problem, alle Zeichen zu verschieben, zunächst auf ein Zeichen
- › Bei Hilfsmethoden stellt sich die Frage, ob sie semantisch alleine sinnvoll sind
  - Ist es sinnvoll nur ein einziges Zeichen zu rotieren? Ja
  - Benötigen wir kontextabhängige Informationen? Nicht wirklich (n ist konstant)
  - Haben wir implizite Annahmen die gelten müssen? Auch nicht
  - In dem Fall empfiehlt sich **public**, sonst eher **private**

```
public static char rotiereCharacter(char c){  
    // Denglish 4 Leben!  
}
```

- › Nun prüfen wir weiter ob es ein Großbuchstaben, ein Kleinbuchstaben oder ein sonstiges Zeichen ist.

```
public static char rotiereCharacter(char c){
    if (isLowercase(c)) {
        return addCyclicOnBase('a', c);
    } else if (isUppercase(c)) {
        return addCyclicOnBase('A', c);
    } else {
        return c;
    }
}
```

Für Kleinbuchstaben ändert sich die Normalisierung

```
private static char addCyclicOnBase(char base, char c) {
    return (char) (base + Math.floorMod(c - base + n, 26));
}
```

Java's '%' beachtet das Vorzeichen!

Magic-Number (Alphabeet)

```
private static boolean isLowercase(char c) { return c >= 'a' && c <= 'z'; }
private static boolean isUppercase(char c) { return c >= 'A' && c <= 'Z'; }
```

Wrap-Around mit Modulo!  
Wir normalisieren, um nicht die 65  
durch das Modulo zu verlieren.

		c - 'A'
?	63	-2
@	64	-1
A	65	0
B	66	1
C	67	2
⋮		
Y	89	24
Z	90	25
[	91	26
\	92	27

# Eine alternative Rotation

- › Und was machen wir ohne Modulo? Wir können Schleifen verwenden!
- › Zusätzlicher Bonus? Es genügt das Anpassen von `addCyclicOnBase(char, char)`:

 [CharRotationLoops.java](#)

```
private static char addCyclicOnBase(char base, char c) {  
    int shiftedBase = c + n; Füge offset hinzu  
    while(shiftedBase < base)  
        shiftedBase += 26; Verschiebe solange um Zyklusbreite, bis (wieder) in Zyklus!  
    while(shiftedBase > base + 25)  
        shiftedBase -= 26;  
    return (char) shiftedBase;  
}
```

*'A' + 25 ≡ 'Z'*

?    < 'A'    |    A    B    C    ...    Y    Z    |    'Z' <    \



# Ein Beispiel zum Abschluss

- › Nun verbleibt es jedes Zeichen im Array zu verschieben:

```
public static void rotiereCharacterArray(char[] arr) {  
    for (int i = 0; i < arr.length; i++) {  
        arr[i] = rotiereCharacter(arr[i]);  
    }  
}
```

- › Nun gilt es noch aufzurufen:

```
public class CharRotation {  
    public static void main(String[] args) {  
        char[] arr = { 'a', 'Z' };  
        rotiereCharacterArray(arr);  
        System.out.println(arr);    System.out.println(char[])  
    }  
}
```

CharRotation.java

- › Die auf den Folien gezeigte Vorgehensweise ist irreführend
- › Insbesondere wenn es aufwändiger wird, schreibt niemand direkt solchen Code
- › Das Stichwort lautet *Refactoring*
  - Code lässt sich immer aufräumen und leserlicher, testbarer, ... gestalten
  - Die eigentliche Implementierung nimmt relativ wenig Zeit ein<sup>[MaL15]</sup>  
Die meiste Zeit „stirbt“ für das Verstehen von bestehendem Code (je nach Größe rund 70 %), aber auch Testen und Anforderungsdefinitionen bekommen einen Teil.
  - Selbst bei kleinen Projekten (ungefähr 1 000 Zeilen) nimmt die Entwicklung nur  $\approx 50\%$  Zeit ein
- › Nehmt euch doch am Ende 20 Minuten und versucht euren Code zu verbessern

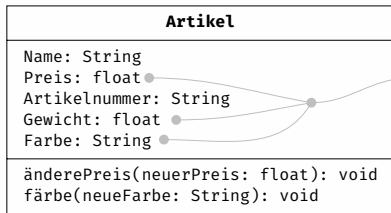
# Aufgabe 3: Objektorientierung: Klassenentwurf

In der Vorlesung haben Sie die Grundlagen der Objektorientierung kennengelernt und anhand zweier Beispiele (Auto und Nachttischlampe auf Folie 13-15, Kapitel 6.7) einen Klassenentwurf gesehen. Entwerfen Sie nach dem in der Vorlesung vorgestellten Muster die folgenden Klassen (es ist noch keine Java Implementierung notwendig und bitte geben Sie auch keine für diese Aufgabe ab).

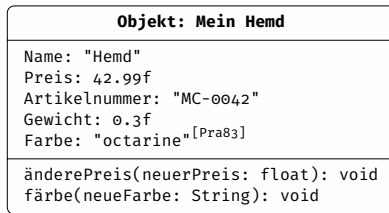
- a) Entwerfen Sie eine Klasse, die einen `Artikel` in einem Online Shop repräsentieren soll. Überlegen Sie sich hierzu, welche Eigenschaften (Attribute) und Funktionen (Methoden) diese Klasse besitzen soll. Geben Sie Datentypen der Attribute so an: „Attributname: Datentyp“. Analog dazu können Sie die Rückgabetypen von Methoden angeben.
- b) Geben Sie eine Instanz der Klasse `Artikel` an.
- c) Als nächstes soll nun die Klasse `Warenkorb` entworfen werden. Überlegen Sie sich hierzu insbesondere, wie die `Artikel` in einem `Warenkorb` modelliert werden können.
- d) Geben Sie eine Instanz der Klasse `Warenkorb` an.

# Ein Artikel, der die Welt verändern wird!

- a) Entwerfen Sie eine Klasse, die einen Artikel in einem Online Shop repräsentieren soll. Überlegen Sie sich hierzu, welche Eigenschaften (Attribute) und Funktionen (Methoden) diese Klasse besitzen soll.
- b) Geben Sie eine Instanz der Klasse Artikel an.



Hier kommen auch  
eigene Datentypen  
in Frage!



Manchmal werden im Objektdiagramm auch keine Methoden wiederholt oder Methoden angepasst. Sichtbarkeiten haben wir hier vernachlässigt.

# Shopping Spreeeeeeee

- c) Als nächstes soll nun die Klasse *Warenkorb* entworfen werden. Überlegen Sie sich hierzu insbesondere, wie die *Artikel* in einem *Warenkorb* modelliert werden können.
- d) Geben Sie eine Instanz der Klasse *Warenkorb* an.

Warenkorb
Kundennummer: int Artikelliste: <i>Artikel</i> []
fügeArtikelHinzu( <i>Artikel</i> : <i>Artikel</i> ): void entferneArtikel(index: int): boolean berechneGesamtpreis(): float

Objekt: Mein Warenkorb
Kundennummer: 744392 Artikelliste: [ <i>Mein Hemd</i> ]
fügeArtikelHinzu( <i>Artikel</i> : <i>Artikel</i> ): void entferneArtikel(index: int): <i>Artikel</i> berechneGesamtpreis(): float

- › Es ist nicht unbedingt notwendig, mit *Main Hemd*, ein Objekt zu referenzieren
- Je nach Problem, reicht vielleicht auch einfach die Artikelnummer, da sie eindeutig ist.
  - Man kann das Objekt auch an Ort und Stelle „schreiben“  
Dann mit meist in JSON-(ähnlicher)-Syntax {Name: "Hemd ", Preis: 42.99f, ...}

- Die so gewählte Darstellungsform ist suboptimal
  - „Gewicht: 0.3f“ – 0,3 was? Gramm? Kilogramm? Tonnen? Pfund? 30 % eines Anzugs?
  - Welchen Artikel soll die Methode `entferneArtikel` zurückliefern? Den entfernten, den neuen an dieser Position, ... ?
  - Sind alle Eigenschaften verpflichtend? Oder gibt es z.B. Artikel ohne Farbe?
  - Drückt man das wiederum durch eine besondere Farbe aus?
- Für diese Veranstaltung interessiert uns aber mehr die Frage, wie wir Probleme in Klassen aufteilen und was diese Klassen dann für Attribute und Methoden haben.
- In dieser Aufgabe waren (zumindest halbwegs) sinnvolle Attribute und Methoden ausreichend

*Aber die Sprache um ein Wort ärmer machen heißt das  
Denken der Nation um einen Begriff ärmer machen  
Arthur Schopenhauer — [SFH59, II, chp. 12]*

# Terminologien

# Ein Beispiel

- › Wir möchten das aufsummierte Alter eines Array an Pinguinen
- › In diesem Zuge möchte ich ein paar Begriffe klären
- › Pinguine beschreiben wir (mit ganz viel Liebe... ich meine) wie folgt...



# Pinguineeee – Struktur

 Penguin.java

Muss gleich heißen wie Datei (Groß- und Kleinschreibung beachten)!

```
public class Penguin { Die Penguin-Klasse!
```

```
    private final int age;
```

```
    final String name;
```

Attribute der Klasse. Sie definieren den **Zustand**.

```
    public Penguin(String name, int age) { Der Konstruktor liefert den initialen Zustand
```

```
        this.name = name;
```

```
        this.age = age;
```

Die Parameter (können beliebig heißen)

```
    }  
    new Penguin("Piep", 3); ✓    new Penguin(); ⚡
```

Der leere default-Konstruktor existiert nur genau dann, wenn kein expliziter existiert.

Ein stinknormaler Getter

Hier existiert Penguin(String, int).

```
    public int getAge() { return age; }
```

```
    public String toString() {
```

```
        return name + " watschelt seit " + age + " Jahr(en)";
```

```
    } toString ist in Java „besonders“ (durch Object). Es wird z.B. bei
```

```
} (String-)Konkatenation automatisch aufgerufen.
```

Methoden der Klasse.

Sie definieren das **Verhalten**.

# Pinguineeee – Gültigkeit & Sichtbarkeit

Penguin.java

**public class** Penguin { *Objekte leben, wie mind. eine Referenz auf sie gültig ist*

**private final** int age; } *Die Attribute besitzen die selbe **Gültigkeit**,  
final String name; wie das Objekt, dem sie gehören.*

**public** Penguin(String name, int age) {

this.name = name;

this.age = age;

}

*Die Parameter **überschatten** (JLS17 6.4) die gleichnamigen Attribute. Wir benötigen nun this um auf diese zuzugreifen.*

*Das age-Attribut wird hier nicht überschattet.*

**public** int getAge() { **return** age; }

**public** String toString() {

**return** name + " watschelt seit " + age + " Jahr(en)";

}

*Beide Attribute werden nicht überschattet!*

}

	Selbe Klasse	Selbes Paket	Unterkasse	Überall sonst	Beispiel
public	✓	✓	✓	✓	age
protected	✓	✓	✓		—
<nichts>	✓	✓			name
private	✓				Penguin

Sichtbarkeitsmodifikatoren

Gültigkeit

# Defaultkonstruktoren und -Werte

- Java erzeugt genau dann einen default-Konstruktor, wenn kein expliziter existiert
- Konstrukturen sind keine Methoden, können aber z.B. überladen werden  
Überladen heißt: gleicher Name, aber andere Signatur (Name & Parametertypenliste)
  - Ein Konstruktor kann nur mit **new** aufgerufen werden
  - Er hat keinen Rückgabotyp
  - Er muss genau so heißen wie die Klasse
  - Er kann andere Konstrukturen über **this** aufrufen (erstes Statement)
- Wie bei Arrays, weißt Java Attributen default-Werte zu, wenn wir dies nicht tun:

```
public class Waddler {  
    int age;  
    float speed;  
    char[] directions;  
    boolean[][] canWaddleOn;  
}
```

```
Waddler w = new Waddler();  
w.age           → 0  
w.speed         → 0.0f  
w.directions    → null  
w.canWaddleOn   → null (kein new!)
```

# Die Summierung

 PenguSum.java

```
public class PenguSum {  
    static long sum(Penguin[] ps) { sum(Penguin[])  
        long sum = 0;  
        for (int i = 0; i < ps.length; i++)  
            sum += ps[i].getAge();  
        return sum;  
    }  
  
    public static void main(String[] args) {  
        Penguin[] penguin = { new Penguin("Hugo", 3),  
            new Penguin("Sophie", 7) };  
        System.out.println(penguin[0]);  
        System.out.println(sum(penguin));  
    }  
}
```

*Hier ist Platz für Heap und Stack.*

*Oh, schon 2 Uhr.*

*Das wird eine Tafelnummer*

# Die Summierung

*Hier ist Platz für Heap und Stack.  
Oh, schon 2 Uhr.  
Das wird eine Tafelnummer*

 PenguSum.java

```
public class PenguSum {  
    static long sum(Penguin[] ps) { sum(Penguin[])  
        long sum = 0;  
        for (int i = 0; i < ps.length; i++)  
            sum += ps[i].getAge();  
        return sum;  
    }  
  
    public static void main(String[] args) {  
        Penguin[] penguin = { new Penguin("Hugo", 3),  
            new Penguin("Sophie", 7) };  
> System.out.println(penguin[0]);    Hugo watschelt seit 3 Jahr(en)  
        System.out.println(sum(penguin));  
    }  
}
```

# Die Summierung

*Hier ist Platz für Heap und Stack.*

*Oh, schon 2 Uhr.*

*Das wird eine Tafelnummer*

 `PenguSum.java`

```
public class PenguSum {  
    static long sum(Penguin[] ps) { sum(Penguin[])  
        long sum = 0;    sum = 3  
    >    for (int i = 0; i < ps.length; i++)    i = 1    (1 < 2)  
        sum += ps[i].getAge();  
    return sum;  
}  
  
public static void main(String[] args) {  
    Penguin[] penguin = { new Penguin("Hugo", 3),  
        new Penguin("Sophie", 7) };  
    System.out.println(penguin[0]);    Hugo watschelt seit 3 Jahr(en)  
    System.out.println(sum(penguin));  
}
```

# Die Summierung

*Hier ist Platz für Heap und Stack.*

*Oh, schon 2 Uhr.*

*Das wird eine Tafelnummer*

 `PenguSum.java`

```
public class PenguSum {  
    static long sum(Penguin[] ps) { sum(Penguin[])  
        long sum = 0;  
        for (int i = 0; i < ps.length; i++)  
            sum += ps[i].getAge();  
        return sum;  
    }  
  
    public static void main(String[] args) {  
        Penguin[] penguin = { new Penguin("Hugo", 3),  
            new Penguin("Sophie", 7) };  
        System.out.println(penguin[0]); Hugo watschelt seit 3 Jahr(en)  
        System.out.println(sum(penguin)); < 10  
    }  
}
```

# Aussicht: Übungsblatt 6



- › Alles was ich zu den Aufgaben auf dem nächsten Blatt sagen könnte, findet sich schon auf den Folien (dies schließt auch letzte Woche mit ein)
- › Man werfe einen Blick zurück!
- › Aber: sich kurz zu fassen heißt nicht, wichtiges auszulassen
  - Die Idee ist eher, dass ihr kurze Beschreibungen findet die alles wichtiges beinhalten
  - Ob dabei wohl Fachbegriffe hilfreich sind?
- › Bei Aufgabe 3 wollen wir die geänderte Java-Datei!

*Those who cannot remember the past are condemned to repeat it.*  
George Santayana – [San05]

# Abschließendes

- [MaL15] Roberto Minelli, Andrea Mocci and und Michele Lanza. „I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time“. 2015
- [Pra83] Terry Pratchett. *The Colour of Magic*. 1983
- [San05] George Santayana. *The life of reason*. 1905
- [SFH59] Arthur Schopenhauer, J. Frauenstädt und A. Hübscher. *Die Welt als Wille und Vorstellung*. 1859
- Javas' Methodensignaturen bestehen aus dem Namen und der Parametertypenliste
  - Java unterscheidet vier verschiedene Sichtbarkeitsbereiche (von denen drei bisher relevant sind)
    - Java verwendet Pakete (Ordner) zur hierarchischen Verwaltung
    - **public**: überall, nichts: „package-private“, **private**: innerhalb der Klasse
  - Klassen besitzen einen Zustand (Attribute) und ein Verhalten (Methoden)
    - Der Konstruktor legt den initialen Zustand fest
    - Es gibt nur dann einen default-Konstruktor, wenn kein expliziter vorliegt
    - (nicht-final) Attribute werden von Java mit default-Werten initialisiert

