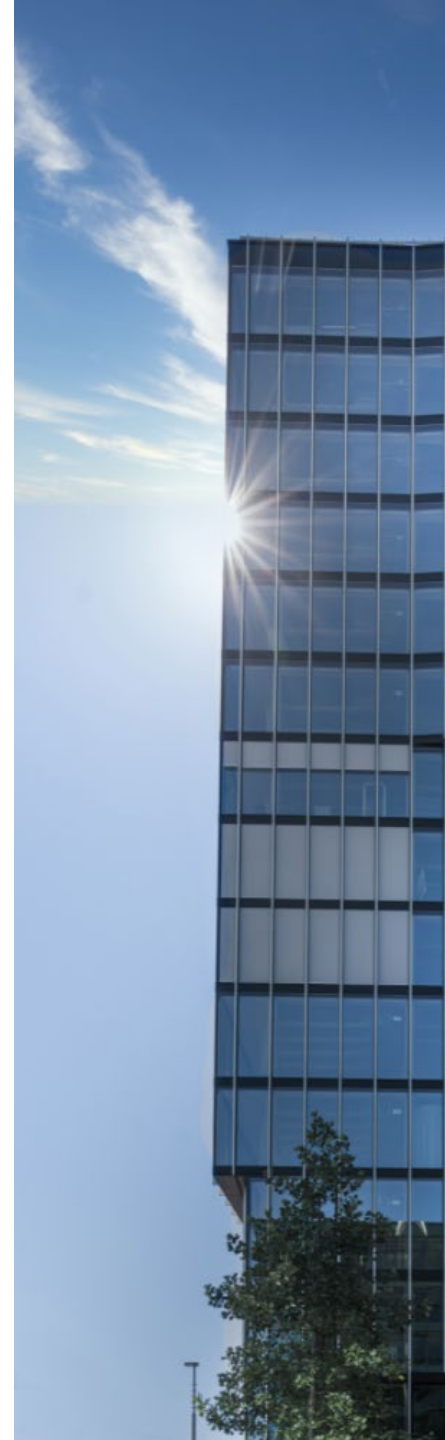


Objektorientierte Programmierung

Klassen

Roland Gisler



Inhalt

- Aufbau einer Klasse
- Attribute
- Methoden
- Parameter
- Rückgabewert (Returnwert)
- Lokale Variablen
- Konstruktoren

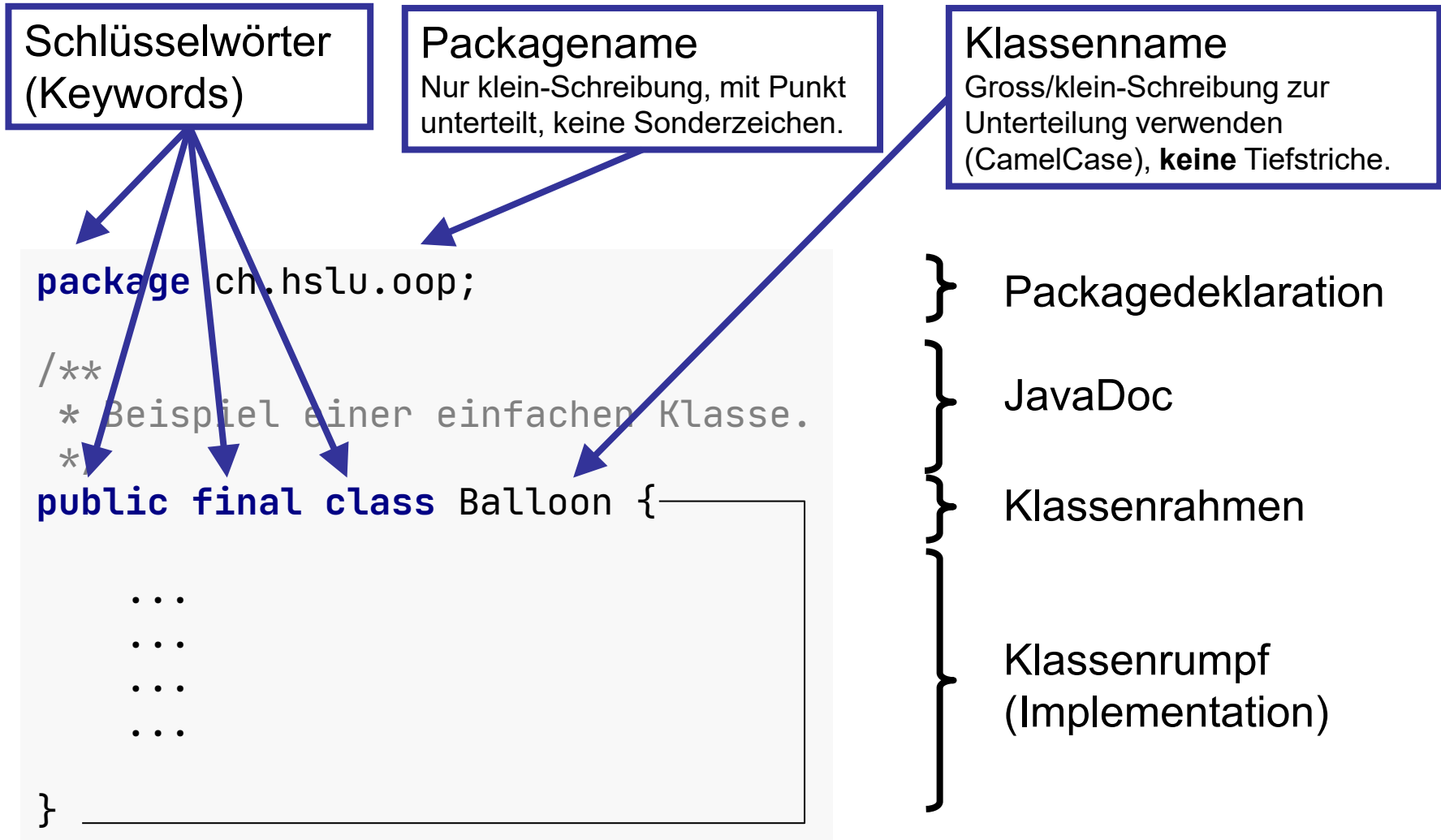
Lernziele

- Die zentralen "Bestandteile" einer Klasse verstehen: Attribute, Konstruktoren, Methoden.
- Merkmale der verschiedenen Arten von Variablen kennen, vgl. Instanzvariablen, lokale Variablen und Parameter.
- Zwischen der Implementation und dem Aufruf einer Methode, sowie zwischen formalen und aktuellen Parametern unterscheiden können.
- Eine einfache Klasse schemamässig implementieren können.

Aufbau einer Klasse

Aufbau einer Klasse - Übersicht

- Klassen sind in Java wie folgt aufgebaut:



Aufbau einer Klasse – Package und Name

- Damit man bei vielen und grossen Projekten mit tausenden von Klassen den Überblick nicht verliert, teilt man die einzelnen Klassen auf Packages auf.
- Für die Namensgebung verwendet man typisch den «reverse domainname» (Beispiel: **hslu.ch** → **ch.hslu**) als Basis.
 - Erst damit erreicht man auch eine eindeutige Identifikation!
- Beispiel: Klasse **Person**, abgelegt im Package **ch.hslu.oop.ex1**:
ch.hslu.oop.ex1.Person
 - Das ist der vollqualifizierte Klassenname!
- Ablage der Datei im Dateisystem (Quellverzeichnis):
 - Im (Unter-)Verzeichnis: **ch/hslu/oop/ex1**
 - Name der Datei: **Person.java**

Aufbau einer Klasse – Klassenrumpf

- Innerer Teil:

```
...  
public final class Balloon {  
  
    // 1. Attribute*  
  
    // 2. Konstruktoren  
  
    // 3. Methoden**  
}
```

→ Zustand eines Objektes

→ Initialisierung eines Objektes

→ Verhalten eines Objektes

Synonyme: * Instanzvariablen, Felder, Membervariablen
 ** Instanzmethoden, Funktionen, Prozeduren

- Tipp: Wir implementieren den inneren Teil einer Klasse konsequent gemäss obiger Reihenfolge!

Attribute

Attribute (1)

■ Deklaration von Attributen:

```
/**
 * Beispiel einer einfachen Klasse.
 */
public final class Balloon {

    private int distance;
    private int altitude;
    private int radius;
    private String color;

    ...

}
```

Zugriffsmodifizierer:
möglichst **private**!

Datentyp:
z.B. **int**

Bezeichner:
z.B. **radius**
(beginnen **immer** mit
Kleinbuchstaben)

Attribute – Merkmale (1)

- Halten den Zustand / die Eigenschaften eines Objektes fest.
 - z.B. die Grösse eines Rechtecks mit Höhe und Breite.
- Jedes Attribut dient zum Speichern von Daten und besitzt einen entsprechenden Datentyp.
 - Daten bzw. Datenwerte können im Verlaufe der Zeit verändert werden → darum auch die Bezeichnung (Instanz-)Variable.
- Die Gesamtheit aller Attribute eines Objektes mit ihren Datenwerten definieren den aktuellen Zustand des Objektes.

Attribute – Merkmale (2)

- Attribute existieren zeitlebens eines Objektes.
 - Also so lange wie ein Objekt zur Laufzeit existiert.
- Sie sind innerhalb der ganzen Klasse sichtbar/ansprechbar
 - In jeder vorhandenen Methode der Klasse sichtbar.
- Sie sollten trotz automatischer Initialisierung ggf. explizit initialisiert werden.
- Sie erfordern zur Laufzeit für jedes Objekt Speicherplatz auf dem so genannten Heap.

Attribute – Automatische Initialisierung

- Bei elementaren (primitiven) Datentypen:

-double	→	0.0, 0.0d, 0.0D	(IEEE-754, 64 Bits)
-float	→	0.0f, 0.0F	(IEEE-754, 32 Bits)
-long	→	0L, 0L	(2er-Komplement, 64 Bits)
-int	→	0	(2er-Komplement, 32 Bits)
-short	→	0	(2er-Komplement, 16 Bits)
-byte	→	0	(2er-Komplement, 8 Bits)
-char	→	\u0000	(Unicode, 16 Bits)
-boolean	→	false	(8 Bits)

- Bei Klassen/Klassentypen:

- z.B. **String** → null (Adresse/Referenz)

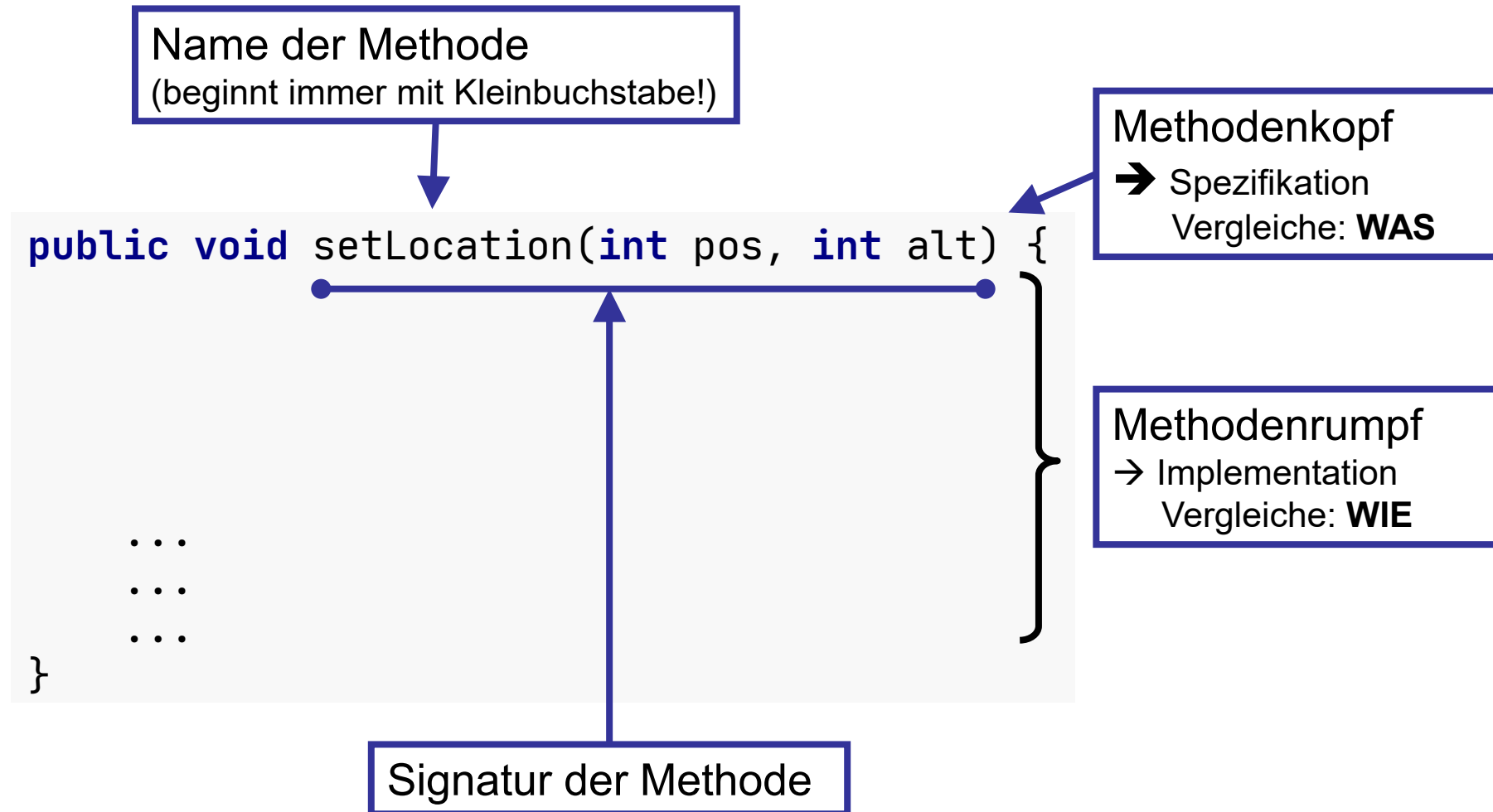
Methoden

Methoden

- Fundamentales Konstrukt bei strukturierter, objektorientierter und funktionaler Programmierung!
 - ➔ Prozedur, Unterprogramm, Subroutine, Funktion, Methode...
- Abstraktion: Ein einfacher Methodenaufruf kann einen sehr komplexen Sachverhalt abstrahieren. Beispiel: **drawLine(...)**
- Wiederverwendung:
 - Methode **einmal** implementieren, aber an **n**-Stellen aufrufen.
 - Beispiel: **sin()**-Funktion der **Math**-Klasse.
- Zusätzlich bei der Objektorientierung:
 - Verhalten von Objekten realisieren, z.B. **blowUp()**
 - Kommunikation/Interaktion zwischen Objekten ermöglichen:
 - Methodenaufruf = Anfrage an ein Objekt, vgl. **getColor()**
 - Rückgabewert eine Methode = Antwort des Objektes.

Methoden - Aufbau

Methoden sind in Java wie folgt aufgebaut:



Methoden – Methodenkopf


- Zugriffsmodifizierer häufig **public** oder **private**.
- spezieller Rückgabetyp **void** möglich, d.h. **kein** Rückgabewert.
- Eine Methode kann auch **keine** Parameter besitzen.
- Begriff der **Signatur** gemäss Java-Spezifikation:
Methodenname inkl. aller Parameter (aber **ohne** Returntyp)
- Innerhalb der selben Klasse müssen alle Methoden in ihrer Signatur unterschiedlich sein!
 - unterschiedliche Methodennamen oder
 - Parameter mit unterschiedlichen Datentypen oder
 - unterschiedliche Anzahl Parameter

Parameter

Formale Parameter

Deklaration von formalen Methodenparametern:

```
public void setLocation(final int pos, final int alt) {  
    this.position = pos;  
    this.altitude = alt;  
}
```



Formale Parameter:
Je mit Datentyp und Bezeichner)

- Formale Parameter sind innerhalb des Methodenrumpfes sehr ähnlich zu → lokalen Variablen.
- Formale Parameter sind bis zum Ende der Methode sichtbar, somit also in der ganzen Methode.
- Werden formale Parameter als **final** markiert, kann man ihnen in der Methode keine neuen Werte zuweisen (nicht überschreibbar).

Parameter – Merkmale und Eigenschaften

- Formale Parameter werden im Kopf einer Methode (oder eines Konstruktors) deklariert.
- Beim Aufruf der Methode / des Konstruktors **müssen** Initialisierungswerte bzw. aktuelle Parameter übergeben werden.
- Wichtig: Variablenwerte werden bei der Übergabe immer kopiert.
 - so genanntes «call by value», ist bei Java **immer** so!
 - belegen Speicherplatz auf dem Call-Stack.
- Sind nur während der Ausführung der Methode / des Konstruktors existent bzw. nur innerhalb der Methode / des Konstruktors sichtbar. Werden auf dem «Stack» abgelegt.
- Über die aktuellen Parameter lässt sich das Verhalten von Methoden quasi pro Aufruf steuern oder verändern.

Parameter – Unterscheidung: Formal und Aktuell

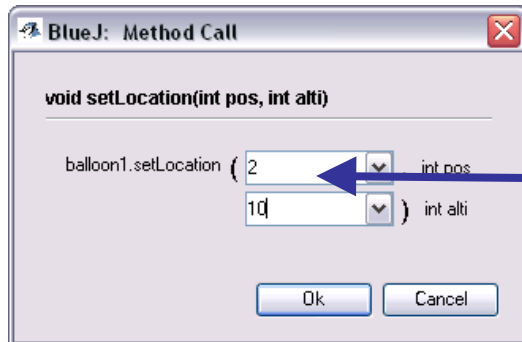
- Gegeben: Methode mit zwei **formalen** Parametern (**pos** und **alt**):

```
public void setLocation(final int pos, final int alt) {  
    this.position = pos;  
    this.altitude = alt;  
}
```

- Programmierter Aufruf der Methode mit **aktuellen** Parametern:

```
...  
balloon.setLocation(2, 10);  
...
```

- Identisches Beispiel, aber interaktiv in BlueJ:



Aktuelle Parameter: konkrete Werte (oder Ausdrücke) vom entsprechenden Datentyp.

Methoden - Implementation

■ Implementation von Methoden:

Zugriffsmodifizierer

Kein Rückgabewert

```
public void setLocation(final int pos, final int alt) {  
    this.position = pos;  
    this.altitude = alt;  
}  
  
public int getPosition() {  
    return this.position;  
}
```

Formale Parameter
(je Datentyp und
Bezeichner)

Rückgabewert
vom Datentyp **int**

Rückgabe des akt.
Wertes von
position

Methoden – Methodenrumpf

- Eigentliche Implementation der Methode
 - { ... }, d.h. der Block nach dem Methodenkopf.
- Beinhaltet Deklarationen und Anweisungen.
- Anweisungen können ihrerseits wieder Blöcke beinhalten.
- Innerhalb eines Blocks werden Deklarationen und Anweisungen durch ";" (Semikolon) voneinander getrennt.
- Der ➔ Rückgabewert wird (falls verlangt) mittels **return** ...; zurück gegeben.
- Ein **return** beendet unmittelbar die Ausführung einer Methode; es darf danach keine unerreichbaren Anweisungen mehr geben!

Rückgabewert

Rückgabewert

- Methode **ohne** Rückgabewert:

```
public void explode() {  
    this.radius = 0;  
    this.altitude = 0;  
    return;  
}
```

void, d.h. die Methode liefert keinen Wert zurück. Das **return**-Statement ist dann **optional**.

- Methode **mit** Rückgabewert:

```
public int getDiameter() {  
    int diameter;  
    diameter = 2 * this.radius;  
    return diameter;  
}
```

int, d.h. die Methode liefert genau einen Wert vom Datentyp **int** zurück.

Zwingendes return mit einem Ausdruck vom Datentyp **int** beendet die Methode und liefert den Wert zurück.

Rückgabewerte – Merkmale

- **return** funktioniert auch mit Ausdrücken
 - Beispiele: `return 117;` `return (a + 7);` `return sin(x);`
- Bei einer Methode mit **void** beendet ein allein stehendes **return** unmittelbar die Ausführung der Methode.
- Eine Methode kann höchstens einen Wert zurückliefern.
- Als Folge einer falsch platzierter **return**-Anweisungen gar nie ausführbare Statements sind **nicht** erlaubt.
 - Es resultiert ein Compilerfehler!
- Man kann bei Java aber eine Methode mit Rückgabewert aufrufen, ohne den Rückgabewert entgegen zu nehmen!
- Methoden mit Rückgabewert sind in der Regel besser lesbar, wenn ein einziges **return** als letzte Anweisung im Block vorkommt!

Methoden – weitere, einfache Beispiele

```
public String getColor() {  
    return this.color;  
}
```

Rückgabewert vom
Klassentyp **String**

keine formalen
Parameter

```
public void blowUp() {  
    this.radius = this.radius + 5;  
}
```

arithmetischer Aus-
druck (Expression)
→ Wert von einem
bestimmten Datentyp

```
public void explode() {  
    this.radius = 0;  
    this.altitude = 0;  
}
```

Wertzuweisung(en)

Variablen
(hier Attribute)

Lokale Variablen

Lokale Variablen

Deklaration von lokalen Variablen:

```
/**
 * Liefert das Volumen der Kugel.
 * @return Volumen in Kubikzentimeter.
 */
public final double getVolume() {

    final double kubik;
    kubik = Math.pow(this.radius, 3.0);
    return (4.0 / 3.0 * Math.PI * kubik);
}
```

Deklaration der **lokalen** Variable **kubik** vom Datentyp **double**.

Sichtbarkeit / Scope von **kubik**.

- Lokale Variablen sind ab der Zeile ihrer Deklaration bis zum Ende ihres umfassenden Blockes sichtbar.

Lokale Variablen – Merkmale und Eigenschaften

- Werden innerhalb (vgl. lokal) einer Methode bzw. eines Blocks deklariert.
- Müssen explizit mit Werten initialisiert werden!
➔ erfolgt im Gegensatz zu Attributen **nicht** automatisch.
- Sind nur während der Ausführung des entsprechenden Blocks existent bzw. nur innerhalb dieses Blocks (aber einschliesslich allfälliger Unterblöcke) sichtbar!
 - Dienen quasi als temporäre Speicher, vgl. "Notizzettel".
- Belegen Speicherplatz auf dem sogenannten Stack.

Konstrukturen

Konstruktor

- Implementation eines Konstruktors:

```
public final class Balloon {  
    ...  
    public Balloon(final String balloonColor) {  
        this.distance = 0;  
        this.altitude = 0;  
        this.radius = 5;  
        this.color = balloonColor;  
    }  
    ...  
}
```

Zugriffsmodifizierer:
häufig **public**

Name muss
identisch mit
Klassenname sein!

Optional:
Formale Parameter
(Datentyp und Bezeichner)

- Achtung: Konstruktoren haben **keinen** Rückgabewert!

Konstruktor - Eigenschaften

- Ermöglicht beim Erzeugen eines Objektes das ordentliche Initialisieren dieses Objektes.
- Wird beim Erzeugen eines Objektes mit `new` wird er **automatisch** aufgerufen und abgearbeitet (ein «direkter» Aufruf ist nicht möglich). Beispiel:

```
Balloon myBalloon = new Balloon("rot");
```

- Initialisiert typischerweise die Instanzvariablen und kann auch optionale Parameter haben.
 - Ohne Parameter ist es ein so genannter Default-Konstruktor.
- Eine Klasse kann mehrere Konstruktoren besitzen!
 - Dann aber zwingend mit unterschiedlichen ➔ Signaturen.
- Vergleichbar mit einer speziellen Methode ohne Rückgabewert.

Destruktoren – in Java nicht vorhanden

- Nach der Regel der Symmetrie müsste es eigentlich auch Destruktoren geben?
 - vgl. Symmetrie von open/close, add/remove, set/get etc.
- Nicht in Java! In Java gibt es keine Destruktoren, weil Java den Speicher mit Hilfe eines Garbage Collectors verwaltet.
- Der Garbage Collector entfernt automatisch nicht mehr benötigte Objekte aus dem Speicher und gibt diesen wieder frei.
 - Grosse Vereinfachung, welche manchen Programmierer*innen schwer fällt zu akzeptieren. 😊

Zusammenfassung

- Einordnung der wichtigsten Elemente einer Klasse: Attribute, Konstruktoren und Methoden
- Methoden mit und ohne Parameter, mit und ohne Rückgabewert
- Signatur einer Methode: Methodenkopf ohne Returntyp
- Unterscheidung zwischen formalen und aktuellen Parametern
- Methodenrumpf mit der Deklaration von lokalen Variablen (optional) und einer Folge von Anweisungen
- Zugriffsmodifizierer für Klassen, Attribute und Methoden
- Finale Parameter



Fragen?

Fragen bitte im
ILIAS-Forum

