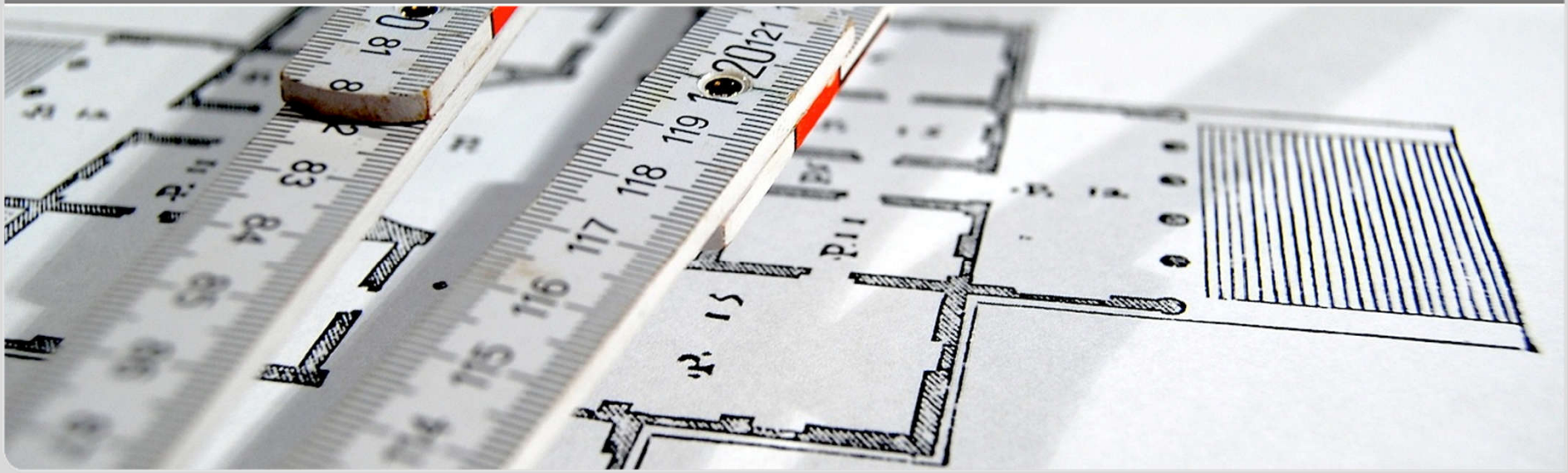
Karlsruher Institut für Technologie

Programmieren-Tutorium Nr. 10

2. Tutorium | Jonas Ludwig

Operatoren, Datentypen, Klassen und Objekte

Architecture-driven Requirements Engineering – Institut für Programmstrukturen und Datenorganisation – Fakultät für Informatik



Fortschrittskontrolle

Habt ihr ...

... die **Einverständniserklärung** abgegeben?

... euch im **ILIAS** Programmieren-Kurs angemeldet?

... das **Java SE 8 SDK** installiert und eingerichtet?

... etwas testweise kompiliert und ausgeführt?

... einen **VPN**-Client eingerichtet und getestet?

... euch im **Praktomat** angemeldet?

... von zu Hause per VPN im Praktomat angemeldet?

Was machen wir Heute?

- Operatoren
- Datentypen
  - int
  - double
  - String
  - enum
- Objektorientierte Programmierung
  - Klassen
  - Objekten

Operatoren

Präzedenz	Operator	Beschreibung
Arithmetische und Vergleichs-Operatoren		
13	`+x, -x`	unäres Plus, unäres Minus
12	`x * y, x / y, x % y`	Multiplikation, Division, Modulo
11	`x + y, x - y`	Addition, Subtraktion
9	`x < y, x <= y, x > y, x >= y`	Größenvergleiche
8	`x == y, x != y`	Gleichheit, Ungleichheit
Operatoren auf ganzen Zahlen		
13	`~x`	Bitweises Komplement (NOT)
10	`x << y`	Linksshift
10	`x >> y`	Rechtsshift (mit Vorzeichen)
10	`x >>> y`	Rechtsshift (ohne Vorzeichen)
Operatoren auf ganzen Zahlen und Wahrheitswerten		
7	`x & y`	Bitweises Und (AND)
6	`x ^ y`	Bitweises Entweder-Oder (XOR)
5	`x | y`	Bitweises Oder (OR)
Operatoren auf Wahrheitswerten		
13	`!x`	Sequenzielles Komplement (NOT)
4	`x && y`	Sequenzielles Und (AND)
3	`x || y`	Sequenzielles Oder (OR)

- Im Zweifel der Präzedenz immer Klammern setzen!

Division und Modulo

- Bei der Division mit Ganzzahlen, wird alles ab dem Komma abgeschnitten
  - Ganzzahlen
    - `4 / 3 = 1`
    - `1 / 3 = 0`
  - Gleitkommazahlen
    - `4.0 / 3.0 = 1.3333333333333333`
    - `1.0 / 3.0 = 0.3333333333333333`
- Ganzzahlen
  - `4 % 3 = 1`
  - `1 % 3 = 1`
- Gleitkommazahlen
  - `4.0 % 3.0 = 1.0`
  - `1.0 % 3.0 = 1.0`

- Die Division durch Null erzeugt meist einen Laufzeitfehler
  - `1 / 0` löst einen Programmabsturz aus (`ArithmeticException`)
  - `1.0 / 0.0` wird zu `Double.POSITIVE_INFINITY`
  - `-1.0 / 0.0` wird zu `Double.NEGATIVE_INFINITY`
  - `0.0 / 0.0` wird zu `Double.NaN`

Primitive Datentypen

Datentyp	Beschreibung	Wertebereich	Beispiel
`boolean`	Boolescher Wahrheitswert	True / False	`true, false`
`char`	16-Bit-Unicode	0 bis 65.535	`'A', '\n', '\u05D0'`
`byte`	8-Bit-Integer	-128 bis 127	`-12`
`short`	16-Bit-Integer	-32.768 bis 32.767	`14`
`int`	32-Bit-Integer	-2.147.483.648 bis 2.147.483.647	`12, -14, 1_000_000`
`long`	64-Bit-Integer	-2^{63} bis $2^{63}-1$	`12L, 14l`
`float`	32-Bit-Gleitkommazahl	$\pm 1{,}4 \times 10^{-45}$ bis $\pm 3{,}4 \times 10^{+38}$	`9.81F, 0.379E-8F, 2f`
`double`	64-Bit-Gleitkommazahl	$\pm 4{,}9 \times 10^{-324}$ bis $\pm 1{,}7 \times 10^{+308}$	`9.81, 0.379E-8, 3e1`

Datentypen

- In Java üblichen Datentypen
  - **`boolean`** für Aussagenlogik
  - **`char`** für eine einzelnen Buchstaben
  - **`int`** für Ganzzahlen
  - **`double`** für Gleitkommazahlen
  - **`enum`** für Aufzählungen
  - **`String`** für Zeichenketten
- Bei Rechenoperationen verwendet der Java-Compiler standardmäßig
  - **`int`** für Ganzzahlen
  - **`double`** für Gleitkommazahlen
- Um ein Literal des Typs **`long`** oder **`float`** zu verwenden, muss ein '**`L`**' oder ein '**`F`**' hinter der Zahl angehängt werden

Variablen und Zuweisungen

- Variablen sind Platzhalter für Werte eines Datentyps
- **Deklaration:** Legt Name und Typ fest
  - `int x;`
- **Zuweisung:** Setzen eines Wertes
  - `x = 4;`
- **Initialisierung:** Kombination von Deklaration und Zuweisung
  - `int y = 2;`
- **Änderung:** Setzen eines (neuen) Wertes
  - `int z = x * 10 + y;`
  - `z = -1 * z;`

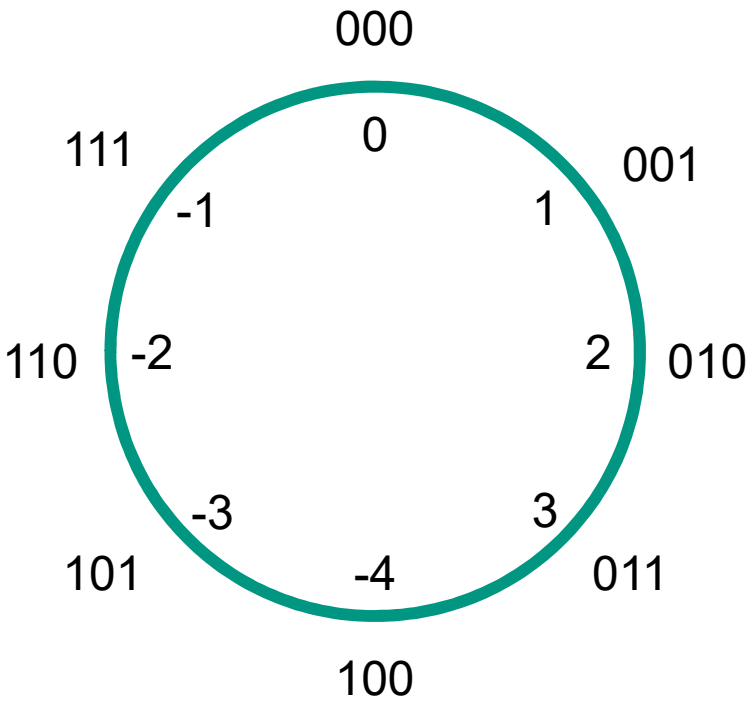
Aufgabe 1

```
int x = 1;
int y = 0;
double z = 1.0;
boolean a = true;
boolean b = false;
boolean c = true;
boolean d;
```

`d = !d;`	Compilerfehler
`d = !a;`	false
`d = a && b;`	false
`d = !a || !c;`	false
`d = (a && b) || !c;`	false
`d = !(!a && b) || !c;`	true
`d = !a || b && c || !d;`	false
`d = !d;`	true
`d = (x == y);`	false
`d = x == z;`	true
`d = (-y != +y);`	false
`d = (x / x == x);`	true
`d = (x % x == y);`	true
`d = y;`	Compilerfehler

Integer

- Datentyp für ganzzahlige Werte
- Der Wertebereich ist endlich
- Berechnungen sind exakt
  - Nur ein Überlauf kann auftreten
- **`int`**: Länge von 4 Byte (32 Bit)
- **`long`**: Länge von 8 Byte (64 Bit)
- LA: Restklassenring



Aufgabe 2

```
int a = 1000000;
int b = 1000000;
int c = a * b;
System.out.println(c);                                          -727379968
System.out.println(a * b);                                      -727379968
System.out.println(1000000 * 1000000);                          -727379968
byte x = 7;
byte y = 14;
byte z = x * y;                                                 Compilerfehler
System.out.println(z);
System.out.println(x * y);                                      98
System.out.println(24*1000*3600*25);                            -2134967296
long l = 24*1000*3600*25;
System.out.println(l);                                          -2134967296
System.out.println(24L*1000L*3600L*25L);                        2160000000
System.out.println(24*1000*3600l*25);                           2160000000
l = 1;
System.out.println(24*1000*3600*25*l);                          2160000000
System.out.println(Integer.MAX_VALUE);                          2147483647
System.out.println(Integer.MIN_VALUE);                          -2147483648
System.out.println(Integer.MAX_VALUE + 1);                      -2147483648
System.out.println(Integer.MIN_VALUE - 1);                      2147483647
System.out.println(Integer.MAX_VALUE - Integer.MIN_VALUE);      -1
System.out.println(Integer.MIN_VALUE - Integer.MAX_VALUE);      1
```

Gleitkommazahlen mit doppelte Genauigkeit

- Vereinfachte Darstellung einer Gleitkommazahl nach IEEE 754
- `double` = ± Mantisse · 2^{Exponent}
  - Vorzeichen 1 Bit
  - Exponent 11 Bits
  - Mantisse 52 Bits
- Länge von 8 Byte (64 Bit)
  - `float` hat nur 4 Byte
- Ungefähre Genauigkeit auf 16 Stellen

Rechnen mit Gleitkommazahlen

- Beim Rechnen kommt es leicht zu Rundungsfehlern!
  - Gleitkommazahlen haben eine endliche Genauigkeit
  - Mehr Informationen im 2. und 3. Semester in „Technische Informatik“

- Gleitkommazahlen nicht direkt auf Gleichheit prüfen!

```
double x;
double y;
boolean b;
// Wertezuweisungen hier...
b = (x == y);   // Sehr schlecht
double epsilon = 0.00001;   // Gewünschte Genauigkeit
b = (Math.abs(x - y) < epsilon);   // Besser
```

- Mehr Informationen in der Java API Dokumentation
  - http://docs.oracle.com/javase/8/docs/api/java/lang/Double.html

Beispiel mit Gleitkommazahlen

- Quelltext

```
public class SuperMarkt {
    public static void main(String[] args) {
        float preisInEuro = 9.90F;
        float einnahmenBei100Kunden = preisInEuro * 100F;
        System.out.println(einnahmenBei100Kunden);
    }
}
```

- Ausgabe

```
989.99994
```

Aufgabe 3

`System.out.println(Double.POSITIVE_INFINITY);`	Infinity
`System.out.println(1.0 / 0.0);`	Infinity
`System.out.println(Double.POSITIVE_INFINITY==(1.0/0.0));`	true
`System.out.println(Double.NEGATIVE_INFINITY);`	-Infinity
`System.out.println(-1.0 / 0.0);`	-Infinity
`System.out.println(Double.NEGATIVE_INFINITY==-1.0/0.0);`	true
`System.out.println(Double.NaN);`	NaN
`System.out.println(0.0 / 0.0);`	NaN
`System.out.println(Double.NaN == (0.0 / 0.0));`	false
`System.out.println(Double.NaN == Double.NaN);`	false
`System.out.println((1E300 * 1E20) == (1.0 / 0));`	true
`System.out.println((1.0 / 0.0) == (-1.0 / 0.0));`	false
`System.out.println((1 / 0.0) == (1.0 / 0));`	true

Assoziativgesetz

- Arithmetische Operationen sind im allgemeinen nicht assoziativ!
  - Rundungsfehler wegen endlicher Genauigkeit
  - Auslöschung signifikanter Stellen

```
public class Associative {
        public static void main(String[] args) {
                float f1 = 1000.0F;
                float f2 = 0.00003F;
                float f3 = f1 + f2 + f2 + f2 + f2;
                float f4 = f2 + f2 + f2 + f2 + f1;
                System.out.println(f3);   // 1000.0
                System.out.println(f4);   // 1000.0001
                System.out.println(f3 == f4);   // false
                System.out.println(Math.abs(f3 - f4) < 0.001);  // true
        }
}
```

String

- Datentyp für Zeichenketten
  - Kein primitiver Datentyp, sondern eine Klasse
  - UTF-16 Zeichen Unterstützung
  - Maximale Länge: 2 147 483 647 Zeichen

- Erzeugen

```
String str1 = "Hallo"; // String mit Inhalt
String str2 = new String(); // leerer String
String str3 = ""; // auch ein leerer String
// "" === new String()
```

- Konkatenieren

```
String str1 = "Hallo";
String str2 = "Welt";
String res1 = str1 + " " + str2; // res1 = Hallo Welt
String res2 = str1 + "+" + str2; // res2 = Hallo+Welt
```

Aufgabe 4

```
boolean b = true;
char c = 'c';
double d = 7;
int i = 2;
String st = "|";
String result = i + st;                    2|
result = st + d;                           |7.0
result = b + st + b + b;                   true|truetrue
result = b + st + !b;                      true|false
result = st + (d + 1);                     |8.0
result = st + d + 1;                       |7.01
result = d + i + "";                       9.0
result = "" + d + i;                       7.02
result = "" + (c + i);                     101
```

enum

- Ein Enumerationstyp bietet die Möglichkeit, einen Satz benannter Konstanten zu definieren
- Das `enum`-Schlüsselwort wird zum Deklarieren einer Enumeration verwendet
- Jedes Aufzählungsobjekt erbt von der Spezialklasse Enum und sind im Grunde nicht andres als statische Objekte

```
enum Name {/*Liste der Werte, durch Kommas getrennt*/}
```

Syntaktischer Zucker

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

class Day extends Enum {
        public static final Day SUNDAY  = new Day("SUNDAY",  0 );
        public static final Day MONDAY  = new Day("MONDAY",  1 );
        public static final Day TUESDAY = new Day("TUESDAY", 2 );
        // weitere Konstanten ...

        private Day(String name, int ordinal) {
                super(name, ordinal);
        }
        // weitere Methoden ...
}
```

Beispiel für enum

- Quelltext

```
public class EnumTest {
    public static void main(String[] args) {
        Day currentDay;
        currentDay = Day.FRIDAY;
        System.out.println(currentDay);
        System.out.println(currentDay == Day.FRIDAY);
    }
}
enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

- Ausgabe

```
FRIDAY
true
```

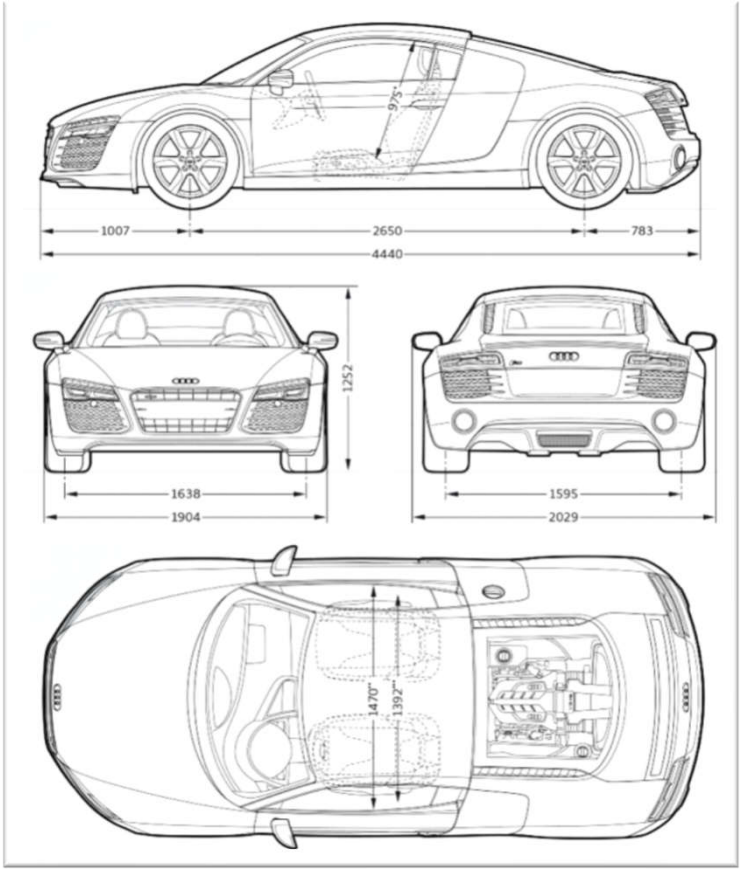
Objektorientierte Programmierung

- Alan Kay definierte Objektorientierte Programmierung folgendermaßen:
  - Alan Kay: The Early History of Smalltalk (1993)

1. Alles ist ein Objekt
2. Objekte kommunizieren durch das Senden und Empfangen von Nachrichten
3. Objekte haben ihren eigenen Speicher (strukturiert als Objekte)
4. Jedes Objekt ist Instanz einer Klasse
5. Die Klasse beinhaltet das Verhalten aller ihrer Instanzen

Beispiel für Objektorientierte Programmierung

Klassendefinition



Objektinstanzen

Bildquelle: AUDI AG

Klassen

- Statische Baupläne für ähnliche Objekte
  - Beschreibt Operationen und Attribute die eine Instanz bekommt
  - Objektinstanzen werden aus Klassen erzeugt
- Die Deklaration einer Klasse leitet das Schlüsselwort `class` ein
- Klassen definieren einen eigenen neuen komplexen Datentypen
- Im Rumpf der Klasse lassen sich deklarieren:
  - Attribute (Variablen)
  - Methoden und Konstruktoren
- Jede Klasse wird in einer eigenen Quelltextdatei abgespeichert
  - Dateiname = Klassenname.java

Beispiel für Klassen

```
class FootballPlayer {
        String name;
        int age;
}
```

```
class Student {
        String name;
        int number;
}
```

```
class Point3D {
    double x;
    double y;
    double z;
}
```

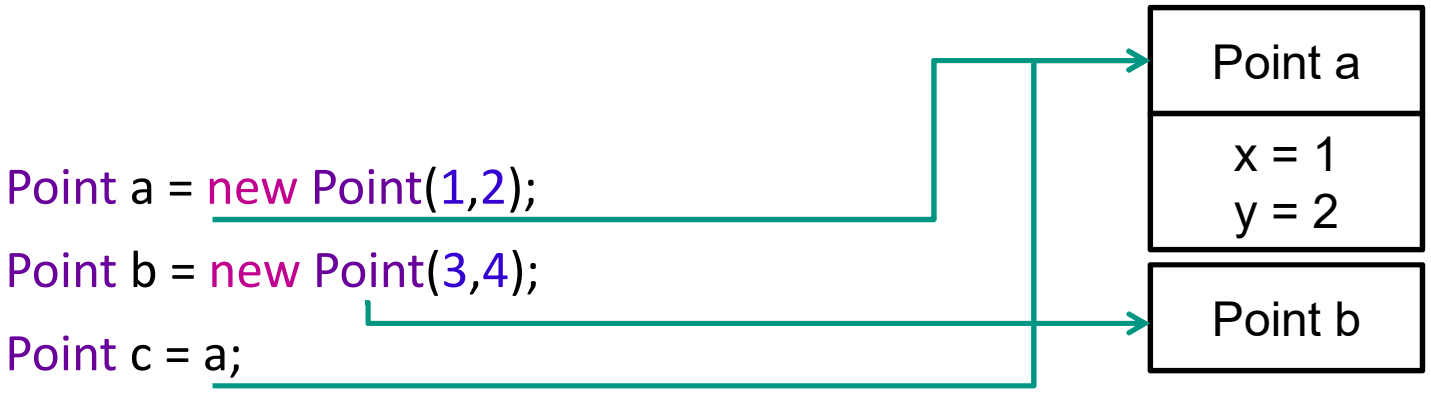
```
class Line {
    Point3D p1;
    Point3D p2;
}
```

Objekte

- Objektinstanzen werden aus Klassen erzeugt
  - Bauplan für ähnliche Objekte
  - Beschreibt Verhalten und Attribute die eine Instanz bekommt
- Haben eine Identität und besitzen:
  - Eigenschaften (Attribute)
  - Verhalten (Methoden)
- Das instanziieren eines Objektes leitet das Schlüsselwort `new` ein
- Kommunizieren mit anderen Objekten durch „Nachrichten“

Objektreferenzen

- Eine Referenz ist ein Verweis auf ein Objekt
  - Variablennamen für ein Objekt
- Erlaubt den Zugriff auf das referenzierte Objekt
- Es kann durchaus mehrere Kopien dieser Referenz geben, die in Variablen mit unterschiedlichen Namen abgelegt sind



Die `null`-Referenz

- Das Literal `null` lässt sich zur Initialisierung von Variablen verwenden
- Hinter `null` verbirgt sich kein Objekt
- null-Referenz ist typenlos

```
Point p = null;

// auf null-Referenzen testen
System.out.println(p == null);

System.out.println(p.y);        // NullPointerException und
System.out.println(p.x);        // Programmabsturz zur Laufzeit
```

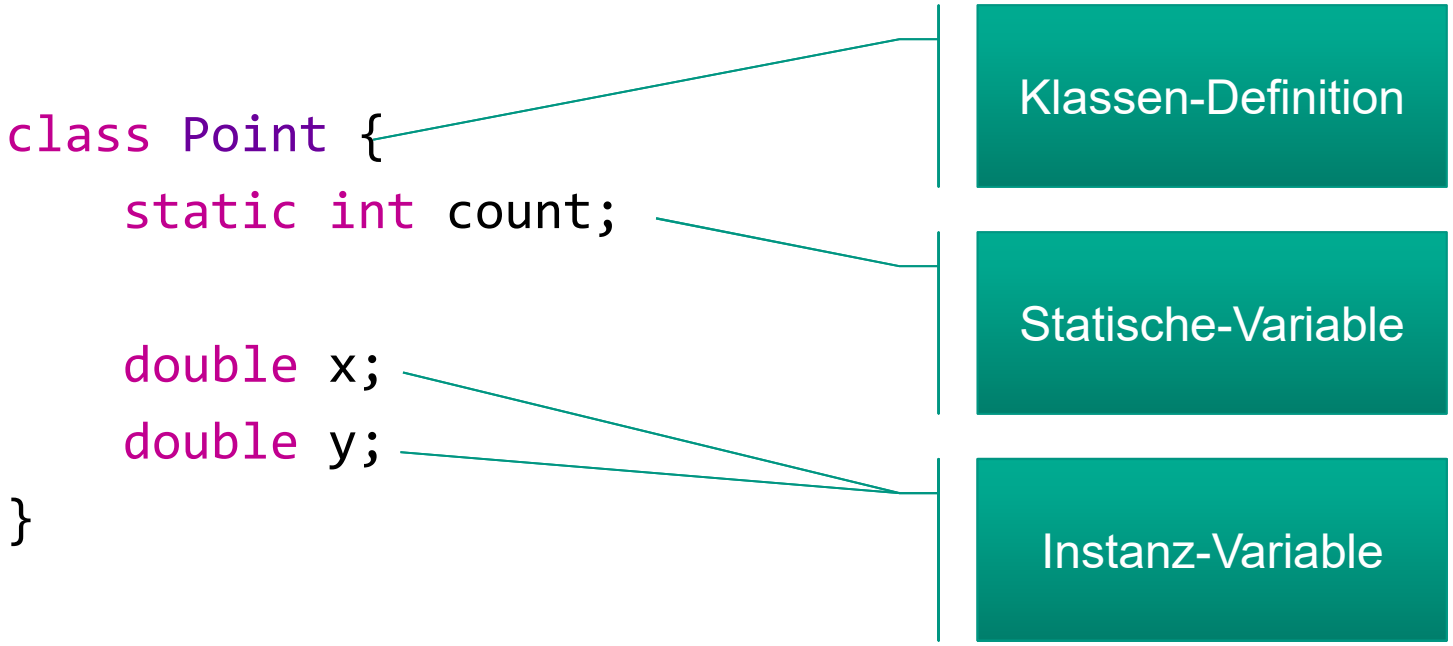
Instanzvariablen

- Ein Objekt speichert seine Zustände in Instanzvariablen
- Alle Objekte einer Klasse haben dieselben Instanzvariablen
- Eine Variable ist eine Dateneinheit von einem bestimmten Datentyp und hat einen Namen
  - Datentyp und Name werden in der Klasse deklariert
  - Der Name ermöglicht den Zugriff auf den Inhalt der Variablen
- Die Werte der Variablen können während der Laufzeit geändert werden
- Instanzvariablen werden am Beginn der Klassendefinition definiert

Statische Elemente

- Variablen und Methoden, die nicht zu einer bestimmten Instanz sondern zur Klasse gehören sind sogenannte statische Elemente
- Statische Variablen/Methoden sind auch dann verfügbar, wenn noch keine Instanz der Klasse erzeugt wurde
- Statische Variablen/Methoden können über den Klassennamen aufgerufen werden
- Deklaration durch das Schlüsselwort `static`

Instanz- und Statische- Variablen



- `Point.count` kann dann zum Beispiel benutzt werden um die Anzahl der Instanzen von `Point` zu speichern. Auf diese Variable kann von (jeder Instanz) der Klasse `Point` aus zugegriffen werden.

Punktoperator

- Erlaubt auf Objekten den Zugriff auf die Methoden oder Zustände
- Er dient dazu, auf Elemente von Klassen oder Datenstrukturen zuzugreifen
- Er steht zwischen einem Ausdruck, der eine Referenz liefert, und der Objekteigenschaft
- Beispiel:
  - `System.out.println(p.x)`

Aufgabe 5 – Instanz- und Statische- Variablen

```
final Point a = new Point();
final Point b = new Point();
final Point c = b;
System.out.println(Point.x);                Compilerfehler
System.out.println(a.x);                    0.0
System.out.println(b.y);                    0.0
System.out.println(Point.count);            0
System.out.println(a.count);                0
System.out.println(c.count);                0
a.count = 2;
System.out.println(a.count);                2
System.out.println(c.count);                2
System.out.println(Point.count);            2
b.x = Point.count;
a.x = 1.0;
b.y = 1.0;
System.out.println(c.x);                    2.0
System.out.println(c.y);                    1.0
Point.count = 3;
System.out.println(c.count);                3
```

```
class Point {
    static int count;

    double x;
    double y;
}
```

Aufgabe 6 – Klassenentwurf für ein Fahrrad

Bevor Sie mit der Klasse Bike, die Fahrräder modellieren soll, loslegen können, sollen Sie jedoch zunächst zwei Hilfsklassen erstellen.

Aufgabe 6.1 – Klasse Gears

Modellieren sie eine Kettenschaltung. Erstellen Sie dafür eine Klasse Gears. Diese Klasse erhält zwei Attribute: Eins für die Anzahl der vorderen Kettenräder und eins für die Anzahl der Ritzel hinten.

Aufgabe 6.2 – Klasse Wheels

Erstellen Sie eine weitere Klasse Wheels, die die Räder das Fahrrads modellieren soll. Auch diese Klasse erhält zwei Attribute: Felgendurchmesser und Reifenstärke. Der Felgendurchmesser wird üblicherweise ganzzahlig angegeben. Die Reifenstärke beträgt maximal 60mm, kann aber gebrochen sein (z.B.: 44,5). Berücksichtigen sie dies bei der Wahl der Datentypen.

Lösung 6.1 – Gears.java

```java
/**
 * This class models the bicycle's gears.
 *
 * @author Jonas Ludwig
 * @version 1.0
 */
class Gears {

        /**
         * Indicates the number of front sprockets
         */
        int frontSprockets;

        /**
         * Indicates the number of rear sprockets
         */
        int rearSprockets;
}
```

Lösung 6.2 – Wheels.java

Karlsruher Institut für Technologie

```java
/**
 * This class models the bicycle's wheels.
 *
 * @author Jonas Ludwig
 * @version 1.0
 */
class Wheels {

        /**
         * Indicates the diameter of the rims (range 150 to 700)
         */
        int diameter;

        /**
         * Indicates the size of the wheels (range 20 to 60)
         */
        double wheelsSize;
}
```

Aufgabe 6 – Klassenentwurf für ein Fahrrad

Karlsruher Institut für Technologie

Aufgabe 6.3 – Klasse Bike

Schreiben Sie jetzt die Klasse Bike. Jedes Fahrrad benötigt sowohl Räder als auch eine Gangschaltung. Die Klasse Bike erhält also sowohl ein Attribut welches den Typ Gears hat, als auch und ein Attribut des Typs Wheels.

Ein weiteres Attribut gibt an aus welchem der folgenden Materialien der Rahmen des Fahrrads besteht. Dabei handelt es sich je nach Fahrrad entweder um Alu, Stahl oder Titan. Im Rahmen dieser Aufgabe bestehen Fahrräder niemals aus anderen Materialen als diesen dreien. Implementieren Sie dieses Attribut indem Sie einen passenden Datentyp hierfür finden und definieren.

Jedes Fahrrad soll auch eine alphanumerische Modellbezeichnung erhalten. Modellbezeichnungen folgen dabei keinem festen Schema, sie können also aus beliebigen Kombinationen von Buchstaben und Zahlen bestehen.

Fügen Sie für die folgenden Ausstattungsmerkmale je eine weitere Eigenschaft hinzu, die angibt ob dieses vorhanden ist oder nicht: Klingel, Beleuchtungseinrichtung.

Lösung 6.3 – Bike.java

```
/**
 * A bike model in Java
 * @author Jonas Ludwig
 */
class Bike {
        /** The gears of the bike */
        Gears gears;

        /** The wheels of the bike */
        Wheels wheels;

        /** local enum provides the possible frame materials */
        enum Material { ALU, STEEL, TITAN };

        /** Indicates the material of the frame */
        Material material;

        /** Indicates the model id */
        String modelId;

        /** Indicates whether or not the bike has a bell */
        boolean hasBell;

        /** Indicates whether or not the bike has lights */
        boolean hasLights;
}
```

Karlsruher Institut für Technologie

Fragen?

Was machen wir nächste Woche?

- Kontrollstrukturen
  - Fallunterscheidungen
  - Schleifen
- Methoden

Vielen Danke für eure Aufmerksamkeit!