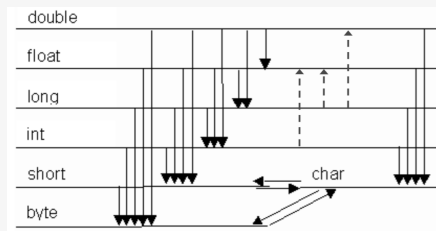


### Typ Konversion



schwarze Pfeile: expliziet  
rote Pfeile: impliziet, evt Genauigkeitsverlust  
alles andere impliziet

### Switch-Statement

```
switch (Ausdruck) {
  case Wert1:
    Anweisung;
    break;
  case Wert2:
    Anweisung;
    break;
  default:
    Anweisung;
}
```

Wenn Ausdruck Wert1 entspricht, wird Anweisung 1 ausgeführt...

### Typ Polymorphismus

```
class Car extends Vehicle {
  @Override
  drive() {}
}
Vehicle v = new Car();
//calls drive of Car not Vehicle
```

### Typ Polymorphismus (cont)

```
> v.drive();
```

Klasse hat eigenen Typ plus alle Typen der Superklassen.  
verschiedene "Brillen"  
Dynamic Dispatch = dynamischer Typ zur Laufzeit entscheidet über aufgerufene Methoden.

### Rekursion

```
public boolean groupSum(int start, int[] nums, int target) {
  if (start >= nums.length)
    return (target == 0);
  if (groupSum(start + 1, nums, target - nums[start])
    return true;
  if (groupSum(start + 1, nums, target)
    return false;
}
```

```
Collection<String> validPairs = new ArrayList<>();
if (nofPairs == 0) {list.add("");} else {
  for (int k = 0; k < nofPairs; k++) {
    Collection<String> infixes = validPairs.subList(0, k);
    Collection<String> suffixes = validPairs.subList(k, nofPairs);
    for (String infix : infixes) {
      for (String suffix : suffixes) {
        list.add("(" + infix + ")" + suffix);
      }
    }
  }
  return list;
}
```

### Generics

allg. Form

```
class Pair<T,U>
```

TypeBound (Bedingungen für Type)

```
class Node<T extends Comparable<T>>
Node<Person>//ok Node<Student>//Error
```

Multiple TypeBounds (mit & anhängen)

```
class Node<T extends Person & Comparable<Person>>
```

generische Methode

```
public <T extends Person> T set(T element){...}
```

<T>: Bedingungen für Input

T : Rückgabotyp

Wildcard-Typ (Nutzen: Typargument nicht relevant)

```
Node<?> undef --> kein Lesen & schreiben
ausser: Object o = undef.getValue();
```

### Lambdas

```
(p1, p2) -> p1.getAge() - p2.getAge() >= 18
people.sort((p1, p2) -> p1.getName().compareTo(p2.getName()));
Utils.removeAll(people, people.sort(Comparator.comparingInt((p) -> p.getAge())));
```

Syntax: (Parameter) -> {Body}

### StreamAPI

```
people
.stream()
.filter(p -> p.getAge() >= 18)
.map(p -> p.getLast Name())
.sorted()
.forEach( System.out::println);
people.stream().mapToInt(p -> p.getAge())
.average().ifPresent( System.out::println);
Map<String, Integer> totalAgeByCity =
    people.stream()
        .collect(
            Collector.of(
                () -> new HashMap<>(),
                (map, person) -> {
                    map.put(person.getCity(),
                        map.getOrDefault(person.getCity(), 0) + person.getAge());
                },
                (map1, map2) -> map1.putAll(map2));
    );
```

**endliche Quelle:** `IntStream.range(1, 100)`

**unendl. Quelle:** `Random random = new Random(); Stream.generate(random::nextInt).limit(100)`

### FileReader/Writer

```
private static void
reverseText() throws
FileNotFoundException,
IOException{
    try (FileReader reader = new
        FileReader("input.txt");
        FileWriter writer =
            new FileWriter("output.txt")){
        int value =
            reader.read();
        String text = "";
        while (value >= 0){
            text = (char) value + text;
            value =
                reader.read();}
    }
```

### FileReader/Writer (cont)

```
> writer.write(text);}
```

### Sichtbarkeit

public	alle Klassen
protected	Package und Sub-Klassen
private	nur innerhalb Klasse
(keines)	innerhalb Package

### Datentypen

byte	8 bit (2 <sup>7</sup> bis 2 <sup>7</sup> -1)
short	16 bit
int	32 bit
long	64 bit (1L)
float	32 bit (0.1f)
double	64 bit

### Operator-Prio

+, -, ++, --, ! (unär)

\*, /, %

+, - (binär)

<, <=, >, >=

==, !=

&&

||

### Rundungsfehler

0.1+0.1+0.1 != 0.3

Problem: `x == y`

double/float

t

Lösung `Math.abs(x - y) < 1e-6`

### Integer Literal

binär `0b10 = 2`

oktal `010 = 8`

hex `0x10 = 16`

opt. `1000 = 1_000`

### Arithmetik

`1 / 0 --> ArithmeticException: / by zero`

`1 / 0.0 --> Infinity`

`-1.0 / 0 --> -Infinity`

`0 / 0.0 --> NaN`

### Arithmetik Overloading

```
int operate(int x, int y) { ... }
double operate(int x, double y) { ... }
double operate(double x, double y) { ... }
```

`operate(1, 2); --> meth 1`

`operate(1.0, 2); --> meth 3`

`operate(1, 2.0); --> meth 2`

### Overloading

```
class Graphic {
    void moveTo(Graphic other)
    // Method 1
}

class Circle extends Graphic {
    void moveTo(Graphic other) {
        // Method 2
    }

    void moveTo(Circle other) {
        // Method 3
    }
}

Graphic g = new Circle();
Circle c = new Circle();

g.moveTo(c); // Compiler: nur Methode 1 -> Laufzeit: Overriding durch Methode 2
c.moveTo(g); // Compiler: Overloading, nur Methode 2 passt für Argument g
c.moveTo(c); // Compiler: Overloading, Methode 3 spezifischer
g.moveTo(g); // Compiler: nur Methode 1 -> Laufzeit: Overriding durch Methode 2
```

### Bedingungsoperator

```
max = left > right ? left :
right;
```

wenn left>right wird max = left, sonst  
max=right

### Package prio

own class (inc. nested class)

single type imports -> import p2.A;

class in same package

import on demand -> import p2.\*

### Enum

```
public enum Color {
    BLUE(1), RED(2);
    private final int code;
    private Color(int code) {
        this.code = code;
    }
    public int getColorValue() {
        return code;
    }
}
```

### Methodenreferenz

```
people.sort( (this::compareToByAge) );
int compareToByAge(Person p1, Person p2) {
    return p1.age - p2.age;
}

people.sort( (Sorter::compareToByAge) );
static int compareToByAge(Person p1, Person p2) {
    return p1.age - p2.age;
}

Sorter sorter = new Sorter();
people.sort(sorter::compareToByAge);
```

### Serialisieren

```
OutputStream fos = new
FileOutputStream("serial.bin");
try (ObjectOutputStream
stream = new ObjectOutputStream(
Stream(fos)) {
    stream.writeObject(
person);
}
```

needs Serializable interface  
serialisiert alle direkt & indirekt referenz.  
Obj

### Input/Output

```
try (BufferedInputStream
inputBuffer = new bis(new
FileInputStream(pathSource));
BufferedOutputStream outputB
= new bos(new FileOutputStream(
pathTarget)) {
    byte[] byteBuffer = new
byte[4096];
    int value = inputBuffer
.read(byteBuffer);
    while (value >= 0) {
        outputB.write(
byteBuffer, 0, value);
        value = inputBuffer
.read(byteBuffer);
    }
}
```

### Input/Output (cont)

```
catch (FileNotFoundException e) {
    System.out.println("File not found: "+e);
} catch (IOException e) {
    System.out.println("reading Error: "+e);
}

static int compareToByAge(Person p1, Person p2) {
    return p1.age - p2.age;
}
```

### String Pooling

```
String a = "OO", b = "Prog", c =
"OOProg";
//true
a == "OO"; c == "OO" + "Pro -
g"; (a+b).equals(c);
//false
a + b == c;
```

String Literals gepoolt. True Fälle --> es  
werden keine neuen Objekte erstellt  
Integer-Pooling -128 bis 127

### Collections

Array	int[]
List	List<T> >;
Set	Set<T>
Map	Map<U, >;
Iterator< T> it = a.iterator(); .}	

Collections.sort(Objekt) needs Comparable



### Collections (cont)

```
for (Map.Entry<Character, Double> entry : map.entrySet()) {
    Set()
```

LIST/SET: add(), remove()

LIST: get(index)

MAP: put(key, value), remove(key), getKey()

### Interface vs. Abstract Class

Methoden	implizit public, abstract	evt abstract, nie private
----------	---------------------------	---------------------------

Variablen	KONSTANTEN (impl. public static final)	normal, Constructor mit super()
-----------	--	---------------------------------

Vererbung	implements a,b	extends a
-----------	----------------	-----------

Abstract Method (in abs. cl.):

```
public abstract void report();
```

kein Rumpf{}

in interface: void report();

I. Methode mit Rumpf: default void report() {...}

```
interface I1 extends I2, I3
class C2 extends C1 implements I1, I2
```

wenn I1&I2 nun gleiche Methoden haben (signatur) --> C2 muss in der beiden überschreiben. Zugriff möglich mit

```
I1.super.methode(); I2.super.methode();
```

### Unchecked vs Checked Exceptions

Unchecked	Checked
-----------	---------

Error	Try/catch oder Methodenkopf
-------	-----------------------------

RunTime-Exceptions	mögl. catch-Block:
--------------------	--------------------

	e.printStackTrace();
--	----------------------

### Unchecked vs Checked Exceptions (cont)

RuntimeException, NullPointerException, IllegalArgumentException, IndexOutOfBoundsException

eigene exception

```
class MyException extends Exception {
    MyException(String message) {
        super(message);
    }
}
```

### Junit

```
@Before
public void setUp() {...}
@Test (timeout = 500, expected =
    assertEquals(expected, actual);
    assertTrue(condition);
```

@After -> tearDown()

EdgeCases testen (Grenzwerte, null,...)

### equals

```
@Override
public boolean equals (Object obj) {
    if (this == obj) {return true;}
    if (obj == null) {return false;}
    if (getClass() != obj.getClass()) {
        return false;}
    Student other = (Student) obj;
    return regNumber == other.regNumber;
}
```

HashCode überschreiben!

```
x.equals(y) -> x.hashCode() == y.hashCode()
```

Grund: Inkonsistenz bei Hashing. obj wird nicht gefunden, obwohl in Hash-Tabelle, x.contains(y) --> nutzt hashCode() & equals()

### HashCode()

```
@Override
public int hashCode() {
    return 31 * firstName.hashCode()
        + 31 * lastName.hashCode();
}
```

### Funktionsschnittstelle

```
@FunctionalInterface
interface Comparator<T> {
    int compare(T first, T second);
}
```

genau eine Methode --> java.util.function

### RegEx

Pattern pattern = Pattern.compile("reg")

vor ? optionaler Teil ([0-9]{2,3})

(){}\*+?|\ als norm text \* \ ( \ ...

Gruppennamen (?<NAME>)

```
Matcher matcher = pattern.matcher(string);
if (matcher.matches()){String one = matcher.group("NAME")}
```

