|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Software Lifecycle** | | | | | | | | | | | | | | | | |
| Anforderungsanalyse 🡪 Entwurf 🡪 Implementierung 🡪 Testen & Verbessern 🡪 Betrieb | | | | | | | | | | | | | | | | |
| **Primitive Datentypen / Mehrdimensionale Arrays** | | | | | | | | | | | | | | | | |
| Alle numerischen Datentypen sind vorzeichenbehaftet! | | | | | | | | | | | | |  | | | |
| int**[][]** m **=** **{**  **{**0**,**0**,**0**},**  **{**0**,**0**,**12**}**  **}**  m**.**length**;** //2  m**[**0**].**length**;** //3 | | | |
|  | | | |
| **Literale** | | | | | | | | | | | | | | | | |
| Binär | | **0b**1101010 | | | | | | Dezimal | | | | | | | | 30,0 |
| Oktal | | **0**36, **0**0 | | | | | | Hexadezimal | | | | | | | | **0x**1e, **0x**0, **0x**ABCD |
| Trennzeichen | | 1\_123\_456 | | | | | | String = "" | | | | | | | | char = '' |
| long | | 12453246**l,** 123**L** | | | | | | int | | | | | | | | Default |
| float | | 1.2**f**, .1**F** | | | | | | double (default) | | | | | | | | 1E23**d**, .1**D** |
| **Mathematische Methoden** | | | | | | | | | | | | | | | | |
| Betrag | | Math.abs(x) | | | | | | Potenz (ab) | | | | | | | | Math.pow(a,b) |
| Wurzel | | Math.sqrt(x) | | | | | | *10*er Log | | | | | | | | Math.log*10*(x) |
| Exponential Func. | | Math.exp(x) | | | | | | Sinus | | | | | | | | Math.sin(x) |
| Pi | | Math.PI | | | | | | Eulersche Zahl | | | | | | | | Math.E |
| **Float / Double Spezialfälle** | | | | | | | | | | | | | | | | |
| **MIN** | | Float.MIN\_VALUE | | | | | | **MAX** | | | | | | | Double.MAX\_VALUE | |
| **Rundungsfehler** | | 0.1 + 0.1 + 0.1 ist nicht exakt 0.3 | | | | | | | | | | | | | | |
|  | | Ca. 8 signifikante Stellen bleiben erhalten (1E8) | | | | | | | | | | | | | | |
| **Vergleich mit Toleranz** | | x == y problematisch | | | | | | | | | | | | | | |
| Math.abs(x – y) < 1E-6 | | | | | | | | | | | | | | |
| **Überlauf** | | Ganzzahlen | | | | | | | | 2147483647 + 1 🡪 -2147483648 | | | | | | |
| Gleitkommazahlen: | | | | | | | | 2 \* 1e308 🡪 POSITIVE\_INFINITY | | | | | | |
| **Unterlauf** | | 5E-324 🡪 0.0 | | | | | | | | | | | | | | |
| **Division durch 0** | | Ganzzahlen | | | | | java.lang.ArithmeticException: / by zero | | | | | | | | | |
| Gleitkommazahl | | | | | 1.0 / 0.0 🡪 Double.POSITIVE\_INFINITY | | | | | | | | | |
|  | | | | | -1.0 / 0.0 🡪Double.NEGATIVE\_INFINITY | | | | | | | | | |
|  | | | | | 0.0 / 0.0 🡪 Double.NaN | | | | | | | | | |
| **Typkonversionen / Casting** | | | | | | | | | | | | | | | | |
| **Explizit**: Schwarze Pfeile (Cast-Operator)  **Implizit(Genauigkeitsverlust)**: Rote Pfeile  **Rest**: implizit vom Compiler  **Gleitkommazahl zu Ganzzahl:**  NaN 🡪 0  3.5 🡪 3 (abhacken)  **Informationeverlust**:  int🡪 byte | | | |  | | | | | | | | | | | | |
| **Autoboxing / Wrapper Klassen (tiefere Prio. Bei Overloading als Typkonversion)** | | | | | | | | | | | | | | | | |
|  | | | | | | | Integer wrapper **=** 123**;** //implizit Integer.valueOf(123);  int value **=** wrapper**;** // implizit wrapper.intValue();  Double d **=** 4**;** // Compile Error  Keine implizite Typkonversion und zusätzlich Auto Boxing | | | | | | | | | |
| **Operatoren Priorität** | | | | | | | | | | | | | | | | |
|  |  | | | | | | | | | // b wird nur ausgewertet wenn a == true  **if** **(**a **&&** b**)** | | | | | | |
| // b wird nur ausgewertet wenn a == false  **if** **(**a **||** b**)** | | | | | | |
| **IF / ELSE Spezialfälle** | | | | | | | | | | | | | | | | |
| **if** **(**x **>** y**)**  t **=** x**;**  x **=** y**;** // wird immer ausgeführt! | | | | | | | | | x **=** a **?** b **:** c // short form | | | | | | | |
| **SWITCH Statement** | | | | | | | | | | | | | | | | |
| **switch** **(**code**)** **{**  **case** "m"**:**  **return** MALE**;**  **case** "f"**:**  **return** FEMALE**;**  **default:**  **throw** **new** IllegalArgumentException**(**""**);**  **}** | | | | | | | | | | | | | | **switch** **(**Ausdruck**)** **{**  **case** Wert1**:**  **case** Wert2**:**  Anweisung**;**  **break;**  **default:**  Anweisung**;**  **}** | | |
| **Loops** | | | | | | | | | | | | | | | | |
| **for** **(**int i **=** 0**;** i **<** array**.**length**;** i**++)** **{**  System**.**out**.**println**(**array**[**i**]);**  **}** | | | | | | | | | | | **do** **{**  Anweisung**;**  **}** **while** **(**Bedingung**);** | | | | | |
| **while** **(**Bedingung**)** **{**  Anweisung**;**  **}** | | | | | | | | | | | // Enhanced for-loop  **for** **(**String s**:** array**)** **{**  **if** **(**s**.**contains**(**"test"**))** **{**  **continue;**  **}** **else** **{**  **break;**  **}**  **}** | | | | | |
| Iterator it **=** collection**.**iterator**();**  **while(**it**.**hasNext**())** **{**  String val **=** it**.**next**();**  it**.**remove**();**  **}** | | | | | | | | | | |
| **Call by Value (Java unterstützt kein Call by Reference)** | | | | | | | | | | | | | | | | |
| Primitive Datentypen: Kopie des Arguments. Änderungen des Parameters beim Aufrufer unsichtbar (*Ausnahme*: *primitiven Arrays !*) | | | | | | | | | Objekte: Kopie der Objektreferenz. Änderungen innerhalb des Objekts sind beim Aufrufer sichtbar | | | | | | | |
| **Offene Parameterliste (nur 1 optionaler Parameter erlaubt)** | | | | | | | | | | | | | | | | |
| static int sum**(**int a**,** int**...** numbers**)** **{**  System**.**out**.**println**(**numbers**[**0**])**  **}** | | | | | | | | | | | | Nur als letzter Parameter erlaubt. Zugriff über Array | | | | |
| **Overloading (Mehrere Methoden mit gleichem Namen und verschied. Parametern)** | | | | | | | | | | | | | | | | |
| Signatur = Methodenname + List der Parametertypen | | | | | static void print**(**int i**,** double j**)** **{}**  static void print**(**double i**,** int j**)** **{}**  static void print**(**double i**,** double j**)** **{}**  print**(**1.0**,**2.0**);** // method 3  print**(**1.0**,** 2**);** // method 2  print**(**1**,**2**);** // mehrdeutig, compile error | | | | | | | | | | | |
| Auswahl der Methode zur Compile Zeit gemäss Signatur | | | | |
| **Klassen / Objekte** | | | | | | | | | | | | | | | | |
| Objekte sind immer auf dem Heap gespeichert (Methoden im Stack) | | | | | | Objektreferenzen dürfen null sein, können aber zu NullPointerException führen | | | | | | | | | | |
| a **==** b // vergleicht Adressen | | | | | | a**.**equals**(**b**)** // vergleicht Inhalt | | | | | | | | | | |
| **Sichtbarkeit / Default Initialisierung von Datentypen** | | | | | | | | | | | | | | | | |
| Sichtbarkeiten können erweitert werden | | | | | | |  | | | | | | | | | |
| **Konstruktor** | | | | | | | | | | | | | | | | |
| public Point**(**int x**,** int y**,** int b**)** **{**  **this(**x**,**y**);**  **this.**b **=** b**;**  **}** | | | | | | | | | public Point**(**int x**,** int y**)** **{**  **super(**x**,** y**)**  **}** | | | | | | | |
| **String Pooling** | | | | | | | | | | | | | | | | |
| String immer mit equals() vergleichen | | | | | | | | | a == b 🡪 false, da Referenzvergleich | | | | | | | |
| Wird ein String direkt mit "S" erzeugt, wird dieser vom Compiler mit gleichen Strings zusammengefasst | | | | | | | | | "S" == "S" 🡪 true  new String("S") == new String("S") 🡪 false | | | | | | | |
| **Arrays vergleichen** | | | | | | | | | | | | | | | | |
| a **==** b | | | Vergleicht nur Objektreferenzen | | | | | | | | | | | | | |
| a**.**equals**(**b**)** | | | Vergleicht ob gleiches Array Objekt | | | | | | | | | | | | | |
| Arrays**.**equals**(**a**,**b**)** | | | Vergleicht Werte des Arrays | | | | | | | | | | | | | |
| Arrays**.**deepEquals**(**a**,**b**)** | | | Vergleicht Werte in geschachtelten Arrays | | | | | | | | | | | | | |
| **Enumeration** | | | | | | | | | | | | | | | | |
| public enum Daytime **{**  NIGHT**(false),** DAY**(true);**  private boolean light**;**  Daytime**(**boolean light**)** **{**  **this.**light **=** light**;**  **}**  public boolean getLight**()** **{**  **return** light**;**  **}**  **}** | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Rekursion** | | | | | | | |
| private long factorial**(**long number**)** **{**  **if** **(**number **==** 0**)** **{**  **return** 1**;**  **}** **else** **{**  **return** number **\*** factorial**(**number **-**1**);**  **}**  **}** | | | | | Endlosrekursion führt zu StackOverflowException | | |
| **Scanner** | | | | | | | |
| **try** **(**Scanner scanner **=** **new** Scanner**(**System**.**in**))** **{**  **while(**scanner**.**hasNextLine**())** **{**  scanner**.**nextInt**();** scanner**.**nextDouble**();** scanner**.**nextFloat**();**  scanner**.**next**();** // String  **}**  **}** | | | | | | | |
| **Aggregation vor Vererbung** | | | | | | | |
| Has-a Beziehung = Aggregation | | | Is-a Beziehung = Vererbung | | | | |
| **Method Overriding , Type Polymorthismus, Dynamic Dispatch (gleiche Signatur)** | | | | | | | |
| class Vehicle **{**  protected Vehicle report**(){**  **...**  **}**  **}** | | | class Car **extends** Vehicle **{**  @Override  public Car report**()** **{**  **super.**report**();**  **...**  **}**  **}** | | | | |
| Vehicle v **=** **new** Car**();** //type polymorphism  v**.**drive**();** // dynamic dispatch, Methode der Kind Klasse wird ausgeführt | | | | | | | |
| Sichtbarkeit darf erweitert werden (package 🡪 protected 🡪 public)  Rückgabetyp kann Subtyp sein (Covarianz) | | | | | | | |
| **Final Methoden/Klassen: Können nicht überschrieben werden** | | | | | | | |
| public final class Motorcycle **{**  final String welcome **=** "Hello"  **}**  public final void stop**()** **{}** | | | | | | | |
| **Up / Downcast** | | | | | | | |
| Upcast=Typ Polymorphism |  | | | | | ~~Vehicle v~~ **~~=~~****~~new~~** ~~Vehicle~~**~~();~~**  ~~Car c~~ **~~(~~**~~Car~~**~~)~~**~~v~~**~~;~~**  //Runtime Exception | |
| Null Referenzen können immer gecasted werden | | |  | | | | |
| **Instanceof** | | | | | | | |
| Vehicle v **=** **new** Car**();**  **if** **(**v **instanceof** Car**)** **{**  Car c **=** **(**Car**)**v  **}** | | Car c **=** **new** Car**();**  **if** **(**c **instanceof** Vehicle**)** **{**  System**.**out**.**println**(**"ich erbe"**);**  **}** | | | | | |
| **Instanzvariablen Hiding** | | | | | | | |
| Subklasse definiert Instanzvariable mit gleichem Namen wie Superklasse neu. Inhalt ist unabhängig voneinander. Zugriff über ***this.variable, super.variable ((SuperSuperClass)this).variable*** | | | | | | | |
| **Abstrakte Klassen (Grundfunktionalität)** | | | | | | | |
| Abstrakte Klassen können nicht instanziiert werden.  Abstrakte Klassen müssen die Methoden von Interfaces nicht implementieren  Abstrakte Methoden werden in den Subklassen implementiert | | | | | | | |
| abstract class Vehicle **{**  abstract void report**();** // implementation in child mandatory !  void print**()** **{**  System**.**out**.**println**(**"I'm a vehicle"**);**  **}**  **}** | | | | | | | |
| **Probleme Mehrfachvererbung** | | | | | | | |
| *Nameskonflikte* (Gleiche Memberdeklarationen in Parent), *Diamant Problem* (Instanzvariablen in mehrfach geerbten Basisklassen), *Compiler Komplexität* (Virtual Method Table ist nicht linear erweiterbar) | | | | | | | |
| **Interfaces (kein final)** | | | | | | | |
| Interfaces können public, package (oder private) sein.  Interface Methoden sind implizit public und abstract (kann weggelassen werden). | | | | | | | |
| interface RoadVehicle **extends** Vehicle **{**  public void tireModel**();**  **}** | | | public class Car **implements** RoadVehicle **{**  public void tireModel**(){...}**  **}** | | | | |
| **Default Methoden** | | | | | | | |
| Keine Instanzvariablen, nur Konstanten (implizit public static final) | | | | | | | |
| interface Vehicle **{**  int IMPLIZIT\_CONST\_FINAL **=** 2**;**    **default** void printModel**()** **{**  **...**  **}**  **}** | | | |  | | | Spezifischere Default Implementierung wird genommen |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exceptions** | | | | | | | | | | | | | | | | | | |
| String clip**(**String s**)** **throws** Exception **{**  **if** **(**s **==** **null)** **{**  **throw** **new** IllegalArgumentException**(**"String is null"**);**  **}**  **}** | | | | | | |  | | | | | | | | | | | |
| **try(**Scanner s **=** **new** Scanner**(**System**.**in**))** **{**  **...** // Multicatch: interface AutoClosable  **}** **catch(**NoStringException **|** ShortStringException e**)** **{}** | | | | | | |
|  | | | | | | 1) IndexOutOfBoundsExeption  2) Catch1, da RuntimeException spezifischer  3) A = null  4) Rethrow IndexOutOfBoundsException (wird in diesem Try-Catch nicht mehr abgefangen)  5) Finally  6) Nullpointer Exception  Nullpointer überschreibt rethrowed IndexOutOfBoundsException | | | | | | | | | | | | |
| private static final long serialVersionUID **=** **-**8326345074436495854L**;** | | | | | | | | | | | | | | | | | | |
| **Packages** | | | | | | | | | | | | | | | | | | |
| 1) Singletype Import ist stärker wie eigene Klasse A | | | | | | | | | | | | | | | **import** p1**.**A | | | |
| 2) Klassen aus eigenem Package | | | | | | | | | | | | | | | | | | |
| 3) Import on Demand | | | | | | | | | | | | | | | **import** p1**.\*** | | | |
| **Static Import: import** static java**.**lang**.**System**.**out**;** out**.**println**(**"Test"**);** | | | | | | | | | | | | | | | | | | |
| **JUnit Testing** | | | | | | | | | | | | | | | | | | |
| public class TestStack **{**  @Before  public void setUp**(){}**    @Test**(**timeout **=** 5000**,** expected**=**Exception**.**class**)**  public void testMethod**()** **{**  Stack stack **=** **new** Stack**(**1**);**  assertEquals**(**"expected"**,** "actual"**);**  assertTrue**(**"message"**,** stack**.**isEmpty**());**  assertNotNull**(**stack**);**  **}**  **}** | | | | | | | | | | | | | | | | Abhängigkeiten in Tests vermeiden  Relevante Fälle testen (Edge Cases: Null, Leer String, Case Sensitive, Exceptions etc.) | | |
| **JavaDoc** | | | | | | | | | | | | | | | | | | |
| /\*\*  \*@param name description  \*@return description  \*@throws type description  \*/ | | | | | | | | | | JavaDoc ist eine Spezifikation und keine Dokumentation! | | | | | | | | |
| **Equals / Hashcode (Java Spec: 2 Objekte müssen selben Hashcode liefern wenn equal)** | | | | | | | | | | | | | | | | | | |
| @Override  public boolean equals**(**Object obj**)** **{**  **if(this** **==** obj**)** **{**  **return** **true;**  **}** **else** **if(**obj **==** **null)** **{**  **return** **false;**  **}** **else** **if** **(**getClass**()** **!=** obj**.**getClass**())** **{**  **return** **false;**  **}** **else{**  Student other **=** **(**Student**)**obj**;** //cast  **return** regNumber **==** other**.**regNumber**;**  **}**  **}** | | | | | | | | | | | | | | | x.equals(y) 🡪 x.hashCode() == y.hashCode()  Ansonsten können inkonsistenzen beim Hashing auftreten. Objekt wird allenfalls nicht gefunden | | | |
| @Override  public int hashCode**()** **{**  **return** firstName**.**hashCode**()**  **+** 31 **\*** lastName**.**hashCode**()**  **+** 31 **\*** address**.**hashCode**();**  **}** | | | | | | | | | | **Hash Verfahren:**  1)Hashcode = Tabellenindex  2)>length 🡪 hashCode%length = index  3)Kollisionsliste bei Doppelbelegung | | | | | | | | |
| **Comparable / CompareTo(Type other)** | | | | | | | | | | | | | | | | | | |
| class Student **implements** Comparable**<**Student**>**  **{**  private int regNumber**;**  @Override  public int compareTo**(**Student other**)** **{**  **return** regNumber **-** other**.**regNumber**;**  **}**  **}** | | | | | | | | | | | | | <0: this ist kleiner als other  >0: this ist grösser als other  ==0: this ist gleich other  Auf Nullpointer achten! | | | | | |
| **Clone (Für alle Member neue Objekte erstellen. Ausgenommen primitive Datentypen)** | | | | | | | | | | | | | | | | | | |
| class Person **implements** Clonable **{**  private String firstName**,** lastName**;**  @Override  public Person clone**()** **{**  **return** **new** Person**(**firstName**,** lastName**);**  **}**  **}** | | | | | | | | | | | | | | | Shalow Clone /  Deep Clone: Auch aggregierte Objekte klonen | | | |
| **Collections** | | | | | | | | | | | | | | | | | | |
| List**<**String**>** list **=** **new** ArrayList**<>();**  list**.**add**([**1**,]**"Java"**);**  list**.**remove**(**2**);**  list**.**set**(**1**,**"OO"**);**  list**.**contains**(**"Java"**);**  list**.**indexOf**(**"Java"**);**  String s **=** list**.**get**(**1**);** | | | | | | | | | | | | Duplikate und Nulleinträge möglich  Dynamische Grösse (Faktor 1.5 grösser) | | | | | | |
| Set**<**String**>** set **=** **new** HashSet**<>();**  set**.**add**(**"Java"**);**  set**.**remove**(**"Java"**);**  set**.**contains**(**"Java"**);** | | | | | | | | | | | | Keine Duplikate  Unsortiert (TreeSet sortiert via Baum) aber schnell | | | | | | |
| Map**<**Integer**,** String**>** map **=** **new** HashMap**<>();**  map**.**put**(**123**,** "Java"**);**  String s **=** map**.**get**(**123**);**  map**.**containsKey**(**123**);**  map**.**containsValue**(**"Java"**);**  map**.**keySet**();**  map**.**values**();** | | | | | | | | | | | | Keine Duplikate (Key)  Key Value Paare  Unsortiert (TreeMap sortiert)  Effizient | | | | | | |
| **for** **(**Map**.**Entry**<**String**,** String**>** entry **:** map**.**entrySet**()){**  System**.**out**.**println**(**entry**.**getKey**()** **+** "/" **+** entry**.**getValue**());**  **}** | | | | | | | | | | | | | | | | | | |
| Deque**<**String**>** queue **=** **new** LinkedList**<>();**  queue**.**addLast**(**element**);**  queue**.**removeFirst**(**element**);** | | | | | | | | | | | | Optimal für FIFO und LIFO | | | | | | |
| **Generics** | | | | | | | | | | | | | | | | | | |
| class Node**<**T**,**U**>** **{**  private U first**;**  private T second**;**  **}**  class Node**<**T **extends** A **&** B **&** C **>** | | | | | | | | | interface Identifiable**<**T**>** **{**  T setId**(**T id**);**  **}**  public **<**T**>** T getValue**(**T param**){}** | | | | | | | | | |
| class Node**<**T **extends** Comparable**<?** **super** T**>** | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | **Type Erasure:** Ersetzen von Generischen Typen durch Object und Casts. (wegen Rückwärtskompatibilität). Deshalb keine primitiven Datentypen als Generic.  **Raw Types:** Generische Klassen ohne Typ-Argumente | | | | | |
| **Funktionsschnittstellen (@FunctionalInterface)** | | | | | | | | | | | | | | | | | | |
| Funktionsschnittstellen sind Interfaces mit genau einer Methode, welche eine passende Signatur und Rückgabetyp hat | | | | | | | | | | | | | | | | | | |
| **Predicate** | | @FunctionalInterface  public interface Predicate**<**T**>** **{**  boolean test**(**T element**);**  **}** | | | | | | | | | | | | | | | | |
| **Function** | | @FunctionalInterface  public interface Function**<**T**,** R**>** **{**  R apply**(**T t**);**  **}** | | | | | | | | | | | | | | | | |
| **Consumer** | | @FunctionalInterface  public interface Consumer**<**T**>** **{**  void accept**(**T t**);**  **}** | | | | | | | | | | | | | | | | |
| **Supplier** | | @FunctionalInterface  public interface Supplier**<**T**>** **{**  T get**();**  **}** | | | | | | | | | | | | | | | | |
| **Comparator** | | int compare**(**T o1**,** T o2**);** | | | | | | | | | | | | | | | | |
| **Comparator** | | | | | | | | | | | | | | | | | | |
| list**.**sort**(new** AgeComparator**());**  class AgeComparator **implements** Comparator**<**Person**>** **{**  @Override  public int compareByAge**(**Person p1**,** Person p2**)** **{**  **return** p1**.**getAge**()** **-** p2**.**getAge**();**  **}**  **}** | | | | | | | | | | | | | | | | | | |
| people**.**sort**(**Comparator**.**comparing**(**Person**::**getLastName**)**  **.**thenComparing**(**Person**::**getAge**).**reversed**());** | | | | | | | | | | | | | | | | | | |
| **Methodenreferenzen (Methoden als Objekt)** | | | | | | | | | | | | | | | | | | |
| people**.**sort**(this::**compareByAge**);**  int compareByAge**(**Person p1**,** Person p2**)** **{**  **return** p1**.**getAge**()** **-** p2**.**getAge**();**  **}** | | | | | | | | | | | obj**::**compareByAge  StaticClass**::**compareByAge | | | | | | | |
| **Lambda** | | | | | | | | | | | | | | | | | | |
| **Closure:** Zugriff auf umgebende lokale Variablen. Variable ist implizit final; Lamda lebt länger wie Variable 🡪 Kopie der Variable wird übergeben, die sich nicht ändern darf! | | | | | | | | | | | | | | | | | | |
| people**.**sort**((**p1**,** p2**)** **->** p1**.**getName**().**compareTo**(**p2**.**getName**());**  people**.**sort**((**p1**,** p2**)** **->** **{return** p1**.**getAge**()** **-** p2**.**getAge**();});**  people**.**removeAll**(**p **->** p**.**getAge**()** **>** 18 **&&** p**.**getAge**()** **<** 100**);** | | | | | | | | | | | | | | | | | | |
| public boolean matches**(**String value**,** Predicate**<**T**>** lambdaCriterion**)** **{**  **return** lambdaCriterion**.**test**(**value**);**  **}** | | | | | | | | | | | | | | | | | | |
| public **<**R**>** void print**(**Function**<**T**,** R**>** criterion**)** **{**  **}** | | | | | | | | | | | | | | | | | | |
| **Stream API** | | | | | | | | | | | | | | | | | | |
| people**.**stream**()**  **.**filter**(**p **->** p**.**getAge**()** **>=** 18**)**  **.**map**(**p **->** p**.**getLastName**())**  **.**sorted**().**distinct**().**limit**(**10**)**  **.**forEach**(**System**.**out**::**println**);** | | | | | | | | | List**<**Person**>** list **=** people**.**stream**()**  **.**sorted**((**p1**,** p2**)** **->** p2**.**getSalary**()** **-** p1**.**getSalary**())**  **.**collect**(**Collectors**.**toList**());** | | | | | | | | | |
| IntStream**.**iterate**(**0**,** i **->** i **+**1**)**  **.**limit**(**1000**)**  **.**reduce**((**i1**,** i2**)** **->** i1 **+** i2**);** //.sum(); | | | | | | | | | | | | | | | | | | |
| **Byte Stream (8Bit) / InputStream, OutputStream** | | | | | | | | | | | | | | | | | | |
| **try** **(**FileInputStream fis**=** **new** FileInputStream**(**"..."**))** **{**  int value **;**  **while((**value **=** fis**.**read**())** **>=** 0**)** **{**  byte b **=** **(**byte**)**value**;** **...**  **}**  **}** **catch(**FileNotFoundException **|** IOException e**)** **{**  System**.**out**.**println**(**"Error"**);**  **}** | | | | | | | | | OutputStream fos **=** **new** FileOutputStream**(**"serial.bin"**);**  **try** **(**ObjectOutputStream stream **=** **new** ObjectOutputStream**(**fos**))** **{**  stream**.**writeObject**(**person**);**  stream**.**flush**();**  **}**  **transient** Keyword wird nicht serialisiert | | | | | | | | | |
| **Character** **Stream (16Bit) / Reader, Writer** | | | | | | | | | | | | | | | | | | |
| **try** **(**BufferedReader bufferedReader **=** **new** BufferedReader**(new** FileReader**(**"test.txt"**))){**  String line**;**  **while** **((**line **=** reader**.**readLine**())** **!=** **null)** **{**  System**.**out**.**println**(**line**);**  **}**  **}** **catch(**FileNotFoundException **|** IOException e**)** **{**  System**.**out**.**println**(**"Error"**);**  **}** | | | | | | | | | | | | | | | | | | |
| Reader reader **=** **new** InputStreamReader**(**inputStream**,** "UTF-8"**);** //bridge  Writer writer **=** **new** OutputStreamWriter**(**outputStream**,** "UTF-8"**);** | | | | | | | | | | | | | | | | | | |
| **Regex (Default 🡪 Greedy Match=So viel wie möglich)** | | | | | | | | | | | | | | | | | | |
| **Joker** | . | | **1..n** | + | **0..n** | | | \* | **Min/Max** | | | | | a{2,5} | | | **NOT** | [^a] |
| Pattern pat **=** Pattern**.**compile**(**"([0-2]?\\d\*?):(?<MIN>[0-5][0-9])"**);**  Matcher match **=** pat**.**matcher**(**timeText**);** //\*? => reluctant match  **if(**match**.**matches**())** **{**  String p1 **=** match**.**group**(**1**);**  String p2 **=** match**.**group**(**"MIN"**);**  **}** | | | | | | | | | | | | | | | | | | |
| **Reflection** | | | | | | | | | | | | | | | | | | |
| Class**<?>** type **=** obj**.**getClass**();**  **for** **(**Field field **:** type**.**getDeclaredFields**())** **{**  String name **=** field**.**getName**();**  String typeName **=** field**.**getType**().**getSimpleName**();**  field**.**setAccessible**(true);**  Object value **=** field**.**get**(**obj**);**  **}** | | | | | | | | | | | | | | | | | | |
| **Annotations** | | | | | | | | | | | | | | | | | | |
| @Retention**(**RetentionPolicy**.**RUNTIME**)**  @Target**(**ElementType**.**TYPE**)** // just classes  public @interface Hidden **{**  // required without default  String reason**()** **default** "unknown"**;**  **}** | | | | | | | | | | | | | | | | | | |
| **Garbage Collection** | | | | | | | | | | | | | | | | | | |
| **Mark and Sweep:** Objekte die vom Root Set erreichbar sind markieren und nicht markierte während der Sweep Phase freigeben (evtl. Objekte = null setzen) | | | | | | | | | | | | | | | | | | |
| **Design Aspekte** | | | | | | | | | | | | | | | | | | |
| 1)Zusammengehörige Aspekte bündeln 2) Klare Benennung 3) Aggregation vor Vererbung 4) Keine zyklische Imports 5) Keine unnötigen Kommentare  6) Kein Overdesign 7) Instanzvariablen kapseln 8) Warnungen behandeln 9) Code Metriken 10) Keine Wiederholungen | | | | | | | | | | | | | | | | | | |
| **Primzahlen** | | | | | | | | | | | | | | | | | | |
| **for** **(**long factor **=** 2**;** factor **\*** factor **<=** number**;** factor**++)** **{**  **if** **(**number **%** factor **==** 0**)** **{**  **return** **false;**  **}**  **}**  **return** **true;** | | | | | | | | | | | | | | | | | | |
| **Palindrom** | | | | | | | | | | | | | | | | | | |
| public boolean isPalindrome**(**String value**)** **{**  **return** value**.**equals**(new** StringBuilder**(**value**).**reverse**().**toString**());**  **}** | | | | | | | | | | | | | | | | | | |
| **Zyklen erkennen** | | | | | | | | | | | | | | | | | | |
| private void checkCycles**(**Company company**)** **throws** Exception **{**  **if(this.**equals**(**company**))** **{**  **throw** **new** Exception**(**"cycle detected"**);**  **}**  **for(**Company curParticipant **:** company**.**participations**)** **{**  checkCycles**(**curParticipant**);**  **}**  **}** | | | | | | | | | | | | | | | | | | |