```
// Syntax = in muss geschrieben werden []=wahlweise
int x, y; //deklaliert x und y als intiger
int x = 5, y = 7; //deklaliert und speicher die werte dazu
final int x = 8 //bleibt immer 8
x = X + 1
// 3.14 double
//3.14f float
// 13E4 13*e4

float f; int i;
f=i; //erlaubt
i=f; //verboten
i=(int)f; //erlaubt, schneidet nachkommazeilen oder mehr ab
f=1.0; //verboten weil 1.0 double ist
f=1.0f; //ok
//double + int //double
//float + int //float
//short + short //int
//if (n!=0) x = x/n; // if (True) mach x =...

if (x>y)
  max = x;
else
  max = y;
// syntax ïf(Ëxpresion ")SStatment [ëlseSStatment]
if (x>y){
// fuer mehr Statements  nennt sich Block
  max = x;
  System.out.println(x);
}
else
  max = y;
int x=0; // muss int oder float sein
x++ //x=1 x++=>x+= oder x=x+1
x-- // x=x-1
== //gleich
!= // ungleich
> //grösser
< //kleiner
>= //grööser oder gleich
<= // kleiner oder gleich
&& // und-Verknüpfung if (... and ... ==True) mach ...
|| // oder-Verknüpfung
!x //nicht-Verknüpfung
x==True
!x==False
boolean p, q;
  p=false;
  q = x<0;
whileschleife
i = 1;
sum = 0;
while (i<=n){ //schleifenbedingung //solange i<= n tu das...
  sum = sum + i; //Schleifenrumpf
```

```
  i += 1;  //
}
do-schleife
int n = In.readInt(); //Zahl von Tastatur
do {
  System.out.println(n%10); // mach das... und wiederhole wenn..
  n=n/10;
} while (n>0);
for-Schleifen
sum = 0;
for (i = 1; i <= n; i++ ) // 1. bestimmung einer Variablen,
  sum = sum + 1;  // 2.voraussetzung zur weiterführung
        // 3. veränderung der Vriable
        // "for("[ForInit] ";"[Expression] ";"[ForUpdate]";")
         // ForInit = Assigment (",Ässigment)
  int = 0;   // = Zahlen Typ deklarieren | Type n =.. (","n=..)
              // ForUpdate = Assigment (",Ässigment)

for (;;){} //endlosschelife
  if (..)break; // Beendet die Schleife in der es sich befiendet

hours_K = JOptionPane.showInputDialog("frage");
//öffnet interaktive Fenster und speichert antwort in variable (parse geht auch)
args[0].equals("String"); //vergleicht Input/Varible mit String

void //nimmt keinen wert an. die methode nimmt keinen wert an kann auch nicht
    zurück geben

static void "methodname"(){ //methodealggm.
}

static void numbers(int x,int y){ //methodenbeispiel
  if (x>y) system.out.println(x);
  else system.out.ptintln(y);
}

static int max(int y, int x){ //Funktion, kein void, gib eine Wert zurück, muss
    mit return enden
  if (x>y) return y;
  else return x;
}
 return //kann als break verwendet werden, springt aus der methode

int[] a; //Array a as int
float[] b; //Array b as float

a = new int[5]; //how many varibles can fit in
b = new float[10];

int[][] a; //zweidimensionales array
a = new int[3][4];
int[][] a = {{1,3,5},{3,5,9},etc}

int[][] a = new int[3][]; //für unterschiedliche längen. (sehr selten brauchbar)
a[0] = new int[4];
```

```
a[1] = new int[2];
etc.

b = null; //can't find the array of b anymore
a = b; //greifen jetzt auf den gleichen Array zu, kann über beide variabeln
    verändert werden

b = (int[]) a.clone(); //koopiert a in b reine
a.lenght //die länge von a +1


char ch = "\u0041" //"Õ041ëntspricht Äïm ascicode
      // 0*16e3 + 0*16e2 + 4*16e1 + 1*16e0 A-F entsprechen 11-16 im 16
         zahlensystem

char bonni = 'c';
int i = bonni; //chars können integers zugewiesen werden

10 + (bonni - '0') //ergebniss int, bonni=99 '0'=48 => (99-48) + 10

if (Character.isLetter(bonni)) //true wenn bonni ein Unicode-Buchstabe ist
"" //for Strings
'' //for chars

if (s.equals("Hellow")) //kann nurnso direkt verglichen werden

//=====================================================
class Date{       //definition einer Variable
  int day;
  String month;
  int year;

    String[] a = {"a", "b", "c"};
for (String element : a) { // gibt a, b und c aus
    System.out.println(element);
}
}

//-----------------------------------------------------

interface "NameOfInterface" { //definiert ein Interface, Interface
    spezifiezieren Methoden welche ein Datentyp zur verfügung stellen muss.
        void scale(double scaleFactor); // methode mit allen Parameter ausser der
            methode selbst
    } // classen die "NameOfInterfaceïmplementieren müssen in sich die Methoden
        aus dem Interface definieren

// Beispiel
class Circle implements Movable, Scalable { //Movable und Scalable haben in sich
    translate, moveToOrigin und scale, diese müssen jetzt in Circle definiert
    werden
    double xPos;
    double yPos;
    double radius;

    public void translate(double x, double y) {
        this.xPos = this.xPos + x;
```

```
        this.yPos = this.yPos + y;
    }

    public void moveToOrigin() {
        this.xPos = 0;
        this.yPos = 0;
    }

    public void scale(double scaleFactor) {
        this.radius = this.radius * scaleFactor;
    }
}

public static void moveAround(Movable m, double xv, double yv) { //m ist vom Typ
    Movable und hat deshalb die Methode translate zur verfügung, Interfaces könne
    auch als Datentypen verwendet werden
    m.translate(xv, yv);
}

//Somit sind solche dinge möglich
Circle aCircle = new Circle();
Point aPoint = new Point();

moveAround(aCircle, 10, 20);
moveAround(aPoint, 0, 1);

abstract class Fruit { //das ist eine abstracte classe, sie vererbt ihre
    eigenschaften(methoden, variabeln) an ihre subcalssen weiter wenn sie
    extended wird
    String name;       // von abstracten klassen könne keine instanzen erstellt
        werden
    String color;

    public Fruit(String name, String color) { //Konstruktor
        this.name = name;
        this.color = color;
    }

    void eat() { System.out.println("eat"); }
}

class Banana extends Fruit {

    public Banana() { //Konstruktor
        super("Banana", "yellow"); //super weist zu name = Banana, color = yellow
    }
}




abstract class Fruit {

    abstract void prepare();

}
```

```java
class Banana extends Fruit {

    void prepare() {
        System.out.println("peel");
        System.out.println("eat");
    }
}

class Mango extends Fruit {

    void prepare() {
        System.out.println("peel");
        System.out.println("cut");
        System.out.println("eat");
    }

}

class FruitExample {

    static void preparationInstructions(Fruit aFruit) {
        aFruit.prepare();
    }

    public static void main(String[] args) {
        Banana aBanana = new Banana();
        Mango aMango = new Mango();

        preparationInstructions(aBanana); //output: peel eat
        preparationInstructions(aMango); // output: peel cut eat
    }
}

//-----------Generics---------------------------------
interface List {
    // adds an Element to the list
    void add(String element);

    // returns the element at the given position
    String get(int index);

    // ...
}

interface List<E> { // parametrischer Datentyp, kann jedem Datentyp entsprechen
    // adds an Element to the list
    void add(E element);

    // returns the element at the given position
    E get(int index);
}

// Deklariert Variable vom Type List<String>, Ë wird zu String"
```

```
List<String> stringList;

//Syntax
List<Double> reverse(List<Double> l) {
    // Implementation
}

//Beispiele Nutzung

// Definition einer konkreten LinkedList Klasse die das Interface definiert
class LinkedListOfDoubles extends List<Double> {
    // Implementation
}

class LinkedList<E> extends List<E> {
    // Implementation
}

// Deklariert Variable vom Typ List<Integer> und initialisiert diese mit einer
    leeren verketteten Liste
List<Integer> intList = new LinkedList<Integer>(); //geht auch mit mehreren
    Datentypen

List<int> //nicht möglich, geht nur mit Referenztypen

List<Integer> list = new LinkedList<Integer>();
int e = 5;
list.add(e); // e wird automatisch in einen Integer umgewandelt.

//----------Typeinschraenkung--------------------------------------

interface Comparable<T> { //check ned was an däm ganze bispiel sätt zeigt wärde
    int compareTo(T other);
}


class Integer implements Comparable<Integer> {
    int value;

    public int compareTo(Integer other) {
        if (value == other.value) {
            return 0;
        } else if (value < other.value) {
            return -1;
        } else {
            return 1;
        }
    }
}

class OrderedTuple<E extends Comparable<E>> {

    E value1;
    E value2;
```

```
    OrderedTuple(E value1, E.value2) {
        if (value1.compareTo(value2) == -1) {
            this.value1 = value1;
            this.value2 = value2;
        } else {
            this.value1 = value2;
            this.value2 = value1;
        }
    }
}

//------Exceptions----------------------------------------

//urlhttps://dmi-programming.dmi.unibas.ch/gdp/theory/exceptions/

//class Exceptions für Fehler die nicht häufig vorkommen aber vorkommen können
//rückgabewert einer Klasse die für häufig aufretende Fehler ist Optional

//Exceptions können wir selber definieren oder die von Java nehmen
//Exceptions erben immer von der Klasse Exception

class MyException extends Exception { //Beispiel selbst geschriebe exception,
    MyException(String errorMessage) {
        super(errorMessage);
    }
}

if (unexpectedSituation) {
    MyException myException = new MyException("something unexpected happens");
    throw myException; // mit dem schlüsselwort "throw"werfen wir die Exception,
        nach throw könne wir mir Kommas getrennt alles Fehler auflisten
}

//wenn eine methode eine Fehler könnte auslösen
try { // Start des geschützten Bereichts
    // ...
    exampleMethod() // hier könnte ein Fehler auftreten
    // ...
} catch (Exception e) {
    // Fehlerbehandlung
}

//die finally clausel
try { // Start des geschützten Bereichts
    // ...
    exampleMethod() // hier könnte ein Fehler auftreten
    // ...
} catch (Exception e) {
    // Fehlerbehandlung
} finally {
    // Code der immer ausgeführt werden muss
}

//Für die RuntimeExceptions (und ableitungen dafon) muss keine deklaration
    gemacht werden das sie immer aufreten könnte
```

```
// und auf die wir auch nicht richtig reagieren können

//-----------------Optionals--------------------------
//Beispiel für anwendunng Optionals
class Employees {
    public Employee getEmployeeWithId(int id) { /* ...*/ } //wenn id nicht
        vorhanden, error aber eig vorhesebar also keine Exceptions
}

//oft verwendet methode diese Problem zu lösen, aber scheisse, weil nicht
    ersichtlich das ein error kann auftauchen
int id = 5;
Employee e = employees.getEmployeeWithId(id);
if (e != null) {
   // Normale Programmlogik
} else {
   // Logik für den Fall, dass Employee nicht gefunden wurde.
}

//class Optional als besser methoden von java
class Optional<T> {
    T value = null;

    private Optional(T value) {
        this.value = value;
    }
    static <T> Optional<T> of(T value) {
        return new Optional<T>(value);
    }

    static <T> Optional<T> empty() {
        return new Optional<T>(null);
    }

    boolean isPresent() {
        return this.value != null;
    }

    public T get() {
        if (value == null) {
            throw new NoSuchElementException("No value present");
        }
        return value;
    }
}

//Fehlerbehandlung
class Employees {
    public Optional<Employee> getEmployeeWithId(int id) { /* ...*/ }
}

int id = 5;
Optional<Employee> maybeAnEmployee = employees.getEmployeeWithId(id); //zeigt
    das Fehler könnte auftauchen
if (maybeAnEmployee.isPresent()) {
    Employee employee = maybeAnEmployee.get() //kann nur über get auf employees
```

```
        mit id zugreifen und falls nicht da wird Fehler angezeigt
    // Normale Programmlogik
} else {
    // Logik für den Fall, dass Employee nicht gefunden wurde.
}

static //vor einer Methode oder Feld, heisst kann genutzt werden ohne Instanz
    der Klasse erstellt zu haben
```