

# JAVA Einführung und Befehle

## **Basis: Java 1.4**

### **Allgemeine Hinweise zum Layout:**

Java-Befehle, also dass was man tatsächlich eintippt sind in `CourierNew` geschrieben.

Variablen des Codes werden im *kursiven CourierNew* geschrieben.

- JVM = Java Virtual Machine -> führt Java-Bytecode aus
- JRE = Java Runtime Environment -> JVM + Standardbibliotheken (API)

### Befehle zum Programm erstellen:

- `java ProgramName` -> Java Bytecode Interpreter ausführen
- `javac ProgramName.java` -> Java Compiler
- `javadoc ProgramName.java` -> Generator für API Dokumentation
- `appletviewer` -> Java Applets ausführen

### **Für Umsteiger:**

- Java ist grundsätzlich objektorientiert. Lediglich die Grundtypen wie `int`, `boolean`, `float`, `double`, `char` sind keine Objekte
- Java ist kostenlos erhältlich. Erfinder und Hauptentwickler ist Sun Microsystems.
- Funktionen/Prozeduren heißen in Java Methoden
- Klassenvariablen/-methoden werden in der Klasse selbst deklariert
- Instanzvariablen/-methoden werden erst im Objekt erzeugt und nur vom Objekt benutzt
- aus C, C++ bekannte Pointer gibt es nicht (jedenfalls fast nicht)

### **Kommentare:**

- `// ...` -> kommentiert eine Zeile aus
- `/* ... */` -> kommentiert Zeilenblock aus
- `/** ... */` -> kommentiert Zeilenblock aus (wird von `javadoc` berücksichtigt)

## Grundgerüst:

```
public class KlassenName {
    public static void main(String[] args) {
        ..
        ..
    }
}
```

## Klasse definieren:

```
public class KlassenName {
    public String oeffentlicheVariable;
    private integer privateVariable;

    // Konstruktor ohne Parameter
    public KlassenName() {
        this.oeffentlicheVariable = „offen“;
        this.privateVariable = 110;
    }

    // Konstruktor mit Parameter
    public KlassenName(String s, int i, boolean b) {
        this.oeffentlicheVariable = s;
        this.privateVariable = i;
        ...
    }
}
```

## Objekt aus Klasse erstellen (meist in eigener Datei):

### ohne Parameter:

```
KlassenName neueKlasse = new KlassenName();
```

### mit Parameter:

```
KlassenName neueKlasse = new KlassenName(„einString“, 110, true);
```

## **Instanzmethoden:**

### Methode ohne Rückgabe eines Wertes:

```
// Klasse ist bereits definiert
public void MethodenName() {
    ..
    ..
}
```

### Methode mit Rückgabe eines Wertes:

```
// Klasse ist bereits definiert
public String MethodenName(String s, int i, boolean b) {
    ..
    ..
    return „hier steht jetzt ein String, kann auch int, usw. sein“;
}
```

### return: springt aus Methode raus.

- **return;** -> keine Rückgabe
- **return „irgendwas“;** -> "irgendwas" als Rückgabe (kein Output auf Screen!!!)

### Bildschirmausgabe:

```
System.out.println(„irgendein String oder andere Variablen“);
```

Hinweis: neue Zeile mit: ' + „\n“ '

### Kapselung:

- lässt Änderung von Variablen zu, wenn diese als **private** deklariert sind
- Zugriff der Variablen nur über Methoden möglich!

```
public AendereWert(String wert) {
    this.wert = wert;
}
```

## Anweisungen:

### WHILE-Anweisung:

```
while (int x = 0) {  
    ..  
    ..  
}
```

### DO-Anweisung:

```
do {  
    ..  
    ..  
}  
while (x == y);
```

### FOR-Anweisung:

```
for (int i = 0; i < 1000; i++) {  
    ..  
    ..  
}
```

### IF-Anweisung:

```
if (x > y) {  
    ..  
}  
else {  
    ..  
}
```

### SWITCH-Anweisung:

```
int c;  
switch( c ) {  
    case '10' :  
        result = „c hat Wert 10“;  
        break; // damit werden die nächsten Fälle nicht beachtet  
    case '20' :  
        result = „c hat Wert 20“;  
    default :  
        result = „c war weder 10 noch 20“;  
}
```

## Sprunganweisungen:

- **break:** verlässt direkt umschliessenden Block
- **continue:** springt zum Ende der Schleife und wertet booleschen Kontrollausdruck aus
- **return:** beendet Methodenausführung und gibt ggf. einen Wert zurück

## **Typen:**

### Strings:

#### - Strings vergleichen:

```
- if (string1.equals(string2)) {  
    ..  
}  
- if (string1.equalsIgnoreCase(string2)) {  
    ..  
}
```

#### - StringBuffer:

- effizientes (speichersparendes) Arbeiten mit Strings
- Methoden: `append()` , `insert()` , `setCharAt()`

```
StringBuffer sb = new StringBuffer();  
while (einString != null) {  
    sb.append(einString);  
    einString = neuerString(); // Methode neuerString() fiktiv
```

### Zahlen (int , double , float):

- ' % ' bedeutet modulo
- `a++` -> erhöhe `a` um 1, nachdem es benutzt wurde (Postinkrement)
- `++a` -> erhöhe `a` sofort um 1 und dann benutze es (Preinkrement)
- ' != ' bedeutet ungleich (gilt auch für andere Typen als Zahlen!)

### Logische Operatoren:

- UND : & sowie &&
- OR: | sowie ||
- NOT: !
- XOR: ^

Vergleich: ==

## Zuweisungsoperatoren:

- = einfache Zuweisung
- += , -= , \*= , /= Kurzschreibweisen (z.B: 'a = a + 3' -> 'a += 3')

## Arrays:

### Deklaration:

```
int[] zahlen;  
String[][] tabelle;
```

### Erzeugung:

```
zahlen = new int[100];  
tabelle = new String[20][5];
```

### Kurzform:

```
String[][] tabelle = new String[20][5];
```

### Elementzugriff:

```
zahlen[0] = 115;
```

## Vektoren:

- Container für alle möglichen Variablen

### Implementierung:

```
import java.util.Vector;
```

### Benutzung:

```
Vector myVector = new Vector();  
String s = „hoppla“;
```

- Element hinzufügen: `myVector.addElement(s);`
- Element an Stelle x ausgeben: `myVector.elementAt(x);`