

Java ist eine objektorientierte Programmiersprache.

Objekte werden abstrahiert in Klassen, Klassen enthalten Attribute und Methoden.

Beziehungen:

- Ist-ein-Beziehung zwischen Objekten steht für Vererbung
- Hat-ein-Beziehung zwischen Objekten und ihren Attributen

Java ist eine **kontextintensive Sprache**, d.h. auf Groß- und Kleinschreibung ist zu achten

Zugriffsbeschränkungen:

- public
- protected: Attribute und Methoden innerhalb der Vererbungshierarchie verwendet werden können
- private: nur innerhalb dieser einen Klasse sind Attribute und Methoden zugreifbar, auch bei Vererbung nur innerhalb dieser Klasse
=> wenn sie nur dort gebraucht werden
- static: public static void main: Instanz einer Klasse
 static Methode kann nur auf Variablen zugreifen, die innerhalb angelegt sind
static heißt, dass es nur einmal existiert, nicht dass es eine Konstante ist!

```
Public class Loop1 {  
public void main(String [] args);    // öffentlich, ohne Rückgabe, Hauptfunktion  
{}
```

```
while (a<b) {}
```

abweisende Schleife:

while-Schleife prüft den Ausdruck. Ist er true, wird die Anweisung ausgeführt, andernfalls wird mit der ersten Anweisung hinter der Schleife weitergemacht. Überprüfung geht dann weiter, bis false herauskommt.

```
do {} while (a<=b);
```

nichtabweisende Schleife:

do-Schleife wird mindestens einmal ausgeführt, da zuerst die Schleifenanweisung ausgeführt und erst dann der Testausdruck überprüft wird.

```
for (int i= 0; i<5; i++) {}
```

```
for (init; test; update)
```

1. **init:** wird einmal vor dem Start der Schleife aufgerufen, er dient zur Initialisierung der Schleife und darf auch aus mehreren Ausdrücken bestehen
2. **test:** wird analog zur while-Schleife am Schleifenanfang ausgeführt. Schleifenanweisung wird nur ausgeführt, wenn die Auswertung true ergibt. Fehlt er, so setzt der Compiler an seiner Stelle die Konstante true
3. **update:** dient dazu den Schleifenzähler zu verändern. Er wird nach jedem Durchlauf der Schleife ausgewertet, bevor der Test-Ausdruck das nächste Mal überprüft wird

die for-Schleife ist eine Zählschleife. Der Schleifenkopf besteht aus drei Ausdrücken, die jeder für sich optional ist.

- Normalfall:
for (int i= 0; i<5; i++) { }
- Inkrementierung der Laufvariablen an anderer Stelle
for (int i= 0; i<5;) { summe += i++; }
- Abbruch der Schleife an anderer Stelle
for (int i= 0; ;) { summe += i++; if (i>4) break; }
- Verwendung einer Variablen, die ausserhalb der for-Schleife existiert
for (;) { summe +=5; if (summe > 100) brak; }

if (a<b) {} else {}

if (i<5) continue; // fängt er wieder oben an bis i = 5, dann wird das ausgeführt, was nach continue steht.

Break und continue:

- break: taucht innerhalb einer Schleife break auf, wird die Schleife verlassen und das Programm mit der ersten Anweisung nach der Schleife fortgesetzt
- continue: taucht innerhalb einer Schleife continue auf, springt das Programm an das Ende des Schleifenrumpfes und beginnt mit der nächsten Wiederholung

Die Lebensdauer und Sichtbarkeit einer Variablen

Erstreckt sich ausschließlich auf den Block, in dem sie deklariert wurde, einschließlich der darin enthaltenen Blöcke.

switch ()

Die Switch-Anweisung ist eine Mehrfachverzweigung. Zunächst wird der Ausdruck ausgewertet und dann in Abhängigkeit vom Ergebnis der Sprungmarke angesprungen, deren Konstante mit dem Ergebnis des Ausdrucks übereinstimmt. Das optionale default-Label wird nur dann angesprungen, wenn keine passende Sprungmarke gefunden wird.

```
switch (ausdruck)
{
    case constant:
        anweisung; break;
    ...
    default:
        anweisung; break;
}
```

```
if(args.length == 0)
{System.err.println("falsche Zahl");
System.exit(1);}

int m = new Integer(args[0]).intValue();
switch (m) {
    case 1:
        System.out.println("Jan");
        break;
    default:
        System.err.println(m + "kein Monat");
        break;
}
```

Der new-Operator

In Java werden Objekte und Arrays mit Hilfe des new-Operators erzeugt. Sowohl das Erzeugen eines Arrays als auch das Erzeugen eines Objekts sind Ausdrücke, deren Rückgabewert das gerade erzeugte Objekt bzw. Array ist.

Bei einem Array muss die Größe bekannt sein und es gilt nur für primitive Datentypen.

```

int[] [] a1; // Verweis auf ein Array von Integerzahlen

int[] [] a2 = new int[8][] // Verweis auf ein bestehendes 2dim Array von Integerzahlen

int[] [] a3 = {{1,3,3}{4,9}{0,50}} //Literales Anlegen eines Arrays

for (int i=0; i<a3.length;i++) // Ausgabe des Inhalts von a3 auf den Bildschirm
    System.out.print(a3[i] + “, ”);
    System.out.println();

for (int i = 0; i<b2.length; i++) // Nachträgliches Erzeugen der 2. Dimension
    b2[i] = new int[5];

int [] [] b3 = new int[8][5]; // Erzeugen eines kompletten 2dim Arrays (8Z, 5S)

for (int i = 0; i < b4.length; i++) // Zeilen
    { for (int h = 0; h < b4[i].length; h++) // Spalten
        System.out.println(b4[i][h] + “, “); // gibt den Inhalt des Arrays aus
    }

```

Arrays

Array-Variablen gehören zur Klasse der Referenztypen. Ihre Größe wird zur Laufzeit des Compilers festgelegt und kann später nicht mehr verändert werden (semidynamisch) Jedes Array hat eine Instanzvariable `length`, die die Anzahl seiner Elemente angibt.

Referenztypen

Referenztypen sind neben den primitiven Datentypen die zweite wichtige Klasse von Datentypen in Java. Anders als die primitiven Datentypen, die fest an ein Objekt gebunden sind, stellen Referenztypen nur einen Verweis auf ein Objekt dar. Während primitive Typen lediglich deklariert werden, reicht dies bei Referenztypen nicht aus. Sie müssen mit Hilfe des `new`-Operators oder – im Fall von Arrays und Strings – durch Zuweisung von Literalen zusätzlich noch explizit erzeugt werden.

Strings können auf drei verschiedene Arten erzeugt werden:

- `String s1 = „s1“`
- `String s2 = new String(“s2”)`
- `String s1;`
`s1 = new String(s2);`

String-Objekte gehören zur Gruppe der Referenztypen in Java. Sie definieren eine beliebige Zeichenkette und werden mit Hilfe des `new`-Operators oder durch literale Zuweisung erzeugt.