Kapitel 2: Grundelemente der Programmierung

Variablen

- Variablen sind **Speicherbereiche** im Arbeitsspeicher
- Anhand eines Namens kann man Werte „hineinschreiben“ und auch wieder auslesen
- Variablen besitzen ein festes Format, das durch einen **Datentyp** festgelegt ist (z.B. Ganze Zahl, Zeichenkette)

Beispiel

```
int x = 5, quadrat;
quadrat = x * x;
System.out.println("Das Quadrat von "
                   + x + " ist " + quadrat);
```

- Speicherabbild:

- Im Speicher stehen eigentlich nur die Werte, nicht die Namen der Variablen. Letztere sind Aliasnamen für Adressen.

Elementare Datentypen

Beschreibung	**Typ**	**Beispiel werte**	**Wertebereich**	**Größe**
Zeichen	char	‘A’, ‘B’, ‘$’, ‘%’	Beliebiges Zeichen	2 Byte
Boolescher Wert (wahr, falsch)	boolean	true, false	true, false	1 Byte
Ganze Zahl	byte short int long	-17, 123	-128 bis +127 -32.768 bis +32.767 -2.147.483.648 bis +2.147.483.647 -9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807	1 Byte 2 Byte 4 Byte 8 Byte
Gleitkommazahl	float double	3.14, 1.414	-3,40282347E+38 bis +3,40282347E+38 -1,797693134862315 7E+308 bis +1,797693134862315 7E+308	4 Byte 8 Byte

Typumwandlung (Elem. Typen)

Umwandlung von Typ der Spalte nach Typ der Zeile:

	byte	short	int	long	float	double	boolean	char
byte		e	e	e	e	e	x	e
short	i		e	e	e	e	x	e
int	i	i		e	e	e	x	i
long	i	i	i		e	e	x	i
float	i	i	i	i		e	x	i
double	i	i	i	i	i		x	i
boolean	x	x	x	x	x	x		x
char	e	e	e	e	e	e	x	

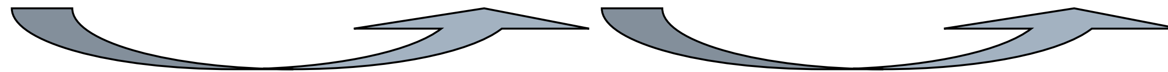
- i: implizite Typumwandlung möglich
- e: nur explizite Typumwandlung möglich (Cast-Operator notwendig)
- x: keine Typumwandlung möglich

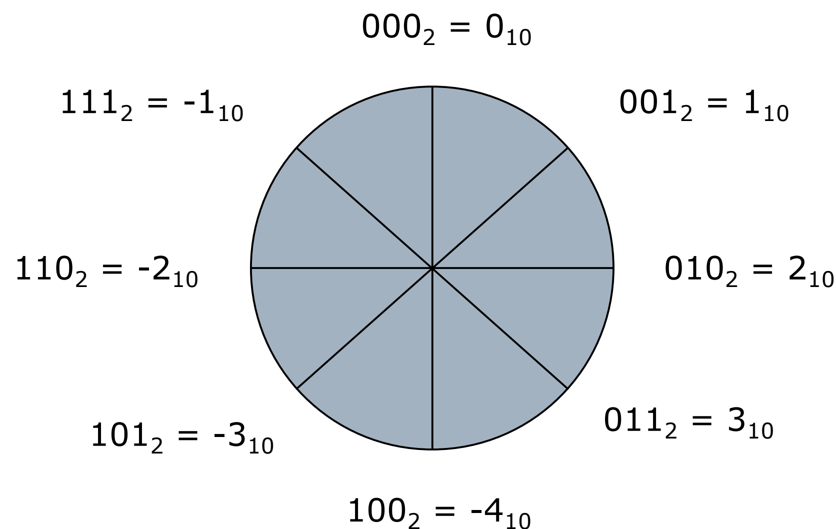
Beispiel

- ```java
  short x = 3;
  byte h = (byte)x;          // expliziter Cast
  short y = h;               // impliziter Cast
  ```

- Hier wird i.d.R. der Wert von x und y vertauscht.
- Was passiert, wenn der Wert von x zu groß für ein `byte` ist?
- Antwort (Beispiel: x = 130):

x : short	h : byte	y : short
130_{10}	-126_{10}	-126_{10}
0000000010000010_2	10000010_2	1111111110000010_2



„Abschneiden“ Korrekte Konvertierung
  ```

Interne Darstellung ganzer Zahlen

- Zahlenring für einen 3bit-Datentyp
- Der Zahlenring ist so konstruiert, dass gilt: -1 + +1 = 0
- Der Datentyp reicht von -4 bis +3.



- Die führende „1“ symbolisiert quasi ein negatives Vorzeichen.

Darstellung von Literalen

Beispiel		**Typ**
`true, false`	Bool`scher Wert	`boolean`
	Ganze Zahl	
`1234`	- Dezimalzahl	`byte/short/int`
`0x1234`	- Hexadezimalzahl	`byte/short/int`
`01234`	- Oktalzahl (führende Null!)	`byte/short/int`
`1234l`	- Dezimalzahl (endet mit kleinem „L“!)	`long`
`2.718281`	Gleitkommazahl	`double`
`'a'`	Einzelnes Zeichen	`char`
`"Hallo"`	Zeichenkette	`String`

Beispiele

Was passiert jeweils bei den folgenden Anweisungen?

- `float x = 1.23;`
- `long x = 123123123123;`
- `short x = 123;`
- `int x = 123;`
- `short x = 123123;`
- `short x = (short)123123;`
- `double x = 123123123123;`
- `double x = 123123123123.0;`
- `double x = 1.123123123123123123123;`
- `int x = 1; short y = x;`

Operatoren

- Arten nach Zahl der Operanden (Stelligkeit)
  - Unär: ein Operand (z.B. „++“)
  - Binär: zwei Operanden (z.B. „*“)
  - Trinär: drei Operanden (z.B. „b?t:f“)
- Assoziativität
  - Linksassoziativ: Abarbeitung von links nach rechts
  - Rechtsassoziativ: Abarbeitung von rechts nach links
- Über die tatsächlich ausgeführte Operation entscheiden:
  - Operatorsymbol
  - Stelligkeit (z.B. „-“ existiert unär und binär)
  - Typen der Operanden (z.B. bei mindestens einem Gleitkommatyp (`float`, `double`) bei einer Division erfolgt eine „Gleitkommadivision“, sonst eine ganzzahlige Divsion)

Prioritätentabelle

Priorität	**Operatoren**	**Assoziativität**
1	() [] . expr++ expr--	links
2	! ~ -unär +unär ++expr --expr	rechts
3	new (type)	links
4	* / %	links
5	+ -	links
6	<< >> >>>	links
7	< <= > >= instanceof	links
8	== !=	links
9	& (bitweises Und)	links
10	^ (bitweises exclusives Oder)	links
11	| (bitweises Oder)	links
12	&& (logisches Und)	links
13	|| (logisches Oder)	links
14	?:	rechts
15	= += -= *= /= %= ^= &= |= <<= >>= >>>=	rechts

Beispiele

Was passiert jeweils bei den folgenden Anweisungen?

- ```
  int x = 3 + 4 / 2 * 5;
  ```
- ```
  double x = 5 / 3;
  ```
- ```
  int x = 3;
  int y = 5;
  int z = x+++y;
  ```
- ```
  int x = 3;
  int y = 5;
  int z = 4 + x == y;
  ```
- ```
  int x = 5.0 / 3;
  ```
  ```

Kontrollstrukturen

- Sequenz
- Verzweigung (if-else, switch)
- Schleife (for, while, do-while)

if-Anweisung

- Beispiel:

```java
int x = 5;

if (x > 10) {
    System.out.println("x ist größer als 10");
}
else {
    System.out.println("x ist nicht größer als 10");
}
```

- Der else-Zweig der if-Anweisung ist optional.
- „x > 10" ist ein logischer bzw. boolescher Ausdruck

Logische Ausdrücke

- Logische Ausdrücke sind entweder wahr („`true`") oder falsch („`false`")
- Logische Ausdrücke können durch **Vergleiche** dargestellt werden.
- Vergleiche werden mittels **Vergleichsoperatoren** gebildet:

Operator	**Bezeichnung**	**Beispielausdruck**
<	kleiner	a < b
>	größer	a > b
<=	kleiner-gleich	a <= b
>=	größer-gleich	a >= b
==	ist gleich	a == b
!=	ungleich	a != b

Logische Ausdrücke (2)

- Mehrere Vergleiche können durch **logische Operatoren** miteinander verknüpft werden.
- Solche **logischen Verknüpfungen** stellen auch wieder logische Ausdrücke dar.

<table>
<tr><th>Operator</th><th>Bezeichnung</th><th>Beispielausdruck</th></tr>
<tr><td>&&</td><td>UND</td><td>a < b && a < c</td></tr>
<tr><td>||</td><td>ODER</td><td>a < b || a < c</td></tr>
<tr><td>!</td><td>NICHT</td><td>!(a < b)</td></tr>
</table>

- Wahrheitstabellen:

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

<table>
<tr><th>a</th><th>b</th><th>a || b</th></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>true</td><td>true</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
</table>

a	!a
true	false
false	true

switch-Anweisung

- Beispiel:

```
int anzahlKinder = 3;
int kindergeld;

switch (anzahlKinder) {
   case 0: kindergeld = 0; break;
   case 1: kindergeld = 1000; break;
   case 2: kindergeld = 2200; break;
   case 3: kindergeld = 3700; break;
   default: kindergeld = anzahlKinder * 1500;
}

System.out.println("Für Ihre " + anzahlKinder +
   " Kinder erhalten Sie ein Kindergeld von " +
   kindergeld + " Euro pro Monat.");
```

- `switch` erlaubt lediglich eine Prüfung auf Gleichheit und keine beliebigen logischen Ausdrücke wie eine `if`-Anweisung
  - ⇒ Performancevorteil: `case` kann intern erreicht werden; kein Vergleich je `case` notwendig
- Wichtig: `break` nicht vergessen!

for-Schleife

- Beispiel:

```
for (int i = 0; i < 5; i++) {
   System.out.println("Hello World");
}
```

Initialisierung	i = 0
Ausführbedingung	**Schleifendurchläufe**
0 < 5 1 < 5 2 < 5 3 < 5 4 < 5 5 < 5	➔ 1. Durchlauf: Ausgabe von „Hello World“, i = 1 ➔ 2. Durchlauf: Ausgabe von „Hello World“, i = 2 ➔ 3. Durchlauf: Ausgabe von „Hello World“, i = 3 ➔ 4. Durchlauf: Ausgabe von „Hello World“, i = 4 ➔ 5. Durchlauf: Ausgabe von „Hello World“, i = 5 ➔ Abbruch

- Allgemein:

for (**Initialisierungsanw.**; **Ausführbedingung**; **Aktualisierungsanw.**) {
  Anweisung(en)
}

while- und do-while-Schleife

While-Schleife

- Syntax:

  while (**Ausführbedingung**)
  {
  	Anweisung(en)
  }

- Kopfgesteuerte Schleife
- Möglicherweise kein einziger Schleifendurchlauf

Do-While-Schleife

- Syntax:

  do {
  	Anweisung(en)
  }
  while (**Ausführbedingung**);

- Fußgesteuerte Schleife
- Mindestens ein Schleifendurchlauf

Schleifen allgemein

- Mit **`break`** kann der Schleifendurchlauf abgebrochen werden.
- Mit **`continue`** ist ein vorzeitiges Springen zur Schleifenbedingung möglich.
- In Java sind Schleifbedingungen immer **Ausführbedingungen** und **keine Abbruchbedingungen** (allerdings stellt eine if-break-Konstellation eine Abbruchbedingung dar).
- In Java gibt es **kopfgesteuerte** Schleifen und **fußgesteuerte** Schleifen.

Arrays

- Array anlegen

```
// Array anlegen mit Platz für sechs int-Werte
int[] lottozahlen = new int[6];

 // Alternativ wird hier der Array gleich initialisiert:
int[] lottozahlen = {5, 18, 32, 42, 45, 48};
```

- Zugriff auf einzelne Elemente des Arrays

```
// Ändert auf Basis des initialisierten Arrays den Wert 32 auf 35
lottozahlen[2] = 35;

// Gibt den Wert 5 auf der Konsole aus
System.out.println(lottozahlen[0]);
```

- Abfragen der Länge des Arrays

```
// Gibt den Wert 6 auf der Konsole aus
System.out.println(lottozahlen.length);
```

Arrays (2)

- In Java kann (im Gegensatz zu C/C++) stets die Länge von Arrays ermittelt werden. (mittels `.length`)
- Der Zugriff auf die Elemente erfolgt per Index. Das erste Element hat den Index 0. Die weiteren Indexe sind aufsteigend durchnummeriert (bis `.length - 1`)
- Arrays sind in Java **Referenztypen** und **keine Wertetypen** wie z.B. `int` und `double`

Wertetyp und Referenztyp

Wertetyp

- Instanzen werden direkt adressiert
- Elementare Typen (`int`, `double`, `char` etc.)
- Standardwert: Je nach Typ festgelegt (z.B. `int`: 0, `boolean`: `false`)

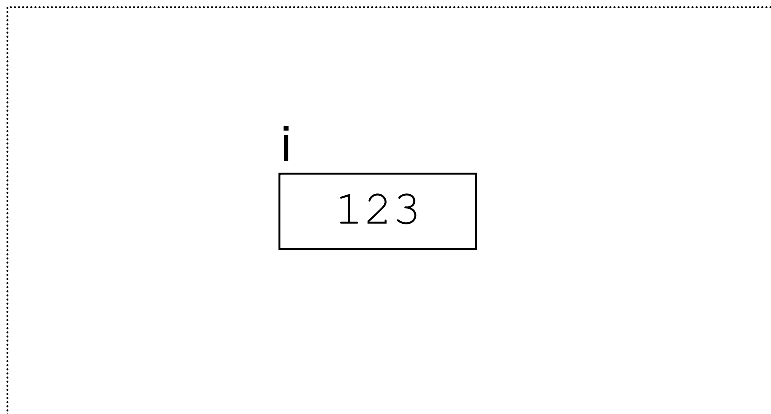
Referenztyp

- Instanzen werden indirekt über einen Zeiger bzw. eine Referenz adressiert
- Abstrakte Typen, also Klassen (siehe Kap. 3) und **Arrays**
- Standardwert: **`null`**, d.h. derzeit wird keine Instanz referenziert

Beispiel

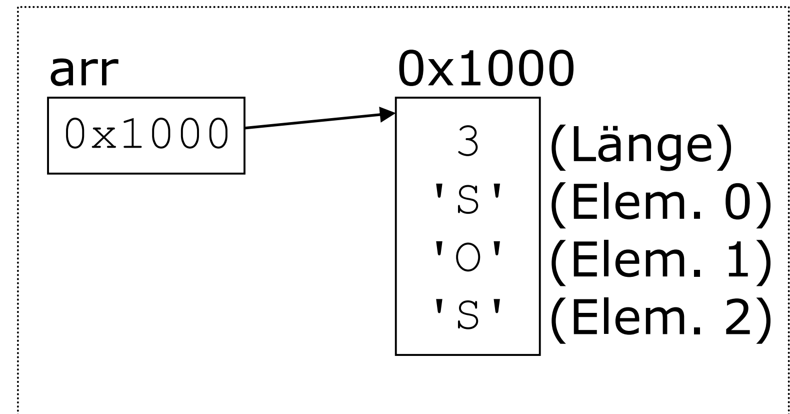
Wertetyp

```
int i;
i = 123;
```



Referenztyp

```
char[] arr;
arr = new char[3];
arr[0] = 'S';
arr[1] = 'O';
arr[2] = 'S';
```



Beispiel

```
// Array anlegen mit Platz für sechs int-Werte
int[] lottozahlen = new int[6];

// Ziehung der Zahlen
for (int i = 0; i < lottozahlen.length; i++) {
   lottozahlen[i] = (int)(Math.random()*49.0)+1;
}

// Ausgabe der Zahlen
for (int i = 0; i < lottozahlen.length; i++) {
   System.out.println(lottozahlen[i]);
}
```

- todo: Zahlen werden noch doppelt gezogen und sind in der Ausgabe nicht sortiert.

Funktionen

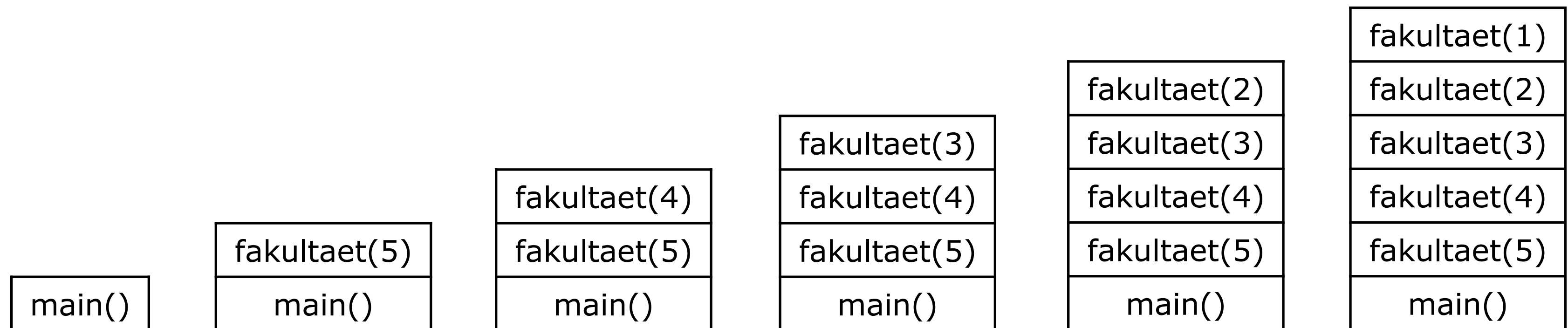
```
public class Fakultaet {

    public static int fakultaet(int x) {
        int f = 1;                              // 2
        for (int i = 2; i <= x; i++)            // 3
            f = f * i;
        return f;                               // 4
    }

    public static void main(String[] args) {
        int f = fakultaet(5);                   // 1
        System.out.println(f);                  // 5
    }
}
```

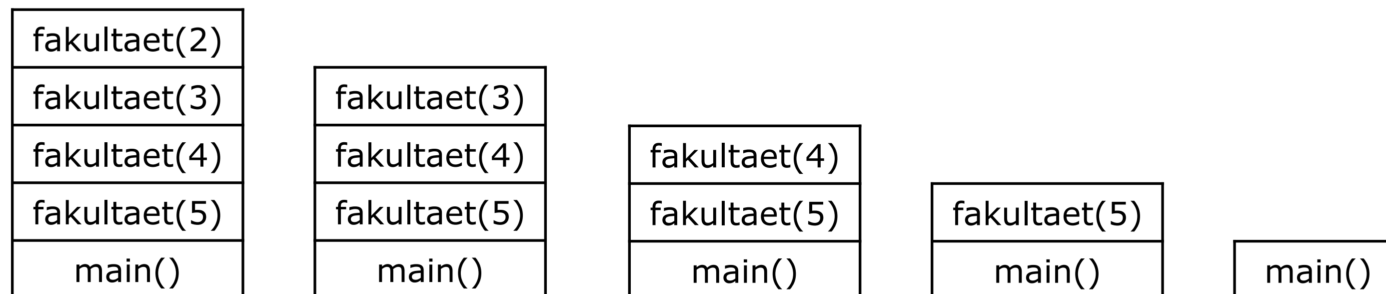
Rekursive Funktionen (Beispiel)

```
public class FakultaetRekursiv {

    public static int fakultaet(int x) {
        if (x <= 1) return 1;
        else        return x * fakultaet(x - 1);
    }

    public static void main(String[] args) {
        int f = fakultaet(5);
        System.out.println(f);
    }
}
```

Also:	5! = 5 * 4!
Und	4! = 4 * 3!
Und	3! = 3 * 2!
Und	2! = 2 * 1!
Und	1! = 1

Funktionsstack (Beispiel)





Funktionsstack

- Ein Stack...
  - ...wird auch *Stapel* oder *Keller* genannt
  - ...ist eine Datenstruktur
  - ...arbeitet nach dem **LIFO-Prinzip**: Last-In-First-Out (demgegenüber arbeitet eine Queue nach dem FIFO-Prinzip)
- Ein Funktionsstack...
  - ...ist ein Stack, dessen Elemente **Funktionsinstanzen** sind.
  - Funktionsinstanzen beinhalten Eingabeparameter, lokale Variablen, Rücksprungadresse etc. (nicht die Implementierung –> diese ist statisch!)

Rekursive Funktionen

- …sind Funktionen, die sich (direkt oder indirekt) selbst aufrufen.
- …müssen wohlüberlegte Abbruchbedingungen besitzen (sonst Stackoverflow!)
- Rekursive Aufrufe geschehen immer mit abgewandelten Eingabeparametern (sonst Stackoverflow!)
- Jede Instanz löst selbst ein gleichartiges Teilproblem.
- Das Gesamtproblem einer Instanz nimmt mit zunehmender Verschachtelung ab. (2! ist weniger komplex als 5!)

Exceptions

- Zur Behandlung von Ausnahmesituationen im Programmablauf
- Beispiel: Der Zugriff auf eine Datei klappt nicht.
- Alternative Prüfung mit if-Anweisungen bietet keine elegante Syntax.

Beispiel

```
public static void dateiVerarbeiten() {

    try {
        // Öffne Datei

        while(/* Dateiende nicht erreicht */) {

            // Lies Zeile aus Datei
        }

        // Schließe Datei
    }
    catch (Exception e) {
        System.out.println(e);
    }
}
```

Exceptions weiterreichen (`throws`)

```java
public static void dateiVerarbeiten() throws Exception {
    // Öffne Datei
    while (/* Dateiende nicht erreicht */) {
        // Lies Zeile aus Datei
    }
    // Schließe Datei
}

public static void main(String[] args) {
    try {
        dateiVerarbeiten();
    }
    catch (Exception e) {
        System.out.println(e);
    }
}
```