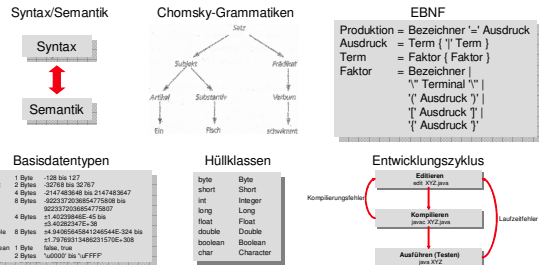


Variablen, Referenzen, Zuweisungen

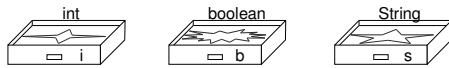
Thomas Röfer

Variablen
Zuweisung / Initialisierung
Konstanten
Lebenszeit / Sichtbarkeit
Java Stack / Heap
Call by Reference / Value

Rückblick „Grammatik, Bezeichner, Datentypen, Entwicklungszyklus“



Variablen



- Speicherstellen haben
 - eine Adresse
 - einen Inhalt
- Variablen haben
 - einen Namen (*i, b, s, ...*)
 - einen Typ (*int, boolean, String, ...*)
 - einen Wert (*7, true, "Hallo", ...*), d.h. den Inhalt der Speicherstelle
 - einen Ort, an dem ihr Wert gespeichert wird, d.h. die Referenz auf die Speicherstelle
 - eine Lebenszeit
- Objekte haben
 - keinen Namen
 - sonst alles, was auch Variablen haben
- Referenzen
 - zeigen auf Speicherstellen
- Referenzen in Java
 - sind Variablen
 - Ihr Wert zeigt auf den Ort, an dem ein Objekt gespeichert ist
 - oder auf *null*

Deklaration von Variablen

- Lokale Variablen in Funktionen
 - Existieren 1x pro Aufruf der Funktion
 - `void fun() { int n; ... }`
- Parameter von Funktionen
 - Sind auch lokale Variablen
 - Existieren 1x pro Aufruf der Funktion
 - `int factorial(int n) { ... }`
- Objektattribute
 - Existieren 1x pro Objekt (instancierter Klasse)
 - `class Name { String vorname; ... }`
- Klassenattribute
 - Existieren 1x pro Klasse (unabhängig von Instanzen)
 - `class System { static PrintStream out; ... }`

Zuweisungen

- Eine Zuweisung ändert den Wert einer Variablen
- Linkswert (L-Wert, L-Value)
 - Ein Wert, der in einer Zuweisung links vom = stehen kann
 - Beispiele:
 - `int a;`
 - `a = 17;`
 - `f(17)[14].vorname = "Hallo";` // Typ von f: `Name[] f(int i)`
- Rechtswert (R-Wert, R-Value)
 - Ein Wert, der in einer Zuweisung rechts stehen kann
 - R-Werte können auch als Funktionsparameter benutzt werden
 - L-Werte sind immer auch R-Werte
 - Beispiele:
 - `int a = 17;`
 - `a = 17 * 29 * a + factorial(5 + 7);`

Initialisierung

- Eine Initialisierung setzt den anfänglichen Wert einer Variablen
- Parameter von Funktionen
 - Initialisierung durch die übergebenen Werte
 - `void f(int n) { ... }`
 - :
 - `f(15);`
 - Parameter *n* von *f()* bekommt den Wert 15, d.h. äquiv. zu `n = 15;`
- Lokale Variablen
 - Werden nicht automatisch initialisiert,
 - aber Compiler erzwingt Initialisierung vor Benutzung als R-Wert
 - `void f() { int m; int n; m = 6; }`
- Klassen- und Objektattribute
 - Automatische Initialisierung mit 0 / *null* / *false*

Konstanten (finals)

- Im Gegensatz zu Variablen kann sich der Wert einer Konstanten nach ihrer Initialisierung niemals ändern
- Kompilierzeit-Konstanten**
 - lassen sich zur Kompilierzeit errechnen
 - sind benannte Literale, können also überall verwendet werden, wo auch Literale zulässig sind (z.B. hinter case in der switch-Anweisung)
 - werden gleich bei ihrer Deklaration initialisiert
 - Beispiel:


```
final int MAX_STUDENTS = 350;
```
- Laufzeit-Konstanten**
 - lassen sich erst zur Laufzeit errechnen
 - sind „schreibgeschützte“ Variablen, d.h. keine Literale
 - Können nach ihrer Deklaration initialisiert werden, z.B. in Konstruktoren

```
class Constant
{
    final int MAX_STUDENTS;
}
Constant(int students)
{
    MAX_STUDENTS = students;
}
```

Lebenszeit

- Von Variablen**
 - In Funktionen bis Ende des aktuellen Blocks {...}
 - Ausnahme: Funktionsparameter
 - `void f(int a) {...}` ist eigentlich `void f { (int a) {...} }`
 - Ausnahme: Definitionen in for
 - `for(int i = 0; i < 10; ++i) {...}` ist eigentlich `for(int i = 0; i < 10; ++i) {...}`
- Von Objekten und Arrays**
 - Beginnt mit dem Erzeugen durch `new`
 - Endet, wenn keine Referenz mehr auf sie zeigt
 - Unreferenzierte Objekte werden bei Bedarf vom **Garbage Collector** freigegeben
 - Wenn Speicher knapp wird
 - Wenn das Programm nichts anderes zu tun hat
 - Wenn `System.gc()` aufgerufen wird

Sichtbarkeit

- Sichtbarkeit, Suchen nach Bezeichner**
 - In Funktion rückwärts, aber nicht in Blöcke hinein
 - In Funktionsparametern
 - In Klasse (auch hinter der aktuellen Funktion)
- Mehrfachdefinitionen in Fkt. verboten**
 - `void f(int a)`

```
{
    int a;
    int a;
}
```
- Zugriff auf gerade nicht sichtbare Variablen**
 - Objektattribute: `this.variable`
 - Klassenattribute: `Klassenname.variable`

```
class HideAndSeek
{
    static int target;
    static void fun()
    {
        System.out.println(target);
    }
    {
        System.out.println(target);
        int target = 17;
        System.out.println(target);
    }
    System.out.println(target);
}
```

Arrays

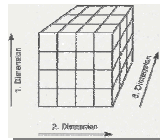
- Definition**
 - Eine Reihung fester Länge von Variablen gleichen Typs, auf die mit einem Index zugegriffen werden kann.
- Arrays**
 - sind Objekte
 - enthalten ein konstantes `int`-Attribut `length`, das ihre Länge angibt
 - Ein Zugriff außerhalb der Array-Grenzen `[0 ... length-1]` erzeugt eine `ArrayIndexOutOfBoundsException`
- Beispiel**
 - `String[] a = { "Herbert Pappelbusch", "Irene Schmidt", "Claudia Miesmuffel", "Tobias Kleinemann", "Mareike Bumtrupp" };`

0	Herbert Pappelbusch
1	Irene Schmidt
2	Claudia Miesmuffel
...	...
48	Tobias Kleinemann
49	Mareike Bumtrupp

```
int i = 0;
while(i < a.length)
{
    System.out.println(a[i]);
    i = i + 1;
}
```

Mehrdimensionale Arrays

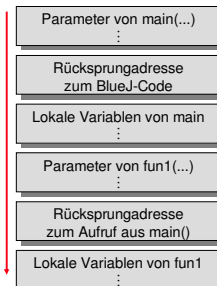
- Dimensionen**
 - Eine beliebige Anzahl von Dimensionen ist möglich
 - Speicherplatzverbrauch bedenken!
 - Ein zweidimensionales Array ist ein Array von Arrays
 - Daher können die Arrays in weiteren Dimensionen selbst auch wieder unterschiedliche Größen haben
- Beispiel**
 - `int[][] a = new int[5][];`
 - `a[0] = new int[3];`
 - `a[1] = new int[2];`
 - `a[2] = new int[4];`



Index	0	1	2	3
0	A	B	C	
1	D	E		
2	F	G	H	I

Java-Stack (Stapel, Keller)

```
class Foo
{
    static int main(int j)
    {
        int i = 1;
        int k = fun1(i,j);
        return k;
    }
    static int fun1(int a, int b)
    {
        bool x;
        return a * b;
    }
}
```

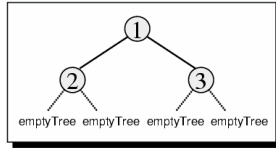


Beispiel: BinTree

```
class BinTree
{
  BinTree leftBranch;
  int val;
  BinTree rightBranch;
  boolean isEmpty;

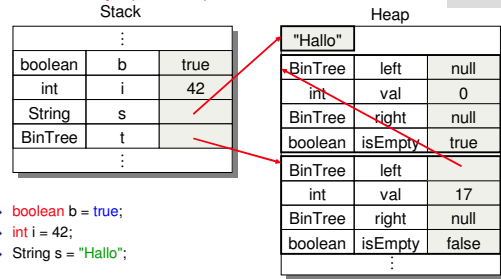
  BinTree() {isEmpty = true;}

  BinTree(BinTree l, int v, BinTree r)
  {
    leftBranch = l;
    val = v;
    rightBranch = r;
    isEmpty = false;
  }
}
```



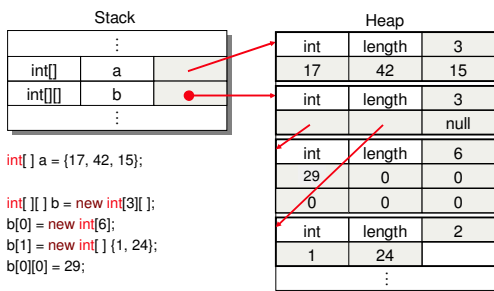
```
BinTree b = new BinTree(
  new BinTree(new BinTree(), 2, new BinTree()),
  1,
  new BinTree(new BinTree(), 3, new BinTree())
);
```

Java-Heap (Halde)



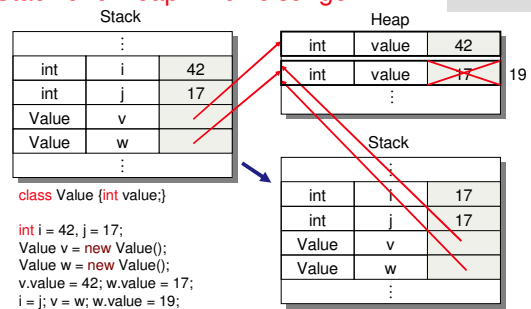
```
boolean b = true;
int i = 42;
String s = "Hallo";
BinTree t = new BinTree(new BinTree(), 17, null);
```

Java-Heap – Arrays



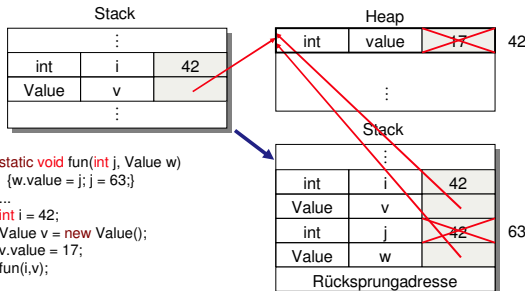
```
int[] a = {17, 42, 15};
int[][] b = new int[3][];
b[0] = new int[6];
b[1] = new int[ ] {1, 24};
b[0][0] = 29;
```

Stack und Heap – Zuweisungen



```
class Value {int value;}
int i = 42, j = 17;
Value v = new Value();
Value w = new Value();
v.value = 42; w.value = 17;
i = j; v = w; w.value = 19;
```

Stack und Heap – Funktionsaufrufe



```
static void fun(int j, Value w)
{w.value = j; j = 63;}
...
int i = 42;
Value v = new Value();
v.value = 17;
fun(i,v);
```

Call by Reference / Call by Value

- „Call by Value“ in Pascal
 - PROCEDURE swap(a, b : INTEGER)
 - VAR temp : INTEGER;
 - BEGIN
 - temp := a; a := b; b := temp
 - END
 - Ergebnis: Lokale a und b werden vertauscht. *Keine Wirkung auf Aufrufer!*
- „Call by Reference“ in Pascal
 - PROCEDURE swap(VAR a, b : INTEGER)
 - VAR temp : INTEGER;
 - BEGIN
 - temp := a; a := b; b := temp
 - END
 - Ergebnis: Übergebene a und b werden vertauscht. *Wirkung auf Aufrufer!*
- In Java
 - Parameterübergabe identisch mit Zuweisung, daher nur „Call by Value“
 - Aber: Eine Funktion kann Daten ändern, auf die ein übergebener Parameter zeigt!

Beispiel: Zahlen austauschen

```

class Value
{
    int value;
}

class Swap
{
    static void swap1(int a, int b)
    {
        int t = a;
        a = b;
        b = t;
    }

    static void swap2(Value a, Value b)
    {
        Value t = a;
        a = b;
        b = t;
    }

    static void swap3(Value a, Value b)
    {
        int t = a.value;
        a.value = b.value;
        b.value = t;
    }
}

Aufruf
int c = 17;
int d = 42;
swap1(c, d);

Value e = new Value();
Value f = new Value();
e.value = 17;
f.value = 42;
swap2(e, f);
swap3(e, f);
    
```

Beispiel: Zahlen austauschen

