Informatik I – Kapitel 6

„ Elementare Konzepte von Programmiersprachen“

Zusammenfassung des Kapitel 6

Küchlin, Weber, Einführung in die Informatik, 2.Auflage

16.1.2004

- Ausdrücke
  - haben einen Wert
    - sin(math.PI) wird zu 1
    - sin(kreisRadius) wird zur Laufzeit ausgewertet
- Anweisungen
  - Zuweisungen
    - kreisRadius = 2.67;
  - Anweisungen zur Ablaufsteuerung
    - elementare Verzweigungen
      - » if-then-else, switch, goto
    - höhere Schleifenkonstrukte
      - » while, do-while, for

- Datentypen
  - elementare (primitive types)
    - schon eingebaut (char, short , long , int, float, double)
  - strukturierte (structured types)
    - array, String, eigene Klassen
- Deklaration von
  - Konstanten
    - public final static int MYPI=3.1415927;
  - Variablen
    - int kreisRadius;

- Unterprogramme
  - Prozeduren
    - kein Rückgabewert
    - public static void meineProzedur(..){ ... }
  - Funktionen
    - ein Rückgabewert
    - public static int meineFunktion(..){ ... }
- dienen der *Kapselung*, also der Strukturierung
- in Java werden sie als *Methoden* einer Klasse realisiert

Begriffe

marc-oliver pahl

– Syntax
  - Schlüsselwörter und die zugehörige Grammatik
  - meist in BNF (Backus-Naur-Form)
    – if_statement ::=
    "if" "(" expression **")"** statement [**"else"** statement]

    BNF Index of JAVA language grammar, http://cui.unige.ch/java/JAVAF/

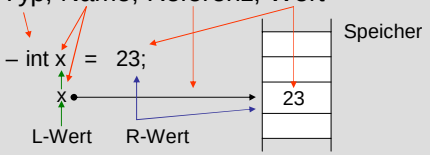
– Semantik
  - Bedeutung/ Sinn des Codes

Variablen, Referenzen, Zuweisungen

marc-oliver pahl

- Variable
  – Deklaration:
    - int x;
  – Zuweisung:
    - x=23;

  – charakterisiert durch:
    - <Typ, Name, Referenz, Wert>



Begriffe

marc-oliver pahl

– Schlüsselwörter in Java
  - abstract, finally, public, assert, float, return, boolean, for, short, break, goto, static, byte, if, strictfp, case, implements, super, catch, import, switch, char, instanceof, synchronized, class, int, this, const, interface, throw, continue, long, throws, default, native, transient, do, new, try, double, package, void, else, private, volatile, extends, protected, while, final

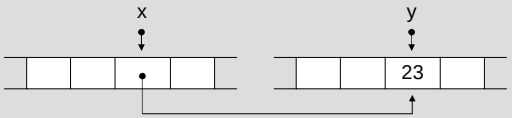
– Literale
  - das Literal 12 hat z.B. die Bedeutung/ den Wert 12
  - "inhalt" steht für einen String mit dem Wert inhalt

– Namen
  - bezeichnen Variablen, Klassen, Methoden
  - Namenskonvention beachten!
    (ichBinMethode, ichVariable, ICHKONSTANTE)

Referenzvariable, Reihungsvariable

marc-oliver pahl

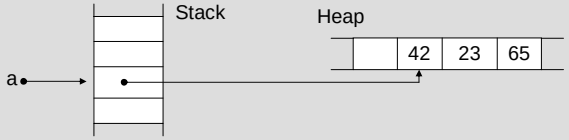
- Referenzvariable (Explizit in Java nicht möglich)



„x ist Referenz darauf, wohin y zeigt"
(C++: int* x = &y;)

- Reihungsvariable

int[ ] a = new int[ ]{42, 23, 65};



- Java-Arithmetik nach Typ der Operanden
  - 3/ 2 = 1, weil Ganzzahlarithmetik (1 [Rest 1])
- Überlauf schneidet zusätzliche Bits ab
  - $(2^{31}-1) + 1 = -2^{31}$ (bei 32bit-Integer-Arithmetik)

    Grund: Zweierkomplementdarstellung:

    01 11111 11111 11111 11111 11111 11111 [2^{31}-1]

    +00 00000 00000 00000 00000 00000 00001 [1]

    =10 00000 00000 00000 00000 00000 00000 [-2^{31}]
  - dadurch „Ring“ und kein „out of range“

- explizite Typkonversion
  - (int) 13.6 ergibt ein Integer mit dem Wert 13
    - Java schneidet hier den Nachkommateil ab (= Runden zur 0)
- implizite Typkonversion
  - Typverengung
    - hin zu einem Typ mit ungenauerem Wertebereich (double -> int)
  - Typaufweitung
    - hin zu einem Typ mit genauerem Wertebereich (int -> double)
    - z.B. <double> + <int> -> <double>
      - » 12.6 + 10 ergibt ein Double mit dem Wert 22.6

- Gleitkomma-Arithmetik (+Infinity, ...)
- Zuweisungsoperatoren (Wert des Ausdrucks)
- Arithmetische Operatoren
- Boolesche Operatoren
- Rechnen auf Bitmustern (&, |)
- Ausdrücke (Präfix, Postfix, Infix, Roundfix, Mixfix) (Präzedenz von Operatoren = Bindungskraft)

- Anweisungen
- Blöcke, Gültigkeitsbereich, Lebensdauer
  - Adressrechnung (dynamisch/ statisch)
- Bedingte Anweisungen
- Schleifenkonstrukte
  - Marken break und continue

```
class Unterprogramme{
    public static int plus(int x, int y){ return x+y; }

    public static void main(String[] args){
        int a = 7;
        int b = 8;
        System.out.println( plus(a, b) );
    }
}
```

Kopf | Rumpf

Ergebnistyp | Name | formale Parameter

Signatur

aktuelle Parameter

- Parameterübergabe
  - call by value (Werteaufruf, JAVA)
  - call by reference (Referenzaufruf)
  - call by name (Namensaufruf)

- **Überladung** von Methoden z.B. für generisches Programmieren mithilfe unterschiedlicher Signaturen

```
class MyMath{
    ...

    public double plus(double x, double y){ ... }

    public float plus(float x, float y){ ... }

    public int plus(int x, int y){ ... }

    ...
}
```

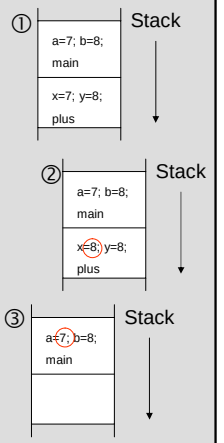

- **call by value** (Der einzige Übergabemechanismus in Java)

```
class CallByValue{
        static int a = 7;
        static int b = 8;

        public static void main(String[ ] args){
                System.out.println("a+b: "+plus(a, b));
                System.out.println("a: "+a);
                System.out.println("b: "+b);
        }

        public static int plus(int x, int y){
                int res = x+y;
                x++;
                return res;
        }
}
```

① ② ③



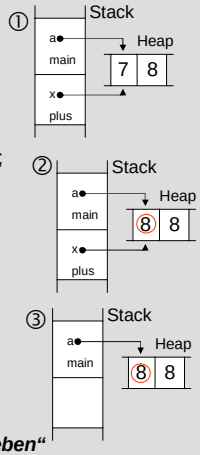
„es wird eine Kopie der Werte angelegt"

- **Call by value mit zusammengesetzten Typen**
  (geschieht in Java bei Variablen vom Typ einer Klasse inkl. Arrays)

```
class PassReferenceByValue{
        static int[ ] a = new int[ ]{7, 8};

        public static void main(String[ ] args){
                System.out.println("a[0]+a[1]: "+plus(a));
                System.out.println("a[0]: "+a[0]);   ← ③
                System.out.println("a[1]: "+a[1]);
        }

        public static int plus(int[ ] x){   ← ①
                int res = x[0]+x[1];
                x[0]++;   ← ②
                return res;
        }
}
```

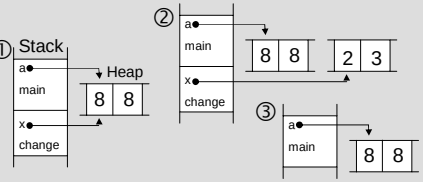


„es wird eine Kopie eines Zeigers auf die Werte übergeben“


```
static void Main(string[] args){
        a = new int[ ]{7, 8};
        change(a);   ← ③        bzw.        change(ref a);   ← ③
}
```

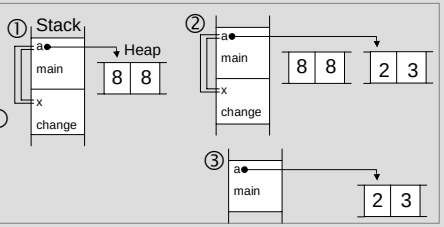
Pass reference **by value:**

```
static void change(int[ ] x){
        x[0]++;   ← ①
        x = new int[ ]{2, 3};   ← ②
}
```



Call by reference:

```
static void change(ref int[ ] x){
        x[0]++;   ← ①
        x = new int[ ]{2, 3};   ← ②
}
```

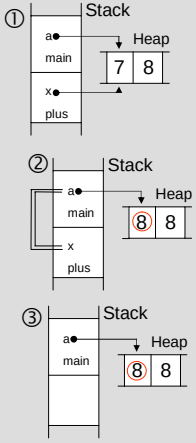


- **call by reference** (In Java nicht möglich, aber z.B. in C# (ref), C++ (&), Pascal (var))

```
public class CallByReference{
  static int[ ] a = new int[ ]{7, 8};          C# !

   public static void Main(String[ ] args)        {
        Console.WriteLine("a[0]+a[1]: "+plus(ref a));
        Console.WriteLine("a[0]: "+a[0]);   ← ③
        Console.WriteLine("a[1]: "+a[1]);
   }

   public static int plus(ref int[ ] x){   ← ①
        int res = x[0] + x[1];
        x[0]++;
        return res;   ← ②
   }
}
```



„es wird eine Referenz, also ein Linkswert, übergeben,
der formale Parameter wird zum Alias für den aktuellen Parameter“

- **call by name** (gibt es in Java nicht, aber z.B. in VisualBasic)
- Da der Mechanismus der Namensübergabe in Java nicht direkt vorgesehen ist, gehen wir im folgenden Beispiel davon aus, dass wir eine Klasse „Assoziativspeicher“ haben, die die Methoden „int getValueByName(String name)“ und „setValueByName(String name, int value)“ enthält.

```
class CallByNameSimulation{
    static Assoziativspeicher A = new Assoziativspeicher();

        public static void main(String[ ] args){
                A. setValueByName( "a", 7);
                A. setValueByName( "b", 8);
                System.out.println("a+b: "+plus("a", "b"));
                System.out.println("a: "+A. getValueByName( "a"));
                System.out.println("b: "+A. getValueByName( "b"));
        }

        public static int plus(String n1, String n2){
                int res = A. getValueByName(n1)+A. getValueByName(n2);
                A. setValueByName( n1, A. getValueByName(n1)+1)
                return res;
        }
}
```

Assoziativspeicher A

Name	a	b
Wert	⑦	8

int getValueByName(**String** Name);
setValuebyName(**String** Name, **int** value);

Name	a	b
Wert	⑧	8

„es wird ein Name übergeben“

Rekursion

marc-oliver pahl

- **Rekursion** haben wir ausführlich auf früheren Folien im Tutorium behandelt!
  - Siehe sum_in_java! (vor allem die Kontexte!)

```
public static int sum_rek(int n){
  if ( n == 0 ) return 0;
  return ( n + sum_rek( n-1 ) );
}
```

- Endrekursion

```
public static int sum_endRek(int n){
  return sum_endRekHelp( n, 0 );
}

public static int sum_endRekHelp(int n, int res){
  if ( n == 0 ) return res;
  return sum_endRekHelp( n-1, res+n ) );
}
```

„es wird nur leerer Kontext erzeugt“

endrekursiv -> iterativ

marc-oliver pahl

endrekursiv

```
public static int sum_rek(int n){
  return sum_endRek( n, 0 );
}

public static
int sum_endRekHelp(int n, int res){
  if ( n == 0 ) return res;
  return sum_endRekHelp( n-1, res+n ) );
}
```

iterativ

```
public static int sum_iter(int n){
  int k = n;
  int res = 0;
  while ( k >= 0) {
    res = k + res;
    k = k - 1;
  }
  return res;
}
```