# Einführung in die Programmierung WS 12-13

Lernskript

Dozent:
Dr. Hildebrandt

LaTeX von:
Sven Bamberger

Zuletzt Aktualisiert:
14. Februar 2013

Dieses Skript wurde erstellt, um sich besser auf die Klausur vorzubereiten.

Dieses Dokument garantiert weder Richtigkeit noch Vollständigkeit, da es aus Mitschriften gefertigt wurde und dabei immer Fehler entstehen können. Falls ein Fehler enthalten ist, bitte melden oder selbst korrigieren und neu hoch laden.

# Inhaltsverzeichnis

# 1 Vorlesungen

## 1.1 Java-Editionen

- Java läuft auf sehr unterschiedlichen Systemen
- Wird in verschiedenen „Packungsgrößen“ angeboten
  - EE (enterprise edition): große Unternehmensserver
  - SE (standard edition): Desktop-Systeme
  - ME (micro edition): Handys, PDAs, Embedded Systems
- Editionen unterscheiden sich in den mitgelieferten Zugaben,nicht in der Programmiersprache

## 1.2 Java-Entwicklungssystem

- Zum Schreiben neuer Programm ist der Compiler javac nötig, zum Ausführen fertiger Programm nicht
- Java für verschiedene Einsatzzwecke:
  - JRE (java runtime environment): zum Ausführen fertiger Programme
  - JDK (java development kit): JRE + Compiler und weitere Hilfsmittel zum Schreiben neuer Programme

## 1.3 Namen in Java

- An vielen Stellen frei wählbare Namen = „Bezeichner“, „Identifier“
- Bestandteile: Große und kleine Buchstaben, Ziffern, Underscore (_)
- Erstes Zeichen darf keine Ziffer sein
- Etwa fünfzig reservierte Wörter dürfen nicht als Identifier benutzt werden (beispielsweise class, int, public)
- Beispiele:
  - Counter
  - colorDepth
  - Iso9660
  - XMLProcessor
  - MAX_VALUE
- Nicht erlaubt sind z.B.:
  - 1stTry (erster Buchstabe darf keine Ziffer sein)

  - Herz Dame (Leerzeichen im Namen nicht erlaubt)
  - const (reserviertes Wort)
  - muenchen-erding (Bindestrich im Namen nicht erlaubt)
- Übliche Konventionen für Java-Identifier:
  - Variablen, Methoden, primitive Typen: CamelCode, erster Buchstabe klein: counter, find1stToken, bottomUp
  - Referenztypen: CamelCode, erster Buchstabe groß: Hello, String, ServerSocket
  - Typvariablen (Generics): einzelne große Buchstaben: T, U
  - statische, öffentliche Konstanten: alle Buchstaben groß, Wortteile getrennt mit Underscore: MAX_VALUE, PI, RGB24

## 1.4 Regel Ebenen

### 1.4.1 Syntax (Rechtschreibung)

Verteilung von Semikolons, Klammern, Schreibweise von Namen

#### Syntaxfehler

Compiler meldet einen Fehler

### 1.4.2 Semantik (Bedeutung)

Zulässige Kombination von Sprachelementen

#### Semantikfehler

Compiler meldet einen Fehler, Programm verhält sich falsch, stürzt nach dem Start ab, ...

### 1.4.3 Pragmatik (Gebrauch)

Bewährte und sinnvolle Konstruktionen

#### Fehler der Pragmatik

Programm ist unleserlich, umständlich, unverständlich

## 1.5 Polymorphismus

- Der Typ des Ergebnisses, und u.U. der Wert, ist abhängig von den Typen der Operanden:
  $20/8 \rightarrow 2$
  $20.0/8.0 \rightarrow 2.5$
- Zwei Operanden gleichen Typs: Operandentyp = Ergebnistyp
- Gemischte Operandentypen: double-Ergebnis:
  $1 + 2 \rightarrow 3$ (int)
  $1.0 + 2 \rightarrow 3.0$ (double)

$1 + 2.0 \rightarrow 3.0$ (double)

$1.0 + 2.0 \rightarrow 3.0$ (double)

## 1.6 Explizite Typkonversionen

- Hohe Priorität, wie andere unäre Operatoren:
  $(\text{int})2.5 * 3 \rightarrow 2 * 3 \rightarrow 6$
  -$(\text{int})2.5 \rightarrow -2$
  Klammern hilft: $(\text{int})(2.5 * 3) \rightarrow (\text{int})(7.5) \rightarrow 7$
- ACHTUNG STOLPERFALLE
  $(\text{int})1e100 \rightarrow 2147483647$
- Typcasts auf ein Minimum beschränken.

## 1.7 Struktogramme

- Elementarbausteine von Struktogrammen: einfache Anweisungen
- Formulierung einzelner Anweisungen
- Beschreibungs- oder Darstellungsformen für Algorithmen:
  - Umgangssprache
    Problematisch: Mißverständnisse, Interpretationsmöglichkeiten, Sprachkenntnisse
  - Quelltext
    Nur mit Kennnis einer konkreten Programmiersprache lesbar
  - Neutrale, abstrakte Form
    Brauchbarer Kompromiss
- Populär: Struktogramme (=Nassi-Schneiderman-Diagramme)
- Früher auch: Flussdiagramme (flow charts), erlauben wirre Konstruktionen
- Ziel: Reduktion auf die Idee, die wesentlichen Strukturen

### 1.7.1 Umgangssprachlich

Definiere n als ganze Zahl
Gib n den Wert 4
Zähle n um 1 hoch
Gib n aus

### 1.7.2 Pseudocode

```
int n
n = 4
n = n + 1
print n
```

### 1.7.3 Nassi-Schneiderman

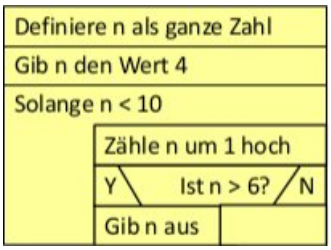


Abbildung 1.1: ein mini Struktogramm

## 1.8 While-Schleife: Euklidischer Algorithmus

```
1  class EuclidGCD
2  {
3      public static void main(String... args)
4      {
5          int m = Integer.parseInt(args[0]);
6          int n = Integer.parseInt(args[1]);
7          int r = m % n;
8          while (r != 0)
9          {
10             m = n;
11             n = r;
12             r = m % n;
13         }
14         System.out.println(n);
15     }
16 }
```

## 1.9 While-Schleife: Collatzfolge (3n + 1 – Folge)

$$z_{n+1} = \begin{cases} \frac{1}{2} z_n, & z_n \text{ gerade} \\ 3 \cdot z_n + 1, & z_n \text{ ungerade.} \end{cases}$$

```
class CollatzMax
{
    public static void main(String... args)
    {
        int z = Integer.parseInt(args[0]);
        int n = 0;
        int max = z;
        while (z != 1)
        {
            if (z%2 == 0){
                z = z/2;
            }
            else{
                z = 3*z + 1;
            }
            n++;
            if (z > max){
                max = z;
            }
        }
        System.out.println(n);
        System.out.println(max);
    }
}
```

## 1.10 Inkrement-/Dekrementoperator

Variable ++; Variable - -;

```
 int a = 1;
int b = a++; // b = 1, a = 2
int c = a--; // c = 2, a = 1
```

++Variable; - -Variable;

```
int a = 1;
int b = ++a; // b = 2, a = 2
int c = --a; // c = 1, a = 1
```

## 1.11 Der bedingte Operator

- Dreistelliger "bedingter Operator"(engl. "conditional operator")
- Syntax:
  condition? yes-expression: no-expression

- Beziehung zu if:
  variable = condition? yes-expression: no-expression;

- äquivalent zu:
  if (condition)
  . variable = yes-expression;
  else
  . variable = no-expression;

## 1.12 do-while

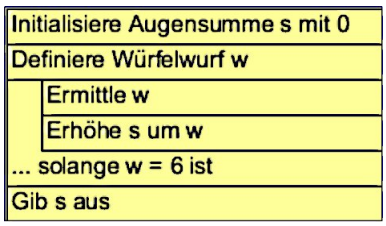


Abbildung 1.2: ein mini Struktogramm

```
int s = 0;
int w;
do{
    w = // wuerfeln ...
    s += w;
} while (w == 6);
System.out.println(s);
```

## 1.13 Break und Continue

### 1.13.1 Break

- Anweisung break beendet eine Schleife sofort der Rest des Rumpfes wird übersprungen
- break = einfache Anweisung (wie Definitionen, Wertzuweisungen)
- Zweck: Entscheidung über Fortsetzung einer Schleife fällt mitten im Rumpf

### 1.13.2 continue

- Anweisung continue startet sofort den nächsten Schleifendurchlauf der Rest des Rumpfes wird übersprungen
- Wie break: Nützlich zur Behandlung von Sonderfällen
- Zweck: Folge von Entscheidungen über Fortsetzung des Schleifendurchlaufes mitten im Rumpf

Achtung: break und continue spalten Kontrollfluss: mit Bedacht verwenden

## 1.14 Gültigkeitsbereiche

- Idee
  - Blöcke ... gruppieren Anweisungen
  - Innnerhalb eines Blocks alle Anweisungsarten erlaubt, auch Definitionen
  - Gültigkeitsbereich (engl. „scope“) einer Variablen...
    - beginnt mit der Definition und
    - endet mit dem Block, in dem die Definition steht
  - Außerhalb des Blocks: Variable gilt nicht
  - Gültigkeitsbereiche bezogen auf Quelltext, werden vom Compiler überprüft
  - Zur Laufzeit irrelevant
- Namenskollision
  - Gültigkeitsbereich umfasst untergeordnete (geschachtelte) Blöcke
  - Namenskollision: Definition des gleichen Namens, wie in einem umfassenden Block
  - Java: Doppelte Definition unzulässig
  - Aber: Kein Problem in disjunkten Blöcken:

## 1.15 for-Schleife

Jede For-Schleife kann durch eine While Schleife ersetzt werden.

```
1 for(int i = 0; i < 10; i++)
2     System.out.println(i);
```

## 1.16 Switch

- Der Wert der expression wird einmal berechnet.
- Das Ergebnis wird nacheinander mit den labels verglichen, bis zum ersten gleichen Wert.
- Die dem label nachfolgenden statements werden ausgeführt, bis zum break;
- Ziel: switch-Anweisungen ersetzen längere, unübersichtliche if-Kaskaden
- Syntax:

```
1  switch (expression)
2  {
3      case label1:
4          statement ...
5          break;
6      case label2:
7          statement ...
8          break;
9      ...
10 }
```

# Abbildungsverzeichnis