Von der Programmiersprache WHILE zum ersten Einstieg in Java

Die Programmiersprache WHILE bildet den Kern jeder imperativen Programmiersprache. Man geht davon aus, dass die Ein- Ausgabe primitiv organisiert wird, d.h. gewisse Variablennamen stehen für spezielle Speicherplätze (Register), die von außen geschrieben und/oder gelesen werden können.

Definition (Die Programmiersprache WHILE): Gegeben sei eine abzählbare, unendliche Menge von Variablen, z.B. die Menge der kleinen lateinischen Buchstaben mit oder ohne Indizes.
Syntax: In der Sprache WHILE gibt es nur folgende vier Anweisungsarten (Befehle):

Wenn `x` und `y` in der Menge der Variablen ist, dann sind

```
x := 0,
x := x + 1,
x := x - 1    und
whlile x ≠ y do <Anweisungsfolge> end
```

syntaktisch korrekte Anweisungen der Sprache WHILE, wobei `<Anweisungsfolge>` induktiv eine endliche Folge syntaktisch korrekter Anweisungen der Sprache WHILE ist.

Ein ***WHILE-Programm*** ist eine endliche Folge syntaktisch korrekter Anweisungen der Sprache WHILE.

Semantik: Die (informelle) Bedeutung dieser Anweisungen ist durch folgende Vereinbarungen gegeben: Die Ausführung einer Anweisung bewirkt eine Änderung der Variablenwerte (Inhalte der zugehörigen Speicherplätze) wie folgt:

Anweisung	Name	Semantk
`x := 0`	Lösche x	`x` erhält den Wert 0, d.h. der Inhalt des Speicherplatzes mit dem Namen `x` wird auf 0 gesetzt.
`x := x + 1`	Nachfolger	Der Wert von `x` wird um 1 erhöht.
`x := x - 1`	Vorgänger	Falls der Wert von `x` größer Null ist, wird er um 1 erniedrigt.
`whlile` `x ≠ y` **`do`** `<Anweisungsfolge>` **`end`**	While-Scheife	Wenn der Wert von `x` gleich dem Wert von `y` ist, dann hat die Ausführung dieser Anweisung keine Auswirkung auf die Variablenwerte. Wenn die Werte ungleich sind ist die Auswirkung induktiv durch die Hintereinanderausführung der `<Anweisungsfolge>`, gefolgt von der nochmaligen Ausführung der gesamten Anweisung erklärt.

Definition (WHILE-berechenbare Funktionen): Eine n-stellige Funktion f über den natürlichen Zahlen heißt WHILE-berechenbar, genau dann wenn es ein WHILE Programm P_f gibt, sodass für alle w_1, ..., w_n gilt:

f (w_1,...,w_n) = w gdw die Ausführung von Pf mit Anfangswerten w_1,...,w_n der Variablen a_1,...,a_n terminiert und die Variable a nach Ausführung von P_f den Wert w besitzt.

Satz 1: Die Klasse der WHILE-berechenbaren Funktionen ist genau die Klasse der berechenbaren Funktionen.
Beweis: s. Vorlesung.

Die Registermaschine ist eine einfache, abstrakte Maschine zur Ausführung von WHILE-Programmen. Die Registermaschine kann als Kern jedes Rechners betrachtet werden.

Definition (Registermaschine): Die Registermaschine besteht aus drei Komponenten:

1. Ein Datenspeicher, bestehend aus endlich vielen Registern r_0, r_1, …, r_s, in die jeweils eine natürlich Zahl gespeichert werden kann.
2. Ein Programmspeicher, bestehend aus endlich vielen Befehlsregistern b_0,...,b_r, in die jeweils ein Befehl (s.u.) gespeichert werden kann.
3. Ein Befehlszähler pc (program counter): ein Register, das eine natürliche Zahl enthält.

Die **Menge der Befehle** und ihre Effekte bei der Ausführung wird durch folgende Tabelle erklärt:

Befehl (Syntax)	Semantik
`zero i`	Setze r_i auf 0, d.h schreibe den Wert 0 in das Register r_i
`succ i`	Erhöhe den Inhalt von r_i um 1
`pred i`	Wenn der Inhalt von r_i größer Null ist, dann erniedrige den Inhalt von r_i um 1
`je i, j, n`	Wenn der Inhalt von r_i gleich dem Inhalt von r_j ist, dann setze pc auf den Wert n
`goto n`	Setze pc auf den Wert n

Ein **Programm für die Registermaschine** besteht aus einer endlichen Folge von Befehlen.

Die **Ausführung eines Programms** beginnt mit

- einer Anfangsregisterbelegung von r_1, …, r_n, d.h. beliebige Eingabewerte w_1, ..., w_n werden in den Registern r_1, …, r_n angenommen,
- dem Wert 0 im pc, d.h. der Befehlszähler ist auf 0 gesetzt,
- einem Programmspeicher, der die Befehle des Programms der Reihe nach im Programmspeicher enthält

und führt danach folgenden Zyklus aus:

Solange der Befehlszähler einen Wert k mit $0 \leq k \leq r$ enthält, erhöhe pc um 1 und führe den Befehl aus dem Befehlsregister b_k aus.

Definition (RM-berechenbare Funktionen): Eine n-stellige Funktion f über den natürlichen Zahlen heißt Registermaschinen-berechenbar (RM-berechenbar), genau dann wenn es ein Programm RP_f für die Registermaschine gibt, so dass für alle w_1,...,w_n gilt:

$f(w_1,...,w_n) = w$ gdw die Ausführung von RP_f mit Anfangswerten w_1,...,w_n der Register r_1, …, r_n terminiert und das Register r_0 nach Ausführung von RP_f den Wert w besitzt.

Satz 2: Die Klasse der RM-berechenbaren Funktionen ist genau die Klasse der berechenbaren Funktionen.

Beweis: Sei f eine n-stellige, berechenbare Funktion über den natürlichen Zahlen. Dann gibt es nach Satz 1 ein WHILE-Programm P_f, das f berechnet, d.h. für alle natürlichen Zahlen $w_1,...,w_n$ gilt $f(w_1,...,w_n) = w$ gdw die Ausführung von P_f bzgl der Variablenbelegung: Wert von a_1 gleich w_1, ... Wert von a_n gleich w_n , mit dem Wert von a gleich w endet. Generiere ein äquivalentes Registermaschinenprogramm RP_f, als **Übersetzung** von P_f wie folgt:

1. Organisiere die Zuordnung **st** von Variablen zu Registern im Datenspeicher in einer **Symboltabelle** so, dass wenn in P_f die Variablen $v_1,...,v_t$ vorkommen, der Variablen vi das Register r_{n+i} zugeordnet wird. Wenn z.B. die Anzahl n der Eingabewerte gleich 2 ist und in P_f die Variablen x,y und z vorkommen, dann ergibt sich folgende Symboltabelle:

Symboltabelle (st)	
Variable in WHILE	Adresse im Datenspeicher
a	0
a_1	1
a_2	2
x	3
y	4
z	5

2. Jeder atomare Anweisung b der Sprache WHILE kann jetzt mittels folgendem Schema in einen äquivalenten Befehl Ü(b) übersetzt werden:

WHILE-Anweisung	Übersetzung in äquivalenten Registerbefehl
`x := 0`	`zero` st(x)
`x := x + 1`	`succ` st(x)
`x := x – 1`	`pred` st(x)
`whlile` `x ≠ y` **`do`** `<Anweisungsfolge>` **`end`**	`je` st(x), st(y), *k* Ü (`<Anweisungsfolge>`) `goto` *m*

wobei *k* und *m* erst bei der Zusammensetzung des Registermaschinenprogramms festgelegt werden kann. Dabei verweist *k* auf die Adresse im Programmspeicher direkt hinter `goto` *m* und *m* verweist auf die Adresse des Befehls `je` st(x), st(y), *k*.

Beispiel: Berechne zu Eingaben a und b den Wert des Ausdrucks a + b – 3.

Ein WHILE-Programm P zur Berechnung der Funktion f(a,b) = a + b – 3

Annahme: Die Eingabe erfolgt durch entsprechendes Setzen der Werte der Variablen a und b.

1. Teilaufgabe: Berechne c := a

```
c := 0
while c ≠ a do
   c := c+1
end
```

2. Teilaufgabe: Berechne c := c + b unter Verwendung des Hilfsregisters h

```
h := 0
while h ≠ b do
   c := c + 1
   h = h + 1
end
```

3. Teilaufgabe: Berechne c := c – 3

```
c := c - 1
c := c - 1
c := c - 1
```

Am Ende der Programmausführung enthält die Variable c das Ergebnis der Berechnung von a + b – 3.

Zur Übersetzung von P wird zunächst folgende Symboltabelle erzeugt:

Symboltabelle (st)	
Variable in WHILE	Adresse im Datenspeicher
c	0
a	1
b	2
h	3

Jetzt kann die Übersetzung von P systematisch generiert werden:

WHILE-Anweisungen	**Registerbefehle**		**Java-Anweisungen**
`c := 0`	0	`zero 0`	`c = 0;`
`whlile` `c ≠ a` **`do`**	1	`je 0, 1, 4`	**`whlile`** `(c != a)`
`c := c + 1`	2	`succ 0`	`c = c + 1;`
`end`	3	`goto 1`	
`h := 0`	4	`zero 3`	`h = 0;`
`whlile` `h ≠ b` **`do`**	5	`je 3, 2, 9`	**`whlile`** `(h != b)`
`c := c + 1`	6	`succ 0`	`{ c = c + 1;`
`h := h + 1`	7	`succ 3`	`h = h + 1;`
`end`	8	`goto 5`	`}`
`c := c - 1`	9	`pred 0`	`c = c - 1;`
`c := c - 1`	10	`pred 0`	`c = c - 1;`
`c := c - 1`	11	`pred 0`	`c = c - 1;`

Während die obige Folge von WHILE-Anweisungen ein gültiges WHILE-Programm darstellt und die Folge der obigen Registermaschinenbefehle ebenso ein Registermaschinenprogramm darstellt, gilt dies für die Java-Anweisungen nicht !

Ein **Java-Programm** besteht aus einer Sammlung von Klassen, wobei eine **Java-Klasse** als Einheit von Variablen (genannt Attribute) und Funktionen (genannt Methoden) zu verstehen ist. In der ersten Zeile erscheint das Schlüsselwort `class` gefolgt von dem Namen der Klasse (beginnend mit einem Großbuchstaben). Anschließend werden in geschweiften Klammern die Attribute und die Methoden dieser Klasse definiert.

Dabei muss mindestens eine Klasse eine spezielle Methode **`main`** besitzen, die als Einstiegspunkt (Hauptprogramm) verstanden werden kann. Wenn ein Programm ausgeführt

werden soll, ist der Name der Klasse mit der main-Methode anzugeben. Das führt dann zur Ausführung dieser main-Methode.

Eine **Methode** besteht syntaktisch aus einem Kopf und einem Rumpf.

Im **Kopf einer Methode** wird die
- **Sichtbarkeit** der Methode festgelegt, z.B. bedeutet `public`, dass diese Methode auch außerhalb der Klasse sichtbar ist;
- optional das Schlüsselwort `static` angegeben (Kennzeichnung als Klassenmethode);
- der **Ergebnistyp** vereinbart, z.B. `int` für ganze Zahl oder `void` für kein Ergebnis;
- eine **Parameterliste** angegeben, z.B. (int i, int j).

Der **Rumpf einer Methode** besteht aus einer Anweisung, wobei eine Folge von Anweisungen, in Mengenklammern eingeschlossen, eine **zusammengesetzte Anweisung** (genannt Block) ist. Alle verwendeten Variablen müssen unter Angabe ihres Typs vereinbart (deklariert) werden.

Kommentare können in Java u.a. durch // markiert werden, d.h. alle Zeichen hinter // bis zum Zeilenende werden als Kommentar interpretiert.

Die **Eingabe** kann hinter dem Aufruf durch Angabe der Argumente in Standarddarstellung erfolgen. In der main-Methode greift man durch den Ausdruck **Integer.parseInt (args[i])** auf das i-te Argument zu (Zählung beginnt bei 0).

Die **Ausgabe** kann über eine Anweisung der Form System.out.println(*string*); erfolgen, wobei *string* ein Parameter vom Typ Srtring ist.

In unserem Beispiel vereinbaren wir nur eine Klasse, die nur die Methode main besitzt:

Ein vollständiges Java-Programm zur Berechnung des Ausdrucks a + b - 3

```
class Ausdruck {  //Aufruf:  Ausdruck a b, Ausgabe: a+b-3 = <Ergebnis>

   //Hauptprogramm
   public static void main (String[] args) {

      // Deklaration aller benötigter Variablen
      int a, b, c, h;

      // Eingabe von a und b
      a=Integer.parseInt (args[0]);
      b=Integer.parseInt (args[1]);

      // Berechne c := a
      c = 0;
      while (c != a)
        c = c + 1;

      // Berechne c := c + b unter Verwendung der Hilfsvaraible h
      h = 0;
      while (h != b)
       { c = c + 1;
         h = h + 1;
       }

      // Berechne c := c - 3
      c = c - 1;
      c = c - 1;
      c = c - 1;

      // Ausgabe des kommentierten Ergebnisses
      System.out.println("a + b - 3 = " + c);

      }
}
```

Um ein Java-Programm JP zum Laufen zu bringen sind folgende Schritte notwendig:

1. Lege ein Dateiverzeichnis JP an und schreibe jede Klasse K, die zu JP gehört unter dem Namen K.java in dieses Verzeichnis. Verwende dabei einen Editor nach freier Wahl, z.B. UltrEdit. Wähle JP als aktuelles Verzeichnis, z.B. durch das DOS-Kommando

   ```
   cd JP
   ```

2. Rufe den Übersetzer durch das Kommando

   ```
   javac K.java
   ```

   für jede Klasse K in JP auf, um das Programm zu übersetzen.

   Dabei wird jeweils ein Programm **`K.class`** in der Sprache Java-Bytecode generiert und automatisch in das Verzeichnis JP geschrieben.

3. Führe das Programm JP auf der abstrakten Maschine VM (virtual machine) durch den Aufruf des Java-Bytecode-Interpreters aus:

   ```
   java K
   ```

   Dabei ist zu beachten, dass die Klasse K eine main-Methode besitzen muss. Falls Eingabewerte, wie oben beschrieben, verwendet werden, so müssen sie bereits beim Aufruf mit angegeben werden, z.B. erzeugt der Aufruf

   java Ausdruck 12 6

   die Ausgabe

   ```
   a + b - 3 = 15
   ```