

## 13.8 Iteration

- Wiederholte Ausführung
- Definite Iteration
- Indefinite Iteration
- Termination

## 13.8.1 Wiederholte Ausführung

### Sparkonto

Eine Bank gibt für ein Sparkonto mit `amount` Euro einen Zinssatz von `interest` Prozent pro Jahr.

Aufgabe:

Erstellung einer Übersicht über die Kontoentwicklung der nächsten 20 Jahre.

### Sparkonto

Attribut	Aufgabe
<code>private double amount</code>	aktueller Kontostand
<code>private double interestFraction</code>	Zinssatz pro Jahr
Methode	Aufgabe
<code>amount() : double</code>	liefert den Kontostand
<code>addInterestYearly()</code>	fügt die jährlichen Zinsen hinzu
<code>addInterestOften(int howOften)</code>	fügt die aufgelaufenen Tageszinsen <code>howOften</code> -mal pro Jahr hinzu

### Hauptprogramm (Ausschnitt)

```
public class RunSA {
    // main omitted
    private static void runAccount (String sAmount, String sInterest, String sYears)
    { double amount = new Double (sAmount).doubleValue ();
      double interest = new Double (sInterest).doubleValue ();
      int nrOffYears = new Integer (sYears).intValue ();
      SavingsAccount sa = new SavingsAccount (amount, interest);
      int year = 0; // Initialisierung
      while (year <= nrOffYears) // Schleifenrest
      { System.out.println ("Year " + year + ": " + sa.amount ());
        sa.addInterestYearly ();
        year = year + 1;
      }
      System.out.println ("Year " + year + ": " + sa.amount ());
    }
}
```

```
public class SavingsAccount
{ private double amount; // current savings
  private double interestFraction; // interest fraction paid per year

  public SavingsAccount (double amount, double interest)
  { this.amount = amount;
    interestFraction = interest / 100.0;
  }

  public double amount ()
  { return amount; }

  private double computeInterest ()
  { return amount * interestFraction; }

  public void addInterestYearly ()
  { amount = amount + computeInterest (); }
}
```

### Monats- und Tageszinsen

```
public void addInterestMonthly ()
{ int MONTHS_PER_YEAR = 12;
  int month = 1;
  while (month <= MONTHS_PER_YEAR)
  { amount = amount + computeInterest () / MONTHS_PER_YEAR;
    month = month + 1;
  }

  public void addInterestOften (int howOften)
  { int count = 1;
    while (count <= howOften)
    { amount = amount + computeInterest () / howOften;
      count = count + 1;
    }
  }
}
```

### Beispielausgabe

```
> echo jährliche Verzinsung
> java RunSA 10000 4.5 8
Year 0: 10000.0
Year 1: 10450.0
Year 2: 10920.25
Year 3: 11411.66125
Year 4: 11925.18600625
Year 5: 12461.81937653125
Year 6: 13022.601248475155
Year 7: 13608.618304656537
Year 8: 14221.00612836608
Year 9: 14860.951404142554

> echo tägliche Verzinsung
> java RunSA 10000 4.5 8
Year 0: 10000.0
Year 1: 10460.24958498585
Year 2: 10941.682138019669
Year 3: 11445.272604326743
Year 4: 11972.040800945862
Year 5: 12523.053481952758
Year 6: 13099.426498735194
Year 7: 13702.327059694751
Year 8: 14332.976093951234
Year 9: 14992.650723836548
```

### Iteration, while-Anweisung, while-Schleife

Wiederholte Ausführung einer Gruppe von Anweisungen

#### Syntax

```
{statement} ::= while ( {expr} ) { {statements} }
```

#### Ausführung

1. Werte `{expr}` aus
2. Falls Ergebnis `false`, so ist die Ausführung der `while`-Anweisung beendet.  
*Die while-Schleife terminiert*
3. Falls Ergebnis `true`, so führe den *Rumpf der Schleife* (`{statements}`) aus  
*Schleifendurchlauf*
4. Weiter bei Punkt 1

## Wetterbericht

Eine Zeitung erhält eine Liste mit Namen von Orten im In- und Ausland, sowie den Mittagstemperaturen des Vortags. Die Temperaturen sind entweder in Celsius oder in Fahrenheit angegeben.

Aufgabe:

Erstellung einer Liste aus Städtenamen sowie Temperatur in Celsius und Fahrenheit.

Vorläufige Version

141

© 2001 Peter Thiemann

## Beispieleingabe

```
New York
F44
Berlin
C0
Paris
C4
San Francisco
F62
Honolulu
F81
Moskau
C-2
Helsinki
C-6
Dallas
F52
```

Vorläufige Version

142

© 2001 Peter Thiemann

## Gerüst: Temperaturkonversion

```
String town = reader.readString("Type the name of the town: ");
String code = reader.readString("Type scale (C or F) and temperature: ");
code = code.trim().toUpperCase();
char tempScale = code.charAt(0); // attention
int temp = new Integer (code.substring(1)).intValue();
if (!(tempScale == 'F' || tempScale == 'C'))
{ System.out.println("WeatherList: bad scale, " + tempScale); }
else
{ int fahrenheit;
  int celsius;
  if ( tempScale == 'F' )
  { fahrenheit = temp;
    celsius = (int) calc.fahrenheitIntoCelsius(temp);
  }
  else // tempScale == 'C'
  { celsius = temp;
    fahrenheit = (int) calc.celsiusIntoFahrenheit(temp);
  }
  System.out.println (town + " : " + celsius + "C = " + fahrenheit + "F");
}
```

Vorläufige Version

143

© 2001 Peter Thiemann

## Wiederholte Temperaturkonversion

```
String town = reader.readString("Type the name of the town: ");
while (town != null && !town.equals(""))
{ String code = reader.readString("Type scale (C or F) and temperature: ");
  code = code.trim().toUpperCase();
  char tempScale = code.charAt(0);
  int temp = new Integer (code.substring(1)).intValue();
  if (!(tempScale == 'F' || tempScale == 'C'))
  { System.out.println("WeatherList: bad scale, " + tempScale); }
  else
  { int fahrenheit;
    int celsius;
    if ( tempScale == 'F' )
    { fahrenheit = temp;
      celsius = (int) calc.fahrenheitIntoCelsius(temp);
    }
    else // tempScale == 'C'
    { celsius = temp;
      fahrenheit = (int) calc.celsiusIntoFahrenheit(temp);
    }
    System.out.println (town + " : " + celsius + "C = " + fahrenheit + "F");
  }
  town = reader.readString("Type the name of the next town: ");
}
```

Vorläufige Version

144

© 2001 Peter Thiemann

## Beispiel: Programmlauf

```
Type the name of the town: New York
Type scale (C or F) and temperature: F44
New York: 6C = 44F
Type the name of the next town: Berlin
Type scale (C or F) and temperature: C0
Berlin: 0C = 32F
Type the name of the next town: Paris
Type scale (C or F) and temperature: C4
Paris: 4C = 39F
Type the name of the next town: San Francisco
Type scale (C or F) and temperature: F62
San Francisco: 16C = 62F
Type the name of the next town: Honolulu
Type scale (C or F) and temperature: F81
Honolulu: 27C = 81F
```

Vorläufige Version

145

© 2001 Peter Thiemann

## Arten von Iteration

**Definite Iteration** Anzahl der Schleifendurchläufe ist vorab bekannt (RunSA)

**Indefinite Iteration** Anzahl der Schleifendurchläufe ist endlich, aber **nicht** vorab bekannt (WeatherList)

**Divergenz** Schleife wird unendlich oft durchlaufen (Nichttermination)

Vorläufige Version

146

© 2001 Peter Thiemann

## 13.8.2 Definite Iteration

Muster:

i Zählvariable (Index) für Anzahl der Schleifendurchläufe

```
int i = Startwert; // Initialisierung
while ( i <= Endwert ) // Schleifentest
// Abbruchbedingung
{ Schleifenrumpf;
  i = i + 1; // Schleifenindex
}
```

Vorläufige Version

147

© 2001 Peter Thiemann

## Beispiel

```
/** summation computes a summation.
 * @param n - the upper bound of the summation; must be nonnegative
 * @return the sum, 0+1+2+...+n */
public int summation(int n)
{ int total = 0;
  int i = 0;
  while ( i != n )
  // Invariant: total == 0+1+...+i
  { i = i + 1;
    total = total + i;
    System.out.println("i = " + i + "; total = " + total);
  }
  // Nach der Schleife: total == 0+1+...+n
  System.out.println("Finished: " + total);
  return total;
}
```

```
i = 1; total = 1
i = 2; total = 3
i = 3; total = 6
i = 4; total = 10
```

Vorläufige Version

148

© 2001 Peter Thiemann

## Divergenz

Berechnung des Methodenaufrufs `summation (-1)`;

terminiert nicht:

```
i = 1; total = 1
i = 2; total = 3
i = 3; total = 6
i = 4; total = 10
i = 5; total = 15
:
:
```

## 13.8.3 Indefinite Iteration

**Aufgabe:** Suche nach einem Zeichen `c` in einem String `s`.

1. Setze `index` auf 0
2. Falls `c` noch nicht gefunden und noch Zeichen in `s` vorhanden sind
  - (a) Falls `s.charAt (index) == c`, dann ist `c` gefunden
  - (b) Anderenfalls, setze `index = index + 1` und fahre fort mit Schritt 2

## 13.8.4 Termination

Eine Schleife terminiert, falls es einen Ausdruck `TE` vom Typ `int` gibt, so dass

- es gilt immer  $TE \geq 0$
- bei jedem Schleifendurchlauf wird der Wert von `TE` echt kleiner.

Hinreichende Bedingung, nicht notwendig

```
/** findChar locates the leftmost occurrence of a character in a string.
 * @param c - the character to be found
 * @param s - the string to be searched
 * @return the index of the leftmost occurrence of c in s;
 * return -1 if c does not occur in s */
public static int findChar (char c, String s)
{
    int index = 0;
    int len = s.length ();
    while (index < len && c != s.charAt (index))
    { // Invariante:
      // für alle 0 <= j < index gilt: c != s.charAt (j)
      index = index + 1;
    }
    // An dieser Stelle gilt:
    // !(index < len && c != s.charAt (index))
    // == (index >= len) || (index < len && c == s.charAt (index))
    if (index >= len)
    { return -1; }
    else
    { return index; }
}
```

### Beispiel: summation

```
/** summation sums up 1...n.
 * @param n - upper bound;
 * nonnegative
 * @return sum, 0+1+2+...+n */
public int summation(int n)
{ int total = 0;
  int i = 0;
  while ( i != n )
  // TE == n - i
  { i = i + 1;
    total = total + i;
    System.out.println
      ("i = " + i + "; total = " + total);
  }
  System.out.println("Finished: " + total);
  return total;
}
```

- $n - i$  wird bei jedem Schleifendurchlauf um 1 kleiner
- Vor Eintritt in die Schleife  $n - i == n - 0 == n >= 0$ , nach Spezifikation
- Zu Beginn des Schleifenrumpfs  $i != n$  &&  $n - i >= 0 \Rightarrow n - i > 0$
- Am Ende des Schleifenrumpfs, nach  $i = i + 1$ ;  $n - i > -1 \Leftrightarrow n - i >= 0$

### Beispiel: terminate

```
/** terminate counts a pair down to (0,0).
 * @param m - most significant number, m >= 0
 * @param n - least significant number, n >= 0
 */
static void terminate (int m, int n)
{ while (m > 0 || n > 0)
  { if (n > 0)
    { n = n - 1; }
    else // m > 0
    { n = 2 * m; m = m - 1; }
  }
}
=> Kriterium hilft nicht!
```

## Lösung: Lexikografische Ordnung

Definiere Ordnungsrelation  $<$  auf  $\text{int} \times \text{int}$  durch  $(m_1, n_1) < (m_2, n_2)$ , falls entweder

- $m_1 < m_2$  oder
- $m_1 = m_2$  und  $n_1 < n_2$ .

Solange die Schleife abläuft, gilt  $(0, 0) \leq (m, n)$  und bei jedem Schleifendurchlauf wird  $(m, n)$  echt kleiner bzgl.  $<$ .

Weitere Verallgemeinerung: Ausdruck wird kleiner in wohlfundierter Ordnung