1.6 Grundlagen/Abweisende Schleife mit while

1.6.1 Schleife

Die meisten Programmiersprachen haben Konstruktionen, die eine beliebige Wiederholung von Programmteilen ermöglichen. Diese Konstruktionen heißen Schleifen. Eine dieser Schleifen in Java ist die while-Schleife. Hier ein Beispiel:

```
import java.lang.*; // fuer Thread.sleep()
import java.io.*; // fuer System.out.print()

public class while_1a
{
   public static void main(String[] args) throws IOException, Exception
   {
      System.out.println("Vor der Schleife.");
      while(true)
      {
         System.out.print("In der Schleife.....\t");
         Thread.sleep(1000);    // 1000 ms
      }
//       System.out.println("Nach der Schleife.");
   }
}
```

Solange der Ausdruck in den runden Klammern hinter dem Schlüsselwort , die sogenannte Eintrittsbedingung (auch Schleifenkopf genannt), logisch wahr (`true`) ist, wird die darauffolgende Anweisung (alles bis zum Semikolon) oder der folgende Block (alles innerhalb geschweifter Klammern) ausgeführt.

In diesem Beispiel läuft die Schleife also endlos, und das Programm kann nur mit Strg-C abgebrochen werden. Anders das folgende Programm:

```
import java.lang.*; // fuer Thread.sleep()
import java.io.*; // fuer System.out.print()

public class while_1b
{
   public static void main(String[] args) throws IOException, Exception
   {
      boolean bedingung = false;
      System.out.println("Vor der Schleife.");
      while(bedingung)
      {
         System.out.print("In der Schleife.....\t");
         Thread.sleep(1000);    // 1000 ms
      }
      System.out.println("Nach der Schleife.");
   }
}
```

Hier würde die Schleife niemals durchlaufen werden. Dieser Schleifentyp heißt deshalb auch *Abweisende Schleife*.

1.6.2 Eigene Variable für die Eintrittsbedingung

In der Praxis ist die Eintrittsbedingung häufig mit einer eigenen Variablen gefüllt. Im folgenden Beispiel kann damit der Benutzer angeben, wie lange die Schleife durchlaufen werden soll:

```
import java.io.*;

public class while_2
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader roheingabe = new InputStreamReader(System.in);
      BufferedReader    eingabe    = new BufferedReader(roheingabe);
      int weiter = 1;

      while(weiter==1)
      {
         double d;
         double Aq;
                                    // Eingabe:
         System.out.print("Bitte den Leiterdurchmesser in mm eingeben:");
         d = Double.parseDouble(eingabe.readLine());
                                    // Verarbeitung:
         Aq=3.14159/4.0*d*d;
                                    // Ausgabe:
         System.out.println("Querschnittsflaeche="+ Aq + "mm^2\n");

                                    // Weiterer Durchlauf?
         System.out.print("Weiter (Ja=1/Nein=andere Zahl)?");
         weiter = Integer.parseInt(eingabe.readLine());
      }// while()
      System.out.println("Ende.");
   }
}
```

1.6.3 Arbeits-Variable für die Eintrittsbedingung

Es ist möglich, den Schleifendurchlauf von einer vorhandenen Variable abhängig zu machen; auch hier kann der Benutzer hiermit vorgeben, wie lange die Schleife durchlaufen werden soll:

```
import java.io.*;

public class while_3
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader roheingabe = new InputStreamReader(System.in);
      BufferedReader    eingabe    = new BufferedReader(roheingabe);
      double d=97.3;

      while(d>=0)
      {
         double Aq;
         System.out.print("Durchm. in mm eingeben (Ende m. Wert<0):");
         d = Double.parseDouble(eingabe.readLine());
         if(d>=0)
         {
            Aq=3.14159/4.0*d*d; // Verarbeitung
            System.out.println("Querschnittsflaeche="+ Aq + "mm^2\n");
```

```
         }// if()
      }// while()
      System.out.println("Ende.");
   }
}
```

Negative Werte für den Drahtdurchmesser d kommen nicht vor, also bedeuten sie das Programmende. Diese Lösung ist problematisch, sobald *doch* einmal ein solcher reservierter Wert eingegeben werden soll — z.B. tauchen negative Innenwiderstände, Frequenzen gleich null, Verstärkungen kleiner eins eben *doch* manchmal auf und bewirken dann ein unerwartetes Programmverhalten.

1.6.4 Eintrittsbedingung als Ergebnis fortlaufender Berechnungen

Im folgenden Beispiel soll gerechnet werden, bis die Zahl null erreicht wird:

```
import java.lang.*;
import java.io.*;

public class while_4a
{
   public static void main(String[] args) throws IOException, Exception
   {
      int zahl;
      zahl = 5;
      while(zahl >= 0)
      {
         System.out.println(zahl);
         zahl = zahl - 1; // oder "--zahl;"
         Thread.sleep(500);
      }
      System.out.println("Ende.");
   }
}
```

`zahl=zahl-1` bedeutet: Der Wert für `zahl` wird aus der Variablen entnommen, dieser Wert wird mit eins in eine Subtraktion einbezogen, das Ergebnis wird wieder in die Variable `zahl` geschrieben. Im Endeffekt wird `zahl` also um eins verringert. Auch bei dieser Konstruktion ist Vorsicht angesagt. Hier ist eine Endlosschleife vorhanden:

```
import java.lang.*;
import java.io.*;

public class while_4b
{
   public static void main(String[] args) throws IOException, Exception
   {
      short zahl;
      zahl = 32760;
      while(zahl <= 32767)
      {
         System.out.println(zahl);
         ++zahl;
         Thread.sleep(500);
      }
      System.out.println("Ende.");
   }
```

```
}
```

Der Wert für Zahl kann nie groesser als 32767 sein (Datentyp `short`), daher ist die Eintrittsbedingung immer erfüllt.

1.6.5 Berechnungen im Schleifenrumpf

Das folgende Programm zeigt die Berechnung einer Summe in einer Schleife.

```
import java.io.*;

public class while_5a
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader roheingabe = new InputStreamReader(System.in);
      BufferedReader    eingabe    = new BufferedReader(roheingabe);
      int anzahl;
      int stueckpreis=0; // irgendein Wert groesser oder gleich null
      int summe=0;

      while(stueckpreis>=0)
      {
         System.out.print("Stueckpreis in ct (Ende mit Wert<0):");
         stueckpreis = Integer.parseInt(eingabe.readLine());
         System.out.print("Anzahl:");
         anzahl = Integer.parseInt(eingabe.readLine());
         if(stueckpreis>=0)
         {
            summe = summe + anzahl * stueckpreis;
            System.out.println("Summe = " + summe);
         }
      }
      System.out.println("Ende.");
   }
}
```

Wie kommt man auf diesen Programmtext? Anhand des folgenden (einfacheren) Beispiels sollen Schritte dazu aufgezeigt werden.

- Geforderte Ausgabe: 1 2 3 4 5 6 7 ...
- Schritt 1: Lösung als einfache Sequenz

  ```
  System.out.println(1);
  System.out.println(2);
  System.out.println(3);
  ...
  ```

- Schritt 2: Variable statt Konstanten verwenden

  ```
  i=1; System.out.println(i);
  i=2; System.out.println(i);
  i=3; System.out.println(i);
  ...
  ```

- Schritt 3: Variable automatisch berechnen

```
i=0;
i=i+1; System.out.println(i);
i=i+1; System.out.println(i);
i=i+1; System.out.println(i);
...
```

- Schritt 4: Immer wiederkehrende Anweisungen in eine Schleife setzen:

```
i=0;
while(1)
{
   i=i+1; System.out.println(i);
}
```

- Schritt 5: Schleifenbedingung wählen, um gegebenenfalls die Anzahl der Durchläufe zu begrenzen:

```
i=0;
while(i<40)
{
   i=i+1; System.out.println(i);
}
```

1.6.6 Gleitkommavariablen und Endlosschleifen

Gleitkommavariablen sollten (wenn irgend möglich) niemals zur Steuerung von Schleifen (oder Verzweigungen) verwendet werden. Die folgende Schleife ist z.B. endlos:

```
import java.lang.*;
import java.io.*;

public class while_4c
{
   public static void main(String[] args) throws IOException, Exception
   {
      float zahl;
      zahl = 21345678.0F; // 21,3 Mio.
      while(zahl >= 0.0F)
      {
         System.out.println(zahl);
         --zahl;
         Thread.sleep(500);
      }
      System.out.println("Ende.");
   }
}
```

Beim Datentyp `float` ist nämlich 20.000.000+1.0 = 20.000.000; bei Gleitkommazahlen bleibt nur die relative Genauigkeit etwa konstant, die absolute Genauigkeit sinkt mit zunehmendem Zahlenwert. Auch die folgende Schleife (20 Durchläufe erwartet) kommt nie an:

```
import java.lang.*;
import java.io.*;

public class while_4d
```

```
{
   public static void main(String[] args) throws IOException, Exception
   {
      float zahl;
      zahl = 16777200.0F; // 21,3 Mio.
      while(zahl < 16777220.0F)
      {
         System.out.println(zahl);
         ++zahl;
         Thread.sleep(500);
      }
      System.out.println("Ende.");
   }
}
```

Nach 16777215 ist die nächste darstellbare Zahl 16777216, die Zahl danach aber 16777218. Nicht nur das:

```
import java.lang.*;
import java.io.*;

public class while_4e
{
   public static void main(String[] args) throws IOException, Exception
   {
      float zahl;
      zahl = 16777200.0F;
      while(zahl < 16777217.0F)
      {
         System.out.println(zahl);
         ++zahl;
         Thread.sleep(500);
      }
      System.out.println("Ende.");
   }
}
```

Die Zahlen 16777216 und 16777217 werden intern gleich dargestellt, sind also nicht unterscheidbar.

```
import java.lang.*;
import java.io.*;

public class while_4f
{
   public static void main(String[] args) throws IOException, Exception
   {
      float zahl;
      zahl = 16777216F;
      while(zahl == 16777217F)
      {
         System.out.println("Dies hier passiert nieeee....");
         Thread.sleep(500);
      }
      System.out.println("Ende.");
   }
}
```