Einführung in die Programmierung

by André Karge

Übung - Schleifen, Kontrollstrukturen, Klassen

Heute

- Vorstellung Übungsblatt 01
- Schleifen
- Kontrollstrukturen
- Klassen

Schleifen

Schleifen - Insel 2.6

Definition

Eine Schleife ist ein Programmkonstrukt, das dazu dient bestimmte Anweisungen mehrfach abzuarbeiten.
Sie besteht aus einer **Schleifenbedingung** und einem **Schleifenkörper**.
Konstrukte:

- while-Schleife
- do-while-Schleife
- for-Schleife

While Schleife - Insel 2.6.1

- While = Abweisende Schleife
- Schleifenbedingung muss vor erstem Abarbeiten erfüllt sein
- vor jedem Schleifendurchlauf wird die Bedingung geprüft

```
int i = 0;
while (i < 10) { // Prüfen der Bedingung bei jedem Ausführen
  System.out.println("aktueller Durchlauf: " + i);
  i++;
  // Springe wieder in die Zeile der Bedingung
}
```

While Schleife - Insel 2.6.1

- While = Abweisende Schleife
- Schleifenbedingung muss vor erstem Abarbeiten erfüllt sein
- vor jedem Schleifendurchlauf wird die Bedingung geprüft

```
int i = 0;
while (i < 10) { // Prüfen der Bedingung bei jedem Ausführen
  System.out.println("aktueller Durchlauf: " + i);
  i++;
  // Springe wieder in die Zeile der Bedingung
}
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*

While Schleife - Insel 2.6.1

- While = Abweisende Schleife
- Schleifenbedingung muss vor erstem Abarbeiten erfüllt sein
- vor jedem Schleifendurchlauf wird die Bedingung geprüft

```
int i = 0;
while (i < 10) { // Prüfen der Bedingung bei jedem Ausführen
  System.out.println("aktueller Durchlauf: " + i);
  i++;
  // Springe wieder in die Zeile der Bedingung
}
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*
Antwort: *10x - Zahlen von 0 - 9*

While Schleife - Insel 2.6.1

```java
int i = 42;
while (i < 10) { // Prüfen der Bedingung bei jedem Ausführen
  System.out.println("aktueller Durchlauf: " + i);
  i++;
  // Springe wieder in die Zeile der Bedingung
}
```

While Schleife - Insel 2.6.1

```java
int i = 42;
while (i < 10) { // Prüfen der Bedingung bei jedem Ausführen
  System.out.println("aktueller Durchlauf: " + i);
  i++;
  // Springe wieder in die Zeile der Bedingung
}
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*

While Schleife - Insel 2.6.1

```java
int i = 42;
while (i < 10) { // Prüfen der Bedingung bei jedem Ausführen
  System.out.println("aktueller Durchlauf: " + i);
  i++;
  // Springe wieder in die Zeile der Bedingung
}
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*
Antwort: *0x - kein Output*

Do-While Schleife - Insel 2.6.2

- Do-While = Annehmende Schleife
- Schleifenbedingung muss am Ende des Blocks erfüllt sein um ihn erneut auszuführen
- wird immer mindestens einmal ausgeführt

```
int i = 0;
do {
  System.out.println("aktueller Durchlauf: " + i);
  i++
} while(i < 10);
```

Do-While Schleife - Insel 2.6.2

- Do-While = Annehmende Schleife
- Schleifenbedingung muss am Ende des Blocks erfüllt sein um ihn erneut auszuführen
- wird immer mindestens einmal ausgeführt

```
int i = 0;
do {
  System.out.println("aktueller Durchlauf: " + i);
  i++
} while(i < 10);
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*

Do-While Schleife - Insel 2.6.2

- Do-While = Annehmende Schleife
- Schleifenbedingung muss am Ende des Blocks erfüllt sein um ihn erneut auszuführen
- wird immer mindestens einmal ausgeführt

```
int i = 0;
do {
  System.out.println("aktueller Durchlauf: " + i);
  i++
} while(i < 10);
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*
Antwort: *10x - Zahlen von 0-9*

Do-While Schleife - Insel 2.6.2

```
int i = 42;
do {
  System.out.println("aktueller Durchlauf: " + i);
  i++
} while(i < 10);
```

Do-While Schleife - Insel 2.6.2

```
int i = 42;
do {
  System.out.println("aktueller Durchlauf: " + i);
  i++
} while(i < 10);
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*

Do-While Schleife - Insel 2.6.2

```
int i = 42;
do {
  System.out.println("aktueller Durchlauf: " + i);
  i++
} while(i < 10);
```

Frage: *Wie häufig wird die Schleife durchlaufen? Was ist der Output?*
Antwort: *1x - Zahl 42*

For Schleife - Insel 2.6.3

- Spezielle Variante der While Schleife
- For Schleife = Abweisende Schleife
- Hat einen komplexeren Schleifenkopf als die While Schleife

Schleifenkopf

```
for (Initialisierung; Schleifenbedingung; Schleifen-Inkrement) {...}
```

- Initialisierung: Schleifenvariable (vgl. *int i = 0;* der while und der do-while Schleife)
- Schleifenbedingung: Bedingung für Eintritt und erneutes Ausführen der Schleife (vgl. *(i< 10)* der while und der do-while Schleife)
- Schleifen-Inkrement: Wie soll die Schleifenvariable am Ende jedes Durchlaufes verändert werden (vgl. *i++;* der while und der do-while Schleife)

Beispiel For Schleife

```
for (int i=0; i<10; i++) {
  System.out.println("aktueller Durchlauf: " + i);
}
```

Selbes Beispiel als While Schleife

```
int i = 0;
while(i<10) {
  System.out.println("aktueller Durchlauf: " + i);
  i++;
}
```

Wir sparen 2 Zeilen mit einem For Loop

Kontrollstrukturen

Kontrollstrukturen - Insel 2.5

Definition

Kontrollstrukturen um Teile von Programmen nur unter bestimmten Bedingungen auszuführen.
Konstrukte:

- if Verzweigung
- if-else Verzweigung
- switch-case Verzweigung

Kontrollstrukturen - *if*-Verzweigung - Insel 2.5.1

Generelle if-Verzweigung

```
if ( Bedingung ) {
  // Befehls-Block, der beim Erfüllen der Bedingung ausgeführt wird
}
else { // else + Block ist Optinal
  // Befehls-Block, der beim Nichterfüllen der Bedingung ausgeführt wird
}
```

Beispiel

```
if (true) {
  System.out.println("Verzweigung erreicht");
}

if (false) { // Bedingung ist false -> if(Bedingung) wird nur ausgeführt, wenn Bedingung true ist
  System.out.println("Wird nicht erreicht");
}
else { // Bedingung ist false -> else wird nur ausgeführt, wenn Bedingung false ist
  System.out.println("Wird erreicht");
}
```

Kontrollstrukturen - *if*-Verzweigung - Insel 2.5.1

Nutzung von Bedingungen

- Hier kommen die logischen Operatoren in Benutzung

```
// Gibt die Differenz zwischen zwei Zahlen aus
int getDifference(int a, int b) {
  // Wenn b größer oder gleich a -> ziehe linke Zahl von rechter ab
  if (a <= b) {
    return b - a;
  }
  // Ansonsten ziehe rechte Zahl von linker ab
  else {
    return a - b;
  }
}
```

Zusammengesetzte Bedingung

```
// logisches UND
if (Bedingung1 & Bedingung2) {
  // Block wird nur ausgeführt, wenn beide Bedingungen erfüllt sind
}
// logisches ODER
if (Bedingung1 | Bedingung2) {
  // Block wird ausgeführt, wenn eine der Bedingungen erfüllt ist
}

// Verschachtelung von Bedingungen
if (Bedingung1 & (Bedingung2 | (Bedingung3 & Bedingung4)) & Bedingung5)

// Beispiel:
int a = 15;
if (3 < 5 & (15 >= a | (2 == 2 & 5 == 4)) & 42 > a) {
  System.out.println("Bedingung erfüllt!");
}
else {
  System.out.println("Bedingung nicht erfüllt!");
}
```

Zusammengesetzte Bedingung

```
// logisches UND
if (Bedingung1 & Bedingung2) {
  // Block wird nur ausgeführt, wenn beide Bedingungen erfüllt sind
}
// logisches ODER
if (Bedingung1 | Bedingung2) {
  // Block wird ausgeführt, wenn eine der Bedingungen erfüllt ist
}

// Verschachtelung von Bedingungen
if (Bedingung1 & (Bedingung2 | (Bedingung3 & Bedingung4)) & Bedingung5)

// Beispiel:
int a = 15;
if (3 < 5 & (15 >= a | (2 == 2 & 5 == 4)) & 42 > a) {
  System.out.println("Bedingung erfüllt!");
}
else {
  System.out.println("Bedingung nicht erfüllt!");
}
```

Frage: *Wird die Bedingung erfüllt oder nicht?*

Zusammengesetzte Bedingung

```
// logisches UND
if (Bedingung1 & Bedingung2) {
  // Block wird nur ausgeführt, wenn beide Bedingungen erfüllt sind
}
// logisches ODER
if (Bedingung1 | Bedingung2) {
  // Block wird ausgeführt, wenn eine der Bedingungen erfüllt ist
}

// Verschachtelung von Bedingungen
if (Bedingung1 & (Bedingung2 | (Bedingung3 & Bedingung4)) & Bedingung5)

// Beispiel:
int a = 15;
if (3 < 5 & (15 >= a | (2 == 2 & 5 == 4)) & 42 > a) {
  System.out.println("Bedingung erfüllt!");
}
else {
  System.out.println("Bedingung nicht erfüllt!");
}
```

Frage: *Wird die Bedingung erfüllt oder nicht?*
Antwort: *(true & (true | (true & false)) & true) = true*

Kontrollstrukturen - Mehrfachverzweigung - Insel 2.5.2

- Wenn man mit einer Verzweigung mehr als nur 2 unterschiedliche Fälle abdecken möchte
- Bspw.: Annahme von 3 Fällen (Kind, Jugendlicher, Erwachsener) → jenachdem, welches Alter ein Nutzer hat, soll eine von 3 Ausgaben gezeigt werden

```
void zeigeAusgabe(int alter) {
  if (alter < 10) {
    System.out.println("Die Legokiste ist hinten rechts");
  }
  else if (alter >= 10 && alter < 18) {
    System.out.println("Die Playstation ist hinten links");
  }
  else { // nur noch Zahlen >= 18 übrig, daher müssen wir das nicht mehr angeben
    System.out.println("Das Bier ist im Kühlschrank");
  }
}
```

Bedingungsvarianten

- Bedingungen, die durch einfaches "&"getrennt sind, werden alle der Reihe nach geprüft und ihr Wahrheitswert berechnet
- Bedingungen, die durch doppeltes "&&"getrennt sind, werden so lange der Reihe nach geprüft, bis ein Ausdruck false ist. Alle folgenden Ausdrücke werden ignoriert
- selbiges gilt auch für den ODER Operator | (| vs. ||)

Kontrollstrukturen - Switch-Case - Insel 2.5.4

Switch-Case Verzweigung löst das Problem von zu vielen Mehrfachverzweigungen.

```
switch ( meineVariable ) {
  case 1:
    // Ausführung wenn meineVariable == 1
    System.out.println("first case");
    break;
  case 2:
    // Ausführung wenn meineVariable == 2:
    System.out.println("first case");
    break;
  default:
    System.out.println("wenn weder case 1 noch case 2 eintreffen");
}
```

Kontrollstrukturen - Switch-Case - Insel 2.5.4

Switch-Case Verzweigung löst das Problem von zu vielen Mehrfachverzweigungen.

```
switch ( meineVariable ) {
  case 1:
    // Ausführung wenn meineVariable == 1
    System.out.println("first case");
    break;
  case 2:
    // Ausführung wenn meineVariable == 2:
    System.out.println("first case");
    break;
  default:
    System.out.println("wenn weder case 1 noch case 2 eintreffen");
}
```

- Es wird hierbei immer die Variable *meineVariable* getestet
  case 1 $\rightarrow$ if(meineVariable $==$ 1) {...}
  case 2 $\rightarrow$ if(meineVariable $==$ 2) {...}
  usw.

Kontrollstrukturen - Switch-Case - Insel 2.5.4

```
switch ( meineVariable ) {
  case 1:
    // Ausführung wenn meineVariable == 1
    System.out.println("first case");
    break;
  case 2:
    // Ausführung wenn meineVariable == 2:
    System.out.println("first case");
    break;
  default:
    System.out.println("wenn weder case 1 noch case 2 eintreffen");
}
```

- Wichtig: **break;** am Ende jedes Cases verhindert Ausführung von mehreren Cases
- Der *default*-case wird nur dann ausgeführt wenn kein anderer case erreicht wird

Klassen

Klassen - Insel 3

Klasse

- Grundkonzept der Objektorientierten Programmierung
- definiert neuen Typen (Komplexer Typ)
- beschreibt Funktionsweise von Objekten dieses Types
- Kann man mit einem Bauplan vergleichen

Klassen - Insel 3

Klasse

- Grundkonzept der Objektorientierten Programmierung
- definiert neuen Typen (Komplexer Typ)
- beschreibt Funktionsweise von Objekten dieses Types
- Kann man mit einem Bauplan vergleichen

Objekte

- Ist ein Exemplar (Instanz / Ausprägung) einer Klasse
- Beispiel: Hammer, Meißel, Spachtel sind alles Objekte der Klasse Werkzeug

Klassen - Insel 3

Klasse

- Grundkonzept der Objektorientierten Programmierung
- definiert neuen Typen (Komplexer Typ)
- beschreibt Funktionsweise von Objekten dieses Types
- Kann man mit einem Bauplan vergleichen

Objekte

- Ist ein Exemplar (Instanz / Ausprägung) einer Klasse
- Beispiel: Hammer, Meißel, Spachtel sind alles Objekte der Klasse Werkzeug

Wozu das ganze? - Um komplexe Softwaresysteme besser modellieren, entwerfen und implementieren zu können

Klassen in Java

- schon in der ersten Übung gesehen:

```
/**
 * @author
 *
 */
public class ClassName {       // Klassenname
  /*
   * Attributes               // Attribute: [visibility] [static] [final] type name;
   */

  /*
   * optional: Constructors   // Konstruktor(en): [visibility] ClassName(...) {...}
   */

  /*
   * Methods                  // Methoden: [visibility] [static] return-type name(...) {...}
   */
}
```

Klassen in Java - Attribute - Insel 3.4.4

- Variablen in Klassen = Objektvariablen
- jede Instanz (jedes Objekt) einer Klasse hat seine eigene Ausprägung der gegebenen Variablen
- Objektvariablen bilden den *Zustand* des Objekts (Objektattribute)

Klassen in Java - Attribute - Insel 3.4.4

- Variablen in Klassen = Objektvariablen
- jede Instanz (jedes Objekt) einer Klasse hat seine eigene Ausprägung der gegebenen Variablen
- Objektvariablen bilden den *Zustand* des Objekts (Objektattribute)

Beispiel:

```
public class Werkzeug {
  int gewicht;
  String besitzer = "Peter Zwegat";
  ...
}
```

Klassen in Java - Instanziierung - Insel 3.4

- nehmen wir an, wir haben eine Klasse definiert
- um ein neues Objekt der Klasse anzulegen nutzt man das Stichwort **new**
- Schema: Klasse bezeichner = new Klasse();

Klassen in Java - Instanziierung - Insel 3.4

- nehmen wir an, wir haben eine Klasse definiert
- um ein neues Objekt der Klasse anzulegen nutzt man das Stichwort **new**
- Schema: Klasse bezeichner = new Klasse();

Beispiel:

```
public class Werkzeug {...} // anlegen einer neuen Klasse

public class Main {
  public static void main(String[] args) {
    Werkzeug hammer = new Werkzeug(); // hammer ist eine neue Instanz der Klasse Werkzeug
  }
}
```

Klassen in Java - Klassenmethoden

- neben Attributen, können Klassen auch Methoden haben
- werden innerhalb einer Klasse angelegt und definieren Funktionen von allen Ausprägungen der Klasse
- Beispiel: eine Klasse 'Auto' hat die Funktion 'fahren', was bedeutet, dass alle Instanzen der Klasse eine Funktion 'fahren' haben

Klassen in Java - Klassenmethoden

- neben Attributen, können Klassen auch Methoden haben
- werden innerhalb einer Klasse angelegt und definieren Funktionen von allen Ausprägungen der Klasse
- Beispiel: eine Klasse 'Auto' hat die Funktion 'fahren', was bedeutet, dass alle Instanzen der Klasse eine Funktion 'fahren' haben

```
public class Werkzeug {
  int gewicht; // anlegen einer Klassenvariable gewicht
  String name; // anlegen einer Klassenvariable name

  public void print() {
    System.out.println("Hallo, ich bin ein Werkzeug");
  }
}
```

Klassen in Java - Klassenmethoden

Aufruf von Klassenmethoden

- Methoden, die in Klassen angelegt sind können nur auf Objektinstanzen der Klasse aufgerufen werden
- visibility von Methoden und Variablen gibt an, ob man außerhalb der Klasse Zugriff auf diese bekommt
- visibility: *public*, *private*, *protected*

Beispiel:

```
public class Main {
  public static void main(String[] args) {
    Werkzeug hammer = new Werkzeug();

    hammer.print(); // Aufruf der Klassenmethode 'print()' auf dem Objekt
  }
}
```

Klassen in Java - Membervariablen

- Zugriff auf Memberattribute über das Stichwort **this.variablename** (Immer im Bezug auf die Ausprägung der Objekte)
- Ein Objekt vom Typen Wekzeug mit *name=Hammer* und *gewicht=1kg* produziert andere Ausgabe beim Aufruf von *printObjectInfo* als ein Objekt mit *name=Feile*

```
public class Werkzeug {
  int gewicht; // anlegen einer Membervariablen gewicht
  String name; // anlegen einer Membervariablen name

  public void printObjectInfo() {
    System.out.println("Hallo, ich bin ein Werkzeug");
    System.out.println("Und zwar bin ich ein " + this.name); // Aufruf der Membervariable über this
    System.out.println("Ich wiege " + this.gewicht + "kg"); // selbiges hier
  }
}
```

Klassen in Java - Klassenkonstruktor - Insel 3.4.6

- Konstruktoren sind besondere Methoden von Klassen
- werden beim Instanziieren (**new** Object) von Objekten aufgerufen
- Hat eine besondere Signatur ohne Rückgabetyp: [visibility] ClassName(...) ...

```
public class Werkzeug {
  int gewicht;
  String name;

  public Werkzeug() { // Default Konstruktor
    this.gewicht = 0; // Default wert für gewicht
    this.name = "";   // Default wert für name
  }
  public Werkzeug(int gewicht, String name) { // Überladener Konstruktor
    this.name = name;       // schreibe gegebenen name in die Membervariable
    this.gewicht = gewicht; // schreibe gegebenes gewicht in die Membervariable
  }
}
```

Klassen in Java - Klassenkonstruktor - Insel 3.4.6

```java
public class Main {
  public static void main(String[] args) {
    Werkzeug meinWerkzeug = new Werkzeug(); // Aufruf des Default-Konstruktors
    Werkzeug anderesWerkzeug = new Werkzeug(5, "Hammer"); // Aufruf des überladenen Konstruktors
  }
}
```

Fragen?

Übungsaufgabe - 15 min

1. Schreiben Sie eine Main-Klasse mit einer Main-Methode
2. Erstellen Sie eine zusätzliche Klasse 'Autor' mit den Attributen 'name', 'nachname', 'geboren', 'webseite', 'notiz'
3. Instanziieren Sie mehrere Objekte der Klasse Autor innerhalb der Main-Methode der Main-Klasse
4. Erweitern Sie Ihre Autor-Klasse um eine Methode 'printInformation', die alle Attribute auf der Konsole ausgibt und rufen Sie die Methode auf allen instanziiereten Objekten auf.
5. Erweitern Sie Ihre Klasse 'Autor' um einen Standard Konstruktor und einen überladenen Konstruktor
6. Testen Sie Ihre neue Konstruktoren auf neuen Instanzen
7. Erweitern Sie die Klasse Autor um eine Methode mit visibility *private* und versuchen Sie diese in der main-Methode aufzurufen

Übungsaufgabe - 5 min

1. Schreiben Sie ein programm, dass in einer Schleife von 1 bis 100 zählt;
2. Erweitern Sie dieses programm und geben sie "fizz"für alle Zahlen teilbar durch 3; "buzz"für alle Zahlen teilbar durch 5 und "fizzbuzz"für alle Zahlen die durch 15 teilbar sind aus