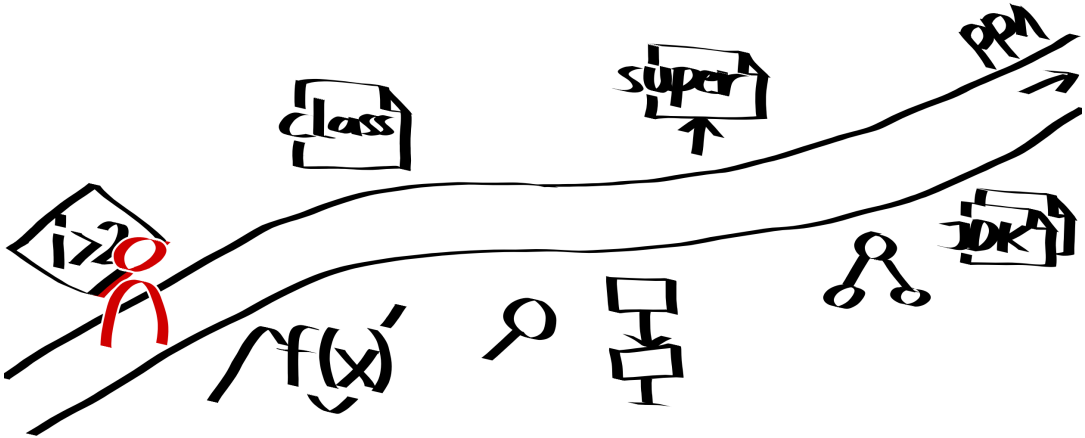
Kapitel 1: Grundlagen

switch-Verzweigung, for-Schleife

Wintersemester 2023/24

Stand 22. Oktober 2023, 18:46 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

Heinrich Heine
Universität Düsseldorf

+ Tempo/Schwierigkeit passt im Schnitt
+ Live-Beispiele
+ zusätzliche Beispiele online
+ Chat-/Online-Aktivität
– Ton in 16.12
? Anleitung zu jedem empfohlenen Editor
? Ergebnis immer Double, wenn Integer und Double verrechnet werden?
? nach keine Kennung und Übungstermin gesucht: siehe (öffentliche) Kursseite zu Woche 2

switch-Verzweigung: Switch Statement

- Motivation: Vergleiche mit derselben Variablen kürzer notieren

Syntax — Java

```java
switch (comparision_expression) {
    case expression_1:
        // ausgeführt, wenn Werte von comparision_expression und expression_1 gleich
        statements_1
    // ...
    case expression_n:
        // ausgeführt, wenn Werte von comparision_expression und expression_n gleich
        // oder vorherige Bedingung(en) erfüllt + zwischendurch kein break;
        statements_n
    default:  // (optional)
        // ausgeführt, wenn kein case erfüllt
        // oder vorherige Bedingung(en) erfüllt + zwischendurch kein break;
        statements_default
}
// nach einem break oder dem letzten case/default geht es hier weiter
```

Beispiel: Wochentage I

Heinrich Heine Universität Düsseldorf

DayOfWeek.java — Java

```java
public class DayOfWeek {

    public static void main(String[] args) {
        int dayOfWeek = Integer.parseInt(args[0]);
        switch(dayOfWeek) {
            case 0:
                System.out.println("Montag");
                break;
            // ...
            case 6:
                System.out.println("Sonntag");
                break;
            default:
                System.out.println("ungültige Eingabe");
        }
    }

}
```

Beispiel: Wochentage II

Heinrich Heine
Universität Düsseldorf

```
% java DayOfWeek 1
Dienstag
% java DayOfWeek 7
ungültige Eingabe
```

Fallthrough

- gezieltes Weglassen von `break;` erlaubt gleichen Code für gleiche Fälle
- wenn in komplexeren Fällen benutzt: Absicht kommentieren!

```
1     public static void main(String[] args) {
2         int choice = Integer.parseInt(args[0]);
3         switch(choice) {
4             case 1:
5             case 3:
6                 System.out.println("1,2 €"); // Ausgabe bei Argument 1 oder 3
7                 break;
8             case 2:
9                 System.out.println("1,0 €"); // Ausgabe bei Argument 2
10                break;
11            default:
12                System.out.println("ungültige Eingabe");
13        }
14    }
```

switch-Verzweigung: Switch Expression

Heinrich Heine
Universität Düsseldorf

- seit Java 14
- wer interessiert ist:
  - Video im Lernmodul
  - `https://howtodoinjava.com/java14/switch-expressions/`

Abkürzungen bei arithmetischen Ausdrücken[1]

- Abkürzung von Zuweisungen, wo auf linker und rechter Seite dieselbe Variable vorkommt
- `i` sei für jedes Beispiel zu Beginn 2:

Ausdruck	Wert	Effekt
`i = i * (10 + 2)`	24	multipliziert `i` mit 12
`i *= 10 + 2`	24	multipliziert `i` mit 12 (Klammern implizit)
`i += 1`	3	erhöht `i` um 1
`i++`	2	erhöht `i` um 1 (Postinkrement)
`++i`	3	erhöht `i` um 1 (Präinkrement)
`i--`	2	erniedrigt `i` um 1 (Postdekrement)

- Praxis: Benutzen Sie den Wert dieser Ausdrücke nicht, sondern nur den Effekt!

[1] Compound Assignment Operators

Es ist kompliziert . . . [2]

```java
int x = 3;

// x = x * 2.5;
```

[2] https://docs.oracle.com/javase/specs/jls/se21/html/jls-15.html#jls-15.26.2

Es ist kompliziert . . . [2]

```java
int x = 3;

// x = x * 2.5;
// error: incompatible types: possible lossy conversion from double to int

x *= 2.5;
// 7
```

- `*=` usw. casten das Ergebnis automatisch auf den Typen der Zielvariable:
  `x *= 2.5;` ist hier gleich zu `x = (int)(x * 2.5);`

[2] https://docs.oracle.com/javase/specs/jls/se21/html/jls-15.html#jls-15.26.2

for-Schleifen

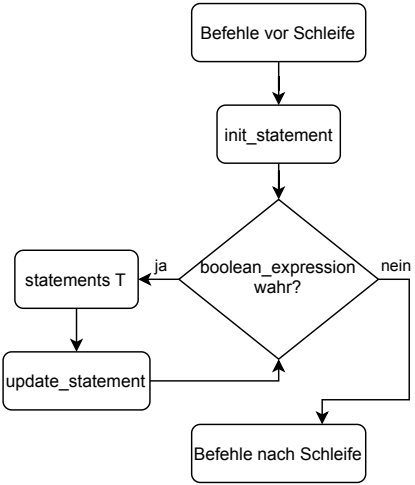
- Motivation: Kompakte Schleifen-Notation beim Durchzählen

Syntax — Java

```java
for (init_statement; boolean_expression; update_statement) {
    statements T  // „Schleifenrumpf“ (Body) wird so lange ausgeführt
                  // wie boolean_expression wahr ist
}
// hier gehts es weiter, sobald boolean_expression falsch ist
```

- `init_statement`: Ausführung einmalig zu Beginn der Schleife
- `update_statement`: Ausführung immer nach dem Ausführen des Schleifenrumpfs (vor Prüfung der Bedingung)

for-Schleife: Ablauf



Beispiel: Argumente ausgeben

Heinrich Heine Universität Düsseldorf

PrintArgs.java — Java

```java
public class PrintArgs {

    public static void main(String[] args) {
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }

}
```

```
% java PrintArgs 1 -2 10 hi
1
-2
10
hi
```

Anmerkungen zur for-Schleife

- Initialisierungs- und Update-Statement dürfen beliebiges Statement enthalten:
  - jeder Teil darf aber auch leer sein, z. B. `for(; zufallszahl < .5;)`
- Variablen, die im Initialisierungs-Statement deklariert werden, in der Schleife sichtbar, aber nicht außerhalb

```
1 for(int i = 10, j = 0; i >= 0; ) {
2     System.out.println(i + " " + j);
3     i--;
4     j++;
5 }
6 // i und j hier nicht mehr verfügbar
```

Was ist das hier?

```
for(;;);
```

Was ist das hier?


```
for(;;);
```

- Bedingungsteil muss boolean Expression oder leer sein
  - leerer Bedingungs-Teil entspricht `true`
    - → Endlosschleife, wenn kein `break` o. ä. vorhanden

for, while, do-while: Wann benutze ich was?

Heinrich Heine Universität Düsseldorf

- bewiesen: alle Schleifenformen gleich mächtig $\rightarrow$TheoInfo
  - $\rightarrow$ Schleifen können ineinander umgeformt werden
- typische Verwendungsszenarien:
  - for-Schleife: Durchzählen (Zahlen, Array), Anzahl maximaler Durchläufe typischerweise vor Beginn der Schleife bekannt
  - while-Schleife: Bedingung von äußeren Umständen abhängig (Zufall, noch zu lesende Eingaben)
  - do-while-Schleife: wie while-Schleifen, aber mind. 1 Durchlauf

Beispiel: Schema Umformung for↔while-Schleife

Heinrich Heine Universität Düsseldorf

```
for (init_statement; boolean_expression; update_statement) {
    statements_T
}
```

. . . ist im Wesentlichen gleichbedeutend zu . . .

```
init_statement;
while (boolean_expression) {
    statements_T
    update_statement;
}
```

Überlegen Sie selbst: Wie formt man . . .

- . . . eine do-while-Schleife in eine while-Schleife um?
- . . . eine switch-Verzweigung in eine if-Verzweigung um?
- . . .

break & continue in Schleifen

Heinrich Heine Universität Düsseldorf

- `break;`
  - bricht umgebene Schleife sofort ab
  - springt hinter die Schleife
- `continue;`
  - bricht aktuellen Schleifendurchlauf ab
  - springt an das Ende des Schleifenrumpfs
- selten verwendet, da Sprünge unübersichtlich
  - besser: geeignete Abbruchbedingungen für Schleifen wählen

Beispiel: break vermeiden I

Heinrich Heine Universität Düsseldorf

Break.java — Java

```java
// gibt alle Argumente aus, bis ein Argument 42 ist (gibt die 42 trotzdem aus)
public class Break {
    public static void main(String[] args) {
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
            if(Integer.parseInt(args[i]) == 42) {
                break;
            }
        }
    }
}
```

- `break` z. B. mit zusätzlichem Boolean vermeidbar
  - Ziel: auch langer Code gut lesbar für Menschen
  - kein Ziel: eine Variable einsparen

Beispiel: break vermeiden II

Heinrich Heine Universität Düsseldorf

NoBreak.java — Java

```java
public class NoBreak {
    public static void main(String[] args) {
        boolean twentyFourFound = false;
        for(int i = 0; i < args.length && !twentyFourFound; i++) {
            System.out.println(args[i]);
            if(Integer.parseInt(args[i]) == 42) {
                twentyFourFound = true;
            }
        }
    }
}
```

return in main-Methode

Heinrich Heine Universität Düsseldorf

- `return;` in main-Methode: beendet Ablauf sofort an dieser Stelle
- kann Code übersichtlicher gestalten als Verwendung von `else`

Return.java — Java

```java
public class Return {
    public static void main(String[] args) {
        int number = Integer.parseInt(args[0]);
        if(number < 0) {
            System.out.println("ERROR: Wurzel aus negativer Zahl");
            return;
        }
        System.out.println(Math.sqrt(number));
    }
}
```

- später: Verwendung von `return` an anderen Stellen

return vs. else

Heinrich Heine Universität Düsseldorf

```java
int number = Integer.parseInt(args[0]);
if(number < 0) {
    System.out.println("ERROR: Wurzel aus negativer Zahl");
    return;
}
System.out.println(Math.sqrt(number));
```

Das gleiche Verhalten mit `else`:

```java
int number = Integer.parseInt(args[0]);
if(number < 0) {
    System.out.println("ERROR: Wurzel aus negativer Zahl");
} else {
    System.out.println(Math.sqrt(number));
}
```

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- switch-Verzweigungen **benutzen**, um Code abhängig von Variablenwerten auszuführen
- abgekürzte arithmetische Ausdrücke **benutzen**, um Variablenwerte zu ändern
- for-Schleifen **benutzen**, um Code abhängig von einer Bedingung zu wiederholen
- die Anzahl übergebener Kommandozeilenargumente **abfragen**
- return **benutzen**, um die main-Methode bei ungültigen Argumenten vorzeitig zu beenden

Wichtige Begriffe

switch	case	break
default	i++	i*=2
for	args.length	return

Übungsblatt

- Korrekturfeedback
- Reihenfolge umkehren
- Minimum & Maximum
- Schachbrett
- pq-Formel
- BugHunt: Newtonverfahren