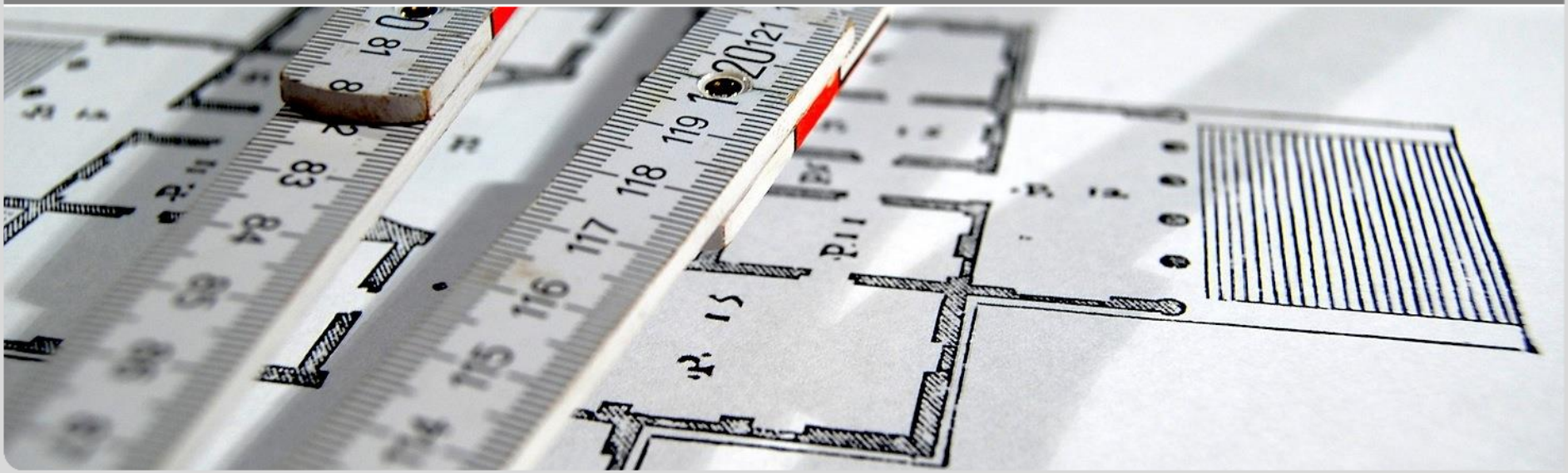
Karlsruher Institut für Technologie

Programmieren-Tutorium Nr. 10

3. Tutorium | Jonas Ludwig

Fallunterscheidungen, Schleifen und Objektorientierung

Architecture-driven Requirements Engineering – Institut für Programmstrukturen und Datenorganisation – Fakultät für Informatik



Was machen wir heute?

- Kontrollstrukturen
  - Fallunterscheidungen
  - Schleifen
- Aufgabe zu Objektorientierung

if-Anweisung

- Realisierung der Fallunterscheidung in Java
  - `if (Bedingung) { Anweisungen }`
  - `if (Bedingung) { Anweisungen`$_1$ `} else { Anweisungen`$_2$ `}`

- Beispiele:

```
int a;
int b;

if (a >= b) {
        a = b;
} else {
        b = a;
}
```

```
int x;
int y;

if (x != y) {
        x = x % y;
}
```

Geschachtelte if-Anweisung

- Man kann die if-Anweisung auch mehrfach schachteln
  - `if (Bedingung) { Anweisungen`$_1$ `}`
    `[else if (Bedingung`$_n$`) { Anweisungen`$_n$ `}]`^{n}
  - `if (Bedingung) { Anweisungen`$_1$ `}`
    `[else if (Bedingung`$_n$`) { Anweisungen`$_n$ `}]`^{n}
    `else { Anweisungen`$_{n+1}$ `}`
- Nach Möglichkeit häufigsten Fälle zuerst
- In jedem else-Zweig gilt die Negation aller vorangegangener Bedingungen
- Achtung mit der Reihenfolge!

Geschachtelte if-Anweisung Beispiel

```
int punkte = 42;
int note = 0;

if (punkte >= 87) {
        note = 1;
} else if (punkte >= 75) {
        note = 2;
} else if (punkte >= 63) {
        note = 3;
} else if (punkte >= 51) {
        note = 4;
} else {
        note = 5;
}
```

Aufgabe 1

```
class SomeIf {
        public static void main(String[] a) {

                int monat = 12;

                if (monat == 12) {
                        System.out.print("12");
                } else if (monat / 2 == 6) {
                        System.out.print("6");
                } else {
                        System.out.print("3");
                }
        }
}
```

- Ausgabe?
  - 12

Aufgabe 2

- Implementieren Sie ein kleines Programm, welches für einen gegebene ganzen Zahl ermittelt, ob sie kleiner, größer oder gleich der Zahl Null ist. Dabei soll die zu testende Zahl mit dem Wert 42 initialisiert werden. Das Ergebnis soll anschließend mit der Methode System.out.println() auf das Terminal ausgegeben werden.

```
public class MoreOrLess {
   public static void main(String[] args) {
      int number = 42;

      // Hier Implementierung vervollständigen

   }
}
```

Aufgabe 2 – Lösung

```
public class MoreOrLess {
    public static void main(String[] args) {
        int number = 42;

        if (number > 0) {
            System.out.println("More than zero");
        } else if (number < 0) {
            System.out.println("Less than zero");
        } else {
            System.out.println("Zero");
        }
    }
}
```

Aufgabe 3

- Implementieren Sie ein kleines Programm, welches in der ersten Zeile den Wert einer int-Variable ausgibt
- In der zweiten Zeile, soll nun „Odd Number“ ausgegeben werden, wenn die Variable ungerade oder „Even Number“, wenn die Variable gerade ist
- Das Programm muss vor dem Beenden in der dritten Zeile immer „BYE!“ ausgeben

Aufgabe 3 – Lösung

```
public class CheckOddEven {
	public static void main(String[] args) {
		int number = 42;
			System.out.println("The number is: " +
number);

		if (number % 2 == 0 ) {
			System.out.println("Even Number");
		} else {
			System.out.println("Odd Number");
		}
		System.out.println("BYE!");
	}
}
```

switch-Anweisung

```
switch (Ausdruck) {
    [case Wert_n : Anweisungen_n]^n
    default : Anweisungen
}
```

- Der **Ausdruck** ist dabei ein primitiver Datentyp, Enum oder ein String
  - Bei Enum am besten immer eine switch-Anweisung verwenden
- Die **Werte** nach case müssen konstant sein
- Die **break**-Anweisung veranlasst das sofortige Verlassen der gesamten switch-Anweisung

switch-Anweisung Beispiel

```
int monat = 11;
int tage = 0;

switch (monat) {
        case 1: tage = 31; break;
        case 2: tage = 28; break;
        case 3: tage = 31; break;
        case 4: tage = 30; break;
        case 5: tage = 31; break;
        case 6: tage = 30; break;
        case 7: tage = 31; break;
        case 8: tage = 31; break;
        case 9: tage = 30; break;
        case 10: tage = 31; break;
        case 11: tage = 30; break;
        case 12: tage = 31; break;
}
```

Aufgabe 4

```
int tagNummer = 5;
String tagName = ”“;

switch (tagNummer) {
        case 1: tagName = ”Montag”;
        case 2: tagName = ”Dienstag”;
        case 3: tagName = ”Mittwoch”;
        case 4: tagName = ”Donnerstag”;
        case 5: tagName = ”Freitag”;
        case 6: tagName = ”Samstag”;
        case 7: tagName = ”Sonntag”;
        default: tagName = ”unbekannt”;
}
```

- Welchen Wert hat tagName am Ende?
  - “unbekannt” (Warum? Es fehlen die break Befehle)

Aufgabe 5

- Implementieren Sie ein kleines Programm, welches für einen gegebenen Wochentag ermittelt, ob es sich um einen Arbeitstag handelt oder nicht. Dabei soll der Wochentag mit dem Wert Day.FRIDAY initialisiert werden. Das Ergebnis soll anschließend mit der Methode System.out.println() auf das Terminal ausgegeben werden.

```
public class Workday {

   enum Day {
      SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
   }

   public static void main(String[] args) {
            Day day = Day.FRIDAY;

      // Hier Implementierung vervollständigen

   }
}
```

Aufgabe 5 – Lösung – So besser nicht?

```
Day currentDay = Day.FRIDAY;
switch (currentDay) {
   case SUNDAY:
      System.out.println("No Workday");
      break;
   case MONDAY:
      System.out.println("Workday");
      break;
   case TUESDAY:
      System.out.println("Workday");
      break;
   case WEDNESDAY:
      System.out.println("Workday");
      break;
   case THURSDAY:
      System.out.println("Workday");
      break;
   case FRIDAY:
      System.out.println("Workday");
      break;
   case SATURDAY:
      System.out.println("Workday");
      break;
}
```

Aufgabe 5 – Lösung – Besser so?

```
public class Workday {

   enum Day {
      SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
   }

   public static void main(String[] args) {

      Day currentDay = Day.FRIDAY;

      switch (currentDay) {
         case SUNDAY:
            System.out.println("No Workday");
            break;
         default:
            System.out.println("Workday");
            break;
      }
   }
}
```

for-Schleife

- Die for-Schleife ist eine Zählschleife
  - ```
    for (Initialisierung; Bedingung; Schritt) {
        Anweisungen
    }
    ```
- **Initialisierung** ist die Anweisung, welche einmalig am Anfang ausgeführt wird
- Vor jedem Schleifendurchlauf wird geprüft ob **Bedingung** wahr ist
  - Verlassen der der Schelfe, wenn die Bedingung nicht mehr wahr ist
- **Schritt** ist die Anweisung, welche nach jedem Durchlauf ausgeführt wird
    ```

for-Schleife Beispiel

```
class SomeLoop {
        public static void main(String[] args) {
                for (int i = 0; i <= 10; i++) {
                        for (int j = 0; j <= 10; j++) {
                                for (int k = 0; k <= 10; k++) {
                                        System.out.println("i=" + i);
                                        System.out.println("j=" + j);
                                        System.out.println("k=" + k);
                                }
                        }
                }
        }
}
```

- Die Buchstaben `i`, `j` und `k` werden in der Regel als Schleifenvariablen verwendet

for-Schleife so besser nicht!

```
for (int i = 0, j = 10; i <= 10; i++, j--) {
      //Anweisungen
}
```

```
int i = 1;
for (; i < 10; i++) {
      //Anweisungen
}
```

```
int j = 2;
for (;;) {
      j++;
}
```

- for-Schleifen sollten immer dann benutzt werden, wenn eine Variable um eine konstante Größe erhöht wird
- Wenn möglich nur eine Schleifenvariable
- Initialisierung und Veränderung der Schleifenvariable nur im Schleifenkopf

Aufgabe 6

- Implementieren Sie ein kleines Programm, welches für einen gegebene ganze Zahl auf eine andere gegebene ganze Zahl herunterzählt. Bei jedem Schritt des Countdowns, soll der aktuelle Zahlenwert ausgegeben werten.

- Initialisieren Sie die erste Zahl des Countdowns mit 10 und die letzte Zahl mit 0.

```
public class Countdown {
	public static void main(String[] args) {
		int from = 10;
		int to = 0;
		// Hier Implementierung vervollständigen
	}
}
```

Aufgabe 6 – Lösung

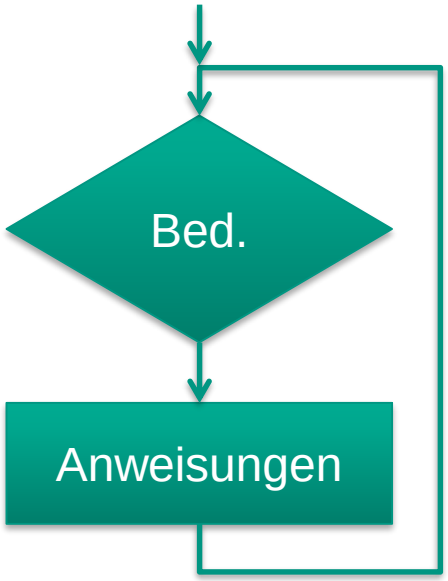
```java
public class Countdown {
    public static void main(String[] args) {
        int from = 10;
        int to = 0;

        for (int i = from; i >= to; i--) {
            System.out.println(i);
        }
    }
}
```

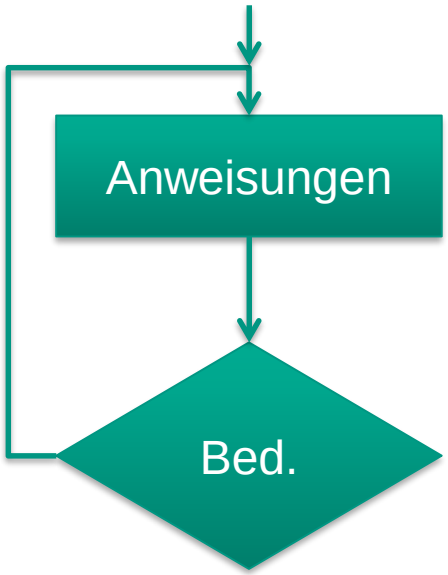
Die While- und Do-While-Schleife

- Die **While**- beziehungsweise **Do-While**-Schleifen realisiert eine sich wiederholende Anweisung
  - `while (Bedingung) {Anweisungen}`
  - `do {Anweisungen} while (Bedingung);`

While:



Do-While:



- Do-While-Schleifen können gefährlich werden, da sie mindestens einmal durchlaufen werden, bevor die Abbruchbedingung überprüft wird

while-Schleife Beispiel

```
class SomeWhile {
        public static void main(String[] args) {
                int a = 5;
                int b = 100;

                int sum = 0;
                int i = a;
                while (i <= b) {
                        sum = sum + i;
                        i = i + 1;
                }
                System.out.println("Summe = " + sum);
        }
}
```

$$\sum_{i=a}^{b} i$$

do-while-Schleife Beispiel

```
class SomeDoWhile {
	public static void main(String[] args) {
		int age;
		do {
			//Enter your age!
		} while (age < 0 || age > 120);
		System.out.println("Alter = " + age);
	}
}
```

Sprunganweisungen

- Manchmal möchte man eine Schleife verlassen, bevor alle Schleifendurchläufe abgearbeitet wurden
- **`break`** veranlasst das sofortige verlassen der Schleife
  - Übergibt die Kontrolle an die erste Anweisung außerhalb der Schleife
- **`continue`** setzt die Schleife mit dem nächsten Durchgang fort
  - Der Rest dieses einen Durchgangs wird übersprungen
- Um Lesbarkeit und Nachvollziehbarkeit des Codes zu gewährleisten empfehlt sich, solche Sprunganweisungen sehr sparsam einzusetzen!

Sprunganweisungen Beispiel

```
class Sprung {
        public static void main (String[] args) {
                int i = 0;
                int j = 0;

                while(i < 10) {
                        i++;
                        System.out.println("1." + i);
                        if(i == 5) {
                                break;
                        }
                }
                while(j < 10) {
                        j++;
                        if(j <= 5) {
                                continue;
                        }
                        System.out.println("2." + j);
                }
        }
}
```

Objektorientierung - Wiederholung

- Klassen (abstrakte Beschreibung, Modellierung)
  - Dateiname = Klassenname.java
  - Stichwort `class`

```
class Point {
        double x;
        double y;
}
```

```
class Line {
    Point p1;
    Point p2;
}
```

- Objektinstanzen (konkrete Vertreter einer Klasse)
  - Stichwort `new`

```
Point a = new Point();
```

- Punktoperator

```
a.x = 5.0;
```

Aufgabe 7

Modellieren Sie die Klasse Tutorial. Erstellen Sie dafür als Vorbereitung zuerst die Klasse Student.

Klasse Student

Modellieren Sie Studenten in der Klasse Student. Jeder Student hat einen Vornamen, einen Nachnamen, ein Geschlecht (männlich/weiblich/andere), ein Alter und eine siebenstellige Matrikelnummer.

Klasse Tutorium

Modellieren Sie Tutorien in der Klasse Tutorial. Jedes Tutorium hat einen Tutor, der ein Student ist, sowie mehrere Tutanden, die ebenfalls Studenten sind. Außerdem hat jedes Tutorium eine zweistellige Nummer und einen Namen, der als Zeichenkette gegeben ist.

Wie modelliert man “mehrere Tutanden”?

=> Benutzen eines Arrays (mehr dazu in 2 Wochen):

```
Student[] tutees;
```

Lösung 7 – Klasse Student

```java
/**
 * This class models a student.
 * @author Jonas Ludwig
 * @version 1.0
 */
class Student {
        /** The student's first name */
        String firstName;

        /** The student's last name */
        String lastName;

        /** local enum provides the possible genders*/
        enum Gender {MALE, FEMALE, OTHER};

        /** The student's gender*/
        Gender gender;

        /** The student's age*/
        int age;

        /** The student's matriculation number*/
        int number;
}
```

Lösung 7 – Klasse Tutorial

```
/**
 * This class models a tutorial.
 * @author Jonas Ludwig
 * @version 1.0
 */
class Tutorial {
        /** The tutorial's tutor */
        Student tutor;

        /** The tutorial's tutees */
        Student[] tutees;

        /** The tutorial's number */
        short number;

        /** The tutorial's name */
        String name;
}
```

Aufgabe – Primfaktorzerlegung

- Eine Primzahl ist eine natürliche Zahl n größer als 1, die nur durch sich selbst und durch 1 ganzzahlig teilbar ist.
- Unter einer Primfaktorzerlegung versteht man die Darstellung einer natürlichen Zahl n > 1 als Produkt aus Primzahlen. Diese Primzahlen werden Primfaktoren der Zahl n genannt. Die Primfaktorzerlegung einer natürlichen Zahl n ist dabei eindeutig. So ist beispielsweise die Primfaktorzerlegung der Zahl 262395 = 3 * 3 * 5 * 7 * 7 * 7 * 17
- Schreiben Sie eine kleines Programm, welches die Primfaktor-zerlegung der Zahl 42 direkt auf der Kommandozeile ausgibt.

Aufgabe – Vorgehensweise

1. Die Suche nach einem geeigneten Primfaktor beginnt am besten bei der kleinsten Primzahl 2
2. Primzahl, welche die Zahl teilt, suchen
3. Primfaktor suchen (wie oft die Primzahl, die Zahl teilt)
4. Primzahl mit ihrem Primfaktor ausgeben
5. Geteilte Zahl durch neu gefundene Primzahl mal Primfaktor teilen
6. Solange weiter machen, bis es sich bei der (geteilten) Zahl um eine Primzahl handelt

Aufgabe – Lösung

```
public class IntegerFactorization {
   public static void main(String[] args) {

      int number = 42;

      for (int i = 2; i <= number; i++) {
         int count = 0;
         while ((number % i) == 0) {
            number /= i;    // number = number / i;
            count++;       // count = count + 1;
         }
         if (count != 0) {
            System.out.println(i + "^" + count);
         }
      }
   }
}
```

Fragen?

Was machen wir nächste Woche?

- Methoden
- Konstruktoren

Vielen Danke für eure Aufmerksamkeit!