

Kontrollstrukturen (Scanner)

Programmieren Tutorium Nr.17

Aleksandr Zakharov | 19. November 2025

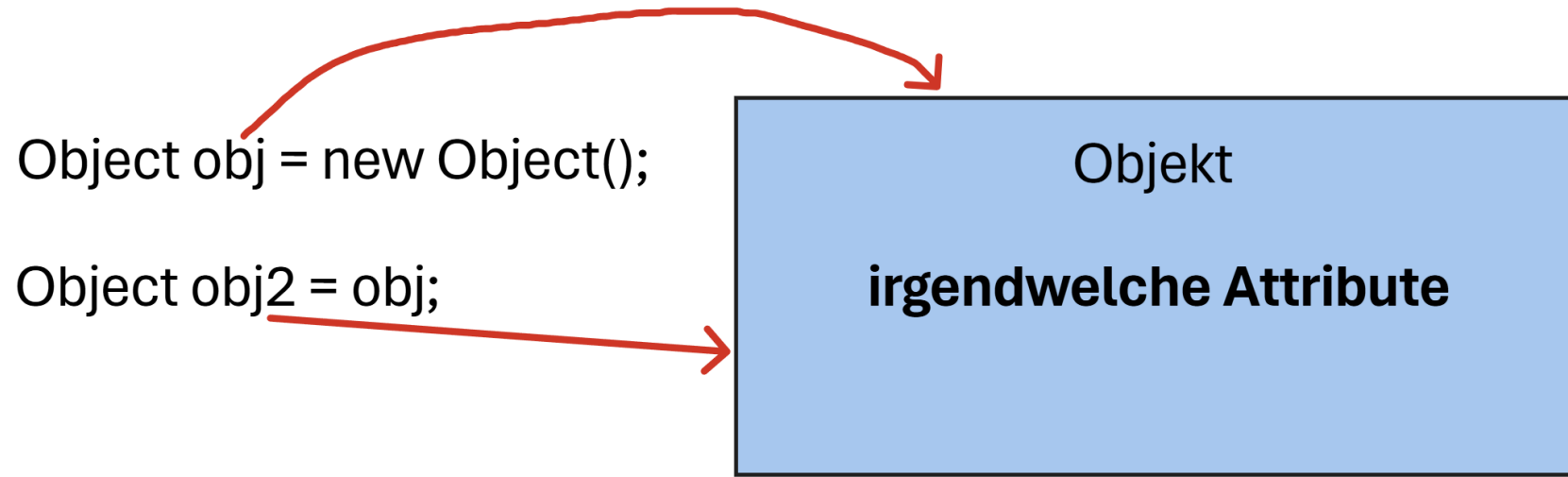
Wiederholung I

Letztes Mal:

- Datentypen (Mengen gleichartiger Objekte)
 - Primitive Datentypen und Referenzdatentypen
- Operatoren (+, *, =)
 - Was ist Unterschied zwischen = und ==
- Was ist der Unterschied zwischen der Deklaration und Initialisierung einer Variable?
 - Eine Deklaration legt den Typ und Namen einer Variable fest, bei der Initialisierung wird gleichzeitig noch ein Wert festgelegt

Wiederholung II

Ein Bildchen zur Vorstellung von Referenzdatentypen:



`new Object()` erzeugt diesen blauen Block, aber `obj` hat nur die Referenz auf diesen Block. Deswegen werden die Änderungen im Block und nicht im `obj` gespeichert.

Wiederholung III

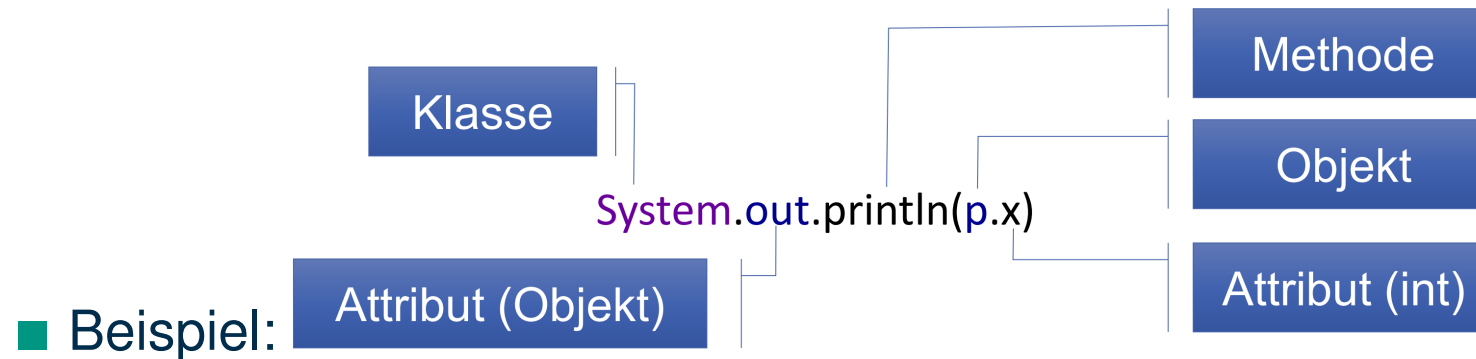
Wichtig für heute: Gleichheitszeichen

- =
 - Zuweisung eines Wertes (ggfs. einer Variablen) zu einer (anderen) Variablen
- ==
 - Überprüft Objekt Identität (Gleichheit der Referenzen)
 - Überprüfung primitiver Datentypen
- === (strict equality)
 - Überprüft auch ob Variablen vom gleichen Typ sind
- .equals()
 - Überprüft ob Objekte die gleichen Attribute haben (oder wie equals() definiert ist)
 - Zum überprüfen von Objekten und Strings

Ergänzung zum letzten Mal

Punktoperator

- Erlaubt auf Objekten den Zugriff auf die Methoden oder Zustände
- Er dient dazu, auf Elemente von Klassen oder Datenstrukturen zuzugreifen
- Er steht zwischen einem Ausdruck, der eine Referenz liefert, und der Objekteigenschaft



Ergänzung zum letzten Mal

static-Schlüsselwort

- `static`-markierte Attribute oder Methoden gehören nicht zu Objekt, sondern zu Klasse
- Statische Variablen/Methoden sind auch dann verfügbar, wenn noch keine Instanz der Klasse erzeugt wurde
- Statische Variablen/Methoden können über den Klassennamen aufgerufen werden
- Deklaration durch das Schlüsselwort: **`static`**

Wiederholung
○○○○●○

Verzweigungen
○○○○○○○○

Schleifen
○○○○○○○○○○○○

Scanner
○○○○○

Heute

Die heutigen Themen sind:

■ Verzweigungen

■ Schleifen

Wiederholung
○○○○●

Verzweigungen
○○○○○○○○

Schleifen
○○○○○○○○○○○○

Scanner
○○○○○

Verzweigungen

Wiederholung
○○○○○

Verzweigungen
●○○○○○○

Schleifen
○○○○○○○○○○

Scanner
○○○○○

Motivation

Wir können im Programm nicht immer im voraus wissen, was bspw. passiert oder welche Parameter in die Methode übergeben werden

Oder sollen wir nicht wissen, da es eine I/O-Operation implementiert werden soll
→ es wurde ein Weg erstellt, der diese benötigte "Abweichungen" von dem festgestellten Verhalten ermöglicht

Wiederholung
○○○○○

Verzweigungen
○●○○○○○

Schleifen
○○○○○○○○○○○

Scanner
○○○○○

Verzweigungen

■ Fallunterscheidungen

- if
- if-else
- if-else-Kurzform
- switch

Wiederholung
○○○○○

Verzweigungen
○○●○○○○

Schleifen
○○○○○○○○○○○

Scanner
○○○○○

if-Verzweigung

In natürlicher Sprache:

- **Wenn** die Bedingung wahr ist, *tue etwas*,
ansonsten *tue etwas anderes*

- **Wenn** die Bedingung A wahr ist, *tue A*
Wenn die Bedingung A nicht wahr,
aber Bedingung B wahr ist, *tue B*
...
ansonsten *tue etwas anderes*

Schema

```
1 if (condition) {  
2     //do something  
3 } else {  
4     //otherwise action  
5 }
```

Schema - Mehr Cases

```
1 if (conditionA) {  
2     // do something (A)  
3 } else if (conditionB) {  
4     // do something else (B)  
5 ...  
6 } else {  
7     // otherwise action  
8 }
```

Wiederholung
○○○○○

Verzweigungen
○○●○○○

Schleifen
○○○○○○○○○○

Scanner
○○○○○

Verzweigung: if-else-Kurzform

if-else Original

```
if (condition) {  
    ifTrue();  
} else {  
    ifFalse();  
}
```

Schema

condition ? ifTrue() : ifFalse();

if-else Original

```
if (condition) {  
    ifTrue();  
} else {  
    ifFalse();  
}
```

Schema

condition ? ifTrue() : ifFalse();

Wiederholung
○○○○○

Verzweigungen
○○○●○○○

Schleifen
○○○○○○○○○○○

Scanner
○○○○○

if-else Original

12/30

19.11.2025

Aleks Zakharov: 3. Tutorium - Kontrollstrukturen

Schema

Verzweigung: switch

- In der Klammer dürfen nur Konstanten stehen
- Ohne **break** werden die restlichen Zeilen weiter bearbeitet
- **default** funktioniert wie **else**
 - An sich optional, gehören aber trotzdem zu gutem Coding-Style
 - Wenn unmöglich zu treffen, leer lassen und einen Kommentar schreiben

Schema:

```
1 switch (variable) {  
2     case (caseA):  
3         // do something  
4         break; // <- optional  
5     case (caseB):  
6         // do something else  
7         break; // <- optional  
8     ...  
9     default: // <- optional  
10            // default action  
11 }
```

Andere Schreibweise für switch

- Switch-Verzweigung kann auch benutzt werden, um:
 - Die Variablenwerte zuzuweisen
 - Die return-Werte zurückzugeben

```
1 enum Color {
2     RED, WHITE, BLACK, YELLOW, BLUE
3 }
4
5 class Main {
6     Color mainColor = Color.BLACK;
7
8
9     public int foo(){
10         String str = switch (mainColor) {
11             case RED -> "yellow";
12             case BLUE -> "white";
13             case BLACK -> "black";
14             case WHITE -> "blue";
15             case YELLOW -> "red";
16         };
17
18         return switch (mainColor) {
19             case RED -> 0;
20             case BLUE -> 1;
21             case BLACK -> 2;
22             case WHITE -> 3;
23             case YELLOW -> 4;
24         };
25     }
26 }
27 }
```

Wiederholung
○○○○○

Verzweigungen
○○○○○●○

Schleifen
○○○○○○○○○○

Scanner
○○○○○

Übung: Verzweigungen

→ IDE

Aufgaben findet ihr im Ordner "Tutorium 3"

Wiederholung
○○○○○

Verzweigungen
○○○○○○●

Schleifen
○○○○○○○○○○○

Scanner
○○○○○

Schleifen

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
●○○○○○○○○○

Scanner
○○○○○

Schleifen

- Motivation: häufig soll man wiederholt einige Codestellen ausführen
- Schleifen erlauben dies und haben einige Arten:
 - while
 - do-while
 - for
 - for-each

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○●○○○○○○○○○

Scanner
○○○○○

Schleife: while

- **So lange** die Bedingung wahr ist, *tue dies*
- Die Bedingung wird zuerst geprüft, erst danach werden die Anweisungen ausgeführt
- Wenn die Bedingung bei der **Überprüfung** falsch ist, bricht die Schleife ab

Schema

```
1 while (condition) {  
2     //do something  
3 }
```

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○●○○○○○○○

Scanner
○○○○○

Schleife: do-while

- **So lange** die Bedingung wahr ist, *tue dies*, aber der erste Durchlauf passiert immer
- Die Anweisungen werden ausgeführt und dann wird die Bedingung überprüft, ob man weiter iterieren darf
- Wenn die Bedingung bei der **Überprüfung** falsch ist, bricht die Schleife wie vorher ab

Schema

```
1 do {  
2     //repeat something  
3 } while (condition);
```

Wiederholung
○○○○○

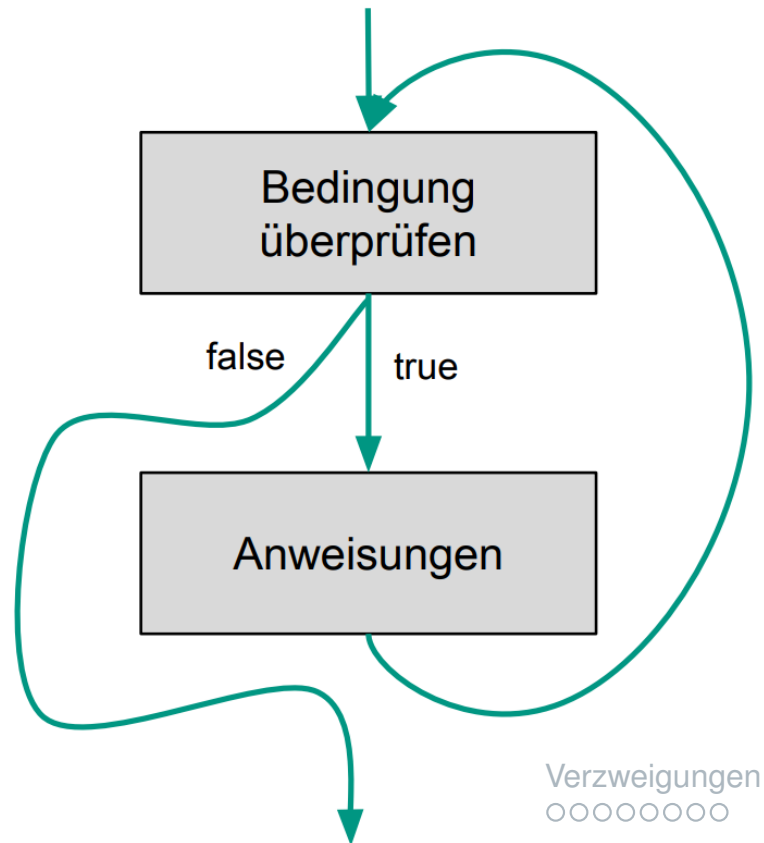
Verzweigungen
○○○○○○○

Schleifen
○○●○○○○○○

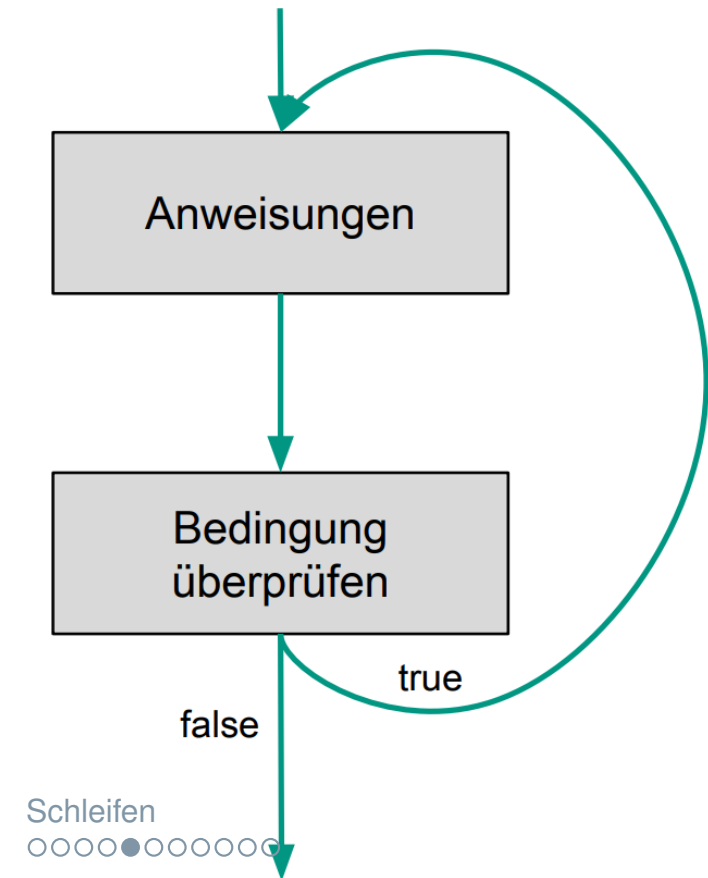
Scanner
○○○○○

Unterschiede zwischen while und do-while

while



do-while



Schleife: for

- Kann alles, was auch `while` kann, ist aber übersichtlicher (guter Stil)

Schema:

```
1 for (variable; condition; step) {  
2     //something to repeat  
3 }
```

Besipiel:

```
1 for (int i = 0; i < 10; i++) {  
2     String output = "Cycle iteration: " + i;  
3     System.out.println(output);  
4 }
```

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○●○○○○

Scanner
○○○○○

for-each-Schleife

- Besondere for-Schleife
- Benötigt iterierbare Datenstrukture
- Mehr dazu in der 5. Vorlesung

Schema:

```
1 for (variable : Datastructure) {  
2     //do something  
3 }
```

Beispiel:

```
1 void foo() {  
2     int[] array = {1, 2, 3, 4, 5};  
3     for (int number : array) {  
4         System.out.println("It's element number " + number);  
5     }  
6 }
```

break und continue

- **break** beendet die Schleife
 - Dies geschieht sofort
- **continue** springt in die nächste Schleifeniteration
 - Der Rest des Durchlaufs / der Iteration wird übersprungen
- Klassische Abbruchbedingungen » **break**

```
1 for (int i = 0; i < 10; i += 2) {  
2     if (i % 5 == 0) {  
3         i++;  
4         continue; //skip this iteration  
5     }  
6     if (i % 8 == 0) {  
7         break; //end of loop  
8     }  
9  
10    ... //something is happening here  
11 }
```

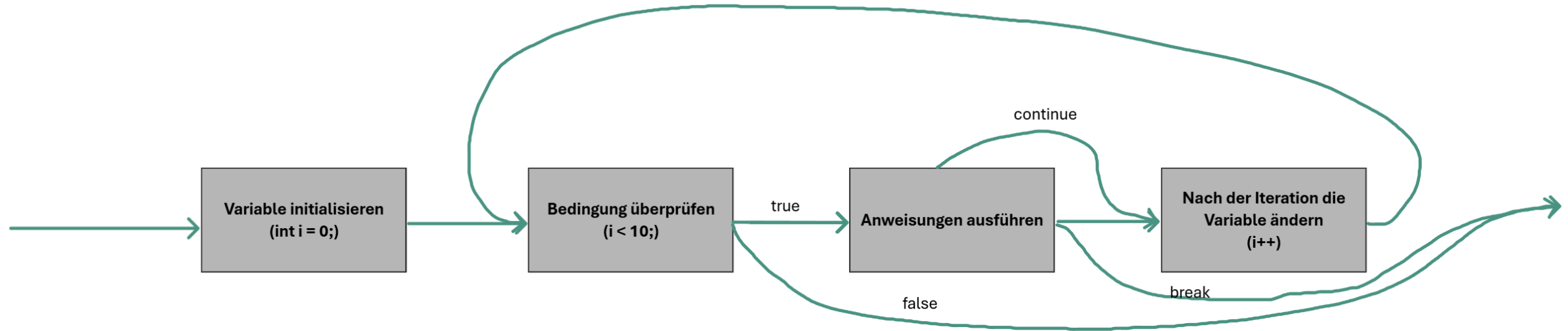
Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○●○○○

Scanner
○○○○○

Detaillierter zu for



Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○○●○○

Scanner
○○○○○

Übung: Schleifen

→ IDE

Aufgaben findet ihr im Ordner "Tutorium 3"

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○○○○●○

Scanner
○○○○○

Kontrollfluss-Anweisungen: Regeln

```
if (10 > 5) {  
    // always true  
}
```

Keine Bedingungen, die Konstant sind!

```
if (a > b) {  
    if (a < b) {  
        // never reachable  
    }  
}
```

Keinen nicht erreichbaren Code.

```
if ((a < b) == true) {  
    //...  
}
```

== true kann weggelassen werden.

```
while (true) {  
    // endless loop  
}
```

Keine Endlosschleifen!

```
while (true) {  
    // do something  
    if (i > 10) {  
        break;  
    }  
}
```

Schleifenabbruch nicht verschachteln!

```
boolean result;  
if (i >= 0) {  
    result = true;  
} else {  
    result = false;  
}
```

if-Verzweigung kann gekürzt werden.

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○○○○●

Scanner
○○○○○

Scanner

- Mit Scannern kann man `InputStreams` einlesen
- `Scanner.nextLine()` liest genau eine Zeile
 - Es gibt auch weitere Methoden, die aber oft fehleranfällig sind
- Nur einen Scanner pro Applikation verwenden
- Man kann den Scanner auch nach der Nutzung schließen, ist es aber schon nicht obligatorisch
 - Achtung: Nach dem Schließen eines Scanners auf einem `InputStream` sind alle Scanner auf diesem Stream geschlossen!

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○○○○○○

Scanner
●○○○○

Scanner

Beispiel:

```
1 import java.util.Scanner;
2 public class ScannerExample {
3     public static void main(String[] args) {
4         // create new scanner (reading from system.in)
5         Scanner scanner = new Scanner(System.in);
6         // read one line
7         String line = scanner.nextLine();
8         System.out.println(line);
9         // close scanner (already optional)
10        scanner.close();
11    }
12 }
```

Wiederholung
○○○○○○

Verzweigungen
○○○○○○○○

Schleifen
○○○○○○○○○○○○

Scanner
○●○○○

Übung: Kontrollfluss-Anweisungen

→ IDE

Aufgaben findet ihr im Ordner "Tutorium 3"

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○○○○○○

Scanner
○○●○○

Vielen Dank für eure Aufmerksamkeit :=)

Wiederholung
○○○○○

Verzweigungen
○○○○○○○

Schleifen
○○○○○○○○○○○

Scanner
○○●○

Enums

- Enums können auch erweitert werden:
 - sie können Attribute haben
 - sie können Methoden haben

Beispiel:

```
1 enum Color {
2     RED("FF0000", "255, 0, 0"),
3     WHITE("FFFFFF", "255, 255, 255"),
4     BLACK("000000", "0, 0, 0"),
5     YELLOW("FFFF00", "255, 255, 0"),
6     BLUE("0000FF", "0, 0, 255");
7
8     Color(String hexCode, String rgbCode) {
9         this.hexCode = hexCode;
10        this.rgbCode = rgbCode;
11    }
12
13    public String getHexadecimalCode() {
14        return this.hexCode;
15    }
16
17    public String getRGBCode() {
18        return this.rgbCode;
19    }
20 }
21
22 class Main {
23     public static void main(String[] args) {
24         System.out.println(Color.RED.getHexadecimalCode());
25         System.out.println(Color.YELLOW.getRGBCode());
26     }
27 }
```

Wiederholung
○○○○○○

Verzweigungen
○○○○○○○○

Schleifen
○○○○○○○○○○○○

Scanner
○○○○●