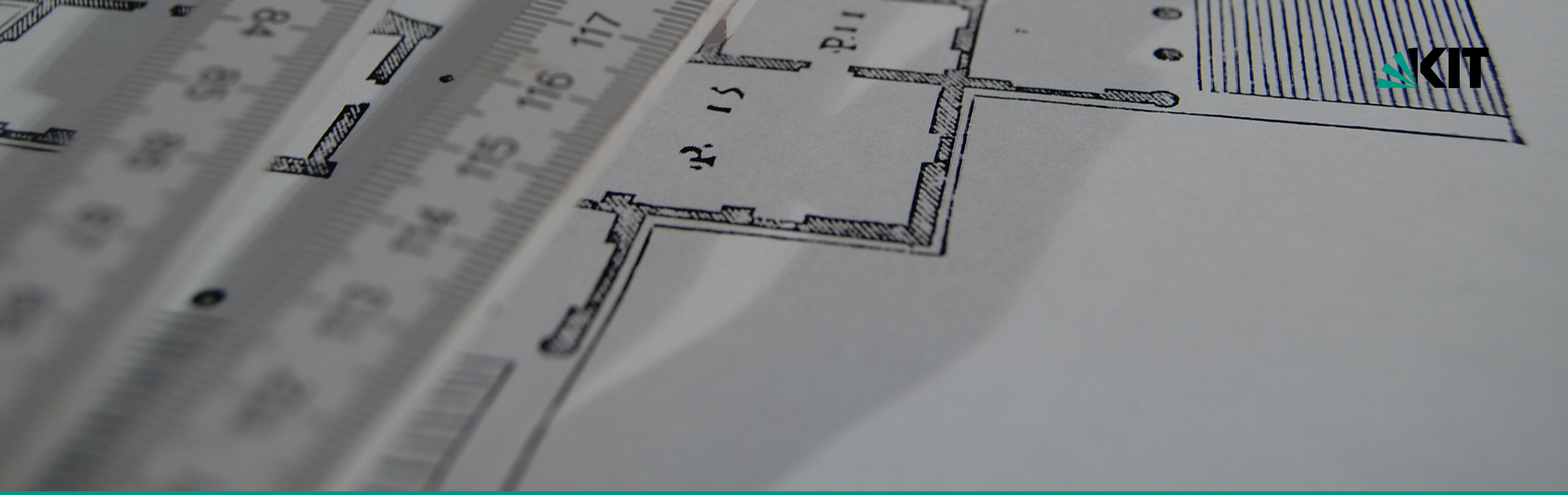


Kontrollstrukturen (Scanner)

Programmieren Tutorium Nr.12

Aleksandr Zakharov | 25. November 2025

Wiederholung I

Letztes Mal:

- Datentypen (Mengen gleichartiger Objekte)

Wiederholung I

Letztes Mal:

- Datentypen (Mengen gleichartiger Objekte)
  - Primitive Datentypen und Referenzdatentypen

Wiederholung I

Letztes Mal:

- Datentypen (Mengen gleichartiger Objekte)
  - Primitive Datentypen und Referenzdatentypen
- Operatoren (+, *, =)

Wiederholung I

Letztes Mal:

- Datentypen (Mengen gleichartiger Objekte)
  - Primitive Datentypen und Referenzdatentypen
- Operatoren (+, *, =)
  - Was ist Unterschied zwischen = und ==

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?

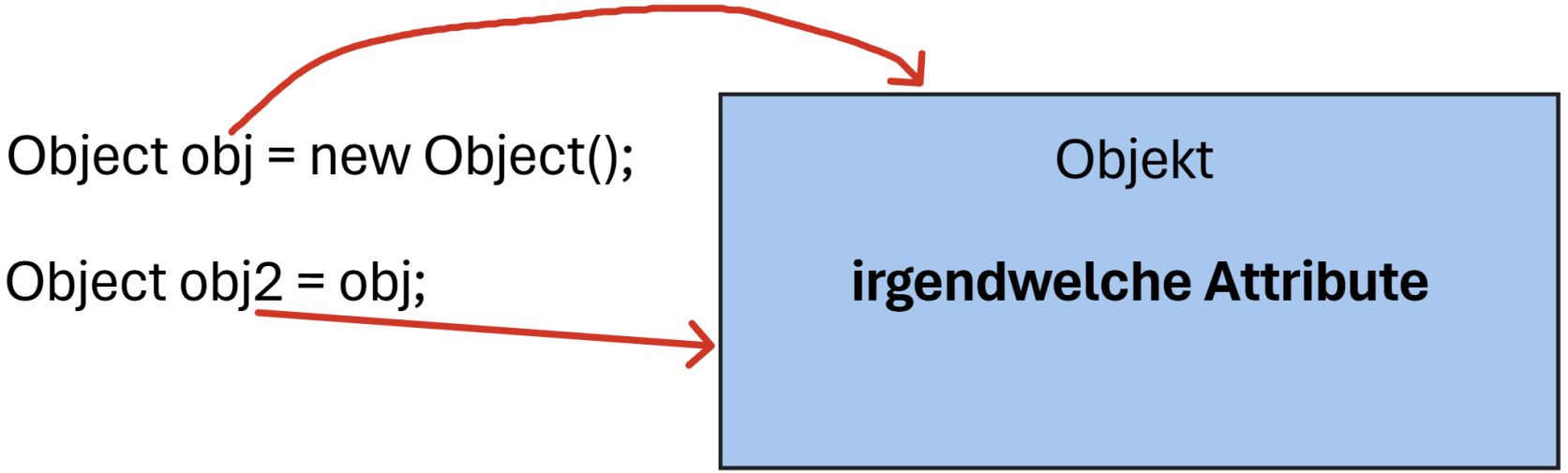
Wiederholung I

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

- =
  - Zusweisung eines Wertes (ggfs. einer Variablen) zu einer (anderen) Variablen

Wiederholung III

Wichtig für heute: Gleichheitszeichen

- =
  - Zusweisung eines Wertes (ggfs. einer Variablen) zu einer (anderen) Variablen
- ==
  - Überprüft Objekt Identität (Gleichheit der Referenzen)
  - Überprüfung primitiver Datentypen

Wiederholung III

Wichtig für heute: Gleichheitszeichen

- =
  - Zusweisung 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

Wiederholung III

Wichtig für heute: Gleichheitszeichen

- =
  - Zusweisung 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

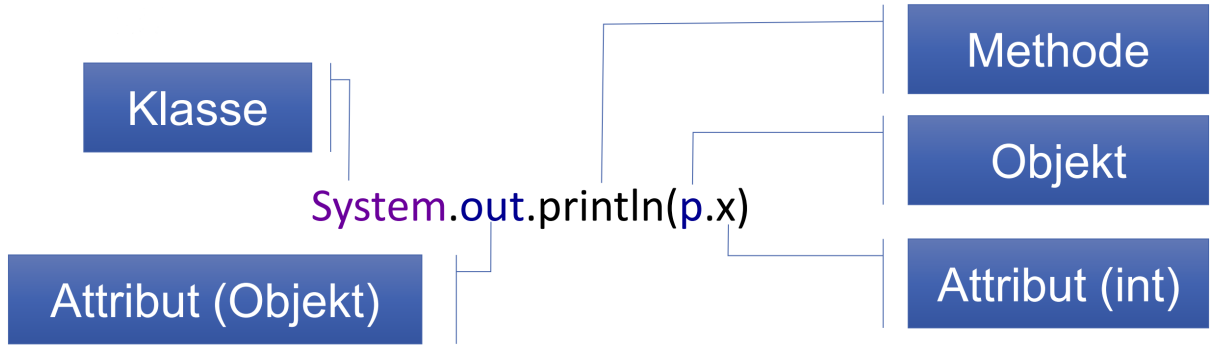
Punktoperator

- 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

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
- Beispiel:



Ergänzung zum letzten Mal

`static`-Schlüsselwor

- `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

Ergänzung zum letzten Mal

`static`-Schlüsselwor

- `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`**

Heute

Die heutigen Themen sind:

- **Verzweigungen**
- **Schleifen**

Verzweigungen

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

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

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
$\rightarrow$ es wurde ein Weg erstellt, der diese benötigte "Abweichungen" von dem festgestellten Verhalten ermöglicht

Verzweigungen

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

if-Verzweigung

In natürlicher Sprache:

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

Schema

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

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 }
```

Verzweigung: if-else-Kurzform

if-else Original

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

Schema

```
condition ? ifTrue() : ifFalse();
```

Verzweigung: if-else-Kurzform

if-else Original

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

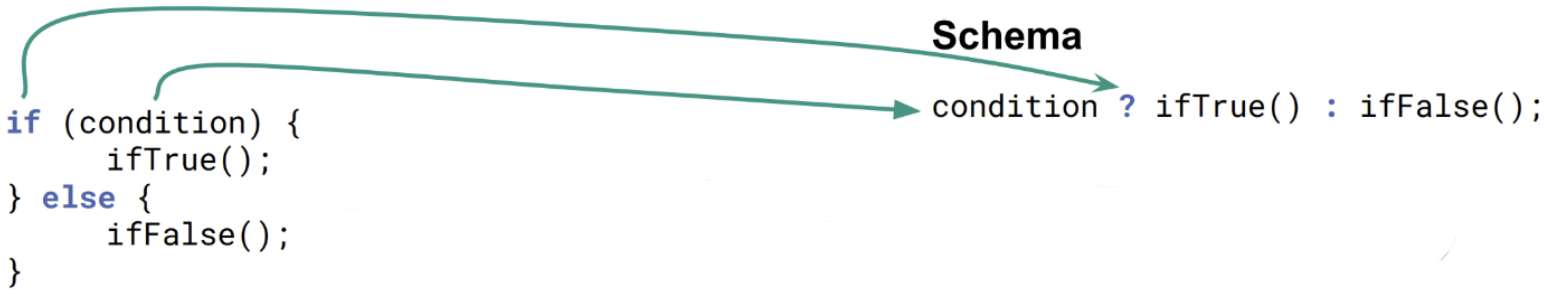
Schema

```
condition ? ifTrue() : ifFalse();
```

Verzweigung: if-else-Kurzform

if-else Original

Schema



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

```java
condition ? ifTrue() : ifFalse();
```

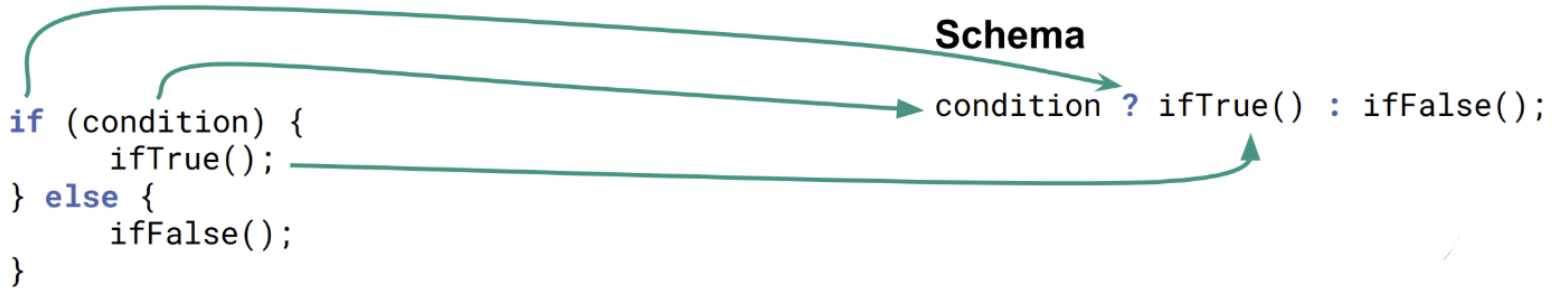
Verzweigung: if-else-Kurzform

if-else Original

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

Schema

```
condition ? ifTrue() : ifFalse();
```



Verzweigung: if-else-Kurzform

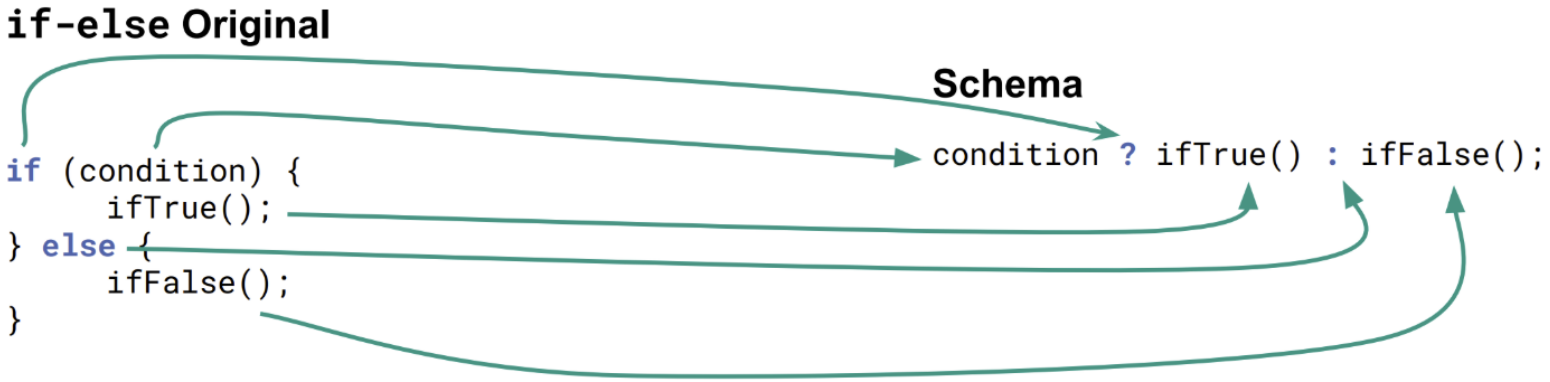
if-else Original

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

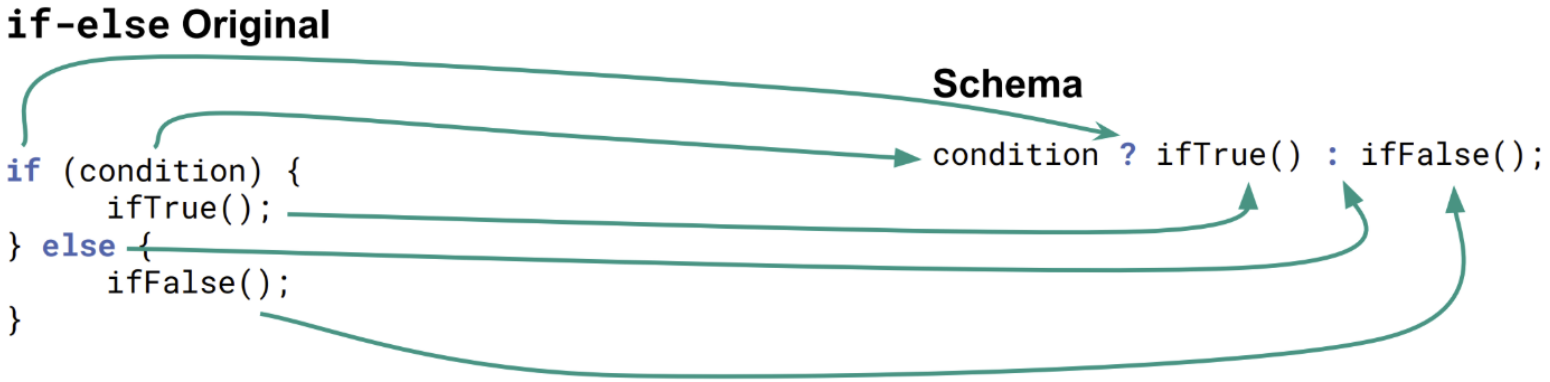
Schema

```
condition ? ifTrue() : ifFalse();
```

Verzweigung: if-else-Kurzform



Verzweigung: if-else-Kurzform



- Nice to know, aber oft nicht wirklich besser lesbar

Beispiel

```java
int number = scanner.nextInt();

String result = "";
result = number >= 0 ? "positive" : "negative";
```

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:

```
switch (variable) {
      case (caseA):
            // do something
            break; // <- optional
      case (caseB):
            // do something else
            break; // <- optional
...
      default: // <- optional
            // default action
}
```

Andere Schreibweise für switch

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

```java
enum Color {
   RED, WHITE, BLACK, YELLOW, BLUE
}

class Main {
   Color mainColor = Color.BLACK;


   public int foo(){
      String str = switch (mainColor) {
         case RED -> "yellow";
         case BLUE -> "white";
         case BLACK -> "black";
         case WHITE -> "blue";
         case YELLOW -> "red";
      };


      return switch (mainColor) {
         case RED -> 0;
         case BLUE -> 1;
         case BLACK -> 2;
         case WHITE -> 3;
         case YELLOW -> 4;
      };
   }
}
```

Übung: Verzweigungen

→ IDE

Aufgaben findet ihr im Ordner "Tutorium 3"

Schleifen

Schleifen

- Motivation: häufig soll man wiederholt einige Codestellen ausführen

Schleifen

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

Schleifen

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

Schleife: while

- **Solange** 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 }
```

Schleife: do-while

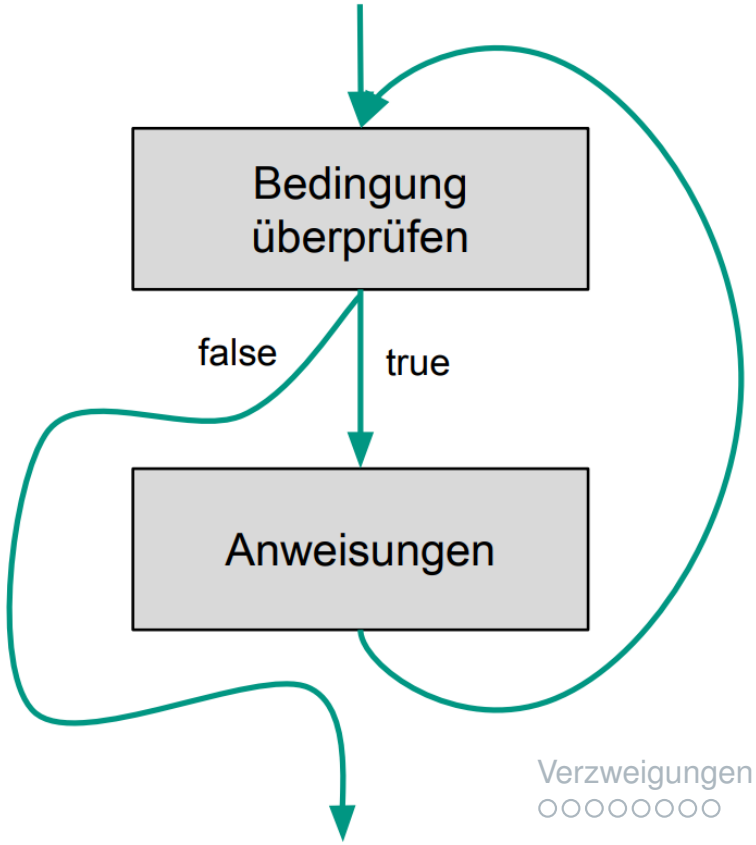
- **Solange** die Bedingung wahr ist, *tue dies*, aber der erste Durchlauf passiert immer
- Die Anweisungen werden ausgeführt und dann wir die Bedingung überprüft, ob man weiter iterieren darf
- Wenn die Bedingung bei der **Überprüfung** falsch ist, bricht die Schleife wie vorher ab

Schema

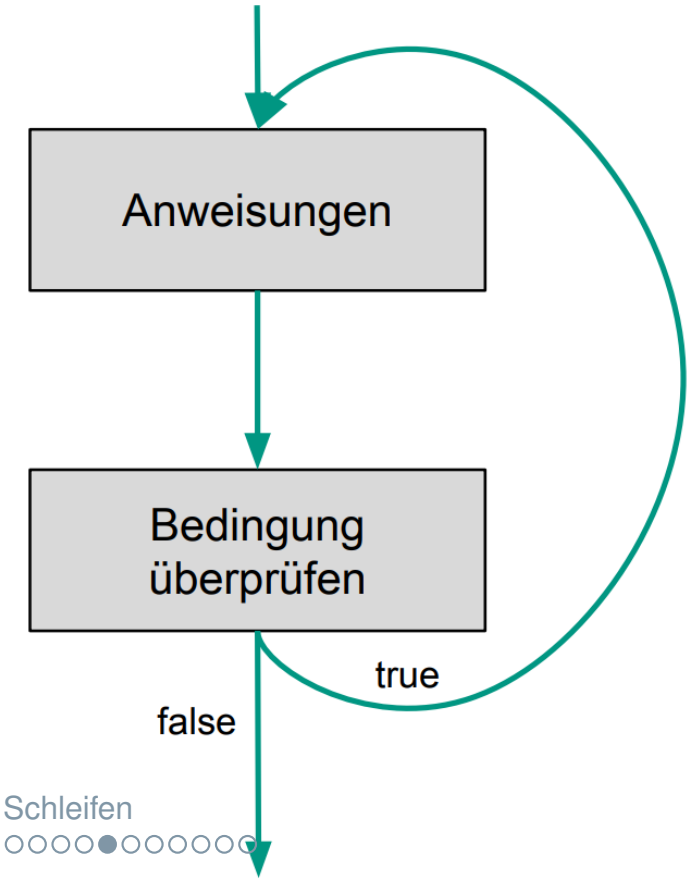
```
1 do {
2       //repeat something
3 } while (condition);
```

Unterschiede zwischen while und do-while

`while`



`do-while`



Schleife: for

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

Schema:

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

Besipiel:

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

for-each-Schleife

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

Schema:

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

Beispiel:

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

`break` und `continue`

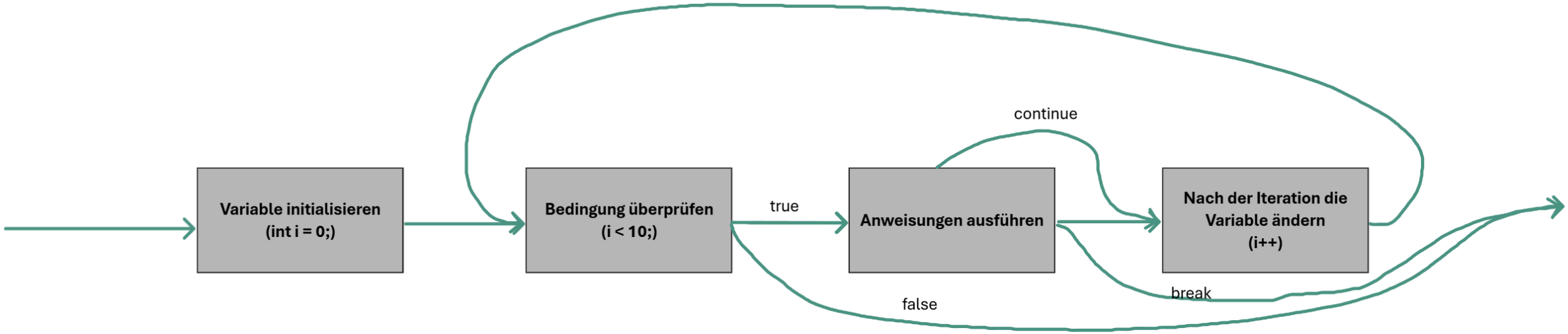
- **`break`** beendet die Schleife
  - Dies geschieht sofort
- **`continue`** springt in die nächste Schleifeniteration
  - Der Rest des Durchlaufs / der Iteration wird übersprungen

```java
for (int i = 0; i < 10; i += 2) {
      if (i % 5 == 0) {
            i++;
            continue; //skip this iteration
      }
      if (i % 8 == 0) {
            break; //end of loop
      }

      ... //something is happening here
}
```

- Klassische Abbruchbedingungen » **`break`**

Detallierter zu for



Übung: Schleifen

→ IDE

Aufgaben findet ihr im Ordner "Tutorium 3"

Kontrollfluss-Anweisungen: Regeln

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

Keine Bedingungen, die Konstant sind!

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

Keinen nicht erreichbaren Code.

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

== true kann weggelassen werden.

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

Keine Endlosschleifen!

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

Schleifenabbruch nicht verschachteln!

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

if-Verzweigung kann gekürzt werden.

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 verweden
- 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!

Scanner

Beispiel:

```java
import java.util.Scanner;
public class ScannerExample {
	public static void main(String[] args) {
		// create new scanner (reading from system.in)
		Scanner scanner = new Scanner(System.in);
		// read one line
		String line = scanner.nextLine();
		System.out.println(line);
		// close scanner (already optional)
		scanner.close();
	}
}
```

Übung: Kontrollfluss-Anweisungen

→ IDE

Aufgaben findet ihr im Ordner "Tutorium 3"

Vielen Dank für eure Aufmerksamkeit :=)

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 }
```