

## Wiederholung

---

Definition:

```
double dividiere(double a, double b) {  
    return a/b;  
}
```

Möglicher Aufruf:

```
double ergebnis = dividiere(5.2, 3.1);
```

```
int x = 0;
while ( x < 5 ) {
    System.out.println(x);
}
```

Wie oft wird der Schleifenrumpf wiederholt? Was gibt das Programm aus?

# Verzweigungen

---

- Ein Programm soll in Abhängigkeit von einer Zufallszahl einen Kreis zeichnen, der entweder schwarz oder weiß gefüllt ist
- Beim Schreiben des Programms kennt man den Wert der Zufallszahl noch nicht, es muss also zur Laufzeit darauf “reagiert” werden
  - Der weitere Programmverlauf hängt davon ab, welchen Wert die Zufallsvariable hat
  - Wie programmieren wir das?
- Lösung:
  - Verzweigungen

Beispiel:

```
if ( x > 5 ) {  
    // Anweisungen  
    // ...  
}
```

**if** Schlüsselwort, das eine  
Verzweigung einleitet

**{ bzw. }** Beginn und Ende des if-Blocks

**x > 5** Bedingung. Ist sie erfüllt, werden  
die Anweisungen im if-Block  
ausgeführt, ansonsten wird der  
if-Block komplett übersprungen

## Verzweigungen 2

Beispiel:

```
if ( x > 5 ) {  
    // Anweisungen  
    // ...  
}  
else {  
    // Anweisungen  
    // ...  
}
```

**else** Schlüsselwort, das den  
“Sonst-Fall” abdeckt. Ist die  
Bedingung nicht erfüllt, werden  
die Anweisungen im else-Block  
ausgeführt

**{ bzw. }** Beginn und Ende des else-Blocks

## Verzweigungen 3

Beispiel:

```
if ( x > 5 ) {  
    // Anweisungen  
    // ...  
}  
else if ( x > 10 ) {  
    // Anweisungen  
    // ...  
}  
else {  
    // Anweisungen  
    // ...  
}
```

- else if** Schlüsselwort, das das Einfügen einer weiteren Bedingung ermöglicht
- x > 10** Weitere Bedingung. Diese wird nur geprüft, wenn die erste Bedingung (x > 5) **nicht** erfüllt ist. Ist sie erfüllt, werden die Anweisungen im else-if-Block ausgeführt.
- { bzw. }** Beginn und Ende des else-if-Blocks



- Es kann beliebig viele **else if**-Blöcke geben (auch gar keinen)
- Der **else**-Block ist optional

# Aufgabe 1

- Schreibe ein Programm, das abhängig von einer Zufallszahl einen Kreis zeichnet, der mit je ca. 50% Wahrscheinlichkeit entweder schwarz oder weiß gefüllt ist
  - Verwende die in Aufgabe 1 beschriebenen Methoden
  - Verwende eine geeignete Verzweigung
- Zusatzaufgabe
  - Erweitere das Programm um weitere Farben, die abhängig vom Zahlenbereich, in den die Zufallsvariable fällt, gewählt werden
  - Mit der Methode `fill(int r, int g, int b)` lassen sich auch Mischfarben aus rot, grün und blau erzeugen
  - Erweitere die Verzweigung

## Verknüpfen von Bedingungen

---

- Bisher können wir in einer Schleife genau eine Bedingung angeben, die die Ausführung kontrolliert
- Bei Verzweigungen können wir mehrere Bedingungen nacheinander mittels **else if** abfragen
  - Pro **else if** steht uns jedoch auch nur eine Bedingung zur Verfügung
- Was ist mit Problemen wie: “Falls der Wert zwischen 10 und 20 liegt, mache irgendwas”?
- Lösung:
  - Verknüpfen von Bedingungen mit aussagenlogischen Verknüpfungen

- Eine Bedingung kann nur entweder wahr oder falsch sein
- Zwei oder mehr Bedingungen können mittels aussagenlogischer Verknüpfungen zu einer neuen Bedingung verbunden werden
- Wir betrachten die Verknüpfungen UND, ODER und EXKLUSIVES ODER (XOR) sowie die Negation

- Die UND-Verknüpfung von zwei Bedingungen ist wahr, wenn **jede** der Bedingungen wahr ist

Bedingung A	Bedingung B	A UND B
false	false	false
false	true	false
true	false	false
true	true	true

- In Java werden zwei Bedingungen durch `&&` mittels UND verknüpft, z. B.

```
if ( x > 5 && x < 20 )
```

# ODER-Verknüpfung

- Die ODER-Verknüpfung von zwei Bedingungen ist wahr, wenn **mindestens eine** der Bedingungen wahr ist

Bedingung A	Bedingung B	A ODER B
false	false	false
false	true	true
true	false	true
true	true	true

- In Java werden zwei Bedingungen durch `||` mittels ODER verknüpft, z. B.

```
if ( x <= 5 || x >= 20 )
```

- Die XOR-Verknüpfung von zwei Bedingungen ist wahr, wenn **genau eine** der Bedingungen wahr ist

Bedingung A	Bedingung B	A XOR B
false	false	false
false	true	true
true	false	true
true	true	false

- In Java werden zwei Bedingungen durch `^` mittels XOR verknüpft, z. B.

```
if ( x <= 5 ^ y == 20 )
```



- Die Negation einer Bedingung verkehrt ihren Wert ins Gegenteil

Bedingung A	Negation von A
<b>false</b>	<b>true</b>
<b>true</b>	<b>false</b>

- In Java wird die Negation durch `!` umgesetzt, z. B.

```
if ( !(x < 5) )
```

- Statt `&&` und `||` stehen auch `&` und `|` zur Verfügung
- Die Operatoren `&&` und `||` sind sogenannte short-circuit-Operatoren
  - Es werden unter Umständen nicht alle Teilbedingungen ausgewertet
  - Beispiel:

```
if ( 5 < 0 && x < 3 )
```

Da `5 < 0` bereits **false** ist, kann die UND-Verknüpfung nicht mehr **true** werden, unabhängig von der zweiten Bedingung.

## Aufgabe 2

- Schreibe ein Programm, das in einem 800x600 Pixel großen Fenster 100000 Punkte an zufällige Positionen zeichnet. Wenn die Punkte innerhalb des Rechtecks mit dem Eckpunkten (300|200) und (500|300) liegen, sollen sie rot gezeichnet werden, ansonsten schwarz.
  - Verwende die Methode `random(float max)` zum Erzeugen von zufälligen Koordinaten (vom Typ `float`) für die einzelnen Pixel
  - Verwende die Methode `point(float x, float y)` zum Zeichnen der Pixel
  - Verwende die Methode `stroke(int r, int g, int b)` zum Setzen der Zeichenfarbe
- Zusatzaufgabe
  - Erweitere das Programm so, dass die Pixel, die innerhalb eines Kreises mit Radius 50 um den Punkt (700|450) liegen, grün gezeichnet werden (Tipp: Verknüpfte Bedingungen sind hier nicht nötig!)

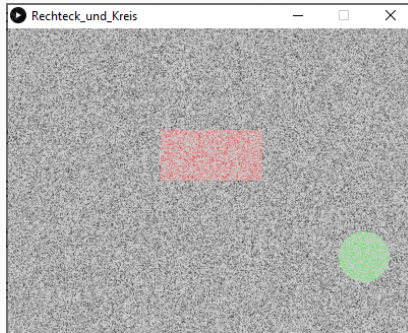


Abbildung 1: Ausgabe von Aufgabe 3

## Der Modulo-Operator

---

- Betrachte folgende Programmzeile. Welchen Wert hat die Variable x?

```
int x = 7 / 3;
```

- Lösung:
  - x hat den Wert 2
- Warum?
  - Da x, 7 und 3 jeweils ganze Zahlen (Typ `int`) sind, wird eine Ganzzahldivision durchgeführt.
  - Siehe Grundschule: "7 geteilt durch 3 ist 2 Rest 1"
  - Was ist mit dem Rest?

- Der Modulo-Operator % kann verwendet werden, um den Rest einer Ganzzahldivision zu berechnen
- Betrachte folgendes Beispiel. Welchen Wert hat die Variable x?

```
int x = 7 % 3
```

- Lösung:
  - x hat den Wert 1, da "7 geteilt durch 3" einen Rest von 1 lässt

- Zähler “zurücksetzen”:
  - Soll eine Variable bei Erreichen wieder auf 0 gesetzt werden, so ginge dies mittels

```
if ( x > MAX ) {  
    x = 0;  
}
```

Kürzer ist aber:

```
x = x % MAX;
```

- Teilbarkeit von Zahlen untersuchen, bspw. um gerade von ungeraden Zahlen zu unterscheiden
  - Eine gerade Zahl lässt beim Teilen durch 2 einen Rest von 0



- Schreibe ein Programm, das alle Pixel mit einer geraden x-Koordinate in schwarz zeichnet, alle anderen in weiß
  - Verwende ein Zeichenfenster der Größe 800x600 Pixel
  - Fülle den kompletten Zeichenbereich (also alle Zeilen) gemäß obiger Anweisung mit Pixeln
- Zusatzaufgabe:
  - Modifiziere das Programm so, dass ein Schachbrettmuster aus schwarzen und weißen Pixeln entsteht