

Programmierung, Datenstrukturen und Algorithmen

Dr. Christian Schönberg

Wintersemester 2020/2021

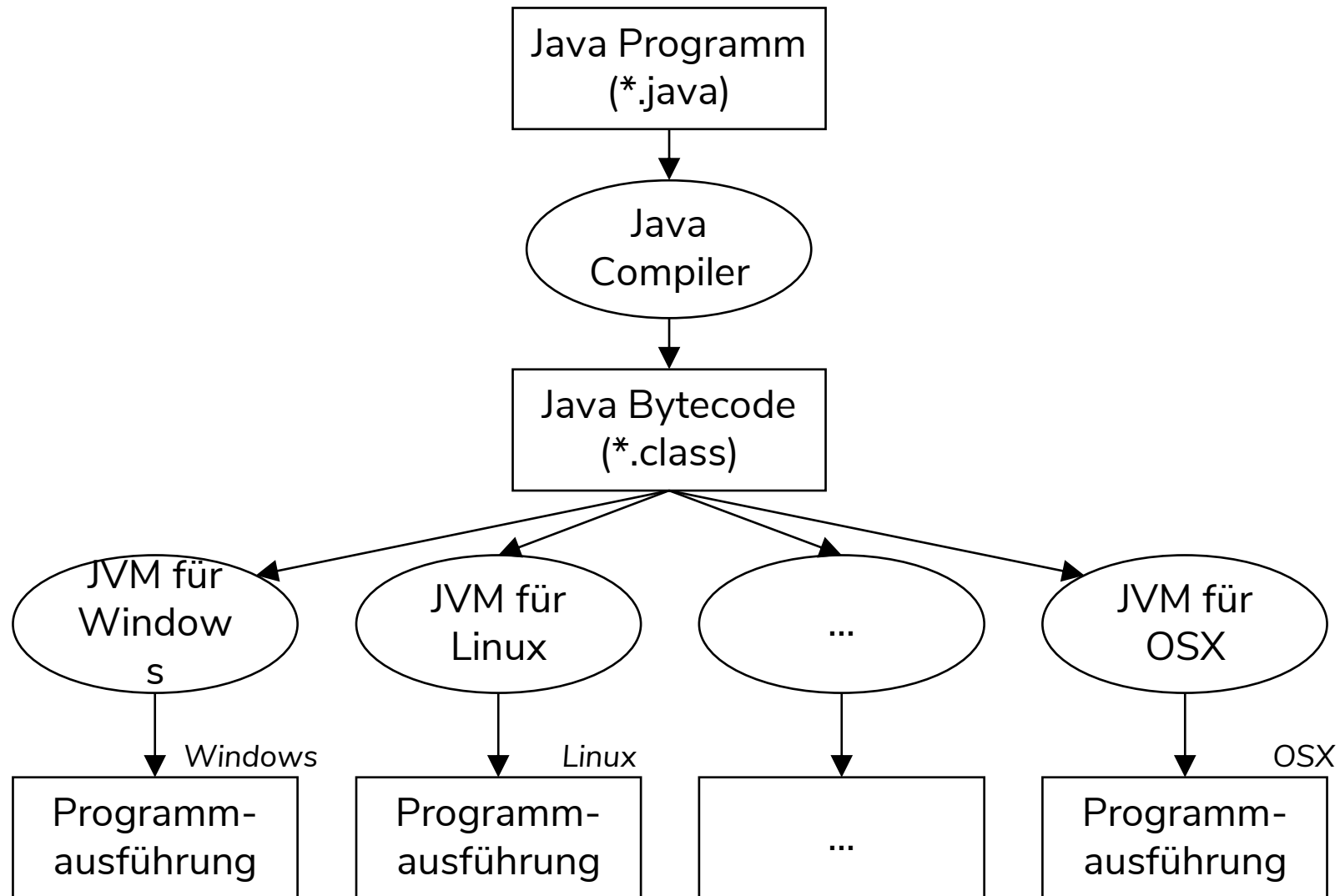
Java

- Eigenschaften, Terminologie und Arbeitsweise von Java
- Variablendeklaration, Wertzuweisung
- Blöcke
- Bedingungen (**if**)
- Schleifen (**while**)
- Kommentare
- Fehler

Was ist Java?

- Imperative, objekt-orientierte Programmiersprache
- Sammlung von Entwicklungswerkzeugen
- Klassenbibliothek
- Begriffe
 - **Java SE/EE**: Standard Edition/Enterprise Edition
 - **JRE**: Java Runtime Environment (zum Ausführen)
 - **JDK**: Java Development Kit (für Entwickler)
 - **JVM**: Java Virtual Machine (siehe Compiler/Interpreter)
- Aktuell: **Java SE 15 JDK**

Funktionsweise



Eigenschaften von Java

- einfach
- objekt-orientiert
- plattformunabhängig
- robust
- sicher
- schnell
- verteilt
- parallel
- dynamisch
- statische Typprüfung
- weit verbreitet
- kostenlos

Programm-Vorlage

```
public class <KLASSEN-NAME> {  
  
    public static void main(String[] args) {  
        <CODE>  
        ...  
    }  
  
}
```

→ speichern als <KLASSEN-NAME>.java

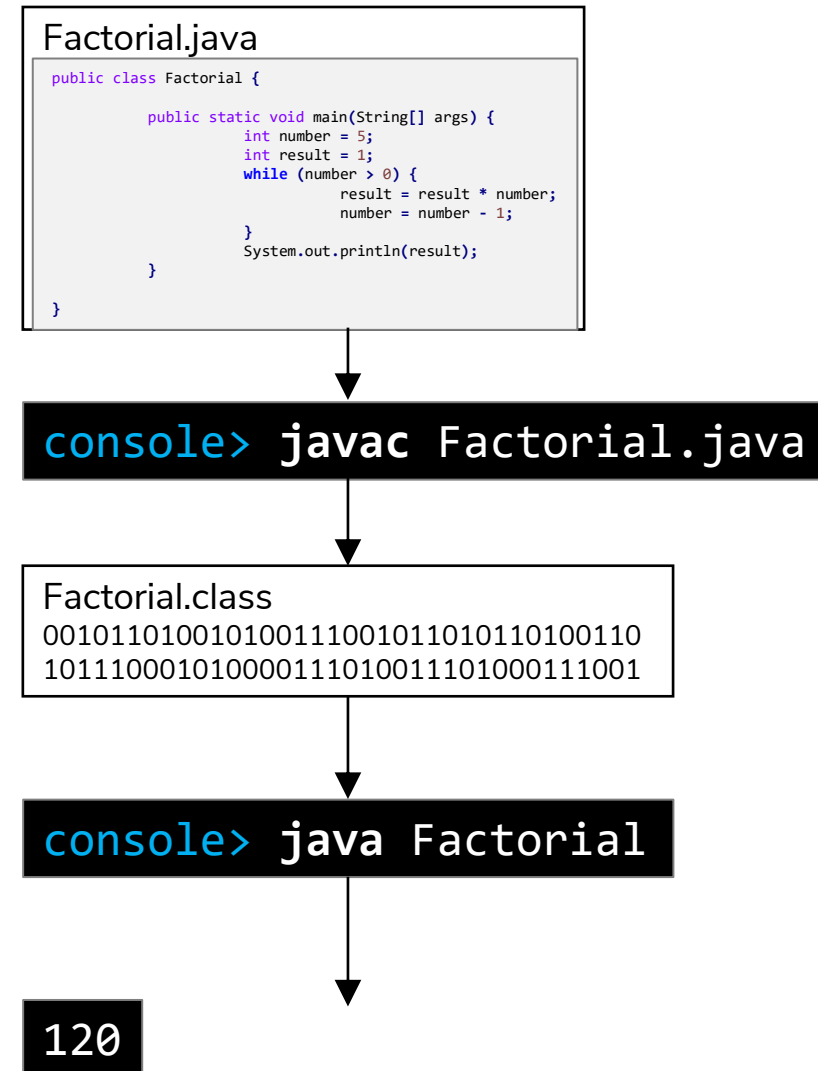
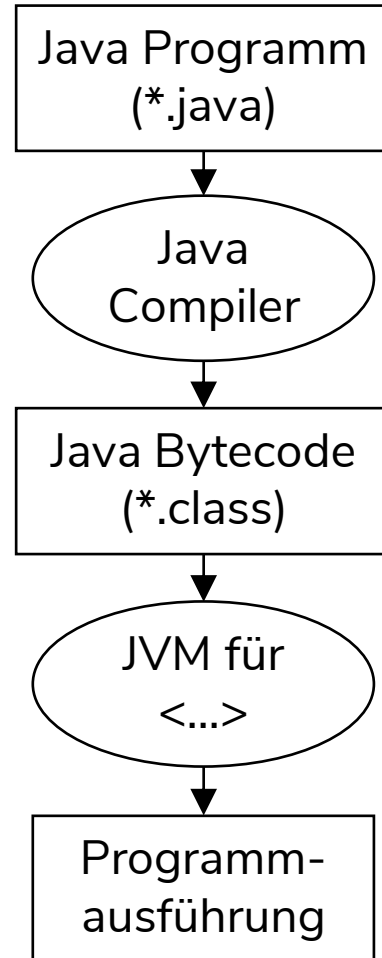
Beispielprogramm

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
  
}
```

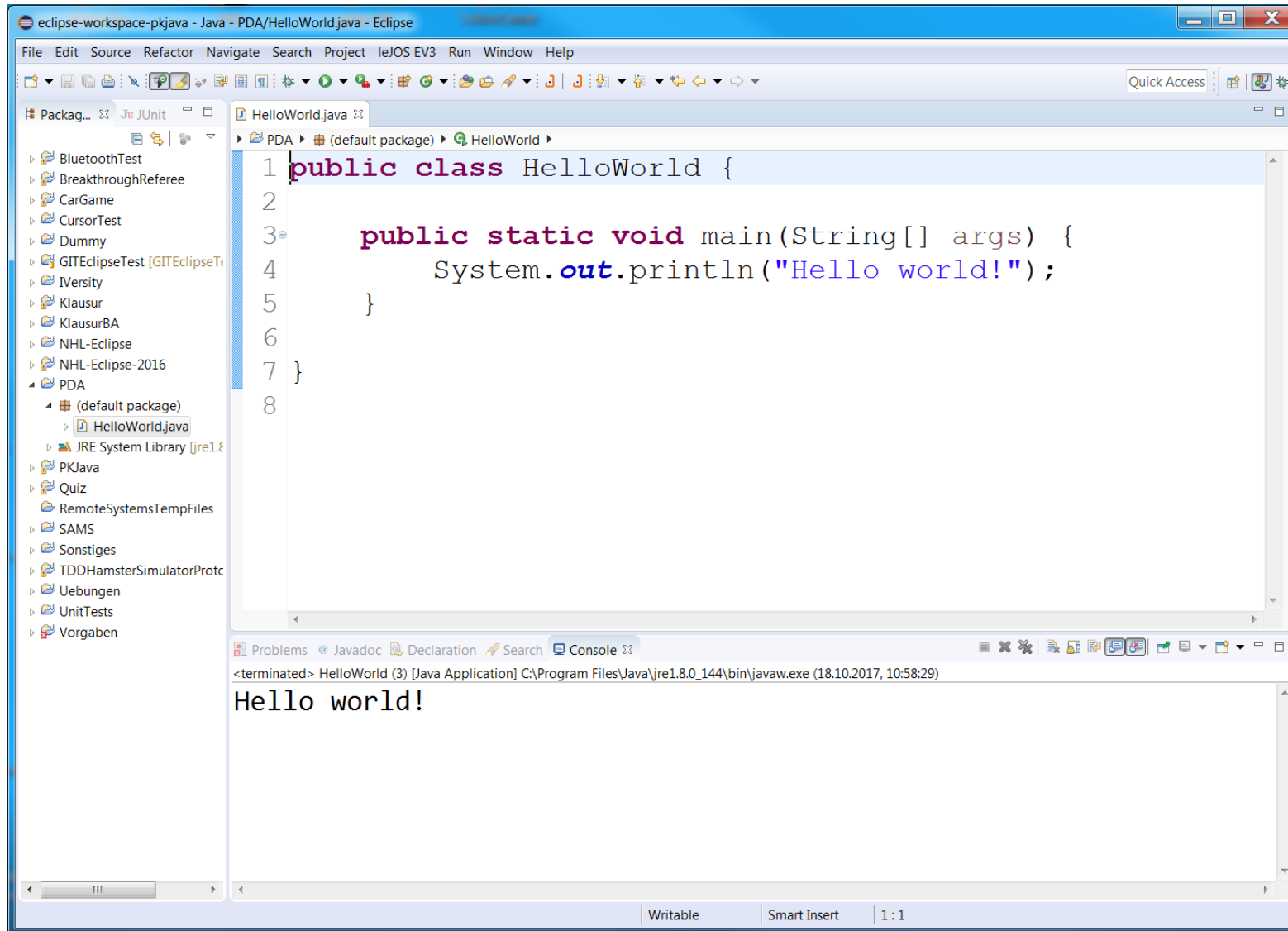

Beispielprogramm: Struktur

Klasse	<pre>public class Factorial {</pre>
Main-Methode	<pre> public static void main(String[] args) { int number = 5; int result = 1; while (number > 0) { result = result * number; number = number - 1; } System.out.println(result); }</pre>
	<pre>}</pre>

Ausführung des Beispielprogramms



Entwicklungsumgebungen



Variablen und Anweisungen

Variablen

- Platzhalter für Werte
- Funktionsweise (anders als in der Mathematik):
 - Variable deklarieren (Name und Typ festlegen)
 - `int number` (die Variable mit Namen „**number**“ kann Zahlenwerte aufnehmen)
 - Wert zuweisen
 - `number = 5` (die Variable hält jetzt den Wert 5)
 - Wert abfragen (mehrfach)
 - `number > 0` (ist der Wert der Variablen größer als 0?)
 - neuen Wert zuweisen
 - `number = number - 1` (die Variable hält jetzt den Wert 4)
- Benennung
 - Kleinbuchstabe gefolgt von beliebig vielen Buchstaben/Ziffern (vgl. **Identifizier**)
 - keine Schlüsselwörter

Beispielprogramm (2)

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
  
}
```

Beispielprogramm (2)

Variablendeklaration

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
}
```

Beispielprogramm (2)

Wertzuweisung

Alte Werte auslesen und
neue Werte zuweisen

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
}
```


Beispielprogramm (2)

Ausgabe des aktuellen Werts
des Variablen auf der Konsole

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
}
```

Einfache Arithmetische Operationen

- Addition, Subtraktion, Multiplikation und ganzzahlige Division
 - von Zahlen (Konstanten) und Variablen
 - „Punkt vor Strich“
 - Klammerausdrücke

```
int x = 5 + 4;  
int y = 9 - 3 * 2;  
int z = x / (y - 1);  
System.out.println(z);
```

4

- Vollständiger „Befehl“ in Java
- Beispielsweise eine einfache Variablendeklaration oder eine Berechnung

```
int x = 7;  
int y = 5 * x;
```

- Wird **immer** mit einem ; abgeschlossen

Struktur: Blöcke


- Sequenz von Anweisungen
- Eingeschlossen in { }
- Anweisungen im Block werden sequentiell ausgeführt

```
{  
    int number = 5;  
    int result = 1;  
    {  
        result = result * number;  
        number = number - 1;  
    }  
    System.out.println(result);  
}
```

- Durch Blöcke können Anweisungen gruppiert werden
- Können überall verwendet werden, wo eine Anweisung stehen kann

Gültigkeitsbereich von Variablen

- Variablen sind **gültig** (**sichtbar**) in dem Block, in dem sie definiert sind
- Sie sind ebenfalls gültig in allen darin verschachtelten Unterblöcken

```
{  
    int x = 5;  
    {  
        int y = x + 3;  
        int z = 7;  
    }  
    x = z - 2;   
}
```

Bedingte und wiederholte Anweisungen

Bedingte Anweisungen (**if**)

- Eine Anweisung/ein Block von Anweisungen wird nur ausgeführt, wenn eine bestimmte Bedingung erfüllt ist

```
int x = ...;  
if (x == 3) {  
    System.out.println(x);  
}
```

- Wenn eine Bedingung *nicht* erfüllt ist, können andere Anweisungen ausgeführt werden

```
int x = ...;  
if (x == 0) {  
    System.out.println(0);  
} else {  
    System.out.println(100 / x);  
}
```

Bedingte Anweisungen (**if**) (2)

```
int x = ...;
if (x == 0) {
    System.out.println(0);
} else if (x > 10) {
    System.out.println(100 / x);
} else if (x > 0) {
    System.out.println(10 / x);
} else {
    System.out.println(-1 * x);
}
```


Einfache Bedingungen

```
int x = ...;
if (x == 0) {
    System.out.println(x);
}
if (x > 10) {
    System.out.println(x);
}
if (x < 10) {
    System.out.println(x);
}
if (x != 0) {
    System.out.println(x);
}
if (x >= 5 && x <= 10) {
    System.out.println(x);
}
if (x <= 5 || x >= 10) {
    System.out.println(x);
}
```

- ==, !=
 - gleich, ungleich
- >, <, >=, <=
 - größer, kleiner, größer oder gleich, kleiner oder gleich
- &&, ||, !
 - und, oder, nicht


Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist

```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher

x = 0

Ausgabe

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```


Speicher

x = 0

Ausgabe

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```


Speicher

x = 1

Ausgabe

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher

x = 1

Ausgabe

1

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher


x = 1

Ausgabe

1

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher


x = 2

Ausgabe

1

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher


x = 2

Ausgabe

1
2

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher


x = 2

Ausgabe

1
2

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher


x = 3

Ausgabe

1
2

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher

x = 3

Ausgabe

1
2
3

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher


x = 3

Ausgabe

1
2
3

Schleifen (**while**)

- Schleifen wiederholen die Ausführung einer Anweisung/eines Blocks von Anweisungen solange, wie eine Bedingung erfüllt ist



```
int x = 0;
while (x < 3) {
    x = x + 1;
    System.out.println(x);
}
```

Speicher

x = 3

Ausgabe

1
2
3

Beispiel: **while**-Schleife

While-Schleife

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
}
```

Beispielprogramm

```
public class Factorial {  
  
    public static void main(String[] args) {  
        int number = 5;  
        int result = 1;  
        while (number > 0) {  
            result = result * number;  
            number = number - 1;  
        }  
        System.out.println(result);  
    }  
}
```


Lesbarkeit

- Kommentare sind Text im Programmcode, der nicht interpretiert wird
- Sie erhöhen die Lesbarkeit des Codes, indem sie erklären
 - welche Funktion ein Codeabschnitt erfüllt (**was**)
 - wie ein Codeabschnitt funktioniert (**wie**)
 - was für Bedingungen gelten müssen, damit ein Codeabschnitt wie erwartet funktioniert (**Vorbedingungen**)
 - was für Bedingungen nach der Ausführung eines Codeabschnitts erfüllt sind (**Nachbedingungen**)
- Kommentare gehören zur guten Programmierpraxis
- Unterscheidung zwischen **Zeilenkommentaren** und **Mehrzeilenkommentaren**

Beispiel: Kommentare

```
public static void main(String[] args) {  
    /*  
     * Calculate the factorial of a given number  
     * using an iterative approach. The given  
     * number must be greater than zero.  
     * The result will be printed to the console.  
     */  
    int number = 5; // set the number to an arbitrary value  
    int result = 1; // initialize the result  
    while (number > 0) { // termination condition  
        result = result * number; // update result  
        // decrement number to satisfy termination condition  
        number = number - 1;  
    }  
    System.out.println(result); // print result  
}
```

Lesbarkeit: Code Conventions

- Eine Anweisung in einer Zeile
- Zeilenumbruch nach {, eigene Zeile für }
- Leerzeichen ()
 - vor öffnenden und nach schließenden Klammern
 - um Operatoren (+, *, =, ==, >, ...)

```
if_(x_==_3)_{  
    ...  
}
```

- Namen
 - Keine Umlaute und andere Sonderzeichen (ö, ß, &, ...)
 - Klassen beginnen mit Großbuchstaben **Factorial**
 - Variablen beginnen mit Kleinbuchstaben **number**
 - CamelCase **FactorialCalculator,**
resultNumber
 - aussagekräftige Namen tmp316, b, rnsTf ⚡

Typische Fehler

- Syntaxfehler

```
While (i = 8) i + 1; }
```

- Laufzeitfehler

```
int x = 3 / y; // y == 0
```

- Unendliche Schleifen

```
int x = 0;  
int y = 0;  
while (x < 3) {  
    y = y + 1;  
}
```

```
int x = 0;  
while (x < 3) {  
    x = x - 1;  
}
```

Typische Fehler

- Syntaxfehler

```
While (i = 8) i + 1; }
```

```
while (i == 8) { i = i + 1; }
```

- Laufzeitfehler

```
int x = 3 / y; // y == 0
```

- Unendliche Schleifen

```
int x = 0;  
int y = 0;  
while (x < 3) {  
    y = y + 1;  
}
```

```
int x = 0;  
while (x < 3) {  
    x = x - 1;  
}
```

Typische Fehler

■ Syntaxfehler

```
While (i = 8) i + 1; }
```

```
while (i == 8) { i = i + 1; }
```

■ Laufzeitfehler

```
int x = 3 / y; // y == 0
```

```
if (y != 0) {  
    int x = 3 / y;  
}
```

■ Unendliche Schleifen

```
int x = 0;  
int y = 0;  
while (x < 3) {  
    y = y + 1;  
}
```

```
int x = 0;  
while (x < 3) {  
    x = x - 1;  
}
```

Typische Fehler

■ Syntaxfehler

```
While (i = 8) i + 1; }
```

```
while (i == 8) { i = i + 1; }
```

■ Laufzeitfehler

```
int x = 3 / y; // y == 0
```

```
if (y != 0) {  
    int x = 3 / y;  
}
```

■ Unendliche Schleifen

```
int x = 0;  
int y = 0;  
while (x < 3) {  
    y = y + 1;  
}
```

```
int x = 0;  
while (x < 3) {  
    x = x - 1;  
}
```

```
int x = 0;  
int y = 0;  
while (x < 3) {  
    x = x + 1;  
}
```

```
int x = 0;  
while (x < 3) {  
    x = x + 1;  
}
```


Lernziele

- Eigenschaften, Terminologie und Arbeitsweise von Java
- Variablendeklaration, Wertzuweisung
- Blöcke
- Bedingungen (**if**)
- Schleifen (**while**)
- Kommentare
- Fehler