Einführung in Java

Was ist Java?

Java

Auch Java

Java
Programming
Language

Java - Geschichte

- Erste Version
  - Java 1.0 (23. Januar 1996)

- Aktuelle Version
  - Java 21 (19. September 2023)

- Programmiersprache
  - Imperativ
  - Stark und statisch typisiert

Dr.-Ing. Christian Braune

Java - Geschichte

- Erste Version
  - Java 1.0 (23. Januar 1996)

- Aktuelle Version
  - ~~Java 21 (19. September 2023)~~
  - Java 22 (19. März 2024)

- Programmiersprache
  - Imperativ
  - Stark und statisch typisiert

Dr.-Ing. Christian Braune

Java - Geschichte

- Erste Version
  - Java 1.0 (23. Januar 1996)

- Aktuelle Version
  - ~~Java 21 (19. September 2023)~~
  - ~~Java 22 (19. März 2024)~~
  - Java 23 (17. September 2024)

- Programmiersprache
  - Imperativ
  - Stark und statisch typisiert

  - Objekt-orientiert

Dr.-Ing. Christian Braune

Hallo, Welt! – in Java

- Ein erstes Java Programm:

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

- Ausgabe

```
>>>  Hallo, Welt!
```


Dr.-Ing. Christian Braune
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF FAKULTÄT FÜR INFORMATIK

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander...

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

public ist ein Keyword, das Objekte sichtbar macht. Das brauchen wir hier, damit wir das Programm überhaupt aufrufen können.

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander...

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

class ist ein Keyword, zum Definieren von Klassen. Jedes Programm in Java besteht aus mindestens einer Klasse.

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander...

```
1    public class HelloWorld {
2      public static void main(String[] args) {
3        System.out.println("Hallo, Welt!");
4      }
5    }
```

Das ist einfach der Name unserer Klasse bzw. unseres Programms

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander…

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

static ist ein Keyword, dass es uns erlaubt auf Methoden oder Attribute zuzugreifen, ohne eine konkrete Instanz einer Klasse zu haben.

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander...

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

void ist ein Keyword, dass anzeigt, dass die in dieser Zeile definierte Funktion **keinen** Rückgabewert hat.

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander...

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

Das ist die Signatur der Funktion. Wenn eine Klasse eine Methode mit genau dieser Signatur hat, kann sie "ausgeführt" werden.

Hallo, Welt! – in Java

- Das nehmen wir jetzt mal auseinander...

```
1   public class HelloWorld {
2     public static void main(String[] args) {
3       System.out.println("Hallo, Welt!");
4     }
5   }
```

System.out erlaubt und den Zugriff auf die Standardausgabe des Systems (im allgemeinen wird das die Konsole sein). println ist eine Funktion, die den übergebenen String ausgibt.

Wie führe ich ein Java-Programm aus?

- Wir brauchen eine Textdatei mit dem Namen der Klasse
- und unserem Programm als Inhalt

HelloWorld.java

```
1  public class HelloWorld {
2    public static void main(String[] args) {
3      System.out.println("Hallo, Welt!");
4    }
5  }
```

- Danach müssen wir die Datei kompilieren und können sie dann ausführen

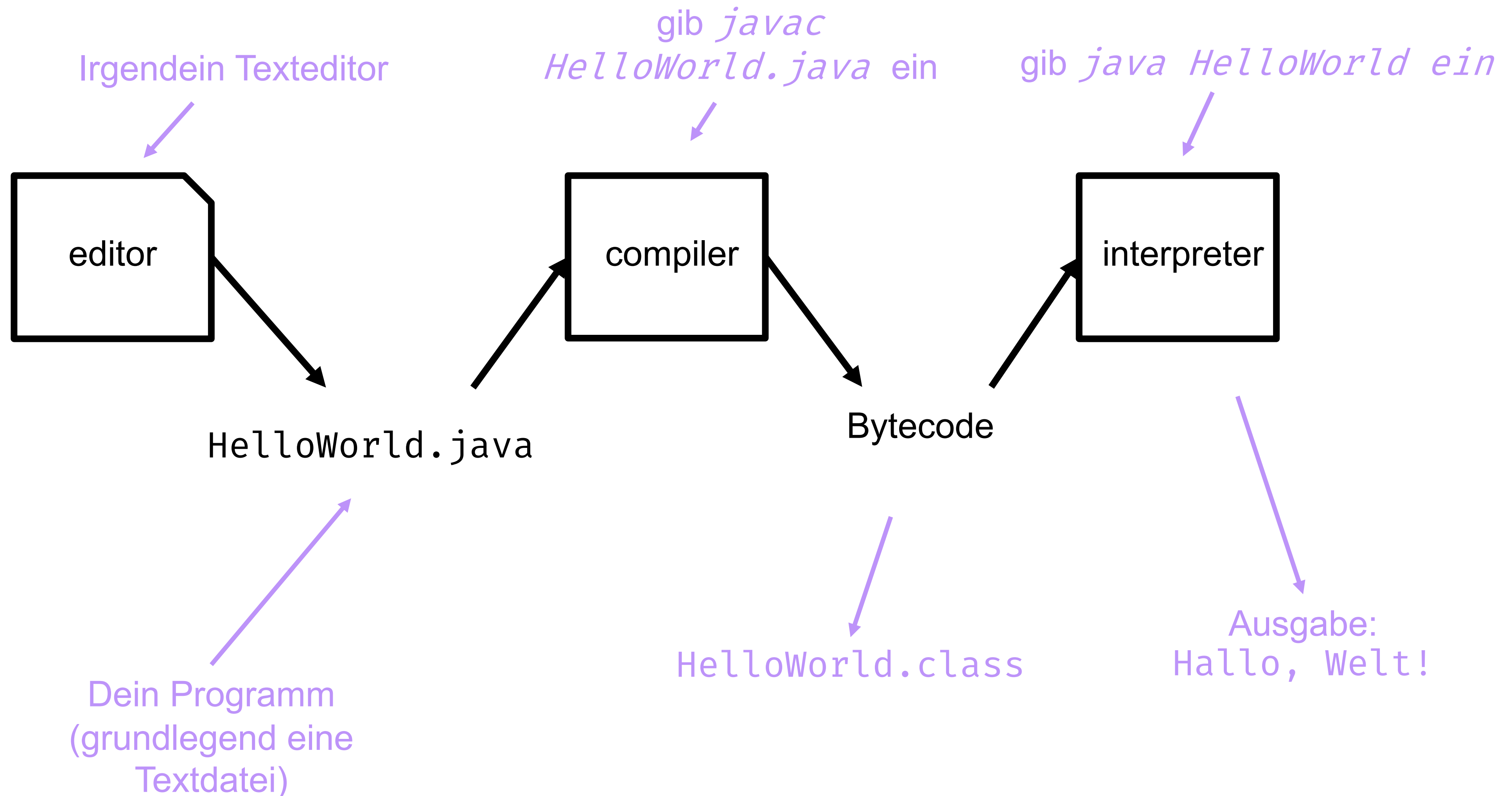
Dr.-Ing. Christian Braune

Java Compiler und Java Interpreter

- Der Java Compiler `javac` übersetzt den Quellcode in Bytecode
  - Dateien müssen den selben Namen haben wie die Klasse, die sie beinhalten
  - Dateiendung: `.java`
- Bytecode ist ein plattform-unabhängiger Zwischencode
  - anders als kompilierte C-Dateien, die immer plattformabhängig sind!
  - Dateiendung: `.class`
- Der Java Interpreter (JVM) interpretiert den Bytecode
  - kann dadurch von der CPU ausgeführt werden

Java ist eine kompilierte Sprache

- Java ist eine kompilierte Sprache



Dr.-Ing. Christian Braune

Datentypen in Java

Variablen und Datentypen

- Jede Variable in Java hat (genau) einen Datentyp
  - Wird festgelegt, wenn das Programm kompiliert wird

- Datentypen können alles mögliche sein

- Fest eingebaut sind Datentypen für:
  - logische Werte `boolean`
  - ganze Zahlen `byte, int, long`
  - rationale Zahlen `float, double`
  - Zeichen und Zeichenketten `char, String`

Dr.-Ing. Christian Braune

- Variablen **müssen** deklariert werden, bevor sie benutzt werden können.
- Durch das deklarieren reserviert Java ausreichend Speicherplatz für den Inhalt der Variablen
- Variablen **sollten** initialisiert werden

Bool'sche Variablen

- Bool'sche Variablen stellen einen Wahrheitswert dar
  - Entweder `true` oder `false`

- Mögliche Operationen:
  - Negation — logisches NOT
  - Konjunktion — logisches ODER
  - Disjunktion — logisches UND

```
1    boolean hasVacation;
2    hasVacation = false;
3
4    boolean isInClass = true;
```


Dr.-Ing. Christian Braune
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF FAKULTÄT FÜR INFORMATIK

Bool'sche Variablen - Negation

- Bool'sche Variablen stellen einen Wahrheitswert dar
  - Entweder `true` oder `false`

- Mögliche Operationen:
  - **Negation** **logisches NOT**
  - Disjunktion logisches ODER
  - Konjunktion logisches UND

- Wandelt einen Wahrheitswert in sein Gegenstück um

```
1    boolean hasMoney = true;
2    boolean hasNoMoney = !hasMoney;
```

Bool'sche Variablen - Disjunktion

- Bool'sche Variablen stellen einen Wahrheitswert dar
  - Entweder `true` oder `false`

- Mögliche Operationen:
  - Negation — logisches NOT
  - **Disjunktion** — **logisches ODER**
  - Konjunktion — logisches UND

- Mindestens eine der Bedingungen muss erfüllt sein

```
1    boolean hasMoney = false;
2    boolean isHappy = true;
3
4    boolean isRich = hasMoney || isHappy;
```


Dr.-Ing. Christian Braune

Bool'sche Variablen - Konjunktion

- Bool'sche Variablen stellen einen Wahrheitswert dar
  - Entweder `true` oder `false`

- Mögliche Operationen:
  - Negation — logisches NOT
  - Disjunktion — logisches ODER
  - **Konjunktion** — **logisches UND**
- Beide Bedingungen müssen erfüllt sein

```
1    boolean isFinal = true;
2    boolean hasWon = true;
3
4    boolean isChampion = isFinal && hasWon;
```


Dr.-Ing. Christian Braune

Integer Variablen

- Integer Variablen stellen ganzzahlige Werte dar
  - Achtung: beschränkter Wertebereich!

- Unterschiedliche Größen:
  - byte - 128 ... 127
  - short - 32.768 ... 32.767
  - **int - 2.147.483.648 ... 2.147.483.647**
  - long -9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807

```
1    byte smallNumber = 17;
2    short mediumNumber = 1024;
3    int normalNumber = 100000;
4    long largeNumber = 9_876_543_210L;
```


Dr.-Ing. Christian Braune

Integer Variablen

- Integer Variablen stellen ganzzahlige Werte dar
  - Achtung: beschränkter Wertebereich!

- Mögliche Operationen
  - Addition, Subtraktion, Multiplikation, Division, Modulo: `+ - * / %`
  - Achtung: Kein Potenzieren! (weder mit `**` noch mit `^`)

- Was passiert bei Überschreiten des Wertebereichs?

```java
1   byte smallNumber = 127;
2   System.out.println(smallNumber + 1);
```

Kurzschreibweise

- Oft kommt es vor, dass den Wert einer Variablen verändern – und das Ergebnis wieder in der selben Variablen abspeichern – wollen

```
1    int x = 5;
2    x = x + 3;
```

- Dafür gibt es eine kürzere Schreibweise:

```
1    int x = 5;
2    x += 3;      // x = x + 3;
3    x -= 1;      // x = x - 1;
4    x *= 4;      // x = x * 4;
5    x /= 7;      // x = x / 7;
6    System.out.println("x = " + x);
```

Pre-Inkrement und Post-Inkrement

- Oft kommt es vor, dass den Wert einer Variablen um 1 erhöhen – und das Ergebnis wieder in der selben Variablen abspeichern – wollen

```
1    int x = 5;
2    x = x + 3;
```

- Dafür gibt es eine kürzere Schreibweise:

```
1    int x = 5;
2    x++;   // x += 1;              post-increment
3    ++x;   // fast das gleiche     pre-increment
```

Pre-Dekrement und Post-Dekrement

- Oft kommt es vor, dass den Wert einer Variablen um 1 verringern – und das Ergebnis wieder in der selben Variablen abspeichern – wollen

```
1    int x = 5;
2    x = x + 3;
```

- Dafür gibt es eine kürzere Schreibweise:

```
1    int x = 5;
2    x++;   // x += 1;                post-increment
3    ++x;   // fast das gleiche        pre-increment
4    x--;   // x -= 1; ;              post-decrement
5    --x;   // fast das gleiche        pre-decrement
```

Unterschied zwischen pre- und post-

- Der Hauptunterschied zwischen Pre-Inkrement und Post-Inkrement (bzw. –dekrement) besteht in der Reihenfolge der Auswertung

- Vergleichen wir

```
1    int x = 5;
2    int y = x++;
```

mit

```
1    int x = 5;
2    int z = ++x;
```

- Welche Werte haben y und z?

Fließkommazahlen

- Fließkommazahlen stellen eine rationale Zahl dar
  - Also einen Bruch, keine reelle Zahl.
  - Bei kleinen Zahlen ist eine hohe Genauigkeit möglich (viele Nachkommastellen), bei großen Zahlen nicht

- Unterschiedliche Größen
  - float ±3.4E±38 Genauigkeit: ca. 7 Nachkommastellen
  - **double ±7.2E±75 Genauigkeit: ca. 16 Nachkommastellen**

```
1    float pi = 3.14159f;
2    double precisePi = 3.14159265358979323846;
```


Dr.-Ing. Christian Braune

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF FAKULTÄT FÜR INFORMATIK

Fließkommazahlen

- Fließkommazahlen stellen eine rationale Zahl dar
  - Also einen Bruch, keine reelle Zahl.
  - Bei kleinen Zahlen ist eine hohe Genauigkeit möglich (viele Nachkommastellen), bei großen Zahlen nicht

- Mögliche Operationen
  - Addition, Subtraktion, Multiplikation, Division, Modulo: `+ - * / %`
  - Achtung: Kein Potenzieren! (weder mit `**` noch mit `^`)

```
1    float pi = 3.14159f;
2    float tau = 2 * pi;
```


Dr.-Ing. Christian Braune

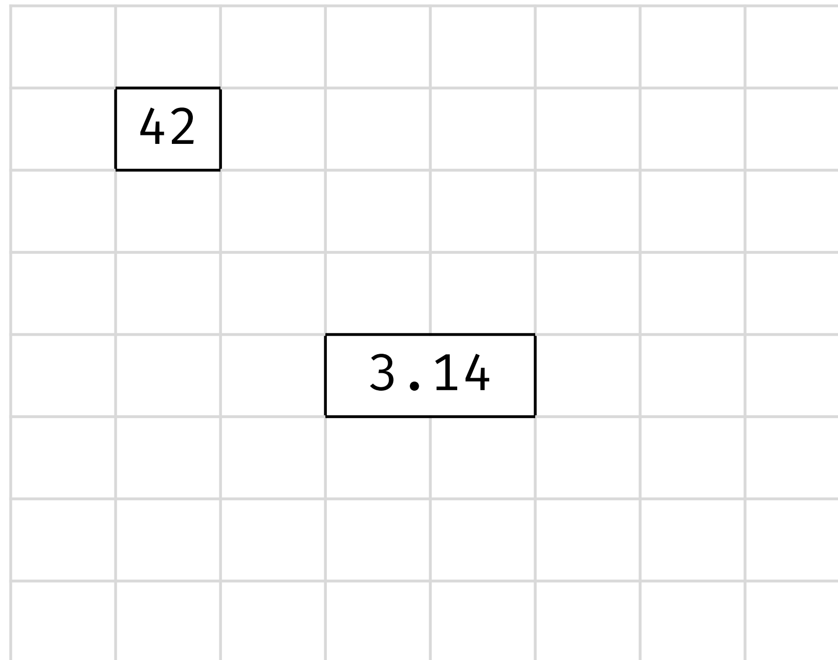
- Eine Array ist eine Art von Datentyp, die es erlaubt, mehrere gleichartige Datentypen in einer Variable zusammenzufassen

- Entweder müssen wir alle Werte des Arrays am Anfang festlegen (Zeile 1) oder wir geben die Größe des Arrays an und legen die Elemente einzeln fest (Zeile 2-4)

```
1    int[] numbers = {0, 1, 2};
2    double[] otherNumbers = new double[2];
3    otherNumbers[0] = 3.14;
4    otherNumbers[1] = 6.28;
```

Referenzdatentypen

- Wie werden Daten im Speicher abgelegt?



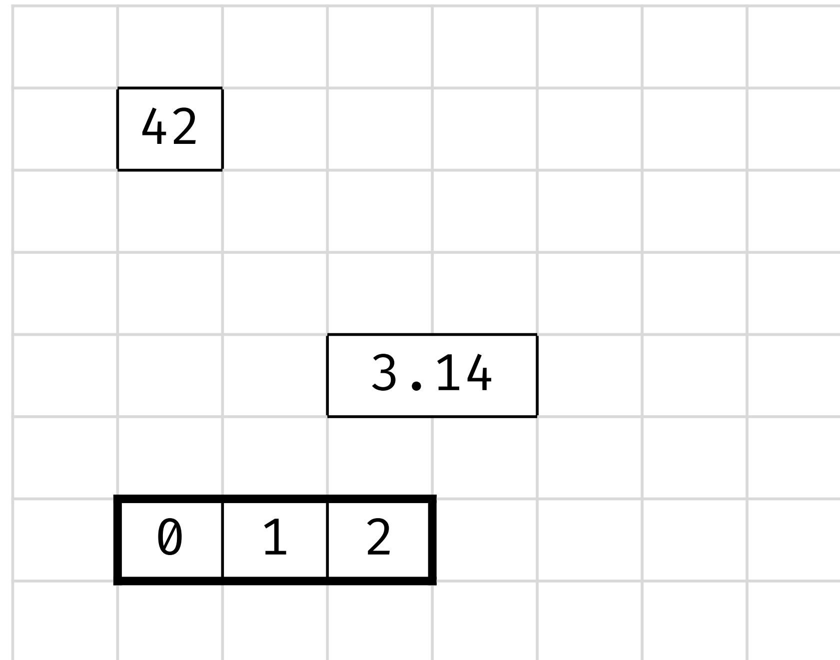
← Speicher*

* jedes Kästchen ist 32bit groß

```
1    int importantNumber = 42;
2    double pi = 3.14;
```


Dr.-Ing. Christian Braune

- Wie werden Daten im Speicher abgelegt?



← Speicher*

* jedes Kästchen ist 32bit groß

```
1    int importantNumber = 42;
2    double pi = 3.14;
3    int[] numbers = {0, 1, 2};
```

- Wie werden Daten im Speicher abgelegt?
- In der Variablen **numbers** sind nicht die Werte 0, 1, 2 abgespeichert sondern die Adresse, wo im Speicher die Werte zu finden sind
- Wie Pointer in C

```
1    int importantNumber = 42;
2    double pi = 3.14;
3    int[] numbers = {0, 1, 2};
```

Zeichen (und Zeichenketten)

- Zeichen stellen ein ... Zeichen dar
  - intern werden sie als `short` (aber ohne Vorzeichen) behandelt
  - Zahlenwert entspricht dem UTF-16 Code (ASCII + Erweiterung)

- Wertebereich
  - `\u0000` ... `\uFFFF`

- "Mögliche" Operationen:
  - Addition / Subtraktion

```
1    char lowerCaseA = 'a';
2    char lowerCaseB = (char)(lowerCaseA + 1);
3    char upperCaseA = (char)(lowerCaseA - ' ');
```

(Zeichen und) Zeichenketten

- Strings sind Variablen, die mehrere Zeichen zu einer Zeichenkette vereinen.
- Im wesentlichen sind Strings also Arrays von Zeichen
- Wertebereich
  - leerer String: `""` bis *was auch immer euer Arbeitsspeicher hergibt...*

- mögliche Operationen:
  - Konkatenation

```
1    String emptyString = "";
2    String hello = "Hello, ";
3    String world = "World!";
4    System.out.println(hello + world);
```

Zeichenketten

- Zeichenketten sind keine primitiven Datentypen
  - Deswegen schreiben wir in Java auch `String` und nicht `string`!

- Die Variablen `hello` und `world` sind unsere ersten Objekte

Objekte

- Objekte haben einen Zustand, Eigenschaften und Methoden, die den Zustand verändern können

```java
1   public class HelloWorld {
2     public static void main(String[] args) {
3       String hello = "Hallo, ";
4       String world = "Welt!";
5       hello = hello.replace('a', 'e');
6       world = world.replaceAll("elt", "orld");
7       System.out.println(hello + world);
8     }
9   }
```

- Punkt-Operator erlaubt Zugriffe auf diese Eigenschaften/Methoden

Weitere nützliche String-Methoden

```
1    String str = "Hallo, Welt!";
```

- Wie lang ist ein String?

```
2    int laenge = str.length();
```

- Beginnt ein String mit einer bestimmten Zeichenkette?

```
3    boolean germanGreeting = str.startsWith("Hallo");
```

- Endet ein String mit einer bestimmten Zeichenkette?

```
4    boolean greetAll = str.endsWith("Welt!");
```

Weitere nützliche String-Methoden

```
1    String str = "Hallo, Welt!";
```

- Ersetze einzelne Zeichen in einem String

```
2    String newStr = str.replace('l', '.');
3    // newStr ist jetzt "Ha..o, We.t!"
```

- Ersetze eine ganze Zeichenkette innerhalb eines Strings

```
4    newStr = str.replaceAll("Welt", "Christian");
5    // newStr ist jetzt "Hallo, Christian!"
```


Dr.-Ing. Christian Braune

Casting

- Unter Casting versteht man das Umwandeln eines Datentyps in einen anderen
- Das kann leise geschehen (implizites Casting):

```
1    byte smallNumber = 17;
```

- oder durch explizites Casting:

```
1    char lowerCaseA = 'a';
2    char lowerCaseB = (char)(lowerCaseA + 1);
3    char upperCaseA = (char)(lowerCaseA – ' ');
```

- Explizites Casting ist immer dann notwendig, wenn die Umwandlung mit einem möglichen Verlust einhergeht

- Explizites Casting ist immer dann notwendig, wenn die Umwandlung mit einem möglichen Verlust einhergeht

```
1    byte smallNumber = 17;  // 0x00000011
```

- 17 entspricht 0x00000011
  - (32 bit, je zwei Hexadezimalstellen entsprechen 8 bit)
  - Beim Umwandeln in ein byte werden die obersten 24bit entfernt (hier: alles Nullen)

- 317 entspricht 0x0000013D

```
2    byte anotherNumber = (byte)317;  // ist 61
3    byte aThirdNumber = 317;         // Fehler
```

- Impliziter Cast nicht möglich!

Flusskontrolle

Zwei Arten von Flusskontrolle

1. Verzweigungen
   - Teile des Programms sollen nur ausgeführt werden, wenn eine bestimmte Bedingung zutrifft
   - Andere Teile vielleich nur, wenn eine Bedingung nicht zutrifft

```
1   boolean flag = true;
2   if (flag) {
3     System.out.println("Flag ist gesetzt.");
4   } else {
5     System.out.println("Flag ist nicht gesetzt.");
6   }
```

Bool'sche Ausdrücke

- Ausdrücke sind Anweisungen, die einen Wert erzeugen

```
1   3 + 5;  // erzeugt den Wert 8
```

- Bool'sche Ausdrücke erzeugen einen Wahrheitswert

```
2   true && false;  // erzeugt den Wert false
3   true || false;  // erzeugt den Wert true
```

- Jeder Vergleich erzeugt einen Wahrheitswert

Vergleichsoperatoren

- Test auf Gleichheit bzw. Ungleichheit
  - `==` bzw. `!=`
  - Testet, ob in beiden Operanden exakt das selbe gespeichert ist

```
1    int i = 5;
2    int j = 2 + 3;
3    boolean flag = (i == j);  // true
```

- Ordnungsrelationen:
  - `>`, `>=`   `(echt größer, größer oder gleich)`
  - `<=`, `<`   `(kleiner oder gleich, echt kleiner)`

Dr.-Ing. Christian Braune

`else` ist optional

- Den else-Teil der Verzweigung können wir weglassen

```java
int amount = 2;
System.out.print("You want " + amount + "apple");
if (amount > 1) {
    System.out.print("s");
}
System.out.println();
```


Dr.-Ing. Christian Braune

- Nützlich, wenn wir mehrere Bedingungen hintereinander prüfen wollen

```
1    int choice = 3;
2    if (choice == 0) {
3      System.out.println("Du hast die 0 gewählt.");
4    } else if (choice == 1) {
5      System.out.println("Du hast die 1 gewählt.");
6    } else if (choice == 2) {
7      System.out.println("Du hast die 2 gewählt.");
8    } else {
9      System.out.println("UNGÜLTIGE AUSWAHL!");
10   }
```

Das muss auch einfacher gehen...

- Das Beispiel auf der vorherigen Folie ist sehr umständlich
- Und es geht noch schlimmer, wenn mehr Unterscheidungen dazu kommen

```
1    int choice = 3;
2    switch(choice) {
3      case 0: System.out.println("0 gewählt");
4              break;
5      case 1: System.out.println("1 gewählt");
6              break;
7      default: System.out.println("UNGÜLTIGE WAHL!");
8    }
```

Was würde hier passieren?

- Was passiert, wenn wir das `break` weglassen?

```
1    int choice = 0;
2    switch(choice) {
3      case 0:
4        System.out.println("0 gewählt");
5      case 1:
6        System.out.println("1 gewählt");
7      default:
8        System.out.println("UNGÜLTIGE WAHL!");
9    }
```

Zwei Arten von Flusskontrolle

1. Verzweigungen
   - Teile des Programms sollen nur ausgeführt werden, wenn eine bestimmte Bedingung zutrifft
   - Andere Teile vielleich nur, wenn eine Bedingung nicht zutrifft

- Schleifen
  - Teile des Programmes sollen mehr als einmal ausgeführt werden

```
1   int counter = 10;
2   while (counter > 0) {
3     System.out.println("Countdown: " + counter);
4     counter = counter - 1;
5   }
```

While-Schleife

- Der Schleifenkörper wird solange ausgeführt, wie eine Bedingung erfüllt ist.
- Die Bedingung wird **VOR** jedem Durchlauf überprüft
- Möglicherweise wird die Schleife niemals ausgeführt
- wird auch *kopfgesteuerte Schleife* genannt

```
1   boolean condition = false;
2   while (condition) {
3     System.out.println("noch in der Schleife");
4     condition = checkCondition();
5   }
```

Do-While-Schleife

- Der Schleifenkörper wird solange ausgeführt, wie eine Bedingung erfüllt ist.
- Die Bedingung wird **NACH** jedem Durchlauf überprüft
- Die Schleife wird immer mindestens einmal ausgeführt
- wird auch *fußgesteuerte Schleife* genannt

```
1   boolean condition = false;
2   do {
3     System.out.println("noch in der Schleife");
4     condition = checkCondition();
5   } while (condition);
```

- Der Schleifenkörper wird für eine bestimmte Anzahl an Wiederholungen ausgeführt

- Die Schleife wird immer mindestens einmal ausgeführt

- wird auch *Zählschleife* genannt

```
1    for(int i = 0; i < 10; ++i) {
2      System.out.println("Wiederholung Nr. " + i);
3    }
```

```
1    for(int i = 0; i < 10; ++i) {
2      System.out.println("Wiederholung Nr. " + i);
3    }
```

- **Initialisierung:** Für die Schleife benötigte Variable(n) können hier initialisiert werden
- **Überprüfung:** hier finden alle nötigen Überprüfungen VOR jedem Schleifendurchlauf statt
- **Inkrement:** Hier werden die Schleifenvariablen verändert
- In jedem dieser Bereiche können auch komplexere Dinge stehen

Unterschiede zwischen den Schleifen

- Eigentlich keine.
  - Jede Schleife kann durch jede andere simuliert werden

- Warum haben wir dann drei verschiedene Schleifen?
  - Code-Lesbarkeit

- Aber das waren jetzt schon alle Schleifen, oder?

Funktionen in Java

- Funktionen sind in Java ähnlich aufgebaut wie in der Mathematik

$$\boldsymbol{f}\colon \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^2 + \boldsymbol{y}^2$$

```
1    int f(int x, int y) {
2      return x * x + y * y;
3    }
```

- Funktionen sind in Java ähnlich aufgebaut wie in der Mathematik

$$f\colon \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

$$f(x, y) = x^2 + y^2$$

- Bestandteile
  1. **Funktionsname**

```
1   int f(int x, int y) {
2     return x * x + y * y;
3   }
```

- Funktionen sind in Java ähnlich aufgebaut wie in der Mathematik

$$f: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$

$$f(x, y) = x^2 + y^2$$

- Bestandteile
  1. Funktionsname
  2. **Funktionskörper (oder Definition)**

```
1    int f(int x, int y) {
2      return x * x + y * y;
3    }
```

- Funktionen sind in Java ähnlich aufgebaut wie in der Mathematik

$$f\colon \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

$$f(x, y) = x^2 + y^2$$

- Bestandteile
  1. Funktionsname
  2. Funktionskörper (oder Definition)
  3. **Rückgabewert**

```
1    int f(int x, int y) {
2      return x * x + y * y;
3    }
```

- Funktionen sind in Java ähnlich aufgebaut wie in der Mathematik

$$f: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$$

$$f(x, y) = x^2 + y^2$$

- Bestandteile
  1. Funktionsname
  2. Funktionskörper (oder Definition)
  3. Rückgabewert
  4. **Parameter**

```
1   int f(int x, int y) {
2     return x * x + y * y;
3   }
```

Funktionen

- Jede Abfolge von Anweisungen kann zu einer Funktion zusammengefasst werden

- Sinnvoll, wenn der selbe Code mehrmals aufgerufen werden muss

- Erlaubt das Strukturieren von Programmcode

- Funktionen, die nichts zurückgeben, haben als Rückgabetyp `void`

Dr.-Ing. Christian Braune

Eine kleine Aufgabe

- Bestimmt das Maximum von zwei Zahlen `a` und `b`
- Die Zahlen sollen vom Typ `int` sein
- Welche Bestandteile brauchen wir?
  1. Funktionsname
  2. Parameter
  3. Rückgabewert
  4. Funktionskörper (oder Definition)

```
1  ___ ___(____________) {
2      __________
3          _________
4      ____
5          _________
6  }
```

Eine kleine Aufgabe

- Bestimmt das **Maximum** von zwei Zahlen `a` und `b`
- Die Zahlen sollen vom Typ `int` sein
- Welche Bestandteile brauchen wir?
  1. **Funktionsname**
  2. Parameter
  3. Rückgabewert
  4. Funktionskörper (oder Definition)

```
1   ___ max(____________) {
2     __________
3       _________
4     ____
5       _________
6   }
```


Dr.-Ing. Christian Braune
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF FAKULTÄT FÜR INFORMATIK

Eine kleine Aufgabe

- Bestimmt das **Maximum** von **zwei Zahlen a und b**
- Die Zahlen sollen vom **Typ `int`** sein
- Welche Bestandteile brauchen wir?
  1. **Funktionsname**
  2. **Parameter**
  3. Rückgabewert
  4. Funktionskörper (oder Definition)

```
1   ___ max(int a, int b) {
2     __________
3       _________
4     ____
5       _________
6   }
```

Eine kleine Aufgabe

- Bestimmt das **Maximum** von **zwei Zahlen a und b**
- Die Zahlen sollen vom **Typ `int`** sein
- Welche Bestandteile brauchen wir?
  1. **Funktionsname**
  2. **Parameter**
  3. **Rückgabewert**
  4. Funktionskörper (oder Definition)

```
1    int max(int a, int b) {
2      __________
3        _________
4      ____
5        _________
6    }
```

Eine kleine Aufgabe

- Bestimmt das **Maximum** von **zwei Zahlen a und b**
- Die Zahlen sollen vom **Typ int** sein
- Welche Bestandteile brauchen wir?
  1. **Funktionsname**
  2. **Parameter**
  3. **Rückgabewert**
  4. **Funktionskörper (oder Definition)**

```
1  int max(int a, int b) {
2    if (a > b)
3      return a;
4    else
5      return b;
6  }
```

Noch eine kleine Aufgabe...

- Jetzt machen wir das gleiche nochmal, aber dieses mal sollen die Zahlen vom Typ `double` sein
- Jetzt machen wir das gleiche nochmal, aber dieses mal sollen die Zahlen vom Typ `char` sein

```
1   char max(char a, char b) {
2     if (a > b)
3       return a;
4     else
5       return b;
6   }
```

- Der Name `max` käme jetzt in unserem Programm mehrfach vor. Darf das passieren?

Dr.-Ing. Christian Braune

Überladen

- Das Beispiel der letzten Folien ist ein Beispiel für das **Überladen** von Funktionen

- Die Funktionen

```
int max(int a, int b) { ... }
double max(double a, double b) { ... }
char max(char a, char b) { ... }
```

  haben zwar den gleichen Namen aber unterschiedliche **Signaturen** → Gelten für Java als unterschiedlich!

- Signatur = Name, Parameterliste und Rückgabewert

Dr.-Ing. Christian Braune

Call by Value vs Call by Reference

- Call-by-Value
  - Funktion erhält Kopien der Werte, die in den übergebenen Variablen gespeichert sind

```
1   public class Example2 {
2     static void doStuff(int x) {
3       x = x + 1;
4     }
5     public static void main(String[] args) {
6       int a = 5;
7       doStuff(a);
8       System.out.println("a hat den Wert " + a);
9     }
10  }
```


Dr.-Ing. Christian Braune

Call by Value vs Call by Reference

- Call-by-Reference
  - Funktion erhält eine Referenz auf den in der Variablen gespeicherten Wert

```
1   public class Example2 {
2     static void doStuff(int[] x) {
3       x[0] = 1;
4     }
5     public static void main(String[] args) {
6       int[] a = {5};
7       doStuff(a);
8       System.out.println("a[0] hat den Wert "+a[0]);
9     }
10  }
```

- Call-By-Value
  - wird für alle primitiven Datentypen verwendet (boolean, char, byte, short, int, long, float, double)
  - Kopiert den Wert
  - Erlaubt keine nach außen sichtbare Manipulation der Parameter

- Call-By-Reference
  - wird für alle anderen Datentypen verwendet
  - gibt eine Referenz auf die Daten zurück
  - erlaubt direkte Manipulation der Daten, sodass die Änderungen auch außerhalb der Funktion bestehen bleiben
  - spart möglicherweise aufwändiges Kopieren der Daten im Speicher

Ein erstes Programm

Wir spielen ein Spiel

- Der Reihe nach zählen wir von 1 bis 100
- Jeder sagt nur eine Zahl

- Wenn die Zahl durch 3 teilbar ist (3, 6, 9, ...), dann darf man die Zahl nicht sagen, sondern muss "FIZZ" sagen

- Wenn die Zahl durch 5 teilbar ist (6, 10, 15, ...), dann darf man die Zahl nicht sagen, sondern muss "BUZZ" sagen

- Und wenn die Zahl durch 3 und 5 teilbar ist, muss man "FIZZBUZZ" sagen

Wir spielen ein Spiel

- Das Spiel lassen wir jetzt unseren Computer spielen!

```
1   public class FizzBuzz {
2     public static void main(String[] args) {
3       ___ (_________; ________; ___) {
4         __ (__________) _________________________
5         __ (__________) _________________________
6         __ (________________________)
7           ____________________
8         _____________________
9       }
10    }
11  }
```


Dr.-Ing. Christian Braune

Otto von Guericke Universität Magdeburg INF Fakultät für Informatik

- Das Spiel lassen wir jetzt unseren Computer spielen!

```java
1   public class FizzBuzz {
2     public static void main(String[] args) {
3       for (int i = 1; i <= 100; i++) {
4         if (i % 3 == 0) System.out.print("FIZZ");
5         if (i % 5 == 0) System.out.print("BUZZ");
6         if (i % 3 != 0 && i % 5 != 0)
7           System.out.print(i);
8         System.out.println()
9       }
10    }
11  }
```