Java Intro

Michael Inden

Freiberuflicher Consultant und Trainer

https://github.com/Michaeli71/JAVA_INTRO

Speaker Intro

- Michael Inden, Jahrgang 1971
- Diplom-Informatiker, C.v.O. Uni Oldenburg

- ~8 ¼ Jahre **SSE** bei Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Jahre **TPL, SA** bei IVU Traffic Technologies AG in Aachen
- ~4 ¼ Jahre **LSA / Trainer** bei Zühlke Engineering AG in Zürich
- ~3 Jahre **TL / CTO** bei Direct Mail Informatics / ASMIQ in Zürich
- **Freiberuflicher Consultant, Trainer und Konferenz-Speaker**
- Autor und Gutachter beim dpunkt.verlag

E-Mail: michael.inden@hotmail.ch
Blog: https://jaxenter.de/author/minden

Kurse: Bitte sprecht mich an!

https://github.com/Michaeli71/JAVA_INTRO

Agenda

Workshop Contents

- **PART 1: Schnelleinstieg Java**
  - Java im Überblick
  - Erste Schritte in der JShell
  - Schnelleinstieg
    - Variablen
    - Operatoren
    - Fallunterscheidungen
    - Schleifen
    - Methoden
    - Rekursion

https://github.com/Michaeli71/JAVA_INTRO

Workshop Contents

- **PART 2: Strings**
  - Gebräuchliche String-Aktionen
  - Suchen und Ersetzen
  - Formatierte Ausgaben
  - Einstieg Reguläre Ausdrücke
  - Strings in Java 11
  - Mehrzeilige Strings (> Java 14)

- **PART 3: Arrays**
  - Gebräuchliche Array-Aktionen
  - Mehrdimensionale Arrays
  - Beispiel: Flood Fill

Workshop Contents

- **PART 4: Klassen & Objektorientierung**
  - Basics
  - Textuelle Ausgaben
  - Gleichheit == / `equals()`
  - Klassen ausführbar machen
  - Imports & Packages
  - Information Hiding
  - Vererbung und Overloading und Overriding
  - Die Basisklasse `Object`
  - Interfaces & Implementierungen

Workshop Contents

- **PART 5: Collections**
  - Schnelleinstieg Listen, Sets und Maps
  - Iteratoren
  - Generics
  - Basisinterfaces für Container
  - Praxisbeispiel Stack und Queue selbst gebaut
  - Sortierung – `sort()` + Comparator

Workshop Contents

- **PART 6: Ergänzendes Wissen**
  - Sichtbarkeits- und Gültigkeitsbereiche
  - Primitive Typen und Wrapper
  - Enums
  - `?`-Operator
  - `switch`
  - Besonderheiten in Schleifen `break` und `continue`
  - Vererbung und Polymorphie
  - Varianten innerer Klassen
  - Records

Workshop Contents

- **PART 7: Exception-Handling**
  - Schnelleinstieg
  - Exceptions selbst auslösen
  - Eigene Exception-Typen definieren
  - Propagation von Exceptions
  - Automatic Resource Management
  - Checked / Unchecked Exceptions

- **PART 8: Dateiverarbeitung**
  - Verzeichnisse und Dateien verwalten
  - Daten schreiben / lesen
  - CSV-Dateien einlesen

https://github.com/Michaeli71/JAVA_INTRO

Workshop Contents

- **PART 9: Einstieg in Lambdas und Streams**
  - Syntax von Lambdas
  - Lambdas im Einsatz mit `filter()`, `map()` und `reduce()`
  - Lambdas im Einsatz mit `Collectors.groupingBy()`
  - `takeWhile()` / `dropWhile()`

- **PART 10: Datumsverarbeitung**
  - Einführung Datumsverarbeitung
  - Zeitpunkte und die Klasse `LocalDateTime`
  - Datumswerte und die Klasse `LocalDate`
  - Zeit und die Klasse `LocalTime`

https://github.com/Michaeli71/JAVA_INTRO

PART 1:
Schnelleinstieg Java

Einführung Java

- Java ist eine objektorientierte Programmiersprache
- Mitte der 1990er Jahre von der Firma SUN entwickelt
- zwar auch schon mehr als 25 Jahre auf dem Buckel, wird aber nicht altersschwach, sondern kontinuierlich gepflegt und weiterentwickelt
- Seit Java 9 auf halbjährliche Releases umgestellt
- Am 14. September erscheint dann Java 17 als LTS Version

Java-Versionen im Überblick

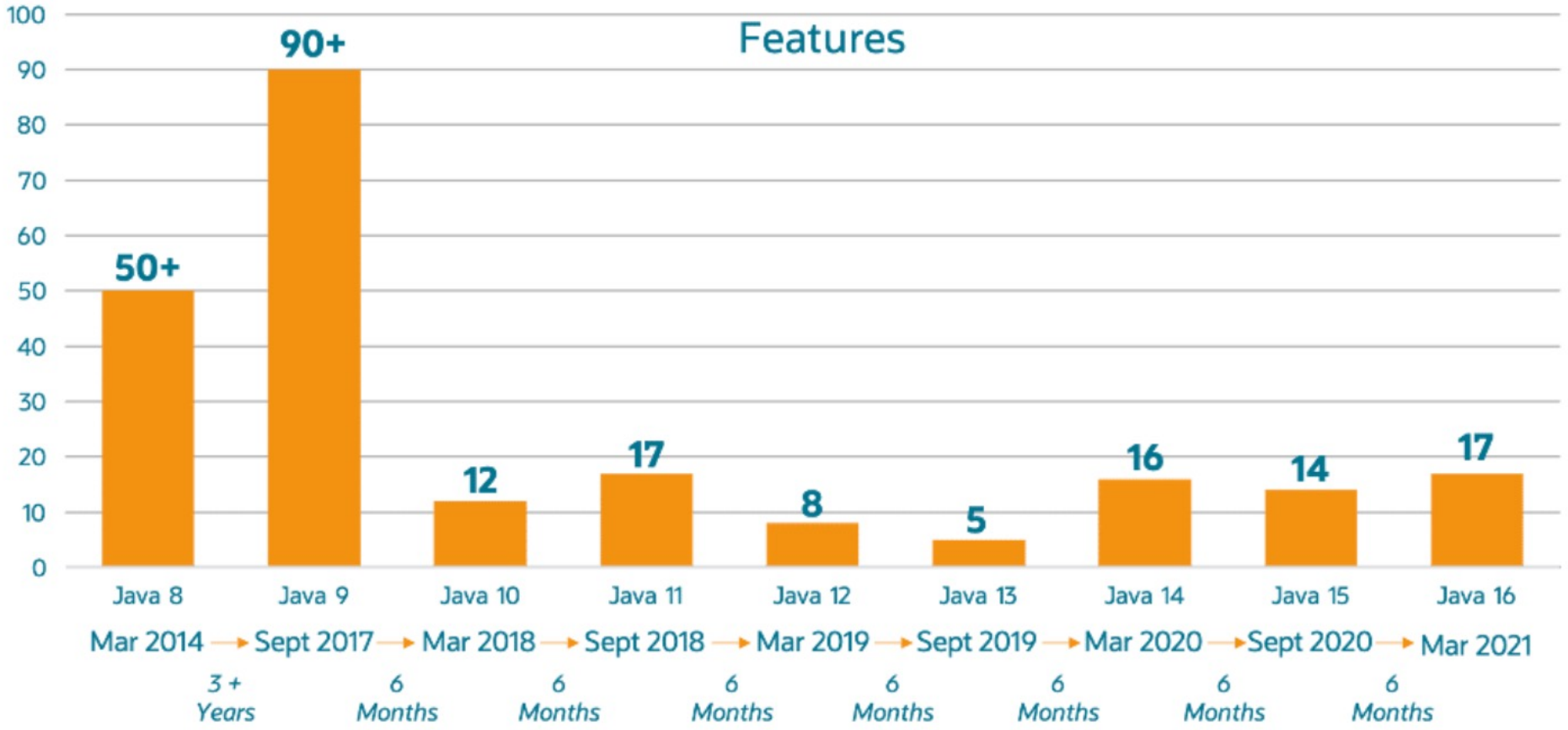
JDK	Release-Datum	Enwicklungszeit	LTS
Oracle JDK 8	**3 / 2014**	**-**	**Ja, *nun kommerziell***
Oracle JDK 9	9 / 2017	3,5 Jahre	-
Oracle JDK 10	3 / 2018	6 Monate	-
Oracle JDK 11	**9 / 2018**	**6 Monate**	**Ja, *kommerziell***
Oracle JDK 12	3 / 2019	6 Monate	-
Oracle JDK 13	9 / 2019	6 Monate	-
Oracle JDK 14	3 / 2020	6 Monate	-
Oracle JDK 15	9 / 2020	6 Monate	-
Oracle JDK 16	3 / 2021	6 Monate	**-**
Oracle JDK 17	**9 / 2021**	**6 Monate**	**Ja, *kommerziell***

Long-Term Support-Modell

- **alle drei Jahre Long Term Support (LTS) Release**
  - erhalten über längere Zeit Updates
  - Produktionsversionen
  - derzeit Java 8 und 11
  - nächstes LTS-Release ist (morgen) Java 17

- **andere Versionen sind "nur" Zwischenversionen**
  - erhalten nur 6 Monate Updates
  - Previews
  - Ideal um neue Features kennenzulernen und zum Experimentieren (vor allem privat)

Einordnung 6 monatiger Releasezyklus



https://www.techzine.eu/news/devops/57122/oracle-releases-version-16-of-java/

Versionsüberblick -- https://javaalmanac.io/

The Java Version Almanac

Collection of information about the history and future of Java.

Details	Status	Documentation	Download	Compare API to										
Java 18	DEV	API | Notes	JDK | JRE	17	16	15	14	13	12	11	10	9	8	...
Java 17	DEV	API | Notes	JDK | JRE	16	15	14	13	12	11	10	9	8	7	...
Java 16	REL	API | Lang | VM | Notes	JDK | JRE	15	14	13	12	11	10	9	8	7	6	...
Java 15	EOL	API | Lang | VM | Notes	JDK | JRE	14	13	12	11	10	9	8	7	6	5	...
Java 14	EOL	API | Lang | VM | Notes	JDK | JRE	13	12	11	10	9	8	7	6	5	1.4	...
Java 13	EOL	API | Lang | VM | Notes	JDK | JRE	12	11	10	9	8	7	6	5	1.4	1.3	...
Java 12	EOL	API | Lang | VM | Notes	JDK | JRE	11	10	9	8	7	6	5	1.4	1.3	1.2	...
Java 11	LTS	API | Lang | VM | Notes	JDK | JRE	10	9	8	7	6	5	1.4	1.3	1.2	1.1	
Java 10	EOL	API | Lang | VM | Notes	JDK | JRE	9	8	7	6	5	1.4	1.3	1.2	1.1		
Java 9	EOL	API | Lang | VM | Notes	JDK | JRE	8	7	6	5	1.4	1.3	1.2	1.1			
Java 8	LTS	API | Lang | VM | Notes	JDK | JRE	7	6	5	1.4	1.3	1.2	1.1				
Java 7	EOL	API | Lang | VM | Notes	JDK | JRE	6	5	1.4	1.3	1.2	1.1					
Java 6	EOL	API | Lang | VM | Notes	JDK | JRE	5	1.4	1.3	1.2	1.1						
Java 5	EOL	API | Lang | VM | Notes		1.4	1.3	1.2	1.1							
Java 1.4	EOL	API		1.3	1.2	1.1								
Java 1.3	EOL	API		1.2	1.1									
Java 1.2	EOL	API | Lang		1.1										
Java 1.1	EOL	API												
Java 1.0	EOL	API | Lang | VM												

Data Source

Lizenzmodell

Lizenzmodell

- **Sofern Sie planen, Ihre Software kommerziell vertreiben (wollen), sollten Sie beim Herunterladen von Java 11 unbedingt die neue Release-Politik von Oracle beachten!**
- **Das Oracle JDK ist nun leider für einige Szenarien kostenpflichtig – während der Entwicklung kann es allerdings weiterhin kostenfrei nutzen.**
- **Alternativen: OpenJDK (https://openjdk.java.net/) oder Adopt Open JDK (https://adoptopenjdk.net/)**

Java SE Development Kit 11 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Important changes in Oracle JDK 11 License

With JDK 11 Oracle has updated the license terms on which we offer the Oracle JDK.
The new Oracle Technology Network License Agreement for Oracle Java SE is substantially different from the licenses under which previous versions of the JDK were offered. Please review the new terms carefully before downloading and using this product.

Oracle also offers this software under the GPL License on jdk.java.net/11

Nach 25 Jahren Java: Wo geht die Reise hin und wie sieht es aktuell mit der Konkurrenz aus?

Zunächst ein Blick zurück (Aug 21)

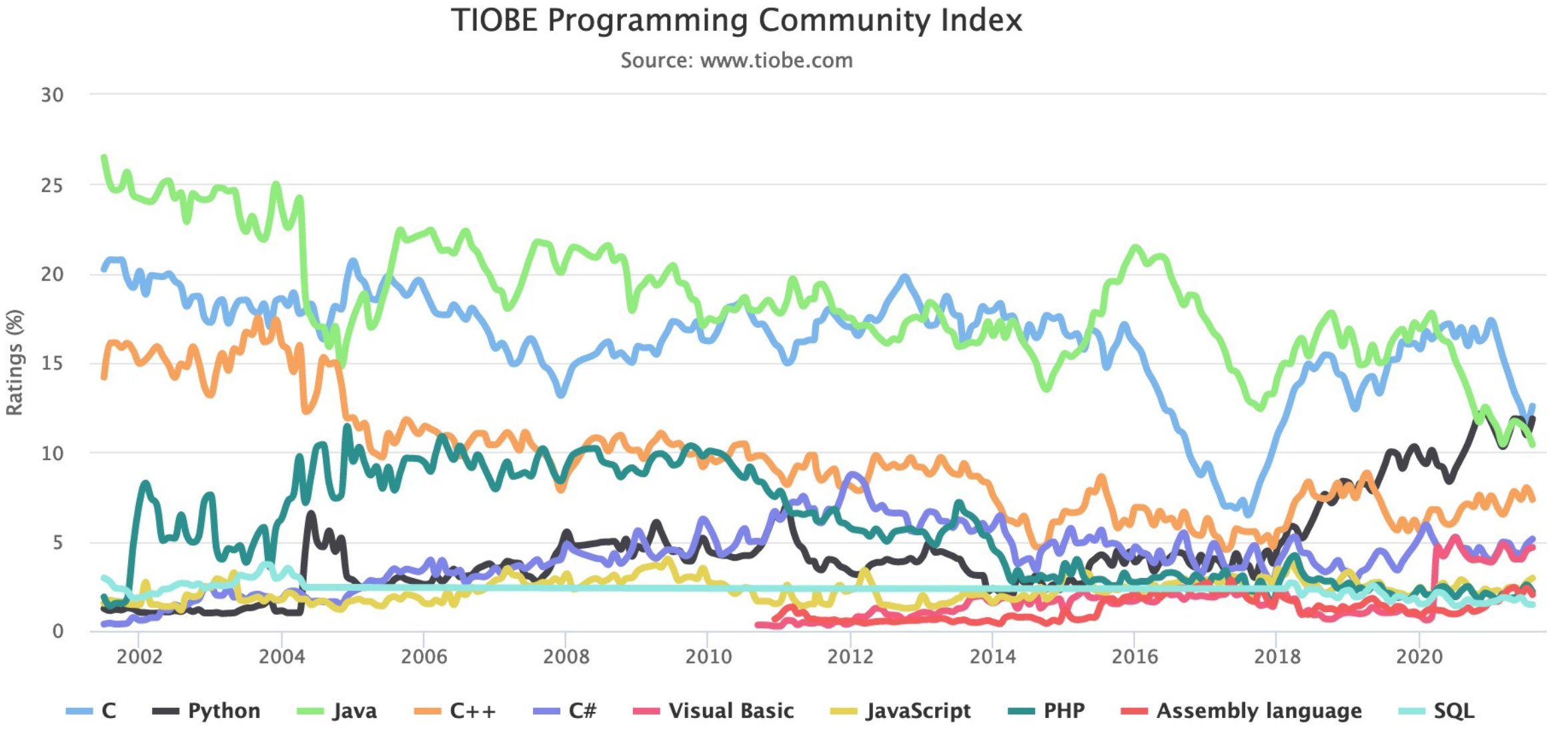
Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2021	2016	2011	2006	2001	1996	1991	1986
C	1	2	2	2	1	1	1	1
Java	2	1	1	1	3	18	-	-
Python	3	5	6	8	26	24	-	-
C++	4	3	3	3	2	2	2	6
C#	5	4	5	7	13	-	-	-

https://www.tiobe.com/tiobe-index/

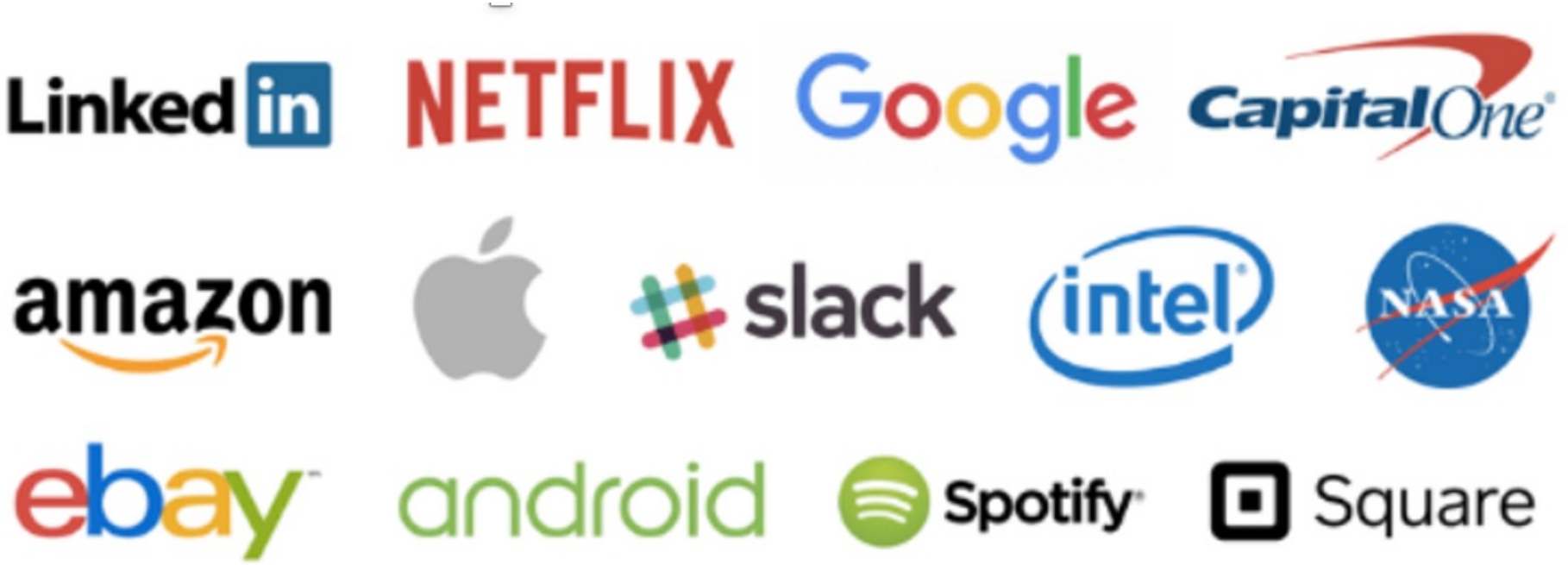
Popularitätstrends der Top 10 Programmiersprachen (Aug 21)



https://www.tiobe.com/tiobe-index/

Nutzung

- **Wer nutzt Java?**



https://trio.dev/blog/python-vs-java

Einsatzgebiete

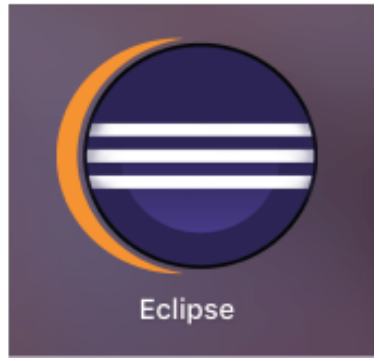
- Breit gefächertes Einsatzspektrum
- Früher auch viel GUI, jetzt vor allem im Backend

- Alle Arten von Businessapplikationen
- Webapplikationen
- Sogar Datenbanken

- Android-Apps
- Spiele wie Minecraft

IDE & Tool Support für Java 16

- **Aktuelle IDEs & Tools grundsätzlich gut**



- **Eclipse: Version 2021-06**
- **IntelliJ: Version 2021.X**

- **Maven: 3.8.1, Compiler-Plugin: 3.8.1**
- **Gradle: 7.x**

Maven™

Gradle

Erste Schritte (mit der JShell)

Bestandteile von Java-Programmen

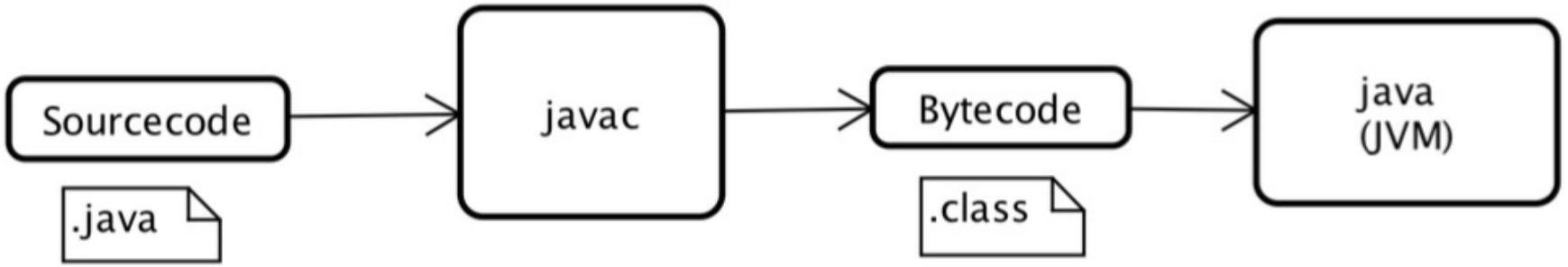
- Syntax & Semantik
- Analog zum Deutschen Schlüsselworte und Grammatik
- Java-Programm bestehen aus „Texten“
- Geschweifte Klammern markieren „Absätze“, gruppieren Java-Anweisungen
- Reine Textangaben in Hochkommata

```
package ch01_quickstart;

public class MyFirstJavaProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```

Bestandteile von Java-Programmen

- Java wird aus Text in spezielle Befehle (Bytecode) übersetzt



- `javac MyFirstJavaClass.java`
- `java MyFirstJavaClass`

- Java wird in einer Laufzeitumgebung, der Java Virtual Machine (JVM), ausgeführt. Das ist ein Computer im Computer.

- Anfangs etwas (zu) komplex => wie geht es einfacher?

Kommandozeileninterpreter

- **Java – JShell**

```
$ jshell
|  Welcome to JShell -- Version 16.0.1
|  For an introduction type: /help intro

jshell>
```

- **Hier können wir erstmal einzelne Java-Kommandos für sich ausprobieren**
- **REPL – Read-Eval-Print-Loop**
- **Etwas fehlertoleranter als «Standard»-Java**

Kommandozeileninterpreter

- **Java – JShell**

```
jshell> 7 * 2
$5 ==> 14

jshell> System.out.print("Hello JAVA")
Hello JAVA

jshell> "Michael " + "mag " + "Krimis"
$6 ==> "Michael mag Krimis"

jshell> "MOIN".repeat(3)
$7 ==> "MOINMOINMOIN"
```

Java-Kommandozeile jshell

- Codeausführung ohne Klassen- und Methodendeklaration
- Semikolon am Zeilenende kann (teilweise) entfallen
- für jeden Befehl wird automatisch eine Variable für den Rückgabewert angelegt ($1, $2, …)
- Deklaration von Methoden und Klassen möglich
- **`jshell`** verlassen: **`/exit`**

Java-Kommandozeile jshell

- Befehlshistorie
  - `/!` wiederholt den letzten Befehl
  - `/list` Liste aller eingegebenen Befehle
  - `/<nr>` Ausführen des Befehls mit der angegebenen Nummer
  - `/reset` löscht Historie
  - `/methods` zeigt Methoden
  - `/imports` zeigt imports
  - `/help` zeigt Hilfe

DEMO

Wo kriege ich Hilfe, wenn es mal nicht weitergeht?

Nützliche Quellen für die Soforthilfe

Online:

- https://stackoverflow.com
- https://www.baeldung.com/
- http://tutorials.jenkov.com/java/index.html
- https://www.google.com/

Offline

- «Einfach Java»
- «Der Weg zum Java-Profi»

Exercises Part 1 – Aufgabe 1 & 2

https://github.com/Michaeli71/JAVA_INTRO

Schnelleinstieg

Variablen und Datentypen

Basisdatentypen

- `String` – Textuelle Informationen, wie z. B. "Hallo". String-Werte sind von doppelten Anführungszeichen eingeschlossen
- `char` – einzelnes Zeichen, in einzelnen Anführungszeichen ‘A’
- `int` / long – Ganzzahlen wie 123, oder -4711
- `float / double` – Gleitkommazahlen, als mit Vor- und Nachkommastellen, wie 72.71 oder -1.357
- `boolean` – Wahrheitswerte als wahr oder falsch, `true` oder `false`.

Basisdatentypen

- **Definition von Variablen**
  - `Typ Name = Wert;`
  - ```
    int age = 40
    age = 50
    System.out.println(age)
    ```
- **Besonderheit Deklaration**

```
int age;
age = 50
```
```

- **Definition von Konstanten (`final`)**

```
public class ConstantsExample
{
    public static void main(String[] args)
    {
        final int age = 40;
        // The final local variable age cannot be assigned.
        // It must be blank and not using a compound assignment
        age = 50;
    }
}
```

- **Namensgebung von Variablen**

  - Namen sollten mit einem Buchstaben beginnen – Ziffern sind als erstes Zeichen eines Variablennamens nicht erlaubt.
  - Namen können danach aus einem beliebigen Mix aus Buchstaben (auch Umlauten), Ziffern, $ und _ bestehen, dürfen aber keine Leerzeichen enthalten.
  - Enthalten Namen mehrere Wörter, dann ist es guter Stil, die sogenannte CamelCase-Schreibweise zu verwenden. Bei dieser beginnt das erste Wort mit einem Klein-buchstaben und danach startet jedes neue Wort mit einem Großbuchstaben, etwa arrivalTime, estimatedDuration, shortDescription.
  - Die Groß- und Kleinschreibung von Namen spielt eine Rolle: arrivalTime und arrivaltime bezeichnen unterschiedliche Variablen.
  - Namen dürfen nicht mit den in Java vordefinierten Schlüsselwörtern übereinstimmen, demzufolge sind public, class oder int keine gültigen Namen für Variablen. PublicTransport, classic oder Winter hingegen sind erlaubt.

- **Namensgebung von Variablen**

```
// Gut verständliche Namen
int minutesPerHour = 60
int daysPerYear = 365
int maxTableRows = 25

// NAJA, oft schwieriger verständlich, was die Abkürzungen bedeuten
int m = 60;
int dpy = 365
int mtr = 25

// ACHTUNG: ziemlich schwierig unterscheidbar
String arrivalTime = "16:50"
double arrivaltime = 16.50
```

Basisdatentypen

```
jshell> int answer = 42
answer ==> 42

jshell> float marathonDistance = 42.195f
marathonDistance ==> 42.195

jshell> String name = "Michael"
name ==> "Michael»

jshell> int bigNumber = 2_000_000_000_000
|  Error:
|  integer number too large
|  int bigNumber = 2_000_000_000_000;
|                  ^

jshell> long bigNumber = 2_000_000_000_000l
bigNumber ==> 2000000000000
```

Local Variable Type Inference => var

- Local Variable Type Inference

- **Neues reserviertes Wort `var` zur Definition lokaler Variablen, statt expliziter Typangabe**

```
var name = "Peter";                    // var => String
var chars = name.toCharArray();        // var => char[]

var mike = new Person("Mike", 47);     // var => Person
var hash = mike.hashCode();            // var => int
```

- **möglich, sofern der konkrete Typ für eine lokale Variable anhand der Definition auf der rechten Seite der Zuweisung vom Compiler ermittelt werden kann**

Basisdatentypen – Kurzschreibweise var

```
jshell> var answer = 42
answer ==> 42

jshell> var marathonDistance = 42.195f
marathonDistance ==> 42.195

jshell> var name = "Michael"
name ==> "Michael»

jshell> int bigNumber = 2_000_000_000_000
|  Error:
|  integer number too large
|  int bigNumber = 2_000_000_000_000;
|                  ^

jshell> var bigNumber = 2_000_000_000_000l
bigNumber ==> 2000000000000
```

Operatoren

- **Arithmetische Operatoren**

```
jshell> 5 + 4 - 2
$191 ==> 7

jshell> 7 * 2
$192 ==> 14

jshell> 14 / 3
$193 ==> 4

jshell> 14 % 3
$194 ==> 2

jshell> 5 * 7 - 14 / 2
$195 ==> 28

jshell> 5 * (7 - 2) / 2
$196 ==> 12
```

- **Inkrement / Dekrement**

```
jshell> int i = 7
i ==> 7

jshell> i = i + 1
i ==> 8

jshell> i++
$222 ==> 8

jshell> ++i
$223 ==> 10

jshell> --i
$224 ==> 9

jshell> i--
$225 ==> 9

jshell> i
i ==> 8
```

- **Zuweisungsoperatoren: Kurzform für z. B. `x = x + 1` => `x += 1`**

```
jshell> int counter = 0
counter ==> 0

jshell> counter += 7
$228 ==> 7

jshell> counter -= 2
$229 ==> 5

jshell> counter *= 10
$230 ==> 50

jshell> counter /= 2
$231 ==> 25

jshell> counter %= 11
$232 ==> 3
```

- **«Rechnen» mit Zeichen**

```
jshell> char a = 'A'
a ==> 'A'

jshell> char b = (char)(a + 1)
b ==> 'B'

jshell> char result = (char)(a + 7)
result ==> 'H'
```

- **Vergleichsoperatoren**

Operator	Name	Beschreibung
==	gleich	x == y
!=	ungleich	x != y
>	größer	x > y
>=	größer gleich	x >= y
<	kleiner	x < y
<=	kleiner gleich	x <= y

- **Logische Operatoren**

Operator	Name	Beschreibung
`&&`	AND	`true`, wenn beide Bedingungen erfüllt sind
`||`	OR	`true`, wenn mindestens eine der Bedingungen erfüllt ist
!	NOT	`true`, wenn die Bedingung nicht erfüllt ist

```
jshell> int age = 25
age ==> 25

jshell> int points = 1234
points ==> 1234

jshell> age > 20 && points > 1000
$235 ==> true

jshell> age > 20 || points > 2000
$236 ==> true
```

- **Logische Operatoren**

Operator	Name	Beschreibung
`&&`	AND	`true`, wenn beide Bedingungen erfüllt sind
`||`	OR	`true`, wenn mindestens eine der Bedingungen erfüllt ist
`!`	NOT	`true`, wenn die Bedingung nicht erfüllt ist

```
jshell> !points > 500
|  Error:
|  bad operand type int for unary operator
'!'
|  !points > 500
|  ^-----^

jshell> !(points > 500)
$237 ==> false
```

Fallunterscheidungen

Fallunterscheidungen: Bedingte Ausführung

```
if (condition)
{
    // bedingt auszuführende Anweisungen
}
```

```
jshell> if (6 * 7 == 42)
   ...>     System.out.println("Answer found")
Answer found
```

```
jshell> if (age >= 18)
   ...>     System.out.println("You are allowed to drive a car")
You are allowed to drive a car
```

Fallunterscheidungen: Bedingte Ausführung mit Alternative

```
if (condition)
{
    // bedingt auszuführende Anweisungen
}
else
{
    // Anweisungen für den Negativfall
}
```

Fallunterscheidungen: Bedingte Ausführung mit Alternative

```
jshell> int age = 50
age ==> 50

jshell> if (age >= 18) {
   ...>     System.out.println("You are allowed to drive a car");
   ...> } else {
   ...>     System.out.println("TOO YOUNG!");
   ...> }
You are allowed to drive a car

jshell> age = 12
age ==> 12

jshell> if (age >= 18) {
   ...>     System.out.println("You are allowed to drive a car");
   ...> } else {
   ...>     System.out.println("TOO YOUNG!");
   ...> }
TOO YOUNG!
```

Fallunterscheidungen: Bedingte Ausführung mit Alternative(n)

```
jshell> int hour = 15
hour ==> 15

jshell> if (hour < 12) {
   ...>     System.out.println("Good morning");
   ...> } else if (hour < 18) {
   ...>     System.out.println("Good afternoon");
   ...> } else if (hour < 22) {
   ...>     System.out.println("Good evening");
   ...> } else {
   ...>     System.out.println("Good night");
   ...> }
Good afternoon
```

Kommentare

Kommentare

```
jshell> int age = 11  // Kommentar: age = 42 wird nicht ausgeführt

jshell> String value = /* Zwischen-Kommentar */ "WERT"
value ==> "WERT"

jshell> /* Blockkommentar
   ...> * Zeile 2
   ...> * Zeile 3
   ...> */

jshell> /**
   ...>  * Javadoc-Kommentar-Stil
   ...>  */
```

Methoden

- **Methoden definieren**

```
rückgabetyp methodenName(typ1 parameter1, typ2 parameter2, ...)
{
   Anweisungen
}
```

- **Methoden aufrufen**

```
methodenName(parameterWert1, parameterWert2)
```

Methoden definieren

- **Minimum aus 3 Werten**

```
static int min_of_3(int x, int y, int z)
{
    if (x < y)
    {
        if (x < z)
        {
            return x;
        }
        else
        {
            return z;
        }
    }
    else if (y < z)
    {
        return y;
    }
    else
        return z;
}
```

Methoden definieren

- **Basierend auf Built-in-Methode `Math.min()`**

```
jshell> int min_of_3(int x, int y, int z)
   ...> {
   ...>     return Math.min(x, Math.min(y, z));
   ...> }
|  created method min_of_3(int,int,int)

jshell> min_of_3(7, 2, 5)
$202 ==> 2
```

Methoden definieren – Var Args

- Summenberechnung inklusive der Ausgabe einer Meldung.
- Die einzelnen Werte lassen sich praktischerweise sehr leicht mit einer for-Schleife durchlaufen:

```
static String var_args_sum(String info, int... args)
{
    int result = 0;
    for (int num : args)
        result += num;

    return info + String.valueOf(result);
}
```

- Am Beispiel sieht man, dass neben den Var Args weitere Parameter möglich sind. Allerdings muss der Var Arg-Parameter am Ende stehen.

Schleifen

Schleifen

- **Indexbasierte for-Schleife**

```
jshell> for (int i = 2; i < 5; i++)
   ...>     System.out.println("Durchlauf: " + i)
Durchlauf: 2
Durchlauf: 3
Durchlauf: 4
```

- **Indexbasierte for-Schleife Countdown**

```
jshell> for (int i = 3; i >= 0; i--)
   ...>     System.out.println(i)
3
2
1
0
```

Schleifen

- **Indexbasierte for-Schleife**

```
jshell> String[] cities = {"Kiel", "Bremen", "Zürich", "Basel"}
cities ==> String[4] { "Kiel", "Bremen", "Zürich", "Basel" }

jshell> for (int i = 0; i < cities.length; i++)
   ...>     System.out.println(cities[i])
Kiel
Bremen
Zürich
Basel
```

- **For-:-VALUES**

```
jshell> for (String currentName : List.of("Barbara", "Lilija", "Sophie"))
   ...>     System.out.println(currentName)
Barbara
Lilija
Sophie
```

Schleifen

- `while <cond>`

```
jshell> int i = 0
i ==> 0

jshell> while (i < 5) {
   ...>     System.out.println("Loop: " + i);
   ...>     i += 1;
   ...> }
Loop: 0
Loop: 1
Loop: 2
Loop: 3
Loop: 4
```

Schleifen

```
do
{
    // Aktionen
}
while <cond>;
```

```
jshell> int i = 0;
i ==> 0

jshell> do {
   ...>     System.out.println("i: " + i);
   ...>     i++;
   ...> }
   ...> while (i < 5);
i: 0
i: 1
i: 2
i: 3
i: 4
```

Syntax im Kurzüberblick

Syntax Quick Recap

	Java

Variablen	`jshell> int age = 50`
	`age ==> 50`
	
	`jshell> long durationInMs = 2745`
	`durationInMs ==> 2745`
	
	`jshell> String message = "Hello JAX"`
	`message ==> "Hello JAX"`

String-Add	`jshell> "The answer is " + 42 + " but " + false` **`$9 ==> "The answer is 42 but false"`**
Ausgaben	`jshell> System.out.println(age + " " + durationInMs + " " + message)` **`50 2745 Hello JAX`**

Syntax Quick Recap

	Java
for-Schleife	`jshell> for (int i = 0; i < 10; i++)` `   ...>     System.out.println("i: " + i);`

Blöcke	`if (a > b) {`
	`    System.out.println("a > b")`
	`}`

Logik Ops	`&&   ||   !`

Miniprogramm	`public class SimpleExample {`
	`    public static void main(String[] args)`
	`    {`
	`        System.out.println("7*2=" + (7*2);`
	`    }`
	`}`

Listen	`List numbers = List.of(0,1,2,3,4)`

Exercises Part 1

https://github.com/Michaeli71/JAVA_INTRO

PART 2: Strings

- Wichtige Funktionen / Methoden
- Formatierte Ausgaben
- Erweiterungen in Java 11
- Mehrzeilige Strings

Strings

- **Definition**

```
jshell> var str = "Double Quoted String"
str ==> "Double Quoted String"

jshell> var strWithSingleQuotes = "Double 'Quoted' String"
strWithSingleQuotes ==> "Double 'Quoted' String"
```

- **Unicode (\u)**

```
jshell> var strUnicode = "\u0333\u0424"
strUnicode ==> "̳Ф"
```

- **ABER: Unicode mit Surrogates krampfig**

```
jshell> new String(Character.toChars(0x0001F601))
$34 ==> "😁"
```

Strings

- ## Groß- und Kleinschreibung

```
jshell> var message = "IMPORTANT: Please consult the doctor"
message ==> "IMPORTANT: Please consult the doctor"

jshell> message.toUpperCase()
$24 ==> "IMPORTANT: PLEASE CONSULT THE DOCTOR"

jshell> message.toLowerCase()
$25 ==> "important: please consult the doctor"
```

- ## Unveränderlichkeit!!!

```
jshell> message
message ==> "IMPORTANT: Please consult the doctor"
```

Strings

- **Inhalt vergleichen (ohne Beachtung von Groß-/Kleinschreibung)**

```
jshell> var text1 = "HALLO Peter"
text1 ==> "HALLO Peter"

jshell> var text2 = "HaLlO PeteR"
text2 ==> "HaLlO PeteR"

jshell> text1.equals(text2)
$90 ==> false

jshell> text1.equalsIgnoreCase(text2)
$91 ==> true
```

Strings

- **Konkatenation (+)**

```
jshell> var lastName = "Inden"
lastName ==> "Inden"

jshell> "Michael" + " " + lastName
$41 ==> "Michael Inden"
```

- **Geht auch mit Zahlen:**

```
jshell> "Bitte " + 2 + " mal klingeln"
$42 ==> "Bitte 2 mal klingeln"
```

- **ABER:**

```
jshell> "Länge = " + 47 - 5 + "cm"
|  Error:
|  bad operand types for binary operator '-'
|    first type:  java.lang.String
|    second type: int
```

Strings

- **trim() / strip()**

```
jshell> var msg = "   This text has blanks at the beginning and the end   "
msg ==> "   This text has blanks at the beginning and the end   "

jshell> msg.trim()
$47 ==> "This text has blanks at the beginning and the end"

jshell> msg.strip()
$48 ==> "This text has blanks at the beginning and the end"
```

Wieso gibt es 2 Varianten?

Strings

- **trim() / strip()**

```
jshell> var specialSpace = " \u2009 MID \u2009 "
specialSpace ==> "   MID   "

jshell> specialSpace.trim()
$93 ==> "  MID  "

jshell> specialSpace.strip()
$94 ==> "MID"
```

Strings

- **Länge ermitteln – `length()`**

```
jshell> "This is a short message".length()
$50 ==> 23
```

- **Leerstring?**

```
jshell> var noContent = ""
noContent ==> ""

jshell> noContent.length() == 0
$53 ==> true
```

- **Prüfungsvariante**

```
jshell> noContent.isEmpty()
$54 ==> true
```

Strings

- **Prüfungsvariante**

```
jshell> noContent.isBlank()
$55 ==> true
```

- **String nur mit Leerzeichen usw.**

```
jshell> var noRealContent = "   "
noRealContent ==> "   "

jshell> noRealContent.isEmpty()
$57 ==> false

jshell> noRealContent.isBlank()
$58 ==> true
```

Strings

- **Auf Zeichen zugreifen**

```
jshell> var content = "This is 1 short message"
content ==> "This is 1 short message"

jshell> content.charAt(0)
$63 ==> 'T'

jshell> content.charAt(3)
$64 ==> ‘s’

jshell> content.charAt(content.length() - 1)
$74 ==> 'e'

jshell> content.charAt(content.length())
|  Exception java.lang.StringIndexOutOfBoundsException: String index out of range:
23
```

Strings

- **Zeichen oder Ziffern?**

```
jshell> var content = "This is 1 short message"
content ==> "This is 1 short message"

jshell> Character num = content.charAt(8)
num ==> '1'

jshell> Character.isDigit(num)
$70 ==> true

jshell> Character.isLetterOrDigit(num)
$71 ==> true

jshell> Character.isLetter(num)
$72 ==> false
```

- **toUpperCase(), toLowerCase(), isWhitespace(), ...**

Strings

- **String positionsbasiert durchlaufen**

```
jshell> for (int i = 0; i < message.length(); i++)
   ...>     System.out.print(i + " " + message.charAt(i))
0 J1 a2 v3 a4  5 h6 a7 s8  9 s10 e11 v12 e13 r14 a15 l16  17 l18 o19 o20
p21  22 v23 a24 r25 i26 a27 n28 t29 s
```

- **Zeichenbasiert**

```
jshell> for (char ch : message.toCharArray())
   ...>     System.out.print(ch)
Java has several loop variants
```

Strings

- **Enthaltensein (`contains()`)**

```
jshell> var msg = "Tim arbeitet in Kiel. Michael lebt in Zürich"
msg ==> "Tim arbeitet in Kiel. Michael lebt in Zürich"

jshell> msg.contains("Michael")
$80 ==> true

jshell> msg.contains("arbeitet")
$82 ==> true

jshell> msg.contains("Bremen")
$83 ==> false
```

Strings

- **Start und Ende prüfen**

```
jshell> var msg = "Important Info"
msg ==> "Important Info"

jshell> msg.startsWith("Impo")
$86 ==> true

jshell> msg.endsWith("Info")
$87 ==> true
```

Strings

- **Suchen**

```
jshell> var msg = "This is a tiny story. This tiny text ends now."
msg ==> "This is a tiny story. This tiny text ends now."

jshell> msg.indexOf("This")
$186 ==> 0

jshell> msg.indexOf("This", 10)
$187 ==> 22

jshell> msg.lastIndexOf("tiny")
$188 ==> 27

jshell> msg.lastIndexOf("tiny", 20)
$189 ==> 10

jshell> msg.indexOf("Michael")
$190 ==> -1
```

Strings Besonderheiten

- **Strings wiederholen (`repeat()`)**

```
>>> var greeting = "MOIN"
>>> greeting.repeat(2)
'MOINMOIN’

>>> var nonsens = "BLA"
>>> nonsens.repeat(3)
'BLABLABLA'
```

Strings Besonderheiten

- **Strings aufspalten**

```
jshell> var timestamp = "11:22:33"
timestamp ==> "11:22:33"

jshell> timestamp.split(":")
$179 ==> String[3] { "11", "22", "33" }
```

- **Strings zusammenführen und ersetzen**

```
jshell> String.join(":", "11", "22", "33")
$180 ==> "11:22:33"


jshell> "One-Two-Three-Four".replace("-", "...")
$181 ==> "One...Two...Three...Four"
```

Strings Besonderheiten

- **Strings aufspalten**

```
jshell> var date = "13.09.2021"
date ==> "13.09.2021"

jshell> date.split(".")
$183 ==> String[0] {  }

jshell> date.split("\\.")
$184 ==> String[3] { "13", "09", "2021" }
```

Strings Besonderheiten

- **Teilbereiche extrahieren**

```
jshell> var info = "Dies ist eine Info"
info ==> "Dies ist eine Info"

jshell> info.substring(5)
$207 ==> "ist eine Info"

jshell> info.substring(5, 8)
$208 ==> "ist"
```

Strings – Formatierte Ausgabe

- **Verschiedene Varianten**

```
jshell> var product = "Apple iMac";
   ...> var price = 3699;
   ...>
   ...> System.out.println(product + " costs " + price);
   ...> System.out.println(String.format("%s costs %d", product, price));
   ...> System.out.println("%s costs %d".formatted(product, price));
   ...> System.out.printf("%s costs %d", product, price);
product ==> "Apple iMac"
price ==> 3699
Apple iMac costs 3699
Apple iMac costs 3699
Apple iMac costs 3699
Apple iMac costs 3699$177 ==> java.io.PrintStream@136432db
```

Strings

- **Mehrzeilige Strings (""")**

```
jshell> var multi_line_string = """
   ...>      Line 1
   ...>      Line 2
   ...>      Line 3
   ...>      """
multi_line_string ==> "    Line 1\n    Line 2\n    Line 3\n"
```

- **Mehrzeilige Strings können einfache Anführungszeichen enthalten**

```
jshell> var multi_line_string_with_quotes = """
   ...>      Line 1
   ...>      the "second" line contains 'quotes'
   ...>      last line"""
multi_line_string_with_quotes ==> "Line 1\nthe \"second\" line contains
'quotes'\nlast line"
```

Erweiterung in der Klasse `String` in Java 11

Erweiterung in `java.lang.String`

- **Die Klasse String existiert seit JDK 1.0 und hat zwischenzeitlich nur wenige API-Änderungen erfahren.**

- **Mit Java 11 ändert sich das. Es wurden folgende Methoden neu eingeführt:**

  - `isBlank()`
  - `lines()`
  - `repeat(int)`
  - `strip()`
  - `stripLeading()`
  - `stripTrailing()`

Erweiterung in `java.lang.String: isBlank()`

- **Für Strings war es bisher mühsam oder mithilfe von externen Bibliotheken möglich, zu prüfen, ob diese nur Whitespaces enthalten.**
- **Dazu wurde mit Java 11 die Methode `isBlank()` eingeführt, die sich auf `Character.isWhitespace(int)` abstützt.**

```
private static void isBlankExample()
{
	final String exampleText1 = "";
	final String exampleText2 = "    ";
	final String exampleText3 = " \n \t ";

	System.out.println(exampleText1.isBlank());
	System.out.println(exampleText2.isBlank());
	System.out.println(exampleText3.isBlank());
}
```

- **Alle geben `true` aus.**

Erweiterung in java.lang.String: lines()

- **Beim Verarbeiten von Daten aus Dateien müssen des Öfteren Informationen in einzelne Zeilen aufgebrochen werden. Dazu gibt es etwa die Methode `Files.lines(Path)`.**
- **Ist die Datenquelle allerdings schon ein String, gab es diese Funktionalität bislang noch nicht. JDK 11 bietet die Methode `lines()`, die einen `Stream<String>` zurückliefert:**

```
private static void linesExample()
{
        final String exampleText = "1 This is a\n2 multi line\r“ +
                                   "3 text with\r\n4 four lines!";
        final Stream<String> lines = exampleText.lines();
        lines.forEach(System.out::println);
}

=>

1 This is a
2 multi line
3 text with
4 four lines!
```

Erweiterung in `java.lang.String: repeat()`

- **Immer mal wieder steht man vor der Aufgabe, einen String mehrmals aneinanderzureihen, also einen bestehenden String n-Mal zu wiederholen.**
- **Dazu waren bislang eigene Hilfsmethoden oder solche aus externen Bibliotheken nötig. Mit Java 11 kann man stattdessen die Methode `repeat(int)` nutzen:**

```
private static void repeatExample()
{
        final String star = "*";
        System.out.println(star.repeat(30));

        final String delimeter = " -*- ";
        System.out.println(delimeter.repeat(6));
}

=>

******************************
 -*-  -*-  -*-  -*-  -*-  -*-
```

Erweiterung in `java.lang.String`: `repeat()` Corner Cases

```
"ERROR".repeat(-1);
```

```
"ERROR".repeat(Integer.MAX_VALUE);
```

Was passiert?

Erweiterung in java.lang.String: repeat() Corner Cases

```
"ERROR".repeat(-1);
```

```
Exception in thread "main" java.lang.IllegalArgumentException: count is negative: -1
at java.base/java.lang.String.repeat(String.java:3149)
at Java11Examples/snippet.Snippet.main(Snippet.java:16)
```

```
"ERROR".repeat(Integer.MAX_VALUE);
```

```
Exception in thread "main"
java.lang.OutOfMemoryError: Repeating 5 bytes String 2147483647 times will
produce a String exceeding maximum size.
at java.base/java.lang.String.repeat(String.java:3164)
at Java11Examples/snippet.Snippet.main(Snippet.java:14)
```

Erweiterung in java.lang.String: repeat() Corner Cases

```
"ERROR".repeat(Integer.MAX_VALUE);
```

```
Exception in thread "main"
java.lang.OutOfMemoryError: Repeating 5 bytes String 2147483647 times will
produce a String exceeding maximum size.
at java.base/java.lang.String.repeat(String.java:3164)
at Java11Examples/snippet.Snippet.main(Snippet.java:14)
```

```
if (Integer.MAX_VALUE / count < len)
{
    throw new OutOfMemoryError("Repeating " + len + " bytes String " + count +
                    " times will produce a String exceeding maximum size.");
}
```

Erweiterung in java.lang.String: strip()/-Leading()/-Trailing()

- **Die Methoden `strip()`, `stripLeading()` und `stripTrailing()` dienen dazu, führende und nachfolgende Leerzeichen (Whitespaces) aus einem String zu entfernen:**

```
private static void stripExample()
{
        final String exampleText1 = "  abc ";
        final String exampleText2 = " \t XYZ \t ";

        System.out.println("'" + exampleText1.strip() + "'");
        System.out.println("'" + exampleText2.strip() + "'");
        System.out.println("'" + exampleText2.stripLeading() + "'");
        System.out.println("'" + exampleText2.stripTrailing() + "'");
}

=>

'abc'
'XYZ'
'XYZ    '
'       XYZ'
```

Text Blocks (> Java 14)

Text Blocks

- **langersehnte Erweiterung, nämlich mehrzeilige Strings ohne mühselige Verknüpfungen definieren zu können und auf fehlerträchtiges Escaping zu verzichten.**

- **Erleichtert unter anderem den Umgang mit SQL-Befehlen, regulären Ausdrücken oder der Definition von JavaScript in Java-Sourcecode.**

- **ALT**

```
String javaScriptCodeOld = "function hello() {\n" +
        "   print(\"Hello World\");\n" +
        "}\n" +
        "\n" +
        "hello();\n";
```

Text Blocks

- **NEU**

```
String javascriptCode = """
                        void print(Object o)
                        {
                            System.out.println(Objects.toString(o));
                        }
                        """;

String multiLineString = """
                         THIS IS
                         A MULTI
                         LINE STRING
                         WITH A BACKSLASH \\
                         """;
```

Text Blocks

- https://openjdk.java.net/jeps/326

Traditional String Literals

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

```
String multiLineHtml = """
                       <html>
                           <body>
                               <p>Hello, world</p>
                           </body>
                       </html>
                       """;
```

Text Blocks

- **NEU**

```
String multiLineSQL = """
                      SELECT `ID`, `LAST_NAME` FROM `CUSTOMER`
                      WHERE `CITY` = 'ZÜRICH'
                      ORDER BY `LAST_NAME;
                      """;


String multiLineStringWithPlaceHolders = """
                        SELECT %s
                          FROM %s
                          WHERE %s
                       """.formatted(new Object[]{"A", "B", "C"});
```

Text Blocks

- **NEU**

```
String jsonObj = """
                        {
                            name: "Mike",
                            birthday: "1971-02-07",
                            comment: "Text blocks are nice!"
                        }
                        """;
```

Besonderheit bei `Text Blocks`

```
String text = """
          This is a string splitted \
          in several smaller \
          strings.\
          """;

System.out.println(text);
```

```
This is a string splitted in several smaller strings.
```

Modifikationen ermöglichen

Strings – Modifikationen ermöglichen

```
// Umwandlung in char[]
jshell> char[] letters = "This is Mava".toCharArray()
letters ==> char[12] { 'T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'M', 'a', 'v', 'a' }

// einzelne Zeichen modifizieren
jshell> letters[8] = 'J'
$204 ==> 'J'

// Rückwandlung aus char[] in String
jshell> new String(letters)
$205 ==> "This is Java"
```

Beispiel toTitleCase()

- Dies ist ein Titel für diese Übung => Dies Ist Ein Titel Für Diese Übung

```
static String toTitleCase(String input)
{
    char[] inputChars = input.toCharArray();
    boolean capitalizeNextChar = true;
    for (int i = 0; i < inputChars.length; i++)
    {
        char currentChar = inputChars[i];
        if (capitalizeNextChar)
        {
            inputChars[i] = Character.toUpperCase(currentChar);
            capitalizeNextChar = false;
        }
        if (Character.isWhitespace(currentChar) || currentChar == '-')
        {
            capitalizeNextChar = true;
        }
    }
    return new String(inputChars);
}
```

Strings – Modifikationen ermöglichen mit StringBuilder

```
// Umwandlung in StringBuilder
StringBuilder sb = new StringBuilder("This is Mava");

// einzelne Zeichen modifizieren
sb.setCharAt(8, 'J');

// Rückwandlung aus char[] in String
System.out.println(sb.toString());
```

DEMO

`StringBuilderExample.java`

Exercises Part 2

https://github.com/Michaeli71/JAVA_INTRO

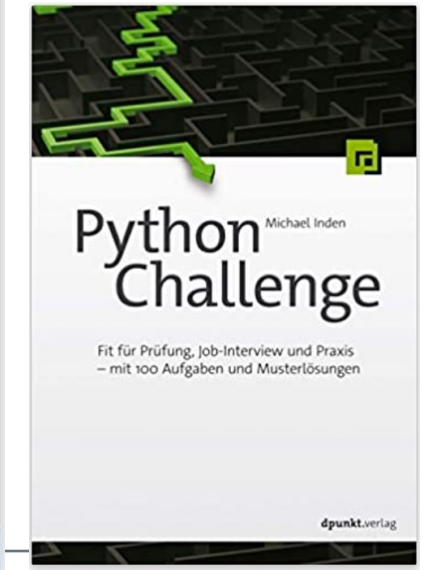
Questions?


HELP
IS ON THE
WAY


Michael Inden
Einfach Java
Gleich richtig programmieren lernen
dpunkt.verlag


5. Auflage
Michael Inden
Der Weg zum Java-Profi
Konzepte und Techniken für die professionelle Java-Entwicklung
dpunkt.verlag


Michael Inden
Java Challenge
Fit für das Job-Interview und die Praxis – mit mehr als 100 Aufgaben und Musterlösungen
dpunkt.verlag


Michael Inden
Python Challenge
Fit für Prüfung, Job-Interview und Praxis – mit 100 Aufgaben und Musterlösungen
dpunkt.verlag

Thank You