Java

Einführung

Marcus Köhler

26. Oktober 2018

Java-Kurs

Überblick

1. Grundlegende Infos

2. Java-Überblick
    - Ein erstes Programm
    - Hello World!
    - Ein neues IntelliJ-Projekt erstellen

3. Java-Basics
    - Definitionen
    - Berechnungen
    - Text mit Strings

Grundlegende Infos

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit
- Du hast das JDK und eventuell die IDE deiner Wahl bereits eingerichtet

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit
- Du hast das JDK und eventuell die IDE deiner Wahl bereits eingerichtet
- Du hast eventuell schon Erfahrung mit anderen Programmiersprachen(nicht notwendig)

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit
- Du hast das JDK und eventuell die IDE deiner Wahl bereits eingerichtet
- Du hast eventuell schon Erfahrung mit anderen Programmiersprachen(nicht notwendig)

Weiteres

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit
- Du hast das JDK und eventuell die IDE deiner Wahl bereits eingerichtet
- Du hast eventuell schon Erfahrung mit anderen Programmiersprachen(nicht notwendig)

Weiteres

- Es wird ~14 Termine geben

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit
- Du hast das JDK und eventuell die IDE deiner Wahl bereits eingerichtet
- Du hast eventuell schon Erfahrung mit anderen Programmiersprachen(nicht notwendig)

Weiteres

- Es wird ~14 Termine geben
- Jeder Termin deckt ~einen Themenbereich ab

Kursinformationen

Voraussetzungen

- Du weißt, wie du deinen Computer bedienst
- Du bringst deinen eigenen Laptop mit
- Du hast das JDK und eventuell die IDE deiner Wahl bereits eingerichtet
- Du hast eventuell schon Erfahrung mit anderen Programmiersprachen(nicht notwendig)

Weiteres

- Es wird ~14 Termine geben
- Jeder Termin deckt ~einen Themenbereich ab
- Zu den meisten Themen gibt es Übungsaufgaben die Du zuhause bearbeiten kannst

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`
- Das Auditorium
  `http://auditorium.inf.tu-dresden.de`

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`
- Das Auditorium
  `http://auditorium.inf.tu-dresden.de`
- StackOverflow, FAQs, Online-tutorials, ...

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`
- Das Auditorium
  `http://auditorium.inf.tu-dresden.de`
- StackOverflow, FAQs, Online-tutorials, ...
- Offizielle Java-Dokumentation
  `https://docs.oracle.com/javase/10/`

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`
- Das Auditorium
  `http://auditorium.inf.tu-dresden.de`
- StackOverflow, FAQs, Online-tutorials, ...
- Offizielle Java-Dokumentation
  `https://docs.oracle.com/javase/10/`
- Programmierkurs-Mailinglist `programmierung@ifsr.de`

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`
- Das Auditorium
  `http://auditorium.inf.tu-dresden.de`
- StackOverflow, FAQs, Online-tutorials, ...
- Offizielle Java-Dokumentation
  `https://docs.oracle.com/javase/10/`
- Programmierkurs-Mailinglist `programmierung@ifsr.de`
- Kurs-Repository
  `https://github.com/B4ckslash/java-lessons`

Ein paar nützliche Ressourcen

- Bei Problemen den Tutor fragen
  `Marcus.Koehler4@tu-dresden.de`
- Das Auditorium
  `http://auditorium.inf.tu-dresden.de`
- StackOverflow, FAQs, Online-tutorials, ...
- Offizielle Java-Dokumentation
  `https://docs.oracle.com/javase/10/`
- Programmierkurs-Mailinglist `programmierung@ifsr.de`
- Kurs-Repository
  `https://github.com/B4ckslash/java-lessons`
- Und natürlich, Google

Java-Überblick

Über Java

Pro:

- Syntax in der Tradition von ALGOL/C/C++
- Entworfen für objektorientiertes Programmieren(OOP)
- Plattformunabhängig (JVM)
- Automatisches Speichermanagement
- Keine zwingend nötigen externen Bibliotheken

  $->$ Einfache und ~~problemlose~~ problemarme Verwendung[1]

[1] Größtenteils

Über Java

Kontra:

- Viele features der JVM/des JDK werden nur selten benötigt (teilweise mit Java 9 behoben[1])
- Langsamer als Maschinensprache
- Keine echte Mehrfachvererbung
- Schwache Generics
- Eher schlechte Unterstützung für andere Programmierparadigmen

  $->$ Weder besonders schnell noch speicherfreundlich

[1] Zumindest kann man die nicht benötigten Features aus einem custom JDK entfernen

Hello World

DEMO

Eine einfache Arbeitsumgebung(Linux/OS X)

In einem neuen Terminal:

```
1       mkdir myProgram
2       cd myProgram
3       # touch Hello.java -- eigentlich nicht nötig
4       vim Hello.java //spezifischer Editor ist egal
```

Eine einfache Arbeitsumgebung(Windows)

- Methode 1: In einem neuen cmd-Fenster:

```
1  mkdir myProgram
2  cd myProgram
3  notepad Hello.java
```

Eine einfache Arbeitsumgebung(Windows)

- Methode 1: In einem neuen cmd-Fenster:

```
mkdir myProgram
cd myProgram
notepad Hello.java
```

- Methode 2: Im Explorer einen neuen Ordner + eine neue Datei anlegen und mit Notepad öffnen

Hello World!

Eine leere Java-Klasse.

Java-Klassen müssen immer mit einem Großbuchstaben beginnen.

```
1 public class Hello {
2
3 }
```

Hello World!

Dieses kurze Programm schreibt `Hello World!` auf die Konsole:

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

Das Programm starten

Die Datei als Hello.java speichern
und dann folgende Befehle ausführen:

```
javac Hello.java
java Hello
```

Output:

```
Hello World!
```

Hello World in einer IDE

DEMO

Java-Basics

Kommentare

```java
public class Hello {
    // Outputs "Hello World!" to the console
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Idealerweise sollte man seinen Code immer kommentieren.
Code wird oft häufiger gelesen als er geschrieben wird.

- `// Einzeiliger Kommentar`
- `/* Mehrzeiliger Kommentar*/`

Kommentare

Ein paar Guidelines für Kommentare:

- „Guter Code dokumentiert sich selbst“

Kommentare

Ein paar Guidelines für Kommentare:

- „Guter Code dokumentiert sich selbst“
  - Aussagekräftige Namen wählen

Kommentare

Ein paar Guidelines für Kommentare:

- „Guter Code dokumentiert sich selbst“
  - Aussagekräftige Namen wählen
  - Komplexe Codeblöcke in kleine, einfache Teile aufbrechen (-> Refactoring)

Kommentare

Ein paar Guidelines für Kommentare:

- „Guter Code dokumentiert sich selbst“
  - Aussagekräftige Namen wählen
  - Komplexe Codeblöcke in kleine, einfache Teile aufbrechen (-> Refactoring)
- Keine offensichtlichen Infos wiederholen:

```
1 // multipliziert y mit x
2 int z = y * x;
```

Kommentare

- Notwendige, aber nicht intuitive Workarounds sollten erklärt werden:[1]

```
function addSetEntry(set, value) {
  /*
     Don't return `set.add` because it's not chainable in
    IE 11.
       */
         set.add(value);
           return set;
}
```

[1] Code snippet: `https://lodash.com/docs`

Code-Grundbausteine

```java
public class Hello {
    // Calculates some stuff and outputs everything on the
    console
    public static void main(String[] args) {
        int x;
        x = 9;
        int y = 23;
        int z;
        z = x * y;

        System.out.println(z);
    }
}
```

Jede Deklaration, Zuweisung & jeder Methodenaufruf ist ein *Statement*

Code-Grundbausteine

Den vorherigen Code kann man leicht auf das folgende reduzieren:

```
1 public class Hello {
2     // Calculates some stuff and outputs everything on the
      console
3     public static void main(String[] args) {
4         System.out.println(9 * 23);
5     }
6 }
```

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)
- `long`, einen 64-bit Integer

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)
- `long`, einen 64-bit Integer
- `float`, eine 32-bit Gleitkommazahl
  (engl. floating point number)

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)
- `long`, einen 64-bit Integer
- `float`, eine 32-bit Gleitkommazahl
  (engl. floating point number)
- `double`, eine 64-bit Gleitkommazahl

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)
- `long`, einen 64-bit Integer
- `float`, eine 32-bit Gleitkommazahl
  (engl. floating point number)
- `double`, eine 64-bit Gleitkommazahl
- `char`, ein 16-bit Unicode Zeichen
  (engl. character)

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)
- `long`, einen 64-bit Integer
- `float`, eine 32-bit Gleitkommazahl
  (engl. floating point number)
- `double`, eine 64-bit Gleitkommazahl
- `char`, ein 16-bit Unicode Zeichen
  (engl. character)
- `void`, den leeren Typ (wird später gebraucht)

Primitive Datentypen

Java unterstützt folgende primitive Datentypen:

- `boolean`, einen Wahrheitswert (entweder `true` oder `false`)
- `int`, eine 32-bit Ganzzahl(a.k.a. Integer)
- `long`, einen 64-bit Integer
- `float`, eine 32-bit Gleitkommazahl
  (engl. floating point number)
- `double`, eine 64-bit Gleitkommazahl
- `char`, ein 16-bit Unicode Zeichen
  (engl. character)
- `void`, den leeren Typ (wird später gebraucht)

Es gibt außerdem noch zwei eher selten verwendete Datentypen:

- `short`, einen 16-bit Integer
- `byte`, einen 8-bit Integer

Blöcke

```
1 public class Hello {
2     public static void main(String[] args) {
3     // prints a "Hello World!" on your console
4         System.out.println("Hello World!");
5     }
6 }
```

Alles zwischen { und } ist ein *Block*.

Blöcke können geschachtelt werden.

Das Semikolon

```java
public class Hello {
    // prints a "Hello World!" on your console
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Jedes *Statement* wird von einem Semikolon abgeschlossen.

Blöcke müssen nicht mit einem Semikolon beendet werden.

Variablennamen

- Variablennamen können mit einem beliebigen Buchstaben oder Unterstrich beginnen.
  Für gewöhnlich beginnen sie mit einem kleinen Buchstaben(abhängig von lokalen Konventionen und coding styles).

Variablennamen

- Variablennamen können mit einem beliebigen Buchstaben oder Unterstrich beginnen.
  Für gewöhnlich beginnen sie mit einem kleinen Buchstaben(abhängig von lokalen Konventionen und coding styles).
- Zusammengesetzte Namen sollten camelCase verwenden.

Variablennamen

- Variablennamen können mit einem beliebigen Buchstaben oder Unterstrich beginnen.
  Für gewöhnlich beginnen sie mit einem kleinen Buchstaben(abhängig von lokalen Konventionen und coding styles).
- Zusammengesetzte Namen sollten camelCase verwenden.
- Variablennamen sollten aussagekräftig sein.

Variablennamen

- Variablennamen können mit einem beliebigen Buchstaben oder Unterstrich beginnen.
  Für gewöhnlich beginnen sie mit einem kleinen Buchstaben(abhängig von lokalen Konventionen und coding styles).
- Zusammengesetzte Namen sollten camelCase verwenden.
- Variablennamen sollten aussagekräftig sein.

```java
public class Calc {
    public static void main(String[] args) {
        int a = 0; // not very meaningful
        float myFloat = 5.3f; // also not meaningful
        int count = 7; // quite a good name

        int rotationCount = 7; // there you go
    }
}
```

Berechnungen mit `int` i

Java unterstützt folgende Operationen nativ:

Addition	`a + b;`
Subtraktion	`a - b;`
Multiplikation	`a * b;`
Division	`a / b;`
Modulo(Restdivision)	`a % b;`
Inkrementierung	`a++;`
Dekrementierung	`a--;`

Weitere arithmetische Operationen(Wurzel etc.) werden von Bibliotheken übernommen.

Berechnungen mit `int` ii

```java
public class Calc {
    public static void main(String[] args) {
        int a; // declare variable a
        a = 7; // assign 7 to variable a
        System.out.println(a); // prints: 7
        a = 8;
        System.out.println(a); // prints: 8
        a = a + 2;
        System.out.println(a); // prints: 10
    }
}
```

Variablen mit primitiven Typen sind nach der ersten Zuweisung initialisiert.

Berechnungen mit **int** iii

```java
public class Calc {
    public static void main(String[] args) {
        int a = -9; // declaration and assignment of a
        int b; // declaration of b
        b = a; // assignment of b
        System.out.println(a); // prints: -9
        System.out.println(b); // prints: -9
        a++; // increments a
        System.out.println(a); // prints: -8
    }
}
```

Berechungen mit **float** i

```java
public class Calc {
    public static void main(String[] args) {
        float a = 9;
        float b = 7.5f;
        System.out.println(a); // prints: 9.0
        System.out.println(b); // prints: 7.5
        System.out.println(a + b); // prints: 16.5
    }
}
```

Berechungen mit **`float`** ii

```java
public class Calc {
    public static void main(String[] args) {
        float a =       8.9f;
        float b = 3054062.5f;
        System.out.println(a); // prints: 8.9
        System.out.println(b); // prints: 3054062.5
        System.out.println(a + b); // prints: 3054071.5
    }
}
```

Gleitkommazahlen sind nicht beliebig genau.
Dies kann manchmal zu unerwarteten und/oder falschen Ergebnissen führen!

Arithmetik mit **int** und **float**

```
1  public class Calc {
2      public static void main(String[] args) {
3          float a = 9.3f;
4          int b = 3;
5          System.out.println(a + b); // prints: 12.3
6          float c = a + b;
7          System.out.println(c); // prints: 12.3
8          int d = a/c; // this shouldn't/doesn't work and should/
       will throw an Exception
9      }
10 }
```

Java wandelt `int` automatisch in `float` um, falls nötig(„casting“).
Umgekehrt aber nicht.

Strings

Ein String ist kein primitiver Datentyp, sondern ein Objekt[1].
Objekte werden in der nächsten Woche behandelt.

```
1 public class Calc {
2     public static void main(String[] args) {
3         String hello = "Hello World!"; // Strings sind immer von
       doppelten Anführungszeichen umgeben
4         System.out.println(hello); // print: Hello World!
5     }
6 }
```

[1] Deshalb beginnt der Datentyp auch mit einem Großbuchstaben

Konkatenation

```java
public class Calc {
    public static void main(String[] args) {
        String hello = "Hello";
        String world = " World!";
        String sentence = hello + world;
        System.out.println(sentence);
        System.out.println(hello + " World!");
    }
}
```

Strings können mit „+“ konkateniert(d.h. aneinandergehängt) werden.
Die ausgegebenen Zeilen sind gleich.

Strings und Zahlen

```java
public class Calc {
    public static void main(String[] args) {
        int factorA = 3;
        int factorB = 7;
        int product = factorA * factorB;
        String answer =
            factorA + " * " + factorB + " = " + product;
        System.out.println(answer); // prints: 3 * 7 = 21
    }
}
```

Primitive Datentypen werden bei Konkatenation in ihren momentanen Wert als `String` umgewandelt[1].

[1] Genau genommen ruft es die toString() Methode des zugehörigen Objekts auf, dazu nächste Woche mehr

Finde den Fehler

```
1  public class calc {
2      public static void main(string[] args) {
3          int myInt = 5
4          int mySecondInt;
5          float myFloat = 7.4f;
6          float result = myInt + myFloat
7          result = result + mySecondInt;
8          String out = Result = + result;
9          System.out.println(out);
10     };
11 }
```

Finde den Fehler

```
1  public class calc {
2      public static void main(string[] args) {
3          int myInt = 5;
4          int mySecondInt;
5          float myFloat = 7.4f;
6          float result = myInt + myFloat;
7          result = result + mySecondInt; //mySecondInt ist nicht
       initialisiert
8          String out = "Result = " + result;
9          System.out.println(out);
10     };
11 }
```