Android Programmierung mit Java Studiengang MI

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- http://mwilhelm.hs-harz.de
- Raum 2.202
- Tel. 03943 / 659 338

Gliederung

Überblick:

- **Einleitung, Installation**
- **Sprache**
  - elementare Datentypen
  - Variablen und Kontrollstrukturen
  - Arrays, Funktionen, Rekursionen
  - Objekte, etc.
- **Android Studio**
- **Grafische Oberfläche**
  - Einfache UI-Elemente
  - Layouts
  - Core-Data
  - Sensoren
- Viele Beispiele

Links

- https://developer.android.com/studio/intro/index.html
- https://developer.android.com/studio/index.html

Java: Kommentare:

- **Allgemeine Kommentare**
  - // nur eine Zeile

  - /* Bereich Anfang
  - */ Bereich Ende

Java: Variablen

Deklaration immer explizit	**Zuweisung**
int implicitInteger;	implicitInteger = 2.0;
double implicitDouble = 70.0;	implicitDouble = 70.1;
final double constantDouble = 42;	~~constantDouble = 43;~~
MyClass mc = new MyClass(); mc.a = 22;	(new MyClass()).a = 22;

Java: Datentypen

- int Integer
- float Float
- double Double,
- booelan
- String
- Object

Java: Datentypen

- Int
  - byte 8-Bit, unsigned-int -128 → +127
  - short 16-Bit, unsigned-int -32768 → 32767
  - int 32-Bit, signed-int -2.147.483.648 → +2.147.483.647
  - long 64-Bit, signed-int -9223372036854775808 ... 9223372036854775807
- Floating-Point
  - float, Float Single-Format (1,8,23-Bit, 7 Stellen)
  - double, Double Double-Format (1,11,52-Bit, 15-Stellen)
- boolean, Boolean
- BigDecimal Numerik-Datentyp mit einfachen Operatoren
- String
- class
- **KEINE** **struct**
- **KEINE** **tupel**

Java: String-Datentyp

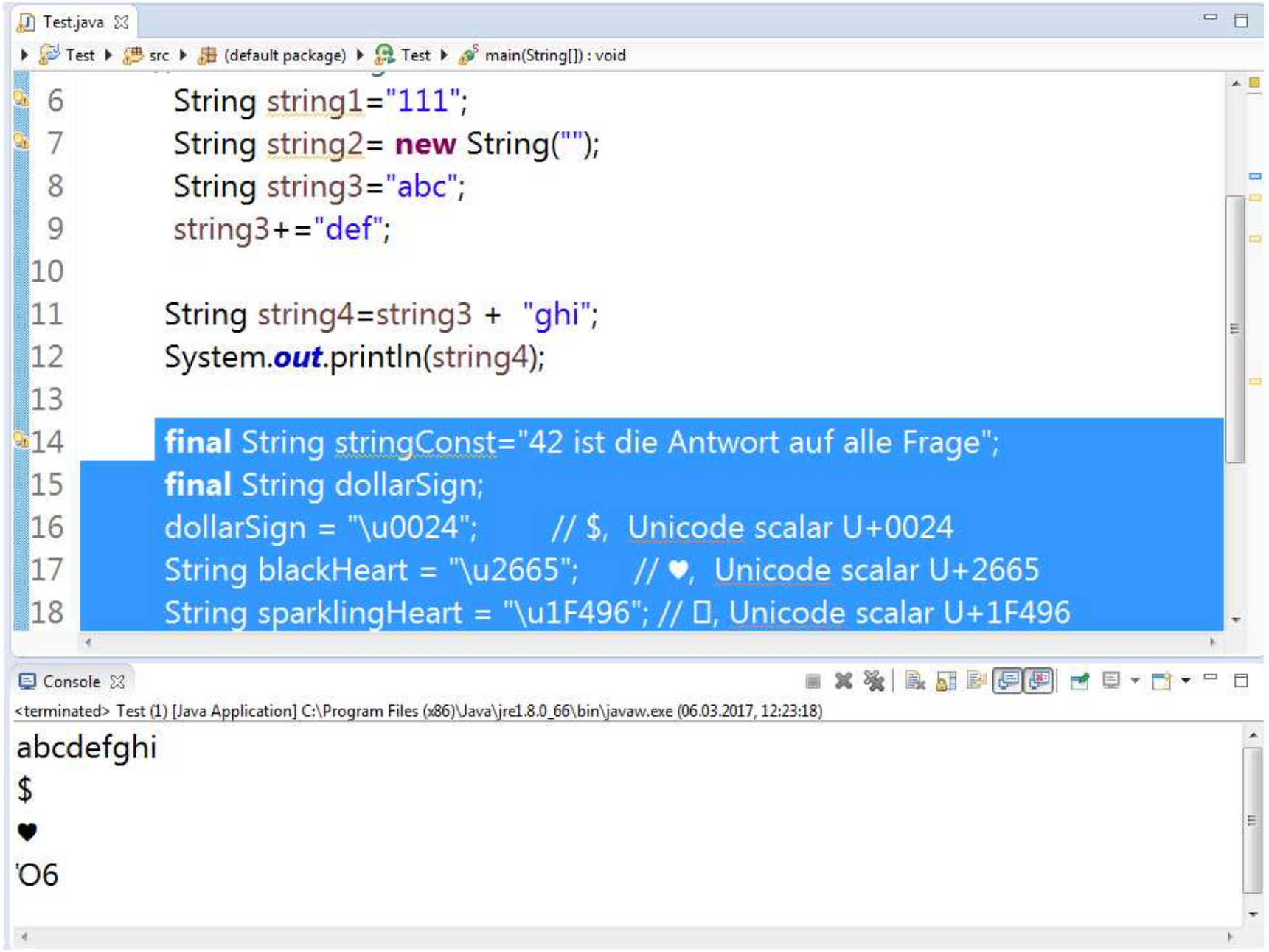
- **Zuweisung**
  - String string1="";
  - String string2= new String("") ;
  - String string3="abc";
  - string3+="def"; // concat
  - string3.append("def") ; // besser StringBuilder
  - String string4=string3 + "ghi";

Java: String-Datentyp

Konstanten

- final String stringConst="42 ist die Antwort auf alle Frage";
- final String dollarSign;
- dollarSign = "\u0024"; // $, Unicode scalar U+0024

- **Unicode**
- String blackHeart = "\u2665"; // ♥, Unicode scalar U+2665
- String sparklingHeart = "\u1F496"; // 💖, Unicode scalar U+1F496
- System.*out.println(dollarSign);*
- System.*out.println(blackHeart);*
- System.*out.println(sparklingHeart);*
- **double \u03C0 = 3.141592653589793;**
- **\u0063\u006C\u0061\u0073\u0073\u0020\u0041\u0020\u007B\u007D**


Test.java
Test ▸ src ▸ (default package) ▸ Test ▸ main(String[]) : void
6 String string1="111";
7 String string2= new String("");
8 String string3="abc";
9 string3+="def";
10
11 String string4=string3 + "ghi";
12 System.out.println(string4);
13
14 final String stringConst="42 ist die Antwort auf alle Frage";
15 final String dollarSign;
16 dollarSign = "\u0024"; // $, Unicode scalar U+0024
17 String blackHeart = "\u2665"; // ♥, Unicode scalar U+2665
18 String sparklingHeart = "\u1F496"; // □, Unicode scalar U+1F496
Console
<terminated> Test (1) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_66\bin\javaw.exe (06.03.2017, 12:23:18)
abcdefghi
$
♥
'O6

Java: String-Datentyp

- **Methoden**
  - equals
  - length
  - charAt
  - indexOf
  - compareTo
  - compareToIgnoreCase
  - concat
  - contains
  - isEmpty
  - lastIndexOf
  - matches

- **Methoden**
  - replace
  - split
  - substring
  - toLowerCase
  - toUpperCase
  - toCharArray()
  - trim
  - format printf
  - valueOf Exception

Java: String-Datentyp

- **Abfragen**
  - String sind Referenzen
  - if (string1.isEmpty()) {
  - }
  - for (char ch : "Dog!🐶".toCharArray() {
    System.out.print(ch);
    }

Java: Konvertierung

String s="1.23";

int i = Integer.*valueOf(s1)*; // Rückmeldung mit einer Exeption

- Exception in thread "main" java.lang.NumberFormatException: For input string: "AA"
- at java.lang.NumberFormatException.forInputString(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at Test.main(Test.java:27)

```
String s1="AA";
    try {
       int i1 = Integer.valueOf(s1);
    }
    catch (NumberFormatException e) {
       System.out.println("Fehler: "+e);
    }
```

Java: Konvertierung

String s="123";

int i = Integer.*valueOf(s1)*; // Rückmeldung mit einer Exeption

- Exception in thread "main" java.lang.NumberFormatException: For input string: "AA"
- at java.lang.NumberFormatException.forInputString(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at Test.main(Test.java:27)

```
String s1="AA";
    try {
       int i1 = Integer.valueOf(s1);
    }
    catch (NumberFormatException e) {
       System.out.println("Fehler: "+e);
    }
```

Java: Konvertierung

String s="a123";

double d = Double.parseDouble(s); // Rückmeldung mit einer Exeption

- Exception in thread "main" java.lang.NumberFormatException: For input string: "AA"
- at java.lang.NumberFormatException.forInputString(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at Test.main(Test.java:27)

- String s1="AA";
- **try {**
- **double d = Double.parseDouble(s);**
- }
- **catch (NumberFormatException e) {**
- System.***out.println("Fehler: "+e);***
- }

Java: Konvertierung

String s="a123";

double d = Double.valueOf(s); // Rückmeldung mit einer Exeption

- Exception in thread "main" java.lang.NumberFormatException: For input string: "AA"
- at java.lang.NumberFormatException.forInputString(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at java.lang.Integer.parseInt(Unknown Source)
- at Test.main(Test.java:27)

- String s1="AA";
- **try {**
- **double d = Double.valueOf(s);**
- }
- **catch (NumberFormatException e) {**
- System.***out.println("Fehler: "+e);***
- }

Java: Operatoren:

- !=
- %
- %=
- &
- &&
- &*
- &+
- &-
- &=
- *
- *=
- +
- +=
- -
- -=
- /
- /=
- <
- <<
- <<=
- <=
- = =
- >
- >=
- >>
- >>=
- ^ xor
- ^=
- |
- |=
- ||
- ~ bit Negation
- ~=
- ~>

- Coalescing Op.
- a ?? b
- wenn a<>nil a!
- sonst b

Java: is-Operator:

- Der Operator **instanceof** testet, ob eine Instanz einem Typ entspricht:

```
int i1 = Integer.valueOf(s1);
if (i1 instanceof Integer) {
    System.out.println("int ja");
}
// Fehler
```

```
Integer i1 = Integer.valueOf(s1);
if (i1 instanceof Integer) {
    System.out.println("int ja");
}
// korrekt
```

Java: as-Operator:

Der Operator **(Typ)** wandelt eine Variable in einen anderen Typ um.

- Hier wird keine Überprüfung durchgeführt. Es wird nur durchgeführt, wenn der Compiler vorher erkennt, dass das Casting gefahrlos möglich ist. Es ist nur sinnvoll beim UpCasting:

  float nach double

  byte nach int
- Hier wird keine Überprüfung durchgeführt. Bei einem Fehlerfall wird eine Exception ausgelöst:
  - CastException

Java: if-Anweisungen: Immer mit den Klammern (), kein elif

```
int  a = 5
int  b = 7
if (a > b) {
        print("a ist größer als b");
}
else {
        print("a ist kleiner als b");
}

if (a > b)
        print("a ist größer als b");
else
        print("b ist kleiner als b");
```

Java: case-Anweisungen

```
int a = 4;

switch (a) {
    case 1:
    case 2:
            System.out.println("1  oder 2");
            break;
    case 4:
            System.out.println("1  oder 2");
            break;
    default:          // muss nicht immer eintragen werden
             System.out.println("else");
}
```

Java: case-Anweisungen

```
String  a = "a";

switch (a) {
    case "a":
            System.out.println("a");
    case "b":
            System.out.println("b");
    default:
            System.out.println("else");
}
```

Ausgabe?

While-Schleife

Aufbau:

1) Initialisierung

2) while (Bedingung/en) {

3) Schleifenrumpf / Anweisungen

4) }

Bedingung zeigt an, wie oft die Schleife durchlaufen wird

Do-While-Schleife

Aufbau:

1) Initialisierung

2) do {

3) Schleifenrumpf / Anweisungen

4) }

5) while (Bedingung/en);

Bedingung zeigt an, wie oft die Schleife durchlaufen wird

For-Schleife

```
for ( Init;  Bedingung; Testvariablen anpassen ) {
  // Fuehre Arbeiten durch
  // Berechne Bedingung neu
}
```

```
for ( int iy=0; iy<10;  iy=iy+1 ) {
  line( 10, iy,  300, iy);
}
```

2. Variante ??

```
for ( int iy=0; iy<10;  iy=iy-1 ) {
  line( 10, iy,  300, iy);
}
```

For-each-Schleife

```
for ( Datentyp  :  Array/Liste) {
  // Fuehre Arbeiten durch
  // Berechne Bedingung neu
}
```

```
int[] feld = {2,3,5,7,11,13,17};
for ( int i  : feld) {
  syso( i);
}
```

Java: Arrays

- List — Oberklasse
  - Array — fest
  - ArrayList — flexibel
  - Vector — flexibel
  - LinkedList — flexibel

- Hashtable

Deklaration einer 10x10 Matrix

```
int n=10;
int m=10;
int[][] a = new int[n][m];
int[][] b = new int[n][m];

for(int i=0; i<n; i++) {
  for(int j=0; j<m; j++) {
    a[i][j] = i+j;
    b[i][j] = i+i  +  j<<1;
  }
}
// <<  bedeutet Multiplikation mit zwei
// >>  bedeutet Multiplikation mit 0,5
```

Sichere Initialisierung von Arrays

Anstatt

```
float[] werte = new float[6];
for(int i = 0; i < 6; i++) {
werte[i] = 0;
}
```

lässt sich sicherer schreiben:

```
int[]
float[] werte = new float[6];
for(int i = 0; i < werte.length; i++) {
  werte[i] = 0;
}
```

Java: Arrays

```
var array: [String] = []
var array: [String]()
Array.append("Michel")
Array.append("Andrea")
for personname in array {
  print("Person: \(personname)")
}
for i in 0…len(array) -1 {
  print("Person: \(array[i])")
}
for i in 0..<len(array) {
  print("Person: \(array[i])")
}
```

ev. FEHLER

Java: mehrdimensionale Arrays

```
var array = [  [1,2,3] [4,5,6,7,8] , [9,10]  ]
array.count      liefert 3
for (row in 0..<array.count) {
    for (col in 0..<array[row].count) {
      print("row: \(row)   col \(col)  : \(array[row][col])   ")
    }
}

var array2D = [ [Int]  ]()
var array3D = [ [  [Int]  ] ]()
```

Java: Enumeration:

- Enumeration sind besser als mehrere Konstante:
  - final int fb_ai =1;
  - final int fb_vw =2;
  - final int fb_w =3;
  - printFB(int fn) { …. }

- Enumeration

```
enum FB {AI, VW, W};

static void test5() {
  FB fb = FB.AI;
}
```

- Enumeration

```
enum FB {AI=1, VW=2, W=3};

static void test5() {
  FB fb = FB.AI;
}
```

- Enum sinnvoll auch in einer switch-Anweisung

Java: void Funktionen: call by value, call by reference

```
void print1() {
    print("hallo");
}
print1();
```

```
void printInt() {
  int a=3;
  print("a hat den Wert:"+a);
}
printInt();
```

```
void print2(a:int) {
    print("a", a);
}
print2(a:33);
```

```
void print3(a:int, b:int) {
    print("a", a, "b", b);
}
print3(a:33, b:11);
```

Java: void Funktionen

```
double Inch2Cm(inch:double)  {
    return inch*2.54;
}
double cm=1.23;
double inch=Inch2Cm(cm);
```

Java: try catch, gibt es nur mit Funktionen

- Fehlerabfangen

```
try{

        …
}  catch (ExceptionTyp1 e1) {
    …
}
}  catch (ExceptionTyp2 e2) {
    …
}
```

Java: Klassen:

```
class MyClass: OptSuperclass, OptInterface, OptInterface2 {
    String myProperty;

    public MyClass() {
        myProperty = "abc„;
    }
    int doIt()  {
      return 0;
    }
    int doIt(int a)  {
      return a+1;
    }
    public int getMyProperty() {
        return myProperty;
    }
}
```

```
public void setMyProperty(int value) {
        myProperty = value;
}
```

Java: Klassen

- class
- Instanz: **mit new**
- MyClass Konstruktor, immer der Name der Klasse
- Properties setter / getter, extra Methoden
- Type Methoden statische und dynamische Methoden
- **interface** **Schnittstelle, WICHTIG**
- this eigene Objekt, Instanz
- null keinen Wert
- Vererbung ja, aber nur einfache Vererbung