Einführung in die Programmierung

by André Karge

Übung - Static, Arrays, Klassenstruktur, Programmstruktur

Heute

- Vorstellung Übungsblatt 02
- Floating Point Checks
- Static
- Arrays
- Klassenstruktur
- Programmstruktur

Floating Point Checks

Floating Point Checks

```
float a = 1.0f/3.0f;
float b = 0.33;
// a == b?
```

Wie vergleicht man zwei Floating Point Nummern?

Floating Point Check

Lösung

- wir können float Werte nicht direkt vergleichen
- dazu brauchen wir ein kleinen Floatwert epsilon ϵ
- zum Vergleich berechnen wir die Differenz und vergleichen mit epsilon
- zum Beispiel: $\epsilon = 10^{-14} = \frac{1}{10^{14}}$

Floating Point Check

Lösung

- wir können float Werte nicht direkt vergleichen
- dazu brauchen wir ein kleinen Floatwert epsilon ϵ
- zum Vergleich berechnen wir die Differenz und vergleichen mit epsilon
- zum Beispiel: $\epsilon = 10^{-14} = \frac{1}{10^{14}}$

Übungsaufgabe - 5min

1. Schreiben sie eine Methode, die zwei float-werte übergeben bekommt und true oder false zurück gibt, jenachdem, ob die Zahlen gleich oder ungleich sind.

Floating Point Check

Lösung

```
public class Test {
  private static final float epsilon = 10e-14;
  public static void main(String[] args) {
    float a = 1.0f/3.0f;
    float b = 0.33;
    boolean result = floatEqual(a, b);
  }
  public boolean floatEqual(float a, float b) {
    float differenz = Math.abs(a - b);
    if(differenz < Test.epsilon) {
      System.out.println("Werte sind gleich");
      return true;
    }
    else {
      System.out.println("Werte sind nicht gleich");
      return false;
    }
  }
}
```

Static Methoden & Attribute

Static Methoden & Attribute - Insel 5.3

- statische Methoden und Attribute vereinbaren, dass sie den Zustand von Objekten nicht ändern
- Schlüsselwort *static*
- Kapselt feste Klassenzugehörige Eigenschaften und Funktionen innerhalb der Klasse
- Alle Instanzen haben somit Zugriff auf die selbe Version der Methode / Attributes

Beispiel Static Methoden & Attribute

```
class Complex {
  private int real;
  private int imag;

  private static counter = 0;

  // Constructor
  public Complex(int real, int imag) {
    this.real = real;
    this.imag = imag;
    this.counter++; // Zugriff auf static Attribut
  }

  // Es wird kein Zustand von vorhandenen Objekten geändert - static
  public static final Complex addStatic(Complex z1, Complex z2) {
    return new Complex(z1.real + z2.real, z1.imag + z2.imag);
  }
  // ändert den Zustand des Objektes, auf dem .add(...) aufgerufen wird - nicht static
  public void addToInstance(Complex z2) {
    this.real += z2.real;
    this.imag += z2.imag;
  }
```

Beispiel Static Methoden & Attribute

```
class TestComplex {
  public static void main(String[] args) {
    Complex c1 = new Complex(1, 2); // counter = 1
    Complex c2 = new Complex(42, 16); // counter = 2

    //Nutzen der static add Methode => neues Objekt
    Complex c3 = Complex.addStatic(c1, c2); // real = 43, imag = 18 (counter = 3)

    //Nutzen der non-static add Methode => kein neues Objekt
    c2.add(c3); // c2.real = 85, imag = 32 (counter = 3)
  }
}
```

Static Methoden & Attribute - Insel 5.3

statische Attribute

- haben wir schon auf einer vorherigen Folie gesehen:

```
public class Test {
  private static final float epsilon = 10e-14;
  ...
}
```

- der Wert soll für alle Ausprägungen (Instanzen) der Klasse gelten
- wird zusätzlich das Schlüsselwort *final* verwendet, dann darf der Wert nicht geändert werden
- Für Konstanten hilfreich

Static Methoden & Attribute - Insel 5.3

Übungsaufgabe - 10min

1. Schreiben Sie eine Klasse, die Schüler repräsentiert (Schüler haben Namen, Vornamen und Noten) mit entsprechendem Konstruktor (Tipp: nennen Sie ihre Klasse *Schueler*)
2. Erweitern Sie diese Klasse um die Möglichkeit, Überblick über die Anzahl bisher erstellter Schülerobjekte zu behalten
3. Testen Sie die Funktionalität in einer neuen *SchuelerTest*

Arrays

Arrays - Insel 3.7

- *Array* ist ein spezieller Datentyp
- auch *Feld* oder *Reihung* genannt
- fasst mehrere Werte in einer Einheit zusammen
- deklariert mit: Typ[] bezeichner = {element1, element2, ...}
- Index = Addresse von Werten im Array
- Index = Ganzzahl vom Typ integer und **startet in Java bei 0**
- Beispiel:

```
int[] meinArray = {15,12,13,1,14};
// indexing:       0  1  2  3 4
int length = meinArray.length; // = 5
int erstes = meinArray[0]; // = 15
//Ausgabe
System.out.println(meinArray[2]); // 13
```

Arrays - Insel 3.7

Mehrdimensionale Arrays

```
// mit konkreten Werten:
int[][] a = {{1, 2}, {3, 4}, {5, 6}}; // 3 Zeilen und 2 Spalten
// leeres Array mit definierter Größe:
int[][] b = new int[4][8]; // 4 zeilen und 8 Spalten
a[1][0] = 15; // Zuweisung

// Iteration über arrays:
for(int i=0; i<a.length; i++) { // Zeilen | Rows
  for(int j=0; j<a[i].length; j++) { // Spalten | Columns
    a[i][j] = i + j;
  }
}
```

String als Char-Array - Insel 4.2

- String ist eine spezielle Klasse (komplexer Datentyp)
- Enthalten Zeichenketten
- kann man auch als character-Array darstellen

```
public static void main(String[] args) {
  String meinString = "Hello World";

  char[] meinCharArr = meinString.toCharArray(); // cast zu char-array
  System.out.println(meinCharArr.length);

  char[] invers = new char[meinCharArr.length]; // neues array der selben länge
  int counter = 0; // counter für indices
  for(int i=meinCharArr.length-1; i>=0; i--) { // rückwärts iterieren
      invers[counter] = meinCharArr[i];
      counter++;
  }
  // invertierter String
  System.out.println(new String(invers));
}
```

String als Char-Array - Insel 4.2

- Die String Klasse bietet jedoch sehr viele indexing Funktionen wie *charAt* oder *substring*
- Um Strings zu vergleichen nutzt man die Methode *equals*

```
public static void main(String[] args) {
  String a = "Hello World";
  String b = "Hallo Welt";

  boolean vergleich = a.equals(b);

  char c = a.charAt(2); // l
  String sub = a.substring(3, 7); // nehme alle Zeichen mit Start an index 3 und Ende bevor index 7
  // = "lo W"
}
```

Command-Line-Arguments

- Sind auch über ein Array ansprechbar

Application.java

```
class Application {
  public static void main(String[] args) {
    // args sind alle argumente, die beim starten der Applikation angegeben werden
    for(int i=0; i<args.length; i++) {
      System.out.println(args[i]);
    }
  }
}
```

```
javac Application.java
java Application hallo welt 123
> hallo
> welt
> 123
```

Arrays - Insel 3.7

Übungsaufgabe - 5min

1. Schreiben Sie eine Methode, die ein festes Float Array der Größe 42 erstellt
2. Iterieren Sie mit einer for-Schleife über dieses Array und schreiben an jede Stelle die aktuelle Schleifenvariable geteilt durch 7
3. geben Sie die Quersumme der Zahlen im Array auf der Konsole aus
4. lassen Sie die größe des Arrays als Command-Line-Argument angeben

Klassenstruktur

Klassenstruktur

- Klassen bilden immer eine sogenannte Kompilationseinheit (engl.: compilation unit)
- Faustregel: eine *.java*-Datei beinhaltet immer eine Klasse
- Der Scope der Klasse beginnt an der geschweiften Klammer { nach dem Header der Klasse und endet immer nach der dazugehörigen schließenden Klammer }
- **Man schreibt keine neuen Klassen in den Scope einer anderen Klasse!**
- um ein Java Programm ausführen zu können benötigt man eine main-Methode (und zwar genau **eine**)
- Wenn mein Programm also aus 6 Klassen besteht, sollte ich auch 6 *.java*-Dateien haben und in einer davon sollte sich die main-Methode befinden
- Die Klasse in einer Datei muss die exakt selbe Bezeichnung haben wie die *java*-Datei in der sie sich befindet

Klassenstruktur - Beispiel

Schueler.java

```
class Schueler {
  private String name;
  public Schueler(String name) {
    this.name = name;
  }
}
```

Application.java

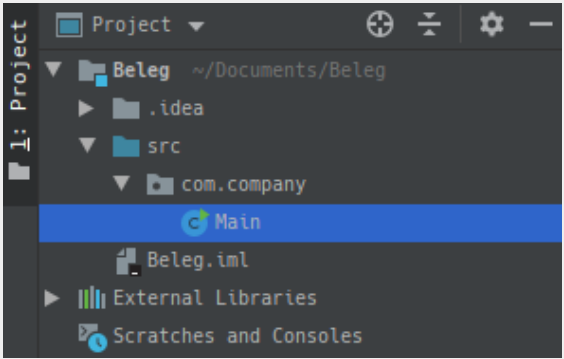
```
class Application {
  public static void main(String[] args) {
    Schueler peter = new Schueler("Peter Zwegat");
    Lehrer hans = new Lehrer("Hans Maulwurf", 16);
  }
}
```

Lehrer.java

```
class Lehrer {
  private String name;
  private double gehalt;
  public Lehrer(String name, double gehalt) {
    this.name = name;
    this.gehalt = gehalt;
  }
}
```

Projektstruktur

- IDEs erstellen bei neuen Projekten immer eine Ordnerstruktur




```
Beleg
|-.idea              // Projekt-Datei
|-src                // Ordner für Source Code
| |-com              // erster Teil des
| | |                   standard-packages
| | |-company        // zweiter Teil des
| | | |                 standard-packages
| | | |-Main.java    // source code
| | | |-AndereKlasse.java
| | | |-NochEineKlasse.java
```

Abgabe von Dateien für Belege

- Ich möchte nur **.java-Dateien** und wenn Theorieaufgaben dabei sind **eine! .pdf-Datei** in Abgaben sehen
- Keine kompletten Projektverzeichnisse!

- Die Abgabe soll sich **immer!** in einem komprimierten Archiv befinden
- Beispiel: *.zip*, *.rar*, *.tar.gz*
- Die Archivdatei soll richtig benannt werden

```
programmierung_uebung03_gruppe03.zip
|-uebung03.pdf  // Alle theoretischen Lösungen
|-aufgabe01     // Aufgabe01 erfordert code
| |-Application.java
| |-Schueler.java
| |-Lehrer.java
|-aufgabe02     // Aufgabe02 erfordert code
| |-Complex.java
|-aufgabe03     // Aufgabe03 erfordert code
| |-Triangle.java
| |-Line.java
| |-Point.java
| |-Triangle2.java
```

Abgaben

- Gebt in **jeder!** Datei eure Gruppennummer an
- in der theorie-PDF immer am Anfang
- in jeder *.java*-Datei immer als Kommentar am Anfang

```java
/**
Gruppe 03
**/

class Application {
  public static void(String[] args) {
    // ...
  }
}
```

Fragen?