Karlsruher Institut für Technologie

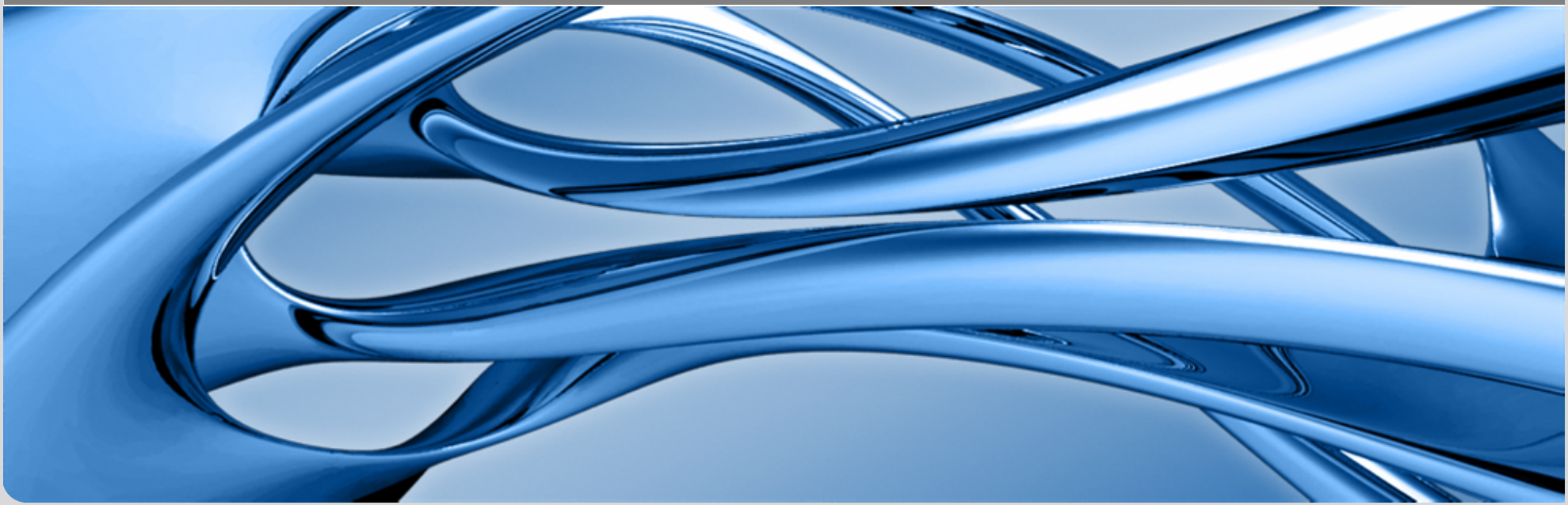
SDQ Software Design and Quality

Tutorium 5

Arrays, Codestyle und Datenkapselung

Christian Zielke | 27. November 2018

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

1. Organisatorisches
   - Übungsblatt 1
2. Codestyle
   - Warum überhaupt Codestyle?
   - Checkliste
3. Arrays
   - Definition
   - Umsetzung
   - Syntax
   - Beispiele
   - Referenzverhalten
   - Mehrdimensionale Arrays
   - for-each-Schleife
   - Beispiele

Gliederung

Häufige Fehler

Aufgabe A

- Uneindeutiger Klassenname
- Klassenname nicht auf Englisch

Aufgabe B

- Nur Ausgabe soll in Datei stehen

Häufige Fehler

Aufgabe C

- Keine Kommentare oder triviale Kommentare geben 0 Punkte.
- Was ist ein trivialer Kommentar?
  - //string zu String geändert
- Ihr sollt eure Änderung nicht nur beschreiben sondern auch begründen.

Aufgabe D

- Leichtsinnsfehler bei Auswertung
- Was genau ist der Fehler?
- (byte) (a + b)

Fragen zur Korrektur?

- E-Mail: `christian_zielke@gmx.de`
- Erklären warum ihr der Ansicht seid, dass ihr falsch bewertet wurdet.

So nicht!

```
1   public class Student{ static int Studentcount = 0;
2   String Name1, Name2;
3   int StudentID;
4   public Student(String firstName,
5   String secondname)
6   {
7   Name1 = firstName; Name2 = secondname;
8   StudentCount++;
9   StudentID = Studentcount; }
10  public void Pname(){System.out.println(Name1 + " " + Name2);}}
```

So ist schön!

```
1   public class Student{
2     static int studentCount = 0;
3     String firstName;
4     String secondName;
5     int studentId;
6
7     public Student(String firstName, String secondName) {
8       this.firstName = firstName;
9       this.secondName = secondName;
10      studentCount++;
11      studentId = studentCount;
12    }
13
14    public void printName() {
15      System.out.println(firstName + " " + secondName);
16    }
17  }
```

Codestyle - Checkliste

- Alles auf Englisch (Kommentare nicht verpflichtend)
- Klassenname:
  - UpperCamelCase $\rightarrow$ Erstes Zeichen groß
  - Aussagekräftiger Name, typischerweise Substantive
  - Nur Zeichen aus A-Z, a-z, 0-9

CamelCase

- Alle Wörter zusammengezogen
- Anfangsbuchstaben in der Wortmitte groß
- Erster Buchstabe groß oder klein
- Beispiel: `new customer ID` $\rightarrow$ `newCustomerId`

Checkliste

- Eine Variablendeklaration pro Zeile
- Attribute:
  - am Anfang der Klasse deklarieren
  - aussagekräftiger Name
  - lowerCamelCase $\rightarrow$ Erstes Zeichen klein
- Konstanten:
  - mit `static final` gekennzeichnet
  - in `BLOCK_SCHRIFT` geschrieben

Checkliste

- Konstruktoren:
  - stehen nach den Attributen
  - alle Konstruktoren zusammen gruppiert
- Methoden:
  - lowerCamelCase
  - aussagekräftige Namen, meist Verben
  - logisch gruppiert

Checkliste

- Leerzeichen:
  - zwischen `if, for` usw. und `()`
  - zwischen `}` und `else`
  - um alle Operanden: a = b + c; statt a=b+c;

```
1   for (int i = 0; i < 5; i++) {
2   //Anweisungen
3   }
```

Checkliste

- Blöcke einrücken
- maximal eine Anweisung pro Zeile
- maximal 120 Zeichen pro Zeile
- keine Gott-Klassen
- { und } immer setzen wo sie optional sind
- { nach einem Leerzeichen in die Zeile des Blockstarts
- } in eine eigene Zeile
- keine Magic-Numbers
- Guter Code braucht fast keine Kommentare

Arrays

- Ein Array (dt. Feld) ist eine Menge von Elementen/Variablen
- Alle Elemente haben denselben (Ober-)Typen
- Ein Array hat eine Länge `n`, die die Anzahl der Elemente angibt
- Die Länge ist nach Initialisierung **fest**
- Will man auf ein Element zugreifen, geschieht dies über einen **ganzzahligen** Index
- Der Index geht von 0 (erstes Element) bis `n-1` (letztes Element)

Umsetzung - In gleicher Reihenfolge

- `int[] array;` → Deklariert ein Array (nicht initialisiert!)
- `int[] array;` → Alle Elemente haben hier Typ `int`
- `array = new int[5];` → Initialisiert das Array für 5 `ints`
- `array.length;`
  → Gibt die Länge zurück (Unabhängig vom Datentyp ein `int`)
- `array.length = 5;` → **NICHT ERLAUBT!**
- `array[4];` → Der Index ist hier 4
- `array[0];` kleinster Wert
  `array[array.length - 1];` größter Wert
  → Alles darüber/darunter führt zu Compilerfehlern (ArrayOutOfBoundsException)

Syntax

- `Typ[] array;`
  Deklariert ein Array mit Typ `Typ` und Namen `array`
- `Typ[] array = new Typ[n];`
  Deklariert ein Array, alloziert Speicher für `n` Elemente und initialisiert mit Standardwerten
- `Typ[] array = {val1, val2, ..., valn};`
  Deklariert ein Array, alloziert Speicher für `n` Elemente und initialisiert diese mit val1 - valn
- `Typ a = array[i];`
  Weißt den Inhalt des Elements an Position i des Arrays der Variable a zu
- `int l = array.length;`
  Schreibt die Länge des Arrays in `int l`

Beispiele

- Vektoraddition

```
1   int[] vectorAddition(int[] a, int[] b) {
2     if (a.length != b.length) {
3       return null;
4     }
5     int[] result = new int[a.length];
6     for (int i = 0; i < a.length; i++) {
7       result[i] = a[i] + b[i];
8     }
9     return result;
10  }
```

```
1   char[] string = {'H', 'e', 'l', 'l', 'o', '!'};
2   for(int i = 0; i < string.length; i++) {
3     System.out.print(string[i]);
4   }
```

Referenzverhalten

- Arrays werden von Java wie Objekte behandelt
- Eine Zuweisung eines Arrays an ein anderes **referenziert** nur!
- Problem

```
1       int[] a = {1, 2, 3};
2       int[] b = a; //a und b zeigen auf das gleiche Array!
3       b[0] = 5;
4       //Ausgabe: 5, 5 (nicht 1, 5)
5       System.out.println(a[0] + ", " + b[0]);
```

Referenzverhalten

- Lösung

```
1       int[] a = {1, 2, 3};
2       int[] b = new int[a.length];
3
4       for(int i = 0; i < a.length; i++) {
5         b[i] = a[i];
6       }
7
8       b[0] = 5;
9       //Ausgabe: 1, 5
10      System.out.println(a[0] + ", " + b[0]);
```

- Den Problemcode nennt man **Shallow Copy**
- Den Lösungscode nennt man **Deep Copy**

Mehrdimensionale Arrays

- Ein Array kann als Elemente auch Arrays beinhalten
- Dadurch entsteht eine “Tabelle” oder eine Matrix
- Beispiel: `int[][] matrix = new int[5][3];`
- Iteriert man über ein mehrdimensionales Array, verwendet man meist geschachtelte for-Schleifen (Zahl der Dimensionen = Zahl der for-Schleifen)

for-each-Schleife

- In Java gibt es für Arrays (u.ä.) eine spezielle for-Schleife:
  Die **for-each-Schleife**
- Syntax:

```
for (Typ element : array) {
    Anweisungen
}
```

- Schleifenanweisungen laufen mit jedem Element von `array` einmal
- Das aktuelle Element wird dabei durch `element` repräsentiert
- `element` muss dabei den Typ des `array` haben
- Die for-each-Schleife kann nur lesen, nicht schreiben

Beispiele

- In Arrays schreiben

```
1       int[] array = new int[5];
2
3       for (int element : array) {
4         element = 2;
5       }
6       for (int number : array) {
7         System.out.print(number);
8       }
9       System.out.print(", ");
10
11      for (int i = 0; i < array.length; i++) {
12        array[i] = (i + 1);
13      }
14      for (int i = 0; i < array.length; i++) {
15        System.out.print(array[i]);
16      }
```

- Ausgabe: 00000, 12345

Datenkapselung - Definition

- Schützt Information vor unkontrollierten Zugriff von außen
- Zugriff ist nur über eine definierte Schnittstelle (Getter/Setter) möglich
- Vermindert Abhängigkeit der Klassen untereinander
- Bei Änderung an der internen Implementierung der Klasse/Information muss nichts am Zugriff geändert werden
  $\rightarrow$ Verbesserte Wartbarkeit

Zugriffsrechte und Sichtbarkeit

- Zugriffs- und Sichtbarkeitssteuerung über Modifier `private` und `public`
- Festlegung der Zugriffsrechte bei Deklaration von
  - Klassen (allerdings **nie** private!)
  - Konstruktoren
  - Methoden
  - Attributen

private vs. public

private

- Zugriff nur innerhalb der eigenen Klasse
- Sichtbar nur innerhalb der eigenen Klasse
- Klassen sind in der Regel **nie** `private`
- Konstruktoren von Utility-Klassen sind `private`
- Hilfsmethoden innerhalb der Klasse (u. a. Auslagerung von Berechnungen) sind `private`
- Attribute sind **immer** `private` → Zugriff über Getter/Setter
- Konstanten sind in den meisten Fällen `private`

private vs. public

public

- Zugriff aus allen Klassen des Programms (global)
- Sichtbar in allen Klassen des Programms (global)
- Klassen sind meist `public`
- Konstruktoren sind in der Regel `public`
- Getter, Setter und Nicht-Hilfsmethoden sind **immer** `public`
- Attribute sind **nie** `public`
- Konstanten können in bestimmten Fällen `public` sein

Getter und Setter

```
1   public class MyNumber {
2     private int number;
3
4     public MyNumber(int number) {
5       this.number = number;
6     }
7
8     public int getNumber() {
9       return number;
10    }
11    public void setNumber(int number) {
12      this.number = number;
13    }
14  }
```

packages

- Ein **package** gruppiert zusammengehörenden Klassen
- Zuteilung über das Schlüsselwort `package`
- Syntax: **`package`** `name;`
- Die package-Zeile **muss** sich am Anfang der Datei befinden (noch vor der class-Zeile)
- Alle Klassen eines packages befinden sich in einem Ordner mit dem Namen des packages → `/src/name/*Klassen*`

```
1  package student;
2
3  public class Composition {
4    private Objects in1;
5    private Objects in2;
6    ...
7  }
```

Zugriff auf Klassen

- Beispiel:
  - Paketstruktur: /edu/kit/ipd/jmjrst/deduplicator/gui
  - Ziel: Erzeugen einer neuen Instanz von DeduplicatorGUI
  - edu.kit.ipd.jmjrst.deduplicator.gui.DeduplicatorGUI g =
    new edu.kit.ipd.jmjrst.deduplicator.gui.DeduplicatorGUI();
- Sparen von Schreibarbeit durch **`import`**:

```
1   package edu.kit.ipd.jmjrst.deduplicator;
2
3   //Alternativ mit ...gui.*; alle Klassen aus gui importieren
4   import edu.kit.ipd.jmjrst.deduplicator.gui.DeduplicatorGUI;
5
6   public class Composition {
7     public static void main(String[] args) {
8       DeduplicatorGUI g = new DeduplicatorGUI();
9       ...
10    }
11  }
```

Sichtbarkeit

Durch die Einführung von packages erhält man eine feinere Abstufung der Sichtbarkeit:

- ohne Modifikator Sichtbarkeit zwischen public und private
- Sichtbarkeit im Package

Übersicht

Modifikator	Sichtbarkeit		
	in Klasse	in Paket	„Welt"
`public`	✓	✓	✓
–	✓	✓	–
`private`	✓	–	–

Scopes - Definition

- Der **scope** gibt den Gültigkeitsbereich einer Variablen an
- Attribute sind innerhalb ihrer Klasse gültig
- Lokale Variablen sind innerhalb des Blocks, in dem sie deklariert wurden gültig

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
         int d = 2;
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
         int d = 2;
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
         int d = 2;
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
         int d = 2;
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
         int d = 2;
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
         int d = 2;
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
      }
   }
}
```

Scopes - Quiz

```
public class Scopes {
   private int a;
   public static void main(String[] args) {
      int b = 10;
      for (int i = 0; i < b; i++) {
      ...
      }
      if (...) {
         int c = 1;
      } else {
      }
   }
}
```

Überschattung

- Wenn Variablen mit gleichem Namen in einem Scope deklariert sind, **überschattet** eine Variable die andere
- Lokale Variable - Parameter: In Java **verboten!**
- Lokale Variable - Attribut: Lokale Variable überschattet Attribut
  $\rightarrow$ Lösung mit **this** (allerdings schlechter Stil! Außer im Konstruktor)

Matrix

Zufällige Matrix

- Schreibe eine Methode, die eine Matrix der Größe (row x column) mit zufälligen Zahlen aus dem Intervall [0, 10) erzeugt.
- public static int[][] createRandomMatrix(int row, int column){...}

Matrix ausgeben

- Schreibe eine Methode, die eine Matrix auf der Konsole ausgibt.
- public static void printMatrix(int [][] matrix){...}

Matrix Summen

Summe

- Schreibe eine Methode, die die Zahlen eines Arrays summiert.
- public static long sum(int[] array){...}

Matrix Summe

- Schreibe eine Methode, die die Einträge einer Matrix summiert.
- public static long matrixSum(int[][] matrix){...}

Matrix Addition

- Schreibe eine Methode, die zwei Matrizen addiert.
- public static int[][] addMatrix(int[][] matrixA, int[][] matrixB){...}