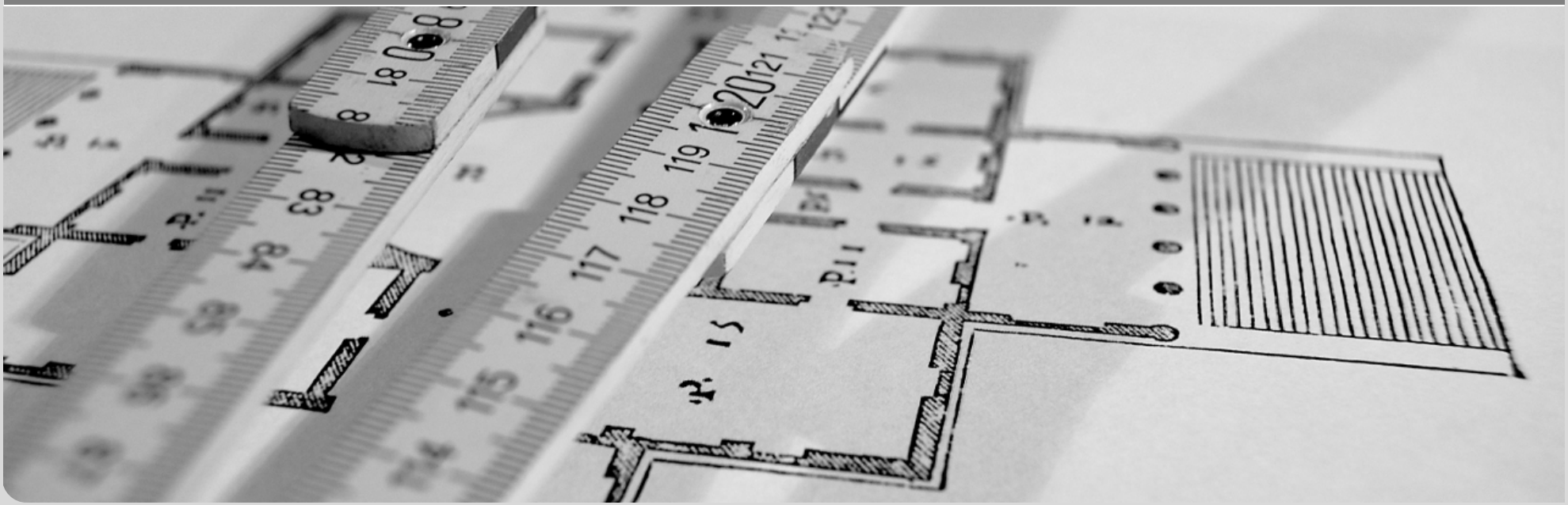
Karlsruher Institut für Technologie

Software Design and Quality

Programmieren
05. Tutorium

Robin Rüde | 1. Dezember 2014



Gliederung

1

Blatt 3

Blatt 3

- Teil A: toString gibt *nicht* etwas auf der Konsole aus, sondern gibt einen String mit der Ausgabe zurück
- Howto: Checkstyle in Eclipse

2

Blatt 2

Blatt 2

- Bei Textabgaben bitte Zeilenumbrüche (wg. Praktomat)
- ^ ist nicht hoch. Auch nicht in Teil D.
- Standardmäßig double nicht float verwenden
- Codeduplikation vermeiden
- String.valueOf() benutzt man normalerweise nicht. Einfach

```
"a: " + a + ", b: " + b
```

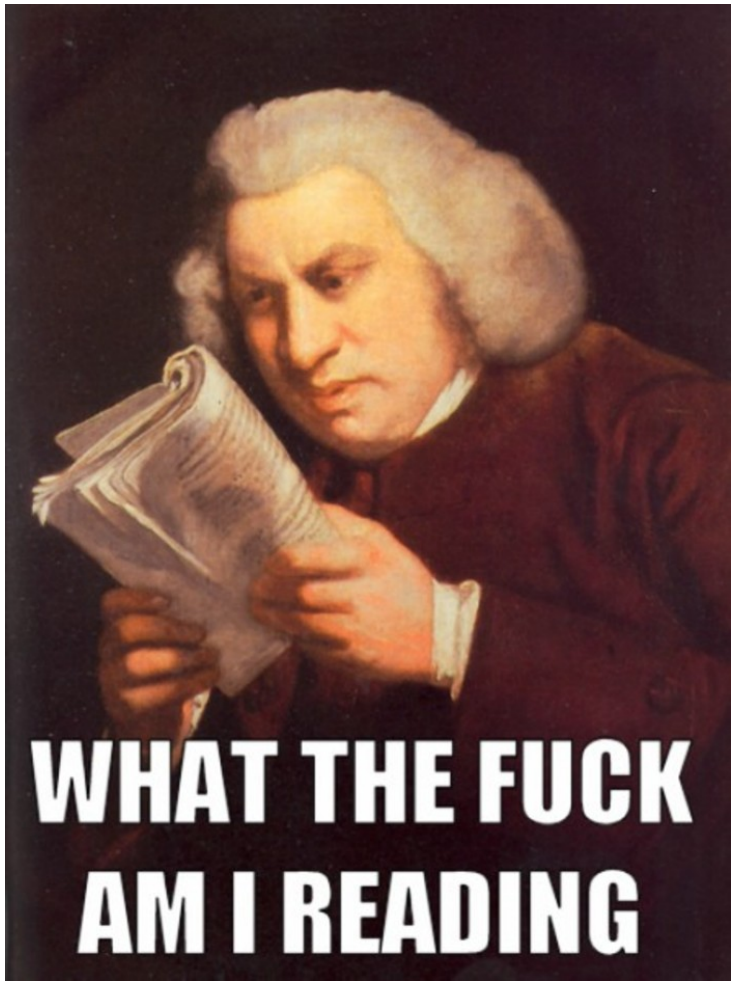
- Metainformationen in toString(), nicht nur viele Zahlen
- Objekt/Klassenvariablen (static)
- Englischer Code!!
- Codestil!!!

Blatt 2: Codestil: wat

```
void calcIops (int iopsuser) {
        int n = 0;
        int x = 0;
        do {
                n = n + 1;
        }
        while (iopsuser >= 10 * n);
                x = 10 * n;
                if (x == iopsuser + 10) {
                        x = x - 10;
        }
        this.iops = x;
}
```

Blatt 2: Codestil: wat

```
void calcIops (int iopsuser) {
        int n = 0;
        int x = 0;
        do {
                n = n + 1;
        }
        while (iopsuser >= 10 * n);
                x = 10 * n;
                if (x == iopsuser + 10) {
                        x = x - 10;
        }
        this.iops = x;
}
```



Blatt 2: Codestil + Static

```
public class Student {
        ...
        static String besuchteVorlesung;
        static String besuchtesTutorium;

        //Welche Vorlesung wird besucht?
        public static void setBesuchteVorlesung(String a, int b) {
                Vorlesung.vorlesungsname = a;
                Vorlesung.vorlesungsnummer = b;
                besuchteVorlesung = b+" "+a;
                }

                String getBesuchteVorlesung() {
                                return besuchteVorlesung;
                        }
```

Blatt 2: Statischer Zähler

```
public class Dozent {
        static String name;
        static String prename;
        static int assistantNumber;
        static int n = 0;
        Dozent(String name, String prename) {
                        Dozent[] array = new Dozent[n];
                for (int i = 0; i < array.length; i++){
                        this.assistantNumber ++;
                        this.n ++;
                }
        }
```

Blatt 2: Statischer Zähler (richtig)

```
class Lecturer {
    static int employeeCounter = 0;

    String firstName, lastName;
    int employeeNumber;
    Lecturer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.employeeNumber = employeeCounter;
        employeeCounter++;
    }
...
```

Blatt 2: Codeduplikation

```
public class approximatePi{
        public static void main(String[] args){
                int i=0;                          //Initialises the loop counter
                double pi=0;                    //Initialises the variable used for storing the results between t
                double pi1=0;                    //Initialising the variables for storing the wanted results 300,
                double pi2=0;
                double pi3=0;
                double pi4=0; ...

                while (i<1001){
                        if(i==300){
                        pi1=pi;
                        pi=pi+(Math.pow(-1,i)/(2*i+1));                                    //number of re
                        i++;}
                        else if(i==400){
                        pi2=pi;
                        pi=pi+(Math.pow(-1,i)/(2*i+1));
                        i++;}
                        else if(i==500){
                        pi3=pi;
                        pi=pi+(Math.pow(-1,i)/(2*i+1));
                        i++;}
                        else if(i==600){
                        pi4=pi;
                        pi=pi+(Math.pow(-1,i)/(2*i+1));
                        i++;}
                        else if(i==700){
                        pi5=pi;
                        pi=pi+(Math.pow(-1,i)/(2*i+1));
                        i++;}
                        else if(i==800){
```

3

Sichtbarkeiten

Datenkapselung

- Verbergen von Daten vor dem Zugriff von außen
- Zugriff nur über festgelegte Schnittstellen (z. Bsp. Methoden)
- schwach gekoppelte Programme $\rightarrow$ Nutzer braucht kein Wissen über interne Implementierung

Zugriffsmodifizierer: public, private

default

- kein Modifier
- Zugriff vom gleichen Paket

Public

- Modifier: public
- Zugriff von beliebigen Paketen und Klassen

Private

- Modifier: private
- Zugriff nur aus der deklarierenden Klasse
- Keine Sichtbarkeit für andere Klassen

Konventionen

- möglichst alle Attribute einer Klasse: `private`
- Zugriff auf Attribute nur via getter/setter
- Methoden i. A. `public` außer es handelt sich nur um lokale Hilfsmethoden
- z.B. eine Methode calculateSize(), die das Attribut size neu berechnet sollte `private` sein, da sie nur intern verwendet wird.

Zugriffsrechte/Sichtbarkeit: Übersicht

Modifier	sichtbar in Klasse	sichtbar in Paket	überall sichtbar
private	✓	✗	✗
default	✓	✓	✗
public	✓	✓	✓

Ab jetzt bitte sinnvoll verwenden!

4

Lebensdauer von Variablen

Lebensdauer von Variablen

- **Gültigkeitsbereich**: Teil des Quelltextes auf den sich die Deklaration der Variablen bezieht
- **Sichtbarkeitsbereich**: Teil des Quelltextes in dem auf die Variable über ihren Namen zugegriffen werden kann
- **Lebensdauer**: Zeitabschnitt der Programmausführung, indem Speicher für die Variable alloziert ist.

Gültigkeit

- Lokale Variablen ab Deklaration bis Ende des Blockes (i. A. bis Klammer des Blocks zu)
- Parameter: innerhalb einer Methode
- Attribute: innerhalb der Klasse, die sie deklariert

Überschattung

- Situation: Variablen mit gleichem Namen gültig. Welche wird durch den Namen angesprochen (Überschattung)?
- Lokale Variablen/Parameter
  - **dürfen nicht** den gleichen Namen haben
  - daher keine Überschattung
- Attribute
  - können gleich heißen wie lokale Variabeln/Parameter
  - bei Namenskonflikt: lokale Variablen haben Vorrang
  - Auflösen des Konflikts: **this.variableName** für Zugriff auf Attribut

Gleichheit von Variablen

- ==: prüft auf Gleichheit der Referenz bzw. bei **primitiven** Datentypen auf Wertgleichheit
- `object1.equals(object2)`: prüft auf innere Objetkgleichheit
- equals sollte wenn Objektvergleiche nötig werden überschrieben werden.

```
public class Vector2D {
    int x, y;
    public boolean equals(Vector2D v) {
        if (v == null) return false;
        return (this.x == v.x) && (this.y == v.y);
    }
}
```

5

Listen

Verkettete Listen

```
public class List {
    private ListCell head;

    public void addFirst(ListCell c) { ... }
    public void addLast(ListCell c) { ... }
    public void remove(ListCell c) { ... }
    public boolean contains(ListCell c) { ... }
}

public class ListCell {
    Content c; // beinhaltete Klasse
    ListCell next; // nächstes Element
    ListCell(Content c, ListCell next) {
        this.c = c;
        this.next = next;
    }
}
```

Iterator

- iteriert über Liste
- nested class (Klasse in der Listenklasse)

```
public class List {
    // ...
    public Iterator iterator() {
        return new Iterator(this.head);
    }
    public class Iterator {
        private ListCell cursor;
        // can only be instantiated by List
        private Iterator(ListCell start) {
            cursor = start;
        }
        public boolean hasNext() {
            return (cursor != null);
        }
        public Content next() {
            Content c = cursor.content;
            cursor = cursor.next;
            return c;
        }
    }
```

Vergleich Listen und Arrays

Arrays

+ random access
- Größe ist fest und muss anfangs angegeben werden
- Löschen und Einfügen nur mit durch vollständiges Kopieren

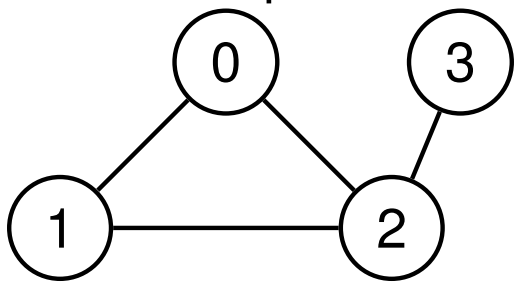
Listen

- Listen können wachsen und schrumpfen
- schnelles Löschen und Einfügen am Anfang und Ende
- Erweiterung: doppelt verkettet mit next und previous-Zeiger

Verwendung von Listen in Graphen

- Adjazenzlisten statt Adjazenzmatrix
- jeder Knoten bekommt eine Liste von „verbundenen“ Knoten

Beispiel:



Adjazenzmatrix:

	0	1	2	3
0	-	1	1	0
1	1	-	1	0
2	1	1	-	1
3	0	0	1	-

Adjazenzliste:

0:	[1,2]
1:	[0,2]
2:	[0,1,3]
3:	[2]

6

Aufgaben

Aufgabe: Matrizenaddition

- Implementiere eine komponentenweise Matrixaddition für 2 Matrizen a und b.
- Beachte: Matrixaddition ist nur für zwei gleichgroße Matrizen definiert
- Gib einen Fehler aus, falls eine Matrixaddition nicht möglich ist.
- Überprüfe deinen Code durch Eingabe von 2 Matrizen und Ausgabe der korrekten berechneten Matrix

Aufgabe: Matrizenmultiplikation

- Implementiere eine Matrixmultiplikation für 2 Matrizen a und b.
- Beachte: Matrixmultiplikation ist nur für zwei gleichgroße, quadratische Matrizen definiert
- Gib einen Fehler aus, falls eine Matrixmultiplikation nicht möglich ist.
- Überprüfe deinen Code durch Eingabe von 2 Matrizen und Ausgabe der korrekten berechneten Matrix

Algorithmus:

$$c_{i,k} = \sum_{j=1}^{n} a_{i,j} \cdot b_{j,k}$$

Eigene Liste

Baue eine eigene Listenklasse, die folgendes kann:

- Speicherung von Elementen vom Typ int
- Methode: public void append(int element) (fügt an das Ende der Liste ein Element mit element als Inhalt an)
- Methode: public int get(int index) (gibt den Inhalt des Elementes an Stell index zurück)
- Optional: einfügen, löschen, etc

Durchschnitt

Schreibe eine Funktion `double average(String line)`, die den Durchschnitt der Zahlen aus dem String line ausrechnet.
Beispiel:

```
average("1,2,3") => 2
average("0,1") => 0.5
```

Tipp: `String.split(",")` teilt einen String an jedem Komma und gibt ein String[] zurück

Durchschnitt 2

Schreibe ein Programm das eine Datei als Argument bekommt und den Durchschnitt jeder Zeile ausgibt.

Beispiel:

```
Input:
0,1
1,2,3,4,5
0,10,-10

Output:
0.5
3
0
```

Datei einlesen

```
BufferedReader in = new BufferedReader(
        new FileReader(filename));
while(in.ready()) {
   String line = in.readLine();
   // do something
}
```

Semifibonacci

Schreibe eine Funktion, die den folgenden Algorithmus für n=1..100 in ein Array (nicht-rekursiv) berechnet

$$f(n) = \begin{cases} 1 & \text{falls } n \leq 1 \\ f(n/2) & \text{falls n gerade} \\ f(n-1) + f(n-2) & \text{falls n ungerade} \end{cases}$$

Ende

Fragen?

Ende

Fragen?
Vielen Dank für eure Aufmerksamkeit!