technische universität dortmund

Fakultät für Informatik
Lehrstuhl 14
Lars Hildebrand

Praktikum zu

Einführung in die Informatik für LogWings, WiMas und MedPhys

Wintersemester 2020/21

Übungsblatt 10

Besprechung:
1.–5.02.2021 (KW 5)

Vorbereitende Aufgaben

Aufgabe 10.1: Wiederholung: Klammern

Quiz

Welche Klammer wird wofür verwendet?

i) Kennzeichnung von Blöcken
   a) [...] b) (...) c) {...} ✓

ii) Parameter-Angabe bei Funktionen
   a) [...] b) (...) ✓ c) {...}

iii) Definition von Arrays und Zugriff auf Arrayelementen
   a) [...] ✓ b) (...) c) {...}

iv) Priorisierung von Berechnungen bzw. Auswertung
   a) [...] b) (...) ✓ c) {...}

Aufgabe 10.2: Wiederholung: Funktionsparameter

Geben Sie einen geeigneten Methodenkopf für die folgenden öffentlichen, statischen Funktionen an.

a) Eine Funktion **average**, die den Durchschnitt eines **double**-Arrays berechnet und zurückgibt.

public static double average(double[] array)

b) Eine Funktion **plus**, die zwei reelle Zahlen miteinander addiert und die Summe zurückgibt.

public static double plus(double a, double b)

c) Eine Funktion **countWords**, die die Wörter in einem **String** zählt und zurückgibt.

public static int countWords(String str)

d) Eine Funktion **printMaximum**, die das Maximum eines **int**-Arrays mit `System.out.println` auf dem Bildschirm ausgibt.

public static void printMaximum(int[] array)

e) Eine Funktion **times**, die einen Integer **n** und einen Integer **x** entgegen nimmt und ein **n** Elemente langes Array, gefüllt mit dem Wert **x** zurückgibt.

public static int[] times(int n, int x)

Präsenzaufgaben

Aufgabe 10.3: Fehlersuche

In den nachfolgenden Klassen haben sich syntaktische und semantische Fehler eingeschlichen. Korrigieren Sie sämtliche Fehler in den Klassen auf dem Blatt. Der Programmcode soll dabei **nicht** implementiert werden.

```
using java.util.Random;

public class Player {
    private int score;
    private String name;
    private Random d20;

    private Player(String name) {
        name = name;
        d20 = new Random();
        score = 0;
        public getName() {
            return name;
    }

    public int getScore {
        return score;
    }

    public String getInfo() {
        return getName() + " with " + getScore() + "points ";
    }

    public void addScore(int score); {
        return null;
        score += score;
    }

    public void int throwDice() {
        return d20.nextInt(20) + 1;
}
```

```
public classe Game {

    private Player[] players;
    private Random d20;

    public Game(double n) {
        d20 = new Random();
        players = new Player[n];
        for (int i = 0, i < n, i++); {
            players[i] = new Player("Player" + i);
```

```
11          }
12      }
13
14      public static int throwDice() {
15          return d20.nextInt(20) + 1;
16      }
17
18      public protected void playRound() {
19          for (int i = 0; j < players.lenght; i++) {
20              if (players[i].throwDice() > throwDice() {
21                  players[i].addScore(10.0);
22              } else {
23                  players[i].addScore(-10.0);
24              }
25          }
26      }
27
28      public Player getWinner[] {
29          Player winner = players(0);
30          for (int i = 1; i < players.lenght; i++) {
31              if (winner.getScore() < player[i].getScore()) {
32                  winner = player[i];
33              }
34          }
35          return winner;
36      }
37  }
```

```
1   public class Program {
2
3       public stativ void main(String args) {
4           Game game = new Game(10);
5           for (int i = 0; i < 100; i++)
6               game.playRound();
7           Player winner = game.getWinner();
8           System.out.println("The first winner is " + winner.getInfo();
9       }
10
11      public stativ void main(String args) {
12          Game game = new Game(25);
13          for (int i = 0; i < 100; i++)
14              game.playRound();
15          Player winner = game.getWinner();
16          System.out.println("The second winner is " + winner.getInfo();
17  }
```

Komplettlösung

zu findende Fehler:

- 11 Fehler in Player:
  - Zeile 1: import statt using
  - Zeile 8: public oder ohne Modifier anstelle von private (private Konstruktor würde in Game zu Fehler führen und ein private Konstruktor ist hier nicht sinnvoll)
  - Zeile 9: this.name = name;

  - Zeile 11f: vor public fehlt }
  - Zeile 12: String als Rückgabe fehlt
  - Zeile 16: () fehlen
  - Zeile 24: ; hinter Methodenkopf muss entfernt werden
  - Zeile 25f: unnötiges return, damit ist Zeile 28 toter Code und null kann nicht in einer Methode ohne Rückgabe zurückgegeben werden
  - Zeile 26: this.score += score;
  - Zeile 29: nur int und nicht void int
  - Zeile 31: eine } fehlt

- 16 Fehler in Game:
  - Zeile 1: fehlendes import von Random
  - Zeile 1: class statt classe
  - Zeile 6: int statt double, weil n später für die Länge eines Arrays genutzt wird
  - Zeile 9: ; anstelle der , im Schleifenkopf
  - Zeile 9: kein ; direkt nach Schleifenkopf
  - Zeile 14: kein static, weil d20 nicht statisch
  - Zeile 18: kein protected
  - Zeile 19: i statt j, j wurde nicht deklariert
  - Zeile 19: length statt lenght
  - Zeile 20: fehlende)
  - Zeile 21 und 23: 10 anstelle von 10.0, da addScore(int score)
  - Zeile 28: () statt []
  - Zeile 29: [] statt ()
  - Zeile 30: length statt lenght
  - Zeile 31f: players[i] statt player[i]

- 4 Fehler + 2 Folgefehler in Program:
  - Zeile 3: static statt stativ
  - Zeile 3: String[] statt String
  - Zeile 8: fehlende) nach getInfo()
  - Zeile 11ff: zweite main-Methode; Methodenrumpf muss in erster main-Methode übertragen werden
  - Zeile 12: *Game game2 = new...* oder *game = new ...*, ansonsten würde in der main-Methode zweimal game deklariert
  - Zeile 15: analog wie Zeile 12 für winner (*Player winner2=* oder *winner=*)

  ⇒ für die letzten beiden Fehler müssen dementsprechend weitere Vorkommen von game bzw. winner in den darauffolgenden Zeilen angepasst werden (also konsequent: game2 bzw. winner2 in den Zeilen 12-16 bspw.)

Aufgabe 10.4: Wiederholung: Rekursion

Programmieren Sie die Fibonacci-Folge rekursiv. Die Fibonacci-Folge ist wie folgt definiert:

$$\text{fib}(n) = \begin{cases} 0 & \text{wenn } n = 0 \\ 1 & \text{wenn } n = 1 \\ \text{fib}(n-2) + \text{fib}(n-1) & \text{sonst} \end{cases}$$

Lösung

```java
public static int fibRec(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fibRec(n-2) + fibRec(n-1);
    }
}
```

Aufgabe 10.5: Objektvariablen und -methoden vs. Klassenvariablen und -methoden

In dieser Aufgabe wollen wir uns mit der unterschiedlichen Verwendung von Objekt- und Klassenelementen vertraut machen. Manche Zuweisungen und Methodenaufrufe sind im unteren Programm nicht erlaubt (vgl. dazu die Folien in Kapitel 6). Notieren Sie auf den Linien neben dem Programmtext, ob die jeweilige Zuweisung oder der jeweilige Methodenaufruf erlaubt ist oder nicht.

```java
class Tester {
    int var1;
    static int var2;

    void test1() {
        var1++;                                  // erlaubt
        var2--;                                  // erlaubt
    }

    static void test2() {
        var1++;                                  // nicht erlaubt
        var2--;                                  // erlaubt
    }

    public static void main(String[] args) {
        var1 = 1;                                // nicht erlaubt
        var2 = 1;                                // erlaubt
        test1();                                 // nicht erlaubt
        test2();                                 // erlaubt

        Tester testObjekt = new Tester();
        testObjekt.var1 = 2;                     // erlaubt
        testObjekt.var2 = 2;                     // erlaubt aber gegen Konvention
        testObjekt.test1();                      // erlaubt
        testObjekt.test2();                      // erlaubt aber gegen Konvention

        Tester.var1 = 3;                         // nicht erlaubt
        Tester.var2 = 3;                         // erlaubt
        Tester.test1();                          // nicht erlaubt
        Tester.test2();                          // erlaubt
    }
}
```

Aufgabe 10.6: Klassenvariablen Implementierung

Erweitern Sie die Klassen **Car** und **Vehicle** um je eine private Klassenvariable **carCounter** bzw. **vehicleCounter**, die die Anzahl der erzeugten **Car**- bzw. **Vehicle**-Objekte zählt. Geben Sie anschließend in Ihrer Testklasse die Anzahl der Instanziierungen aus.

Hinweis: Sie benötigen hierzu eine funktionierende Lösung der Aufgaben aus Blatt 9.

Lösung

```java
public class Vehicle {
    private int wheels;
    private String fuel;
    private static int vehicleCounter = 0; /* ergaenzt */

    public Vehicle(int inWheels, String inFuel) {
        wheels = inWheels;
        fuel = inFuel;
        vehicleCounter++;   /* ergaenzt */
    }
```

```
11
12      public int getWheels() {
13          return wheels;
14      }
15
16      /* Methode ergaenzt */
17      public static int getVehicleCounter() {
18          return vehicleCounter;
19      }
20
21      public void print() {
22          System.out.println("Fuel: "+fuel);
23          System.out.println("Wheels: "+wheels);
24      }
25  }
```

```
1   public class Car {
2
3       private String manufacturer;
4       private String model;
5       private int horsePower;
6       private static int carCounter = 0; /* ergaenzt */
7
8       public static final double WATT_PER_HORSEPOWER = 735.5;
9
10      public Car(String inManufacturer, String inModel, int inHorsePower) {
11          manufacturer = inManufacturer;
12          model = inModel;
13          horsePower = inHorsePower;
14          carCounter++;       /* ergaenzt */
15      }
16
17      public double getPower() {
18          return horsePower * WATT_PER_HORSEPOWER;
19      }
20
21      /* Methode ergaenzt */
22      public static int getCarCounter() {
23          return carCounter;
24      }
25
26      public void print() {
27          System.out.println(model + " von " + manufacturer
28              + " mit " + getPower() + " Watt Leistung");
29      }
30
31  }
```

Ergänzende Aufgaben

Aufgabe 10.7: Minimum rekursiv

In einer Aufabe von Blatt 8 haben Sie eine Funktion geschrieben, die das Minimum eines **int**-Arrays findet. Höchstwahrscheinlich haben Sie dies iterativ mit einer **for**-Schleife gelöst. Diese Funktion wollen wir nun rekursiv implementieren. Legen Sie dazu die Klasse **MinRec** an. Verwenden Sie bei der Implementierung *keine* Schleifen!

a) Schreiben Sie nun eine Funktion **minArray** mit einem **int**-Array und einem Index als Parameter. Die Funktion soll das Minimum des Arrays ab dem Index zurückgeben.

   Beispiel: Sei das Array `a = {30, 10, 50, 20, 40, 60}`, soll `minArray(a, 2)` den Rückgabewert 20 haben.

b) Überladen Sie die Funktion **minArray** mit einer Funktion, die nur ein **int**-Array als Parameter hat. Diese soll das Minimum des **ganzen** Arrays zurückgeben. Rufen Sie dazu die soeben geschriebene **minArray**-Funktion auf.

Komplettlösung

```
1  public class MinRec {
2      public static int minArray(int[] array, int i) {
3          if (i == array.length - 1) {
4              return array[i];
5          }
6
7          int nextMin = minArray(array, i + 1);
8          if (nextMin < array[i]) {
9              return nextMin;
10         } else {
11             return array[i];
12         }
13     }
14
15     public static int minArray(int[] array) {
16         return minArray(array, 0);
17     }
18
19     public static void main(String[] args) {
20         int[] a = {30, 10, 50, 20, 40, 60};
21
22         System.out.println(minArray(a, 2));
23         System.out.println(minArray(a));
24     }
25 }
```