

Lösungen: Einführung in die Programmierung, 19.02.2018

Ausgangsdatei: 09_EidP_Klausur_19.02.2018.pdf

Die Lösungen umfassen alle Aufgaben 1 bis 8 einschließlich sämtlicher Teilaufgaben. Java-Code ist jeweils in die angegebene Klasse einzusetzen; öffentliche Klassen gehören beim Übersetzen in getrennte, entsprechend benannte Dateien. Die auf dem Deckblatt genannten Bonuspunkte sind keine zusätzliche Aufgabenstellung.

Aufgabe 1: Multiple Choice und Wissen

a) Aussagen in der Reihenfolge der Vorlage

Zahlendarstellungen:

Aussage	Bewertung und Begründung
106 hat binär die Darstellung 111010.	Falsch. $111010_2 = 58$; $106 = 64 + 32 + 8 + 2 = 1101010_2$.
106 hat hexadezimal die Darstellung 6A.	Richtig. $6 \cdot 16 + 10 = 106$.
Die Binärdarstellung von 106 benötigt 7 Stellen.	Richtig. $64 \leq 106 < 128$. Führende Nullen werden nicht mitgezählt.
A6 entspricht dezimal 167.	Falsch. $10 \cdot 16 + 6 = 166$.

Java-Bezeichner:

Zeichenfolge	Bewertung und Begründung
Snoke	Richtig: zulässige Buchstabenfolge, kein Schlüsselwort.
Jabba	Richtig: zulässige Buchstabenfolge, kein Schlüsselwort.
R2-D2	Falsch: ein Bindestrich ist in einem Bezeichner nicht zulässig.
Leia Organa	Falsch: das Leerzeichen trennt zwei Bezeichner.

Objektattribut mit Paketsichtbarkeit:

Aussage	Bewertung und Begründung
Wird in UML unterstrichen.	Falsch. Unterstreichungen kennzeichnen ein Klassenattribut, also static.
Wird in UML durch ~ gekennzeichnet.	Richtig. ~ bezeichnet Paketsichtbarkeit.
Wird in Java durch static gekennzeichnet.	Falsch. static würde es zum Klassenattribut machen.
Wird in Java durch package gekennzeichnet.	Falsch. Paketsichtbarkeit entsteht durch Weglassen eines Sichtbarkeitsmodifizierers.

Abstrakte Klasse A:

Aussage	Bewertung und Begründung
A darf keinen Konstruktor besitzen.	Falsch. Ein Konstruktor initialisiert den A-Anteil von Unterklassenobjekten.
Von A können keine Objekte erzeugt werden.	Richtig im gemeinten Sinn: <code>new A()</code> ist verboten. Konkrete Unterklassen dürfen instanziiert werden.
Von A dürfen keine Unterklassen gebildet werden.	Falsch. Gerade dafür eignen sich abstrakte Klassen.
A darf nur abstrakte Methoden besitzen.	Falsch. Auch vollständig implementierte Methoden sind erlaubt.

b) Mehrfachvererbung

Mehrfachvererbung bedeutet, dass ein Typ unmittelbar von mehreren Obertypen erbt. Java erlaubt einer Klasse nur eine direkte Oberklasse; sie kann aber mehrere Interfaces implementieren. Ein Interface kann seinerseits mehrere Interfaces erweitern.

Aufgabe 2: Einlesen und Auswahlmenü

```
import java.util.Scanner;

public class Auswahl {
    public static void zeigeMenue() {
        System.out.println("(1) Zeige die 1. bis 7. eingegebene Zahl");
        System.out.println("(2) Zeige die achte eingegebene Zahl");
        System.out.println("(3) Zeige die neunte eingegebene Zahl");
        System.out.println("(4) Beende das Programm");
    }

    public static void main(String[] args) {
        Scanner eingabe = new Scanner(System.in);
        int[] zahlen = new int[9];
        for (int i = 0; i < zahlen.length; i++) {
            zahlen[i] = eingabe.nextInt();
        }
        zeigeMenue();
        int auswahl;
        do {
            auswahl = eingabe.nextInt();
            switch (auswahl) {
                case 1:
                    for (int i = 0; i < 7; i++) {
                        System.out.println(zahlen[i]);
                    }
                    break;
                case 2:
                    System.out.println(zahlen[7]);
                    break;
                case 3:
                    System.out.println(zahlen[8]);
                    break;
                case 4:
                    break;
                default:
                    System.out.println("Bitte eine Zahl von 1 bis 4 eingeben.");
            }
        } while (auswahl != 4);
    }
}
```

Die neun Zahlen und das Menü stehen vor der Wiederholung und werden deshalb nur einmal eingelesen beziehungsweise ausgegeben. Arrayindizes beginnen bei 0. Die Auswahl erfolgt ausschließlich mit switch; if und if-else werden nicht benötigt. Die break-Anweisungen verhindern das Weiterlaufen in den nächsten Fall.

Aufgabe 3: Teiler und Primfaktoren

a) Anzahl der positiven Teiler

```
public static int teileranzahl(int n) {
    // Voraussetzung: n > 0
    int anzahl = 1; // n selbst ist immer ein Teiler.
    for (int t = 1; t <= n / 2; t++) {
        if (n % t == 0) {
            anzahl++;
        }
    }
    return anzahl;
}
```

Außer n selbst ist jeder positive Teiler höchstens $n/2$. Ein Rest von 0 bedeutet, dass die Division ganzzahlig aufgeht. Für 6 werden 1, 2 und 3 gefunden; mit 6 ergibt das 4. Für 1 ist das Ergebnis 1. Die Schleifengrenze vermeidet zudem ein Überlaufen des Schleifenzählers bei `Integer.MAX_VALUE`.

b) Primzahlzerlegung

```
public static String primzahlzerlegung(int n) {
    // Voraussetzung: n > 0
    if (n == 1) {
        return "1";
    }
    String ergebnis = "";
    int rest = n;
    boolean ersterFaktor = true;

    for (int t = 2; t <= rest / t; t++) {
        while (rest % t == 0) {
            if (!ersterFaktor) {
                ergebnis += "*";
            }
            ergebnis += t;
            ersterFaktor = false;
            rest /= t;
        }
    }
    if (rest > 1) {
        if (!ersterFaktor) {
            ergebnis += "*";
        }
        ergebnis += rest;
    }
    return ergebnis;
}
```

Die Kandidaten werden aufsteigend ausprobiert. Sobald ein Kandidat teilt, wird er so oft wie möglich herausdividiert. Zusammengesetzte Kandidaten können dann nicht mehr teilen, denn ihre kleineren Primfaktoren wurden bereits entfernt. Endet die Schleife mit $\text{rest} > 1$, ist dieser Rest prim. Die Bedingung $t \leq \text{rest}/t$ entspricht $t^2 \leq \text{rest}$, ohne das Quadrat mit möglichem int-Überlauf zu bilden. Für 36 entsteht $2*2*3*3$.

Die Aufgabenstellung ist für $n = 1$ ungenau: 1 ist keine Primzahl und hat keine Primfaktoren; mathematisch ist seine Zerlegung das leere Produkt. Hier wird dafür ausdrücklich die praktische Darstellung "1" gewählt. Es werden keine Bibliotheksmethoden aufgerufen.

Aufgabe 4: Zweidimensionale Arrays

a) Transponierte

```
public static int[][] berechneTransponierte(int[][] feld) {
    int zeilen = feld.length;
    int spalten = zeilen == 0 ? 0 : feld[0].length;
    int[][] transponiert = new int[spalten][zeilen];
    for (int z = 0; z < zeilen; z++) {
        for (int s = 0; s < spalten; s++) {
            transponiert[s][z] = feld[z][s];
        }
    }
    return transponiert;
}
```

Ein Element an Zeile z , Spalte s kommt an Zeile s , Spalte z . Aus einem 3×4 -Feld wird ein 4×3 -Feld. Das Original bleibt erhalten. Bei einem äußeren Array der Länge 0 ist in Java keine Spaltenzahl ablesbar; dafür wird ein leeres 0×0 -Ergebnis verwendet.

b) Symmetrie

```
public static boolean istSymmetrisch(int[][] feld) {
    int n = feld.length;
    if (n == 0) {
        return true;
    }
    if (feld[0].length != n) {
        return false;
    }
    for (int z = 0; z < n; z++) {
        for (int s = z + 1; s < n; s++) {
            if (feld[z][s] != feld[s][z]) {
                return false;
            }
        }
    }
    return true;
}
```

Ein symmetrisches Feld muss quadratisch sein und für alle Positionen $\text{feld}[z][s] = \text{feld}[s][z]$ erfüllen. Jedes Paar muss nur einmal verglichen werden; die Diagonale stimmt immer mit sich selbst überein. Das leere Feld gilt hier als symmetrisch.

Aufgabe 5: Klassen und Objekte

a) Fortlaufende Personennummern

Zusätzliches Attribut in Person und ergänzter Konstruktor:

```
private static int anzahlPersonen = 0;

public Person(String name, int alter) {
    this.name = name;
    this.alter = alter;
    this.id = ++anzahlPersonen;
}
```

Das Klassenattribut gehört allen Personen gemeinsam. Der Präinkrementoperator erhöht erst auf 1 beziehungsweise 2 und liefert dann die jeweilige id. Ein gewöhnliches Objektattribut als Zähler würde dagegen bei jedem neuen Objekt wieder von 0 beginnen.

b) Objekte und Referenzen erzeugen

```
Kfz porsche = new Kfz("Porsche", p1);
Person p3 = new Person("Hans", 40);
Kfz vwkaefer = new Kfz("VW Käfer", p3);
```

Der Porsche verweist auf die bereits vorhandene Person Rolf. Hans ist die dritte erzeugte Person und erhält id = 3. Die bereits erzeugte Person Gudrun bleibt über p2 erreichbar, auch wenn sie im zweiten Ausschnitt des Objektdiagramms nicht erneut dargestellt ist.

c) Gewünschte Textausgabe

In Person:

```
@Override
public String toString() {
    return name + ", " + alter + " Jahre alt";
}
```

In Kfz:

```
@Override
public String toString() {
    return bezeichnung + " mit Besitzer " + besitzer;
}
```

println ruft die Textdarstellung des Kfz auf. Die Verkettung mit besitzer benutzt wiederum Person.toString(). Damit ergeben sich exakt:

```
Porsche mit Besitzer Rolf, 60 Jahre alt
VW Käfer mit Besitzer Hans, 40 Jahre alt
```

Aufgabe 6: UML nach Java

a) Assoziation, b) Konstruktor und c) Methoden

Die vorgegebenen Klassen Gerät und Sender werden wie im UML-Diagramm vorausgesetzt. Das Radio besitzt höchstens 100 Favoriten. Die belegten Plätze sind zusammenhängend von Index 0 bis `anzahlFavoriten - 1`.

```
public interface IPGerät {
    String getIPAdresse();
}

public class InternetRadio extends Gerät implements IPGerät {
    private String ipAdresse;
    private Sender[] favoriten = new Sender[100];
    private int anzahlFavoriten = 0;

    public InternetRadio(int id, String hersteller,
                        String ipAdresse) {
        super(id, hersteller);
        this.ipAdresse = ipAdresse;
    }

    @Override
    public String getIPAdresse() {
        return ipAdresse;
    }

    public int getAnzahlFavoriten() {
        return anzahlFavoriten;
    }

    public int addFavorit(String bezeichnung, String genre) {
        if (anzahlFavoriten == favoriten.length) {
            return -1;
        }
        int index = anzahlFavoriten;
        favoriten[index] = new Sender(bezeichnung, genre);
        anzahlFavoriten++;
        return index;
    }

    public boolean removeFavorit() {
        if (anzahlFavoriten == 0) {
            return false;
        }
        for (int i = 1; i < anzahlFavoriten; i++) {
            favoriten[i - 1] = favoriten[i];
        }
        anzahlFavoriten--;
        favoriten[anzahlFavoriten] = null;
        return true;
    }

    public Sender getFavorit(int index) {
        if (index < 0 || index >= anzahlFavoriten) {
            return null;
        }
        return favoriten[index];
    }
}
```

```
public void ausgebenFavoriten() {  
    for (int i = 0; i < anzahlFavoriten; i++) {  
        Sender sender = favoriten[i];  
        System.out.println(sender.getBezeichnung()  
                            + ", " + sender.getGenre());  
    }  
}
```

Das Array und der Zähler verwalten die Assoziation. super initialisiert den Gerät-Anteil. Das Interface verlangt eine öffentliche Implementierung von getIPAdresse. Beim Entfernen rücken alle späteren Referenzen nach links; der freigewordene letzte Platz wird auf null gesetzt. System.out ist für die ausdrücklich geforderte Konsolenausgabe erforderlich; weitere Hilfsklassen oder Sammlungen werden nicht benutzt.

Aufgabe 7: Speicherabbild

Zum Zeitpunkt 1 befindet sich die Ausführung in A.methode auf dem durch b referenzierten B-Objekt. Benenne die beiden Heapobjekte O1 und O2:

Speicherbereich	Inhalt
Klassenattribut A.c	18
Objekt O1, Typ A	A.m = 5; A.n = 0
Objekt O2, Typ B	A.m = 11; A.n = 0; B.m = 8
main: a	Referenz auf O1
main: b	Referenz auf O2; deklarierter Typ A
main: args	null gemäß der vorgegebenen Zeichnung
A.methode: this	Referenz auf O2
A.methode: i	6

Im Konstruktor A wird das Attribut n nie beschrieben: Der Parameter n verdeckt es, ohne dass `this.n = n` ausgeführt wird. Deshalb bleibt A.n in beiden Objekten beim Standardwert 0. c wird zunächst $1 + 5 + 3 = 9$ und danach $9 + 5 + 4 = 18$.

O2 startet mit A.m = 5 und B.m = $1 + 1 = 2$. A.methode(6) setzt A.m auf -1. Der virtuelle Aufruf `erhoehe(6)` führt B.erhoehe aus: B.m wird 8; `super.erhoehe` setzt A.m auf $-1 + 12 = 11$. Die `erhoehe`-Aufrufe sind am markierten Zeitpunkt bereits beendet und haben keine aktiven Aufrufrahmen mehr. A.m und B.m sind zwei getrennte Felder, obwohl beide m heißen.

Aufgabe 8: Programmausgabe

A: 11
A: 12
A: 2
B: 11
B: 101
B: 5
B: 101
B: 8
C: 1000
C: 9
C: 1000

Beim ersten A-Objekt erhöht der Konstruktor x auf 11 und z auf 1. A.print erhöht beide Werte noch einmal und gibt 12 und 2 aus.

Beim B-Objekt erhöht der A-Konstruktor das Feld A.x auf 11 und z auf 3; der B-Konstruktor erhöht sein eigenes Feld B.x auf 101 und z auf 4. einB ist als A deklariert, deshalb liest einB.x das Feld A.x = 11. Der Methodenaufruf einB.print() wird dagegen dynamisch an B gebunden: ausgegeben werden B.x = 101 und das auf 5 erhöhte z.

Beim C-Objekt laufen A- und B-Konstruktor erneut; z steigt auf 7. C.x erhält 1000. C.print ruft zunächst B.print auf, das 101 und z = 8 ausgibt. Danach folgen C.x = 1000 und z = 9. Der abschließende Feldzugriff auf einC.x ergibt erneut 1000. Das static-Feld z existiert nur einmal für die gesamte Hierarchie.