

Lösungen zur Klausur vom 10.03.2021

Quelle: 15_EidP_Klausur_A_10_03_2021.pdf. Einführung in die Programmierung, Klausur A. Alle vier Aufgaben sind bearbeitet. Der Bonuspunkte-Eintrag auf dem Deckblatt enthält keine zusätzliche Aufgabe.

Aufgabe 1 - Hexadezimaldarstellung

Bei der wiederholten Division durch 16 notiert man die Reste. Die Reste von 10 bis 15 werden als A bis F geschrieben. Anschließend liest man die Reste in umgekehrter Reihenfolge.

$47 : 16 = 2$ Rest 15, also F. Danach: $2 : 16 = 0$ Rest 2. Damit gilt: 47 (dezimal) = 2F (hexadezimal). Die Rückrechnung lautet $2 \cdot 16 + 15 = 47$.

Aufgabe 2 - Zeichen löschen

Die folgenden beiden Methoden gehören zur selben Klasse. Als Eingabe wird ein von null verschiedener String vorausgesetzt. Es werden ausschließlich die erlaubten String-Methoden `length()` und `charAt()` verwendet; die Verkettung mit `+` ist ein Sprachoperator.

```
public class ZeichenFinden {
    public String löscheErstesVorkommen(String w, char c) {
        String ergebnis = "";
        boolean geloescht = false;
        for (int i = 0; i < w.length(); i++) {
            char aktuell = w.charAt(i);
            if (!geloescht && aktuell == c) {
                geloescht = true;
            } else {
                ergebnis = ergebnis + aktuell;
            }
        }
        if (!geloescht) return w;
        return ergebnis;
    }

    public String löscheAlleVorkommen(String w, char c) {
        String ergebnis = "";
        for (int i = 0; i < w.length(); i++) {
            if (w.charAt(i) != c) {
                ergebnis = ergebnis + w.charAt(i);
            }
        }
        return ergebnis;
    }
}
```

i) Erstes Vorkommen

`geloescht` wird genau beim ersten Treffer auf `true` gesetzt. Danach werden auch weitere Vorkommen von `c` übernommen. Bei "abbA" und 'b' wird nur das Zeichen an Index 1 übersprungen: Ergebnis "abA". Ohne Treffer wird `w` unverändert zurückgegeben.

ii) Alle Vorkommen

Ein Zeichen wird nur übernommen, wenn es von `c` verschieden ist. Dadurch verschwinden alle Treffer, während die Reihenfolge der übrigen Zeichen erhalten bleibt. Aus "abbA" wird "aA". Ein leerer Eingabestring ergibt in beiden Methoden den leeren String.

Aufgabe 3 - Temperaturwerte für sieben Tage

Eine Zeile enthält genau die bisher gemessenen Werte eines Tages. Leere Zeilen haben Länge 0. Da Arrays ihre Länge nicht ändern können, erzeugt das Hinzufügen für den betroffenen Tag ein neues, um ein Element längeres Array.

```
public class Temperaturen {
    public double[][] legeFeldAn() {
        return new double[7][0];
    }

    public double[][] fuegeWertHinzue(
        double[][] feld, int tag, double wert) {
        if (feld == null || feld.length != 7
            || tag < 0 || tag >= 7) {
            return null;
        }
        int alt = feld[tag] == null ? 0 : feld[tag].length;
        double[] neu = new double[alt + 1];
        for (int i = 0; i < alt; i++) {
            neu[i] = feld[tag][i];
        }
        neu[alt] = wert;
        feld[tag] = neu;
        return feld;
    }

    public double berechneDurchschnitt(double[][] feld, int tag) {
        if (feld == null || feld.length != 7
            || tag < 0 || tag >= 7
            || feld[tag] == null || feld[tag].length == 0) {
            return 0.0;
        }
        double summe = 0.0;
        for (int i = 0; i < feld[tag].length; i++) {
            summe += feld[tag][i];
        }
        return summe / feld[tag].length;
    }
}
```

i) Feld anlegen

`new double[7][0]` erzeugt sieben vorhandene, leere Zeilen. Es werden noch keine Messwerte gespeichert. Eine Zeile mit einem einzelnen Wert 0 wäre dagegen bereits ein Temperaturmesswert und daher nicht passend.

ii) Messwert hinzufügen

Zuerst werden die vorgegebenen Fehlerfälle geprüft. Die neue Zeile übernimmt alle bisherigen Werte und setzt den neuen Wert an den letzten Index. Nur die Referenz der betreffenden Tageszeile wird ausgetauscht; die übrigen Tage bleiben erhalten. Eine null-Zeile wird zusätzlich wie eine leere Zeile behandelt.

iii) Tagesdurchschnitt

Der Mittelwert ist die Summe der Tageswerte geteilt durch ihre Anzahl. Bei 10,0 und 14,0 ergibt sich 12,0. Für einen Tag ohne Messwerte ist kein Durchschnitt definiert; diese Lösung gibt dann ebenso wie bei fehlerhaften Parametern 0,0 zurück. Dieser Rückgabewert ist eine Fehler-/Leerfallkonvention und kein gemessener Mittelwert.

Aufgabe 4 - Garten, Pflanze und Busch

Das Diagramm verlangt ein Interface Pflanze, dessen Implementierung Busch und einen Garten mit 1 bis 100 Pflanzen. Die Assoziation ist vom Garten zu den Pflanzen navigierbar. Für jeden öffentlichen Typ wird eine gleichnamige Java-Datei verwendet.

Interface und Klasse Busch

```
public interface Pflanze {
    int getFlaeche();
}

public class Busch implements Pflanze {
    protected int flaeche;

    public Busch(int flaeche) {
        this.flaeche = flaeche;
    }

    public int getFlaeche() {
        return flaeche;
    }
}
```

Die Interface-Methode ist implizit public und abstract. Die Implementierung muss daher public sein. Das UML-Zeichen # beim Attribut flaeche bedeutet protected.

Garten und Verwaltung der Assoziation

Für den Konstruktor werden nichtnegative Abmessungen und eine vorhandene erste Pflanze vorausgesetzt. Pflanzen haben nichtnegative Flächen. Die erste Pflanze im Konstruktor erfüllt die UML-Untergrenze 1. Die letzte Pflanze darf nicht entfernt werden.

```
public class Garten {
    private int laenge;
    private int breite;
    private Pflanze[] pflanzen = new Pflanze[100];
    private int anzahl;

    public Garten(int laenge, int breite, Pflanze erste) {
        this.laenge = laenge;
        this.breite = breite;
        pflanzen[0] = erste;
        anzahl = 1;
    }

    public boolean fuegeHinzu(Pflanze pflanze) {
        if (pflanze == null || anzahl == pflanzen.length) {
            return false;
        }
        for (int i = 0; i < anzahl; i++) {
            if (pflanzen[i] == pflanze) return false;
        }
        pflanzen[anzahl++] = pflanze;
        return true;
    }

    public boolean entferne(Pflanze pflanze) {
        if (anzahl <= 1) return false;
        for (int i = 0; i < anzahl; i++) {

```

```
        if (pflanzen[i] == pflanze) {
            for (int j = i; j < anzahl - 1; j++) {
                pflanzen[j] = pflanzen[j + 1];
            }
            pflanzen[--anzahl] = null;
            return true;
        }
    }
    return false;
}

public int getAnzahlPflanzen() {
    return anzahl;
}

public Pflanze getPflanze(int index) {
    if (index < 0 || index >= anzahl) return null;
    return pflanzen[index];
}

public boolean testeFlaeche() {
    long summe = 0;
    for (int i = 0; i < anzahl; i++) {
        summe += pflanzen[i].getFlaeche();
    }
    return summe <= (long) laenge * breite;
}
}
```

Das Array speichert bis zu 100 Referenzen. Nur die ersten `anzahl` Plätze sind belegt; beim Entfernen werden die nachfolgenden Referenzen verschoben. Der Identitätsvergleich verhindert, dass dieselbe Pflanze zweimal in derselben Assoziation steht.

`testeFlaeche` addiert die Flächen und vergleicht sie mit `Länge · Breite`. Gleichheit ist ausdrücklich erlaubt. Beispielsweise passen Pflanzenflächen 4 und 6 in einen Garten der Maße 2×5 , aber 4 und 7 nicht. `long` vermeidet bei der Summe und der Multiplikation einen `int`-Überlauf. Die Methode bewertet die Flächensumme; sie löst kein geometrisches Anordnungsproblem.