

Lösungen zur Klausur vom 27.08.2021

Quelle: 16_EidP_Klausur_27.08.2021.pdf. Einführung in die Programmierung. Alle fünf Aufgaben sind bearbeitet. Die auf dem Deckblatt genannten Bonuspunkte sind keine zusätzliche Aufgabenstellung.

Aufgabe 1 - Zweierkomplement von -47 mit 8 Bit

Zunächst stellt man den Betrag 47 binär dar: $47 = 32 + 8 + 4 + 2 + 1$, also 00101111. Man invertiert alle acht Bits und addiert anschließend 1:

+47:	00101111
Bits invertieren:	11010000
1 addieren:	11010001

Die gesuchte Darstellung lautet 11010001. Das führende Bit besitzt im 8-Bit-Zweierkomplement das Gewicht -128. Rückrechnung: $-128 + 64 + 16 + 1 = -47$. Gleichwertig ergibt sich der gespeicherte unsigned Wert aus $256 - 47 = 209$. Der darstellbare Bereich reicht von -128 bis +127.

Aufgabe 2 - Zeichenwiederholung

Die Methoden stehen in einer gemeinsamen Klasse. Nicht-null-Strings werden vorausgesetzt. Für negative Wiederholungszahlen bzw. ungültige Präfixlängen wird hier null zurückgegeben; diese Fälle legt das Aufgabenblatt nicht fest. Das leere Wort wird entsprechend der formalen Wortdefinition als zweimal bzw. mehrfach wiederholt angesehen: $\varepsilon = \varepsilon\varepsilon$.

```
public class ZeichenWiederholung {
    public String wiederhole(String w, int n) {
        if (n < 0) return null;
        String ergebnis = "";
        for (int i = 0; i < n; i++) {
            ergebnis = ergebnis + w;
        }
        return ergebnis;
    }

    public String berechnePräfix(String w, int i) {
        if (i < 0 || i > w.length()) return null;
        String ergebnis = "";
        for (int j = 0; j < i; j++) {
            ergebnis = ergebnis + w.charAt(j);
        }
        return ergebnis;
    }

    public boolean istWiederholt(String w) {
        if (w.length() % 2 != 0) return false;
        int haelfte = w.length() / 2;
        for (int i = 0; i < haelfte; i++) {
            if (w.charAt(i) != w.charAt(i + haelfte)) {
                return false;
            }
        }
        return true;
    }

    public boolean istMehrfachWiederholt(String w) {
        if (w.length() == 0) return true;
        for (int laenge = 1; laenge <= w.length() / 2; laenge++) {
            if (w.length() % laenge == 0) {
                String v = berechnePräfix(w, laenge);
                String kandidat = wiederhole(v, w.length() / laenge);
                if (kandidat.equals(w)) return true;
            }
        }
        return false;
    }
}
```

i) wiederhole

Nach i Schleifendurchläufen steht w genau i-mal im Ergebnis. Bei n = 0 ist das Ergebnis ""; "abc" mit n = 3 ergibt "abcabcabc".

ii) berechnePräfix

Die Indizes der ersten i Zeichen lauten 0 bis i - 1. Diese Zeichen werden unverändert übernommen. Das Präfix der Länge 2 von "abac" ist "ab"; das Präfix der Länge 0 ist leer.

iii) istWiederholt

Ein Wort `vv` hat gerade Länge. Seine beiden Hälften müssen zeichenweise gleich sein. "abab" erfüllt den Test, "abba" nicht. Der Vergleich benutzt die Zeichenwerte; ein Referenzvergleich zweier String-Objekte wäre hierfür ungeeignet.

iv) istMehrfachWiederholt

Die Länge eines nichtleeren Grundwortes `v` muss die Gesamtlänge teilen und höchstens deren Hälfte sein, damit mindestens zwei Wiederholungen entstehen. Die Methode prüft alle solchen Möglichkeiten. Für "ababab" wird die Präfixlänge 2 gefunden: "ab" wird dreimal wiederholt. Bei "aba" scheitert die einzige mögliche Länge 1. Falls eine Lehrveranstaltung ausdrücklich nur nichtleere Grundwörter zulässt, wären bei leerem `w` beide Wiederholungstests stattdessen `false`; für nichtleere Eingaben ändert sich nichts.

Aufgabe 3 - Häufigkeiten in einem zweidimensionalen Feld

Vorausgesetzt sind ein nicht-null-Feld mit nicht-null-Zeilen, nichtnegative Werte und insgesamt mindestens ein Element. Die Zeilen dürfen verschieden lang oder leer sein. Es werden keine Bibliotheksmethoden benötigt.

```
public class Haeufigkeiten {
    public int max(int[][] feld) {
        int maximum = 0;
        for (int i = 0; i < feld.length; i++) {
            for (int j = 0; j < feld[i].length; j++) {
                if (feld[i][j] > maximum) maximum = feld[i][j];
            }
        }
        return maximum;
    }

    public int[] berechneHaeufigkeiten(int[][] feld) {
        int[] anzahl = new int[max(feld) + 1];
        for (int i = 0; i < feld.length; i++) {
            for (int j = 0; j < feld[i].length; j++) {
                anzahl[feld[i][j]]++;
            }
        }
        return anzahl;
    }
}
```

i) Maximum

Da kein Wert negativ ist, kann maximum mit 0 beginnen. Nach jedem besuchten Eintrag enthält es das Maximum aller bislang gesehenen Werte. Damit ist es am Ende das globale Maximum M.

ii) Häufigkeitsfeld

Für die Werte 0 bis M werden genau $M + 1$ Plätze benötigt. Jedes Auftreten eines Wertes x erhöht $anzahl[x]$. Da M tatsächlich im Eingabefeld vorkommt, ist das letzte Element $anzahl[M]$ positiv. Beispiel: $\{\{2, 0\}, \{2, 1, 2\}\}$ ergibt $\{1, 1, 3\}$. Wie bei einer arraybasierten Zählung üblich wird vorausgesetzt, dass das benötigte Array in Java angelegt werden kann; ein Maximalwert `Integer.MAX_VALUE` wäre mit einem solchen Array grundsätzlich nicht darstellbar.

Aufgabe 4 - Regal und Medien

Die folgenden öffentlichen Typen gehören jeweils in eine eigene Datei. Für Breiten, Gewichte und Tragkräfte werden nichtnegative Werte vorausgesetzt.

1) Attribute für Regal und RegalBrett

Ein Regal benötigt Referenzen auf seine 2 bis 10 Bretter. Ein Array kann genau die vorhandenen Bretter speichern; seine Länge liefert dann die Anzahl. Die Klasse Regal wird unten gemeinsam mit der in Teil 3 geforderten Methode angegeben. Der ergänzte Konstruktor erhält ein Array mit 2 bis 10 verschiedenen, nicht-null-Brettern als Vorbedingung und kopiert dessen Referenzen.

2) Medium, Buch, CD und RegalBrett

```
public interface Medium {
    double getGewicht();
    int getBreite();
}

public class Buch implements Medium {
    private int breite;
    private double gewicht;

    public Buch(int breite, double gewicht) {
        this.breite = breite;
        this.gewicht = gewicht;
    }

    public int getBreite() { return breite; }
    public double getGewicht() { return gewicht; }
}

public class CD implements Medium {
    private int breite;

    public CD(int breite) {
        this.breite = breite;
    }

    public int getBreite() { return breite; }
    public double getGewicht() { return 1.5; }
}

public class RegalBrett {
    private int breite;
    private double tragkraft;
    private Medium[] medien = new Medium[100];
    private int anzahl;

    public RegalBrett(int breite, double tragkraft) {
        this.breite = breite;
        this.tragkraft = tragkraft;
    }

    public boolean fuegeHinzu(Medium medium) {
        if (medium == null || anzahl == medien.length) return false;
        long neueBreite = medium.getBreite();
        double neuesGewicht = medium.getGewicht();
        for (int i = 0; i < anzahl; i++) {
            if (medien[i] == medium) return false;
        }
    }
}
```

```
        neueBreite += medien[i].getBreite();
        neuesGewicht += medien[i].getGewicht();
    }
    if (neueBreite > breite || neuesGewicht > tragkraft) {
        return false;
    }
    medien[anzahl++] = medium;
    return true;
}

public boolean entferne(Medium medium) {
    for (int i = 0; i < anzahl; i++) {
        if (medien[i] == medium) {
            for (int j = i; j < anzahl - 1; j++) {
                medien[j] = medien[j + 1];
            }
            medien[--anzahl] = null;
            return true;
        }
    }
    return false;
}

public int getAnzahlMedien() { return anzahl; }

public Medium getMedium(int i) {
    if (i < 0 || i >= anzahl) return null;
    return medien[i];
}
}
```

Ein Brett darf leer sein, denn die Multiplizität lautet 0..100. Beim Hinzufügen werden sowohl Breite als auch Gewicht des neuen Mediums in die Prüfung einbezogen. Erst nach beiden erfolgreichen Prüfungen wird die Assoziation verändert. Exaktes Erreichen der Grenze ist zulässig. Eine CD liefert unabhängig von ihrer Breite stets 1,5 als Gewicht.

3) Medium in einem passenden Brett unterbringen

```
public class Regal {
    private RegalBrett[] bretter;

    // Vorbedingung: 2 bis 10 verschiedene, vorhandene Bretter.
    public Regal(RegalBrett[] vorhandeneBretter) {
        bretter = new RegalBrett[vorhandeneBretter.length];
        for (int i = 0; i < bretter.length; i++) {
            bretter[i] = vorhandeneBretter[i];
        }
    }

    public boolean fügeHinzu(Medium medium) {
        if (medium == null) return false;
        for (int i = 0; i < bretter.length; i++) {
            for (int j = 0; j < bretter[i].getAnzahlMedien(); j++) {
                if (bretter[i].getMedium(j) == medium) return false;
            }
        }
        for (int i = 0; i < bretter.length; i++) {
            if (bretter[i].fuegeHinzu(medium)) return true;
        }
        return false;
    }
}
```

Die Methode versucht die Bretter der Reihe nach. Ein erfolgloser Versuch verändert das Brett nicht. Beim ersten erfolgreichen Einfügen endet die Suche; dadurch wird das Medium nur einmal eingefügt. Scheitern alle Bretter, lautet das Ergebnis false. Die vorgeschaltete Identitätsprüfung verhindert auch eine zweite Aufnahme desselben Mediums auf einem anderen Brett dieses Regals.

Aufgabe 5 - Speicherabbild

Die Bezeichner O1 und O2 stehen für zwei verschiedene B-Objekte. S steht für das String-Literal mit der eigenen Matrikelnummer; beispielsweise wäre sein Inhalt bei Matrikelnummer 0000000 der String "0000000". Beide Objekte verwenden dasselbe Literal und referenzieren daher dasselbe String-Objekt.

Konstruktion und Feldbindung

x beginnt bei 1. new B(1) ruft A(3) auf: O1.A.m wird 3 und x wird 2. Danach setzt B den eigenen, davon getrennten Feldwert O1.B.m auf 2 und erhöht x auf 3.

b1 hat den deklarierten Typ A. Daher liest b1.m das Feld A.m, also 3. new B(b1, b1.m) ruft somit B(O1, 3) auf. Dessen super(a.m + m) ergibt A(3 + 3): O2.A.m wird 6 und x wird 4. Anschließend wird O2.B.m = a.m = 3 gesetzt und x wird 5.

Statischer Speicher

Eintrag	Wert
A.x	5

x ist genau ein gemeinsames statisches Feld; die Unterklasse besitzt keine zweite unabhängige Kopie.

Heap

Objekt	Dynamischer Typ	Felder
O1	B	A.m = 3; B.m = 2; matrNr → S
O2	B	A.m = 6; B.m = 3; matrNr → S
S	String	Inhalt: eigene Matrikelnummer
ARGS	String[]	Referenz aus main; Inhalt hängt vom Programmaufruf ab

Aufrufstapel am Zeitpunkt 1

Stackframe	Variable	Wert
main	args	→ ARGS
main	b1	→ O1
main	b2	→ O2
main	erg	noch nicht zugewiesen
B.gleich(A)	this	→ O1
B.gleich(A)	a	→ O2
B.gleich(A)	erg	false

b1.gleich(b2) wird dynamisch an B.gleich gebunden, weil das Objekt O1 vom Typ B ist. Die beiden getM()-Aufrufe benutzen ebenfalls B.getM, also die B-Felder 2 und 3. Somit wird `2 == 3` ausgewertet und erg ist false.

Die Getter sind am markierten Zeitpunkt bereits zurückgekehrt und haben keine aktiven Stackframes mehr. Die lokale Variable erg in main ist noch nicht initialisiert, da die rechte Seite ihrer Zuweisung noch läuft. Erst nach return erg erhält sie false. Lokale Variablen werden im Gegensatz zu Objektfeldern nicht automatisch mit false oder 0 initialisiert.