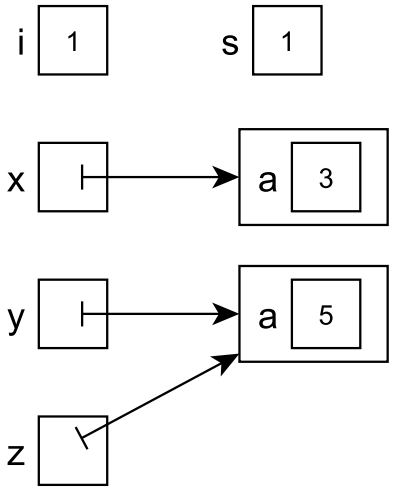
Sigrid Weil

# Klausur SS 2024
# - Musterlösung -

**Aufgabe 1 (15 P)**

**Lösung 1**

a) Speicherzustand zum „Zeitpunkt 1“:




b) Tracetabelle mit Ausgabe:

| Zeile | i | s | x.a | y.a | z.a | (n) | Ausgabe |
|---|---|---|---|---|---|---|---|
| 4 | 1 | | | | | | |
| 5 | | 1 | 3 | | | | |
| 6 | | | | 5 | | | |
| 7 | | | | | 5 | | |
| 10 | | | | | | | i: 1 |
| 11 | 1 | | | | | | (Aufruf methode1(1) ) |
| 25 | | | | | | 1 | (Aufruf methode1(3) ) |
| 26 | | | | | | 2 | |
| 12 | 1 | | | | | | |
| 25 | | | | | | 3 | |
| 26 | | | | | | 6 | |
| 13 | 1 | | | | | | i: 1 |
| 15 | | | | | | | x: s = 1 a = 3 |
| 16 | | | | | | | y: s = 1 a = 5 |
| 17 | | | | | | | z: s = 1 a = 5 |
| 18, 30 | | | | | | | (Aufruf methode2(x) ) |
| (KlasseA) 12 | | | 6 | | | | |
| (KlasseA) 13 | | 2 | | | | | |
| 19, 30 | | | | | | | (Aufruf methode2(z) ) |
| (KlasseA) 12 | | | | 10 | 10 | | |
| (KlasseA) 13 | | 4 | | | | | |
| 20 | | | | | | | x: s = 4 a = 6 |
| 21 | | | | | | | y: s = 4 a = 10 |
| 22 | | | | | | | z : s = 4 a = 10 |

**Aufgabe 2 (15 P)**

Ein Array von Werten können wir auch als „endliche Folge“ der Werte bezeichnen.
Wir sagen, dass eine Folge $b$ eine *Teilfolge* von einer Folge $a$ ist, wenn man $b$ aus $a$ erhält, indem man einige (egal wieviele) Elemente aus $a$ weglässt (ohne die Reihenfolge der übrigen Elemente zu ändern).

a) Betrachten Sie als weiteres Beispiel die Arrays
$a = [42, 5, -7, 0, 8, 15, 47, 11, 43, 5, -6, 0, 8, 15, 47, 12]$ und
$b = [-7, 0, 15, 8, 11]$ bzw. $b' = [-7, 0, 15, 8, 12]$
Sind $b$ bzw $b'$ Teilfolgen von $a$?

b) Schreiben Sie eine (statische) Java-Methode, die zwei int-Arrays $a$ und $b$ als Parameter erhält und prüft, ob $b$ eine Teilfolge von $a$ darstellt.

Tipp:
Beobachten Sie und formulieren Sie in Pseudocode oder umgangssprachlich, wie Sie vorgegangen sind, um Frage a) zu beantworten.
Sie dürfen sich Hilfsmethoden definieren, wenn Sie das als hilfreich empfinden.

**Lösung 2**

a) $b$ ist keine Teilfolge, $b'$ ist eine.
Man betrachtet sukzessive jedes Element aus $b$ und sucht dieses im Array $a$.
Wird es gefunden, merkt man sich die Position, an der es in $a$ gefunden wurde und sucht das nächste Element aus $b$ nur im Array $a$ **ab dieser Position**. Entdeckt man ein Element aus $b$, das auf diese Weise nicht in dem durchsuchten Teil von $a$ gefunden wird, dann ist $b$ keine Teilfolge von $a$.

b) Die Hilfsmethode gefunden(**int**[] a, **int** b, **int** k) sucht das Element $b$ im Array $a$ **ab Position k** und liefert die Position, an der es gefunden wurde, als Ergebnis. Der return-Wert a.length bedeutet, dass der Wert *nicht* im durchsuchten Teil von $a$ enthalten war.

```
public static boolean teilfolge(int[] a, int[] b) {
        int k = -1;
        for(int j = 0; j < b.length; j++) {
                // suche b[j] in a ab Index k
                // setze k auf die gefundene Position
                k = gefunden(a, b[j], k);
        }
        return (k < a.length);
}

private static int gefunden(int[] a, int b, int k) {
        for (int i = k+1; i < a.length; i++) {
                if (b == a[i]) {
                        return i;
                }
        }
        return a.length;
}
```

**Aufgabe 3 (15 P)**
Schreiben Sie eine (statische) Methode, die eine quadratische Matrix (dh. ein 2-dimensionales quadratisches `int`-Array) der Größe $n \times n$ erzeugt und mit Werten belegt, wie es unten in den Beispielen für $n = 3$ und $n = 4$ gezeigt ist.

Die Anzahl $n$ der Zeilen und Spalten wird der Methode als Parameter übergeben, das erzeugte Array wird als Ergebnis geliefert.

Beispiele:

n = 3

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 1 | 2 |
| 2 | 2 | 2 |

n = 4

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

Gehen Sie wie folgt vor:

a) Überlegen Sie, wie allgemein die Zeile mit (Index-)Nummer $i$ der Matrix aussieht, indem Sie sich Beispiele konstruieren:
Geben Sie für den Fall n = 10 die Zeilen Nummer 4 und Nummer 8 an!
(Wir beginnen wie üblich die Nummerierung bei 0.)

b) Implementieren Sie nun!

Sie dürfen davon ausgehen, dass für die Kantenlänge $n$ nur positive Werte übergeben werden.

Bemerkung: Es ist *keine* Bildschirmausgabe nötig!

**Lösung 3**
a) In der Zeile i wird bis zur Spalte j = i der Wert i eingetragen;
in den folgenden Spalten wird der Wert j eingetragen.
Für n = 10 und i = 4 sieht das so aus: 4 4 4 4 4 5 6 7 8 9
Für n = 10 und i = 8 sieht das so aus: 8 8 8 8 8 8 8 8 8 9

b)
```
public static int[][] sose24(int n) {
        int[][] a = new int[n][n];
        for(int i = 0; i < n; i++) {
                for (int j = 0; j <= i; j++)
                        a[i][j] = i;
                for(int j = i+1; j < n; j++)
                        a[i][j] = j;
        }
        return a;
}
```

**Aufgabe 4 (25 P)**
Lesen Sie die Aufgabenstellung **bis zum Ende** durch, bevor Sie mit der Bearbeitung beginnen:

In einer Schule soll eine Projektwoche geplant werden. Das Organisationskomitee erfasst für jeden Projektvorschlag

- den Titel des Projektes
- eine Person als Projektleiter:in
  (Achtung! Projektleiter sind *Personen*, hier genügt nicht nur der *Name* einer Person!
  Eine entsprechende Klasse finden Sie im Anhang.)
- die maximale Anzahl der Teilnehmer:innen

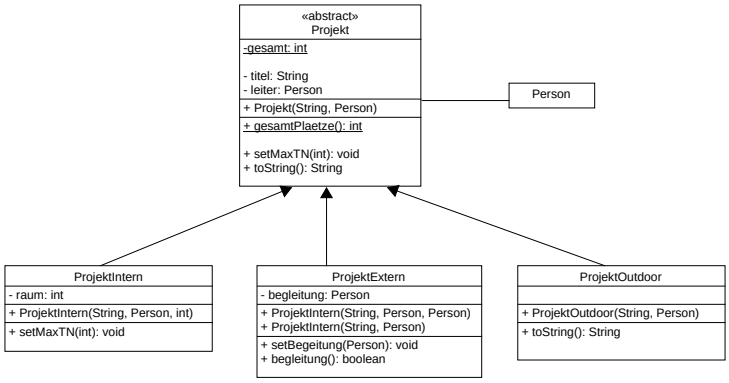
a) Entwerfen Sie eine Klassenstruktur, die den in b) gestellten Anforderungen entspricht und stellen Sie sie in UML-ähnlicher Notation dar. (Details der einzelnen Klassen brauchen nicht dargestellt werden.)
   Nutzen Sie die Konzepte von Abstraktion und Vererbung!

b) Implementieren Sie Java-Klassen, sodass folgende Anforderungen erfüllt sind:
   - Projekte sind zwingend von einem der (weiter unten beschriebenen) Typen „intern“, „extern“ oder „outdoor“.
   - Titel und Projektleiter:in müssen bei Erzeugung eines Projektes angegeben werden.
     Zu einem späteren Zeitpunkt kann die maximale TN-Zahl eines Projektes festgelegt werden. Sie kann auch nachträglich geändert werden.
     Die `toString()`-Methode liefert den Projekt-Titel und den Namen der Projektleitung.
   - Um zu garantieren, dass *alle* Schüler:innen einen Platz in einem Projekt finden, müssen genügend Projektplätze angeboten werden.
     Stellen Sie eine Methode bereit, die die Gesamtzahl an angebotenen Projektplätzen ermittelt.
     (Achtung! Wenn für ein Projekt die maximale Teilnehmerzahl *geändert* wird, muss sich auch die Anzahl der insgesamt verfügbaren Plätze entsprechend ändern.)
   - Projekte sind „intern“, „extern“ oder „outdoor“-Projekte.
     - Für interne Projekte muss bei Anlegen des Projektes eine Raumnummer angegeben werden. Raumnummern sind int-Werte.
       (Sie brauchen hier nicht zu kontrollieren, ob es zu Raumkollisionen kommt. Bei der Planung darf sich jedes Projekt einen Raum „wünschen“.)
     - Externe Projekte brauchen zwar keine Raumnummer, aber für solche Projekte muss eine zweite Person als Begleitung benannt werden.
       Diese Begleitperson kann bereits bei Anlegen des Projektes genannt werden, sie kann aber auch nachträglich benannt oder geändert werden.
       Für externe Projekte soll eine Methode die Information liefern, **ob** bereits eine Begleitperson benannt wurde.
     - „outdoor“-Projekte sind z.B. solche, die auf dem Schulhof stattfinden können.
       Sie benötigen weder einen Raum noch eine zusätzliche Begleitperson.
       Die `toString()`-Methode eines solchen Projektes nennt zusätzlich zu Titel und Projektleiter:in auch den Vermerk „outdoor“.

Sie brauchen für keines der Attribute eine get-Methode zu implementieren, aber ggf (beachten Sie die Aufgabenstellung) set-Methoden.

**Lösung 4**

a) Klassenstruktur in Pseudo-UML:




b)

```
public abstract class Projekt {

        private static int gesamt = 0;

        private String titel;
        private Person leiter;
        private int maxTN;

        public Projekt(String t, Person p) {
                this.titel = t;
                this.leiter = p;
        }

        public static int gesamtPlaetze() {
                return gesamt;
        }

        public void setMaxTN(int max) {
                gesamt-= maxTN;
                maxTN = max;
                gesamt += max;
        }

        public String toString() {
                return titel + " " + leiter.name();
        }
}

public class ProjektIntern extends Projekt {

        private int raum;

        public ProjektIntern(String t, Person p, int r) {
                super(t, p);
                this.raum = r;
        }
}

public class ProjektExtern extends Projekt {

        private Person begleitung;

        public ProjektExtern(String t, Person p, Person b) {
```

```
                super(t, p);
                this.begleitung = b;
        }

        public ProjektExtern(String t, Person p) {
                super(t, p);
                this.begleitung = null;
        }

        public void setBegleitung(Person b) {
                this.begleitung = b;
        }

        public boolean begleitung() {
                return (this.begleitung != null);
        }
}

public class ProjektOutdoor extends Projekt {

        public ProjektOutdoor(String t, Person p) {
                super(t, p);
        }

        public String toString() {
                return super.toString() + " (outdoor)" ;
        }
}
```

**Aufgabe 5 (10 P)**

a) Implementieren Sie in einer MainKlasse die (statische) Methode
checkDatum(**int** tag, **int** monat).
Die Methode soll zwei int-Werte als Parameter annehmen, die ein Tagesdatum darstellen sollen.

Die Methode löst eine DatumFormatException aus, wenn tag nicht im Bereich $1, \ldots 31$ liegt oder wenn monat nicht zwischen 1 und 12 liegt (einschließlich).

Die unterschiedliche Länge von Monaten brauchen Sie **nicht** zu berücksichtigen!
Die Methode akzeptiert also Angaben wie $tag = 31, monat = 2$
(Dies bezeichnen wir im folgenden als „halbwegs sinnvolle“ Angabe.)

Im Anhang finden Sie eine Implementierung der Klasse DatumFormatException.

b) Ergänzen Sie die Klasse Person (siehe Anhang) um Instanz-Attribute und eine Instanz - Methode setGeburtstag(**int** tag, **int** monat),
die die entsprechenden Attribute auf die angegebenen Werte setzt.
Es soll aber zunächst geprüft werden, ob die angegebenen Werte ein (halbwegs) sinnvolles Datum darstellen. Falls nicht, soll die Methode setGeburtstag() eine aufgetretene Fehlermeldung an ihre aufrufende Methode weiterleiten.

c) Implementieren Sie in der Klasse Person einen weiteren Konstruktor, der neben dem Namen auch ein volles Geburtsdatum (Tag, Monat, Jahr) annimmt.
Der Konstruktor soll die Methode setGeburtstag() aufrufen.
Falls die Angaben kein (halbwegs sinnvolles) Datum darstellen, sollen für das Objekt die Defaultwerte tag = 1, monat = 1 eingestellt werden.

**Lösung 5**

```
// in TestMain ////////////////////////////////////////////

public static void checkDatum(int tag, int monat) throws
    DatumFormatException {
    if(tag < 1)
       throw new DatumFormatException("Tag muss mindestens 1 sein");
    if( tag > 31 )
       throw new DatumFormatException("Tag kann nicht > 31 sein");
    if(monat < 1)
       throw new DatumFormatException("Monat muss mindestens 1 sein");
    if(monat > 12)
       throw new DatumFormatException("Monat kann nicht > 12 sein");
}


// in Person ////////////////////////////////////

    private String name;
    private int gebJahr;
    private int gebMonat;
    private int gebTag;

    // Konstruktoren
    public Person(String name, int gebJahr) {
       this.name = name;
       this.gebJahr = gebJahr;
    }

    public Person(String name, int t, int m, int j) {
```

```
        this(name, j);
        try {
            setGeburtstag(t, m);
        }
        catch (DatumFormatException e) {
            System.out.println(e.getMessage());
            this.gebMonat = 1;
            this.gebTag = 1;
        }
    }

    // Instanzmethoden

    public void setGeburtstag(int t, int m) throws DatumFormatException {
        TestException.checkDatum(t, m);
        this.gebMonat = m;
        this.gebTag = t;
    }

    // ...
```

**Aufgabe 6 (10 P)**
Viele Mobiltelefone bieten eine Kalenderfunktion an, mit der Termine in einen Kalender eingetragen und auch wieder gelöscht werden können.

Ein Termin enthält einen String-Eintrag (was?) und ein Datum (wann?).
Außerdem können Termine „Wiederhol-Termine“ (wTermin == **true**) sein, die ab dem angegebenen Datum des Termins täglich stattfinden.

Eine Klasse zur Darstellung von Terminen finden Sie im Anhang.
(Nehmen Sie an, dass eine Klasse Datum zur Darstellung von Datumsangaben existiert.)

Eine (nicht sehr komfortable) App auf Ihrem Handy implementiert das Interface Kalender. Die Definition des Interfaces finden Sie ebenfalls im Anhang.

Nehmen Sie an, die App sei in einer Klasse HandyKalender implementiert, deren Quellcode Sie aber nicht kennen.

Sie möchten die App für sich um eine Funktion erweitern, die es erlaubt, Termine von einem Start-Datum (einschließlich) bis zu einem End-Datum (ausschließlich) einzutragen.

**Implementieren Sie eine Unterklasse** MeinBessererKalender von HandyKalender mit einer Methode **void** eintragen(String was, Datum start, Datum ende).

Nutzen Sie die Tatsache aus, dass HandyKalender das Interface Kalender implementiert!
**Die Klasse Handykalender brauchen und sollen Sie nicht implementieren!**

**Lösung 6**

```
public class MeinBessererKalender extends HandyKalender {

    public MeinBessererKalender() {
        super();
    }

    public void eintragen(String was, Datum ab, Datum bis) {
        Termin t = new Termin(was, ab, true);
        super.eintragen(t);
        super.loeschenAlleAb(t, bis);
    }
}
```

**Aufgabe 7 (10 P)**
Es sei folgende *rekursiv definierte* Methode gegeben:

```
public static int rek(int n) {
   if(n <= 1)
      return 1;
   return rek(n-1) + n + 2;
}
```

a) Berechnen Sie („von Hand“) den Wert rek(3). Geben Sie nicht nur das End-Ergebnis, sondern auch die Aufruf-Reihenfolge an.

b) Implementieren Sie eine *iterative* Variante der Methode.

**Lösung 7**
a) Es ist

$$\begin{aligned} rek(3) &= rek(2) + 3 + 2 = rek(2) + 5 \\ &= (rek(1) + 2 + 2) + 5 = rek(1) + 9 \\ &= 1 + 9 = 10 \end{aligned}$$

b) Iterativ:

```
public static int iter(int n) {
   int x = 1;
   for (int i = 2; i <= n; i++)
      x = x+i+2;
   return x;
}
```