
Bevor Sie mit der Bearbeitung der Klausur beginnen, lesen Sie bitte folgende Hinweise:

- Prüfen Sie die **Vollständigkeit** Ihres Exemplars. Jede Klausur umfasst
  - das Deckblatt
  - diese Hinweisseite
  - 7 Aufgaben mit Lösungsblättern
  - einen Anhang mit Quellcode, den Sie als gegeben annehmen und verwenden dürfen

  Bei Unvollständigkeit Ihres Exemplars wenden Sie sich bitte an die Aufsicht.
- Tragen Sie auf **jedem Lösungsblatt** oben Ihre **Matrikelnummer** ein und **unterschreiben** Sie auf dem Deckblatt!
- Hinter den Aufgaben ist jeweils hinreichend Platz für Ihre Lösungen. Bei Bedarf nutzen Sie auch die **Rückseiten** der Aufgabenblätter und der Deck- oder Hinweisblätter. Sollten Sie weiteres Papier benötigen, so wenden Sie sich bitte an die Aufsicht.
  Verwenden Sie kein eigenes Papier!
- Stellen Sie sicher, dass die Zuordnung von Lösung zu Aufgabe auf jeden Fall deutlich erkennbar ist.
- Verwenden Sie keine Klassen oder Methoden der Java-API wie **Arrays, ArrayList** o.ä. und auch nicht die erweiterte for-Schleife („for each“).

Es werden maximal 100 Punkte vergeben.
Sie haben bestanden, wenn Sie mindestens 40 Punkte erreicht haben.

Ergebnis (bitte nichts eintragen):

Aufgabe	1	2	3	4	5	6	7	$\sum$
Punkte	15	15	15	25	10	10	10	100
erreicht								

Aufgabe 1 (15 P)
Betrachten Sie die Klassendefinition von KlasseA und den Testablauf in der Methode main, die Sie im Anhang finden:
(Sie dürfen die Seiten des Anhangs abreissen, um lästiges Umblättern zu vermeiden.)

a) Zeichnen ein Speicherbild, das den Zustand nach Ausführung der Zeilen 3-7 der main-Methode darstellt. (Die Stelle ist im Code mit dem Kommentar „Zeitpunkt 1“ markiert.)

b) Erstellen Sie eine Trace-Tabelle für alle vorkommenden Variablen (auch Attribute) von *simple types*, also für alle Variablen vom Typ **int**. Geben Sie auch die Zeilennumern und in einer speziellen Spalte die Ausgabe an, die jeweils erzeugt wird.
(Zeilen, die keine Änderung eines Variablenwertes bewirken und keine Ausgabe erzeugen, brauchen Sie nicht aufzulisten.)

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).

Beispiele:

- Für $a = [42, 5, -7, 0, 8, 15, 47, 11]$
  ist $b = [5, -7, 15, 11]$ eine Teilfolge von a.
- $[42, -7, 5]$ ist *keine* Teilfolge von a, weil sich die Reihenfolge der Elemente geändert hat.
- Jede Folge ist Teilfolge von sich selber und die leere Folge $[\ ]$ ist Teilfolge von jeder anderen Folge.

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.

(Platz für Ihre Lösung)

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!

(Platz für Ihre Lösung)

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.

(Platz für Ihre Lösung)

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.

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!

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.

Anhang

Code zu Aufgabe 1:

```java
public class KlasseA {

    private static int s = 1;

    private int a;

    public KlasseA(int a) {
        this.a = a;
    }

    public void dopple() {
        a = 2*a;
        s = 2*s;
    }

    public String toString() {
        return "s = " + s + " a = " + a;
    }
}
```

```java
public class MainKlasseA {

    public static void main(String[] args) {
        int i = 1;
        KlasseA x = new KlasseA(3);
        KlasseA y = new KlasseA(5);
        KlasseA z = y;
        // Zeitpunkt 1

        System.out.println("i: " + i);
        methode1(i);
        methode1(3*i);
        System.out.println("i: " + i);

        System.out.println("x: " + x);
        System.out.println("y: " + y);
        System.out.println("z: " + z);
        methode2(x);
        methode2(z);
        System.out.println("x: " + x);
        System.out.println("y: " + y);
        System.out.println("z: " + z);
    }

    public static void methode1(int n) {
        n = 2*n;
    }

    public static void methode2(KlasseA p) {
        p.dopple();
    }
}
```

Code zu den Aufgaben 4 und 5:

```java
public class Person {
    private String name;
    private int gebJahr;

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

    public String name() {
        return name;
    }

    public int gebJahr() {
        return gebJahr;
    }
}
```

```java
public class DatumFormatException extends Exception {

    public DatumFormatException() {
        super();
    }

    public DatumFormatException(String message) {
        super(message);
    }
}
```

Code zu Aufgabe 6:

```java
public class Termin {

    private String was;
    private Datum wann;
    private boolean wTermin;
    // true: taegliche Wiederholung
    // false: einmaliger Termin

    public Termin(String was, Datum wann, boolean wTermin) {
        this.was = was;
        this.wann = wann;
        this.wTermin = wTermin;
    }

    public Termin(String was, Datum wann) {
        this(was, wann, false);
    }

    public String was() {
        return was;
    }

    public Datum wann() {
        return wann;
    }

    public boolean wTermin() {
        return wTermin;
    }
}
```

```java
public interface Kalender {

    void eintragen(Termin t);
    // traegt den uebergebenen Termin in den Kalender ein
    // wenn es sich um einen wTermin handelt,
    // dann wird der Termin auch in allen folgenden Tagen eingetragen

    void loeschen(Termin t);
    // loescht den uebergebenen Termin aus dem Kalender
    // bei einem wTermin wird nur der erste geloescht

    void loeschenAlleAb(Termin t, Datum d);
    // loescht alle Wiederholungen eines wTermins ab dem uebergebenen Datum
    // (einschliesslich)
}
```