Prof. Dr. J. Giesl
J.-C. Kassing, M. Leven Rosarius, N. Lommen, E. Meyer

Allgemeine Hinweise:

- Die **Deadline** zur **Abgabe** der Hausaufgaben ist am **Donnerstag, den 13.11.2025, um 14 Uhr**.
- Der **Workflow** sieht wie folgt aus. Die Abgabe der Hausaufgaben erfolgt **im Moodle-Lernraum** und kann nur in **Zweiergruppen** stattfinden. Dabei müssen die Abgabepartner*innen **dasselbe Tutorium** besuchen. Nutzen Sie ggf. das entsprechende **Forum** im Moodle-Lernraum, um eine*n Abgabepartner*in zu finden. Es darf **nur ein*e** Abgabepartner*in die Abgabe hochladen. Diese*r muss sowohl die **Lösung** als auch den **Quellcode** der Programmieraufgaben hochladen. Die Bepunktung wird dann von uns für **beide** Abgabepartner*innen **separat** im Lernraum eingetragen. Die Feedbackdatei ist jedoch nur dort sichtbar, wo die Abgabe hochgeladen wurde und muss innerhalb des Abgabepaars **weitergeleitet** werden.
- Die **Lösung** muss als PDF-Datei hochgeladen werden. Damit die Punkte beiden Abgabepartner*innen zugeordnet werden können, müssen **oben** auf der **ersten Seite** Ihrer Lösung die **Namen**, die **Matrikelnummern** sowie die **Nummer des Tutoriums** von **beiden** Abgabepartner*innen angegeben sein.
- Der **Quellcode** der Programmieraufgaben muss als `.zip`-Datei hochgeladen werden und **zusätzlich** in der PDF-Datei mit Ihrer Lösung enthalten sein, sodass unsere Hiwis ihn mit Feedback versehen können. Auf diesem Blatt muss Ihre Codeabgabe Ihren vollständigen Java-Code in Form von `.java`-Dateien enthalten. Aus dem Lernraum heruntergeladene Klassen dürfen nicht mit abgegeben werden.

  Stellen Sie sicher, dass Ihr Programm von `javac` **in der Version 25 akzeptiert** wird. Generell sollten alle Programme für alle Eingaben terminieren, solange in der Spezifikation (bzw. der Aufgabenstellung) nicht explizit etwas anderes verlangt wird!
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden. Klicken Sie dazu im Lernraum rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Hausaufgabe 3 (Verifikation): (15 Punkte)

Gegeben sei folgendes Java-Programm P über der `int[]`-Variable `a` sowie den `int`-Variablen `i` und `s` und der `boolean`-Variable `res`:

$\langle$ `a.length` > 0 $\rangle$ (Vorbedingung)

```
i = 1;
res = false;
s = a[0];
while (i < a.length) {
    if (s == a[i]) {
        res = true;
    }
    s = s + a[i];
    i = i + 1;
}
```

$\langle$ `res` $= \exists j$ mit $1 \leq j <$ `a.length`. `a`$[j] = \sum_{k=0}^{j-1}$ `a`$[k]$ $\rangle$ (Nachbedingung)

a) Vervollständigen Sie die folgende Verifikation der partiellen Korrektheit des Algorithmus im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Geben Sie bei der Anwendung der Bedingungsregel (zur Behandlung der `if`-Anweisung) an, welche Formel der Art "$\varphi \wedge \neg B \Rightarrow \psi$" hierbei gezeigt werden muss.

Hinweise:

- Die Nachbedingung besagt, dass `res` genau dann `true` ist, wenn es einen Eintrag `a`$[j]$ mit $j \geq 1$ im Array gibt, sodass `a`$[j] =$ `a`$[0] +$ `a`$[1] + \cdots +$ `a`$[j-1]$ gilt.
- Um Platz zu sparen, können Sie statt `a.length` nur `n` schreiben.
- Gehen Sie wieder bei allen Aufgaben zum Hoare-Kalkül davon aus, dass keine Integer-Überläufe stattfinden, d.h., behandeln Sie Integers als die unendliche Menge $\mathbb{Z}$.
- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x + 1 = y + 1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.
- Es empfiehlt sich oft, bei der Erstellung der Zusicherungen in der Schleife von unten (d. h. von der Nachbedingung aus) vorzugehen.
- Die Implikationen bei Konsequenzregeln müssen nicht separat bewiesen werden.

$\langle \mathtt{n} > 0 \rangle$

$\langle \_\_\_\_\_ \rangle$

```
i = 1;
```

$\langle \_\_\_\_\_ \rangle$

```
res = false;
```

$\langle \_\_\_\_\_ \rangle$

```
s = a[0];
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
while (i < a.length) {
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    if (s == a[i]) {
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
        res = true;
```

$\langle \_\_\_\_\_ \rangle$

```
    }
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    s = s + a[i];
```

$\langle \_\_\_\_\_ \rangle$

```
    i = i + 1;
```

$\langle \_\_\_\_\_ \rangle$

```
}
```

$\langle \_\_\_\_\_ \rangle$

$\langle \mathtt{res} = \exists j \text{ mit } 1 \leq j < \mathtt{n}.\ \mathtt{a}[j] = \sum_{k=0}^{j-1} \mathtt{a}[k] \rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und mit Hilfe des Hoare-Kalküls die Terminierung unter der Voraussetzung `a.length` > 0 bewiesen werden. Geben Sie auch hier die Formel an, die bei der Anwendung der Bedingungsregel (zur Behandlung der `if`-Anweisung) gezeigt werden muss.

In den Aufgaben 4 und 5 sollen Sie Speicherzustände zeichnen. Angenommen wir haben folgenden Java Code:

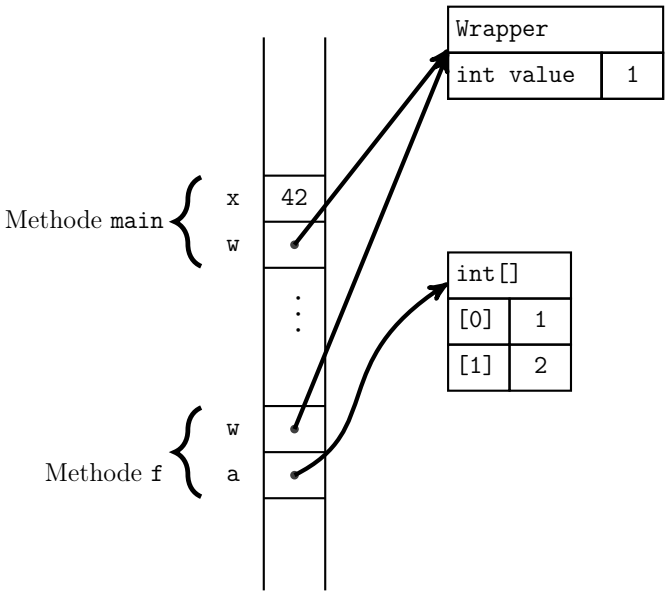
```
public class Wrapper {
      int value;
}
```

```
public class Main {
      public static void main() {
            int x = 42;
            Wrapper w = new Wrapper();
            w.value = 0;
            f(w);
      }

      public static void f(Wrapper w) {
            int[] a = {1,2};
            w.value = 1;

            // Speicherzustand hier gezeichnet
      }
}
```

Dann sieht der Speicher an der markierten Stelle wie folgt aus:



Hausaufgabe 5 (Seiteneffekte): (18 Punkte)

Betrachten Sie das folgende Programm:

```
public class HSeiteneffekte {
        public static void main() {
                HWrapper w1 = new HWrapper();
                HWrapper w2 = new HWrapper();

                w1.i = 0;
                w2.i = 3;

                int[] a = { 1, 2 };

                f(a, w1);
                int[] b = {a[0] + a[1],a[0] * a[1]};
                f(b, w1, w2);
                f(a);
        }

        public static void f(int[] a, HWrapper... ws) {
                if(ws.length == 0) {
                        a = new int[2];
                        a[0] = 3;
                        a[1] = 2 * a[0];
                } else {
                        a[1] += a[0];
                        ws[ws.length-1].i = a[0];
                        ws[0].i += ws[ws.length-1].i;
                }
                //Speicherzustand jeweils hier zeichnen
        }
}

public class HWrapper {

    int i;

}
```

Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher (d.h. alle (implizit) im Programm vorkommenden Arrays und alle Objekte sowie die zu dem Zeitpunkt existierenden Programmvariablen) am Ende jeder Ausführung der Methode `f` graphisch dar. Insgesamt sind also drei Speicherzustände zu zeichnen.

Hausaufgabe 7 (Programmierung): (17 Punkte)

Heutzutage werden sehr viele Daten gesammelt. Um einen Überblick über die Daten zu erhalten, ist es sinnvoll, sich diese automatisch zusammenfassen zu lassen.

a) Implementieren Sie die Klasse `Statistics` im Code-Fragment unten. Ein Objekt der Klasse `Statistics` kann bis zu 100 `int`-Werte zusammenfassen. Die Klasse `Statistics` soll folgende Methoden bereitstellen:

- `public void addValues(int... values) { ... }`
  Fügt eine Folge von Werten zu den bisherigen Werten im aktuellen `Statistics`-Objekt hinzu.
- `public double getAverage() { ... }`
  Gibt den durchschnittlichen Wert aller bisher hinzugefügten Werte zurück.
- `public Statistics generate(int min, int max, int size) { ... }`
  Generiert zufällig `size` viele Werte im Intervall von (jeweils inklusive) `min` bis `max` und fügt die Werte zu einem neuen `Statistics`-Objekt hinzu. Dieses neue `Statistics`-Objekt wird als Ergebnis zurückgegeben.

Implementieren Sie diese drei Methoden. Überlegen und begründen Sie, welche dieser Methoden statisch sein sollen und fügen Sie ggf. das Schlüsselwort `static` hinzu. Sie können beliebige Attribute und Hilfsmethoden einfügen. Beachten Sie außerdem folgende Randfälle:

- Wird `getAverage` aufgerufen, ohne dass zuvor Werte hinzugefügt wurden, so wird eine Fehlermeldung ausgegeben und der Wert `0` zurückgegeben.
- Wird `addValues` mit mehr Werten aufgerufen, als es noch Platz im aktuellen `Statistics`-Objekt gibt, so werden solange Werte hinzugefügt, bis das Objekt 100 Werte "enthält". Anschließend wird eine (oder mehrere) Fehlermeldung(en) ausgegeben und die weiteren Werte aus `values` werden nicht mehr hinzugefügt.
- Wird `generate` aufgerufen, sodass `size` negativ oder größer als 100 ist, oder `max` kleiner als `min` ist, dann sollen jeweils entsprechende Fehlermeldungen ausgegeben werden (und aber dennoch ein neues `Statistics`-Objekt zurückgegeben werden).

Hinweise:

- Verwenden Sie die angegebene `main`-Methode zum Testen Ihrer Implementierung.
- Bei der Berechnung des Durchschnitts dürfen Sie Variablenüberläufe ignorieren.
- Um Zufallszahlen aus $\{0, \ldots, \texttt{n}\}$ zu generieren, erzeugen Sie zuerst ein Objekt `rand` mit `Random rand = new Random();`. Nun liefert `rand.nextInt(n + 1)` eine Zufallszahl aus der geforderten Menge. Damit die Klasse `Random` verfügbar ist, importieren Sie diese mit `import java.util.Random;`.

Beim Ausführen der `main`-Methode soll Folgendes ausgegeben werden:

```
Durchschnitt: -40.8
```

b) Damit unsere Daten auch für "Laien" interpretierbar sind, ordnen wir den Daten die Farben `GRUEN`, `GELB` und `ROT` zu. Schreiben Sie ein Enum `OurColor` für diese Farben und schreiben Sie in der Klasse `Statistics` die Methode `public OurColor interpret(double ratio, int value) { ... }`. Wenn d der Durchschnitt der Werte des `Statistics`-Objekts ist, auf dem `interpret` aufgerufen wurde, dann gibt `interpret` für den Wert `value` die Farbe `GELB` zurück, falls `value` im Intervall von $d - \texttt{ratio} \cdot |d|$ bis $d + \texttt{ratio} \cdot |d|$ liegt. Hierbei nehmen wir an, dass `ratio` eine Gleitkommazahl zwischen 0 und 1 ist. Ansonsten kann sich Ihre Methode beliebig verhalten. Bei Werten, die kleiner sind als $d - \texttt{ratio} \cdot |d|$, wird die Farbe `ROT` zurückgegeben. Hingegen wird bei Werten, die größer sind als $d + \texttt{ratio} \cdot |d|$, die Farbe `GRUEN` zurückgegeben.

Hinweise:

- Sie können die Methode `Math.abs(double x)` verwenden, um den Absolutbetrag einer `double`-Zahl `x` zu berechnen.

```
import java.util.Random;

public class Statistics {

    // ...

    public static void main() {
        Statistics statistics = new Statistics();
        statistics.addValues(2,105,-366,44,11);
        IO.println("Durchschnitt: " + statistics.getAverage());
    }

    public void addValues(int... values) {
        // ...
    }

    public double getAverage() {
        // ...
    }

    public Statistics generate(int min, int max, int size) {
        // ...
    }

    public OurColor interpret(double ratio, int value) {
        // ...
    }
}
```

Hausaufgabe 8 (Deck 4): (Codescape)

Lösen Sie die Missionen von Deck 4 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn sie Ihre Lösung vor der einheitlichen Codescape Deadline am Freitag, den 30.01.2026, um 23:59 Uhr abschicken.