Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, N. Lommen, D. Meier, F. Meyer

Tutoraufgabe 1 (Überblickswissen):

a) Wie unterscheiden sich Referenzvariablen von Wertvariablen? Geben Sie ein kleines Java-Beispiel an, in dem dieser Unterschied deutlich wird.

b) Welche Schleifenarten haben Sie bisher kennengelernt? Nennen Sie je einen typischen Anwendungsfall!

c) Was ist der grundlegende Unterschied zwischen einer Klasse und einem Objekt?

d) Variablen können in einer Klasse, aber auch in einer Methode deklariert werden. Worin besteht der Unterschied und in welchen Fällen deklariert man Variablen typischerweise an welcher Stelle?

e) Was ist der Unterschied zwischen *Call By Reference* und *Call By Value*? Inwieweit unterstützt Java die beiden Konzepte?

Tutoraufgabe 2 (Programmierung):

Bubblesort ist ein Algorithmus zum Sortieren von Arrays, der wie folgt vorgeht, um ein Array `a` zu sortieren: Das Array wird wiederholt von links nach rechts durchlaufen. Am Ende des n-ten Durchlauf gilt, dass die letzten n Array-Elemente an ihrer endgültigen Position stehen. Folglich müssen im $(n+1)$-ten Durchlauf nur noch die ersten `a.length`$-n$ Elemente betrachtet werden. In jedem Durchlauf wird in jedem Schritt das aktuelle Element mit seinem rechten Nachbarn verglichen. Falls das aktuelle Element größer ist als sein rechter Nachbar, werden sie getauscht.
Als Beispiel betrachten wir das Array `{3,2,1}`. Im ersten Durchlauf wird erst `3` mit `2` getauscht (dies ergibt `{2,3,1}`) und dann `3` mit `1`, was `{2,1,3}` ergibt. Im zweiten Durchlauf wird `2` mit `1` getauscht, was zu `{1,2,3}` führt.
Implementieren Sie eine Klasse `BubbleSort` mit einer Methode `public static void sort(int[] a)`, die das Array `a` mithilfe des Algorithmus *Bubblesort* aufsteigend sortiert.

Hinweise:

- Im Moodle-Lernraum stehen die beiden Java-Dateien `BubbleSort.java` und `BubbleSortTest.java` zum Download zur Verfügung. Speichern Sie beide Dateien in einem neuen Ordner.
  Die Klasse `BubbleSort` enthält eine Methode `sort` mit leerem Rumpf. Wenn Sie die Implementierung vervollständigen und anschließend mit `javac BubbleSortTest.java` kompilieren, dann können Sie Ihre Implementierung mit `java BubbleSortTest` testen.

Tutoraufgabe 4 (Verifikation mit Arrays):

Gegeben sei folgendes Java-Programm P, wobei `x` und `i` `int`-Variablen sind, `res` eine `boolean`-Variable und `a` ein Array vom Typ `int[]` ist:

⟨ `true` ⟩ (Vorbedingung)

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

⟨ `res` $=$ `x` $\in \{$`a`$[j] \mid 0 \leq j \leq$ `a.length`$-1\}$ ⟩ (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.

Beachten Sie bei der Anwendung der "Bedingungsregel 1" mit Vorbedingung φ, Nachbedingung ψ und `if`-Bedingung B, dass $\varphi \wedge \neg B \implies \psi$ gelten muss, d. h. die Nachbedingung ψ der `if`-Anweisung muss aus der Vorbedingung φ der `if`-Anweisung und der negierten Bedingung $\neg B$ folgen. Geben Sie beim Verwenden der Regel einen entsprechenden Beweis an.

Hinweise:

- Gehen Sie davon aus, dass keine Integer-Überlaufe 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.
- Der Ausdruck `x` $\in$ `M` hat den Wert `true`, wenn `x` in der Menge `M` enthalten ist, sonst hat der Ausdruck den Wert `false`.

$\langle$`true`$\rangle$

$\langle$_____$\rangle$

```
i = 0;
```

$\langle$_____$\rangle$

```
res = false;
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

$\langle$_____$\rangle$

```
        res = true;
```

$\langle$_____$\rangle$

```
    }
```

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

```
}
```

$\langle$_____$\rangle$

$\langle \texttt{res} = \texttt{x} \in \{\texttt{a}[j] \mid 0 \leq j \leq \texttt{a.length} - 1\} \rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und unter Verwendung des Hoare-Kalküls die Terminierung bewiesen werden.

Geben Sie auch bei dieser Teilaufgabe einen Beweis für die Aussage $\varphi \wedge \neg B \implies \psi$ bei der Anwendung der "Bedingungsregel 1" an.

In den Aufgaben 6 bis 8 sollen Sie Speicherzustände zeichnen. Angenommen wir haben folgenden Java Code:

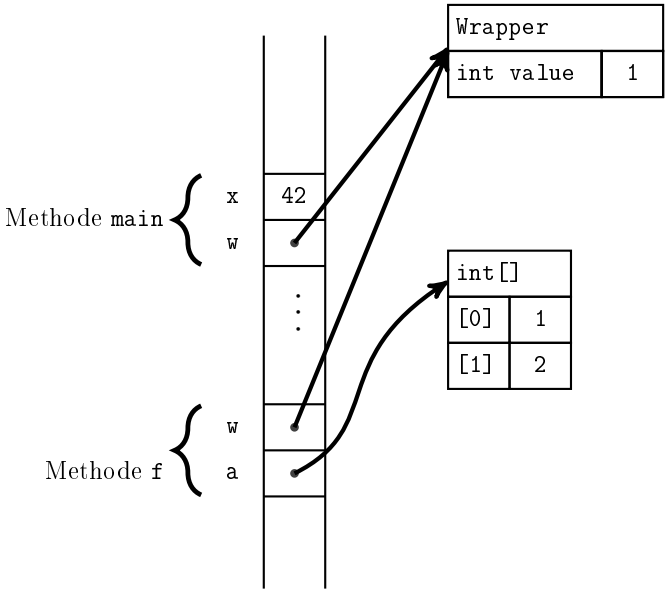
```
public class Wrapper {
    int value;
}
```

```
public class Main {
    public static void main(String[] args) {
        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:



Tutoraufgabe 6 (Seiteneffekte):

Betrachten Sie das folgende Programm:

```
public class TSeiteneffekte {
    public static void main(String[] args) {
        TWrapper[] ws = new TWrapper[2];
        ws[0] = new TWrapper();
        ws[1] = new TWrapper();

        ws[0].value = 2;
        ws[1].value = 1;

        f(ws[1], new TWrapper[] { ws[1], ws[0] });

        // Speicherzustand hier zeichnen
    }

    public static void f(TWrapper w1, TWrapper[] ws) {
```

```
        int sum = 0;

        // Speicherzustand hier zeichnen

        for (int j = 0; j < ws.length; j++) {
            TWrapper w = ws[j];
            sum += w.value;
            w.value = j + 2;
        }

        // Speicherzustand hier zeichnen

        w1 = ws[1];
        w1.value = -sum;
    }
}

public class TWrapper {
    int value;
}
```

Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher an allen drei markierten Programmzuständen graphisch dar. Achten Sie darauf, dass Sie alle (implizit) im Programm vorkommenden Arrays (außer `args`) und alle Objekte sowie die zu dem Zeitpunkt existierenden Programmvariablen darstellen.

Tutoraufgabe 7 (Seiteneffekte (Video)):

Betrachten Sie das folgende Programm:

```
public class VSeiteneffekte {
    public static void main(String[] args) {
        VWrapper w1 = new VWrapper();
        VWrapper w2 = w1;

        w1.i = 1;
        w2.i = 2;

        int x = 3;
        int[] a = { 1, 2 };

        f(w1, x, new int[] { 4, 5 });
        f(w2, x, a);
        //Speicherzustand hier zeichnen
    }

    public static void f(VWrapper w, int x, int[] a) {
        //Speicherzustand hier zeichnen
        x = a[0];
        a[0] = w.i;
        w.i = x;
    }
}

public class VWrapper {
    int i;
}
```

Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher an allen drei markierten Programmzuständen graphisch dar. Achten Sie darauf, dass Sie alle (implizit) im Programm vorkommenden Arrays (außer `args`) und alle Objekte sowie die zu dem Zeitpunkt existierenden Programmvariablen darstellen.