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

Aufgabe 3 (Programmierung): (8 + 9 = 17 Punkte)

Selectionsort 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 $(n-1)$-ten Durchlaufs gilt, dass die ersten $(n-1)$ Array-Elemente an ihrer endgültigen Position stehen, d.h. bis zur Stelle $n-2$ ist das Array bereits korrekt sortiert. Folglich müssen im n-ten Durchlauf nur noch die letzten `a.length` $- n + 1$ Elemente betrachtet werden. Im n-ten Durchlauf wird die Position i_{min} des kleinsten Elements der letzten `a.length` $- n + 1$ Elemente bestimmt. Danach wird das Element an Position $n-1$ mit dem Minimum an Position i_{min} vertauscht.
Als Beispiel betrachten wir das Array `{3,7,1,5,8}`.
Im ersten Durchlauf $(n = 1)$ wird $i_{min} = 2$ bestimmt, da die kleinste Zahl (1) des Arrays an Position 2 steht. Sie wird daher mit dem Element 3 an Position $n - 1 = 0$ vertauscht und es entsteht das Array `{1,7,3,5,8}`.
Im zweiten Durchlauf $(n = 2)$ bestimmt man $i_{min} = 2$ und erhält `{1,3,7,5,8}`.
Im dritten Durchlauf ergibt sich $i_{min} = 3$, dies führt zu `{1,3,5,7,8}`.
Im vierten und letzten Durchlauf $(n = 4)$ wird wieder $i_{min} = 3$ bestimmt. Die Vertauschung des Elements an der Position $n - 1 = 3$ mit dem Element an der Position $i_{min} = 3$ ändert daher nichts an dem Array. Das sortierte Array ist also `{1,3,5,7,8}`.

Hinweise:

- Im Moodle-Lernraum stehen die Java-Dateien `SelectionSort.java`, `ImprovedSelectionSort.java`, `SelectionSortTest.java` und `ImprovedSelectionSortTest.java` zum Download zur Verfügung. Speichern Sie die Dateien in einem neuen Ordner.
  Die Klassen `SelectionSort` und `ImprovedSelectionSort` enthalten je eine Methode `sort` mit leerem Rumpf. Wenn Sie die Implementierung vervollständigen und dann mit `javac SelectionSortTest.java` bzw. `javac ImprovedSelectionSortTest.java` kompilieren, dann können Sie die Implementierungen mit `java SelectionSortTest` bzw. `java ImprovedSelectionSortTest` testen.

a) Implementieren Sie eine Klasse `SelectionSort` mit einer Methode `public static void sort(int[] a)`, die das Array `a` mithilfe des Algorithmus *Selectionsort* aufsteigend sortiert.

b) Eine verbesserte Variante des Algorithmus (*ImprovedSelectionSort*) bestimmt in jedem Durchlauf nicht nur die Position i_{min} des kleinsten Elements, sondern auch die Position i_{max} des größten Elements. Am Ende des n-ten Durchlaufs werden die Elemente dann so getauscht, dass das Element, das vor dem Durchlauf an Position i_{min} gestanden hat, nun an Position $n-1$ steht und das Element, das vor dem Durchlauf an Position i_{max} gestanden hat, nun an Position `a.length` $- n$ steht. Durchsucht werden in dieser Variante im n-ten Durchlauf dann die Elemente von Position $n-1$ bis einschließlich `a.length` $- n$. Sollte es sich dabei um 1 oder 0 Elemente handeln, bricht der Algorithmus ab.

Als Beispiel betrachten wir das Array `{16,11,12,17,13,15,14}`.

Im ersten Durchlauf $(n = 1)$ wird $i_{min} = 1$ und $i_{max} = 3$ bestimmt, da die kleinste Zahl (11) des Arrays an Position 1 und die größte Zahl (17) an Position 3 steht. Es wird daher so getauscht, dass die 17 an Position `a.length` $- n = 6$ und die 11 an Position $n - 1 = 0$ steht. Es entsteht z.B. das Array `{11,16,12,14,13,15,17}`.

Im zweiten Durchlauf $(n = 2)$ wird $i_{min} = 2$ und $i_{max} = 1$ bestimmt. (Beachten Sie zu dieser Tauschoperation den untenstehenden Hinweis.) Es entsteht das Array `{11,12,15,14,13,16,17}`.

Im dritten und letzten Durchlauf $(n = 3)$ wird $i_{min} = 4$ und $i_{max} = 2$ bestimmt, es entsteht das Array `{11,12,13,14,15,16,17}`.

Hinweise:

- Beachten Sie, dass die Tauschoperationen in dieser Variante fehleranfällig sind: So sind, je nach Position des kleinsten und größten Elements, entweder 2, 3 oder 4 Elemente in die Tauschoperationen eingebunden. Gestalten Sie Ihre Implementierung so, dass jeder dieser Fälle korrekt behandelt wird.

Lösung:

a)
```
public class SelectionSort {

    public static void sort(int[] a) {
        int len = a.length;
        int i_min, min, tmp;
        for (int i = 0; i < len; i++) {
            i_min = i;
            min = a[i];
            for (int j = i+1 ; j < len ; j++) {
                if (min > a[j]) {
                  min = a[j];
                  i_min = j;
                }
            }
            tmp = a[i];
            a[i] = a[i_min];
            a[i_min] = tmp;
        }
    }

}
```

b)
```
public class ImprovedSelectionSort {

    public static void sort(int[] a) {
        int len = a.length;
        int i_min, i_max, min, max, tmp;
        for (int i = 0; i < len/2; i++) {
            i_min = i;
            i_max = i;
            min = a[i];
            max = a[i];
            for (int j = i+1 ; j < len - i; j++) {
                if (min > a[j]) {
                    min = a[j];
                    i_min = j;
                }
                if (max < a[j]) {
                    max = a[j];
                    i_max = j;
                }
            }

            tmp = a[i];
            a[i] = a[i_min];
            a[i_min] = tmp;

            if (i == i_max) {
                i_max = i_min;
            }

            tmp = a[len - (i + 1)];
            a[len - (i + 1)] = a[i_max];
            a[i_max] = tmp;
        }
    }
```

```
}
```

Aufgabe 5 (Verifikation mit Arrays): (10 + 4 + 2 = 16 Punkte)

Gegeben sei folgendes Java-Programm P. Es hat Ähnlichkeit zum *Bubblesort*-Algorithmus, den wir schon aus Tutoraufgabe 2 kennen. Hierbei sind `i` und `ac` `int`-Variablen und `a` ist ein Array vom Typ `int[]`.

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

```
i = 0;
while (i < a.length - 1) {
    if (a[i] > a[i+1]) {
        ac = a[i];
        a[i] = a[i+1];
        a[i+1] = ac;
    }
    i = i + 1;
}
```

$\langle$ `a[a.length - 1]` $= \max\{$ `a[`j`]` $\mid 0 \leq j <$ `a.length` $\}$ $\rangle$ (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.

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, auch die bereits ausgefüllten Zeilen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Die bereits ausgefüllten Zusicherungs-Zeilen zwischen Vor- und Nachbedingung sollen lediglich als Hinweis dienen. Sie müssen von Ihnen nicht benutzt werden, kommen in der Musterlösung allerdings genau so vor.
- 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.

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 unter der Voraussetzung `a.length` > 0 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.

c) Der Algorithmus P ist ähnlich zum in Tutoraufgabe 2 behandelten Algorithmus *Bubblesort*. Dennoch wurde in Teilaufgabe a) dieser Hausaufgabe noch nicht die partielle Korrektheit dieses Sortieralgorithmus bewiesen. Nennen Sie die beiden Hauptgründe dafür!

Lösung:

a) Mit Hilfe der Schleife wird das Array `a` von der ersten Stelle (`a[0]`) bis zur vorletzten Stelle (`a[a.length-2]`) durchlaufen. Gleich zu Beginn wird die Variable `i` mit `0` initialisiert, deshalb gilt dann lokal: $\texttt{a[i]} = \max\{\, \texttt{a[}j\texttt{]} \mid 0 \leq j \leq \texttt{i} \,\}$. Im Schleifenkörper wird der Wert von `a[i+1]` (ggf. durch Tausch mit `a[i]`) so angepasst, dass der nächstgrößere Index des Arrays für das Maximum mit betrachtet wird. So kann man die Aussage zu $\texttt{a[i]} = \max\{\, \texttt{a[}j\texttt{]} \mid 0 \leq j \leq \texttt{i} \,\}$ verallgemeinern. Um mit der negierten Schleifenbedingung $\neg(\texttt{i} < \texttt{a.length} - 1)$ die Nachbedingung folgern zu können, brauchen wir zusätzlich noch die Aussage, dass nach dem Schleifendurchlauf $\texttt{i} = \texttt{a.length} - 1$ gilt. Dies können wir folgern, wenn wir zu der bisherigen Invariante $\texttt{i} \leq \texttt{a.length} - 1$ hinzufügen.

So ergibt sich die Schleifeninvariante $\texttt{i} \leq \texttt{a.length} - 1 \wedge \texttt{a[i]} = \max\{\, \texttt{a[}j\texttt{]} \mid 0 \leq j \leq \texttt{i} \,\}$ (diese Begründung war nicht gefordert und dient nur zur Erklärung der Musterlösung).

```
                        ⟨a.length > 0⟩
                        ⟨a.length > 0 ∧ 0 = 0⟩
i = 0;
                        ⟨a.length > 0 ∧ i = 0⟩
                        ⟨i ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i }⟩
while (i < a.length - 1) {
                        ⟨i ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i } ∧ i < a.length − 1⟩
                        ⟨i + 1 ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i }⟩ es macht keinen unterschied,
    if (a[i] > a[i+1]) {                                                        ob man bis i oder i+1 geht
                        ⟨i + 1 ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i } ∧ a[i] > a[i+1]⟩
                        ⟨i + 1 ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i + 1 }⟩
                        ⟨i+1 ≤ a.length−1∧a[i] = max({a[j] | 0 ≤ j ≤ i−1} ∪ {a[i], a[i+1]})⟩
        ac = a[i];
                        ⟨i + 1 ≤ a.length − 1 ∧ ac = max({a[j] | 0 ≤ j ≤ i − 1} ∪ {ac, a[i+1]})⟩
        a[i] = a[i+1];
                        ⟨i + 1 ≤ a.length − 1 ∧ ac = max({a[j] | 0 ≤ j ≤ i − 1} ∪ {ac, a[i]})⟩
        a[i+1] = ac;
                        ⟨i+1 ≤ a.length−1∧a[i+1] = max({a[j] | 0 ≤ j ≤ i−1}∪{a[i+1], a[i]})⟩
                        ⟨i + 1 ≤ a.length − 1 ∧ a[i + 1] = max{ a[j] | 0 ≤ j ≤ i + 1 }⟩
    }
                        ⟨i + 1 ≤ a.length − 1 ∧ a[i + 1] = max{ a[j] | 0 ≤ j ≤ i + 1 }⟩
    i = i + 1;
                        ⟨i ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i }⟩
}
                        ⟨i ≤ a.length − 1 ∧ a[i] = max{ a[j] | 0 ≤ j ≤ i } ∧ ¬(i < a.length − 1)⟩
                        ⟨a[a.length - 1] = max{ a[j] | 0 ≤ j ≤ a.length − 1 }⟩
```

Damit die Bedingungsregel angewendet werden darf, muss überprüft werden, dass aus $\texttt{i}+1 \leq \texttt{a.length}-1 \wedge \texttt{a[i]} = \max\{\,\texttt{a}[j] \mid 0 \leq j \leq \texttt{i}\,\} \wedge \neg(\texttt{a[i]} > \texttt{a[i+1]})$ die Aussage $\texttt{i}+1 \leq \texttt{a.length}-1 \wedge \texttt{a[i+1]} = \max\{\,\texttt{a}[j] \mid 0 \leq j \leq \texttt{i}+1\,\}$ folgt. Da $\neg(\texttt{a[i]} > \texttt{a[i+1]})$ gilt, ist schon ohne Tausch das Element an Stelle `i+1` das größte, wenn das Element `a[i+1]` zusätzlich berücksichtigt wird.

b) Eine gültige Variante für die Terminierung ist $V = \texttt{a.length} - \texttt{i}$, denn die Schleifenbedingung $B = \texttt{i} < \texttt{a.length} - 1$ impliziert $\texttt{a.length} - \texttt{i} \geq 0$ und es gilt:

```
                        ⟨a.length − i = m ∧ i < a.length⟩
                        ⟨a.length − (i + 1) < m⟩
if (a[i] > a[i+1]) {
                        ⟨a.length − (i + 1) < m ∧ a[i] > a[i+1]⟩
                        ⟨a.length − (i + 1) < m⟩
    ac = a[i];
                        ⟨a.length − (i + 1) < m⟩
    a[i] = a[i+1];
                        ⟨a.length − (i + 1) < m⟩
    a[i+1] = ac;
                        ⟨a.length − (i + 1) < m⟩
}
                        ⟨a.length − (i + 1) < m⟩
i = i + 1;
                        ⟨a.length − i < m⟩
```

Die drei Zuweisungen in der `if`-Anweisung ändern offensichtlich die Länge des Arrays `a` nicht.

Damit die Bedingungsregel angewendet werden darf, muss überprüft werden, dass aus $\texttt{a.length} - (\texttt{i} + 1) < m \wedge \neg(\texttt{a[i]} > \texttt{a[i+1]})$ die Aussage $\texttt{a.length} - (\texttt{i} + 1) < m$ folgt. Dies ist allerdings offensichtlich.

Damit ist die Terminierung der einzigen Schleife in P gezeigt.

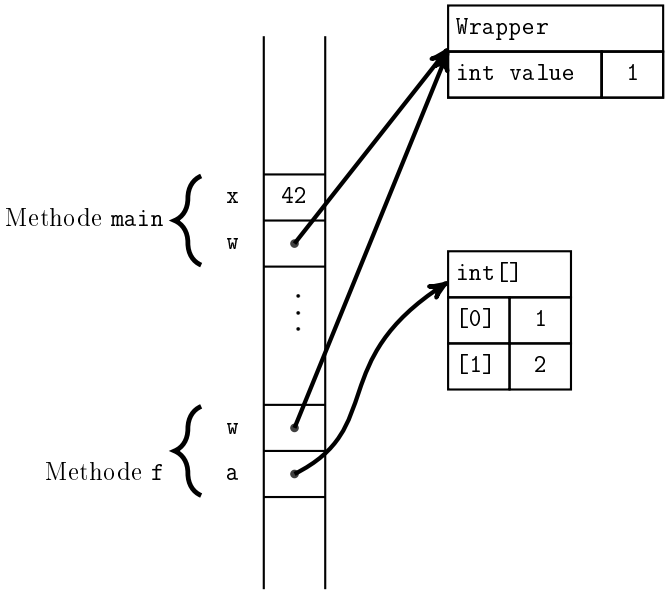
c) Zum Einen besteht *Bubblesort* aus zwei Schleifen, einer inneren und einer äußeren. Der Algorithmus P bildet aber nur die innere Schleife ab, also nur einen Durchgang durch das Array von vorne nach hinten. Danach ist ein Array der Länge n aber noch nicht notwendigerweise sortiert, denn dafür werden $n-1$ solcher Durchgänge benötigt.
Zum Anderen fordern wir in der Nachbedingung nur, dass nach Abschluss des Algorithmus das letzte Element auch das größte Element (oder zumindest eines der größten Elemente) im Array ist. Wir fordern aber nicht, dass sich nach wie vor dieselben Elemente (ggf. in anderer Reihenfolge) im Array befinden wie zu Beginn. Die Nachbedingung wäre bspw. auch erfüllt, wenn pauschal alle Elemente auf `0` gesetzt würden und nur das letzte auf `1`, auch, wenn ursprünglich eigentlich ganz andere Werte im Array gestanden haben.

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
    }
}
```

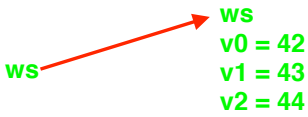
Dann sieht der Speicher an der markierten Stelle wie folgt aus:



Aufgabe 8 (Seiteneffekte): (17 Punkte)

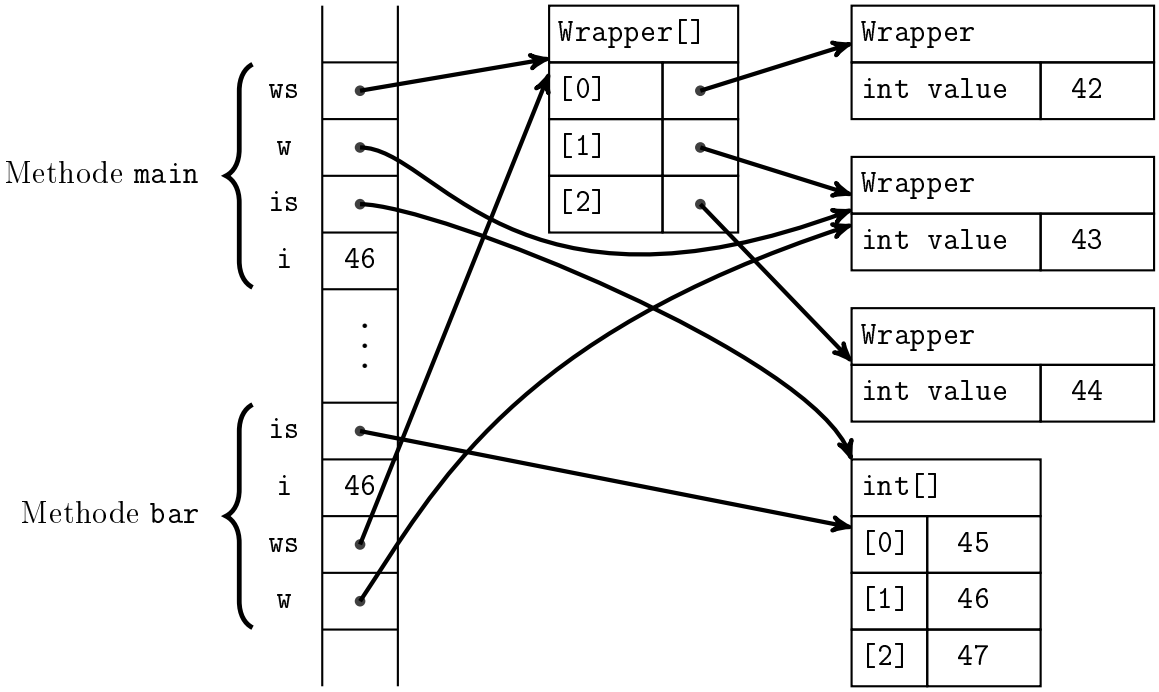
Betrachten Sie das folgende Programm:

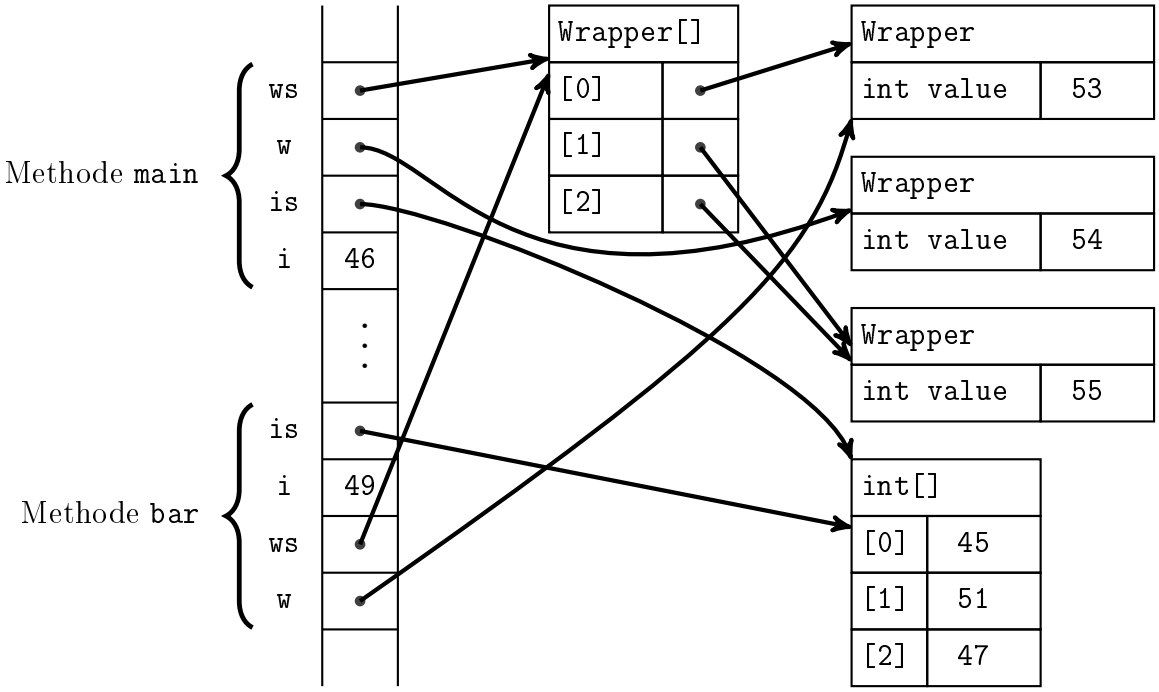
```
public class HSeiteneffekte {
    public static void main(String[] args) {
        Wrapper[] ws = new Wrapper[3];
        ws[0] = new Wrapper();
        ws[0].value = 42;
        ws[1] = new Wrapper();
        ws[1].value = 43;
        ws[2] = new Wrapper();
        ws[2].value = 44;
        Wrapper w = ws[1];

        int[] is = { 45, 46, 47 };
        int i = is[1];

        bar(is, i, ws, w);

        // Speicherzustand hier zeichnen
    }

    public static void bar(int[] is, int i, Wrapper[] ws, Wrapper w) {
        // Speicherzustand hier zeichnen

        i = 48;
        i = is[0];
        i = 49;
        is[1] = 50;
        is[1] = is[2];
        is[1] = 51;

        w.value = 52;
        w = ws[0];
        w.value = 53;
        ws[1].value = 54;
        ws[1] = ws[2];
        ws[1].value = 55;

        // Speicherzustand hier zeichnen

        is = new int[1];
        is[0] = 56;
        i = is[0];
        i = 57;

        ws = new Wrapper[1];
        ws[0] = new Wrapper();
        w = ws[0];
        w.value = 58;

        // Speicherzustand hier zeichnen
    }
}

public class Wrapper {
    int value;
}
```

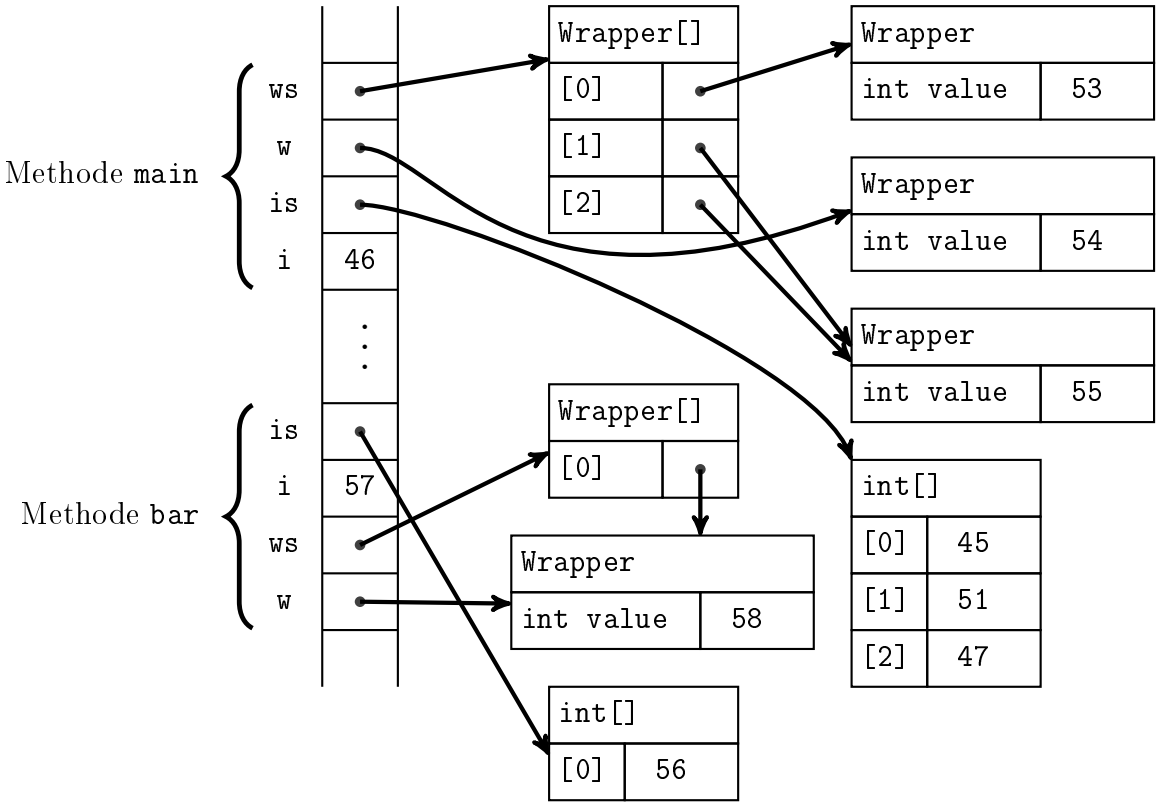


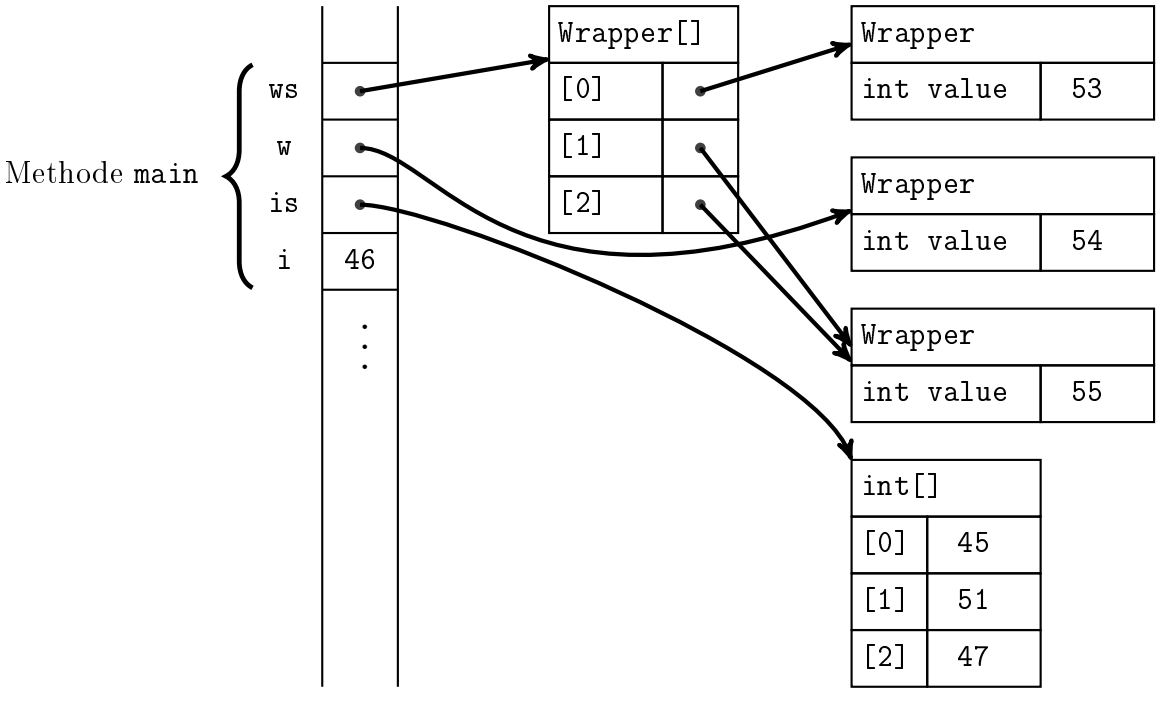
Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher an allen vier markierten Programmzuständen graphisch dar. Achten Sie darauf, dass Sie alle (implizit) im Programm vorkommenden Arrays (außer `args`) und alle Objekte sowie die zu dem Zeitpunkt existierenden Programmvariablen darstellen.

Lösung:


Methode main
ws
w
is
i
46
Methode bar
is
i
46
ws
w
Wrapper[]
[0]
[1]
[2]
Wrapper
int value
42
Wrapper
int value
43
Wrapper
int value
44
int[]
[0]
45
[1]
46
[2]
47


Methode main
ws
w
is
i
46
Methode bar
is
i
49
ws
w
Wrapper[]
[0]
[1]
[2]
Wrapper
int value
53
Wrapper
int value
54
Wrapper
int value
55
int[]
[0]
45
[1]
51
[2]
47





Aufgabe 9 (Deck 3): (Codescape)

Lösen Sie die Missionen von Deck 3 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 Samstag, den 22.01.2022, um 23:59 Uhr abschicken.

Lösung: