Prof. Dr. J. Giesl
S. Dollase, M. Hark, D. Korzeniewski

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter oben links!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Freitag, den 16.11.2018 um 12:00** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Freitag, den 16.11.2018 um 12:00 an Ihre Tutorin/Ihren Tutor. Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, ansonsten werden keine Punkte vergeben.
- Benutzen Sie in ihrem Code keine Umlaute, auch nicht in Strings und Kommentaren. Diese führen oft zu Problemen, da diese Zeichen auf verschiedenen Betriebssystemen unterschiedlich behandelt werden. Dadurch kann es dazu führen, dass Ihr Programm bei Ihrer Tutorin/Ihrem Tutor bei Verwendung von Umlauten nicht mehr von `javac` akzeptiert wird.
- Halten Sie sich beim Lösen von Programmieraufgaben an die auf der Website zur Vorlesung verfügbaren Codekonventionen. Verstöße, die zu unleserlichem Code führen, können zu Punktabzug führen.
- Für einige Programmieraufgaben benötigen Sie die Java Klasse `SimpleIO`. Diese können Sie auf der Website herunterladen.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden:
  https://codescape.medien.rwth-aachen.de/progra/
  Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Tutoraufgabe 1 (Programmierung mit Arrays):

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.

Aufgabe 2 (Programmierung mit Arrays): (3 + 3 + 3 = 9 Punkte)

In der Mathematik ist das Transponieren einer Matrix eine elementare Rechenoperation. Dazu werden alle Einträge der Matrix an ihrer Hauptdiagonalen gespiegelt.
So wird zum Beispiel die Matrix

$$A = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$

durch Transponieren umgewandelt in

$$A^T = \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

Implementieren Sie eine Methode, welche

- eine Matrix beliebiger Größe (mindestens 1×1) vom Benutzer einliest,
- die Matrix transponiert und
- die Matrix sowie die transponierte Matrix in geeigneter Form ausgibt.

Die Matrixgröße sowie ihre Einträge sollen also vom Benutzer festgelegt werden können.
Nutzen Sie für die Ausgabe der Matrix eine `foreach` Schleife.
Zur Ein- und Ausgabe dürfen Sie die Klasse `SimpleIO` sowie den Befehl `System.out.println(...)` benutzen.
Weitere vordefinierte Methoden dürfen nicht verwendet werden.
Ein Beispieldurchlauf kann wie folgt aussehen:

```
Wie viele Zeilen hat die Matrix? (>= 1)
-2
Wie viele Zeilen hat die Matrix? (>= 1)
2
Wie viele Spalten hat die Matrix? (>= 1)
3
Wie lautet die Zahl fuer Position (1, 1)?
1
Wie lautet die Zahl fuer Position (1, 2)?
2
Wie lautet die Zahl fuer Position (1, 3)?
3
Wie lautet die Zahl fuer Position (2, 1)?
4
Wie lautet die Zahl fuer Position (2, 2)?
5
Wie lautet die Zahl fuer Position (2, 3)?
6
Matrix:
1, 2, 3
4, 5, 6
Transponierte Matrix:
1, 4
2, 5
3, 6
```

Tutoraufgabe 3 (Verifikation):

Gegeben sei folgendes Java-Programm P über der `int[]`-Variable `a`, der `int`-Variable `i` und der `boolean`-Variable `result`:

$\langle\, 0 \leq \texttt{a.length} \,\rangle$ (Vorbedingung)

```
i = 0;
result = false;
while(i < a.length) {
    if(x == a[i]) {
        result = true;
```

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

$\langle$ `result` $=$ `x` $\in \{$`a`$[j] \mid 0 \leq j \leq$ `a.length`$-1\}$ $\rangle$ (Nachbedingung)

a) Vervollständigen Sie die folgende Verifikation 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 und Nachbedingung ψ, dass auch $\varphi \wedge \neg B \implies \psi$ 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:

- 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`.

$\langle 0 \leq$ `a.length` $\rangle$

$\langle$ _____ $\rangle$

```
i = 0;
```

$\langle$ _____ $\rangle$

```
result = false;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

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

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

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

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
        result = true;
```

$\langle$ _____ $\rangle$

```
    }
```

$\langle$ _____ $\rangle$

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

$\langle$ _____ $\rangle$

```
}
```

$\langle$ _____ $\rangle$

$\langle$ `result` $= x \in \{$`a`$[j] \mid 0 \leq j \leq$ `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.

Aufgabe 4 (Verifikation): (8 + 2 = 10 Punkte)

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

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

```
x = 1;
result = true;
```

```
i = 0;
while (i < a.length) {
    if (a[i] != x) {
        result = false;
    }
    x = x * 2;
    i = i + 1;
}
```

$\langle$ `result` $= \forall 0 \leq j <$ `a.length` $:$ `a`$[j] = 2^j$ $\rangle$ (Nachbedingung)

a) Vervollständigen Sie die folgende Verifikation 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 und Nachbedingung ψ, dass auch $\varphi \wedge \neg B \implies \psi$ 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:

- 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 `result` $= \forall 0 \leq j <$ `a.length` $:$ `a`$[j] = 2^j$ hat den Wert `true`, wenn für jede natürliche Zahl $j \in \{0, 1, \ldots,$ `a.length` $- 1\}$ jeweils `a`$[j] = 2^j$ gilt, sonst hat der Ausdruck den Wert `false`. Ist `a.length` kleiner als 1 (`a` ist ein leeres Array), so hat der Ausdruck ebenfalls den Wert `true`. Die Nachbedingung in unserem Beispiel besagt also, dass `result` genau dann den Wert `true` hat, wenn an jeder Stelle j des Arrays der Wert 2^j steht.

$\langle 0 \leq$ `a.length`$\rangle$

$\langle$_____$\rangle$

```
x = 1;
```

$\langle$_____$\rangle$

```
result = true;
```

$\langle$_____$\rangle$

```
i = 0;
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

$\langle$_____$\rangle$

```
        result = false;
```

$\langle$_____$\rangle$

```
    }
```

$\langle$_____$\rangle$

```
    x = x * 2;
```

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

```
}
```

$\langle$_____$\rangle$

$\langle$`result` $= \forall 0 \leq j <$ `a.length` $:$ `a`$[j] = 2^j\rangle$

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 nächsten beiden Aufgaben sollen Sie Speicherzustände zeichnen. Angenommen wir haben folgenden `Java` Code.

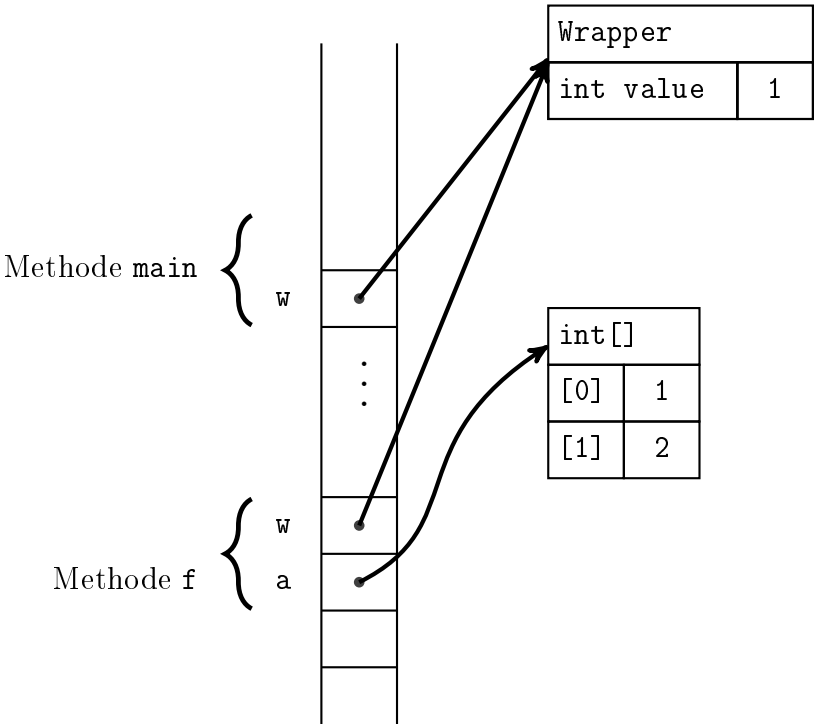
```
public class Wrapper {
    int value;
}
```

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

Betrachten Sie das folgende Programm:

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

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

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

```
        // Speicherzustand hier zeichnen
    }

    public static void f(Wrapper w1, Wrapper[] ws) {
        int sum = 0;

        // Speicherzustand hier zeichnen

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

        // Speicherzustand hier zeichnen

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

public class Wrapper {
    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.

Aufgabe 6 (Seiteneffekte): (2 + 2 + 2 + 1 = 7 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 = 100;
        ws[1] = new Wrapper();
        ws[1].value = 101;
        ws[2] = new Wrapper();
        ws[2].value = 102;
        Wrapper w = ws[1];

        int[] is = { 103, 104, 105 };
        int i = is[1];

        f(ws, w, is, i);

        // Speicherzustand hier zeichnen
    }

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

        i = 106;
```

```
        i = is[0];
        i = 107;
        is[1] = 108;
        is[1] = is[2];
        is[1] = 109;

        w.value = 110;
        w = ws[0];
        w.value = 111;
        ws[1].value = 112;
        ws[1] = ws[2];
        ws[1].value = 113;

        // Speicherzustand hier zeichnen

        is = new int[1];
        is[0] = 114;
        i = is[0];
        i = 115;

        ws = new Wrapper[1];
        ws[0] = new Wrapper();
        w = ws[0];
        w.value = 116;

        // 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.

Tutoraufgabe 7 (Einfache Klassen):

In dieser Aufgabe beschäftigen wir uns mit dem berühmten Gaunerpärchen *Bonnie und Clyde*. Wenn die beiden nicht gerade Banken ausrauben, gehen sie gerne im Wald Pilze sammeln (bzw. klauen).
Wir verwenden hier die Klassen `Main`, `Mensch` und `Pilz`, die Sie auf der Homepage herunterladen können.
Jeder dieser (beiden) **Mensch**en hat einen Korb, in den eine feste Anzahl von Pilzen passt. Weiterhin hat jeder Mensch einen Namen. Wie Sie in der Klasse `Mensch` sehen können, gibt es hierfür drei Attribute. Das Attribut `anzahl` gibt hierbei an, wie viele Pilze bereits im Korb enthalten sind.
Zu jedem **Pilz** kennen wir den Namen.

a) Vervollständigen Sie die Klasse `Main` wie folgt:

- Ergänzen Sie an den mit `TODO a.1)` markierten Stellen den Code so, dass die Variablen `steinpilz`, `champignon`, `pfifferling` auf Pilz-Objekte mit passenden Namen verweisen.
- Ergänzen Sie an den mit `TODO a.2)` markierten Stellen den Code so, dass die Variablen `bonnie` und `clyde` auf passende Mensch-Objekte zeigen. Setzen Sie hierfür jeweils den passenden Namen und sorgen Sie dafür, dass in Bonnies Korb maximal 3 Pilze Platz haben. Bei Clyde passen 4 Pilze in den Korb.

b) Gehen Sie in dieser Teilaufgabe davon aus, dass die Attribute bereits alle auf vernünftige Werte gesetzt sind.

- Implementieren Sie in der Klasse `Mensch` eine Methode `hatPlatz()`, die `true` genau dann zurückgibt, wenn im Korb Platz für einen weiteren Pilz ist. Anderenfalls wird `false` zurückgegeben.
- Schreiben Sie in der Klasse `Mensch` eine Methode `ausgabe()`. Diese gibt kein Ergebnis zurück, aber sie gibt den Namen und eine lesbare Übersicht der von der Person gesammelten Pilze aus. Geben Sie in der ersten Zeile den Namen der Person gefolgt von einem Doppelpunkt („:“) aus. Schreiben Sie pro Pilz im Korb eine weitere Zeile, in der (nur) der Name des jeweiligen Pilzes steht. Verwenden Sie hierzu eine `foreach` Schleife.

  Hier ist eine Beispielausgabe von einem Menschen mit Namen „Gustav“ und einem Korb, der einen Pilz mit Namen „Morchel“ und einen Pilz mit Namen „Steinpilz“ enthält:

  ```
  Gustav:
  Morchel
  Steinpilz
  ```

c) Vervollständigen Sie die Klasse `Main` wie folgt:

- Schreiben Sie eine Methode `public static void sammle(Pilz[] wald, Mensch[] menschen)`. Diese arbeitet die Pilze im Wald nach und nach ab. Dabei füllt sie die Körbe der eingegebenen Menschen der Reihe nach auf. Hierbei wird zuerst überprüft, welcher der erste der eingegebenen Menschen ist, der noch Platz hat. Gibt es noch einen solchen Menschen, so wird der Pilz in seinen Korb eingefügt. Benutzen Sie hierfür auch das Attribut `anzahl` und passen Sie seinen Wert entsprechend an. Gibt es keinen Menschen, der noch Platz in seinem Korb hat, passiert nichts. Nutzen Sie hierfür eine `foreach`-Schleife. Rufen Sie nach Abarbeitung eines jeden Pilzes die Methode `ausgabe()` für jeden Menschen auf. Geben Sie anschließend eine Zeile aus, in der nur „`---`“ (drei Bindestriche) steht.

Listing 1: Main.java

```
1  public class Main {
2    public static void sammle(Pilz[] wald, Mensch[] menschen){
3      // TODO c)
4    }
5
6
7    public static void main(String[] args) {
8      Pilz steinpilz = // TODO a.1)
9
10     Pilz champignon = // TODO a.1)
11
12     Pilz pfifferling = // TODO a.1)
13
14     Mensch bonnie = // TODO a.2)
15
16     Mensch clyde = // TODO a.2)
17
18     Pilz[] wald = {steinpilz, champignon, champignon, pfifferling,
19       steinpilz, pfifferling, champignon};
20
21     Mensch[] menschen = { bonnie, clyde };
22
23     sammle(wald, menschen);
24   }
25 }
```

Listing 2: Mensch.java

```
1  public class Mensch {
2    String name;
3    Pilz[] korb;
4    int anzahl = 0;
5  }
```

Listing 3: Pilz.java

```
1  public class Pilz {
2    String name;
3  }
```

Aufgabe 8 (Einfache Klassen): (6 Punkte)

Heutzutage werden sehr viele Daten gesammelt. Um einen Überblick über die Daten zu erhalten, ist es sinnvoll sich diese automatisch zusammenfassen zu lassen.
Implementieren Sie die Klasse `Statistics`. Ein Objekt der Klasse `Statistics` kann bis zu 100 `int`-Werte zusammenfassen. Die Klasse `Statistics` soll folgende Methoden bereitstellen:

- `public void addValue(int value) { ... }`
  Fügt einen einzelnen Wert hinzu.
- `public int getMinimum() { ... }`
  Gibt den minimalen Wert aller bisher hinzugefügten Werte zurück.
- `public int getMaximum() { ... }`
  Gibt den maximalen Wert aller bisher hinzugefügten Werte zurück.

- `public double getAverage() { ... }`
  Gibt den durchschnittlichen Wert aller bisher hinzugefügten Werte zurück.

Implementieren Sie diese vier Methoden. Dazu können Sie beliebige Attribute und Hilfsmethoden einfügen. Beachten Sie außerdem folgende Randfälle:

- Wird `getMinimum`, `getMaximum` oder `getAverage` aufgerufen, ohne dass zuvor Werte per `addValue` hinzugefügt wurden, so wird eine Fehlermeldung ausgegeben und der Wert `0` zurückgegeben.
- Wird `addValue` aufgerufen, nachdem bereits 100 Werte hinzugefügt wurden, so wird eine Fehlermeldung ausgegeben und der neue Wert wird nicht hinzugefügt.

Hinweise:

- Variablen des Typs `int` haben einen minimalen und einen maximalen Wert, welche im Programmcode durch die Ausdrücke `Integer.MIN_VALUE` sowie `Integer.MAX_VALUE` verfügbar sind.
- Verwenden Sie die angegebene `main`-Methode zum Testen Ihrer Implementierung.
- Bei der Berechnung des Durchschnitts dürfen Sie Variablenüberläufe ignorieren.

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

```
Minimum: -366
Maximum: 105
Durchschnitt: -40.8
```

Listing 4: Statistics.java

```
1  public class Statistics {
2      public static void main(String[] args) {
3          Statistics statistics = new Statistics();
4          statistics.addValue(2);
5          statistics.addValue(105);
6          statistics.addValue(-366);
7          statistics.addValue(44);
8          statistics.addValue(11);
9          SimpleIO.output("Minimum: " + statistics.getMinimum());
10         SimpleIO.output("Maximum: " + statistics.getMaximum());
11         SimpleIO.output("Durchschnitt: " + statistics.getAverage());
12     }
13
14     // ...
15
16     public void addValue(int value) {
17         // ...
18     }
19
20     public int getMinimum() {
21         // ...
22     }
23
24     public int getMaximum() {
25         // ...
26     }
27
28     public double getAverage() {
29         // ...
30     }
31 }
```

Aufgabe 9 (Codescape): (Codescape)

Lösen Sie die Räume von **Deck 2** des Spiels Codescape.
Ihre Lösung für Räume dieses Codescape Decks wird nur dann für die Zulassung gezählt, wenn Sie die Lösung bis Freitag, den 16.11.2018 um 12:00 abschicken.