Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 02

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwurf

Achtung: Dieses Dokument ist ein Entwurf und ist noch nicht zur Bearbeitung/Abgabe freigegeben. Es kann zu Änderungen kommen, die für die Abgabe relevant sind. Es ist möglich, dass sich **alle** Aufgaben noch grundlegend ändern. Es gibt keine Garantie, dass die Aufgaben auch in der endgültigen Version überhaupt noch vorkommen und es wird keine Rücksicht auf bereits abgegebene Lösungen genommen, die nicht die Vorgaben der endgültigen Version erfüllen.

Hausübung 02

Gesamt: 18 Punkte

Let them march

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* im Moodle-Kurs.

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit `TODO` markierten `crash`-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/h02` und ggf. `src/test/java/h02`.

Einleitung

In diesem Übungsblatt werden Sie einen Einblick in eines der wichtigsten Konstrukte der Programmiersprache Java erhalten: Bisher haben Sie vielleicht gemerkt, dass es etwas umständlich war, mehrere Roboter auf einmal bestimmte Aktionen ausführen zu lassen. Um nun auch wesentlich mehr, als ein paar wenige Roboter in der `World` agieren zu lassen, werden wir in diesem Übungsblatt das Konstrukt der *Arrays* verwenden, um dies zu vereinfachen. Einen ersten Einblick in die Funktionalität von Arrays sollten Sie nun bereits im Kapitel 01d der FOP gesammelt haben. Nun werden Sie eine erste Anwendung dieser erfahren.

Konkret werden Sie hier zunächst lernen, wie Sie ein Array des Typs `Robot` anhand eines „Array von Array von `boolean`“ besetzen. Damit Sie auch mit Fehlermeldungen (Exceptions), die bei der Verwendung von Arrays auftreten können, umgehen können, sollten Sie diese besser verstehen. Dafür werden Sie bewusst(!) solche Fehler provozieren, um zu verstehen, wie diese zustande kommen und was sie konkret bedeuten. Darüber hinaus, werden Sie auch die Roboter eines `Robot`-Arrays in der `World` „marschieren“ lassen und dabei ein bisschen den konkreten Umgang mit Arrays üben.

Hinweis:

Screenshots aus der Welt der Roboter dürfen Sie unbedenklich mit anderen Studierenden teilen – nur eben nicht den Quelltext oder eine übersetzte Variante des Quelltexts!

Hinweise:

In der `main`-Methode der Klasse `Main` finden Sie bereits Code-Zeilen, die eine `World` erstellen. Dabei werden die Anzahl der Spalten bzw. Zeilen der `World` mittels der Attribute `numberOfColumns` und `numberOfRows` festgelegt. Diese wiederum werden mittels der vorher definierten Methode `getRandomWorldSize` auf einen Wert zwischen 4 (*inklusiv*) und 10 (*exklusiv*) initialisiert, damit die `World` nicht zwangsweise immer die selbe Größe hat. Das der `World` in der Methode `setDelay` übergebene Attribut `DELAY` legt die Verzögerungen von Operationen (wie etwa `Robot.move`) in der `World` fest.

Es ist Ihnen natürlich erlaubt die erwähnten Attribute für Ihre persönlichen Tests zu modifizieren. Sie dürfen darüber hinaus auch die Codevorlage zunächst entfernen, um etwa H2.2 zu bearbeiten, damit sich nicht mit jedem Start der `main`-Methode die `World` öffnet.

H1: Erstellen eines `Robot`-Arrays — 2 Punkte

H1.1: Zählen von `true` in einem Array von Array von `boolean` — 2 Punkte

Sie finden nun also in der Vorlage die Methode `countRobotsInPattern`. Diese Methode hat einen Parameter `pattern` vom Typ „Array von Array von `boolean`“, sowie zwei Parameter namens `numberOfColumns` und `numberOfRows` vom primitiven Datentyp `int`. Die Methode darf bei Ihrer Implementation davon ausgehen, dass `pattern` nicht auf `null` verweist, also immer ein korrekt gebildetes „Array von Array von `boolean`“ enthält. Darüber hinaus dürfen Sie davon ausgehen, dass `numberOfColumns` und `numberOfRows` tatsächlich die Anzahl an Spalten bzw. Zeilen der `World` enthalten, wie es die Namen bereits suggerieren.

Konkret zählt Ihre Methode `countRobotsInPattern` die Anzahl an *zu besetzenden* Paaren (x, y) von natürlichen Zahlen[1]. Im Weiteren heißt ein Paar (x, y) von natürlichen Zahlen *zu besetzen gemäß* der `World` und des `pattern`, wenn die folgenden fünf Bedingungen erfüllt sind:

(a) x ist ein Spaltenindex der `World`;

(b) y ist ein Zeilenindex der `World`;

(c) x ist im Indexbereich des Arrays, auf das `pattern` verweist;

(d) y ist im Indexbereich des Arrays, auf das `pattern[`x`]` verweist;

(e) es gilt `pattern[`x`][`y`]==true`.

Um nun diese Anzahl zu bestimmen, erstellen Sie zunächst eine Variable namens `numberOfRobots`[2] und initialisieren diese mit 0. Danach zählen Sie in zwei ineinander geschachtelten `for`-Schleifen alle *zu besetzenden* Paare, erhöhen dabei sukzessive `numberOfRobots` und liefern diese Variable mittels `return` zurück.

H1.2: Erstellen des `Robot`-Arrays mittels eines Patterns

Nun finden Sie in der Vorlage die Methode `initializeRobotsPattern`. Auch hier gelten die selben Voraussetzungen bezüglich der gegebenen Parameter `pattern`, `numberOfColumns` und `numberOfRows`, wie auch schon in H1.1 bei der Methode `countRobotsInPattern` beschrieben.

[1] In der FOP gilt natürlich $0 \in \mathbb{N}$.
[2] Der Name ist natürlich Ihnen überlassen, jedoch wird die Variable im Folgenden mit diesem Namen beschrieben.

In der Methode `initializeRobotsPattern` erstellen Sie nun ein Array namens `allRobots`, befüllen dieses mit Robotern und liefern es mittels `return` zurück. Konkret soll das Array `allRobots` für jedes *zu besetzende Paar* genau *eine* und darüber hinaus *keine* weitere Komponente besitzen. Um nun bei der Erstellung des Arrays die richtige Größe zu bestimmen, verwenden Sie die in H1.1 implementierte Methode, der Sie genau die selben Parameter übergeben, die auch die Methode `initializeRobotsPattern` erhalten hat und zwar in der selben Reihenfolge.

Das Befüllen von `allRobots` verläuft nun ziemlich analog zu der Art und Weise des Zählens von *zu besetzenden* Paaren in H1.1: Sie verwenden wieder zwei ineinander geschachtelte `for`-Schleifen und erstellen zu jedem *zu besetzenden* Paar (x, y) ein `Robot`-Objekt. Dieser Roboter soll in Spalte x und in Zeile y stehen. Dabei sollen *alle* Roboter die Richtung `RIGHT` haben. Darüber hinaus soll ein Roboter in Spalte x genau über `numberOfColumns`$-x$ Münzen verfügen.

Verbindliche Anforderung:

Nutzen Sie in Methode `initializeRobotsPattern` die Methode `countRobotsInPattern`.

Bevor Sie nun mit der nächsten Aufgabe weitermachen, sollten Sie sich die Zeit nehmen und Ihre Implementation von `countNumberOfRobotsInPattern` und `initializeRobotsPattern` zu testen.

Darüber hinaus haben wir bereits in der Vorlage eine Datei namens `ExamplePattern.txt` zur Verfügung gestellt. Diese beinhaltet ein Muster von 1 und 0, bezeichnend für „Roboter“ und „kein Roboter“. Diese Zahlen sind (allerdings nicht zwangsweise) mit einem Leerzeichen getrennt. Um nun Ihre Implementation von `initializeRobotsPattern` zu testen, können Sie verschiedene Muster (auch größere oder kleinere) in die Datei eingeben und jedes Mal überprüfen, ob das Muster richtig in der `World` dargestellt wird, wenn Sie die `main`-Methode starten. Wir erwarten jedoch, wie auch schon bei Übungsblatt 01, dass Ihre Implementation auch mit anderen Werten funktioniert und die alle anderen Voraussetzungen erfüllt.

Um nun auch von Ihnen aus die Methode zu testen, können sie gerne verschiedene „Array von Array von `boolean`“ erstellen und auch hier Ihre Methode auf Korrektheit testen.

Exkurs:

Bislang haben Sie lediglich folgende Art und Weise kennengelernt, wie Sie Arrays zunächst konkret befüllen (siehe Kapitel 01d, ab Folie 4 der FOP):

```
int[] ints = new int[3];
ints[0] = 1;
ints[1] = 2;
ints[2] = 3;
```

Diese Zuweisungen können Sie auch in einer einzigen Anweisung zusammenfassen:

```
int[] ints = new int[]{1, 2, 3};
```

Auch das Erstellen eines „Array von Array“ lässt sich so vereinfachen:

```
int[][] ints = new int[][]{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Hinweis:

Sollten Sie eigene „Arrays von Array von `boolean`“ erstellen, müssen Sie darauf achten, dass die Besetzung der `World` *unten links* beginnt, wie Sie bereits in Kapitel 01a, Folien 12-19 der FOP kennengelernt haben.

H2: Vorübungen für das Weitere

Die folgenden Aufgaben sollen Sie auf die eigentliche Hauptaufgabe, das Implementieren der Hauptschleife, vorbereiten und Ihr generelles Verständnis von der Programmiersprache Java verbessern. Damit Sie Ihren Code direkt überprüfen können, empfehlen wir Ihnen diesen direkt in die `main`-Methode der Klasse `Main` zu schreiben, wenn nicht anders beschrieben (etwa in H2.1).

H2.1: Allgemein Fehlermeldungen besser verstehen

> **Hinweis:**
>
> Implementieren Sie alle Anweisungen zunächst in Methode `initializeRobotsPattern`.

Richten Sie eine Variable vom Typ `boolean` und eine Variable vom Typ `Robot` am Ende von `initializeRobotsPattern` ein; die Namen können Sie frei wählen im Rahmen der Regeln für Identifier (Kapitel 01a, Folien 168-191 der FOP). Weisen sie diesen Variablen nacheinander verschiedene Kompontenen von `pattern` bzw. `allRobots` zu. Lassen Sie sich nach jeder Zuweisung mit `System.out.println` den Inhalt Ihrer `boolean`-Variable sowie Zeile und Spalte bei Ihrer `Robot`-Variable ausgeben, um zu prüfen, ob Ihre Initialisierung der beiden Arrays korrekt ist.

Nun zum eigentlichen Thema von H2.1: Greifen Sie nun analog auf verschiedene Indizes von `pattern` und `allRobots` zu, die ***nicht*** im Indexbereich der beiden Arrays liegen, also auf ***nichtexistente*** Komponenten: jeweils mindestens eine negative Zahl, die Arraylänge (ein Blick auf Folie 13 in Kapitel 01d ist zu empfehlen) sowie eine Zahl größer Arraylänge. Bei `pattern` greifen Sie sowohl auf nichtexistente Komponenten `pattern[`x`][`y`]` zu, so dass x nicht im Indexbereich von `pattern` liegt, als auch auf nichtexistente Komponenten `pattern[`x`][`y`]`, so dass x im Indexbereich von `pattern`, aber y nicht im Indexbereich von `pattern[`x`]` liegt.

Sie können alle diese Zugriffe konkret dadurch realisieren, dass Sie ***eine einzelne*** Schreibanweisung einfügen, die eine solche Arraykomponente an einem Index außerhalb des Indexbereichs des Arrays ausgeben soll. Dort setzen Sie die verschiedenen Indizes für nichtexistente Komponenten nacheinander ein, kompilieren jeweils neu und lassen das Programm laufen.

> **Tipp:**
>
> Schreiben Sie alle Anweisungen untereinander, aber beim Compilen sind alle bis auf eine Anweisung auskommentiert.

Der Quelltext sollte jeweils durch den Compiler gehen, aber der Prozess sollte jeweils mit einer Fehlermeldung abgebrochen werden. Ergibt die in der Konsole ausgegebene Fehlermeldung für Sie Sinn?

> **Hinweis:**
>
> Entfernen Sie allen Java-Code, den Sie für diese Aufgabe eingefügt haben und potenziell eine Fehlermeldung bei Ausführung generieren würde, wieder aus `Main`.

H2.2: Vertauschen von Werten von Variablen

Sie haben nun in H2.1 verschiedene Fehlermeldungen kennengelernt, die Ihnen beim Gebrauch von Arrays über den Weg laufen könnten. In dieser Aufgabe widmen wir uns nun der Vorübungen für die **??**:

Richten Sie zwei Variable `euler` und `pi` von Typ `double` ein und initialisieren Sie sie mit den Werten 3.14 und 2.71. Lassen Sie sich die Werte von `euler` und `pi` wie üblich auf der Konsole ausgeben. Richten Sie nun eine weitere `double`-

Variable `tmp` ein,[3] mit deren Hilfe Sie die Werte von `euler` und `pi` vertauschen. Sie weisen dazu als erstes `tmp` den Wert von `euler` zu (also `tmp = euler;`). Machen Sie sich durch eine Ausgabe in der Konsole klar, dass nun der initiale Wert von `euler` in `tmp` „gerettet" ist und daher durch eine zweite Anweisung in `euler` durch den Wert von `pi` überschrieben werden kann, ohne verlorenzugehen. Damit ist auch der initiale Wert von `pi` gerettet, nämlich in `euler`. Als dritte und letzte Anweisung weisen Sie daher `pi` den passenden (welchen?) Wert zu und schließen damit die Vertauschung der Werte von `euler` und `pi` ab. Geben Sie auch nach jeder dieser Anweisungen die Werte der drei Variablen auf der Konsole aus, um mit eigenen Augen zu sehen, wie die Vertauschung Schritt für Schritt zustande kommt.

Als nächstes eine kleine Steigerung: Die Werte von ***drei*** statt zwei `double`-Variablen namens `d1`, `d2` und `d3` sollen zyklisch vertauscht werden. Das heißt, der initiale Wert von `d1` soll hinterher in `d2` stehen, der initiale Wert von `d2` in `d3` und der initiale Wert von `d3` in `d1`. Schreiben Sie übungshalber zwei verschiedene Codestücke für diese zyklische Vertauschung, und zwar beide Male so, dass jeweils nur ***eine einzige*** zusätzliche Variable `tmp` neben `d1`, `d2` und `d3` verwendet wird: (i) Vertauschen Sie zuerst, wie oben gesehen, die Werte von `d1` und `d3` miteinander und danach ebenfalls wie oben gesehen die Werte in `d2` und `d3` miteinander. (ii) Realisieren Sie die zyklische Vertauschung mit insgesamt nur vier Anweisungen. Lassen Sie sich in (i) und (ii) analog zu oben nach jeder Anweisung die Werte der vier Variablen auf der Konsole ausgeben.

Machen Sie dasselbe wie mit den zwei Werten oben, aber nun nicht mit zwei Variablen vom primitiven Datentyp `double`, sondern mit zwei Variablen von Klasse `Robot`, die Sie einfach `robot1` und `robot2` nennen können. Diese beiden Variablen lassen Sie auf jeweils ein Roboter-Objekt verweisen, und diese beiden Roboter-Objekte sind auf verschiedenen Feldern platziert. Die Position der Felder, mit welcher Anzahl Münzen und mit welcher Blickrichtung die Roboter initialisiert sind, all das ist egal, nur unterschiedliche Felder sind wichtig. Analog zum Anfang vertauschen Sie nun mit Zuweisungen (also mit „=") die Werte von `robot1` und `robot2` mit einer Hilfsvariablen `tmp`, die ebenfalls vom Typ `Robot` ist. Sie machen aber jetzt noch etwas anderes: Nachdem Sie `tmp` mit einem der beiden Roboter initialisiert haben, rufen Sie über `tmp` Methoden auf, mit denen Sie Zeile und Spalte des Roboters ändern, aber damit Sie weiterhin die Roboterobjekte gut unterscheiden können, sollen die beiden Roboter auch nach dieser Änderung auf unterschiedlichen Feldern stehen. Geben Sie sofort danach und nach den anderen beiden für die Vertauschung notwendigen Zuweisungen jeweils die Koordinaten von `robot1`, `robot2` und `tmp` auf der Konsole aus.

> **Unbewertete Verständnisfrage:**
>
> Passen die Ausgaben zu Ihrem Verständnis von Referenzen und Objekten aus Kapitel 01a, Folien 23-29 der FOP, sowie insbesondere Kapitel 01b, ab Folie 100 der FOP?

Nun sollten Sie nach Bearbeitung der Aufgaben bestens für die folgende Aufgabe zur eigentlichen Hauptschleife vorbereitet sein.

H3: Hilfsmethoden für die Hauptschleife — 10 Punkte

Zunächst beginnen wir mit der Implementation von einigen Hilfsmethoden, die Ihnen die Arbeit in der *Hauptschleife* erleichtern werden.

H3.1: Zählen von Arraykomponenten gleich `null` — 1 Punkt

Implementieren Sie als erstes die Methode `numberOfNullRobots`, welche ein Array vom Typ „Array von `Robot`" namens `allRobots` übergeben bekommt. Die Methode darf ohne Überprüfung davon ausgehen, dass `allRobots` nicht auf `null` verweist und liefert die Anzahl an Komponenten in `allRobots`, für die `allRobots[i] == null` gilt.

[3] Häufig werden Hilfsvariable, die wie hier nur kurzfristig benötigt werden, `tmp` für `temporary` genannt.

H3.2: Drei (pseudo-)zufällige `int`-Werte als Array — 4 Punkte

Nun implementieren Sie die Methode `generateThreeDistinctRandomIndices`. Ziel der Methode ist es ein Array der Länge 3, dessen Komponenten *verschiedene* und (pseudo-)zufällige Werte vom Typ `int` sind, zurückzuliefern. Dazu nutzen Sie für jede Komponente des Arrays die bereits oben in Aufgabe H2 indirekt erwähnte Methode `ThreadLocalRandom.current().nextInt`, der Sie einfach den Parameter `bound` der Methode `generateThreeDistinctRandomIndices` übergeben. `bound` legt hierbei den (*exklusiven*) Maximalwert der (pseudo-)zufällig generierten Zahl zurück.

Unbewertete Verständnisfrage:

Warum sprechen wir hier immer von „pseudo-zufälligen“ Zahlen und nicht einfach von „zufälligen“ Zahlen?

H3.3: Sortierung eines 3-elementigen `int`-Arrays — 2 Punkte

Um nun das aus Aufgabe H3.2 erstellte Array schöner zu gestalten, implementieren Sie in Methode `sortArray` eine kleine Sortierung des Arrays. Für diese Methode darf davon ausgegangen werden, dass das übergebene Array namens `array` nicht `null` ist und genau drei verschiedene `int`-Werte beinhaltet. Es soll nach Aufruf der Methode `array[0]` $<$ `array[1]` $<$ `array[2]` gelten.

Verbindliche Anforderung:

Arbeiten Sie lediglich mit if-Anweisungen.

H3.4: Vertauschen von Indizes in einem Array — 1 Punkt

Des Weiteren implementieren Sie die Methode `swapRobots`. Diese erhält zwei Arrays namens `indices` und `robots` als Parameter, Ersteres vom Typen `int` und Zweiteres vom Typen `Robot`. Es darf hier davon ausgegangen werden, dass `indices` immer drei `int`-Werte einkapselt, die im Indexbereich von `robots` liegen und dass `robots` auch mindestens drei Komponenten beinhaltet (jedoch nicht zwangsweise ! $=$`null`).

Ziel der Methode ist es nun drei Roboter im `robots`-Array zu vertauschen. Seien dafür $i < j < k$ die drei übergebenen Indizes. Dann soll der Roboter an Index i in `robots` hinterher an Stelle j, der Roboter an Stelle j hinterher an Stelle k und zuletzt der Roboter an Stelle k hinterher an Stelle i sein, also im Grunde eine Vertauschung analog zu Aufgabe H2.2.

H3.5: Reduzieren eines Arrays — 2 Punkte

Als letzte Hilfsmethode implementieren Sie `reduceRobotArray`. Dieser Methode werden ein Parameter `robots` vom Typen „Array von `Robot`“, sowie ein Parameter `reduceBy` vom primitiven Datentyp `int` übergeben. Die Methode darf dabei davon ausgehen, dass `robots` nicht auf `null` verweist und dass `reduceBy` die Anzahl an `null`-Komponenten des Arrays `robots` beinhaltet. Es gilt also sowohl `reduceBy` ≥ 0, als auch `reduceBy` $\leq$ `robots.length`.

Die Funktionalität ist nun die Folgende: Sie richten mit Hilfe einer Variable `reducedArray`[4] vom Typ „Array von `Robot`“ ein Array ein, dessen Länge um genau `reduceBy` *kürzer* ist, als das Array, auf das `robots` verweist. Auf jedes `Robot`-Objekt, auf das in `robots` verwiesen wird, soll auch in `reducedArray` verwiesen werden und auch in der

[4] Auch hier ist der Name natürlich Ihnen überlassen.

selben Reihenfolge, wie im aktualen Parameter übergebenen Array. Rückgabe der Methode ist nun das neu erstellte Array `reducedArray`.

H4: Die Hauptschleife 6 Punkte

Auch hier bietet es sich an, dass Sie zunächst ihre Implementationen aus H3 mittels der Public-Tests und eigenen Aufrufen mit Konsolenausgaben testen.

Es sind nun alle Vorbereitungen für die Hauptschleife und das „Marschieren der Roboter“[5] getroffen. Sie finden in der Vorlage auch die Methode `letRobotsMarch`. In dieser implementieren Sie eine `while`-Schleife, die im Folgenden die *Hauptschleife* genannt wird. Für die Fortsetzungsbedingung (siehe Kapitel 01b der FOP) der Schleife, wird in jedem Durchlauf der Hauptschleife als erstes getestet, ob es noch mindestens eine Arraykomponente im der Methode als Parameter übergebenen Array `allRobots` ungleich `null` existiert.

Falls die Fortsetzungsbedingung erfüllt ist, wird als nächstes in einem Durchlauf durch die Hauptschleife mit jedem Roboter das Folgende gemacht, und zwar in der Reihenfolge nach aufsteigenden Indizes, das heißt, erst `allRobots[0]`, dann `allRobots[1]` usw.: Zuerst legt `allRobots[`i`]` auf seinem momentanen Feld genau eine Münze ab. Dann gibt es eine Fallunterscheidung: Falls der Roboter `allRobots[`i`]` durch einen Vorwärtsschritt die `World` verlassen würde, wird einfach `allRobots[`i`]` auf `null` gesetzt; andernfalls macht `allRobots[`i`]` einen einzelnen Vorwärtsschritt, das heißt, die Roboter marschieren schrittweise nach rechts aus der `World` hinaus.

Bevor Sie mit dem Folgenden weitermachen, testen Sie erst einmal diesen Zwischenstand per Augenschein und überprüfen, ob sich die Roboter wie erwartet verhalten.

Als nächstes passiert in einem Durchlauf durch die Hauptschleife aber noch etwas: Falls mindestens 3 Komponenten in `allRobots` existieren, wird die Reihenfolge der Roboter in `allRobots` zufällig verändert, indem drei verschiedene(!) Indizes von `allRobots` zufällig ausgewählt und die Inhalte dieser drei Komponenten von `allRobots` zyklisch vertauscht werden wie in H2.2: Seien i, j und k die drei gewählten Indizes, und zwar so sortiert, das $i < j < k$ ist. Dann soll der Inhalt von Komponente i hinterher in j sein, der von j hinterher in k und der von k hinterher in i.

Nun sollten Sie erneut Ihre Implementation testen. Das heißt, Sie überprüfen, ob die Menge der Roboter in der `World` durch diese zyklische Vertauschungen auch in unterschiedlicher Reihenfolge die Münzen ablegt, bzw. einen Vorwärtsschritt ausführt.

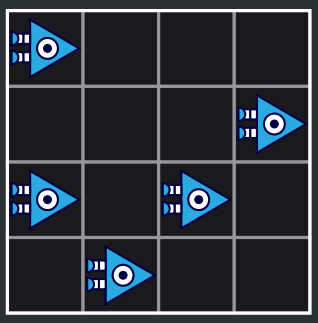
Schließlich passiert in einem Durchlauf durch die Hauptschleife noch eine letzte Aktion: Sei $\ell \geq 0$ die Anzahl Komponenten von `allRobots`, die gleich `null` sind. Falls $\ell \geq 3$, aktualisieren sie das Array `allRobots`, sodass es keine Komponenten gleich `null` mehr beinhaltet.

Verbindliche Anforderungen:

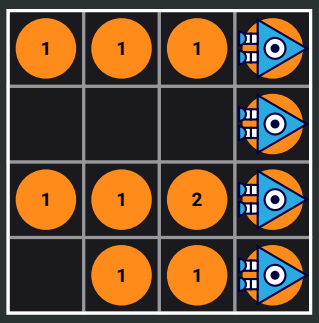
- Kein Roboter ändert jemals seine Richtung. Das bedeutet, dass alle Roboter die `World` nach rechts verlassen und auch zwischendrin niemals die Richtung eines Roboters verändert wird.
- Verwenden sie an geeigneten Stellen die in Aufgabe H3 implementieren Hilfsmethoden.

Nutzen Sie nun noch einmal die zur Verfügung gestellten Public-Tests und eigene kleine Tests, um die Korrektheit Ihrer Methode `letRobotsMarch` zu überprüfen. Falls Sie Ihre Implementation wieder per Augenschein testen, finden Sie hier eine Veranschaulichung, wie die `World` vor und nach Durchlauf durch die Hauptschleife aussieht:

[5] Wie bereits auf Übungsblatt 01 erwähnt, ist solch eine vermenschlichte Formulierung natürlich nicht wirklich korrekt.



(a) Besetzung der `World` *bevor* die Hauptschleife durchgelaufen ist.



(b) Besetzung der `World` *nachdem* die Hauptschleife durchgelaufen ist.

Abbildung 1: Veranschaulichung der `World` *bevor* und *nachdem* die Hauptschleife durchgelaufen ist