Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 02

Prof. Karsten Weihe

Übungsblattbetreuer:	Tim-Michael Krieg
Wintersemester 22/23	v1.0-SNAPSHOT
Themen:	Arrays in Java mit Hilfe von FopBot
Relevante Foliensätze:	01a-01d
Abgabe der Hausübung:	11.11.2022 bis 23:50 Uhr

Hausübung 02 — Gesamt: -6 Punkte

<Übungstitel>

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* in unserem 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 in der Vorlage relevanten Verzeichnisse sind `src/main/java/h02` und `src/test/java/h02`.

Einleitung

Hinweise:

- Wir verfolgen „Abschreiben“ und andere Arten von Täuschungsversuchen. Disziplinarische Maßnahmen treffen nicht nur die, die abschreiben, sondern auch die, die abschreiben lassen. Allerdings werden wir nicht unbedingt zeitnah prüfen, das heißt, es hat noch nichts zu bedeuten, wenn Sie erst einmal nichts von uns hören.
- Screenshots der `World` mit Ihren Robotern darin können Sie unbedenklich mit anderen teilen und in Foren posten, um zu klären, ob Ihr Programm das tut, was es soll. Quelltext und übersetzten Quelltext dürfen Sie selbstverständlich nicht teilen, posten oder sonstwie weitergeben außer an die Ausrichter und Tutoren der FOP 21/22!
- Wenn in den ***Verbindliche Anforderungen*** gefordert wird, dass mit bestimmten Methoden gearbeitet werden soll, werden diese einzeln getestet und müssen folglich korrekt implementiert sein, damit die volle Punktzahl für die Aufgabe erhalten werden kann. Ändern Sie insbesondere nichts an der Parameterliste und dem Rückgabetyp der einzelnen Methoden! Bei Unklarheiten zu Parametern, Rückgabe oder der Funktionalität einer Methode, schauen Sie sich die Dokumentation der Methode in der Code-Vorlage an.
- In der `main`-Methode der Klasse `Main.java` finden sie bereits einige Zeilen Code, die eine FopBot-`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: Initialisierungen vor der Hauptschleife ?? Punkte

In der Klasse `Main.java` finden Sie schon eine Methode `initializeRobotsPattern`. 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 `initializeRobotsPattern` darf ohne Nachprüfung davon ausgehen, dass weder `pattern` selbst, noch eine Komponente des Arrayobjektes, auf das `pattern` verweist, gleich `null` sind. Weiter darf `initializeRobotsPattern` ohne Nachprüfung davon ausgehen, dass `numberOfColumns` und `numberOfRows` tatsächlich die Anzahl der Spalten bzw. Zeilen der `World` sind, wie es die Namen suggerieren.

Konkret erstellen Sie in dieser Methode ein Arrayobjekt namens „`allRobots`“ vom Typ `Robot`, befüllen dieses mit Robotern und liefern es mittels `return` zurück. Um zunächst die Größe des zu befüllenden Arrayobjektes zu bestimmen, müssen wir erst festlegen, wie genau das Arrayobjekt besetzt werden soll:

Ein Paar (x, y) von natürlichen Zahlen[1] heißt im Weiteren *zu besetzen gemäß* der `World` und des `pattern`, wenn die folgenden fünf Bedingungen erfüllt sind:

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

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

[1] In der FOP gilt natürlich $0 \in \mathbb{N}$.

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

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

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

Hinweis:

Es fällt Ihnen wahrscheinlich auf, dass Voraussetzung (e) (und auch bereits die beiden vorhergehenden) etwas sonderbar aussieht. Warum sollte `pattern[`y`][`x`]` gleich `true` sein und nicht `pattern[`x`][`y`]`? Dieser Umstand liegt an der „Unverträglichkeit“ von mathematischen Koordinaten mit Besetzungen von „Arrays von Arrays“. Da in der Mathematik eben zunächst der „Spaltenindex“ (also x) behandelt wird, dieser allerdings natürlich in einem „Array von Arrays“ an zweiter Stelle folgt, ergibt sich diese unintuitive Darstellung.

Für jedes gemäß `World` und `pattern` zu besetzende Paar (x, y) soll das Arrayobjekt, auf das `allRobots` verweist, genau eine Komponente haben. Darüber hinaus soll es *keine* weiteren Komponenten haben. Sie erstellen also zunächst ein Attribut „`numberOfRobots`“ und initialisieren dieses mit 0. Um nun die Größe des Arrayobjektes zu bestimmen, zählen Sie in zwei ineinander geschachtelten `for`-Schleifen alle *zu besetzenden* Paare und richten auf dieser Basis das zurückzuliefernde Arrayobjekt `allRobots` ein.

Das Befüllen des Arrayobjektes verläuft analog: Sie verwenden wieder zwei `for`-Schleifen und erstellen zu jedem *zu besetzenden* Paar (x, y) ein `Robot`-Objekt. Dieser Roboter soll in Spalte x und Zeile y stehen. Dabei sollen *alle* Roboter die Richtung `RIGHT` haben. Darüber hinaus soll ein Roboter in Spalte x über genau `numberOfColumns`$-x$ Münzen verfügen.

Bevor Sie mit der nächsten Aufgabe weitermachen, testen Sie zunächst, ob Ihre Initialisierung von `allRobots` korrekt ist.

Dazu haben wir Ihnen 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.

Darüber hinaus sollten Sie natürlich auch durch geeignete Konsolenausgaben sicherstellen, dass auch die Anzahl der Münzen eines jeden Roboters korrekt initialisiert ist. Dazu bietet es sich an nicht nur die Münzzahl eines Roboters, sondern auch seine x-Koordinate, als auch die Spaltenanzahl der `World` ausgeben zu lassen, um direkt die Korrektheit der Münzzahl zu überprüfen.

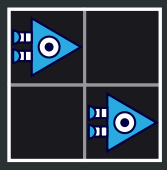
Hinweise:

- Ihnen wird dabei wahrscheinlich auffallen, dass die `World` nicht wie zu erwarten *oben links* beginnend besetzt wird, sondern eben *unten links*. Wie Sie bereits in Kapitel 01a, Folien 12-19 der FOP kennengelernt haben, beginnt die Nummerierung in der `World` unten links mit $(0, 0)$, wie eben in der Mathematik üblich (Stichwort Koordinatensysteme). Machen Sie sich also keine Sorgen, wenn Ihre `World` nur unten links besetzt wird, obwohl Sie die Roboter eigentlich oben links erwartet hätten.
- Sollten Sie auf die Idee kommen ein eigenes Pattern mittels eines „Array von Arrays von `boolean`“ zu testen, müssen Sie auf den oben erläuterten Umstand achten. Dieser ist für unsere Verwendung eines „Array von Array von `boolean`“ leider etwas hinderlich, da hierbei die Nummerierung der „Zeilen“ und „Spalten“ im Grunde *oben links* beginnt. Um nun also ein eigenes Pattern zu testen, müssen Sie bei der Erstellung eines „Array von Arrays von `boolean`“ daran denken.

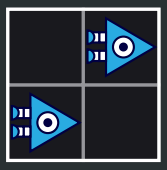
  Im Folgenden werden wir das Ganze etwas verdeutlichen:

Beispielarray

```
1 boolean[][] array = {{true, false},
2                      {false, true}};
```



(a) Dies wäre die zu erwartende Besetzung der `World` nach Verwendung des Arrays „`array`“ aus dem obigen Codeblock in der Methode `initializeRobotsPattern`.



(b) Dies ist die tatsächliche Besetzung der `World` nach Verwendung des Arrays „`array`“ aus dem obigen Codeblock in der Methode `initializeRobotsPattern`.

Abbildung 1: Anhand dieses Beispiels sehen Sie den Unterschied zwischen *zu erwartender* Besetzung links und *tatsächlicher* Besetzung rechts.

H2: Vorübungen für das Weitere ?? Punkte

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 ?? Punkte

Hinweis:

Die folgenden Anweisungen werden zunächst alle in der Methode `initializeRobotsPattern` implementiert.

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 diese Anweisungen untereinander, aber bei jedem Kompiliervorgang sind alle diese Anweisungen bis auf eine 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.java`.

H2.2: Vertauschen von Werten von Variablen ?? Punkte

Sie haben nun in H2.1 verschiedene Fehlermeldungen kennengelernt, die Ihnen beim Gebrauch von Arrayobjekten über den Weg laufen könnten. In dieser Aufgabe widmen wir uns nun der Vorübungen für die H3:

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,[2] 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.

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

H3: Die Hauptschleife ?? Punkte

Zunächst beginnen wir mit der Implementation von einigen Hilfsmethoden, die Ihnen die Arbeit in der Hauptschleife erleichtern werden. Alle Methoden finden sich bereits in der Vorlage in der Klasse `Main.java`, müssen also nicht selber erstellt werden.

H3.1: Arraykomponenten gleich `null` ?? Punkte

Implementieren Sie als erstes die Methode `numberOfNullRobots`, welche ein Arrayobjekt `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.

H3.2: Drei (pseudo-)zufällige `int`-Werte ?? 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` übergeben. `bound` legt hierbei den (*exklusiven*) Maximalwert der (pseudo-)zufällig generierten Zahl zurück.

Verbindliche Anforderung:

Nutzen Sie für die Erstellung der drei Werte lediglich *eine* `while`-Schleife.

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 ?? 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 nicht `null` ist und genau drei `int`-Werte beinhaltet. Es soll nach Aufruf der Methode für das übergebene Array `array[0]` $<$ `array[1]` $<$ `array[2]` gelten.

Verbindliche Anforderung:

Arbeiten Sie lediglich mit if-Anweisungen.

H3.4: Vertauschen von Robotern ?? Punkte

Als letzte kleine Hilfsmethode implementieren Sie `swapRobots`. Diese erhält zwei Arrays als Parameter, eines vom Typen `int` und eines vom Typen `Robot`. Es darf hier davon ausgegangen werden, dass `indices` immer drei `int`-Werte einkapselt, die im Indexbereich von `allRobots` liegen und dass `allRobots` auch mindestens drei Komponenten beinhaltet (jedoch nicht zwangsweise $!=$`null`).

Ziel der Methode ist es nun drei Roboter im `allRobots`-Array zu vertauschen. Seien dafür $i < j < k$ die drei übergebenen Indizes. Dann soll der Roboter an Index i in `allRobots` 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 analog zu Aufgabe H2.2.

H3.5: Die Hauptschleife ?? Punkte

Auch an dieser Stelle bietet es sich an, die bisher implementierten Methoden mit einfachen Konsolenausgaben zu testen.

Nun sind alle Vorbereitungen für das eigentliche Thema der dritten Aufgabe getroffen. Sie finden in `Main.java` auch eine Methode `letAllRobotsGo`. In der Methode implementieren Sie eine `while`-Schleife, die im Folgenden die *Hauptschleife* genannt wird. Für die Fortsetzungsbedingung der Schleife, wird in jedem Durchlauf der Hauptschleife als erstes getestet, ob es noch mindestens eine Arraykomponente im Arrayobjekt `allRobots` ungleich `null` gibt.

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 wandern schrittweise nach rechts aus der `World` hinaus.

Bevor Sie mit dem Folgenden weitermachen, testen Sie erst einmal diesen Zwischenstand Ihrer Implementation von `letAllRobotsGo` auf die übliche Art: per Augenschein, ob die Roboter in der `World` sich wie erwartet verhalten, sowie geeignete Konsolenausgaben.

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.

Bevor Sie mit dem Folgenden weitermachen, testen Sie per Augenschein, ob die Menge der Roboter in der `World` durch diese zyklischen Vertauschungen unverändert bleibt, wie es ja sein sollte. Das ist natürlich kein hundertprozentig „wasserdichter“ Korrektheitstest für die zyklische Vertauschung, sollte aber schon alle potentiellen Programmierfehler aufdecken können, die weder durch den Compiler noch durch Inaugenscheinnahme des Codes („Code Review“) sofort gefunden werden.

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 $\ell \geq 3$, richten Sie mit Hilfe eines Verweises `tmp` vom Typ „Array von `Robot`“ ein Arrayobjekt ein, dessen Länge um genau ℓ kürzer ist als das Arrayobjekt, auf das `allRobots` verweist. Auf jedes `Robot`-Objekt, auf das in `allRobots` verwiesen wird, soll auch in `tmp` verwiesen werden. Schließlich lassen Sie noch `allRobots` mittels „=“ auf das Arrayobjekt verweisen, auf das `tmp` verweist.

Testen Sie nochmals per Augenschein, ob die Robotermenge durch solche Arrayersetzungen unverändert bleibt, wie es ja sein sollte.

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 den vorhergehenden Aufgaben implementierten Methoden.