Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 02

TECHNISCHE UNIVERSITÄT DARMSTADT

Rückfragen zu diesem Übungsblatt vorzugsweise im moodle-Forum zu diesem Blatt!

Wintersemester 21/22	v1.0
Themen:	Arrays in Java mit Hilfe von FopBot
Relevante Foliensätze:	01d
Abgabe der Hausübung:	12.11.2021 bis 23:50 Uhr

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!
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.

Wichtiger Hinweis: 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.

H1 Initialisierungen vor der Hauptschleife — 4 Punkte

Richten Sie in Datei `Main.java` in dem mit „`// Hier programmieren`“ bezeichneten Bereich ein Array `allRobots` mit Komponententyp `Robot` ein. Implementieren Sie dazu die Methode `initializeRobots(int cols, int rows)` und rufen Sie sie geeignet auf. Die Methode bekommt die Anzahl der Spalten und Zeilen der `World` übergeben und gibt ein Array vom Komponententyp `Robot` zurück. Die Anzahl Komponenten des Arrays `allRobots` soll wie folgt definiert sein: Falls das Minimum aus Anzahl Zeilen und Anzahl Spalten der `World` gerade ist, soll das die Anzahl der Komponenten von `allRobots` sein, ansonsten soll die Anzahl Komponenten von `allRobots` genau um 1 kleiner als dieses Minimum sein. Jede Komponente von `allRobots` soll auf ein Roboterobjekt verweisen. Die Roboter in Array `allRobots` bilden initial eine Diagonale in der World, und zwar so, dass der Roboter an Index `i` von `allRobots` in Zeile `i` und Spalte `i` der `World` steht. Die Roboter in der oberen Hälfte der Indizes von `allRobots` schauen initial nach links, die anderen initial nach rechts (vgl. Abbildung 1). Jeder Roboter in `allRobots` hat initial 1.000 Münzen.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie wie auf Blatt 01, ob Ihre Initialisierung von `allRobots` soweit korrekt ist. Für die Überprüfung der initialen Koordinaten und der initialen Richtung bieten sich nicht unbedingt Konsolenausgaben an, sondern Ihre Prüfung per Augenschein, ob der Inhalt im von `FopBot` eröffneten Fenster – also die `World` mit den Robotern darin – korrekt ist. Für die Überprüfung der Münzzahlen bietet es sich an, nur die Information auf der Konsole auszugeben, dass tatsächlich alle Münzzahlen korrekt sind oder – falls doch nicht alle korrekt sind – alle falschen gefundenen Münzzahlen mit den zugehörigen Indizes in `allRobots`.

Richten Sie nun ein Array `paces` mit Komponententyp `int` ein, indem Sie die Methode `initializePaces(Robot[] allRobots)` implementieren und geeignet aufrufen. Das von `initializeRobots` zurückgegebene Array hat genau so viele Komponenten wie das Array `allRobots`. Jede Komponente dieses Arrays wird initialisiert als eine ganzzahlige Zufallszahl im Intervall $1 \ldots 5$ (alle fünf Zahlen gleich wahrscheinlich).

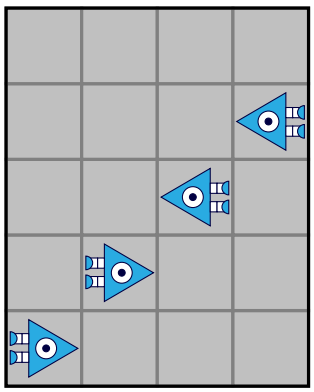

Abbildung 1: Initialer Zustand der World mit Größe 4×5

.

***Verbindliche Anforderungen*:**

- Arbeiten Sie mit den Methoden `initializeRobots(int cols, int rows)` und `initializePaces(Robot[] allRobots)`.
- Zur Festlegung der Länge von `paces` verwenden Sie die Länge von `allRobots`, nicht die Zeilen- und Spaltenzahl der `World`.
- Für jede Komponente von `paces` lassen Sie nur eine einzige Zufallszahl mit `ThreadLocalRandom.current()` (vgl. Übungsblatt 00) berechnen. Es ist also insbesondere nicht erlaubt, solange Zufallszahlen zu generieren, bis eine davon einmal zufällig im Bereich $1 \ldots 5$ ist, und diese dann zu nehmen.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie nun die Länge von `paces` und ob tatsächlich alle Komponenten von `paces` im Bereich $1 \ldots 5$ sind. Es bietet sich wieder an, nur die Information auszugeben, dass alles korrekt ist, oder die inkorrekten Daten zu spezifizieren.

H2 Vorübungen — 0 Punkte

H2.1 Fehlermeldungen besser verstehen — 0 Punkte

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

Greifen Sie nun analog auf verschiedene Indizes von `paces` und `allRobots` zu, die ***nicht*** im Indexbereich der beiden Arrays liegen, mindestens eine negative Zahl, die Arraylänge sowie eine Zahl größer Arraylänge. Das heiß konkret, Sie fügen eine einzelne Schreibabweisung ein, die eine Arraykomponente an einem Index außerhalb des Indexbereichs des Arrays ausgeben soll, dort setzen Sie die verschiedenen Werte nacheinander ein, kompilieren jeweils neu und lassen das Programm laufen. Der Quelltext sollte durch den Compiler gehen, aber der Prozess sollte mit einer Fehlermeldung abgebrochen werden. Macht die Fehlermeldung für Sie Sinn?

Am Ende entfernen Sie allen Java-Code, den Sie speziell für H2.1 eingefügt haben, wieder aus `Main.java`.

H2.2 Vertauschen von Werten von Variablen — 0 Punkte

(a) Richten Sie zwei Variable `d123e` und `e321d` von Typ `double` ein und initialisieren Sie sie mit den Werten 3.14 und 2.71. Lassen Sie sich die Werte von `d123e` und `e321d` wie üblich auf der Konsole ausgeben. Richten Sie nun eine weitere `double`-Variable `tmp` ein,[1] mit deren Hilfe Sie die Werte von `d123e` und `e321d` vertauschen. Und zwar weisen Sie dazu als erstes `tmp` den Wert von `d123e` zu (also „`tmp = d123e;`“). Machen Sie sich klar, dass nun der initiale Wert von `d123e` in `tmp` „gerettet“ ist und daher – zweite Anweisung – in `d123e` überschrieben werden kann, ohne verlorenzugehen. Als dritte und letzte Anweisung weisen Sie `e321d` den passenden (welchen also?) Wert zu und schließen damit die Vertauschung der Werte von `d123e` und `e321d` ab. Geben Sie nach jeder der drei Anweisungen die Werte der drei Variablen auf der Konsole aus, um mit eigenen Augen zu sehen, wie die Vertauschung Schritt für Schritt zustande kommt.

(b) 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 in (a) die Werte von `d1` und `d3` miteinander und danach wie in (a) 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 (a) nach jeder Anweisung die Werte der vier Variablen auf der Konsole ausgeben.

(c) Machen Sie dasselbe wie in (a), 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 – auf welchen Feldern genau, mit welcher Anzahl Münzen und mit welcher Blickrichtung, all das ist egal, nur unterschiedliche Felder sind wichtig. Analog zu (a) 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. Passen die Ausgaben zu Ihrem Verständnis von Referenzen und Objekten aus Kapitel 01a, Folien 23-29?

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

H3 Die Hauptschleife — 13 Punkte

Als nächstes schreiben Sie in `Main.java` eine `while`-Schleife. Diese Schleife soll im Folgenden die *Hauptschleife* genannt werden. Abbruchbedingung ist, dass die Länge von `allRobots` gleich 0 ist. In H3.3 werden wir sehen, wie die Hauptschleife zu dieser Abbruchbedingung kommt.

(1 Punkt)

H3.1 Vorwärtsgehen der Roboter — 4 Punkte

In jedem Durchlauf der Hauptschleife wird jeder Roboter in `allRobots` um so viele Schritte vorwärtsbewegt, wie am selben Index in `paces` steht. Implementieren Sie dazu die Methode `moveForward(Robot[] allRobots, int[] paces)` und rufen Sie sie geeignet auf. `moveForward` soll alle Bewegungen durchführen, die im aktuellen Durchlauf der Hauptschleife nötig sind. Beachten Sie: Jedesmal, wenn ein Vorwärtsschritt um ein Feld dazu führen würde, dass der Roboter die `World` verlässt, dreht er zuerst solange um 90 Grad nach links, bis ein Vorwärtsschritt nicht mehr die `World` verlassen würde, und erst dann macht der Roboter den Vorwärtsschritt.

***Verbindliche Anforderungen*:**

- Arbeiten Sie mit der Methode `moveForward(Robot[] allRobots, int[] paces)`.
- Dass alle Roboter sich in einem Durchlauf der Hauptschleife jeweils bewegen, soll durch eine `for`-Schleife über die Indizes von `allRobots` realisiert sein. Die Vorwärtsschritte eines Roboters `allRobots[i]` sollen darin durch eine `for`-Schleife realisiert sein, die `paces[i]` viele Durchläufe hat und in jedem Durchlauf den Roboter um einen Schritt vorwärtsbewegt (ggf. Vorwärtsbewegung erst nach Drehung wie im vorangehenden Absatz beschrieben).

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie wieder wie gehabt, bevor Sie mit H3.2 weitermachen. Zu Überprüfungszwecken können sie mittels der Methode `World.setDelay()` die Framerate anpassen.

H3.2 Vertauschen der Schrittweiten — 4 Punkte

Im jedem Durchlauf durch die Hauptschleife, in dem die Länge der beiden Arrays noch mindestens drei ist, passiert aber noch etwas unmittelbar nach den Vorwärtsschritten aus H3.1: Zufällig werden drei unterschiedliche(!) Indizes von paces ausgewählt. Dies soll in der Methode `generateThreeDistinctInts(Robot[] allRobots)` implementiert werden, welche ein `int` Array mit den drei distinkten Indizes zurückgibt. Nun sollen die drei Indizes (im Folgenden als `i1`, `i2` und `i3` bezeichnet) so vertauscht werden, dass hinterher $\texttt{i1} < \texttt{i2}$ und $\texttt{i2} < \texttt{i3}$ gilt. Dann vertauschen Sie die Werte der Komponenten von `paces` an diesen drei Indizes, und zwar so, dass $\texttt{paces[i1]} \leq \texttt{paces[i2]}$ und $\texttt{paces[i2]} \leq \texttt{paces[i3]}$ ist. Implementieren und benutzen Sie dafür die Methode `swapPaces(int[] paces, int i1, int i2, int i3)`. Diese erhält die drei ursprünglichen distinkten Zufallszahlen `i1`, `i2` und `i3` als Parameter, sortiert diese (wobei die Methode `orderThreeInts(int i1, int i2, int i3)` benutzt werden soll) und vertauscht die Einträge in `paces` anschließend wie oben beschrieben.

***Verbindliche Anforderungen*:**

- Arbeiten Sie mit den Methoden `generateThreeDistinctInts(Robot[] allRobots)`, `orderThreeInts(int i1, int i2, int i3)` und `swapPaces(int[] paces, int i1, int i2, int i3)`.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie wieder wie gehabt, bevor Sie mit H3.3 weitermachen.

H3.3 Roboterzahl schrittweise reduzieren — 4 Punkte

Oben in der Datei `Main.java` finden Sie eine `byte`-Konstante namens `NUMBER_OF_STEPS_BETWEEN_REDUCTIONS`. Von jeweils `NUMBER_OF_STEPS_BETWEEN_REDUCTIONS` aufeinanderfolgenden Duchläufen soll der letzte ein ***reduzierender Durchlauf*** genannt werden.

***Beispiel*:** `NUMBER_OF_STEPS_BETWEEN_REDUCTIONS` habe den Wert 10. Die ersten 9 Durchläufe durch die Hauptschleife sind nicht reduzierend, der 10-te ist reduzierend, die Durchläufe Nr. $11 - 19$ sind wieder nicht reduzierend, der 20-te ist wieder reduzierend usw.

In jedem reduzierenden Durchlauf sollen die beiden Arrays jeweils um eine Komponente verkleinert werden (Erinnerung: Abbruchbedingung der Hauptschleife ist, dass die beiden Arrays Länge 0 haben).

Konkret sollen in jedem dieser reduzierenden Durchläufe die beiden Arrayobjekte, auf die `allRobots` und `paces` verweisen, ersetzt werden durch jeweils ein Arrayobjekt, das eine Komponente weniger hat. Dazu wählen Sie am Ende jedes reduzierenden Durchlaufs durch die Hauptschleife zufällig einen Index aus der momentanen Indexmenge von `allRobots` und `paces`. Mit Hilfe der Methoden `reduceRobots(Robot[] allRobots, int deleteIndex)` und `reducePaces(int[] paces, int deleteIndex)` richten Sie für beide jeweils ein neues Arrayobjekt mit einer Komponente weniger ein und kopieren alle Komponenten außer der zufällig ausgewählten in die beiden neuen Arrayobjekte, aber so, dass in beiden Arrayobjekten die Reihenfolge der Komponenten dieselbe bleibt. Lassen Sie `allRobots` und `paces` auf die beiden durch die Aufrufe von `reduceRobots` und `reducePaces` erhaltenen *reduzierten* Arrays verweisen. [2]

***Verbindliche Anforderungen*:**

- Arbeiten Sie mit den Methoden `reduceRobots(Robot[] allRobots, int deleteIndex)` und `reducePaces(int[] paces, int deleteIndex)`.

***Tests zur eigenen Kontrolle* (0 Punkte):** Machen Sie analog zu Blatt 01 den finalen Test.

[2] Die beiden Arrayobjekte, auf die `allRobots` und `paces` bis dahin noch verwiesen hatten, sind dann nicht mehr aus Ihrem Programm heraus erreichbar, aber das muss Sie nicht kümmern. Um diese beiden Objekte kümmert sich das Laufzeitsystem, konkret der Garbage Collector, bei Interesse: siehe Kapitel 03b, Folien 214 ff.