Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 03

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Karsten Weihe

Wintersemester 23/24	v1.1
Themen:	Klassen mit FOPBot
Relevante Foliensätze:	01e-01f
Abgabe der Hausübung:	17.11.2023 bis 23:50 Uhr

Hausübung 03 — Gesamt: 32 Punkte

Multi-Family Robots & Synchronizers

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/h03` und ggf. `src/test/java/h03`.

Verbindliche Anforderung: Dokumentieren Ihres Quelltexts

Alle von Ihnen deklarierten Klassen, Interfaces, Enumerationen und Methoden (inklusive Konstruktoren), die nicht `private` sind, *müssen* für diese Hausübung mittels JavaDoc in Englisch oder alternativ Deutsch dokumentiert werden. Für jede korrekte Deklaration ohne Dokumentation verlieren Sie jeweils einen Punkt.

Beachten Sie die Seite *Hausübungen → Dokumentieren von Quelltext* im Studierenden-Guide.

Auf diesem Übungsblatt werden Aufzählungen überschneidender Namen verkürzt, indem nur disjunkte Teile von Namen innerhalb geschweifter Klammern aufgezählt werden und bei vorangegangener Aufzählungen einsetzbare Elemente durch `*` ersetzt werden. Beispiel: `set{X,Y}For{A,B}` ist die Abkürzung für `setXForA`, `setXForB`, `setYForA` und `setYForB`. `set*For*` ist die Abkürzung für `set{X,Y}For{A,B}`.

Verwenden Sie in Ihrem Quelltext 1:1 die auf diesem Übungsblatt gewählten Identifier! Andernfalls wird die jeweilige Aufgabe nicht automatisiert bewertet.

Roboter-Familien

Mit diesem Übungsblatt lernen Sie verschiedene Roboter-Familien kennen, welche sich in ihrem Aussehen unterscheiden: Neben der Ihnen aus der Vorlesung bekannten Roboter-Familie der *Triangle Robots* kommt nun auch die Roboter-Familie der *Square Robots* zum Einsatz. Die *Square Robots* präsentieren sich in einer Vielzahl von Farben und bereichern somit die Roboterwelt.

Um einen Roboter einer anderen Familie zu konstruieren, nutzen Sie einen beliebigen Konstruktor von `Robot` mit einem Parameter des Typs `RobotFamily`. Bei `RobotFamily` handelt es sich wie bei `Direction` um eine Enumeration – mit dem Unterschied, dass `RobotFamily` anstelle der Blickrichtungen die Roboter-Familien aufzählt.

Beispiel:

Einen Roboter der Familie `RobotFamily.SQUARE_RED` können Sie beispielsweise wie folgt konstruieren:

```
1 Robot robot = new Robot(4, 2, Direction.UP, 69, RobotFamily.SQUARE_RED);
```

H1: Multi-Family Robots ?? Punkte

In dieser Aufgabe erstellen Sie eine Klasse für sogenannte Multi-Family Robots. Als *Multi-Family Robot* bezeichnen wir einen Roboter, welcher beim Laufen seine Roboter-Familie tauscht.

Erstellen Sie zuerst innerhalb des Package `h03` ein Package `robots`.

H1.1: First Class ?? Punkte

Erstellen Sie nun innerhalb des Package `h03.robots` eine `public`-Klasse `MultiFamilyRobot`, welche direkt von `Robot` abgeleitet ist.

H1.2: Robot under Construction ?? Punkte

Implementieren Sie nun in der Klasse `MultiFamilyRobot` einen `public`-Konstruktor mit folgenden Eigenschaften:

Die ersten beiden formalen Parameter des Konstruktors `x` und `y` sind vom Typ `int` und geben die Position des zu konstruierenden *Multi-Family Robot* an, wobei der erste aktuale Parameter gleich der Position auf der x-Achse und der zweite Parameter gleich der Position auf der y-Achse ist. Der dritte formale Parameter des Konstruktors `families` ist vom Typ „Array von `RobotFamily`“. Als aktuale Parameter sind nur Positionen innerhalb der Welt und Arrays (also nicht `null`), welche nicht die Länge 0 besitzen erlaubt. Andere aktuale Parameter müssen *nicht* beachtet werden. Der Konstruktor von `MultiFamilyRobot` ruft den Konstruktor der Basisklasse `Robot` auf, dessen ersten beiden formalen Parameter vom Typ `int` und dessen dritter formaler Parameter vom Typ `RobotFamily` ist. Der erste bzw. zweite aktuale Parameter für den Aufruf des Konstruktors der Basisklasse `Robot` ist gleich `x` bzw. `y`. Der dritte aktuale Parameter für den Aufruf des Konstruktors der Basisklasse `Robot` ist gleich dem in `families` an Index 0 referenzierten Objekt. Weiter soll `families` einer gleichnamigen Konstante zugewiesen werden, welche den gleichen Typen wie der dritte formale Parameter hat. Setzen Sie die Zugriffsrechte des Attributs `families` so, dass nur ein Zugriff von der Klasse `MultiFamilyRobot` aus möglich ist.

H1.3: Familientausch ?? Punkte

Implementieren Sie in `MultiFamilyRobot` eine parameter- und rückgabelose `public`-Methode `exchange`: Mit jedem Aufruf von `exchange` soll die *Robot Family* auf die Familie gesetzt werden, welche in `families` am jeweils nächsten Index referenziert wird. Wenn der aktuelle Index der letzte Index von `families` ist, wird 0 als nächster Index gewählt.

Verbindliche Anforderung:

Die Verwendung von bedingten Operationen (`if-else` und ternärer Operator) ist nicht erlaubt. Verwenden Sie stattdessen den Restwert-Operator (aus der Vorlesung als *Modulo-Operator* bekannt).

Hinweise:

Die Familie eines Roboters kann mittels der Methode `setRobotFamily(RobotFamily)` in `Robot` gesetzt werden. Richten Sie ein geeignetes Attribut ein, welches den aktuellen Index verwaltet.

Ob mit dem Aufruf von `exchange` die Familie tatsächlich getauscht wird, ist davon abhängig, ob die aktuelle Familie gleich der in `families` am nächsten Index referenzierten Familie ist. Wir sagen der Einfachheit halber, dass mit einem Aufruf von `exchange` ein Tausch der Familie stattfindet.

H1.4: *Nur noch ein Schritt bis zur neuen Familie.* ?? Punkte

Aus der Klasse `Robot` kennen Sie die Methode `move`. Überschreiben Sie die Methode `move` in der Klasse `MultiFamilyRobot` so, dass *nach* der Ausführung der in der Basisklasse `Robot` gegebenen Funktionalität weiter die Familie des Roboters getauscht wird, also die Methode `exchange` aus der Aufgabe H1.3 aufgerufen wird.

H1.5: *Ich möchte aber bei meiner Familie bleiben!* ?? Punkte

Mit dem Überschreiben von `move` besteht keine Möglichkeit mehr, einen Multi-Family Robot bewegen zu lassen, ohne dass dabei die Familie des Roboters getauscht wird. Das soll nicht so sein!

Überladen Sie die Methode `move` nun, indem Sie in der Klasse `MultiFamilyRobot` eine rückgabelose `public`-Methode `move` implementieren, welche einen formalen Parameter `exchange` des Typs `boolean` hat. `move(boolean)` ruft in jedem Fall die Methode `move` aus `Robot` auf. Wenn der aktuale Parameter `exchange` gleich `true` ist, soll danach die Familie des Roboters getauscht werden – also die Methode `exchange` aus H1.3 aufgerufen werden.

Unbewertete Verständnisfrage:

In unserem Fall lassen wir die Methode `move()` die Methode `exchange()` direkt aufrufen. Warum könnte es sinnvoller sein, statt der Methode `exchange()` die Methode `move(boolean)` aufrufen zu lassen?

H2: Spezielle Multi-Family Robots ?? Punkte

In dieser Aufgabe erstellen Sie zwei Klassen für spezielle Varianten von *Multi-Family Robots*: Zum einen den *RGB Robots*, zum anderen den *Chess Board Robots*.

H2.1: Rhythms of RGB ?? Punkte

Ein *RGB Robot* zeichnet sich dadurch aus, dass dieser zwischen der drei Roboter-Familien *Square Robots* in den drei Grundfarben rot, grün und blau tauscht.

Erstellen Sie zuerst im Package `h03.robots` eine `public`-Klasse `RGBRobot`, welche direkt von der Klasse `MultiFamilyRobot` abgeleitet ist.

Implementieren Sie nun in der Klasse `RGBRobot` einen `public`-Konstruktor, dessen ersten beiden formalen Parameter `x` und `y` vom Typ `int` und dritter formaler Parameter `inverted` vom Typ `boolean` ist. Die ersten beiden aktualen Parameter sind gleich den ersten beiden aktualen Parametern für den Aufruf des Konstruktors der Basisklasse. Der dritte aktuale Parameter für den Aufruf des Konstruktors der Basisklasse wird wie folgt gewählt: Wenn `inverted` gleich `false` ist, wird als aktualer Parameter für `families` ein Array mit den Konstanten (in dieser Reihenfolge) `SQUARE_RED`, `SQUARE_GREEN` und `SQUARE_BLUE` der Enumeration `RobotFamily` verwendet. Im anderen Fall wird als aktualer Parameter für `families` ein Array mit denselben Konstanten in invertierter Reihenfolge verwendet.

Implementieren Sie zuletzt in der Klasse `RGBRobot` eine parameter- und rückgabelose `public`-Methode `testRGB`, mittels welcher alle Familien des Roboters „ausprobiert“ werden können, indem die Familie des Roboters dreimal getauscht wird, also dreimal die Methode `exchange` aufgerufen wird.

H2.2: Robo Chess — ?? Punkte

Ein *Chess Board Robot* tauscht nur zwischen zwei Roboter-Familien. Die Reihenfolge seiner Roboter-Familien wählt ein Chess Board Robot so, dass sich die initiale Roboter-Familie bei Feldern mit gerader und ungerader Summe aus x- und y-Koordinate unterscheidet. Wenn wir eine Welt mit füllen, sehen wir also ein Schachbrettmuster!

Erstellen Sie zuerst im Package `h03.robots` eine `public`-Klasse `ChessBoardRobot`, welche ebenfalls direkt von `MultiFamilyRobot` abgeleitet ist.

Implementieren Sie nun in der Klasse `ChessBoardRobot` einen `public`-Konstruktor, dessen ersten beiden formalen Parameter `x` und `y` vom Typ `int` und letzten beiden formalen Parameter `initial{Even,Odd}` vom Typ `RobotFamily` sind. Die ersten beiden aktualen Parameter sind wieder gleich den ersten beiden aktualen Parametern für den Aufruf des Konstruktors der Basisklasse. Der dritte aktuale Parameter für den Aufruf des Konstruktors der Basisklasse ist ein Array von `RobotFamily`, welches die aktualen Parameter `initial{Even,Odd}` folgendermaßen referenziert. Wenn die Summe der aktualen Parameter `x` und `y` *gerade* ist, wird `initialEven` an Index 0 referenziert. Andernfalls wird `initialOdd` an Index 0 referenziert.

Implementieren Sie in `ChessBoardRobot` einen weiteren `public`-Konstruktor, dessen ersten beiden formalen Parameter `x` und `y` wieder vom Typ `int` sind. Im Gegensatz zum anderen Konstruktor hat dieser Konstruktor keinen dritten und vierten formalen Parameter. Die beiden aktualen Parameter sind wieder gleich den ersten beiden Parametern für den Aufruf des ersten Konstruktors. Als dritter bzw. vierter aktualer Parameter wird `SQUARE_BLACK` bzw. `SQUARE_WHITE` der Enumeration `RobotFamily` verwendet.

H3: Sync Star — ?? Punkte

In dieser Aufgabe lösen Sie sich davon, `Robot` als Basisklasse zu verwenden und schreiben eine eigene Klasse für sogenannte *Robot Synchronizer*. Eine *Robot Synchronizer* dient dazu, mehrere Roboter auf eine festgelegte Position bewegen und in eine festgelegte Richtung zu drehen.

H3.1: *Bevor wir syncen ...* — ?? Punkte

Erstellen Sie zuerst im Package `h03` eine `public`-Klasse `RobotSynchronizer`.

H3.2: *Wer synct mit?* — ?? Punkte

Implementieren Sie nun in der Klasse `RobotSynchronizer` einen `public`-Konstruktor, welcher einen formalen Parameter `robots` vom Typ „Array von `Robot`“ besitzt. Der Konstruktor weist `robots` einer gleichnamigen Konstante zu, welche weiter den gleichen statischen Typ besitzt. Setzen Sie die Zugriffsrechte von `robots` so, dass nur ein Zugriff von der Klasse `RobotSynchronizer` aus möglich ist.

H3.3: *Was syncen wir?* ?? Punkte

Implementieren Sie in der Klasse `RobotSynchronizer` die drei `public`-Methoden `set{X,Y,Direction}`, wobei `set{x,y}` jeweils einen formalen Parameter `x` bzw. `y` des Typs `int` und `setDirection` einen formalen Parameter `direction` des Typs `Direction` hat.

Weiter existiert in der Klasse `RobotSynchronizer` für jeden formalen Parameter ein gleichnamiges Attribut, wobei `x` und `y` mit `-1` und `direction` mit `null` initialisiert wird. Setzen Sie die Zugriffsrechte der drei Attribute so, dass nur ein Zugriff von der Klasse `RobotSynchronizer` aus möglich ist.

Wir bezeichnen einen aktualen Parameter für `x`, `y` bzw. `direction` als *valide*, wenn `x` bzw. `y` sich innerhalb der Welt befinden bzw. `direction` ungleich `null` ist. Jede dieser drei Methoden weist ihren aktualen Parameter dem jeweiligen Attribut der Klasse `RobotSynchronizer` zu, sofern der aktuale Parameter valide ist. Andernfalls bleibt der bisherige Wert erhalten.

> **Erinnerung:**
>
> Die Größe der Welt kann mittels `World.getWidth()` und `World.getHeight()` abgerufen werden.

H3.4: *I like to sync it, sync it!* ?? Punkte

Implementieren Sie in `RobotSynchronizer` die `public`-Methode `sync`, welche jeden in `robots` referenzierten Roboter an die mittels `x` und `y` gegebene Position bewegt und in die mittels `direction` gegebene Richtung ausrichtet. Sie können davon ausgehen, dass in der gegebenen Welt *keine* Wände enthalten sind.

Wenn `x`, `y` bzw. `direction` invalide ist, soll stattdessen der jeweilige ursprüngliche Wert des jeweiligen Roboters verwendet werden.

> **Verbindliche Anforderung:**
>
> Die Anzahl an von den Robotern tatsächlich durchgeführten Bewegungen und Drehungen zum Erreichen des Endzustands darf die hierfür minimal notwendige Anzahl nicht überschreiten.

H4: Testen ?? Punkte

Implementieren Sie die Methode `main` der Klasse `Main` folgendermaßen:

Erstellen Sie zuerst *mindestens* einen Multi-Family Robot, wobei Sie eine beliebige Position und ein beliebiges Array von Roboter-Familien mit *mindestens* drei *verschiedenen* Roboter-Familien verwenden. Bewegen Sie diesen Roboter mindestens einmal mehr als dieser Roboter-Familien hat.

Erstellen Sie nun *mindestens* zwei RGB Robots, wobei Sie jeweils eine beliebige Position und unterschiedliche Werte für `inverted` verwenden. Bewegen Sie diese beiden Roboter jeweils mindestens viermal.

Erstellen Sie nun *mindestens* vier Chess Board Robots so, dass diese eine rechteckige Fläche ausfüllen und oberhalb der obersten Chess Board Robots *mindestens* zwei Felder frei sind. Verwenden Sie den zweiparametrigen Konstruktor. Bewegen Sie diese Roboter jeweils mindestens zweimal.

Erstellen Sie einen Robot Synchronizer, wobei Sie beliebige zuvor erstelle Robots verwenden. Setzen Sie für den Robot Synchronizer eine beliebige Position und Richtung. Rufen Sie dann die Methode `sync` auf.

Führen Sie das Programm nun aus.