Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 03

Prof. Karsten Weihe

Wintersemester 23/24	v1.0
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 & 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 implementieren Sie eine Klasse für sogenannte *Multi Family Robots*. Als *Multi Family Robot* bezeichnen wir einen Roboter, welcher beim Laufen seine Roboter-Familie tauscht.

H1.1: First Class ?? Punkte

Erstellen Sie zunächst das *Package* `h03.robots` und dann innerhalb von `h03.robots` eine `public`-Klasse `MultiFamilyRobot`, welche direkt von `Robot` abgeleitet ist.

> **Hinweis:**
>
> `h03.robots` bezeichnet das *Package* `robots` innerhalb des *Package* `h03`.

H1.2: Robot under Construction ?? Punkte

Implementieren Sie in `MultiFamilyRobot` einen Konstruktor mit folgenden Eigenschaften:

Die ersten beiden Parameter des Konstruktors `x` und `y` sind vom Typ `int` und geben die Position des zu konstruierenden *Multi Family Robot* an, wobei der erste Parameter gleich der Position auf der x-Achse und der zweite Parameter gleich der Position auf der y-Achse ist. Der dritte Parameter des Konstruktors `families` ist vom Typ „Array von `RobotFamily`“. Als aktuale Parameter sind nur Positionen innerhalb der Welt und Arrays, welche nicht die Länge 0 besitzen und nicht `null` referenzieren, erlaubt. Andere aktuale Parameter müssen nicht beachtet werden.

Der Konstruktor von `MultiFamilyRobot` ruft den Konstruktor der Basisklasse `Robot` auf, dessen ersten beiden Parameter vom Typ `int` sind, und dessen dritter 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 Objektkonstante zugewiesen werden, welche den gleichen Typen wie der Parameter hat. Setzen Sie die Zugriffsrechte des Objektattributs `families` so, dass nur ein Zugriff von der Klasse, dem Package und allen von `MultiFamilyRobot` abgeleiteten Klassen möglich ist.

H1.3: Familientausch ?? Punkte

Implementieren Sie in `MultiFamilyRobot` eine parameter- und rückgabelose Objektmethode `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* bekannt).

Hinweise:

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

Anmerkung:

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: Ein Schritt zur neuen Familie ?? Punkte

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

H1.5: *Ich möchte 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 `move` nun, indem Sie in `MultiFamilyRobot` eine rückgabelose `public`-Objektmethode `move` implementieren, welche einen Parameter `exchange` des Typs `boolean` besitzt. `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: Referenzimplementation ?? Punkte

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

H2.1: Rhythms of RGB ?? Punkte

Zuerst implementieren Sie die Klasse für *RGB Robots*: Ein *Multi Family Robot* dieser Art zeichnet sich dadurch aus, dass dieser zwischen Familien von *Square Robots* in den drei Grundfarben rot, grün und blau tauscht.

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

Implementieren Sie in `RGBRobot` einen Konstruktor, dessen ersten beiden Parameter `x` und `y` vom Typ `int` und dritter 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) `RobotFamily.SQUARE_RED`, `RobotFamily.SQUARE_GREEN` und `RobotFamily.SQUARE_BLUE` verwendet. Im anderen Fall wird als aktualer Parameter für `families` ein Array mit denselben Konstanten in umgekehrter Reihenfolge verwendet.

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

H2.2: Robo Chess ?? Punkte

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

Implementieren Sie in `ChessBoardRobot` einen Konstruktor, dessen ersten beiden Parameter `x` und `y` vom Typ `int` und letzten beiden 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 `initial{Even,Odd}` referenziert. Wenn die Summe von `x` und `y` gerade ist, wird `initialEven` an Index 0 referenziert. Andernfalls wird `initialOdd` an Index 0 referenziert.

Implementieren Sie in `ChessBoardRobot` einen weiteren Konstruktor, dessen ersten beiden Parameter `x` und `y` wieder vom Typ `int` sind. Die beiden aktualen Parameter sind wieder gleich den ersten beiden Parametern für den Aufruf des ersten Konstruktors. Als dritter bzw. vierter aktualer Parameter wird `RobotFamily.SQUARE_BLACK` bzw. `RobotFamily.SQUARE_WHITE` 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, eine gegebene Menge von Robotern mit einer festgelegten Position und Richtung zu synchronisieren.

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

Erstellen Sie zuerst in `h03` eine `public`-Klasse `RobotSynchronizer`.

H3.2: *Wer synct mit?* ?? Punkte

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

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

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

Jede der dieser drei Objektmethoden weist ihren aktualen Parameter einem Objektattribut zu, welches jeweils den Namen und den Typ des jeweiligen Parameters besitzt. Wenn ein aktualer Parameter invalide ist (`x` oder `y` außerhalb der Welt oder `direction` gleich `null`), soll der derzeitig gesetzte Wert beibehalten werden. Setzen Sie die Zugriffsrechte der drei Objektattribute so, dass nur ein Zugriff von der Klasse möglich ist.

Die beiden Objektattribute `x` und `y` sollen initial jeweils den invaliden Wert `-1` haben.

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

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

Implementieren Sie in `RobotSynchronizer` die `public`-Objektmethode `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` nicht gesetzt wurde, 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 minimal notwendige Anzahl nicht überschreiten.