Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 04

Prof. Karsten Weihe

Wintersemester 23/24	v1.0
Themen:	<Themen>
Relevante Foliensätze:	bis 01g
Abgabe der Hausübung:	XX.XX.202X bis 23:50 Uhr

Hausübung 04
<Übungstitel>

Gesamt: 32 Punkte

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/h04` und ggf. `src/test/java/h04`.

Einleitung

H1: *Move Strategies* ?? Punkte

In dieser Aufgabe beschäftigen Sie sich mit *Move Strategies* für Roboter, welche im weiteren Verlauf dieses Übungsblatts Verwendung finden. Als *Move Strategy* bezeichnen wir eine Strategie, mit welcher sich ein Roboter von seiner aktuellen Position aus zu einer gegebenen Position bewegt.

Erstellen Sie zunächst im Package `h04` ein Package `strategy`.

H1.1: Interface für *Move Strategies* ?? Punkte

Erstellen Sie im Package `strategy` ein `public`-Interface `MoveStrategy`.

Deklarieren Sie im Interface `MoveStrategy` eine rückgabelose Methode `start`, welche einen ersten formalen Parameter vom Typ `Robot` und einen zweiten formalen Parameter vom Typ `Field` hat. Die Methode `start` wird in Implementationen des Interface `MoveStrategy` aufgerufen, um einen gegebenen Roboter mittels der jeweiligen Move Strategy zu einer gegebenen Position zu bewegen.

H1.2: Interface für *Move Strategies* mit *Counters* ?? Punkte

Erstellen Sie im Package `strategy` ein `public`-Interface `MoveStrategyWithCounter`, welche das Interface `MoveStrategy` erweitert.

Das Interface `MoveCounter` deklariert eine parameterlose Methode `getMoveCount`, welche einen Wert des Typs `int` liefert. Hierbei handelt es sich um die Anzahl der beim letzten Aufruf der Methode `start` mittels der Methode `move` der Klasse `Robot` durchgeführten Schritte.

H1.3: *Move by Teleport* ?? Punkte

Nun implementieren Sie die erste Move Strategy *Move by Teleport*.

Erstellen Sie hierzu im Package `strategy` eine `public`-Klasse `MoveByTeleport`, welche das in H1.1 erstellte Interface `MoveStrategy` implementiert.

Implementieren Sie die Methode `start` in der Klasse `MoveByTeleport` so, dass der gegebene Roboter auf das gegebene Field „teleportiert“ wird, indem die Position des Roboters mittels der Methoden `setX` und `setY` *oder* `setField` der Klasse `Robot` gesetzt wird.

> **Verbindliche Anforderung:**
>
> Die Methode `move` der Klasse `Robot` darf *nicht* verwendet werden.

H1.4: *Move by Walk* ?? Punkte

Nun implementieren Sie die zweite Move Strategy *Move by Walk*.

Erstellen Sie hierzu im Package `strategy` eine `public`-Klasse `MoveByWalk`, welche das in Aufgabe H1.2 erstellte Interface `MoveStrategyWithCounter` implementiert.

Implementieren Sie die Methode `start` aus dem Interface `MoveStrategy` in der Klasse `MoveByWalk` so, dass der gegebene Roboter auf das gegebene Field *bewegt* wird, indem die Methoden `move` und `turnLeft` aus der Klasse `Robot` aufgerufen werden.

Weiter implementieren Sie die Methode `getMoveCount` aus dem Interface `MoveStrategyWithCounter` in der Klasse `MoveByWalk` folgendermaßen: Wenn `start` mindestens einmal aufgerufen wurde, liefert die Methode `getMoveCount` die Anzahl an Aufrufen der Methode `move` aus der Klasse `Robot` im *letzten* Aufruf von der Methode `start`. Andernfalls liefert die Methode `getMoveCount` den Wert `-1`.

> **Verbindliche Anforderung:**
>
> Die Methoden `setX`, `setY` und `setField` der Klasse `Robot` dürfen *nicht* verwendet werden.

H2: *Field Selectors* und *Field Selection Listeners* ?? Punkte

In dieser Aufgabe beschäftigen Sie sich mit *Field Selectors* und *Field Selection Listeners*. Ein *Field Selector* ist ein Element, welches der Auswahl eines Field (*Field Selection*) in der World dient. Ein *Field Selection Listener* ist ein Element, welches bei einem Field Selector *registriert* wird und daraufhin über Field Selections dieses Field Selector *benachrichtigt* wird.

Erstellen Sie zunächst im Package `h04` ein Package `h04.selection`.

H2.1: Interface für *Field Selection Listeners* ?? Punkte

Erstellen Sie zuerst im Package `h04.selection` ein `public`-Interface `FieldSelectionListener`.

Deklarieren Sie im Interface `FieldSelectionListener` eine rückgabelose Methode `onFieldSelection`, welche einen formalen Parameter des Typs `Field` hat. Die Methode `onFieldSelection` wird von einem Field Selector aufgerufen, wenn mittels dieses Field Selector ein Field ausgewählt wurde, wobei als aktualer Parameter ebendieses Field verwendet wird.

H2.2: Interface für *Field Selectors* ?? Punkte

Erstellen Sie nun im Package `h04.selection` ein `public`-Interface `FieldSelector`.

Deklarieren Sie im Interface `FieldSelector` eine rückgabelose Methode `setFieldSelectionListener`, welche einen formalen Parameter des Typs `FieldSelectionListener` hat. Die Methode `setFieldSelectionListener` wird von einem Field Selection Listener aufgerufen, um diesen bei dem jeweiligen Field Selector zu registrieren.

H2.3: *Die Eingabe mit der Maus ...* ?? Punkte

Erstellen Sie eine Klasse `MouseFieldSelector`, die das in der Aufgabe H2.2 erstellte Interface `FieldSelector` sowie das im Package `fopbot.FieldClickListener` gegebene Interface `FieldClickListener` implementiert.

> **Anmerkung:**
>
> Im Interface `FieldClickListener` ist als einzige Methode die Methode `onKeyPress` deklariert, welche einen formalen Parameter des Typs `FieldClickEvent` besitzt.
>
> Wenn ein Objekt einer Klasse, welche das Interface `FieldClickListener` implementiert, beim zentralen *Input Handler* registriert ist, wird die Methode `onFieldClick` bei jedem *Field Click Event* (Klick mit der Tastatur auf ein Field in der *World*) aufgerufen.
>
> Dabei liefert der aktuale Parameter der Methode `onFieldClick` Informationen über das Field Click Event: Die Methode `getField` der Klasse `FieldClickEvent` liefert das Field, welches angeklickt wurde.
>
> Ein Objekt `object` einer Klasse, welche das Interface `FieldClickListener` implementiert, kann dem zentralen Input Handler folgendermaßen hinzugefügt werden:
>
> ```
> 1  InputHandler.addFieldClickListener(object);
> ```

Implementieren Sie die Methode `setFieldSelectionListener` aus dem Interface `FieldSelector` so, dass der aktuale Parameter innerhalb jeder anderen Methode der Klasse `MouseFieldSelector` abrufbar ist.

Nun implementieren Sie die Methode `onFieldClick` aus dem Interface `FieldClickListener` so, dass wenn die Methode `onFieldClick` *nicht* zum ersten Mal aufgerufen wird *und* das Field *dasselbe* wie beim letzten Aufruf der Methode `onFieldClick` ist, das *Field Click Event* als *Field Selection* gewertet wird und damit die Methode `onFieldSelection` des registrierten Field Selection Listener mit dem Field des Field Click Event aufgerufen wird.

Zuletzt implementieren Sie in der Klasse `MouseFieldSelector` einen parameterlosen `public`-Konstruktor, welcher das aktuelle Objekt dem zentralen *Input Handler* als Field Click Listener hinzufügt.

H2.4: *... und der Tastatur* ?? Punkte

Erstellen Sie im Package `h04.selection` eine Klasse `KeyFieldSelector`, welche das in der Aufgabe H2.2 erstellte Interface `FieldSelector` sowie das im Package `fopbot` gegebene Interface `KeyPressListener` implementiert.

> **Anmerkung:**
>
> Im Interface `KeyPressListener` ist als einzige Methode die Methode `onKeyPress` deklariert, welche einen formalen Parameter des Typs `KeyPressEvent` besitzt.
>
> Wenn ein Objekt einer Klasse, welche das Interface `KeyPressListener` implementiert, beim zentralen *Input Handler* registriert ist, wird die Methode `onKeyPress` bei jedem *Key Press Event* (Druck einer Taste auf der Tastatur innerhalb einer *World*) aufgerufen.
>
> Dabei liefert der aktuale Parameter der Methode `onKeyPress` Informationen über das *Key Press Event*: Die Methode `getKey` der Klasse `KeyPressEvent` liefert den *Key* (die gedrückte Taste auf der Tastatur in Form einer Konstanten der Enumeration `Key`). Die Methode `getWorld` der Klasse `KeyPressEvent` liefert die *World*, innerhalb welcher das *Key Press Event* ausgelöst wurde.
>
> Ein Objekt `object` einer Klasse, welche das Interface `KeyPressListener` implementiert, kann dem zentralen *Input Handler* folgendermaßen hinzugefügt werden:
>
> ```
> 1  InputHandler.addKeyPressListener(object);
> ```

Wir bezeichnen ein Field als *markiert*, wenn die Farbe dieses Field auf `Color.RED` gesetzt wurde und als *nicht markiert*, wenn die Farbe dieses Field nicht *oder* auf `null` ist. Weiter bezeichnen wir ein nicht markiertes Field als *entmarkiert*, wenn dieses Field zuvor markiert war. Sie können davon ausgehen, dass maximal ein Field gleichzeitig markiert ist.

Implementieren Sie die Methode `setFieldSelectionListener` aus dem Interface `FieldSelector` analog zur Klasse `MouseFieldSelector` so, dass der aktuale Parameter innerhalb jeder anderen Methode der Klasse `MouseFieldSelector` abrufbar ist.

Implementieren Sie die Methode `onKeyPress` folgendermaßen: Wenn die Methode `onKeyPress` zum ersten Mal aufgerufen wird, wird das Field an Position $(0, 0)$ markiert. Sie können davon ausgehen, dass in diesem Fall kein anderes Field markiert ist. Wenn ein Field markiert ist *und* der Key des Key Press Event gleich der Konstanten `UP`, `LEFT`, `DOWN` oder `RIGHT` der Enumeration `Key` ist, wird das bisher markierte Field entmarkiert und das *erste* Field oberhalb, links, unterhalb bzw. rechts von dem zuvor markierten Field markiert. Sollte sich an der jeweiligen Stelle kein weiteres Field befinden, wird stattdessen das *letzte* Field in der jeweiligen entgegengesetzen Richtung markiert. Wenn ein Field markiert ist *und* der Key des Key Press Event gleich der Konstanten `SPACE` der Enumeration `Key` ist, wird die Methode `onFieldSelection` des Field Selection Listener mit dem markierten Field als aktualen Parameter aufgerufen.

Zuletzt implementieren Sie in der Klasse `KeyFieldSelector` einen parameterlosen `public`-Konstruktor, welcher das aktuelle Objekt dem zentralen Input Handler als Key Press Listener hinzufügt.

> **Hinweis:**
>
> Die Farbe eines Field kann mittels eines Aufrufs der Methode `setColor` der Klasse `Field` gesetzt werden, wobei als aktualer Parameter die jeweilige Farbe in Form eines Objekts der Klasse `Color` oder `null` verwendet wird.

H3: Mover ?? Punkte

Zuerst erstellen Sie im Package `h04` ein Package `h04.moveable`.

H3.1: *Robot Mover* ?? Punkte

Zuerst erstellen Sie im Package `h04.moveable` eine `public`-Klasse `RobotMover`, welche das Interface `FieldSelectionListener` implementiert.

Nun implementieren Sie in der Klasse `RobotMover` einen `public`-Konstruktor, welcher einen formalen Parameter vom Typ `MoveStrategy` hat. Der aktuale Parameter muss innerhalb jeder anderen Methode der Klasse `RobotMover` abrufbar sein.

Nun implementieren Sie in der Klasse `RobotMover` eine rückgabelose `public`-Methode `addRobot`, welche einen formalen Parameter vom Typ `Robot` hat. Jeder aktuale Parameter muss innerhalb jeder anderen Methode der Klasse `RobotMover` abrufbar sein.

Zuletzt implementieren Sie in der Klasse `RobotMover` die Methode `onFieldSelection` aus dem Interface `FieldSelectionListener` so, dass diese die Methode `start` der dem Konstruktor gegebenen `MoveStrategy` mit *jedem* der Methode `addRobot` gegebenen `Robot` und dem aktualen Parameter der Methode `onFieldSelection` aufruft.

H3.2: *Movable Robot* ?? Punkte

Zuerst erstellen Sie im Package `h04.moveable` eine `public`-Klasse `MoveableRobot`, welche direkt von der Klasse `Robot` abgeleitet ist und das Interface `FieldSelectionListener` implementiert.

Nun implementieren Sie in der Klasse `MoveableRobot` einen `public`-Konstruktor, welcher einen formalen Parameter vom Typ `MoveStrategy` hat und einen beliebigen Konstruktor der Basisklasse aufruft, welcher den Robot an Position $(0, 0)$ platziert. Der aktuale Parameter muss innerhalb jeder anderen Methode der Klasse `MoveableRobot` abrufbar sein.

Zuletzt implementieren Sie in der Klasse `MoveableRobot` die Methode `onFieldSelection` aus dem Interface `Field` so, dass diese die Methode `start` der dem Konstruktor gegebenen `MoveStrategy` mit dem aktuellen Robot aufruft. Wenn die Klasse des dem Konstruktor gegebenen Objekts weiter das Interface `MoveStrategyWithCounter` implementiert, dreht der Roboter nach dem Aufruf von `start` so oft wie dieser Schritte ausgeführt hat.

H4: *It's better together!* ?? Punkte