Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 04

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwurf

Achtung: Dieses Dokument ist ein Entwurf und ist noch nicht zur Bearbeitung/Abgabe freigegeben. Es kann zu Änderungen kommen, die für die Abgabe relevant sind. Es ist möglich, dass sich **alle** Aufgaben noch grundlegend ändern. Es gibt keine Garantie, dass die Aufgaben auch in der endgültigen Version überhaupt noch vorkommen und es wird keine Rücksicht auf bereits abgegebene Lösungen genommen, die nicht die Vorgaben der endgültigen Version erfüllen.

Hausübung 04

Gesamt: 31 Punkte

Roboter mit Senf

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`.

Verbindliche Anforderung: Dokumentieren Ihres Quelltexts

Alle von Ihnen deklarierten Klassen, Interfaces und Methoden (inklusive Konstruktoren) *müssen* für diese Hausübung mittels JavaDoc 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}`.

Wenn die Rede davon ist, dass eine Klasse, ein Interface oder eine Enumeration *erstellt* werden soll, muss für diese Hausübung zunächst die dazugehörige Datei erstellt werden.

Einleitung

In den vorherigen Übungen haben Sie nur mit Klassen gearbeitet, aber in dieser Übung schreiben Sie zusätzlich erste Interfaces sowie Klassen, die diese Interfaces implementieren. Bisher haben Sie auch nur Klassen direkt oder indirekt von `Robot` abgeleitet oder bereits `Robot` als Attribut in einer anderen Klasse verwendet. Jetzt werden Sie erstmals eine Klasse implementieren, die nichts mit `Robot` zu tun hat.

Zuerst haben Sie in Kapitel 01f der FOP gesehen, wie man eine Klasse von einer anderen Klasse ableiten kann. Dann haben Sie in Kapitel 01g der FOP gesehen, wie eine Klasse ein Interface implementiert.

Eine Subklasse, mehrere Interfaces

In dieser Übung erstellen Sie eine Klasse, die sowohl von einer Klasse abgeleitet ist als auch mehrere Interfaces implementiert.

Die zu implementierenden Interfaces werden separiert mittels Komma nach der Basisklasse notiert – und zwar wie gewohnt mit `implements`.

Beispiel:

Eine Klasse `X`, die eine Klasse `Y` erweitert und Interfaces `A`, `B` sowie `C` implementiert, kann wie folgt deklariert werden:

```
public class X extends Y implements A, B, C {
    ...
}
```

H1: Zwei Interfaces ?? Punkte

Das Ziel dieser Aufgabe ist es zwei Interfaces zu schreiben, die in den späteren Aufgaben benötigt werden.

H1.1: Keine Referenz? Ist nicht egal! ?? Punkte

Schreiben Sie ein `public`-Interface mit dem Namen `RobotWithReferenceState`, welches fünf Methoden enthält: Zum einen die Methode `setCurrentStateAsReferenceState` ohne Parameter und ohne Rückgabe. Zum anderen die Methoden `getDiff{X,Y,Direction,NumberOfCoins}` ohne Parameter. Die Methode `getDiffDirection` liefert eine Konstante von `Direction`, die anderen drei Methoden einen Wert vom Typ `int` zurück.

Die Idee dieses Interface ist, über `getDiff*` die jeweilige Differenz zu einem Referenzstatus gesetzt werden kann.

H1.2: Keine Münzen? Ist nicht egal! ?? Punkte

Erstellen Sie zuerst eine Enumeration `CoinType` mit Konstanten `SILVER`, `BRASS` und `COPPER` wie in Kapitel 01e der FOP beschrieben.

Erstellen Sie nun ein `public`-Interface `WithCoinTypes` mit Methoden `{set,get}NumberOfCoinsOfType`:

Die Methode `setNumberOfCoinsOfType` hat einen ersten Parameter vom formalen Typ `CoinType` und einen zweiten Parameter von Typ `int`. Die Methode liefert nichts zurück.

Die Methode `getNumberOfCoinsOfType` hat einen Parameter vom formalen Typ `CoinType` und liefert einen Wert von Typ `int` zurück.

Die Idee dieses ist, dass ein Objekt von `WithCoinTypes` zu jedem dieser drei Münztypen jeweils eine Menge von Münzen dieses Typs verwaltet. Genauer: Durch die Methode `setNumberOfCoinsOfType` soll die Anzahl der Münzen für den durch den Parameter `CoinType` spezifizierten Münztyp gesetzt werden und durch die Methode `getNumberOfCoinsOfType` soll die Anzahl der Münzen des spezifizierten Münztyps zurückgegeben werden.

> **Unbewertete Verständnisfragen:**
>
> Probieren Sie einmal aus, was der Compiler dazu sagt, wenn Sie versuchen, `WithCoinTypes` zu variieren. Deklarieren Sie `setNumberOfCoinsOfType` von `WithCoinTypes` versuchsweise `private`. Als nächstes versuchen Sie für `getNumberOfCoinsOfType` einen Methodenrumpf (Anweisungen in geschweiften Klammern) zu schreiben und ein `private`-Attribut `silverCoins` vom Typ `int` in das Interface einzufügen. Als letztes versuchen Sie in Klasse `Main` ein Objekt von `WithCoinTypes` mit Operator `new` zu erzeugen. Vergessen Sie nicht, alle diese Änderungen rückgängig zu machen, bevor Sie weitermachen.

H2: Implementierende Klassen ?? Punkte

In den folgenden Aufgaben werden Sie Klassen schreiben, die die Interfaces aus den vorherigen Aufgaben implementieren. Dabei werden Sie zuerst eine Klasse schreiben, die an Roboter angelehnt ist und das Interface `WithCoinTypes` implementiert. Sie soll es ermöglichen Roboter zu erstellen, die nicht nur einen, sondern drei Münztypen (Silver, Brass und Copper) verwalten. In den darauffolgenden Aufgaben werden Sie angelehnt an diese Klasse einmal eine Roboterklasse schreiben, die anhand von Attributen einen Referenzstatus speichert, und einmal eine Roboterklasse schreiben, die anhand eines Objekts einen Referenzstatus speichert. Beide Klassen implementieren entsprechend das Interface `RobotWithReferenceState`. Schließlich werden Sie eine Klasse schreiben, die nicht mehr an Roboter angelehnt ist, sondern nur Münztypen verwaltet.

H2.1: Keine Scheine? Ist mir egal! ?? Punkte

Erstellen Sie eine `public`-Klasse `RobotWithCoinTypes`, die direkt aus der von Ihnen aus der Vorlesung bekannten Klasse `Robot` abgeleitet ist. Weiter soll diese Klasse das Interface `WithCoinTypes` aus H1.2 implementieren.

`RobotWithCoinTypes` hat drei `private`-Attribute `numberOf{Silver,Brass,Copper}Coins` von Typ `int`.

Erstellen Sie in `RobotWithCoinTypes` einen `public`-Konstruktor: Dieser soll die gleichen vier formalen Parameter in der gleichen Reihenfolge wie der entsprechende Konstruktor von `Robot` haben. Den vierten Parameter von `RobotWithCoinTypes` (also `numberOfCoins`) ersetzen Sie im nächsten Schritt durch drei Parameter `numberOf{Silver,Brass,Copper}Coins` vom formalen Typ `int`. Dieser Konstruktor ruft wie üblich mit `super` den Konstruktor seiner direkten Basisklasse auf – und zwar mit seinen eigenen ersten drei aktuale Parameterwerten

für x-Koordinate, y-Koordinate und Richtung. Als vierter aktualer Parameterwert wird die Summe aus den einzelnen Werten für die drei Münztypen verwendet. Als nächstes werden mit den drei Münzzahlen die drei entsprechenden Münzzahlattribute initialisiert.

Zuerst wird `setNumberOfCoins` von `Robot` in `RobotWithCoinTypes` überschrieben. `setNumberOfCoins` von `Robot` bietet aber keine Differenzierung nach Münztypen, sodass nicht klar ist, welche Art von Münzen gemeint ist. Als Teil der Logik von `RobotWithCoinTypes` legen wir hiermit folgendes fest: Falls die Zahl der Münzen im aktualen Parameter negativ ist, soll die Methode `setNumberOfCoins` von `Robot` mit dem Wert des aktualen Parameters aufgerufen werden. Im umgekehrten Fall, dass die Gesamtzahl Münzen positiv oder null ist, wird nur die Anzahl Kupfermünzen verändert. Damit das Attribut `numberOfCoins` von `Robot` sich konsistent ändert, muss `setNumberOfCoins` von `Robot` mit einem entsprechenden Wert aufgerufen werden.

Nun gibt es zwei Implementationen von `setNumberOfCoins`: eine in `Robot` und eine in `RobotWithCoinTypes`. Wie Sie in Kapitel Kapitel 01g der FOP gesehen haben, rufen Sie in Methoden von `RobotWithCoinTypes` die erste Implementation mit `super.setNumberOfCoins` auf, die zweite nur mit `setNumberOfCoins` ohne `super`.

In Klasse `RobotWithCoinTypes` implementieren Sie die Methoden `{set,get}NumberOfCoinsOfType` von `WithCoinTypes` auf Basis der drei Attribute `numberOf{Silver,Brass,Copper}Coins`:

`getNumberOfCoinsOfType` soll die Anzahl der Münzen des gegebenen Münztyps zurückgeben und die Methode `setNumberOfCoinsOfType` soll die Anzahl der Münzen für den durch den Parameter `CoinType` spezifizierten Münztyp setzen. Falls der zweite aktuale Parameterwert von `setNumberOfCoinsOfType` negativ ist, soll die Methode `setNumberOfCoins` von `Robot` aufgerufen werden.

> **Anmerkung:**
>
> In Kapitel 05 der FOP und in späteren Hausübungen werden wir sehen, wie man in Java mit einem Fehlerfall wie dem, dass die Anzahl der Münzen in `setNumberOfCoinsOfType` negativ ist, umgehen sollte: durch Wurf einer geeigneten *Exception*. In unserem Fall leiten wir das Problem an die Methode der Oberklasse weiter, die die entsprechende Exception wirft.

Damit das Attribut `numberOfCoins` von `Robot` konsistent bleibt, muss auch in `setNumberOfCoinsOfType` die Methode `setNumberOfCoins` von `Robot` mit einem entsprechenden Wert aufgerufen werden. Die Methoden `{put,pick}Coin` überschreiben Sie konsistent mit der oben formulierten Logik von `setNumberOfCoins`. Für `pickCoin` heißt das: Die aufgenommene Münze wird als Kupfermünze interpretiert und dann die entsprechende Methode von `Robot` aufgerufen. Bei `putCoin` haben wir dasselbe Problem wie bei der Methode `setNumberOfCoins`: Es gibt keine Differenzierung zwischen verschiedenen Münztypen. Wir legen dementsprechend folgendes fest: Falls die Gesamtzahl an Münzen durch einen Aufruf von `putCoin` zu verringern ist, soll als erstes die Anzahl Kupfermünzen verringert werden. Falls sie nicht mehr vorhanden sind, wird die Anzahl Messingmünzen verringert. Nur wenn beides nicht vorhanden ist, wird die Anzahl Silbermünzen verringert. Falls keine Münzen vorhanden sind, wird die Anzahl von keiner Münzart verringert. Als letztes wird die Methode `putCoin` von `Robot` aufgerufen [1].

H2.2: Keine Geld? Ist nicht egal! ?? Punkte

Erstellen Sie eine `public`-Klasse `RobotWithCoinTypesAndRefStateOne`, die direkt von `RobotWithCoinTypes` abgeleitet ist und das Interface `RobotWithReferenceState` aus H1.1 implementiert.

Die Klasse hat vier `private`-Objektattribute: `ref{X,Y,Direction,NumberOfCoins}`. Bis auf `refDirection` sind alle genannten Attribute vom Typ `int` – `refDirection` ist vom Typ `Direction`.

Fügen Sie in `RobotWithCoinTypesAndRefStateOne` einen `public`-Konstruktor ein. Er soll dieselben formalen sechs Parameter in derselben Reihenfolge wie der entsprechende Konstruktor von `RobotWithCoinTypes` haben. Dieser Konstruktor ruft wie üblich mit super den Konstruktor seiner direkten Basisklasse auf, und zwar mit seinen

[1] Diese Methode wird für Sie die Fehlermeldung generieren, falls die Anzahl der Silbermünzen nicht ausreicht.

aktualen Parameterwerten für x-Koordinate, für y-Koordinate, für die Richtung und für die einzelnen Werte für die drei Münztypen. Danach werden im Konstruktor die Werte für x-Koordinate, y-Koordinate, Richtung und Summe aus den einzelnen Werten für die drei Münztypen zur Initialisierung der vier `ref`-Attribute verwendet. Das bedeutet der initiale Status des Objektes nach seiner Konstruktion ist also auch der initiale Referenzstatus des Objektes.

Methode `setCurrentStateAsReferenceState` von Klasse `RobotWithCoinTypesAndRefStateOne` setzt die vier `ref`-Attribute auf die momentanen Werte von `Robot`, wie sie durch die zugehörigen `get`-Methoden von `Robot` zurückgeliefert werden.

Methoden `getDiff{X,Y,NumberOfCoins}` von Klasse `RobotWithCoinTypesAndRefState1` liefern die Differenz aus der Rückgabe der jeweiligen `get`-Methode von `Robot` minus dem Wert des entsprechenden `ref`-Attributs von `RobotWithCoinTypesAndRefStateOne` zurück. Die Rückgaben der Methoden zeigen also auf, wie weit der Roboter momentan in positiver oder negativer Richtung von seinem Referenzstatus entfernt ist. Es ist also kein Betrag der Differenz, sondern die vorzeichenbehaftete Differenz selbst.

Bei `getDiffNumberOfDirections` ist die Rückgabe wie folgt: `UP`, falls momentane Richtung und Richtung im Referenzstatus identisch sind. `LEFT`, falls die momentane Richtung um 90^o im Gegenuhrzeigersinn gegenüber der Richtung im Referenzstatus gedreht ist. `DOWN` bei 180^o und `RIGHT` im verbliebenen vierten Fall.

Hinweis:

Schauen Sie sich die Methode `ordinal` von Enumerationen an.

Unbewertete Verständnisfragen:

Deklarieren Sie eine der in `RobotWithCoinTypesAndRefStateOne` implementierten Methoden

(i) als `private` statt `public` und

(ii) streichen sie `public` auch einmal ersatzlos.

In beiden Fällen sollte das Compilen mit einer Fehlermeldung abbrechen. Kommentieren Sie eine ganze Methode in `RobotWithCoinTypesAndRefStateOne` einmal aus (zum Beispiel `getDiffNumberOfCoins`). Auch jetzt sollte das Compilen mit einer Fehlermeldung abbrechen. Erscheinen Ihnen diese Fehlermeldungen sinnvoll?

Vergessen Sie nicht, alle Fehler wieder rückgängig zu machen!

H2.3: Roboter in Roboter? Ist mir egal! ?? Punkte

Erstellen Sie eine `public`-Klasse `RobotWithCoinTypesAndRefStateTwo`, die mit folgendem Unterschied analog zu `RobotWithCoinTypesAndRefStateOne` ist: `RobotWithCoinTypesAndRefStateTwo` hat keine vier Attribute, sondern nur ein `private`-Attribut `refRobot` vom Typ `ReferenceRobot`, welcher Ihnen in der Vorlage bereits vorgegeben wird.

Implementieren Sie analog zu H2.2 den Konstruktor von `RobotWithCoinTypesAndRefStateTwo`. Statt der Initialisierung der vier Attribute soll nach dem Aufruf des Konstruktors der Basisklasse ein neues Objekt vom Typ `ReferenceRobot` mit den entsprechenden Werten für die vier Ihnen bekannten Roboter-Attribute.

Implementieren Sie analog zu H2.2 die Methoden `setCurrentStateAsReferenceState` sowie die vier Methoden `getDiff{X,Y,Direction,NumberOfCoins}`. Statt der vier Attribute `ref{X,Y,Direction,NumberOfCoins}` sollen jetzt die Referenzwerte in dem Attribut `refRobot` gespeichert und abgerufen werden. Sie können auf die jeweiligen Attribute von `refRobot` über die jeweiligen Getter- und Setter-Methoden zugreifen.

Beispiel:

Sie können über die Objektmethode `setRefX` den Wert von `refX` von `refRobot` setzen und dann über die Objektmethode `getRefX` von `refRobot` abrufen.

H2.4: Klasse ohne Roboter? Ist mir egal! ?? Punkte

Erstellen Sie eine `public`-Klasse `CoinCollection`, die das Interface `WithCoinTypes` implementiert. Analog zu der Klasse aus H2.1 hat ein Objekt dieser Klasse drei `private`-Attribute `numberOf{Silver,Brass,Copper}Coins` vom Typ `int`.

Der `public`-Konstruktor von `CoinCollection` hat drei Parameter `numberOf{Silver,Brass,Copper}Coins` vom Typ `int` in gegebener Reihenfolge. Der Konstruktor initialisiert die Objektattribute mit den gegebenen Anzahlen.

Wieder soll `getNumberOfCoinsOfType` die Anzahl der Münzen des spezifizierten Münztyps zurückgeben und `setNumberOfCoinsOfType` die Anzahl der Münzen für den durch den Parameter `CoinType` spezifizierten Münztyp setzen . Falls der zweite aktuale Parameterwert von `setNumberOfCoinsOfType` negativ ist, soll der Wert auf 0 gesetzt werden.

Die Klasse `CoinCollection` hat weiter zu jedem der drei Attribute eine `get`-Methode, also die drei Methoden `getNumberOf{Silver,Brass,Copper}Coins` ohne Parameter und mit Rückgabetyp `int`.

Außerdem soll `CoinCollection` eine `public`-Methode `insertCoin` und eine `public`-Methode `removeCoin` haben. Beide Methoden haben einen Parameter vom formalen Typ `CoinType` und keine Rückgabe. Die erste Methode inkrementiert die Anzahl des spezifizierten Münztyps um 1. Die zweite Methode dekrementiert die Anzahl des spezifizierten Münztyps um 1, falls diese Anzahl positiv ist. Andernfalls hat die zweite Methode keinen Effekt.