Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 04

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Karsten Weihe

Übungsblattbetreuer:	Darya Nikitina
Wintersemester 22/23	v1.0-SNAPSHOT
Themen:	Interfaces
Relevante Foliensätze:	01g (und natürlich auch 01a-f)
Abgabe der Hausübung:	25.11.2022 bis 23:50 Uhr

Hausübung 04 — Gesamt: 35 Punkte

Roboter mit Referenzstatus

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* in unserem 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 in der Vorlage relevanten Verzeichnisse sind `src/main/java/h04` und `src/test/java/h04`.

Einleitung

Screenshots der `World` mit Ihren Robotern 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 (Pseudocode) dürfen Sie selbstverständlich nicht teilen, posten oder sonstwie an Andere weitergeben (außer an die Ausrichter und Tutoren der FOP 22/23)!

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 dies nicht unbedingt zeitnah prüfen - das heißt, es hat noch nichts zu bedeuten, wenn Sie erst einmal nichts von uns hören.

Verbindliche Anforderung: Dokumentation in Java

Mittels JavaDoc und den Tags `@param` und `@return` wollen wir nun in Java eine geeignete Dokumentation unserer Methoden vornehmen. Das JavaDoc können Sie automatisch vor jeder Methode mittels `/**` gefolgt von der Enter-Taste generieren. Ein Beispiel könnte wie folgt aussehen:

```java
/**
* This method accepts two real numbers belonging to a
* vector and calculates the euclidean norm of said
* vector.
*
* @param x first component of two-dimensional vector (x, y)
* @param y second component of two-dimensional vector (x, y)
* @return Euclidean norm of the vector (x,y)
*/
double euclid2(float x, float y) {
    return Math.sqrt(x*x + y*y);
}
```

Diese Art von Dokumentation mit Vertrag ist Pflicht in dieser Hausübung für alle Java-Methoden! Auch Konstruktoren müssen dokumentiert werden! Orientieren Sie sich dabei an obigem Beispiel. Für jede fehlende Dokumentation einer Methode Ihrer abgegeben Hausübung erhalten Sie einen Punkt Abzug auf Ihre erreichten Punkte. Sie können auf diese Art und Weise bis zu 20% der Punkte einer Hausübung abgezogen bekommen!

Verbindliche Anforderung: Alle Aufgaben sind in Package `h04` umzusetzen. Achten Sie darauf, dass Sie alle Dateien genau in diesem Package erzeugen, sowie auf die korrekte Schreibweise aller Datei- und Methodennamen und aller Identifier. Bei falscher Schreibweise werden die Klassen nicht gefunden und Sie bekommen nicht die Punkte, die Sie sich eigentlich erarbeitet haben.

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.

***Kleiner Vorgriff*:**[1] In 01f haben Sie gesehen, wie man eine Klasse von einer anderen Klasse mit `extends` ableiten kann. In 01g haben sie dann Klassen betrachtet, die jeweils mit `implements` ein Interface implementieren. Wir gehen hier gleich zwei Schritte weiter und definieren

(i) Klassen, die sowohl von einer anderen Klasse abgeleitet sind als auch Interfaces implementieren

(ii) Klassen, die dabei sogar mehr als ein Interface implementieren.

Abstraktes Beispiel für die Schreibweise: Die Klasse `X` soll von der Klasse `Y` abgeleitet werden und drei Interfaces namens `Inf1`, `Inf2` und `Inf3` implementieren. Dann sieht die Definition der Klasse `X` so aus, wobei die Reihenfolge von `Inf1`, `Inf2` und `Inf3` egal ist:

```
public class X extends Y implements Inf1, Inf2, Inf3 {
  ..........
}
```

Wie Sie es schon kennen, ist der eigentliche Inhalt der Klasse `X`, also die Attribute und Methoden von `X`, im obigen Beispiel ausgelassen und nur durch „`..........`“ angedeutet.

H1: Zwei Interfaces ?? Punkte

Das Ziel dieser Aufgabe ist es zwei Interfaces zu schreiben, die in den späteren Aufgaben benötigt werden. Das erste Interface `RobotWithReferenceState` ist noch an Klasse `Robot` angelehnt, aber das zweite Interface `WithCoinTypes` ist nicht mehr nur für Roboter gedacht.

H1.1: Interface `RobotWithReferenceState` ?? Punkte

Schreiben Sie in einer Datei `RobotWithReferenceState.java` ein `public`-Interface mit dem Namen `RobotWithReferenceState`. Es soll fünf Methoden enthalten: Eine Methode `setCurrentStateAsReferenceState` ohne Parameter und ohne Rückgabe und die vier Methoden `getDiffX`, `getDiffY`, `getDiffDirection` und `getDiffNumberOfCoins`. Diese vier Methoden haben ebenfalls keine Parameter. Ansonsten liefert `getDiffDirection` `Direction` zurück und die anderen drei Methoden liefern `int` zurück.

Die Idee von diesem Interface ist es, dass man bei einem Roboter immer mit `setCurrentStateAsReferenceState` den momentanen Status, also die momentanen Werte der vier Attribute `x, y, direction` und `numberOfCoins`, speichern kann. Dann kann man die gespeicherten Werte der vier Attribute relativ (genauer: als Differenz) zum jeweils letzten festgehaltenen Status mit der jeweiligen Methode (beispielsweise `getDiffX` für das Attribut `x`) ausgeben lassen.

H1.2: Interface `WithCoinTypes` ?? Punkte

Schreiben Sie in einer Datei `CoinType.java` eine `public`-Enumeration `CoinType` mit Konstanten `SILVER`, `BRASS`

[1] Bei Interesse schauen Sie sich in Kapitel 03b rund um Folie 110 um.

und `COPPER`[2].

Schreiben Sie in einer Datei `WithCoinTypes.java` ein `public`-Interface `WithCoinTypes` mit zwei Methoden: `getNumberOfCoinsOfType` und `setNumberOfCoinsOfType`. Dabei hat `getNumberOfCoinsOfType` einen Parameter vom formalen Typ `CoinType` und liefert `int` zurück und `setNumberOfCoinsOfType` liefert nichts zurück und hat einen ersten Parameter von Typ `CoinType` und einen zweiten Parameter vom Typ `int`.

Die Idee von diesem Interface ist, dass ein Objekt einer `WithCoinTypes` implementierenden Klasse 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.

***Fehlermeldungen besser verstehen* (0 Punkte):** 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 auch einmal in `Main.java` 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: Implementierende Roboterklasse mit verschiedenen Münztypen ?? Punkte

Leiten Sie in einer Datei `RobotWithCoinTypes.java` eine `public`-Klasse `RobotWithCoinTypes` direkt von `Robot` aus `FopBot` ab. Zudem soll diese Klasse das Interface aus H1.2 implementieren.

Ein Objekt der Klasse `RobotWithCoinTypes` hat drei `private`-Attribute: `numberOfSilverCoins`, `numberOfBrassCoins` und `numberOfCopperCoins`. Alle Attribute sind vom Typ `int`.

Fügen Sie in `RobotWithCoinTypes` einen `public`-Konstruktor ein. Er soll dieselben formalen vier Parameter in derselben Reihenfolge wie der entsprechende Konstruktor von `Robot` haben. Den vierten Parameter von `RobotWithCoinTypes`, `numberOfCoins`, ersetzen Sie im nächsten Schritt durch drei Parameter vom formalen Typ `int`: `numberOfSilverCoins`, `numberOfBrassCoins` und `numberOfCopperCoins`. Dieser Konstruktor ruft wie üblich mit `super` den Konstruktor seiner direkten Basisklasse auf, und zwar mit seinen eigenen ersten drei aktualen 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.

***Zur eigenen Kontrolle (also 0 Punkte)*:** Richten Sie nun ein Objekt von `RobotWithCoinTypes` ein und lassen Sie sowohl eine Variable von Klasse `RobotWithCoinTypes` als auch eine Variable von Klasse `Robot` darauf verweisen. Prüfen sie mit `get{X, Y, Direction, NumberOfCoins}` mit beiden Robotervariablen, ob die Initialisierung der vier

[2] vgl. Kapitel: 01e, Folien: 2-14

Attribute im Konstruktor korrekt ist. Ihre Tests können Sie in der Methode `yourTests` in `Main.java` schreiben und der Inhalt dieser Methode wird nicht bewertet.

Zuerst soll `setNumberOfCoins` von `Robot` in `RobotWithCoinTypes` überschrieben werden. Leider bietet die Methode `setNumberOfCoins` von `Robot` 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.

***Achtung*:** Es gibt also jetzt zwei Implementationen von `setNumberOfCoins`: eine in Klasse `Robot` und eine in Klasse `RobotWithCoinTypes`. Wie Sie in Kapitel 01g gesehen haben, rufen Sie in Methoden von `RobotWithCoinTypes` die erste Implementation mit `super.setNumberOfCoins` auf, die zweite nur mit `setNumberOfCoins` ohne `super`.

Die Methoden `getNumberOfCoinsOfType` und `setNumberOfCoinsOfType` von `WithCoinTypes` implementieren Sie in `RobotWithCoinTypes` auf Basis der drei Attribute `numberOfSilverCoins`, `numberOfBrassCoins` und `numberOfCopperCoins`.

Die Methode `getNumberOfCoinsOfType` soll die Anzahl der Münzen des spezifizierten Münztyps zurückgeben und die Methode `setNumberOfCoinsOfType` soll die Anzahl der Münzen für den durch den Parameter `CoinType` spezifizierten Münztyp setzten. Falls der zweite aktuale Parameterwert von `setNumberOfCoinsOfType` negativ ist, soll die Methode `setNumberOfCoins` von `Robot` aufgerufen werden. [3]. Damit das Attribut `numberOfCoins` von `Robot` sich konsistent ändert, muss in `setNumberOfCoinsOfType` auch `setNumberOfCoins` von `Robot` mit einem entsprechenden Wert aufgerufen werden.

Die Methoden `putCoin` und `pickCoin` 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 [4].

***Zur eigenen Kontrolle (also* 0 *Punkte)*:** Erweitern Sie die obigen Tests um einen Test der in H2.1 implementierten Methoden: Richten Sie eine Variable von `RobotWithCoinTypes` ein und lassen Sie sie auf das im obigen Test eingerichtete Objekt von `RobotWithCoinTypes` verweisen. Prüfen Sie über diese Variable, dass die Methoden `setNumberOfCoinsOfType`, `setNumberOfCoins`, `putCoin` und `pickCoin` in `RobotWithCoinTypes` korrekt implementiert sind und die entsprechenden Münzwerte konsistent zu den Werten der Klasse `Robot` sind. Bei Interesse können Sie auch prüfen, was passiert, wenn Sie beispielsweise der Methode `setNumberOfCoins` einen negativen Wert übergeben.

H2.2: Implementierende Roboterklasse mit vier `int`-Attributen für den Referenzstatus ?? Punkte

Leiten Sie in einer Datei `RobotWithCoinTypesAndRefStateOne.java` eine `public`-Klasse `RobotWithCoinTypesAndRefStateOne` von `RobotWithCoinTypes` ab. Zudem soll diese Klasse das Interface aus H1.1 implementieren.

[3] Ausblick: In Kapitel 05 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.

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

Ein Objekt der Klasse `RobotWithCoinTypesAndRefStateOne` hat vier `private`-Attribute: `refX`, `refY`, `refDirection` und `refNumberOfCoins`. Das Attribut `refDirection` ist vom Typ `Direction`, die anderen drei Attribute sind vom Typ `int`.

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 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.

Zur eigenen Kontrolle (also 0 Punkte): Richten Sie nun ein Objekt von `RobotWithCoinTypesAndRefStateOne` ein und lassen Sie sowohl eine Variable von Klasse `RobotWithCoinTypesAndRefStateOne` als auch eine Variable von Klasse `Robot` darauf verweisen. Prüfen sie mit `get{X, Y, Direction, NumberOfCoins}` mit beiden Robotervariablen, ob die Initialisierung der vier Attribute im Konstruktor korrekt ist. Ihre Tests können Sie in der Methode `yourTests2` in `Main.java` schreiben, der Inhalt dieser Methode wird nicht bewertet.

Die 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.

Die Methoden `getDiffX`, `getDiffY` und `getDiffNumberOfCoins` 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. Das bedeutet die Rückgabe der Methoden zeigt 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.

***Tipp*:** Schauen Sie sich die Methode `ordinal` von Enumerationen an.

Zur eigenen Kontrolle (also 0 Punkte): Erweitern Sie die obigen Tests um einen Test der in H2.2 implementierten Methoden: Richten Sie eine Variable von `RobotWithCoinTypesAndRefStateOne` ein und lassen Sie sie auf das im obigen Test eingerichtete Objekt von `RobotWithCoinTypesAndRefStateOne` verweisen. Prüfen Sie über diese Variable, dass die Methoden, die Sie in dieser Aufgabe geschrieben haben, korrekt implementiert sind.

***Fehlermeldungen besser verstehen* (0 Punkte):** 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 die Kompilierung mit einer Fehlermeldung abbrechen. Kommentieren Sie eine ganze Methode in `RobotWithCoinTypesAndRefStateOne` einmal aus (zum Beispiel `getDiffNumberOfCoins`). Auch jetzt sollte die Kompilierung mit einer Fehlermeldung abbrechen. Erscheinen Ihnen diese Fehlermeldungen sinnvoll? Vergessen Sie nicht, alle Fehler wieder rückgängig zu machen, bevor Sie mit den eigentlichen Aufgaben weitermachen.

H2.3: Implementierende Roboterklasse mit `Robot`-Objekt für den Referenzstatus ?? Punkte

Völlig analog zu `RobotWithCoinTypesAndRefStateOne` schreiben Sie in `RobotWithCoinTypesAndRefStateTwo`

`.java` eine `public`-Klasse `RobotWithCoinTypesAndRefStateTwo`, mit einen Unterschied: Die Klasse `RobotWith CoinTypesAndRefState2` hat keine vier Attribute `ref{X,Y,Direction,NumberOfCoins}`, sondern stattdessen ein `private`-Attribut `refRobot` vom Typ `ReferenceRobot`. Schreiben Sie analog zu der Aufgabe 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 x-Koordinate, y-Koordinate, Richtung und Anzahl der Münzen erstellt und mit `refRobot` referenziert werden.

Schreiben Sie analog zu der Aufgabe H2.2 die Methoden `setCurrentStateAsReferenceState`, `getDiffX`, `getDiffY`, `getDiffDirection` und `getDiffNumberOfCoins`. 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. Also beispielsweise können Sie über die Objektmethode `setRefX` den Wert von `refX` in `refRobot` setzen und über die Objektmethode `getRefX` den Wert von `refX` in `refRobot` abrufen.

Zur eigenen Kontrolle (also 0 Punkte): Erweitern Sie die obigen Tests um einen Test der in H2.3 implementierten Methoden: Richten Sie eine Variable von `RobotWithCoinTypesAndRefStateTwo` ein und lassen Sie sie auf ein eingerichtetes Objekt von `RobotWithCoinTypesAndRefStateTwo` verweisen. Prüfen Sie über diese Variable, dass die Methoden, die Sie in dieser Aufgabe geschrieben haben, korrekt implementiert sind.

H2.4: Klasse ohne Roboter, aber mit Münztypen ?? Punkte

Schreiben Sie in einer Datei `CoinCollection.java` eine `public`-Klasse `CoinCollection`, die das Interface `WithCoinTypes` implementiert. Analog zu der Klasse aus H2.1 hat ein Objekt dieser Klasse drei `private`-Attribute `numberOfSilverCoins`, `numberOfBrassCoins` und `numberOfCopperCoins` vom Typ `int`.

Der `public`-Konstruktor von `CoinCollection` hat drei formale Parameter vom Typ `int` in der Reihenfolge `numberOfSilverCoins`, `numberOfBrassCoins` und `numberOfCopperCoins`. Er initialisiert die entsprechenden Münzzahlattribute mit den drei Münzzahlen.

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

Außerdem hat die Klasse `CoinCollection` zu jedem der drei Attribute eine `get`-Methode, also die drei Methoden `getNumberOfSilverCoins`, `getNumberOfBrassCoins` und `getNumberOfCopperCoins`, jeweils ohne Parameter und mit Rückgabetyp `int`.

Zudem 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 erhöht die Anzahl des spezifizierten Münztyps um 1. Die zweite Methode vermindert diese Anzahl um 1, falls diese Anzahl positiv ist. Andernfalls hat die zweite Methode keinen Effekt.

Zur eigenen Kontrolle (also 0 Punkte): Testen Sie auch die Klasse `CoinCollection`.