Funktionale und objektorientierte Programmierkonzepte
Übungsblatt 11

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Karsten Weihe

Wintersemester 23/24	v1.0
Themen:	Streams
Relevante Foliensätze:	08 (erster Teil: Streams)
Abgabe der Hausübung:	31.01.2024 bis 23:50 Uhr

Hausübung 11 — Gesamt: 32 Punkte

Running a Business

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/h11` und ggf. `src/test/java/h11`.

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.

Verbindliche Anforderung für die gesamte Hausübung:

- In dieser Hausübung dürfen Sie **keine Schleifen** und **keine Rekursion** verwenden. Sie müssen alle Aufgaben **ausschließlich** mit `Stream` und den entsprechenden Methoden lösen.
- Nutzen Sie nur die Attribute, welche wir Ihnen vorgeben, und fügen Sie keine eigenen Attribute hinzu.

Einleitung

In dieser Aufgabe werden Sie nicht nur mit Klassen, sondern auch mit Records in Java arbeiten. Gucken Sie sich hierfür einmal die Folien 3b Systematische Abrundung bisheriges in Java und dort das Kapitel zu Record-Classes an.

Die Vorlage besteht aus den folgenden Klassen und Records:

- `Employee`:
  Die Klasse `Employee` repräsentiert einen Angestellten in der Firma. Ein Objekt der Klasse `Employee` besitzt zwei `private`-Objektattribute:

  1. Das erste `salary` vom Typ `double`, welches das Gehalt eines Angestellten angibt
  2. Das zweite `position` vom Typ `Position`, welches die entsprechende Position eines Angestellten angibt. `Position` ist dabei eine Enumeration

  Außerdem besitzt es eine `private`-Objektkonstante:

  1. `NAME` vom Typ `String`, welche den Namen eines Angestellten angibt.

     Der Name ist dabei wie folgt formatiert: `"Vorname Nachname"`

- `Department`:
  Der Record `Department` repräsentiert eine Abteilung/ein Department der Firma. Ein Department hat mehrere Angestellte. Entsprechend besitzt ein Objekt des Records `Department` ein `private`-Objektattribut:

  1. employees vom Typ `List<Employee>`, welches die Liste der Angestellten in dem Department angibt

- `Product`:
  Der Record `Product` repräsentiert ein Produkt. Ein Objekt des Records `Product` besitzt vier `private`-Objektattribute:

  1. `type` vom Typ `ProductType`, welches den Typen angibt, vom dem das Produkt ist, wobei `ProductType` eine Enumeration ist
  2. `price` vom Typ `double`, welches den aktuellen Preis für das Produkt angibt
  3. `quantity` vom Typ `int`, welches die Anzahl angibt, wie oft das Produkt vorhanden ist
  4. `name` vom Typ `String`, welches den Namen des Produktes angibt

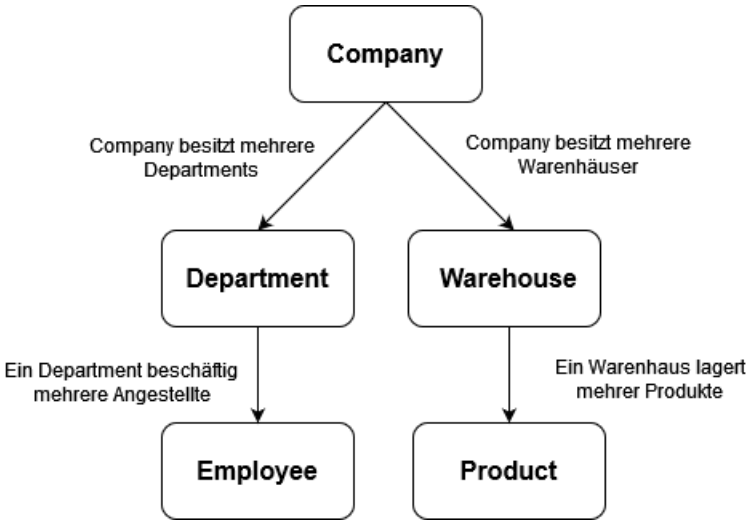
- `Warehouse`:
  Die Klasse `Warehouse` repräsentiert ein Warenhaus, in welchem Produkte gelagert werden. Ein Objekt der Klasse `Warehouse` besitzt folgende 3 `private`-Objektattribute:

  1. `products` vom Typ `List<Products>`, welches die Liste aller Produkte ist, die in diesem Warenhaus sind
  2. `maxCapacity` vom Typ `int`, welches die maximale Kapazität des Warenhauses angibt
  3. `currentCapacity` vom Typ `int`, welches die aktuelle Menge an Produkten im Warenhaus angibt.

- `Company`:
  Der Record `Company` repräsentiert nun die Firma. Ein Objekt des Records `Company` besitzt zwei `private`-Objektattribute:

  1. `departments` vom Typ `List<Department>`, welches die zu verwaltenden Departments angibt
  2. `warehouses` vom Typ `List<Warehouse>`, welches die zu verwaltenden Warenhäuser angibt

Um Ihnen einen groben Überblick zu geben, wie die Klassen und Records zusammenhängen, geben wir hier eine kleines Diagramm an:



Hinweis:

Beachten Sie bitte, dass es sich bei dem Diagramm um **kein** spezielles Diagramm handelt, wie ein UML-Klassendiagramm. Das Diagramm hier dient jediglich dazu, sich einen groben Überblick zu verschaffen.

H1: Das Department

In dieser Aufgabe, werden sie das Verwalten eines Departments realisieren

H1.1: Liste aller Positionen — 2 Punkte

Zuerst wollen Sie eine Liste aller Positionen haben, welche es in einem Department gibt.

Implementieren Sie in dieser Aufgabe die `public`-Objektmethode `getListOfPositionsInDepartment`. Diese Methode hat keinen Parameter und liefert ein Objekt vom Typ `List<Position>` zurück. Die Methode soll eine Liste aller Positionen, welche im Department enthalten sind, zurückliefern. Achten Sie dabei darauf, dass keine Position doppelt enthalten ist.

> **Hinweis:**
>
> Gucken Sie sich hier nochmal in der Dokumentation von `Stream` die folgende Methode an:
>
> - `java.util.stream#distinct()`

H1.2: Liste aller Angestellten einer Position — 2 Punkte

Nun, da Sie wissen, welche Positionen es in ihrem Department gibt, würden Sie gerne die Angestellten filtern, welche eine bestimmte Position besitzen.

Implementieren Sie nun die `public`-Objektmethode `filterEmployeeByPosition`, welche einen formalen Parameter `position` vom Typ `Position` besitzt und als Rückgabety `List<Employee>` hat.
Die Rückgabe der Methode soll eine Liste aller Angestellten sein, welche die im aktualen Parameter übergebenen Position besitzen, welche gleich dem akktualen Parameter ist.

H1.3: Nach Gehalt filtern — 2 Punkte

Implementieren Sie die `public`-Objektmethode `getNumberOfEmployeesBySalary`, welche einen formalen Parameter `salary` vom Typ `double` besitzt und als Rückgabetyp `long` hat.
Die Methode liefert einfach die Anzahl aller Angestellten zurück, welche ein Gehalt größer oder gleich dem im aktualen Parameter gegebenen Wert haben.

H1.4: Gehaltserhöhung? — 2 Punkte

Als Nächstes wollen Sie die Möglichkeit haben, das Gehalt entsprechend anzupassen.

Implementieren Sie die `public`-Objektmethode `adjustSalary`, welche einen formalen Parameter `amount` vom Typ `double` und einen zweiten formalen Parameter `increase` vom Typ `boolean` hat und nichts zurückliefert.
Die Methode soll für jeden Angestellten in dem Department das entsprechende Gehalt, um die im ersten aktualen Parameter angegebene Menge erhöhen oder verringern, je nach Wert im zweiten aktualen Parameter.

H2: Das Warenhaus

In dieser Aufgabe realisieren Sie das Verwalten eines Warenhauses.
Sie können für alle Methoden, außer H2.1, davon ausgehen, dass die Eingaben nicht `null` sind.

H2.1: Produktpreis — 2 Punkte

Implementieren Sie dazu die `public`-Objektmethode `getPrice`. Diese besitzt einen formalen Parameter `product` vom Typ `Product` und liefert `double` zurück.
Die Rückgabe der Methode soll einfach der Preis des im aktualen Parameter übergebenen Produktes sein. Achten Sie dabei darauf, dass, wenn der übergebene Parameter `null` ist, Sie dann einen Preis von 0.0 zurückliefern.

Verbindliche Anforderung:

In dieser Aufgabe dürfen Sie keine `if-else`-Statements oder den ternären Operator verwenden. Sie müssen mit `Optional` arbeiten.

Hinweis:

Schauen Sie sich für diese Aufgabe in `Optinal`-Dokumentation die Methoden `ofNullable` und `orElse` an.

H2.2: Nur bestimmte Produkte gefordert — 1 Punkt

Anstatt einer normalen `get`-Methode für die Produkte werden Sie in dieser Aufgabe eine zusätzliche `get`-Methode implementieren, welche nur bestimmte Produkte ausgibt

Implementieren Sie dafür die `public`-Objektmethode `getProducts`, welche eine Liste von Produkten zurückgibt und ein formalen Parameter `predicate` vom Typ `Predicate<? super Product>` besitzt.
Die Methode soll entsprechend eine Liste von Produkten zurückliefern, für die das Prädikat `true` ausgibt.

H2.3: Übersicht über die Stückzahl — 2 Punkte

Implementieren Sie die `public`-Objektmethode `getTotalQuantityOfProduct`. Die Methode besitzt den Rückgabetyp `long` und hat einen formalen Parameter `product` vom Typ `Product`.
Die Methode soll die Gesamtmenge des im formalen Parameter übergebenen Produktes in dem aktuellen `Warehouse`-Objekt zurückliefern.

Hinweis:

Sie kommen an die Gesamtmenge eines Produktes über die Anzahl der Objekte, welche in einem Warenhaus vorliegen und gleich dem `Product`-Objekt sind.

H2.4: Wieviel Wert steckt denn nun hier drinnen ? — 2 Punkte

Es kann durchaus von Wichtigkeit sein, zu wissen wieviel denn nun die Produkte wert sind, die Sie in ihrem Warenhaus haben.

In dieser Aufgabe implementieren Sie die `public`-Objektmethode `getTotalPrice`, welche als Rückgabetyp `double` hat und keine formalen Parameter besitzt.
Die Methode soll die Summe der Preise aller Produkte im aktuellen `Warehouse`-Objekt zurückliefern.

H2.5: Eine Lieferung kommt rein — 2 Punkte

Um das Hinzufügen von Produkten in das Warenhaus zuz realisieren, implementieren Sie in dieser Aufgabe eine Hilfsmethode dafür. Fangen Sie zuerst damit an, Produkte zu erzeugen/generieren, die Sie hinzufügen möchten.

Implementieren Sie nun die `public`-Objektmethode `generateProducts`. Die Methode besitzt drei formale Parameter. Der erste `type` vom Typ `ProductType`, der zweite `price` vom Typ `double` und der dritte `name` vom Typ `String`. Der Rückgabetyp der Methode ist `Stream<Product>`.
Die Methode soll einen `Stream` erzeugen, welcher beliebig viele Objekte vom Typ `Product`, mit den in den aktualen Parametern übergebenen Spezifikationen, erzeugt. Da der `Stream` beliebig viele Objekte vom Typ `Product` erzeugen soll, reicht es aus, wenn Sie die `quantity` eines Produktes auf 1 setzen.

> **Hinweis:**
>
> Gucken Sie sich hier nochmal in der Dokumentation von `Stream` die folgende Methode an:
>
> - `java.util.stream#generate(java.util.function.Supplier)`

H2.6: Aufstocken — 2 Punkte

Implementieren Sie jetzt die `public`-Objektmethode `addProducts`, welche einen formalen Parameter `product` vom Typ `Product` und einen formalen Parameter `numberOfProducts` vom Typ `int` besitzt.
Die Methode soll nun Produkte aus dem, in der vorherigen Aufgabe implementierten, `Stream<Product>` zu dem Attribut `products` hinzufügen, und zwar so viele Elemente, wie der Wert im aktualen Parameter `numberOfProducts` vorgibt.

> **Hinweis:**
>
> Gucken Sie sich hier nochmal in der Dokumentation von `Stream` die folgende Methode an:
>
> - `java.util.stream#limit(long)`

H3: Die Companny

H3.1: Übersicht aller Mitarbeiter — 1 Punkt

Nachdem Sie im Department jetzt wissen wer alles dort arbeitet, wollen Sie nun einen Gesamtüberblick über alle haben.

In dieser Aufgabe implementieren Sie die `public`-Objektmethode `getListOfAllEmployee`, welche keine Parameter besitzt und den Rückgabetyp `List<Employee>` hat.
Die Methode soll eine Liste aller Angestellten aus allen Departments zurückliefern.

Hinweis:

Gucken Sie sich hier nochmal in der Dokumentation von `Stream` die folgende Methode an:

- `java.util.stream#flatMap(java.util.function.Function)`

H3.2: Übersicht Gesamtanzahl der Produkte — 1 Punkt

Als nächsten brauchen wir noch den Überblick über die gesamte Anzahl an Produkten, welche die Firma besitzt.

Implementieren Sie die `public`-Objektmethode `getQuantityOfProduct`, welche als Rückgabetyp `long` hat und einen formalen Parameter `product` vom Typ `Product` besitzt.
Die Methode soll die gesamte Anzahl des im aktualen Parameter übergebene Produktes aus allen Warenhäusern zurückliefern.

Hinweis:

Es lohnt sich hier einen Blick auf die folgende Methode von Stream zu werfen:

- `java.util.stream#mapToInt`

Sowie sich das Interface `IntStream` anzugucken

H3.3: Filtern der Produkte — 3 Punkte

Vielleicht kennen Sie das, wenn Sie auf einer Seite Filter anwenden, damit Sie das Produkt bekommen, welches Sie suchen. Genau das implementieren Sie in dieser Aufgabe.

Implementieren Sie die `public`-Objektmethode `getFilteredProductNames`. Die Methode hat den Rückgabetyp `List<String>` und besitzt einen formalen Parameter predicates vom Typ `List<Predicate<Product>>`.
Die Rückgabe der Methode soll eine Liste mit allen Produktnamen sein, aus allen Warenhäusern, welche alle Prädikate aus dem aktualen Parameter erfüllen.

Hinweis:

Gucken Sie sich hier noch einmal die Dokumentation zu `Predicate`. Außerdem können Sie auch noch einen Blick auf folgende Methode von Stream werfen:

- `java.util.stream#allMatch(Predicate<? super T> predicate)`

H3.4: Preisspanne vorgeben — 2 Punkte

Jetzt werden Sie noch den Filter hinzufügen, welcher nur Produkte in einer bestimmten Preisspanne anzeigt und entsprechend sortiert.

Implementieren Sie die `public`-Objektmethode `priceRange`. Die Methode besitzt zwei formale Parameter `low` und `high` beide vom Typ `double`. Der Rückgabetyp der Methode ist `List<Product>`.
Die Rückgabe der Methode soll eine Liste aller Produkte aus allen Warenhäusern sein, deren Preis in dem Intervall [`low`, `high`] liegt. Außerdem soll die Liste aufsteigend sortiert sein nach den Preisen der Produkte.

H3.5: Übersicht der Namen — 2 Punkte

Anstatt eine Liste mit vielen Informationen zu bekommen, welche man alle gar nicht braucht, implementieren Sie nun eine Methode, welche nur die Namen von Angestellten ausgibt und zudem noch schön formatiert.

Implementieren Sie nun die `public`-Objektmethode `getEmployeesSortedByName`, welche keine Parameter und als Rückgabetyp `List<String>` besitzt.
Die Methode soll eine Liste der Namen aller Angestellten aus allen Departments zurückliefern, welche aufsteigend nach Nachnamen sortiert ist.
Außerdem sollen die `String`-Objekte wie folgt formatiert werden:

`"Max Mustermann"` → `"Mustermann, Max"`

Hinweis:

In dieser Aufgabe gibt es mehrere Möglichkeiten, wie Sie die Formatierung der Strings realisieren können.

- Sie können einmal auf `String` arbeiten, gucken Sie sich dafür in der Klasse `String` die Methode `split` an
- Sie können aber auch, wenn Sie wollen, mit regulären Ausdrücken arbeiten. Gucken Sie sich dafür einmal die Klasse `java.util.regex.Pattern` an

H3.6: Schnellübersicht von Produkten — 4 Punkte

Um schnell einen Überblick über Produkte zu bekommen, implementieren Sie in dieser Aufgabe eine Methode, welche genau das für Sie übernimmt.

Implementieren Sie nun abschließend die `public`-Objektmethode `getAllProductsByType`. Die Methode besitzt einen formalen Parameter `type` vom Typ `ProductType` und hat den Rückgabetyp `List<String>`.
Die Rückgabe der Methode soll eine Liste aller Produkte, des im aktualen Parameter übergebenen Produkttypens und dem entsprechenden Preis sein. Die Liste soll zudem absteigend sortiert sein, nach den Preisen der Produkte, und es sollen nur die im aktualen Parameter `numberOfProducts` gegebene Anzahl an Produkten angezeigt werden.

Beispiel:

Ein String in der Rückgabe sollte dann wie folgt aussehen:

`"Laptop: 1000"`