13. Übung Programmierung I: Java

Ausgabe: 28.01.2019
Abgabe: 5.02.2019 12^{00}

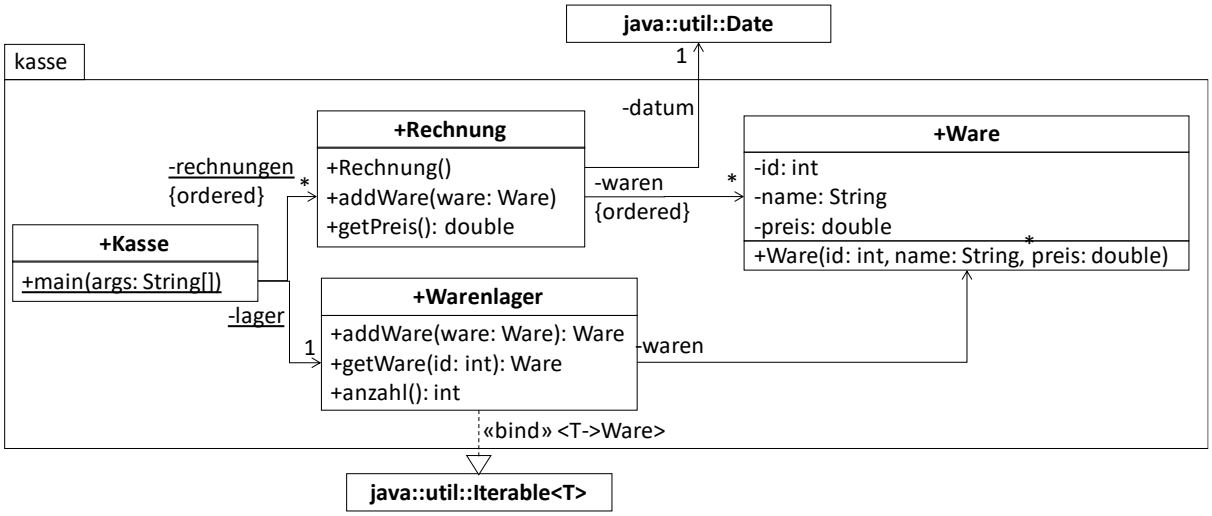
Aufgabe	Punkte	Basislösung	Abgabetests
1	123	Blatt13Aufgabe1Prj.zip	Übersetzbarkeit, Javadoc-Konfiguration, (Teil-) Korrektheit des Algorithmus‘
Gesamtpunkte	**123**		

1. Aufgabe:

In dieser Aufgabe ist ein einfaches Kassensystem zu entwickeln.

Teilaufgabe a) – 123 Punkte

Die Grundstruktur ist in der folgenden Abbildung dargestellt, wobei nicht alle erforderlichen Methoden/Klassen eingezeichnet sind, d.h., diese sind ggf. zu ergänzen. Möglicherweise unbekannte UML-Eigenschaften werden in dieser Aufgabenstellung erklärt. In dieser Aufgabe sind geeignete Datenstrukturen aus `java.util` zu verwenden, insbesondere vom Typ `List` und `Map`.



- Eine `Ware` stellt die Verkaufseinheit des Kassensystems dar und ist durch eine numerische Identifikation (`id`), einen Namen (`name`) und einen Preis (`preis`) beschreiben. Alle Informationen werden bei der Erzeugung einer Instanz festgelegt und sind danach nicht mehr veränderlich. Falls dabei der `name` `null` oder leer ist, soll eine `IllegalArgumentException` geworfen werden. `Ware` ist mit einer `toString`-Methode auszustatten, die `id`, `name` und den `preis` mit zwei Nachkommastellen ausgibt.
  Hinweis: Für die Nachkommastellen lohnt ein genauer Blick in die Dokumentation von `java.lang.String` oder `java.text.DecimalFormat`.
- Eine `Rechnung` besteht aus einem Datum (`datum`) und Waren (`waren`), wobei die waren in Reihenfolge ihres Hinzufügens gespeichert werden sollen (`{ordered}`).

Wenn beim Hinzufügen einer `Ware` zur `Rechnung` (`add`) `null` übergeben wird, soll eine `IllegalArgumentException` geworfen werden. Das `datum` wird beim Erzeugen einer Rechnungsinstanz auf das aktuelle (heutige) Datum gesetzt. Der Preis einer Rechnung ist die Summe der Preise aller `waren`. `Rechnung` ist mit einer `toString`-Methode auszustatten, die `datum` im Format *Tag.Monat.Jahr* und alle `waren` ausgibt.
Hinweis: Für Datumsformatierung lohnt ein Blick in die Dokumentation von `java.text.SimpleDateFormat`.

- Das Warenlager speichert alle verfügbaren Waren, die in Rechnungen verwendet werden können. Waren können dem Warenlager hinzugefügt werden (`add`), wobei eine `IllegalArgumentException` geworfen werden soll, wenn die hinzuzufügende `ware` `null` ist. Falls für die angegebene `id` bereits eine Ware im Warenlager ist, soll diese `Ware` zurückgegeben und im Warenlager überschrieben werden. Falls keine `Ware` mit der `id` vorhanden ist, wird `null` zurückgegeben. Waren können über die `id` abgefragt werden (`get`), d.h., falls die `id` bekannt ist, wird die zugeordnete Ware und sonst `null` zurückgegeben. `anzahl()` gibt die Anzahl der Waren im Warenlager zurück. `Warenlager` ist zudem ein `Iterable` über dem Inhaltstyp `ware` (der UML-Stereotyp `bind` drückt aus, dass der Typ-Parameter `T` an `Ware` zu binden ist).
  Hinweis: Eine geschickte Wahl der Datenstruktur in Warenlager vereinfacht die Realisierung wesentlich.
- Die `Kasse` realisiert das Hauptprogramm und besteht aus dem `Warenlager` und Rechnungen die in der Reihenfolge ihrer Erzeugung gespeichert werden sollen.

Die `Kasse` soll vom Benutzer interaktiv zu bedienen sein und die folgenden Menüoptionen bieten, wobei die Menüoptionen vor jeder Menüauswahl anzuzeigen sind:

- `R`: Der Benutzer kann eine neue Rechnung anlegen, die in `rechnungen` gespeichert wird. Dabei werden solange Eingaben abgefragt, bis der Benutzer eine leere Zeichenkette eingibt. Falls die Eingaben in eine Ganzzahl übersetzbar sind, die als `id` im Warenlager gespeichert ist, wird die entsprechende Ware der neuen Rechnung hinzugefügt. Sonst wird eine Fehlermeldung ausgegeben und weiter auf die nächsten Benutzereingaben gewartet.
- `Z`: Das Programm zeigt alle gespeicherten `rechnungen` mit Datum (in Format *Tag.Monat.Jahr*) und Preis an.
- `W`: Der Benutzer kann eine neue `Ware` zum `lager` hinzufügen. Dafür ist der `name` (darf nicht leer sein), die `id` (muss positiv sein) und der `preis` (darf nicht negativ sein) einzugeben. Bei Fehleingaben ist der Benutzer zu informieren und solange nachzufragen, bis eine gültige Eingabe getätigt wurde. Wird beim Hinzufügen ins `lager` eine `Ware` überschrieben, ist die alte `Ware` auszugeben.
- `A`: Das Programm zeigt alle gespeicherten Waren im `lager` an.
- `E`: Das Programm wird beendet.

Realisieren Sie die Menüoptionen in geeigneten Unterprogrammen. Erstellen Sie zunächst eine Zielbeschreibung, dann einen Plan und schließlich ihr Programm.

Teilaufgabe b) – Unbewertet (entspricht 35 Punkte)

Das Programm aus Teilaufgabe a) ist mit den folgenden Funktionen und Menüpunkten zu erweitern:

- `F`: Der Benutzer kann filtern, welche Rechnungen auszugeben sind. Dafür kann der Benutzer ein Datum (im Format *Tag.Monat.Jahr*) und einen Preis eingeben (darf nicht

negativ sein) eingeben. Das Programm gibt dann alle Rechnungen aus, die an dem Datum erstellt wurden und mindestens den eingegebenen Gesamtpreis haben.

- `S`: Das Programm speichert das `lager` und die `rechnungen` per Objektserialisierung in die Datei `data.ser` im Wurzelverzeichnis des Projekts. Der Benutzer ist zu informieren, wenn die Daten erfolgreich geschrieben wurden oder falls Fehler beim Schreiben aufgetreten sind.
- `L`: Das Programm lädt `lager` und `rechnungen` per Objektserialisierung aus der Datei `data.ser` im Wurzelverzeichnis des Projekts. Informationen, die vor dem Laden im Programm gespeichert waren, sollen verworfen werden. Falls die Datei nicht existiert, Dateifehler aufgetreten sind oder Klassen beim Laden nicht gefunden werden können, ist der Benutzer entsprechend zu informieren. Wurden die Daten erfolgreich geladen, ist der Benutzer zu informieren, wieviel Waren im `lager` bzw. `rechnungen` vorhanden sind.

Realisieren Sie die Menüoptionen in geeigneten Unterprogrammen. Erweitern Sie zunächst ihren Plan und dann ihr Programm.

Hinweis: Filtern sowie Datei Ein- und Ausgabe wurden bislang nur kurz im Tutorium besprochen. Daher ist dieser Aufgabenteil unbewertet, aber dennoch ggf. bereits lösbar.

Hinweise

- Diese Aufgabe muss mit der vorgegebenen Basislösung und der darin enthaltenen Realisierung der Klassen durchgeführt werden (siehe auch Hinweise zu Übungsblatt 11). Wir empfehlen, sich zunächst die enthaltene JavaDoc-Dokumentation (im Zip-Archiv im Ordner `libs`) genau anzusehen und dann erst die Aufgabe zu bearbeiten.
- Für das Laden und Speichern muss mit Dateien gearbeitet werden. Dateipfade müssen relativ, d.h., nicht absolut und unter Angabe der Dateisystemwurzel sowie betriebssystemunabhängig angegeben werden.
- Alle Fehleingaben und damit zusammenhängende Exceptions sind zu behandeln. Dabei soll die Eingabe wiederholt werden bis ein valider Wert eingegeben wurde.
- **Alternative 1:** Sollten Sie nicht mit Objektserialisierung umgehen können, dürfen Sie für das Schreiben und Lesen der geforderten Daten auch in ein eigenes, textuelles Format verwenden. Allerdings wird die Realisierung dann wesentlich aufwändiger.
- Das Abgabesystem ist so eingestellt, dass Projekte, die mit getrennten src/bin-Ordnern angelegt wurden, nicht angenommen werden (siehe auch Kapitel 2). Oft ist ein derartig angelegtes Projekt ein Hinweis darauf, dass die Eclipse-Grundeinstellungen aus dem LearnWeb (immer noch) nicht in den aktuellen Workspace importiert wurden.
- In dieser Aufgabe sind main-Methoden zum Testen für die Klassen `Rechnung`, `Warenlager` und `Ware` erforderlich. Die Methoden in `Main` werden durch Verwendung im Hauptprogramm getestet.
- Alle Klassen müssen korrekt und wie besprochen mit JavaDoc kommentiert sein. Beachten Sie die oben genannte Checkstyle-Konfiguration.
- Wie eingangs beschrieben können Sie weitere Methoden hinzufügen. Beachten Sie, dass jede Methode entsprechende Kommentierung und Tests erfordert und insbesondere nicht benötigte Methoden zu überflüssiger Mehrarbeit führen.
- Falls das Abgabesystem ihre Lösung wegen entdeckter Fehler abweist, kann es hilfreich sein, Checkstyle vorrübergehend auszuschalten, die dann noch verbleibenden Fehler zu bereinigen und schließlich Checkstyle vor der Abgabe wieder einzuschalten (und natürlich die restlichen Fehler dann zu bereinigen).