14.11.2012

Kim Michael Jansen

Mat.Nr. 2279390

**Objektorientierte Programmierung in Java**

**– Teilleistung 1 –**

**Erläuterung zum Klassendiagramm**

**(Klassennamen sind im Folgenden fett hervorgehoben. Dieser ist per Konvention im Singular, wird für einen flüssigen Sprachgebraucht hier jedoch auch im Plural verwendet.)**

Die Anwendung wird in die Teilbereiche Model (Datenhaltung), View (Interaktion mit dem Benutzer) und Controller (Geschäftslogik) aufgeteilt. Hierdurch werden die Aufgaben aufgeteilt und beherrschbar gemacht. Zudem bietet sich das Design für eine Entwicklung im Team an, damit die Entwickler möglichst unabhängig von anderen Teammitgliedern entwickeln können. Ein fehleranfälliges Mergen von Quellcodedateien soll hierdurch vermieden werden.

View

Die Interaktion mit dem Benutzer wird hier nur grobkonzeptionell vorgesehen. Die Basisleistung ist per Kommandozeile erreichbar. In der Basisstufe wird die Verarbeitung dort gestartet und erzeugt ohne weitere Benutzereingriffe das Verarbeitungsergebnis (Kantinenplan, Einkaufsliste mit Mengen, Preisen und Händlern). Eine Konfiguration erfolgt als Schlüssel-Wert-Paare über eine Propertiesdatei (Dateinamen für Eingangsdateien, Dateipfade für Ausgabedateien, Format der Ausgabedateien, Anzahl der Mitarbeiter).



Model

Die Datenhaltung beinhaltet die Klassen **Kantine**, **Gericht**, **Zutat**, **Haendler**, **ZutatTyp**.

Die Klasse **Kantine** kennt die Anzahl ihrer Mitarbeiter und die **Gerichte**, die ihr Koch kochen kann. Sie dient zudem als Ablageort um den berechneten Speiseplan zu halten. Dieser kann ausgegeben werden (vgl. **SpeiseplanWriter**) und wird als Berechnungsgrundlage für die Einkaufsliste benötigt. Die Einkaufsliste wird nicht gehalten, sondern kann immer neu berechnet werden. Der Speiseplan, kann auch aus den Eingabedateien neu berechnet werden. Aus Performancegründen wird auf diesen I/O-Zugriff verzichtet und der Speiseplan im Arbeitsspeicher gehalten. Zur Identifikation hat die Kantine einen Namen.

Das **Gericht** hat einen Namen, welcher jedoch nicht eindeutig sein muss. Das System sieht mehrere Kantinen vor, die verschiedene **Gerichte** anbieten könnten. In der einen könnte Sauerbraten mit Pferdefleisch gekocht werden, in einer anderen hingegen mit Rindfleisch. Daher ist ein **Gericht** nur gleich einem anderen **Gericht**, wenn es in den **Zutaten** und Zutatenmengen übereinstimmen. Der Name wird für den Speiseplan als Bezeichnung benötigt. Anhand seiner Zutatenliste kann das **Gericht** bestimmen, ob es vegetarisch, ein Fleisch- oder ein Fischgericht ist. Die Verbindung zur **Zutat** liegt als Schlüssel-Wert-Paar vor, wobei die Zutat als Schlüssel dient und der zugeordnete Wert die benötigte Menge ist.

Die **Zutat** hat eine eindeutige Bezeichnung (Singleton auf Name). Um dies sicherzustellen, kann nur die Klasse **Zutat** Instanzen von sich selbst erzeugen. Zudem hat eine **Zutat** genau einen Typ (**ZutatTyp**).

**ZutatTyp** ist eineabschließende Aufzählung (Enumeration) und sieht die Typen „Vegetarisch“, „Fleisch“ und „Fisch“ vor. Für das System ist ein Fisch niemals eine vegetarische Mahlzeit. Auch werden Fische nicht als „Fleisch“ betrachtet, da Fischgerichte nach den Anforderungen separat von Fleischgerichten behandelt werden müssen.

**Haendler** haben einen Namen, eine Strategie zum Berechnen der Transportkosten (vgl. **TransportKostenRechner)** undkennen die **Zutaten**, die sie verkaufen. Außerdem kennt ein **Haendler** noch seine Transportkostenbasis in Euro (vgl. **TransportKostenRechner**). Die Assoziation zu **Zutaten** liegt wird über die Zuordnungsklasse **ZuoZutatPreis** abgebildet. Hierbei wird unterstellt, dass eine Zutat (z.B. Milch) immer in derselben Einheit (hier ml) gemessen wird. Dadurch kann die konkrete Einheit entfallen und als Zahl dargestellt werden. Der Preis pro Einheit wird in Euro angegeben. Zusätzlich ist noch eine kleinste Abgabemenge vorgesehen. Als Beispiel müsste von einem **Haendler**, der Milch in 1000ml Paketen verkauft, 1000ml Milch gekauft werden, um 700ml für einen Kuchen zu erhalten.

Controller

Der **KantinenPlaner** dient als Schnittstelle zum View. Er stellt jeglicher Form von Benutzeroberfläche (Kommandozeile, GUI, oder andere) die benötigte Funktionalität des Systems zur Verfügung. Im derzeitigen Konzept beinhaltet der **KantinenPlaner** Teile der Geschäftslogik. Um eine leichtgewichtige Schnittstelle zur Oberfläche bereit zu stellen, könnten diese Teile in weitere Klassen ausgelagert werden.

Zur Delegation kennt der **KantinenPlaner** eine **SpeiseplanWriter**. Durch das Interface **SpeiseplanWriter** ist es für den KantinenPlaner unwichtig, in welchem Format der Speiseplan später geschrieben wird. Als Formate sind bisher eine einfache Textausgabe (**TextWriter**) und eine Html-Variante (**HtmlWriter**) vorgesehen. Die Darstellung ist nicht Aufgabe des Systems, sondern erfolgt durch ein externes System des Nutzers (z.B. Notepad oder ein Browser).

Der **TransportKostenRechner** stellt eine Methode zur Berechnung der Transportkosten anhand der Anzahl gekaufter Artikel und der Transportkostenbasis des **Haendlers** bereit. Hierdurch können **Haendler** allgemein behandelt werden und müssen nicht in Großhändler, lokale Händler, o.a. unterschieden werden. Zurzeit ist eine Berechnung pro Artikel oder pro Bestellung vorgesehen. Bei einer Berechnung pro Artikel wird die Transportkostenbasis mit der Anzahl Artikel multipliziert. Wenn die Transportkosten pro Bestellung berechnet werden, ist die Transportkostenbasis die Pauschale für den Transport.

Die Dienste führen unter anderem allgemeine CRUD-Aufgaben (Create, Read, Update, Delete) zur ihren jeweiligen Modelklassen aus. Da es sich hierbei um Standartaufgaben handelt, werden diese nicht näher erläutert.

Der **Gerichtdienst** übernimmt das Parsen der Eingabedateien „Beliebteste Gerichte“ und „Vom Koch bekannte Gerichte“. Die Dateinamen sind hierauf natürlich nicht festgelegt, sondern werden als Parameter übergeben. Außerdem kennt er Sortieralgorithmen für Listen von **Gerichten**, z.B. Sortierung nach günstigstem Preis oder beliebtesten Gerichten. Da diese Algorithmen als Liste vorliegen, ist auch eine Kombination möglich. Beispielsweise eine erste Sortierung nach Beliebtheit und innerhalb derer eine Sortierung nach Preis. Bei gleichem Preis, wäre das beliebtere Gericht weiter vorne in der Ergebnisliste.

Der **HaendlerDienst** kennt alle **Haendler** des Systems. Er stellt dazu eine Methode zum Erzeugen von Haendlerinstanzen zur Verfügung. Von einer statischen Erzeugungsmethode in der Klasse **Haendler** wird abgesehen, um keine Kopplung zwischen der Datenschicht und dem **HaendlerDienst** zu erzeugen. Da es sich ausschließlich um interne Kommunikation im System handelt, kann darauf vertraut werden, dass in keinem Verarbeitungsprozess **Haendler** am **HaendlerDienst** vorbei erzeugt werden. Der **HaendlerDienst** kannprüfen, obeine Zutat von einem ihm bekannten **Haendler** verkauft wird, einen **Haendler** anhand seines Namens finden und aus einer Liste von Gerichten die herausfiltern, von denen nicht alle Zutaten gekauft werden können. Zudem übernimmt er das Parsen der Händlerdateien, da hierbei Händler erzeugt werden müssen und dies ebenfalls eine Aufgabe des Dienstes ist.