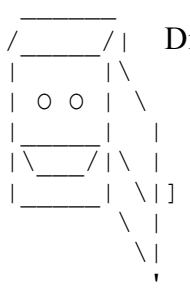
Übung 6

Hilfe, mein Kühlschrank spricht zu mir – Teil 3 (15 Punkte)



Diesmal mit Grafischer Benutzeroberfläche (GUI) und Design Patterns.

Produkt	Anzahl
Salat	1
Quark	2
Lachs	1
Milch	1
Bier	1
Senf	0
Käse	2
Wurst	2
Eier	10
Butter	1
Ketchup	0
Cola	2
Joghurt	4
Kaviar	0

*Abbildung: Erläuterung der **vorgegebenen** Klasse* `FridgeContentTable`

Aufgabe 1: Observer-Pattern im Model-View-Controller (9 Punkte)

Mit Swing (http://de.wikipedia.org/wiki/Swing_(Java)) können in Java einfach Grafische Benutzeroberflächen (GUIs) programmiert werden: Der vorgegebene Code `FridgeContentTable` (ein `JFrame`, Oberklasse aller einfachen Swing-GUIs), enthält die Tabellenanzeige `JTable`, die ihrerseits über ein selbst implementiertes `SimplePropertiesTableModel` auf den

Kühlschrankinhalt **lesend** zugreift (die Tabelle soll auch im Weiteren nur angezeigt, der Inhalt aber nicht über die GUI vom Benutzer geändert oder umsortiert werden). Die GUI greift **nicht direkt** auf unsere bekannten Inventory-Properties zu, sondern nur **indirekt** über ein Modell-Objekt, das die (und nur die) für die `JTable` notwendigen Anzeigedaten bereit stellt. Das in `template_ue06.rar` gegebene `SimplePropertiesTableModel` ist eine Unterklasse von `javax.swing.table.AbstractTableModel`, welches wiederum das Interface `javax.swing.table.TableModel` mit nützlichen Default-Implementierungen füllt und so dem Programmierer lästige Arbeit abnimmt: Lediglich die Methoden `getValueAt(int row, int column)`, `getRowCount()` und `getColumnCount()` mussten noch implementiert werden. Um auf die Bestandsdaten zuzugreifen, hält das Modell eine Kopie der Properties-Werte im String-Array `value`, mit einer Übersetzungs-Hashtabelle `propertyMap`, die von den Property-Namen auf das String-Array mit den Tabelleninhalten abbildet.

Die GUI-Implementierung folgt dem Architekturmuster **Model-View-Controller** (http://de.wikipedia.org/wiki/Model_View_Controller), das die Trennung von Daten (**Model**), Anzeige (**View**) und Programmlogik (**Controller**) propagiert. Hauptziel ist Wartbarkeit und Modularisierung des Programmcodes grafischer Benutzeroberflächen. In unserer Anwendung ist der Controller das (bisherige) Kühlschrank-Programm, Model ist das `SimplePropertiesTableModel`, View die `FridgeContentTable` (Swing-GUI).

Zur eigentlichen Aufgabe: Die in der GUI angezeigte Tabelle wird automatisch bei Änderungen des Kühlschrankinhalts durch die Textbefehle des Benutzers aktualisiert. Während das Aktualisieren zwischen View und Model bereits durch die in Java vorgegebene Default-Implementierung mittels Observer-Pattern erfolgt (`TableModelListener`; fire-Methoden in der Klasse `AbstractTableModel`), ist die Integration zwischen `SimplePropertiesTableModel` und Inventory-Properties nur provisorisch mit direktem Methodenaufruf in `SimpleTableModelFridge` realisiert (`tableModel.updateValue()` am Ende von `.modifyStock()`). Schreibe zwei neue Klassen `AdvancedPropertiesTableModel` (kann von der gegebenen Klasse `SimpleTableModelFridge` abgeleitet werden) und `AdvancedTableModelFridge` (sollte wie `SimpleTableModelFridge` direkt von `FridgeEx2` abgeleitet werden), so dass auch zwischen Model (`AdvancedPropertiesTableModel`) und Controller (Fridge-Properties) ein Observer-Pattern die Aktualierungen am Modell vornimmt, wenn der Benutzer den Bestand (=Properties) ändert.

Hinweise: Verwende dazu das in Java bereits implementierte generische Observer-Pattern (`java.util.Observer` bzw. `java.util.Observable`). Implementiere eine neue Klasse `ObservableProperties` als Unterklasse von `java.util.Observable`. Achtung: Da multiple Vererbung in Java nicht möglich ist, wurde für die `java.util.Properties` ein Interface `PropertiesIF` extrahiert. Die bisher verwendeten Properties wurden in `SimpleProperties` überführt, die das `PropertiesIF`-Interface implementieren. `ObservableProperties` sollte ebenfalls `PropertiesIF` implementieren. Es sollte, analog zu `SimpleProperties`, ein `Properties` Objekt enthalten, an das die Aufrufe delegiert werden. Verwende `ObservableProperties` in `AdvancedTableModelFridge` und weise `AdvancedPropertiesTableModel` entsprechend die Rolle des Observer zu. Der Aufruf `tableModel.updateValue()` wie am Ende von `SimpleTableModelFridge.modifyStock()` ist dann nicht mehr nötig!

Aufgabe 2: Command-Pattern (6 Punkte)

Implementiere mit Hilfe eines Command-Patterns (http://en.wikipedia.org/wiki/Command_pattern) den neuen Kühlschrank-Befehl

["Alles"] "Rückgängig",

der die jeweils letzte Aktion bzw. alle Aktionen auf dem Kühlschrank**bestand** rückgängig macht. Erweitere dazu die Klasse `AdvancedTableModelFridge` entsprechend und überschreibe wo nötig Methoden der Superklassen.

Inhalt von template_ue06.rar enthält folgende neue Klassen:

`FridgeContentTable.java` – die Swing GUI

`SimpleTableModelFridge.java` – abgeleitet von `FridgeEx2`, erweitert um:

- `SimplePropertiesTableModel`
- die Swing GUI
- erweiterte `modifyStock` Methode:
- `this.tableModel.updateValue(product, this.formatNumber(currentAmount));`

`SimplePropertiesTableModel.java` – einfache `TableModel`-Implementierung

`PropertiesIF.java` – Interface, das den Zugriff auf `Properties` delegiert

`SimpleProperties.java` – Basisimplementierung des `PropertiesIF`-Interfaces, entspricht der `java.util.Properties`-Implementierung

Die restlichen Klassen sind aus der Musterlösung zu Übung 5 und wurden, falls nötig, leicht modifiziert für die Verwendung von `PropertiesIF`, insbesondere der Konstruktor von `Fridge`().