Objektorientierte Programmiertechniken
LVA 185.A01, VU, 3 Ects, 2013/2014 W

9. Übungsaufgabe

Themen:

Design-Patterns, aspektorientierete Programmierung

Termine:

Ausgabe: 11.12.2013
Abgabe: 08.01.2014, 12:00 Uhr

Abgabeverzeichnis:

Aufgabe9

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf den Abschnitten 4.4, 5.1 und 5.2

Aufgabe

Welche Aufgabe zu lösen ist:

Die Automatisierung macht auch vor dem Mixen von Cocktails keinen Halt (Roböxotica). Jeder Cocktail hat einen eindeutigen Namen, eine Gesamtmenge (in Milliliter als ganze Zahl) und die Namen von bis zu drei verschiedenen Flüssigkeiten. Bei einem alkoholischen Cocktail ist zusätzlich der Alkoholgehalt bekannt (die gewichtete Summe des Alkoholgehalts aller Zutaten). Weiters gibt es noch Cocktails, die mit Crusheis serviert werden (dabei wird die Menge Eis in Gramm (Gleitkommazahl) mit angegeben), oder Cocktails die heiß serviert werden (dabei wird die Temperatur in Grad (ganze Zahl) mit angegeben). Die Cocktails werden von Cocktailrobotern gemixt, die nur eine bestimmte Art von Cocktail mischen können. Einfache Roboter mixen Cocktails mit bis zu drei unterschiedlichen Flüssigkeiten aus einer Menge von 20 verschiedenen Flüssigkeiten. Ein Roboter speichert von Flüssigkeiten die Namen und die

verfügbaren Mengen (in ganzahligen Milliliter) und von alkoholischen Flüssigkeiten zusätzlich die Alkoholgehalte. Dann gibt es noch Roboter, die zusätzlich Eis hinzufügen können (ein Roboter speichert die verfügbare Menge in Gramm (Gleitkommazahl)) und Roboter die den Cocktail erhitzen können. Auf einer Cocktailkarte sind alle verfügbaren Cocktails mit allen ihren Zutaten inklusive ihrer Mengen (in ganzahligen Milliliter) und den Informationen, ob sie heiß oder mit Eis gekühlt sind, aufgelistet. Eine Bestellung eines Gasts enthält eine beliebige Anzahl an Namen von Cocktails, die auch mehrfach vorkommen können. Als Ergebnis erhält man ein Serviertablett mit Cocktails.

Schreiben Sie ein Javaprogramm, das eine automatisierte Cocktailbar simuliert. Entwickeln Sie Klassen(hierarchien) um Cocktails, Cocktailroboter, Bestellungen, Serviertabletts und eine Cocktailkarte darzustellen.

Folgende Methoden und Funktionalitäten sollen unterstützt werden:

- Eine Methode *content* für ein Serviertablett, die Beschreibungen der auf dem Serviertablett stehenden Cocktails auf die Standardausgabe ausgibt.
- Eine Methode *list* für eine Bestellung, die alle Positionen der Bestellung in der Standardausgabe auflistet.

Schreiben Sie eine Klasse Test, die eine Cocktailkarte erstellt, die mehrere Bestellungen erstellt, die Bestellungen auf die Standardausgabe ausgibt, entsprechende Cocktails von den Robotern mixt, ein Serviertablett befüllt und den Inhalt des Serviertabletts anzeigt.

Daneben soll die Datei *Test.java* wie gewohnt als Kommentar eine kurze, aber verständliche Beschreibung der Aufteilung der Arbeiten auf die einzelnen Gruppenmitglieder enthalten – wer was gemacht hat.

Da in dem Programm viele Objekte erzeugt werden, soll die Anzahl der dynamisch erzeugten Objekte gezählt werden. Kapseln sie das Zählen der Objekte in einem Aspekt und geben Sie die Anzahl am Ende des Programms am Bildschirm aus.

Warum die Aufgabe diese Form hat:

Diese Aufgabe enthält Details, die den Einsatz bestimmter Entwurfsmuster nahelegen. Durch mehrere Arten von Cocktailrobotern, Cocktails, Flüssigkeiten, etc. ergibt sich eine umfangreiche Typstruktur. Alle Schwierigkeiten in der Aufgabe sollten mit bekannten Mitteln leicht zu lösen sein. Das gibt Ihnen zum Abschluss der Laborübung Gelegenheit zu zeigen, dass Sie schöne, einfache und gut wartbare Programme schreiben können.

Was im Hinblick auf die Beurteilung zu beachten ist:

Die insgesamt 100 für diese Aufgabe erreichbaren Punkte sind folgendermaßen

auf die zu erreichenden Ziele aufgeteilt:

Entwurfsmuster richtig verwendet, sinnvolle Klassenhierarchien	30 Punkte
Aspektorientierte Programmierung richtig eingesetzt	20 Punkte
Lösung wie vorgeschrieben und sinnvoll getestet	20 Punkte
Geforderte Funktionalität vorhanden (so wie in Aufgabenstellung beschrieben)	15 Punkte
Zusicherungen richtig und sinnvoll eingesetzt	10 Punkte
Sichtbarkeit auf so kleine Bereiche wie möglich beschränkt	5 Punkte

Kräftige Punkteabzüge gibt es für

- Verletzungen des Ersetzbarkeitsprinzips bei Verwendung von Vererbungsbeziehungen,
- den Verzicht auf mögliche Untertypbeziehungen,
- fehlende, falsche, nichtssagende und unverständliche Zusicherungen,
- unnötigen Code und mehrfache Vorkommen gleicher oder ähnlicher Code-Stücke,
- komplizierte und schwer verständliche Programmstrukturen, die sich leicht aus nicht oder falsch eingesetzten Entwurfsmustern ergeben,
- unpassende Sichtbarkeit.

Wie die Aufgabe zu lösen ist:

Es ist wichtig, alle Untertypbeziehungen zu finden und die Ersetzbarkeit auf geeignete Weise (durch durchdachte Zusicherungen, aber auch durch Testfälle) sicherzustellen. Vererbungsbeziehungen, die nicht gleichzeitig auch Untertypbeziehungen sind, müssen unbedingt vermieden werden. Andererseits sollen überall Untertypbeziehungen hergestellt werden, wo das möglich ist. Überlegen Sie sich, wo Sie (abstrakte) Klassen und Interfaces verwenden. Beachten Sie die Sichtbarkeit. Benutzen Sie Generizität und vorgefertigte Containerklassen, wo dies sinnvoll erscheint.

Diese Aufgabe enthält wahrscheinlich weniger inhaltliche Schwierigkeiten als vorangegangene Aufgaben. Gerade deswegen sollten Sie ein schönes, einfaches und gut wartbares Programm abliefern, da solche Qualitätskriterien stärker in die Beurteilung einfließen als gewohnt.

Achten Sie darauf, dass Sie den Programm-Code so klein und einfach wie möglich halten. Jede unnötige Anweisung kann eine versteckte Fehlerquelle darstellen.

Was im Hinblick auf die Abgabe zu beachten ist:

Dieses Programm wird nicht mit dem Standardcompiler *javac*, sondern mit dem Compiler *ajc* übersetzt. Damit die Übersetzung funktioniert, müssen alle

Klassen gemeinsam mit dem Aufruf übersetzt werden, z.B. mit einem Aufruf *ajc *.java*. Wenn Sie unbedingt wollen, können Sie Packages verwenden, aber dann müssen Sie auch die Dateien in Unterverzeichnissen in den *ajc*-Aufruf einbeziehen. Testen Sie ihre Lösung auf der g0, da sich die Version von *ajc* auf der g0 eventuell von der Version unterscheidet, die Sie zu Hause verwenden. Üblicherweise ist bei einer Standardinstallation von Java *ajc* nicht inkludiert, laden Sie sich für zu Hause *ajc* von der Eclipse Webseite. Dort findet sich auch ein detailierter Programming Guide. Gerade für diese Aufgabe ist es besonders wichtig, dass Sie (abgesehen von geschachtelten Klassen) nicht mehr als eine Klasse in jede Datei geben und auf aussagekräftige Namen achten. Sonst ist es schwierig, sich einen Überblick über Ihre Klassen und Interfaces zu verschaffen. Achten Sie darauf, dass Sie keine Java-Dateien abgeben, die nicht zu Ihrer Lösung gehören (alte Versionen, Reste aus früheren Versuchen, etc.).