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

6. Übungsaufgabe

Themen:

kovariante Probleme, mehrfaches dynamisches Binden

Termine:

Ausgabe: 20.11.2013
Abgabe: 04.12.2013, 12:00 Uhr

Abgabeverzeichnis:

Aufgabe6

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf den Abschnitten 3.3.3 und 3.4

Aufgabe

Welche Aufgabe zu lösen ist:

Entwickeln Sie einen Teil einer Anwendung zur Verwaltung der in einem Lager gelagerten Waren. Ein Lager kann folgende Arten von Lagerplätzen anbieten:

- Lagerplatz mit Raumtemperatur,
- gekühlter Lagerplatz,
- tiefgekühlter Lagerplatz bei minus 18 Grad,
- tiefgekühlter Lagerplatz bei minus 30 Grad.

Jedes Lager hat eine fixe Anzahl an Lagerplätzen aller vier Arten.

Ebenso gibt es Waren, die

- bei Raumtemperatur,
- gekühlt,
- tiefgekühlt bei minus 18 Grad oder
- tiefgekühlt bei minus 30 Grad

gelagert werden sollen. Falls alle Lagerplätze mit Raumtemperatur belegt sind, dürfen ausnahmsweise bei Raumtemperatur zu lagernde Waren an einem gekühlten Lagerplatz gelagert werden. Falls alle Tiefkühllagerplätze mit minus 18 Grad belegt sind, dürfen ausnahmsweise alle tiefgekühlt zu lagernden Waren an einem tiefgekühlten Lagerplatz bei minus 30 Grad gelagert werden. Jede Ware ist mit einer eindeutigen Seriennummer und einem Namen versehen.

Die Lagerverwaltungsanwendung muss zumindest folgende Methoden zur Verfügung stellen:

- *store* trägt ein Warenobjekt in ein Lager ein und liefert einen Fehlercode zurück, der angibt, ob das Einlagern erfolgreich war.
- *remove* entfernt ein Warenobjekt mit einer bestimmten Seriennummer aus einem Lager und liefert die Referenz auf das Warenobjekt zurück. Falls dieses Warenobjekt nicht existiert, wird *null* zurückgeliefert.
- *inventar* zeigt die Seriennummern und Namen aller in einem Lager gelagerten Waren auf dem Bildschirm an.
- *utilization* zeigt den Grad der Auslastung (wieviele der verfügbaren Lagerplätze belegt sind) für ein gesamtes Lager und jeweils für jede Art der Lagerplätze auf dem Bildschirm an.

Die Klasse *Test* soll wie üblich die wichtigsten Normal- und Grenzfälle überprüfen und die Ergebnisse in allgemein verständlicher Form darstellen. Dabei sind Instanzen aller in der Lösung vorkommenden Typen zu erzeugen. Auch für Lager sind eigene Objekte zu erzeugen und mindestens 3 unterschiedliche Lager (jeweils mit allen Arten von Lagerplätzen) zu testen. Testfälle sind so zu gestalten, dass sich deklarierte Typen von Variablen im Allgemeinen von den dynamischen Typen ihrer Werte unterscheiden.

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.

In der Lösung der Aufgabe dürfen Sie folgende Sprachkonzepte nicht verwenden:

- dynamische Typabfragen *getClass* und *instanceof* sowie Typumwandlungen;
- bedingte Anweisungen wie *if*- und *switch*-Anweisungen sowie bedingte Ausdrücke (= Ausdrücke der Form *x*?*y*:*z*), die Typabfragen emulieren (d.h., zusätzliche Felder eines Objekts, die einen Typ simulieren und abfragen sind nicht erlaubt; z.B. ein *enum* der Warenarten ist nicht sinnvoll weil Abfragen darauf nicht erlaubt sind; bedingte Anweisungen, die einem

anderen Zweck dienen, sind dagegen schon erlaubt);
- Werfen und Abfangen von Ausnahmen.

Bauen Sie Ihre Lösung stattdessen auf (mehrfaches) dynamisches Binden auf.

Warum die Aufgabe diese Form hat:

Die Aufgabe lässt Ihnen viel Entscheidungsspielraum. Es gibt zahlreiche sinnvolle Lösungsvarianten. Die Form der Aufgabe legt die Verwendung kovarianter Eingangsparametertypen nahe, die aber tatsächlich nicht unterstützt werden. Daher wird mehrfaches dynamisches Binden (durch simulierte Multi-Methoden bzw. das Visitor-Pattern) bei der Lösung hilfreich sein. Alternative Techniken, die auf Typumwandlungen und dynamischen Typabfragen beruhen, sind ausdrücklich verboten. Durch dieses Verbot wird die Notwendigkeit für dynamisches Binden noch verstärkt. Sie sollen sehen, wie viel mit dynamischem Binden möglich ist, aber auch, wo ein übermäßiger Einsatz zu Problemen führen kann.

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:

(mehrfaches) dynamisches Binden richtig verwendet, sinnvolle minimale Typhierarchie, möglichst geringe Anzahl an Methoden und gute Wiederverwendung	40 Punkte
Lösung wie vorgeschrieben und sinnvoll getestet	20 Punkte
Zusicherungen richtig und sinnvoll eingesetzt	15 Punkte
Geforderte Funktionalität vorhanden (so wie in Aufgabenstellung beschrieben)	15 Punkte
Sichtbarkeit auf so kleine Bereiche wie möglich beschränkt	10 Punkte

Schwerpunkte bei der Beurteilung liegen auf der selbständigen Entwicklung geeigneter Untertypbeziehungen und dem Einsatz (mehrfachen) dynamischen Bindens. Kräftige Punkteabzüge gibt es für

- die Verwendung der verbotenen Sprachkonzepte,
- die Verwechslung von statischem und dynamischem Binden (insbesondere die Verwechslung überladener Methoden mit Multimethoden),
- Verletzungen des Ersetzbarkeitsprinzips (also Vererbungsbeziehungen, die keine Untertypbeziehungen sind)
- und nicht der Aufgabenstellung entsprechende oder falsche Funktionalität des Programms.

Punkteabzüge gibt es unter anderem auch für mangelhafte Zusicherungen, schlecht gewählte Sichtbarkeit und unzureichendes Testen (z.B. wenn

grundlegende Funktionalität nicht überprüft wird).

Wie die Aufgabe zu lösen ist:

Vermeiden Sie Typumwandlungen, dynamische Typabfragen und verbotene bedingte Anweisungen von Anfang an, da es schwierig ist, diese aus einem bestehenden Programm zu entfernen. Akzeptieren Sie in einem ersten Entwurf eher kovariante Eingangsparametertypen bzw. Multimethoden und lösen Sie diese dann so auf, dass Java damit umgehen kann (unbedingt vor der Abgabe, da sich sonst sehr schwere Fehler ergeben).

Halten Sie die Anzahl der Klassen, Interfaces und Methoden möglichst klein und überschaubar. Durch die Aufgabenstellung ist eine große Anzahl an Klassen und Methoden ohnehin kaum vermeidbar, und durch weitere unnötige Strukturierung oder Funktionalität könnten Sie leicht den Überblick verlieren.

Es gibt mehrere sinnvolle Lösungsansätze. Bleiben Sie bei dem ersten von Ihnen gewählten sinnvollen Ansatz und probieren Sie nicht zu viele Ansätze aus, damit Ihnen nicht die Zeit davonläuft. Unterschiedliche sinnvolle Ansätze führen alle zu etwa demselben hohen Implementierungsaufwand.

Was im Hinblick auf die Abgabe zu beachten ist:

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.).