Objektorientierte Programmiertechniken
LVA 185.A01, VU 2.0, 2011 W

7. Übungsaufgabe

Themen:

kovariante Probleme, mehrfaches dynamisches Binden

Termine:

Ausgabe:	07.12.2011
reguläre Abgabe:	14.12.2011, 13:45 Uhr
nachträgliche Abgabe:	21.12.2011, 13:45 Uhr

Abgabeverzeichnis:

Aufgabe7

Programmaufruf:

java Test

Grundlage:

Skriptum bis Seite 135, Schwerpunkt auf Abschnitt 3.4

Aufgabe

Welche Aufgabe zu lösen ist:

Der Tiergarten Springbrunn hat seit einiger Zeit mit Platzproblemen zu kämpfen. Um diese zu lösen, sollen einige Tiere in neue Gehege übersiedeln.

Implementieren Sie einen Teil der Anwendung zur Planung der Übersiedelung. Dabei sollen Tiere in einem Spezialtransporter mit unterschiedlichen Abteilen, die nach den Bedürfnissen der Tiere ausgestattet sind, untergebracht werden. Folgende Tiere müssen unterstützt werden:

- Großwild: Elefant und Giraffe,
- Reptilien wie Mamba, Python, Riesenschildkröte und Zwergschildkröte,
- Delfin, Schwertfisch und Seehund, die zu den Meereslebewesen zählen.

Jedes Tier hat einen Namen. Eine Liste aller in und am Transporter befindlichen Tiere (direkt im Transporter, in einem seiner Anhänger oder angebunden am Transporter bzw. einem seiner Anhänger) ist durch Aufruf der Methode *list* im Transporter zu ermitteln.

Großwild kann aufgrund der begrenzeten Ladefläche des Transporters nicht eingeladen werden. Solche Tiere können nur einmalig fix (mit einem Strick) an die Anhängekupplung eines Transporters oder eines Anhängers angebunden werden. Wiederholtes An- und Abbinden dieser Tiere ist aufgrund der komplexen Logistik nicht möglich.

Der Aufruf der Methode *load* in einem geeigneten Objekt stellt eine Verbindung zwischen einem noch nicht verladenen Tier bzw. noch nicht angekoppelten Anhänger und einem freien Abteil bzw. einer freien Anhängekupplung her. Die Methode liefert einen Wahrheitswert zurück, der besagt, ob das Tier untergebracht werden konnte. Durch Aufruf der Methode *unload* wird das belegte Abteil bzw. die belegte Anhängekupplung wieder freigegeben. Die Methode gibt eine Referenz auf das ausgeladene Objekt zurück.

Ein Transporter enthält genauso wie ein Anhänger eine festgelegte (zur Laufzeit nicht änderbare) Menge von

- fest angebundenem Großwild (Elefant bzw. Giraffe, ohne Unterstützung für *load* und *unload*),
- Anhängekupplungen,

sowie eine festgelegte (zur Laufzeit nicht änderbare) Menge der folgenden Abteile:

- Terrarien sind ausschließlich für Reptilien ausgelegt. In der Regel wird für jede Reptilienart ein speziell eingerichtetes Terrarium benötigt. Ausnahme: Mambas passen auch in Python-Terrarien und Zwergschildkröten auch in Riesenschildkröten-Terrarien.
- Unterschiedliche Arten von Wassertanks, wobei Seehunde nur in ein gemischtes Wasser-/Land-Abteil aufgenommen werden dürfen. Alle anderen Meereslebewesen können sowohl in Wasser-/Land-Abteile, als auch in reine Wassertanks kommen.

Tiere, Abteile und Anhänger sollen jeweils als Objekte unterschiedlicher Typen dargestellt werden. Zur Unterscheidung zwischen verschiedenen Arten von Objekten soll nur die in jedem Objekt vorhandene dynamische Typinformation dienen.

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, verschiedene Einladeversuche anzustellen (auch solche, wo ein Tier nicht untergebracht werden kann) und zur Kontrolle wiederholt über *list* die

eingeladenen Tiere zu ermitteln. Einladeversuche sind so zu gestalten, dass man die Art der Tiere, Abteile, etc. nicht schon anhand der deklarierten Typen feststellen kann.

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

- dynamische Typabfragen über *getClass* und *instanceof* sowie Typumwandlungen,
- bedingte Anweisungen wie *if*- und *switch*-Anweisungen sowie bedingte Ausdrücke (= Ausdrücke der Form x ? y : z) (ausgenommen Vergleiche auf *null*),
- Schleifenkonstrukte, deren einziger Zweck in der Emulation bedingter Anweisungen besteht,
- das 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 das Verbot bedingter Anweisungen und bedingter Ausdrücke 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. Anders als bei den vorangegangenen Aufgaben gibt es keine vorgeschriebenen Testfälle, die zur Selbstkontrolle genutzt werden könnten.

Was im Hinblick auf die Beurteilung zu beachten ist:

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 Ã¼berladener 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 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 die Verwendung von Multimethoden einen schwerwiegenden Fehler darstellt).

Ein vollständiger Verzicht auf bedingte Anweisungen kann ungeahnte Schwierigkeiten nach sich ziehen. Verzichten Sie trotzdem auf bedingte Anweisungen und umgehen Sie die Schwierigkeiten mittels kreativer Lösungen, vor allem durch dynamisches Binden. Bedingte Anweisungen, die nur auf *null* vergleichen, sind zwar erlaubt, sollen aber möglichst vermieden werden. Eine bedingte Anweisung lässt sich durch eine Schleife mit Abbruchbedingung simulieren, aber genau solche Schleifen sind nicht erlaubt. Verwenden Sie Schleifenkonstrukte nur in Fällen, in denen tatsächlich Schleifenrümpfe wiederholt ausgeführt werden sollen.

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 bald 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 Implementierungsaufwand.

Was im Hinblick auf die Abgabe zu beachten ist:

Verzichten Sie wie üblich auf die Verwendung von packages und Verzeichnissen innerhalb des Abgabeverzeichnisses. 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.).