7. Programmieraufgabe

Programmierparadigmen

LVA-Nr. 194.023
2023/2024 W
TU Wien

Themen:

kovariante Probleme, mehrfaches dynamisches Binden, aspektorientierte Programmierung

Ausgabe:

27. 11. 2023

Abgabe (Deadline):

11. 12. 2023, 14:00 Uhr

Abgabeverzeichnis:

`Aufgabe7`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 4.3.3, 4.4 und 4.6

Kontext A

Ein Ameisenforschungsinstitut führt Forschungen an vielen europäischen und tropischen Ameisenarten in seinen eigenen Formicarien durch. Europäische Ameisen stellen keine Anforderungen an die Regelung des Klimas eines Formicariums. Tropische Ameisen benötigen eine Regelung des Klimas (Temperatur und Luftfeuchtigkeit). Ameisen bilden unterschiedlich große Staaten und benötigen für ihre Haltung daher unterschiedlich große Formicarien. Abhängig von der Art werden die Ameisen in folgenden unterschiedlich klimatisierten und unterschiedlich großen Formicarien gehalten:

- Formicarium ungeregelt, klein
- Formicarium ungeregelt, mittel
- Formicarium ungeregelt, groß
- Formicarium geregelt, klein
- Formicarium geregelt, mittel
- Formicarium geregelt, groß

Von jedem Formicarium ist der Preis bekannt. Von jedem Formicarium ist weiters bekannt, ob es frei ist, oder wenn es belegt ist, mit welcher Ameisenart es belegt ist. Je nach Art können Ameisen ausschließlich in Formicarien mit oder ohne Klimaregelung gehalten werden (europäisch → ungeregelt, tropisch → geregelt). Abhängig von der Größe des Ameisenstaates benötigen die Ameisen ein kleines, ein mittleres oder ein großes Formicarium. Ist kein freies kleines Formicarium für einen kleinen Ameisenstaat vorhanden, darf dieser ausnahmsweise in einem mittleren Formicarium gehalten werden. Ist kein freies mittleres Formicarium für einen mittleren Ameisenstaat vorhanden, darf dieser ausnahmsweise in einem großen Formicarium gehalten werden. Die vorhandenen freien Formicarien werden in einer Inventarliste verwaltet.

Welche Aufgabe bezüglich Kontext A zu lösen ist

Entwickeln Sie ein Programm zur Verwaltung der Formicarien und der Ameisenstaaten des Ameisenforschungsinstitut. Zumindest folgende Methoden sind zu entwickeln:

- `addForm` fügt ein neues Formicarium zur Inventarliste hinzu.
- `deleteForm` löscht ein defektes Formicarium aus der Inventarliste.
- `assignForm` gibt ein passendes Formicarium für einen Ameisenstaat zurück und entfernt dieses aus der Inventarliste. Falls kein passendes Formicarium existiert, wird `null` zurückgeliefert.

- `returnForm` löscht die Zuordnung eines Formicariums zu einem Ameisenstaaten und fügt dieses Formicarium zur Inventarliste hinzu (falls ein Ameisenstaaten ausgestorben ist oder in ein anderes Formicarium übersiedelt werden soll).
- `priceFree` zeigt die Summe der Preise aller Formicarien der Inventarliste auf dem Bildschirm an.
- `priceOccupied` zeigt die Summe der Preise aller Formicarien des Ameisenforschungsinstituts, die von Ameisenstaaten belegt sind, auf dem Bildschirm an.
- `showFormicarium` zeigt alle Formicarien der Inventarliste mit allen Informationen auf dem Bildschirm an.
- `showAnts` zeigt alle Ameisenstaaten eines Ameisenforschungsinstitut mit allen Informationen (insbesondere in welchem Formicarium sich ein Ameisenstaat befindet) auf dem Bildschirm an.

Die Formicarien dürfen nur in einer einzigen Inventarliste pro Forschungsinstitut gespeichert werden. Es ist nicht zulässig unterschiedliche Inventarlisten (etwa getrennt für Formicarien ohne bzw. mit Klimaregelung oder kleine, mittlere und große Formicarien) zu führen.

nur eine einzige Inventarliste

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 Ameisenforschungsinstitute mit einer Inventarliste und Ameisenstaaten sind eigene Objekte zu erzeugen, und mindestens 3 unterschiedliche Institute sind zu testen. Testfälle sind so zu gestalten, dass sich deklarierte Typen von Variablen im Allgemeinen von den dynamischen Typen ihrer Werte unterscheiden.

Zusätzlich soll die Klasse `Test.java` als Kommentar eine kurze, aber verständliche Beschreibung der Aufteilung der Arbeiten auf die einzelnen Gruppenmitglieder enthalten – wer hat was gemacht.

Aufgabenaufteilung beschreiben

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 (also 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 Formicariumtypen 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.

keine Typabfragen erlaubt

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

Kontext B

Für die Lösung dieser Aufgabe wird eine Form des Visitorpatterns benötigt. Es ist leicht ersichtlich, dass dadurch eine große Anzahl an Methoden spezifiziert werden muss. Die Anzahl der Methodenaufrufe zur Laufzeit ist aber nicht einfach erkennbar. Die Anzahl der Methodenaufrufe kann aber einfach mittels aspektorientierter Programmierung ermittelt werden.

Welche Aufgabe bezüglich Kontext B zu lösen ist

Ermitteln Sie die Anzahl der Aufrufe der Methode `assignForm`. Ermitteln Sie die Anzahl der Aufrufe aller zum Visitor gehörenden Methoden, das sind sowohl die Methoden der Elementklassen als auch die der Visitorklassen. Kapseln sie das Zählen der Methodenaufrufe in einem Aspekt und geben Sie die beiden vorher spezifizierten Zahlen am Ende des Programms nach allen anderen Testausgaben am Bildschirm aus.

Was im Hinblick auf die Beurteilung wichtig 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 — 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 kleinstmögliche Bereiche beschränkt — 5 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.

ajc statt javac

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 (`http://www.eclipse.org/aspectj/`). Dort findet sich auch ein detailierter Programming Guide (`http://www.eclipse.org/aspectj/doc/released/progguide/`).

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. Darüber hinaus geht es auch darum, den Umgang mit Aspekten zu üben.

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. Verwenden

Sie keine Umlaute in Dateinamen. Achten Sie darauf, dass Sie keine Java-Dateien abgeben, die nicht zu Ihrer Lösung gehören (alte Versionen, Reste aus früheren Versuchen, etc.).

keine Umlaute