7. Programmieraufgabe

Programmierparadigmen

LVA-Nr. 194.023
2024/2025 W
TU Wien

Themen:

kovariante Probleme, mehrfaches dynamisches Binden, Annotationen

Ausgabe:

02. 12. 2024

Abgabe (Deadline):

16. 12. 2024, 14:00 Uhr

Abgabeverzeichnis:

`Aufgabe7`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 4.3.3, 4.4 und 4.5

Kontext A

Ein Bürobetreiber will seine Büros CO_2-neutral mit Wärmepumpen beheizen. Er benötigt ein Programm, um seine Büroeinheiten und die dort installierten Wärmepumpen zu verwalten. Um die Aufgabe einfach zu halten, wird die Realität etwas aufgeweicht.

Der Bürobetreiber hat einen großen Bestand an Büroeinheiten. Diese sind entweder mit einer modernen Fußbodenheizung oder mit alten gusseisernen Rippenheizkörpern ausgestattet. Die Büroeinheiten sind in drei unterschiedlichen Größen vorhanden (klein, mittel und groß). Daher gibt es folgende Kombinationen:

- Büroeinheit mit Fußbodenheizung, klein
- Büroeinheit mit Fußbodenheizung, mittel
- Büroeinheit mit Fußbodenheizung, groß
- Büroeinheit mit Rippenheizkörper, klein
- Büroeinheit mit Rippenheizkörper, mittel
- Büroeinheit mit Rippenheizkörper, groß

Eine Fußbodenheizung wird ausschließlich mit einer Niedertemperatur-Wärmepumpe betrieben, die Rippenheizkörper können nur mit einer weniger effizienten Hochtemperatur-Wärmepumpe betrieben werden. Die Wärmepumpen stehen in drei Leistungsstufen zur Verfügung, mit kleiner, mittlerer und großer Leistung. Abhängig von der Größe einer Büroeinheit benötigen diese eine Wärmepumpe mit kleiner, mittlerer oder großer Leistung. Ist keine Wärmepumpe mit kleiner Leistung für eine kleine Büroeinheit verfügbar, darf diese ausnahmsweise mit einer Wärmepumpe mit mittlerer Leistung beheizt werden. Ist keine Wärmepumpe mit mittlerer Leistung für eine mittelgroße Büroeinheit verfügbar, darf diese ausnahmsweise mit einer Wärmepumpe mit großer Leistung beheizt werden. Die verfügbaren Wärmepumpen werden in einer Inventarliste verwaltet. Von jeder Wärmepumpe ist der Preis bekannt. Von jeder Büroeinheit ist bekannt, ob sie mit einer Wärmepumpe beheizt wird, und falls das der Fall ist, welche Leistung diese Wärmepumpe hat.

Welche Aufgabe bezüglich Kontext A zu lösen ist

Entwickeln Sie ein Programm zur Verwaltung der Wärmepumpen und der Büroeinheiten eines Bürobetreibers. Die zur Verfügung stehenden Wärmepumpen werden in einer Inventarliste verwaltet. Weiters existiert eine Liste aller Büroeinheiten mit den Informationen über die Art und Leistungsstärke der dort installierten Wärmepumpen oder ob keine Wärmepumpe installiert ist. Zumindest folgende Methoden sind zu entwickeln:

- `addHeatPump` fügt ein neue Wärmepumpe zur Inventarliste hinzu.

- `deleteHeatPump` löscht eine defekte Wärmepumpe aus der Inventarliste.
- `assignHeatPump` gibt eine passende Wärmepumpe für eine Büroeinheit zurück und entfernt diese aus der Inventarliste. Falls keine passende Wärmepumpe verfügbar ist, wird `null` zurückgeliefert.
- `returnHeatPump` löscht die Zuordnung einer Wärmepumpe zu einer Büroeinheit und fügt diese Wärmepumpe zur Inventarliste hinzu (falls zum Beispiel ein funktionierende Wärmepumpe ausgetauscht werden soll).
- `priceAvailable` zeigt die Summe der Preise aller Wärmepumpen der Inventarliste auf dem Bildschirm an.
- `priceInstalled` zeigt die Summe der Preise aller Wärmepumpen des Bürobetreibers, die in Büroeinheiten installiert sind, auf dem Bildschirm an.
- `showHeatPumps` zeigt alle Wärmepumpen der Inventarliste mit allen Informationen auf dem Bildschirm an.
- `showOffices` zeigt alle Büroeinheiten eines Bürobetreibers mit allen Informationen (insbesondere ob und welche Wärmepumpe in einer Büroeinheit installiert ist) auf dem Bildschirm an.

Die Wärmepumpen dürfen nur in einer einzigen Inventarliste pro Bürobetreiber gespeichert werden. Es ist nicht zulässig, unterschiedliche Inventarlisten (etwa getrennt für Niedertemperatur- und Hochtemperatur-Wärmepumpen oder mit kleiner, mittlerer und großer Leistung) 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 Bürobetreiber mit je einer Inventarliste und einer Liste von Büroeinheiten sind eigene Objekte zu erzeugen, und mindestens drei unterschiedliche Bürobetreiber sind zu testen. Testfälle sind so zu gestalten, dass sich deklarierte Typen von Variablen im Allgemeinen von den dynamischen Typen ihrer Werte unterscheiden.

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 Wärmepumpenarten 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: Datenextraktion

Hinter der Softwareentwicklung stehen Managementmaßnahmen, die Projekte vorantreiben und in gewünschte Richtungen lenken. Alle Maßnahmen beruhen auf Daten. Daten sind auch im Quellcode enthalten, müssen aber daraus extrahiert und aufbereitet werden, um verwendbar zu sein. Management benötigen wir auf allen Ebenen, vom Entwerfen einzelner Modularisierungseinheiten bis zu Optimierungen von Kapital, Personal und Finanzen. Entsprechend unterschiedlich sind die benötigten Daten. Konkret sind in dieser Aufgabe folgende Daten mittels Reflexion aus dem laufenden Programm zu extrahieren und übersichtlich darzustellen:

1. Namen aller zur Lösung dieser Aufgabe selbst definierten Klassen, Interfaces und Annotationen. Um Fehler bei Lösungsvarianten zu vermeiden, die auf falscher Verwendung des ClassLoaders oder eines Zugriffs auf das aktuelle Verzeichnis mit einem falschen Pfad beruhen, ist empfohlen, in der Test Klasse eine eigene Annotation zu verwenden, die beschreibt, welche Klassen, Interfaces und Annotationen zu diesem Programm gehören.

2. Eine Zuordnung zwischen den Namen aller zur Lösung dieser Aufgabe selbst definierten Klassen, Interfaces und Annotationen zum Namen jeweils eines Gruppenmitglieds, das für die Entwicklung der Einheit hauptverantwortlich ist.

3. Für jede Klasse und jedes Interface die Signaturen aller darin enthaltenen nicht-privaten Methoden und Konstruktoren sowie alle dafür geltenden Zusicherungen (getrennt nach Vor- und Nachbedingungen auf Methoden und Konstruktoren, sowie Invarianten und History-Constraints auf Klassen und Interfaces), einschließlich der Zusicherungen, die aus Obertypen übernommen („geerbt“) wurden.

4. Für jedes Gruppenmitglied die Anzahl der Klassen, Interfaces und Annotationen, für die dieses Gruppenmitglied hauptverantwortlich ist (als eine einzige ganze Zahl).

5. Für jedes Gruppenmitglied die Anzahl der Methoden und Konstruktoren in den Klassen (nur Klassen), für die dieses Gruppenmitglied hauptverantwortlich ist (als eine einzige ganze Zahl).

6. Für jedes Gruppenmitglied die Anzahl der Zusicherungen in den Klassen und Interfaces (samt Inhalten), für die dieses Gruppenmitglied hauptverantwortlich ist (als eine einzige ganze Zahl).

Welche Aufgabe bezüglich Kontext B zu lösen ist

Um die Datenextraktion zur Laufzeit zu ermöglichen, sind hauptverantwortliche Gruppenmitglieder in Form von Annotationen bei Definitionen von Klassen, Interfaces und Annotationen (Definitionen, nicht Verwendungen) anzugeben. Ebenso sind Zusicherungen (getrennt nach Arten) als

Annotationen an passenden Programmstellen anzugeben, nicht in Form von Kommentaren. Zusicherungen sollen (trotz Annotationen) für Menschen lesbare Texte darstellen, nicht ausführbaren Code.

Die unter „Kontext B“ beschriebenen Daten sind zur Laufzeit mittels Reflexion zu ermitteln und auszugeben. Achten Sie bitte darauf, dass alle „Kontext B“ betreffenden Daten visuell deutlich sichtbar getrennt nach denen zu „Kontext A“ ausgegeben werden.

In „Kontext B“ gibt es keine Einschränkungen bei der Verwendung der Sprachkonzepte von Java.

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 von Methoden und gute Wiederverwendung 30 Punkte
- Annotationen und Reflexion richtig verwendet 20 Punkte
- Lösung wie vorgeschrieben und sinnvoll getestet 20 Punkte
- Geforderte Funktionalität vorhanden (so wie in der 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 von 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.

Berücksichtigen Sie dabei auch „Kontext B“, das heißt, schreiben Sie keine Kommentare, sondern verwenden Sie selbst eingeführte Annotationen für Zusicherungen und zur Spezifikation der Hauptverantwortlichen, und sorgen Sie dafür, dass die genannten Daten über Reflexion aus dem laufenden Programm extrahierbar sind.

Kommentare sind nicht nötig. Alles, was bisher über Kommentare gemacht wurde, soll über Annotationen erledigt werden. Entsprechende Arten von Annotationen sind selbst zu definieren.

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.

Der Umgang mit Annotationen und Reflexion wird auf eine Weise geübt, die zwar das restliche Programm als Übungsobjekt heranzieht, aber keinen wesentlichen Einfluss darauf ausübt.

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-

keine Umlaute

Dateien abgeben, die nicht zu Ihrer Lösung gehören (alte Versionen, Reste aus früheren Versuchen, etc.).