6. Programmieraufgabe

Programmierparadigmen

LVA-Nr. 194.023
2023/2024 W
TU Wien

Themen:

dynamische Typinformation, homogene Übersetzung von Generizität, Annotationen

Ausgabe:

20. 11. 2023

Abgabe (Deadline):

4. 12. 2023, 14:00 Uhr

Abgabeverzeichnis:

`Aufgabe6`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 4.3.1, 4.3.2 und 4.5

Kontext A

Ein Ameisenforschungsinstitut führt Forschungen an polygynen, Kolonien bildenden Ameisenarten in seinen eigenen Formicarien durch. Das Institut will einige Daten über seine Formicarien sammeln und auswerten. Um die Aufgabe einfach zu halten, wird die Realität etwas aufgeweicht.

Das Institut betreibt mehrere Formicarien. Ein Formicarium hat einen eindeutigen Namen. In einem Formicarium kann ein Ameisenstaat gehalten werden. Ein Formicarium kann eines oder mehrere Nester enthalten. Von einem Formicarium wird dessen Name, der Name der Ameisenart und Information zu den Nestern gespeichert.

Ein Nest besteht aus einem Rahmen mit 2 Glasscheiben in einem fixen Abstand von 2 cm (Tiefe des Nests). Nester gibt es in unterschiedlichen Höhen und Breiten. Ein Nest hat eine eindeutige Nummer. Von jedem Nest wird diese eindeutige Nummer (ganze Zahl) und die Höhe und Breite in cm jeweils als Gleitkommazahl gespeichert. Aus der Höhe und Breite und der fixen Tiefe von 2 cm kann das Volumen in cm^3 berechnet werden.

Es gibt genau 2 unveränderliche Arten von Nestern, die sich in der Art der Regelung des Nestklimas unterscheiden: Nester mit Heizung zur Erhöhung der Temperatur und Nester mit Luftbefeuchter zur Erhöhung der Luftfeuchtigkeit. Es gibt keine Nester ohne Regelung und es gibt auch keine Nester, die sowohl eine Heizung als auch einen Luftbefeuchter besitzen. Von Nestern mit Heizung wird die Leistung der Heizung in Watt als ganze Zahl gespeichert. Von Nestern mit Luftbefeuchter wird das Volumen des Wasserbehälters (in cm^3) als Gleitkommazahl gespeichert.

Die Nester können mit 2 unterschiedlichen Materialen gefüllt werden: einem Sand-Lehmgemisch oder einer 2 cm dicken Gasbetonplatte. Die Füllung kann jederzeit getauscht werden (auch gegen die andere Art der Füllung). Bei einer Füllung mit Sand-Lehmgemisch wird das Gewicht in kg als Gleitkommazahl gespeichert. Bei einer Füllung mit Gasbetonplatte wird die Höhe und die Breite in cm jeweils als Gleitkommazahl gespeichert.

Welche Aufgabe bezüglich Kontext A zu lösen ist

Entwickeln Sie Java-Klassen bzw. Interfaces zur Darstellung von Formicarien mit Nestern mit wechselnden Nestfüllungen. Folgende Funktionalität soll unterstützt werden:

- Erzeugen eines Nests mit Nestnummer, Höhe, Breite und Leistung bei Nest mit Heizung oder Volumen bei Nest mit Luftbefeuchter.
- Auslesen der Höhe und der Breite.
- Auslesen der Leistung der Heizung oder des Volumens des Wasserbehälters.
- Einmaliges Setzen des Gewichts, wenn die Füllung aus einem Sand-Lehmgemisch besteht.

- Einmaliges Setzen der Höhe und der Breite bei Füllung mit einer Gasbetonplatte.
- Auslesen des Gewichts bei Füllung mit einem Sand-Lehmgemisch.
- Auslesen der Höhe und der Breite bei Füllung mit einer Gasbetonplatte.
- Ändern der Füllung eines Nests, wobei Informationen über frühere Füllungen verloren gehen.

Schreiben Sie eine Klasse `Formicarium`, die Informationen über Formicarien mit Nestern verwaltet und statistische Auswertungen über diese Formicarien ermöglicht. Jedes Formicarium hat einen unveränderlichen Namen. Folgende Methoden sollen unterstützt werden:

- Erzeugen eines Formicariums.
- Setzen, Löschen und Auslesen des Names der Ameisenart.
- Hinzufügen von Nestern zu einem Formicarium.
- Entfernen von Nestern eines Formicarium.
- Ändern der Informationen von Nestern wie oben beschrieben.
- Methoden zum Berechnen folgender (statistischer) Werte:
  - Das durchschnittliche Nestvolumen aller Nester eines Formicariums.
  - Das durchschnittliche Nestvolumen aller Nester mit Heizung eines Formicariums.
  - Das durchschnittliche Nestvolumen aller Nester mit Luftbefeuchter eines Formicariums.
  - Die durchschnittliche Leistung aller Nester mit Heizung eines Formicariums.
  - Das durchschnittliche Volumen des Wasserbehälters aller Nester mit Luftbefeuchter eines Formicariums.
  - Das durchschnittliche Gewicht des Sand-Lehmgemisches aller Nester eines Formicariums – alle Nester zusammen und zusätzlich aufgeschlüsselt nach der Art der Nestklimaregelung (Heizung oder Luftbefeuchter).
  - Das durchschnittliche Volumen der Gasbetonplatte aller Nester eines Formicariums – alle Nester zusammen und zusätzlich aufgeschlüsselt nach der Art der Nestklimaregelung (Heizung oder Luftbefeuchter).

Schreiben Sie eine Klasse `Institute`, die Informationen über alle Formicarien eines Instituts verwaltet. Jedes Institut hat einen unveränderlichen Namen. Folgende Methoden sollen unterstützt werden:

- Erzeugen eines Instituts (Objekt von `Institute`).

- Hinzufügen von Formicarien zu einem Institut.
- Entfernen von Formicarien eines Instituts.
- Anzeigen aller Formicarien eines Instituts mit allen Informationen auf dem Bildschirm.

Die Klasse `Test` soll die wichtigsten Normal- und Grenzfälle (nicht interaktiv) überprüfen und die Ergebnisse in allgemein verständlicher Form in der Standardausgabe darstellen. Machen Sie unter anderem Folgendes:

- Erstellen und ändern Sie mehrere Institute mit mehreren Formicarien mit jeweils einigen Nestern. Jedes Formicarium eines Instituts soll über seinen eindeutigen Namen angesprochen werden, und jedes Nest eines Formicariums über seine eindeutige Nummer.
- Fügen Sie zu Instituten einzelne Formicarien hinzu, entfernen Sie einzelne Formicarien, wobei Sie Formicarien nur über deren Namen ansprechen.
- Fügen Sie zu einigen Formicarien einzelne Nester hinzu, entfernen Sie einzelne Nester, und ändern Sie die Informationen zu einzelnen Nestern, wobei Sie Nester und Formicarien nur über deren Nummern und Namen ansprechen.
- Berechnen Sie die statistischen Werte aller Formicarien (wie oben beschrieben).

keine vorgefertigten Klassen

Code nicht duplizieren

Generizität, Arrays und vorgefertigte Container-Klassen dürfen zur Lösung dieser Aufgabe nicht verwendet werden. Vermeiden Sie mehrfach vorkommenden Code für gleiche oder ähnliche Programmteile.

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.
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.

Wie die Aufgabe zu lösen ist

Es wird empfohlen, die Aufgabe entsprechend Kontext A zuerst mit Hilfe von Generizität zu lösen und in einem weiteren Schritt eine homogene Übersetzung der Generizität (wie im Skriptum beschrieben) händisch durchzuführen. Durch diese Vorgehensweise erreichen Sie eine statische Überprüfung der Korrektheit vieler Typumwandlungen und vermeiden den unnötigen Verlust an statischer Typsicherheit. Zur Lösung dieser Aufgabe ist die Verwendung von Typumwandlungen ausdrücklich erlaubt. Versuchen Sie trotzdem, die Anzahl der Typumwandlungen klein zu halten und so viel Typinformation wie möglich statisch vorzugeben. Das hilft Ihnen dabei, die Lösung überschaubar zu halten und einen unnötigen Verlust an statischer Typsicherheit zu vermeiden. Gehen Sie auch möglichst sparsam mit dynamischen Typabfragen und Ausnahmebehandlungen um.

Achten Sie darauf, dass Sie Divisionen durch 0 vermeiden. Führen Sie zumindest einen Testfall ein, bei dem eine statistische Auswertung ohne entsprechende Vorkehrungen eine Exception aufgrund einer Division durch 0 auslösen würde.

Bedenken Sie, dass es mehrere sinnvolle Lösungsansätze für diese Aufgabe gibt. Wenn Sie einmal einen gangbaren Weg gefunden haben, bleiben Sie dabei, und vermeiden Sie es, zu viele Möglichkeiten auszuprobieren. Das könnte Sie viel Zeit kosten, ohne die Lösung zu verbessern.

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.

Was im Hinblick auf die Beurteilung wichtig ist

Die insgesamt 100 für diese Aufgabe erreichbaren Punkte sind folgendermaßen auf die zu erreichenden Ziele aufgeteilt:

- Container richtig und wiederverwendbar implementiert, Typumwandlungen korrekt — 25 Punkte
- Annotationen und Reflexion richtig verwendet — 20 Punkte
- Geforderte Funktionalität vorhanden (so wie in Aufgabenstellung beschrieben) — 20 Punkte
- Lösung wie vorgeschrieben und sinnvoll getestet — 20 Punkte
- Zusicherungen richtig und sinnvoll eingesetzt — 10 Punkte
- Sichtbarkeit auf kleinstmögliche Bereiche beschränkt — 5 Punkte

Schwerpunkte berücksichtigen

Der Schwerpunkt bei der Beurteilung liegt auf der vernünftigen Verwendung von dynamischer und statischer Typinformation. Kräftige Punkteabzüge gibt es für

- die Verwendung von Generizität bzw. von Arrays oder vorgefertigten Container-Klassen,
- mehrfach vorkommende gleiche oder ähnliche Programmteile (wenn vermeidbar),
- den unnötigen Verlust an statischer Typsicherheit,
- Verletzungen des Ersetzbarkeitsprinzips bei Verwendung von Vererbungsbeziehungen (also Vererbungsbeziehungen, die keine Untertypbeziehungen sind),
- und mangelhafte Funktionalität des Programms.

Aufgabe nicht abändern

Code nicht duplizieren

Punkteabzüge gibt es unter anderem auch für mangelhafte Zusicherungen und falsche Sichtbarkeit.

Warum die Aufgabe diese Form hat

Die gleichzeitige Unterscheidung von Nestern mit Heizung beziehungsweise Luftbefeuchter sowie zwischen unterschiedlichen Füllungen stellt eine Schwierigkeit dar, für die es mehrere sinnvolle Lösungsansätze gibt. Sie werden irgendeine Form von Container selbst erstellen müssen, wobei die genaue Form und Funktionalität nicht vorgegeben ist. Da Container an mehreren Stellen benötigt werden, wäre die Verwendung von Generizität sinnvoll. Dadurch, dass Sie Generizität nicht verwenden dürfen und trotzdem mehrfache Vorkommen ähnlichen Codes vermeiden sollen, werden Sie gezwungen, Techniken ähnlich denen einzusetzen, die der Compiler zur homogenen Übersetzung von Generizität verwendet. Vermutlich sind Typumwandlungen kaum vermeidbar. Sie sollen dadurch ein tieferes Verständnis des Zusammenhangs zwischen Generizität und Typumwandlungen bekommen.

Die Art der Füllung eines Nests kann sich im Laufe der Zeit ändern. Am besten stellt man solche Beziehungen über Rollen dar: Für jede Art von Nestern gibt es eine eigene Klasse mit den für die jeweilige Art typischen Daten, und ein gemeinsamer Obertyp ermöglicht den Zugriff auf diese Daten auf einheitliche Weise. In jedem Nest gibt es eine Referenz auf die aktuelle Füllung (= die Rolle, die das Nest gerade spielt). Wenn sich die Art der Füllung ändert, braucht nur diese Referenz neu gesetzt zu werden. Durch geschickte Auswahl der Methoden einer Rolle sind die meisten Fallunterscheidungen vermeidbar, das heißt, Fallunterscheidungen werden durch dynamisches Binden ersetzt.

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

Schreiben Sie (abgesehen von geschachtelten Klassen) nicht mehr als eine Klasse in jede Datei. 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