6. Programmieraufgabe

Programmierparadigmen

LVA-Nr. 194.023
2024/2025 W
TU Wien

Themen:

dynamische Typinformation, homogene Übersetzung von Generizität

Ausgabe:

25. 11. 2024

Abgabe (Deadline):

09. 12. 2024, 14:00 Uhr

Abgabeverzeichnis:

`Aufgabe6`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 4.1 bis 4.3.2

Kontext

Ein Bürobetreiber will Daten über seine Büros sammeln und auswerten. Um die Aufgabe einfach zu halten, wird die Realität etwas aufgeweicht. Der Bürobetreiber besitzt mehrere Bürogebäude. Ein Bürogebäude hat einen eindeutigen Namen. Ein Bürogebäude kann eine oder mehrere Büroeinheiten enthalten.

Eine Büroeinheit wird durch eine eindeutige Nummer (ganze Zahl) identifiziert. Eine Büroeinheit besteht aus mehreren nutzbaren Räumen (mindestens ein Raum muss vorhanden sein) und nicht nutzbaren Nebenräumen (z.B. Flure, Nassräume). Für die Nebenräume wird nur die Gesamtfläche pro Büroeinheit in m^2 als Gleitkommazahl gespeichert.

Jeder nutzbare Raum wird durch einen eindeutigen Namen identifiziert. Von jedem Raum werden Länge und Breite in m als Gleitkommazahlen gespeichert. Es gibt genau zwei Arten von Räumen, Räume mit Fenstern und fensterlose Räume (Nebenbemerkung – während in den USA fensterlose Büroräume üblich sind, müssen in Österreich und Deutschland Büroräume ausreichend Tageslicht erhalten und Sichtverbindungen nach außen haben). Von einem fensterlosen Raum wird der Lichtstrom der Beleuchtung in lm (Lumen) als ganze Zahl gespeichert. Von einem Raum mit Fenstern wird die Fensterfläche in m^2 als Gleitkommazahl gespeichert.

Räume können auf zwei unterschiedliche Arten genutzt werden, als Büroräume oder Lagerräume. Die Raumnutzung kann jederzeit getauscht werden. Bei der Nutzung als Büroraum wird die Anzahl der Arbeitsplätze als ganze Zahl gespeichert. Bei der Nutzung als Lagerraum wird das Lagervolumen in m^3 als Gleitkommazahl gespeichert.

Welche Aufgabe zu lösen ist

Entwickeln Sie Java-Klassen bzw. Interfaces zur Darstellung von Büroeinheiten mit Räumen mit wechselnden Nutzungen. Folgende Funktionalität soll unterstützt werden:

- Erzeugen eines Raums mit Name, Länge, Breite und Fensterfläche bei einem Raum mit Fenster oder Lichtstrom bei einem fensterlosen Raum und Setzen der entsprechenden Nutzungsart.
- Auslesen des Namens eines Raums.
- Auslesen der Länge und der Breite eines Raums.
- Auslesen der Fensterfläche bei einem Raum mit Fenstern.
- Auslesen des Lichtstroms bei einem fensterlosen Raum.
- Ändern der Nutzungsart eines Raumes, wobei Informationen über frühere Nutzungen verloren gehen.
- Auslesen der Anzahl der Arbeitsplätze bei Nutzung als Büroraum.

- Auslesen des Lagervolumens bei Nutzung als Lagerraum.

Schreiben Sie zwei Klassen `Office` und `OfficeGen` (siehe unten), die jeweils Informationen über Büroeinheiten mit Räumen verwaltet und statistische Auswertungen über diese Büroeinheiten ermöglichen. Jede Büroeinheit hat eine eindeutige unveränderliche Nummer. Folgende Methoden sollen unterstützt werden:

- Erzeugen einer Büroeinheit mit gleichzeitigem Setzen einer eindeutigen Nummer und der Fläche der Nebenräume.
- Auslesen der Nummer einer Büroeinheit.
- Auslesen der Fläche der Nebenräume einer Büroeinheit.
- Berechnen der Gesamtfläche einer Büroeinheit (Nebenräume + Fläche der nutzbaren Räume).
- Hinzufügen von Räumen zu einer Büroeinheit.
- Entfernen von Räumen von einer Büroeinheit.
- Ändern der Informationen von Räumen wie oben beschrieben.
- Methoden zum Berechnen folgender (statistischer) Werte:
  - Die durchschnittliche Fläche aller Räume (ohne Nebenräume) einer Büroeinheit.
  - Die durchschnittliche Fläche aller Räume mit Fenstern (ohne Nebenräume) einer Büroeinheit.
  - Die durchschnittliche Fläche aller fensterlosen Räume (ohne Nebenräume) einer Büroeinheit.
  - Das durchschnittliche Lagervolumen pro Raum aller Lagerräume einer Büroeinheit.
  - Die durchschnittliche Anzahl der Arbeitsplätze pro Raum aller Büroräume einer Büroeinheit.
  - Das durchschnittliche Verhältnis von Fensterfläche zu Raumfläche aller Räume mit Fenstern (ohne Nebenräume) einer Büroeinheit (Nebenbemerkung – sollte in Österreich größer 0,10 sein) – alle Räume zusammen und zusätzlich aufgeschlüsselt nach der Art der Nutzung (Lager- oder Büroraum).
  - Das durchschnittliche Verhältnis von Lichtstrom zu Raumfläche (das ergibt die Beleuchtungsstärke in Lux ($1lx = 1lm/m^2$)) aller fensterlosen Räume (ohne Nebenräume) einer Büroeinheit – alle Räume zusammen und zusätzlich aufgeschlüsselt nach der Art der Nutzung (Lager- oder Büroraum).

Schreiben Sie zwei Klassen `Building` und `BuildingGen`, die jeweils Informationen über ein Gebäude verwalten. Jedes Gebäude hat einen unveränderlichen Namen. Folgende Methoden sollen unterstützt werden:

- Erzeugen eines Gebäudes (Objekt von `Building`).

- Hinzufügen von Büroeinheiten (`Office`) zu einem Gebäude.
- Entfernen von Büroeinheiten eines Gebäude.
- Anzeigen aller Büroeinheiten eines Gebäude mit allen Informationen auf dem Bildschirm.

Schreiben Sie zwei Klassen `Company` und `CompanyGen`, die jeweils Informationen über alle Gebäude eines Betreibers verwaltet. Jeder Betreiber hat einen unveränderlichen Namen. Folgende Methoden sollen unterstützt werden:

- Erzeugen eines Betreibers (Objekt von `Company` bzw. `CompanyGen`).
- Hinzufügen von Gebäuden (`Building` bzw. `BuildingGen`) zu einem Betreiber.
- Entfernen von Gebäuden eines Betreibers.
- Anzeigen aller Gebäude eines Betreibers 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 ein Objekt von `Company` und ein entsprechendes von `CompanyGen`, erstellen und ändern Sie mehrere Gebäude mit mehreren Büroeinheiten mit jeweils einigen Räumen. Jede Büroeinheit eines Gebäudes soll über ihre eindeutige Nummer angesprochen werden, und jeder Raum einer Büroeinheit über seinen eindeutigen Namen.
- Fügen Sie zu Gebäuden einzelne Büroeinheiten hinzu, entfernen Sie einzelne Büroeinheiten, wobei Sie Büroeinheiten nur über deren Nummern ansprechen.
- Fügen Sie zu einigen Büroeinheiten einzelne Räume hinzu, entfernen Sie einzelne Räume, und ändern Sie die Informationen zu einzelnen Räumen, wobei Sie Büroeinheiten und Räume nur über deren Nummern und Namen ansprechen.
- Berechnen Sie die statistischen Werte aller Büroeinheiten (wie oben beschrieben) und zeigen Sie diese Werte auf dem Bildschirm an.

Zwei unterschiedliche Lösungen gefordert

2 Lösungen gefordert

Variante ohne Generizität

Von jedem Typ (Klasse oder Interface), in dem die Verwendung von Generizität sinnvoll wäre (die also auch ohne Ersetzbarkeit für Objekte unterschiedlicher Typen verwendet werden können), sind zwei Varianten zu implementieren: Eine Variante (deren Name nicht auf `Gen` endet, z. B. `Company`) muss zur Gänze ohne Verwendung von Generizität auskommen; es dürfen auch keine Objekte generischer Klassen oder entsprechender Raw-Types verwendet werden. Die zweite Variante (deren Name mit `Gen`

endet, sonst aber gleich heißt wie die erste Variante, z. B. `CompanyGen`) soll überall Generizität verwenden, wo dies sinnvoll ist. Die Klasse `Test` soll sowohl die generische als auch nicht generische Lösungsvariante testen, sodass es nur eine Klasse `Test` gibt, keine zusätzliche Klasse `TestGen`.

Variante mit Generizität

Arrays und vorgefertigte Container-Klassen dürfen zur Lösung dieser Aufgabe nicht verwendet werden. In der Variante ohne Generizität dürfen zusätzlich keine generischen Methoden, Klassen und Interfaces verwendet werden (auch nicht selbst implementierte oder Raw-Types). Das betrifft auch die meisten vordefinierten Functional-Interfaces.

kein vorgefertigter Code

Vermeiden Sie mehrfach vorkommenden Code für gleiche oder ähnliche Programmteile. Davon ausgenommen sind nur Ähnlichkeiten zwischen Code in einer generischen Variante und der entsprechenden nicht generischen Variante.

Code nicht duplizieren

Wie die Aufgabe zu lösen ist

Es wird empfohlen, die Aufgabe zuerst mit Hilfe von Generizität zu lösen (generische Variante) und in einem zweiten Schritt daraus die nicht generische Variante durch eine händische homogene Übersetzung der Generizität (wie im Skriptum beschrieben) durchzuführen. Durch diese Vorgehensweise erreichen Sie eine statische Überprüfung der Korrektheit vieler Typumwandlungen und vermeiden den unnötigen Verlust an statischer Typsicherheit. Die generische und nicht generische Variante müssen einander nicht direkt entsprechen (weil das wegen unterschiedlich starker Einschränkungen nicht immer leicht erreichbar wäre), sollten aber die gleiche Funktionalität aufweisen. Testfälle sollen auch überprüfen, ob beide Varianten das gleiche Verhalten aufweisen. Zur Lösung dieser Aufgabe ist die Verwendung von Typumwandlungen ausdrücklich erlaubt und im nicht generischen Teil nicht vermeidbar. 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.

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

Aufgabenaufteilung beschreiben

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 ohne Generizität richtig und wiederverwendbar implementiert, Typumwandlungen korrekt 25 Punkte
- Container unter Verwendung von Generizität wiederverwendbar implementiert 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 in der nicht generischen Variante 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 Räumen ohne/mit Fenstern sowie zwischen unterschiedlichen Nutzungsarten 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, ist die Verwendung von Generizität sinnvoll. Dadurch, dass Sie in einer Lösungsvariante 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 Nutzung eines Raums kann sich im Laufe der Zeit ändern. Am besten stellt man solche Beziehungen über Rollen dar: Für jede Art der Nutzung 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. Für jeden Raum gibt es eine Referenz auf die aktuelle Nutzung (= die Rolle, die der Raum gerade spielt). Wenn sich die Art der Nutzung 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.

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