5. Programmieraufgabe

Objektorientierte Programmiertechniken

LVA-Nr. 185.A01
2020/2021 W
TU Wien

Themen:

Generizität, Container, Iteratoren

Ausgabe:

11. 11. 2020

Abgabe (Deadline):

18. 11. 2020, 12:00 Uhr

Abgabeverzeichnis:

`Aufgabe5`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 3.1 und 3.2

Kontext

Ausschnitte aus der Lebensgemeinschaft einer Wiese sind als Container mit Pflanzen, Tieren, etc. darstellbar. Zur Reduktion von Programmtexten bietet sich Generizität an. Wir benötigen folgende, bei Bedarf generische Interfaces oder (abstrakte oder konkrete) Klassen:

`Related` ist ein Interface mit einer Methode `relatedTo`, die einen Parameter (Typ durch Typparameter bestimmt) nimmt und eine Fließkommazahl zwischen 0 und 1 als Ergebnis zurück gibt. Das Ergebnis von `x.relatedTo(y)` beschreibt die Wahrscheinlichkeit dafür, dass `x` in einer (hier nicht näher bestimmten) Beziehung zu `y` steht.

`Identity` ist ein Untertyp von `Related` mit geeigneten Typparameterersetzungen, sodass `relatedTo` Vergleiche mit Objekten beliebiger Typen durchführt. Dabei gibt `x.relatedTo(y)` 1.0 zurück wenn `x` und `y` identisch sind, sonst 0.0.

`Plant` ist ein Untertyp von `Related` mit geeigneten Typparameterersetzungen, sodass `relatedTo` Vergleiche mit Objekten von `Plant` durchführt. Konkret gibt `x.relatedTo(y)` die Wahrscheinlichkeit dafür zurück, dass `x` bei beengten Platzverhältnissen (Lichtmangel) `y` verdrängt. Das ist abhängig von der Wuchskraft der Pflanze, die außerhalb von Objekten von `Plant` nicht bekannt ist. Über eine oder mehrere Methoden (die nicht detailliert vorgegeben sind) gibt ein Objekt von `Plant` bekannt, wie viele Blüten sie hat, wie auffällig diese Blüten für Insekten sind und, falls es sich um Röhrenblüten handelt, wie tief die Blütenröhre ist.

`Insect` ist ein Untertyp von `Related` mit geeigneten Typparameterersetzungen, sodass `relatedTo` Vergleiche mit Objekten vom Typ `Plant` durchführt. Konkret gibt `x.relatedTo(y)` die Wahrscheinlichkeit dafür zurück, dass das Insekt `x` die Pflanze `y` bestäubt. Die Wahrscheinlichkeit hängt davon ab, wie viele Blüten die Pflanze hat und wie auffällig diese Blüten für Insekten sind.

`Bee` ist ein Untertyp von `Insect`. Das Ergebnis von `relatedTo` hängt auch davon ab, wie tief eine Röhrenblüte ist; zu tiefe Röhrenblüten werden kaum bestäubt. Die Methode `honey` gibt die Menge an Honig (vom Typ `double`, in Kilogramm) zurück, die eine Biene in ihrem ganzen Leben voraussichtlich sammelt.

`Butterfly` ist ein Untertyp von `Insect`. Das Ergebnis von `relatedTo` hängt auch davon ab, wie tief eine Röhrenblüte ist; tiefe Röhrenblüten werden bevorzugt bestäubt. Die Methode `color` gibt eine textuelle Beschreibung (vom Typ `String`) der Farbe und Musterung des Schmetterlings zurück.

`RelationSet` ist mit geeigneten Typparameterersetzungen der Typ eines Containers, der Einträge zweier Typen `X` und `Y` enthält, wobei `X` und `Y` über Typparameter festgelegt sind. Entsprechend `Related` soll `x.relatedTo(y)` aufrufbar sein wenn `x` vom Typ `X` und `y` vom Typ `Y` ist. Ein Objekt von `Related` hat folgende Methoden:

- **`add`** nimmt je ein Objekt von `X` und `Y` und fügt die beiden Objekte zu `this` hinzu. Mehrfacheinträge sind erlaubt.
- **`iterX`** gibt einen Iterator vom Typ `java.util.Iterator<X>` zurück. Die Reihenfolge ist vorgegeben: Es werden jene Einträge `x` und `y` ermittelt, für die `x.relatedTo(y)` den größten Wert liefert; `x` wird zurückgegeben, `y` wird nicht zurückgegeben, aber als abgehandelt betrachtet. Von den restlichen Einträgen werden wieder jene `x` und `y` mit dem größten `x.relatedTo(y)` ermittelt, `x` zurückgegeben und so weiter. Mehrfacheinträge werden mehrfach berücksichtigt. Ein Aufruf von `remove` entfernt das zuletzt ermittelte Paar `x` und `y`.
- **`iterY`** gibt einen Iterator vom Typ `java.util.Iterator<Y>` zurück. Er entspricht dem durch `iterX` erzeugten Iterator, gibt jedoch die gefundenen `y` zurück, nicht die `x`.

`AutoRelationSet` ist eine Variante von `RelationSet`, die für `X` und `Y` den gleichen Typ verwendet. `AutoRelationSet` ist mit geeigneten Typparameterersetzungen Untertyp von `Related`. Die Methode `relatedTo` verlangt, dass auch das Argument ein `AutoRelationSet` ist. Ein Aufruf von `x.relatedTo(y)` gibt 1.0 zurück wenn `x` mehr Einträge hat als `y`, 0.5 wenn `x` und `y` gleich viele Einträge haben und 0.0 wenn `y` mehr Einträge hat als `x`. Wenn möglich soll `AutoRelationSet` Untertyp von `RelationSet` sein.

Welche Aufgabe zu lösen ist

Implementieren Sie die oben beschriebenen Klassen und Interfaces.

Ein Aufruf von `java Test` im Abgabeverzeichnis soll wie gewohnt Testfälle ausführen und die Ergebnisse in allgemein verständlicher Form darstellen. Anders als in bisherigen Aufgaben sind einige Überprüfungen vorgegeben und in dieser Reihenfolge auszuführen:

vorgegebene Tests

1. Erzeugen Sie mindestens je ein Objekt sinngemäß folgender Typen (wobei hier für `RelationSet` als Reihenfolge der Typparameter `<X,Y>` entsprechend obigen Beschreibungen angenommen wird):

```
RelationSet<Insect, Plant>
RelationSet<Bee, Plant>
RelationSet<Butterfly, Plant>
RelationSet<Plant, Plant>
RelationSet<Identity, Bee>
AutoRelationSet<Plant>
AutoRelationSet<Identity>
AutoRelationSet<AutoRelationSet<Plant>>
RelationSet<AutoRelationSet<Plant>,
            AutoRelationSet<Identity> >
```

Andere Reihenfolgen von Typparametern und zusätzliche Typparameter sind erlaubt, wenn Sie das für sinnvoll erachten. Befüllen Sie die Container mit einigen Einträgen.

2. Überprüfen Sie die Funktionalität der oben eingeführten Objekte und ihrer Einträge durch Ausgabe aller abfragbaren Daten in die Standardausgabe. Vergewissern Sie sich, dass in Containereinträgen des Typs `Bee` die Methode `honey` und in denen von `Butterfly` die Methode `color` ausführbar ist.

3. Falls `AutoRelationSet` bei geeigneten Typparameterersetzungen ein Untertyp von `RelationSet` ist, wenden Sie die Testmethoden, die in Punkt 2 zur Überprüfung von `RelationSet`-Objekten entwickelt wurden, auch auf entsprechende `AutoRelationSet`-Objekte an um die Ersetzbarkeit zu überprüfen. Andernfalls beschreiben Sie bitte kurz, warum keine Ersetzbarkeit gegeben ist.

4. Wählen Sie ein `RelationSet<Insect,Plant>`-Objekt. Fügen Sie ihm alle Einträge je eines in Punkt 1 aufgebauten Objekts von `RelationSet<Bee,Plant>` sowie `RelationSet<Butterfly,Plant>` durch Aufrufe von `add` hinzu, wobei diese Einträge über Iteratoren aus letzteren Objekten ausgelesen werden. Überprüfen Sie (z. B. durch Ausgabe), ob die so aufgefüllten Container wirklich die richtigen Einträge in der richtigen Reihenfolge enthalten.

   Generizität so planen, dass das gehen kann

5. Machen Sie optional weitere Überprüfungen.

   nicht verpflichtend

Daneben 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

Wie die Aufgabe zu lösen ist

Von allen oben beschriebenen Interfaces, Klassen und Methoden wird erwartet, dass sie überall verwendbar sind. Der Bereich, in dem weitere eventuell benötigte Klassen, Methoden, Variablen, etc. sichtbar sind, soll jedoch so klein wie möglich gehalten werden.

Sichtbarkeit beachten

Alle Teile dieser Aufgabe sind ohne Arrays und ohne vorgefertigte Container (wie z. B. Klassen des Collection-Frameworks) zu lösen. Benötigte Container und Iteratoren sind selbst zu schreiben.

Verbote beachten!!

Typsicherheit soll vom Compiler garantiert werden. Auf Typumwandlungen (Casts) und ähnliche Techniken ist zu verzichten, und der Compiler darf keine Hinweise auf mögliche Probleme im Zusammenhang mit Generizität geben. Raw-Types dürfen nicht verwendet werden.

Generizität statt dynamischer Prüfungen

Übersetzen Sie die Klassen mittels `javac -Xlint:unchecked ...` Dieses Compiler-Flag schaltet genaue Compiler-Meldungen im Zusammenhang mit Generizität ein. Andernfalls bekommen Sie auch bei schweren Fehlern möglicherweise nur eine harmlos aussehende Meldung. Überprüfungen durch den Compiler dürfen nicht ausgeschaltet werden.

Compiler-Feedback einschalten

Es ist nicht beschrieben, woher von einigen Methoden zurückgegebene Daten stammen. Setzen Sie diese Daten am besten über Konstruktoren.

Konstruktoren

Was im Hinblick auf die Beurteilung wichtig ist

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

Schwerpunkte berücksichtigen

- Generizität und geforderte Untertypbeziehungen richtig eingesetzt, sodass die Tests ohne Tricks durchführbar sind — 40 Punkte
- Lösung wie vorgeschrieben und sinnvoll getestet — 20 Punkte
- Zusicherungen konsistent und zweckentsprechend — 15 Punkte
- Sichtbarkeit auf kleinstmögliche Bereiche beschränkt — 15 Punkte
- Lösung vollständig (entsprechend Aufgabenstellung) — 10 Punkte

Am wichtigsten ist die korrekte Verwendung von Generizität. Es gibt bedeutende Punkteabzüge, wenn der Compiler mögliche Probleme im Zusammenhang mit Generizität meldet oder wichtige Teilaufgaben nicht gelöst bzw. umgangen werden. Beachten Sie, dass Raw-Types nicht verwendet werden dürfen, der Compiler aber auch mit `-Xlint:unchecked` nicht alle Verwendungen von Raw-Types meldet.

Ein zusätzlicher Schwerpunkt liegt auf dem gezielten Einsatz von Sichtbarkeit. Es gibt Punkteabzüge, wenn Programmteile, die überall sichtbar sein sollen, nicht `public` sind, oder Teile, die nicht für die allgemeine Verwendung bestimmt sind, unnötig weit sichtbar sind. Durch die Verwendung innerer Klassen kann das Sichtbarmachen mancher Programmteile nach außen verhindert werden.

Nach wie vor spielen auch Untertypbeziehungen und Zusicherungen eine große Rolle bei der Beurteilung.

Aufgabe nicht abändern

Generell führen verbotene Abänderungen der Aufgabenstellung – beispielsweise die Verwendung von Typumwandlungen, Arrays oder vorgefertigten Containern und Iteratoren oder das Ausschalten von Überprüfungen durch `@SuppressWarning` – zu bedeutenden Punkteabzügen.

Warum die Aufgabe diese Form hat

Schwierigkeiten erkennen
Skriptum anschauen

Die Aufgabe ist so konstruiert, dass dabei Schwierigkeiten auftauchen, für die wir Lösungsmöglichkeiten kennengelernt haben. Wegen der vorgegebenen, in die Typparameter einzusetzenden Typen muss Generizität über mehrere Ebenen hinweg betrachtet werden. Vorgegebene Testfälle stellen sicher, dass einige bedeutende Schwierigkeiten erkannt werden. Um Umgehungen außerhalb der Generizität zu vermeiden sind Typumwandlungen ebenso verboten wie das Ausschalten von Compilerhinweisen auf unsichere Verwendungen von Generizität. Das Verbot der Verwendung vorgefertigter Container-Klassen verhindert, dass Schwierigkeiten nicht selbst gelöst, sondern an Bibliotheken weitergereicht werden. Zusätzlich wird das Erstellen typischerweise generischer Programmstrukturen geübt.

Daneben wird auch der Umgang mit Sichtbarkeit geübt. Am Beispiel von Iteratoren soll intuitiv klar werden, welchen Einfluss innere Klassen auf die Sichtbarkeit von Implementierungsdetails haben.

innere Klassen verwenden