4. Programmieraufgabe

Objektorientierte Programmiertechniken

LVA-Nr. 185.A01
2021/2022 W
TU Wien

Themen:

Untertypbeziehungen, Zusicherungen

Ausgabe:

16. 11. 2021

Abgabe (Deadline):

23. 11. 2021, 10:00 Uhr

Abgabeverzeichnis:

`Aufgabe4`
nicht mehr `Aufgabe1-3`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf Kapitel 2

Kontext

Die Aufgabe bezieht sich auf ein geplantes System zur Analyse von Laubbaumbeständen (fokusiert auf Österreich, aber nicht darauf beschränkt). Vorerst soll ein Programm entstehen, das zentrale Klassen bzw. Interfaces unabhängig vom Gesamtsystem auf Konsistenz prüft und testet. Bilden Sie zu diesem Zweck jeden der folgenden Begriffe, beschrieben in alphabetischer Reihenfolge, durch je einen Typ im Programm ab:

`CarpinusBetulus:` Eine (Gemeine) Hainbuche. Bäume dieser Art gehören zur Familie der Birkengewächse (Betulaceae) und sind trotz ihres Namens nicht mit Buchen verwandt. Sie sind in Österreich vor allem unter kontinentalem Einfluss weit verbreitet und vertragen Beschattungen gut, fast so gut wie Rotbuchen. Im Gegensatz zu Rotbuchen vertragen sie Verbiss und Rückschnitt gut, bleiben aber meist kleiner als Rotbuchen. Hainbuchen liefern gutes Brennholz, abgesehen davon ist der forstwirtschaftliche Nutzen gering.

`ContinentalClimate:` Ein Baum einer Art, die unter kontinentalem Einfluss heimisch ist. In Österreich liegen, topographisch bedingt anders als in den meisten Ländern, Gebiete mit ozeanischem und kontinentalem Einfluss kleinräumig beieinander. Kontinentaler Einfluss, definiert durch größere Temperaturunterschiede zwischen Winter und Sommer, vermehrte Spätfröste und ungleiche Verteilung der Niederschläge im Sommer, ist an der Vegetation ablesbar. Zur Unterscheidung dienen Zeigerpflanzen, also Pflanzen, die nur unter entweder ozeanischem oder kontinentalem Einfluss gedeihen. Die Rotbuche ist eine Zeigerpflanze für ozeanischen Einfluss, das heißt, wo eine Rotbuche vorkommt, herrscht ozeanischer Einfluss vor, kein kontinentaler. Die Umkehrung gilt nicht, weil das Fehlen von Rotbuchen auch andere Ursachen haben kann, etwa ungeeignete Böden. Alle anderen in diesem Text genannten Baumarten (nicht Familien) sind sowohl unter ozeanischem als auch kontinentalem Einfluss heimisch, wenn auch unterschiedlich stark vertreten. Die Methode `incidence` gibt eine positive Zahl zurück, die besagt, wie viel stärker (≥ 1) oder schwächer (≤ 1) die Art unter kontinentalem Einfluss vertreten ist als unter ozeanischem.

`Domestic:` Ein Baum einer Art, die in österreichischen Wäldern heimisch ist. Nicht jeder Baum, der in Österreich gefunden wird, ist hier auch heimisch, sondern nur Bäume jener Arten, die sich hier ohne menschliche Unterstützung dauerhaft ausbreiten und gegen Konkurrenz durchsetzen. Die Methoden `longitude` und `latitude` liefern die geographische Länge bzw. Breite des Standorts des Baums.

`Fagaceae:` Ein Baum aus der Familie der Buchengewächse. Viele in österreichischen Wäldern heimische Arten zählen zu dieser Familie, neben Rotbuchen auch Eichen und Kastanien, aber bei weitem nicht alle Arten dieser Familie sind in Europa heimisch.

`FagusSylvatica:` Eine Rotbuche. Die Rotbuche ist die einzige in Europa, auch in Österreich heimische Buchenart (Fagus) aus der Familie der Buchengewächse (Fagaceae). Rotbuchenreiche Wälder gelten in großen, ozeanisch beeinflussten Teilen Mitteleuropas als die natürlichste Vegetation und breiten sich zunehmend aus. Rotbuchen vertragen starke Beschattung und wachsen auch unter dichten Baumkronen, wobei Rotbuchen den Waldboden stark beschatten, sodass darunter weniger schattenverträgliche Arten, insbesonders Lichtbaumarten, nicht wachsen, was ohne menschliches oder tierisches Zutun (Verbiss) eine Monokultur entstehen lässt. Das Holz der Rotbuche ist vielseitig verwendbar. Der sich daraus ergebende forstwirtschaftliche Nutzen wird jedoch dadurch getrübt, dass ein großer Teil des Holzes in den wirtschaftlich kaum verwertbaren ausladenden, dichten Kronen steckt (großer Flächenverbrauch) und schon der Ausfall eines einzigen Baums zu großen Freiflächen führt.

`LightDemanding:` Baum einer Lichtbaumart. Solche Bäume können im Schatten nicht wachsen, sie benötigen direktes Sonnenlicht. Lichtbaumarten sind in der Regel leicht daran zu erkennen, dass die Wuchsrichtung der Verfügbarkeit von Sonnenlicht folgt, während Baumspitzen schattenverträglicher Arten auch bei ungleichmäßiger Beschattung senkrecht nach oben wachsen.

`Quercus:` Eine Eiche. Bäume aus der Gattung der Eichen zählen zur Familie der Buchengewächse (Fagaceae). Viele Eichenarten sind nur in Amerika heimisch, einige in Europa. Gelegentlich wird in Europa die nur in Amerika heimische Roteiche gepflanzt, weil sie Schatten und schlechte Umweltbedingungen gut verträgt.

`QuercusPetraea:` Eine Traubeneiche. Eichen (Quercus) dieser Art sind in Österreich heimisch und weit verbreitet. Es handelt sich um eine Lichtbaumart, obwohl Jungbäume auch eine gewisse Beschattung vertragen. Traubeneichen wachsen langsam, werden aber sehr alt und bilden, wo die Bedingungen das zulassen, riesige Kronen und gewaltige Stammumfänge. Sie bilden die Lebensgrundlage für unzählige Insekten und sind daher von großer ökologischer Bedeutung. Aufgrund des wertvollen, sehr widerstandsfähigen Holzes sind Traubeneichen auch forstwirtschaftlich interessant, allerdings getrübt durch das langsame Wachstum. Die Kronen können zwar riesig sein, lassen aber einen gewissen Anteil am Sonnenlicht bis zum Boden durch, sodass darunter viele schattenverträgliche Baumarten gedeihen können. Traubeneichen wachsen daher fast immer in Gemeinschaft mit anderen Baumarten, häufig etwa Hainbuchen.

`QuercusRobur:` Eine Stileiche. Eichen (Quercus) dieser Art sind ebenso in Österreich heimisch und wachsen hier häufig an den gleichen Standorten wie Traubeneichen. Stil- und Traubeneichen sind einander sehr ähnlich, physiologisch ebenso wie hinsichtlich der ökologischen und forstwirtschaftlichen Bedeutung, weswegen die eine Art gelegentlich für eine lokale Variation der anderen gehalten wird. Aber anhand der Früchte sind die beiden Arten leicht unterscheidbar: einzeln stehende größere Eicheln in Bechern an Stilen bei der

Stileiche, mehrere Becher mit etwas kleineren Eicheln in traubenförmigen Büscheln bei der Traubeneiche. Auch die Stileiche ist eine Lichtbaumart, aber stärker ausgeprägt (also weniger schattenverträglich) als die Traubeneiche. Das Verbreitungsgebiet der Stileiche reicht viel weiter nach Osteuropa, während die Traubeneiche im westlichen Mitteleuropa deutlich stärker vertreten ist. Daraus lässt sich schließen, dass Stileichen unter kontinentalem Einfluss bessere Wachstumsbedingungen vorfinden als Traubeneichen.

`Tree:` Ein Baum. Die Methode `species` gibt den wissenschaftlichen (lateinischen) Namen der Baumart zurück, die Methode `size` die geschätzte Baumhöhe in Metern. Mit Hilfe der Methode `changeSize` lässt sich die geschätzte Höhe vergößern (positives Argument) oder verringern (negatives Argument).

Es ist davon auszugehen, dass ein Baum nicht gleichzeitig mehreren Arten oder Familien angehören kann, gegensätzliche Eigenschaften einander ausschließen (z. B. „ist Lichtbaumart“ und „verträgt Schatten gut“) und geographische Gegebenheiten gelten (z. B. liegt Österreich in Europa). Obige Beschreibungen der Typen sind dahingehend als vollständig anzusehen, dass alle Themen, die bei der Typbildung eine Rolle spielen sollen, angesprochen wurden. Weitere Eigenschaften, etwa maximale Wuchshöhe oder typische Höhenstufen (Seehöhe) sollen unberücksichtigt bleiben.

Selbstverständliches

keine zusätzlichen Eigenschaften annehmen

Die Anzahl der obigen Beschreibungen von Methoden ist klein gehalten. Zur Realisierung von Untertypbeziehungen können zusätzliche Methoden in den einzelnen Typen nötig sein, weil Methoden von Obertypen auf Untertypen übertragen werden.

Methoden aus Obertypen übernehmen

Welche Aufgabe zu lösen ist

Schreiben Sie ein Java-Programm, das für jeden unter *Kontext* angeführten Typ eine (abstrakte) Klasse oder ein Interface bereitstellt. Versehen Sie alle Typen mit den notwendigen Zusicherungen und stellen Sie sicher, dass Sie nur dort eine Vererbungsbeziehung (`extends` oder `implements`) verwenden, wo eine Untertypbeziehung besteht. Ermöglichen Sie Untertypbeziehungen zwischen *allen* diesen Typen, außer wenn sie den Beschreibungen der Typen (siehe *Kontext*) widersprechen würden.

alle Zusicherungen und Untertypbeziehungen

Besteht zwischen zwei Typen keine Untertypbeziehung, geben Sie in einem Kommentar in `Test.java` eine Begründung dafür an. Bitte geben Sie eine textuelle Begründung, auskommentierte Programmteile reichen nicht. Das Fehlen einer Methode in einer Typbeschreibung ist als Begründung ungeeignet, weil zusätzliche Methoden hinzugefügt werden dürfen. Sie brauchen keine Begründung dafür angeben, dass A kein Untertyp von B ist, wenn B ein Untertyp von A ist.

Begründung wenn keine Untertypbeziehung

Alle oben genannten Typen müssen mit vorgegebenen Namen vorkommen, auch solche, die Sie für unnötig erachten. Vermeiden Sie wenn möglich zusätzliche abstrakte Klassen und Interfaces. Zum Testen können Sie zusätzliche Klassen für konkrete Baumarten einführen, aber kennzeichnen Sie diese bitte klar als nicht zum eigentlichen System gehörend. Vorgegebene Methoden sind semantisch sehr einfach und sollen auch so einfach implementiert sein (z. B. einen konstant vorgegebenen oder auf einfache Weise berechneten Wert zurückgeben).

gegebene Typen mit gegebenen Namen

Schreiben Sie eine Klasse `Test` zum Testen Ihrer Lösung. Das Programm muss vom Abgabeverzeichnis (`Aufgabe4`) aus durch `java Test` ausführbar sein. Überprüfen Sie mittels Testfällen, ob dort, wo Sie eine Untertypbeziehung annehmen, Ersetzbarkeit gegeben ist.

Testklasse

Daneben soll die Datei `Test.java` als Kommentar 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:

Schwerpunkte berücksichtigen

- Untertypbeziehungen richtig erkannt und eingesetzt — 40 Punkte
- nicht bestehende Untertypbeziehungen gut begründet — 15 Punkte
- Zusicherungen passend und zweckentsprechend — 20 Punkte
- Lösung entsprechend Aufgabenstellung getestet — 15 Punkte
- Lösung vollständig (entsprechend Aufgabenstellung) — 10 Punkte

Die 15 Punkte für das Testen sind nur erreichbar, wenn die Testfälle mögliche Widersprüche in den Untertypbeziehungen aufdecken könnten. Abfragen mittels `instanceof` sowie Casts sind dafür ungeeignet, weil dies nur die auf Signaturen beruhenden Typeigenschaften berücksichtigt, die ohnehin vom Compiler garantiert werden.

inhaltlich testen

Die 10 Punkte für eine „vollständige Lösung“ gelten für jene Bereiche, für die keine speziellen Abzüge vorgesehen sind. Das können z. B. kleine Logikfehler, fehlende kleine Programmteile, unzureichende Beschreibungen der Aufgabenaufteilung, oder Probleme mit Data-Hiding bzw. beim Compilieren sein. Fehlen wichtige Programmteile oder treten schwere Fehler auf, werden auch Untertypbeziehungen, Zusicherungen und das Testen betroffen sein, was zu deutlich größerem Punkteverlust führt.

Die größte Schwierigkeit dieser Aufgabe liegt darin, *alle* Untertypbeziehungen zu finden und Ersetzbarkeit sicherzustellen. Vererbungsbeziehungen, die keine Untertypbeziehungen sind, führen zu sehr hohem Punkteverlust. Hohe Punkteabzüge gibt es auch für nicht wahrgenommene Gelegenheiten, Untertypbeziehungen zwischen vorgegebenen Typen herzustellen, sowie für fehlende oder falsche Begründungen (geeignet wären z. B. Gegenbeispiele) für nicht bestehende Untertypbeziehungen.

alle Untertypbeziehungen

Eine Grundlage für das Auffinden der Untertypbeziehungen sind gute Zusicherungen. Wesentliche Zusicherungen kommen in obigen Beschreibungen vor. Allerdings ist nicht jeder Teil einer Beschreibung als Zusicherung von Bedeutung. Untertypbeziehungen ergeben sich aus erlaubten Beziehungen zwischen Zusicherungen in Unter- und Obertypen. Es ist günstig, alle Zusicherungen, die im Obertyp gelten, auch im Untertyp (noch einmal) hinzuschreiben, weil dadurch so mancher Widerspruch deutlich sichtbar und damit eine falsche Typstruktur mit höherer Wahrscheinlichkeit vermieden wird. Nicht die Quantität der Kommentare ist entscheidend, sondern die Qualität (Verständlichkeit, Vollständigkeit, Aussagekraft, ...). Zusicherungen in `Test.java` werden bei der Beurteilung aus praktischen Gründen nicht berücksichtigt.

Zusicherungen

Auch beim Testen kommt es auf Qualität, nicht Quantität an. Testfälle sollen grundsätzlich in der Lage sein, Verletzungen der Ersetzbarkeit aufzudecken, die nicht ohnehin vom Compiler erkannt werden.

Qualität vor Quantität

Zur Lösung dieser Aufgabe müssen Sie Untertypbeziehungen und den Einfluss von Zusicherungen genau verstehen. Lesen Sie Kapitel 2 des Skriptums. Folgende zusätzlichen Informationen könnten hilfreich sein:

nicht nur intuitiv vorgehen

- Konstruktoren werden in einer konkreten Klasse aufgerufen und sind daher vom Ersetzbarkeitsprinzip nicht betroffen.
- Zur Lösung der Aufgabe sind keine Exceptions nötig. Generell darf ein Objekt eines Untertyps nur Exceptions werfen, die man auch von Objekten des Obertyps in dieser Situation erwarten würde.
- Mehrfachvererbung gibt es nur auf Interfaces. Sollte ein Typ mehrere Obertypen haben, müssen diese (bis auf einen) Interfaces sein.

Lassen Sie sich von der Form der Beschreibung nicht täuschen. Aus Ähnlichkeiten im Text oder einem Verweis auf einen anderen Typ folgt noch keine Ersetzbarkeit. Sie sind auf dem falschen Weg, wenn es den Anschein hat, A könne Untertyp von B und gleichzeitig B Untertyp von A sein, außer wenn A und B gleich sind.

Form $\neq$ Inhalt

Achten Sie auf die Sichtbarkeit. Alle beschriebenen Typen und Methoden sollen überall verwendbar sein, sonst nichts. Sichtbare Implementierungsdetails beeinflussen die Ersetzbarkeit. Wenn Sie Implementierungsdetails unnötig weit sichtbar machen, sind möglicherweise bestimmte Untertypbeziehungen nicht mehr gegeben, wodurch Sie das Ziel verfehlen, alle realisierbaren Untertypbeziehungen zu ermöglichen.

Sichtbarkeit

Was man generell beachten sollte (alle Aufgaben)

Es werden keine Ausnahmen bezüglich des Abgabetermins gemacht. Beurteilt wird, was im Abgabeverzeichnis `Aufgabe4` im Repository steht. Alle `.java`-Dateien im Abgabeverzeichnis (einschließlich Unterverzeichnissen) müssen auf der `g0` gemeinsam übersetzbar sein. Auf der `g0` sind nur Standardbibliotheken installiert; es dürfen keine anderen Bibliotheken verwendet werden. Viele Übersetzungsfehler kommen von falschen `package`- oder `import`-Anweisungen und unabsichtlich abgegebenen, nicht zur Lösung gehörenden `.java`-Dateien (Reste früher Lösungsversuche oder Backup).

Abgabetermin einhalten

auf g0 testen

Schreiben Sie nur eine Klasse in jede Datei (außer geschachtelte Klassen), halten Sie sich an übliche Namenskonventionen und verwenden Sie die Namen, die in der Aufgabenstellung vorgegeben sind. Andernfalls könnte es zu Fehlinterpretationen Ihrer Absichten kommen, die sich negativ auf die Beurteilung auswirken.

eine Klasse pro Datei

vorgegebene Namen

Warum die Aufgabe diese Form hat

Die Beschreibungen der Typen bieten wenig Interpretationsspielraum bezüglich Ersetzbarkeit. Die Aufgabe ist so formuliert, dass Untertypbeziehungen eindeutig sind. Über Testfälle und Gegenbeispiele sollten Sie schwere Fehler selbst finden können.

eindeutig

Selbstkontrolle