7. Programmieraufgabe

Objektorientierte Programmiertechniken

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

Themen:

kovariante Probleme, mehrfaches dynamisches Binden

Ausgabe:

07. 12. 2021

Abgabe (Deadline):

14. 12. 2021, 10:00 Uhr

Abgabeverzeichnis:

`Aufgabe7`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 3.3.3 und 3.4

Kontext

Bei der Pflege von Jungbäumen, die sich erst gegen Konkurrenz durchsetzen müssen, ist die Beschattung zu berücksichtigen. Wir unterscheiden Lichtbaumarten, die sich bei ausreichend Licht durchsetzen, von Schattenbaumarten, die mit Beschattung zurechtkommen, aber bei ausreichend Licht verdrängt werden. Von einem Jungbaum einer Lichtbaumart ist die Zahl der ausreichend mit Licht versorgten Blätter (ganze Zahl) bekannt, von einem Jungbaum einer Schattenbaumart die Wuchshöhe in Metern (Fließkommazahl). Unter den Jungbäumen der Lichtbaumarten betrachten wir Birken (`Betula`), die auch auf kargen Böden schnell wachsen und später Holz niedriger Qualität liefern, sowie Eichen (`Quercus`), die gute Böden benötigen, langsam wachsen, aber später sehr wertvolles Holz liefern. Unter den Jungbäumen der Schattenbaumarten betrachten wir Buchen (`Fagus`), die Bäume mit wertvollem Holz und dichten Kronen hervorbringen, unter denen wieder nur Buchen wachsen können, sowie Hainbuchen (`CarpinusBetulus`), die später auch im Schatten Holz mittlerer Qualität liefern. Wir unterscheiden drei Arten der Beschattung: Beschattung durch etablierte Buchen (`BelowFagus`), durch andere etablierte Bäume (`BelowNonFagus`) und ohne Beschattung (`OpenArea`). Abhängig davon können sich nur folgende Arten von Jungbäumen etablieren:

`BelowFagus`:	`Fagus`
`BelowNonFagus`:	`Fagus`, `CarpinusBetulus`
`OpenArea`:	`Betula`, `Quercus`

Bei der Pflege der Jungbäume wird Einfluss darauf genommen, welche Bäume sich etablieren, wobei die genannten Einschränkungen auf jeden Fall eingehalten werden müssen (für `BelowFagus` nur `Fagus`, etc.). Man bevorzugt Eichen gegenüber Birken und Hainbuchen gegenüber Buchen, damit der Charakter des Walds erhalten bleibt und weil der Klimawandel in Österreich Buchen durch verstärkten kontinentalen Einfluss gefährdet.

Für eine Simulation nehmen wir an, dass alle Jungbäume des Walds in einer Liste verwaltet werden (eine gemeinsame Liste für alle vier Arten von Jungbäumen). Jeder Jungbaum enthält die ganzzahligen Koordinaten seines Standorts, wobei das verwendete zweidimensionale Koordinatensystem so grobmaschig ist, dass viele Jungbäume am gleichen Standort wachsen. Eine weitere Datenstruktur verwaltet die Beschattung des Waldbodens für alle gültigen Koordinaten des Walds (Methode `get` gibt Objekte von `BelowFagus`, `BelowNonFagus` und `OpenArea` zurück). Methoden simulieren das Wachsen und Durchforsten der Jungbäume (Entfernen überzähliger oder ungeeigneter Bäumchen).[1] Weitere Methoden ändern die Beschattung (durch Fällen eines etablierten Baums bzw. Etablierung eines Jungbaums an den entsprechenden Koordinaten).

[1] Die in der Aufgabe geforderte rigorose Durchforstung der Jungbäume gibt es in der Praxis kaum. Stattdessen verlässt man sich auf natürliche Selektion, die jedoch zu komplex ist, um sie hier zu simulieren. Aktuelle Bewirtschaftungsformen greifen nur an wenigen Stellen durch Knicken oder Entfernen unerwünschter Bäumchen ein.

Welche Aufgabe zu lösen ist

Entwickeln Sie ein Java-Programm, das die Pflege von Jungbäumen simuliert, wie unter „Kontext“ beschrieben. Das Programm soll aus den dort genannten Typen (`BelowFagus`, `BelowNonFagus`, `OpenArea`, `Fagus`, `CarpinusBetulus`, `Betula`, `Quercus`), wenn nötig zusätzlichen Obertypen davon, einer Liste[2] aller Jungbäume, einer Datenstruktur zur Verwaltung der Beschattung, den zur Simulation benötigten Methoden und einer Klasse `Test` zum Testen der Lösung bestehen. Details der Programmstrukturen bleiben Ihnen überlassen, aber es darf nur eine einzige Liste aller Jungbäume geben und die Lösung soll ganz auf (mehrfachem) dynamischem Binden beruhen; genaue Einschränkungen siehe weiter unten.

nur eine Liste mit Jungbäumen

dynamisches Binden

Folgende Methoden sind an geeigneten Stellen zu implementieren:

`void fill()`: Fügt eine zufällige Zahl an Bäumchen zufällig gewählter Arten an zufällig gewählten Koordinaten zur Liste der Jungbäume hinzu. Anfangs ist die Zahl der Blätter bzw. die Wuchshöhe klein.

`void grow()`: Sorgt dafür, dass alle Bäumchen in der Liste der Jungbäume um einen zufälligen Betrag wachsen und dabei die Anzahl der Blätter bzw. die Wuchshöhe vergrößert wird.

`void thin()`: Entfernt ungeeignete und überzählige Bäumchen aus der Liste der Jungbäume:

1. Für die Beschattungsart unpassende Bäume werden entfernt.
2. Die Anzahl der Jungbäume an gleichen Koordinaten wird auf eine (selbst zu wählende) Maximalzahl reduziert. Objekte von `CarpinusBetulus` oder `Quercus` werden nur entfernt, wenn es an den gleichen Koordinaten keine Objekte von `Fagus` oder `Betula` (mehr) gibt. Von Bäumen gleicher Art bleiben jene mit größerer Blattanzahl bzw. Wuchshöhe bevorzugt erhalten.

`void establish(int x, int y)`: Der am besten geeignete Jungbaum an den Koordinaten `x`,`y` in der Liste der Jungbäume etabliert sich. Dieser Baum wird aus der Liste der Jungbäume entfernt und die Art der Beschattung an diesen Koordinaten ändert sich auf `BelowFagus` oder `BelowNonFagus`, abhängig von der Art des Baums. Die Auswahl des am besten geeigneten Baums verwendet die gleichen Kriterien wie `thin`: geeignete Art der Beschattung, Hainbuche oder Eiche bevorzugt gegenüber Birke oder Buche, unter diesen Bäumen wird jener mit der größten Blattanzahl oder Wuchshöhe gewählt.

`void cut(int x, int y)`: Der etablierte Baum an den Koordinaten `x`,`y` wird gefällt. Die Art der Beschattung ändert sich auf `OpenArea`. Es ändert sich nichts, wenn die Beschattung schon `OpenArea` war.

`... get(int x, int y)`: Liefert die Beschattungsart am Standort `x`,`y`.

`void print()`: Ausgabe des Bestands an Jungbäumen mit den bekannten Daten (Blattanzahl, Wuchshöhe), aufgeschlüsselt nach Koordinaten mit Informationen zur Art der Beschattung.

[2] „Liste“ ist hier als Datenabstraktion (abstrakter Datentyp) zu verstehen. Es muss sich nicht notwendigerweise um eine linear verkettete Liste handeln.

Die Klasse `Test` soll wie üblich die wichtigsten Normal- und Grenzfälle überprüfen und Ergebnisse in allgemein verständlicher Form darstellen. Objekte aller in der Lösung vorkommender Klassen (außer `Test`) sind zu erzeugen und alle Methodenimplementierungen auszuführen. Testfälle sind so zu gestalten, dass sich deklarierte Typen von Variablen im Allgemeinen von den dynamischen Typen ihrer Werte unterscheiden.

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

Sie dürfen folgende Sprachkonzepte *nicht* verwenden:

- dynamische Typabfragen `getClass` und `instanceof` sowie Typumwandlungen;

keine Typabfragen erlaubt

- bedingte Anweisungen wie `if`- und `switch`-Anweisungen sowie bedingte Ausdrücke (also Ausdrücke der Form `x?y:z`), die Typabfragen emulieren (zum Beispiel ist eine Objektvariable, deren Wert für einen Typ steht, nicht erlaubt und ein entsprechendes `enum` nicht sinnvoll; bedingte Anweisungen, die einem anderen Zweck dienen, sind dagegen schon erlaubt);

Verbot von Typabfragen nicht umgehen

- Werfen und Abfangen von Ausnahmen.

auch keine Umgehung durch Exception-Handling

Bauen Sie Ihre Lösung auf (mehrfaches) dynamisches Binden auf.

dynamisches Binden

Was im Hinblick auf die Beurteilung wichtig ist

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

Komplexität klein halten

- Typhierarchie sinnvoll, (mehrfaches) dynamisches Binden richtig verwendet, möglichst geringe Anzahl an Methoden — 40 Punkte
- Lösung wie vorgeschrieben und sinnvoll getestet — 20 Punkte
- Geforderte Funktionalität vorhanden — 15 Punkte
- Zusicherungen richtig und sinnvoll eingesetzt — 15 Punkte
- Sichtbarkeit auf kleinstmögliche Bereiche beschränkt — 10 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,
- unnötig große Komplexität (unnötig viele Typen oder Methoden),
- die Verwechslung von statischem und dynamischem Binden (insbesondere auch den falschen Einsatz überladener Methoden),
- 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 an Typen und Methoden ohnehin kaum vermeidbar, und durch weitere unnötige Strukturierung oder Funktionalität könnten Sie leicht den Überblick verlieren.

Verbote stets beachten

Komplexität klein halten

Es gibt mehrere sinnvolle Lösungsansätze. Bleiben Sie beim 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.

nicht zu viel probieren

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.

Inhaltliche Ähnlichkeiten mit früheren Aufgaben sind nicht zufällig. Sie sollen erkennen, dass relativ kleine Änderungen in der Beschreibung einer Aufgabe zu ganz anderen Programmiertechniken führen.

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

vorgegebene Namen

keine Umlaute