7. Programmieraufgabe

Objektorientierte Programmiertechniken

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

Themen:

kovariante Probleme, mehrfaches dynamisches Binden

Ausgabe:

25. 11. 2020

Abgabe (Deadline):

02. 12. 2020, 12:00 Uhr

Abgabeverzeichnis:

`Aufgabe7`

Programmaufruf:

`java Test`

Grundlage:

Skriptum, Schwerpunkt auf 3.3.3 und 3.4

Kontext

Ein Gartenpflegebetrieb benötigt für die Bearbeitung von Wiesen unterschiedliche Mäher. Es gibt Rasenmäher, die das Gras mähen und in einer Grasfangbox sammeln. Es gibt Mulchmäher, die das Gras mähen, fein häckseln und in der Wiese verteilen. Es gibt kleine Mäher, wo der Bediener hinter dem Mäher hergeht, und große Aufsitzmäher, bei welchen der Bediener auf dem Mäher sitzt. Von jedem Mäher ist die Anzahl der Betriebsstunden (Gleitkommazahl) bekannt. Von einem Rasenmäher ist das Volumen der Grasfangbox (Gleitkommazahl), von einem Mulchmäher ist die Schnitthöhe (ganze Zahl) bekannt. Damit ergeben sich folgende vier Mäherarten:

- Rasenmäher, klein
- Rasenmäher, groß
- Mulchmäher, klein
- Mulchmäher, groß

Es gibt Wiesen, die gemäht werden und wo das Mähgut entfernt wird. Auf diesen Wiesen werden ausschließlich Rasenmäher eingesetzt. Es gibt Wiesen, die gemulcht werden und wo das Mähgut zur Düngung auf der Wiese verbleibt. Auf diesen Wiesen werden ausschließlich Mulchmäher eingesetzt. Auf Feuchtwiesen können ausschließlich kleine Mäher eingesetzt werden, da der Boden für die schweren großen Mäher zu weich ist. Auf allen anderen Wiesen werden im Normalfall große Mäher eingesetzt, da dadurch die Einsatzzeit minimiert wird. Sind keine großen Mäher mehr verfügbar, werden ausnahmsweise auch kleine Mäher eingesetzt. Auf einer Wiese können auch gleichzeitig mehrere Mäher im Einsatz sein.

Die verfügbaren Mäher werden in einer einzigen Mäherliste eines Gartenpflegebetriebs verwaltet (eine gemeinsame Liste für alle vier Mähertypen, keine getrennten Listen).

Welche Aufgabe zu lösen ist

Entwickeln Sie ein Programm zur Verwaltung und der Einsatzplanung der Mäher und Wiesen eines Gartenpflegebetriebs. Zumindest folgende Methoden sind zu entwickeln:

- `addMower` fügt einen neuen Mäher zur Mäherliste hinzu.
- `delMower` löscht einen defekten Mäher aus der Mäherliste.
- `assignMower` gibt einen passenden Mäher für eine Wiese zurück, entfernt diesen aus der Mäherliste und ordnet diesen Mäher einer Wiese zu (auf einer Wiese können mehrere Mäher im Einsatz sein). Falls kein passender Mäher existiert, wird `null` zurückgeliefert.

- `returnMower` löscht die Zuordnung eines Mähers zu einer Wiese und fügt diesen Mäher zur Mäherliste hinzu.
- `hoursAvailable` zeigt die Summe der Betriebsstunden aller Mäher der Mäherliste auf dem Bildschirm an.
- `hoursInUse` zeigt die Summe der Betriebsstunden aller Mäher eines Gartenpflegebetriebs, die auf Wiesen im Einsatz sind, auf dem Bildschirm an.
- `mowersShow` zeigt alle Mäher der Mäherliste mit allen Informationen auf dem Bildschirm an.
- `meadowsShow` zeigt alle Wiesen eines Gartenpflegebetriebs mit allen Informationen (insbesondere welche Mäher auf dieser Wiese im Einsatz sind) auf dem Bildschirm an.

Die Mäher dürfen nur in einer einzigen Mäherliste pro Gartenpflegebetrieb gespeichert werden. Es ist nicht zulässig unterschiedliche Mäherlisten für Rasenmäher und Mulchmäher oder kleine und große Mäher zu führen.

nur eine einzige Mäherliste

Wie Sie Wiesen und die Zuordnung der Mäher gestalten oder wie Sie Betriebsstunden setzen und erhöhen, liegt in Ihrem freien Ermessen.

Die Klasse `Test` soll wie üblich die wichtigsten Normal- und Grenzfälle überprüfen und die Ergebnisse in allgemein verständlicher Form darstellen. Dabei sind Instanzen aller in der Lösung vorkommenden Typen zu erzeugen. Auch für einen Gartenpflegebetrieb mit einer Mäherliste und einigen Wiesen sind eigene Objekte zu erzeugen und mindestens 3 unterschiedliche Gartenpflegebetriebe sind zu testen. 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

In der Lösung der Aufgabe dürfen Sie 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 (d.h.: Zusätzliche Felder eines Objekts, die einen Typ simulieren und abfragen, sind nicht erlaubt; z.B.: Ein `enum` der Mäherarten ist nicht sinnvoll, weil Abfragen darauf nicht erlaubt sind; bedingte Anweisungen, die einem anderen Zweck dienen, sind dagegen schon erlaubt);
- Werfen und Abfangen von Ausnahmen.

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

Was im Hinblick auf die Beurteilung wichtig ist

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

- (mehrfaches) dynamisches Binden richtig verwendet, sinnvolle minimale Typhierarchie, möglichst geringe Anzahl an Methoden und gute Wiederverwendung 40 Punkte
- Lösung wie vorgeschrieben und sinnvoll getestet 20 Punkte
- Geforderte Funktionalität vorhanden (so wie in Aufgabenstellung beschrieben) 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,
- die Verwechslung von statischem und dynamischem Binden (insbesondere die Verwechslung überladener Methoden mit Multimethoden),
- 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 Klassen und Methoden ohnehin kaum vermeidbar, und durch weitere unnötige Strukturierung oder Funktionalität könnten Sie leicht den Überblick verlieren.

Es gibt mehrere sinnvolle Lösungsansätze. Bleiben Sie bei dem 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.

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.

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 aussagekräftige Namen achten. Sonst ist es schwierig, sich einen Überblick über Ihre Klassen und Interfaces zu verschaffen. Verwenden Sie keine Umlaute in Dateinamen (Mäher als Dateiname ist unzulässig). Achten Sie darauf, dass Sie keine Java-Dateien abgeben, die nicht zu Ihrer Lösung gehören (alte Versionen, Reste aus früheren Versuchen, etc.).

keine Umlaute