Objektorientierte Programmiertechniken
LVA 185.A01, VU, 3 ECTS, 2014/2015 W

9. Übungsaufgabe

Themen:

Design-Patterns, aspektorientierte Programmierung

Termine:

Ausgabe: 10.12.2014
Abgabe: 17.12.2014, 12:00 Uhr

Abgabeverzeichnis:

Aufgabe9

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf den Abschnitten 4.4, 5.1 und 5.2

Aufgabe

Welche Aufgabe zu lösen ist:

Der Weihnachtsmann verpackt seine Geschenke in Schachteln, die er in seinem Schachtellager gelagert hat. Geschenke und Schachteln haben unterschiedliche Formen. Die Grundfläche einer Schachtel kann entweder ein Kreis, ein gleichseitiges Sechseck, ein Quadrat oder ein Rechteck sein. Auch Geschenke haben eine Grundfläche, die entweder in einen Kreis, in ein gleichseitiges Sechseck, in ein Quadrat oder in ein nichtquadratisches Rechteck hineinpasst. Geschenke sollen so in einer Schachtel verpackt werden, dass sie nicht verrutschen können. So passt z.B. eine Weinflasche in eine Schachtel mit einem kreisförmigen, aber auch mit einem sechseckigen oder quadratischen Boden. Eine sechseckige Kerze passt in eine Schachtel mit einem sechseckigen, aber auch mit einem kreisförmigen oder rechteckigen Boden. Ein Würfel passt in eine Schachtel mit quadratischem Boden, aber nicht in eine mit rechteckigem

Boden, da der Würfel darin verrutschen würde. Ebenso passt eine Flasche nicht eine Schachtel mit rechteckigem Boden und eine sechseckige Kerze nicht in eine Schachtel mit quadratischem Boden, da sie darin verrutschen würde. Ein Buch passt nur in eine Schachtel mit rechteckigem Boden. Rechteckige oder quadratische Geschenke sollen aber nicht in kreisförmigen oder sechseckigen Schachteln verpackt werden, da zu viel Raum verloren gehen würde. Um die Spannung beim Auspacken zu erhöhen ist es sehr beliebt, ein Geschenk in mehreren Schachteln zu verpacken, z.B. etwas Wertvolles in eine kleine Schachtel, diese Schachtel in eine mittlere und diese in eine große Schachtel, d.h. auch eine Schachtel kann als Geschenk gesehen werden.

Geschenke haben einen Namen, eine Höhe (in cm) und abhängig von der Art des Geschenks Informationen über die Grundfläche (in cm) gespeichert (Kreis: Durchmesser; Sechseck, Quadrat: Seitenlänge; Rechteck: Länge und Breite). Von einer Schachtel werden die Innenmaße (in cm, Höhe und abhängig von der Art Informationen über die Grundfläche) gespeichert. Die Kartondicke beträgt 5mm, d.h. die Aussenmaße einer Schachtel sind um 1cm größer als die Innenmaße. Weiters wird zu jeder Schachtel gespeichert, welches Geschenk sich in der Schachtel befindet.

Entwickeln Sie ein Javaprogramm, das ein Schachtellager mit leeren Schachteln und einen Geschenkesack des Weihnachtsmanns, der Schachteln mit Geschenken enthält, verwalten kann. Entwickeln Sie Klassen(hierarchien) um Geschenke, Schachteln, das Lager und den Geschenkesack darzustellen. Geschenke und Schachteln, ein Lager und ein Geschenkesack sollen erzeugt werden. Das Lager und der Geschenkesack sollen befüllt werden.

Weiters sollen folgende Methoden und Funktionalitäten unterstütz werden:

- Eine Methode *verpacke* für ein Geschenk, die eine passende Schachtel aus dem Lager (Geschenk darf nicht verrutschen können) für das Geschenk liefert und das Geschenk in die Schachtel gibt. Gibt es keine passende Schachtel im Lager (für Geschenke mit runder Grundfläche weder eine Schachtel mit runder noch mit sechseckiger oder quadratischer Grundfläche; für Geschenke mit sechseckiger Grundfläche weder eine Schachtel mit sechseckiger noch mit runder oder rechteckiger Grundfläche), soll eine neue Schachtel passender Form und Größe erstellt werden.
- Eine Methode *volumen* für eine Schachtel, die das Volumen einer Schachtel zurückliefert (Außenmaße). Das Volumen berechnet sich aus Grundfläche*Höhe. Flächenformeln für Kreis und Sechseck (auch der Innkreisradius) finden sich unter anderem in der Wikipedia.
- Eine Methode *volumen* für den Sack, die die Summe der Volumen aller im Sack befindlichen Schachteln zurückliefert.
- Eine Methode *inhalt* für den Sack, die für alle Schachteln im Sack den Namen der darin enthaltenen Geschenke auf die Standardausgabe ausgibt.

Entwickeln Sie eigene Klassen für Geschenke (Bücher, Bälle, ...), mindestens zwei von unterschiedlicher Größe für jede Art von Grundflächen. Sorgen Sie dafür, dass zwischen allen Klassen für Geschenke und Schachteln, die zueinander in einer Untertypbeziehung stehen können, auch tatsächlich Untertyp- und Vererbungsbeziehungen bestehen.

Schreiben Sie eine Klasse *Test*, die das Lager, den Geschenkesack, alle Klassen von Geschenken und Schachteln und alle Untertypbeziehungen testet.

Da in dem Programm viele Objekte erzeugt werden, soll die Anzahl der dynamisch erzeugten Objekte gezählt werden. Kapseln sie das Zählen der Objekte in einem Aspekt und geben Sie die Anzahl am Ende des Programms am Bildschirm aus.

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

Warum die Aufgabe diese Form hat:

Diese Aufgabe enthält Details, die den Einsatz bestimmter Entwurfsmuster nahelegen. Durch mehrere Arten von Schachteln, Geschenken, etc. ergibt sich eine umfangreiche Typstruktur. Alle Schwierigkeiten in der Aufgabe sollten mit bekannten Mitteln leicht zu lösen sein. Das gibt Ihnen zum Abschluss der Laborübung Gelegenheit zu zeigen, dass Sie schöne, einfache und gut wartbare Programme schreiben können.

Was im Hinblick auf die Beurteilung zu beachten ist:

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

Entwurfsmuster richtig verwendet, sinnvolle, korrekte Klassenhierarchien	30 Punkte
Aspektorientierte Programmierung richtig eingesetzt	20 Punkte
Lösung wie vorgeschrieben und sinnvoll getestet	20 Punkte
Geforderte Funktionalität vorhanden (so wie in Aufgabenstellung beschrieben)	15 Punkte
Zusicherungen richtig und sinnvoll eingesetzt	10 Punkte
Sichtbarkeit auf so kleine Bereiche wie möglich beschränkt	5 Punkte

Kräftige Punkteabzüge gibt es für

- Verletzungen des Ersetzbarkeitsprinzips bei Verwendung von Vererbungsbeziehungen,
- den Verzicht auf mögliche Untertypbeziehungen,

- fehlende, falsche, nichtssagende und unverständliche Zusicherungen,
- unnötigen Code und mehrfache Vorkommen gleicher oder ähnlicher Code-Stücke,
- komplizierte und schwer verständliche Programmstrukturen, die sich leicht aus nicht oder falsch eingesetzten Entwurfsmustern ergeben,
- unpassende Sichtbarkeit.

Wie die Aufgabe zu lösen ist:

Es ist wichtig, alle Untertypbeziehungen zu finden und die Ersetzbarkeit auf geeignete Weise (durch durchdachte Zusicherungen, aber auch durch Testfälle) sicherzustellen. Vererbungsbeziehungen, die nicht gleichzeitig auch Untertypbeziehungen sind, müssen unbedingt vermieden werden. Andererseits sollen überall Untertypbeziehungen hergestellt werden, wo das möglich ist. Überlegen Sie sich, wo Sie (abstrakte) Klassen und Interfaces verwenden. Beachten Sie die Sichtbarkeit. Benutzen Sie Generizität und vorgefertigte Containerklassen, wo dies sinnvoll erscheint.

Diese Aufgabe enthält wahrscheinlich weniger inhaltliche Schwierigkeiten als vorangegangene Aufgaben. Gerade deswegen sollten Sie ein schönes, einfaches und gut wartbares Programm abliefern, da solche Qualitätskriterien stärker in die Beurteilung einfließen als gewohnt.

Achten Sie darauf, dass Sie den Programm-Code so klein und einfach wie möglich halten. Jede unnötige Anweisung kann eine versteckte Fehlerquelle darstellen.

Was im Hinblick auf die Abgabe zu beachten ist:

Dieses Programm wird nicht mit dem Standardcompiler *javac*, sondern mit dem Compiler *ajc* übersetzt. Damit die Übersetzung funktioniert, müssen alle Klassen gemeinsam mit dem Aufruf übersetzt werden, z.B. mit einem Aufruf *ajc* **.java*. Wenn Sie unbedingt wollen, können Sie Packages verwenden, aber dann müssen Sie auch die Dateien in Unterverzeichnissen in den *ajc*-Aufruf einbeziehen. Testen Sie ihre Lösung auf der g0, da sich die Version von *ajc* auf der g0 eventuell von der Version unterscheidet, die Sie zu Hause verwenden. Üblicherweise ist bei einer Standardinstallation von Java *ajc* nicht inkludiert, laden Sie sich für zu Hause *ajc* von der Eclipse Webseite. Dort findet sich auch ein detailierter Programming Guide. 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. Achten Sie darauf, dass Sie keine Java-Dateien abgeben, die nicht zu Ihrer Lösung gehören (alte Versionen, Reste aus früheren Versuchen, etc.).