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

4. Übungsaufgabe

Themen:

Untertypbeziehungen, Zusicherungen

Termine:

Ausgabe: 29.10.2014
Abgabe: 05.11.2014, 12:00 Uhr

Abgabeverzeichnis:

Aufgabe4

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf Kapitel 2

Aufgabe

Welche Aufgabe zu lösen ist:

Jahreszeitlich bedingt geht mit der Temperatur auch der Umsatz des Surfstores deutlich zurück. Deshalb wird das Angebot auf Wintersportartikel ausgeweitet. Entsprechende Softwareunterstützung wird benötigt.

Folgendes Interface ist vorgegeben:

```
public interface Artikel {
    Zustand zustand();
    // gibt den aktuellen Zustand wider:
    // - Artikel ist verleihbar,
    // - Artikel ist verliehen,
    // - Artikel ist benutzt (nach Retournierung).

    void verleihe(String kunde);
```

```
    // verleihe Artikel an Kunden namens kunde;
    // nur aufrufbar wenn Artikel verleihbar ist;
    // Artikel kommt in den Zustand verliehen.

    String kunde();
    // Name des Kunden, an den Artikel verliehen;
    // nur aufrufbar wenn Artikel verliehen ist.

    void retour();
    // nur aufrufbar wenn Artikel verliehen ist;
    // Artikel kommt in den Zustand benutzt.

    boolean kontrolle();
    // nur aufrufbar wenn Artikel benutzt ist;
    // falls Artikel als verleihbar einstufbar ist
    // - kommt Artikel in den Zustand verleihbar und
    // - es wird true zurueckgegeben;
    // sonst: Ergebnis false und Zustand unveraendert.
}
```

Für `Zustand` ist ein geeigneter Typ vorzusehen. Unter welchen Bedingungen `kontrolle()` als Ergebnis `true` liefern kann, ist von der Art des Artikels abhängig. Die Anzahl der Verleihungen sowie der technische und hygienische Zustand spielen dabei möglicherweise eine Rolle. Folgende Untertypen von `Artikel` sollen als Klassen, abstrakte Klassen oder Interfaces erstellt werden:

- Ein Objekt des Typs `FunSet` stellt ein Paar Skier inklusive Bindungen und Stöcken, ein Snowboard inklusive Bindungen oder ähnliches dar. Je nach Produkt wird bereits bei der Erstellung des Objekts eine maximale Anzahl an Verleihungen festgelegt, die nicht überschritten werden darf. Folgende Methoden sind nur in benutztem Zustand aufrufbar: Die Methode `void service()` sollte spätestens nach jeder dritten Retournierung aufgerufen werden. Die Methode `kontrolle()` gibt beeinflusst durch Zufall `true` oder `false` zurück, aber `true` darf nur zurückgegeben werden, wenn die maximale Anzahl an Verleihungen noch nicht überschritten und spätestens nach jeweils drei Verleihungen zuvor `service()` aufgerufen wurde.
- Ein Objekt des Typs `ProfiSet` stellt ebenso Skier, ein Snowboard oder ähnliches (inklusive Zubehör) dar. Die Methoden entsprechen denen von `FunSet`, jedoch gibt es keine Obergrenze für die Anzahl der Verleihungen und `kontrolle()` kann nur dann `true` zurückgeben, wenn nach jeder Retournierung zuvor `service()` aufgerufen wurde.
- `Set` ist ein gemeinsamer Obertyp von `FunSet` und `ProfiSet`. Zusicherungen und Methoden sind möglichst konkret zu wählen, soweit dies ohne Verletzung des Ersetzbarkeitsprinzips möglich ist.
- Ein Objekt des Typs `Schutz` stellt eine Schutzausrüstung dar, etwa ein Lawinensuchgerät, einen Helm oder einen Rückenprotektor, aber auch Skischuhe und Skier mit Sicherheitsbindungen oder mehrere Produkte als Set, wovon mindestens eines zur Schutzausrüstung zählt. Aus Sicherheitsgründen gibt es für solche Artikel eine vom Produkt abhängige maximale Anzahl an Verleihungen. Bei Überschreitung dieser Anzahl muss `kontrolle()` stets `false` zurückgeben.

- Ein Objekt des Typs `Bekleidung` stellt einen Gegenstand dar, der üblicherweise am Körper getragen wird, also Bekleidung im weitesten Sinne (inklusive Schuhen). Nach jeder Retournierung muss mindestens einmal die Methode `void desinfiziere()` aufgerufen werden. Andernfalls muss `kontrolle()` verpflichtend `false` zurückgeben.
- Objekte des Typs `Schutzbekleidung` wie etwas Skischuhe und Helme fallen gleichzeitig in die Kategorie `Schutz` und `Bekleidung`.
- Objekte des Typs `Jacke` gehören zwar zur Kategorie `Bekleidung`, aber nicht zur `Schutzbekleidung`.

Versehen Sie (abstrakte) Klassen und Interfaces mit allen notwendigen Zusicherungen und stellen Sie sicher, dass Sie nur dort eine Vererbungsbeziehung (*extends* oder *implements*) verwenden, wo tatsächlich eine Untertypbeziehung auch hinsichtlich der Zusicherungen besteht. Ermöglichen Sie Untertypbeziehungen zwischen allen diesen Typen, außer wenn Untertypbeziehungen den Beschreibungen der Typen widersprechen würden.

Falls zwischen zwei Typen keine Untertypbeziehung besteht, geben Sie in einem Kommentar in der Datei *Test.java* eine Begründung dafür an. Bitte geben Sie eine textuelle Begründung; auskommentierte Programmzeilen reichen dafür nicht.

Sie können so viele zusätzliche (abstrakte) Klassen und Interfaces einführen, wie Sie als vorteilhaft erachten. Die Typstruktur soll trotzdem möglichst einfach und klein bleiben, wobei jedoch alle oben genannten Typen (mit den vorgegebenen deutschen Namen) vorkommen müssen, auch solche, die Sie vielleicht für nicht nötig erachten.

Schreiben Sie eine Klasse *Test* zum Testen Ihrer Lösung. Erzeugen Sie Instanzen der oben genannter Typen. Überprüfen Sie so gut Sie können mittels Testfällen, ob dort, wo Sie eine Untertypbeziehung annehmen, Ersetzbarkeit gegeben ist.

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.

Wie die Aufgabe zu lösen ist:

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

Untertypbeziehungen richtig erkannt und eingesetzt, fehlende Untertypbeziehungen richtig beschrieben	50 Punkte
Zusicherungen richtig und sinnvoll eingesetzt	25 Punkte
Lösung sinnvoll getestet	10 Punkte

Vollständigkeit der Lösung (so wie in Aufgabenstellung beschrieben) 15 Punkte

Fehler in Untertypbeziehungen sowie ungeeignete Zusicherungen führen zu einem sehr hohen Punkteverlust. Obwohl für das Testen der Lösung nur 10 Punkte veranschlagt sind, kann unzureichendes Testen doch zu größerem Punkteverlust führen, wenn dadurch bestehende Mängel nicht rechtzeitig erkannt werden. Auch Mängel in der Funktionalität können einen Verlust von deutlich mehr als 15 Punkten bedeuten, weil diese sehr wahrscheinlich auch aus Fehlern in Untertypbeziehungen und Zusicherungen resultieren.

Die größte Schwierigkeit liegt darin, *alle* Untertypbeziehungen zu finden und Ersetzbarkeit sicherzustellen. Vererbungsbeziehungen, die nicht gleichzeitig auch Untertypbeziehungen sind, führen zu sehr hohem Punkteverlust. Ebenso gibt es hohe Punkteabzüge für nicht wahrgenommene Gelegenheiten, Untertypbeziehungen zwischen den Untertypen von *Artikel* herzustellen, sowie für fehlende oder falsche Begründungen für nicht bestehende Untertypbeziehungen. Geeignete Begründungen wären etwa Gegenbeispiele.

Eine Grundlage für das Auffinden der Untertypbeziehungen sind gute Zusicherungen. Wesentliche (aber nicht alle) Zusicherungen kommen in obigen Beschreibungen vor. Untertypbeziehungen ergeben sich aus den erlaubten Beziehungen zwischen Zusicherungen in Unter- und Obertypen. Es hat sich als günstig erwiesen, alle Zusicherungen, die in einem Obertyp gelten, im Untertyp bei den betroffenen Methoden nochmals hinzuschreiben, da sie sonst leicht übersehen werden.

Vergewissern Sie sich der Korrektheit der Untertypbeziehungen zusätzlich über Testfälle. Die Anzahl der Testfälle ist nicht entscheidend, wohl aber deren Qualität: Es kommt darauf an, dass die Testfälle mögliche Verletzungen der Ersetzbarkeit aufdecken können. Umgekehrt sollen Sie sich auch vergewissern, dass Sie keine Gelegenheit für Untertypbeziehungen verpasst haben, indem Sie Beispiele dafür finden, wie angenommene Untertypbeziehungen das Ersetzbarkeitsprinzip verletzen würden.

Zusicherungen in Testklassen werden aus praktischen Überlegungen bei der Beurteilung nicht berücksichtigt. Sorgen Sie aber bitte dafür, dass ein Aufruf von *java Test* einigermaßen nachvollziehbaren Output generiert.

Zur Lösung dieser Aufgabe müssen Sie Untertypbeziehungen und vor allem den Einfluss von Zusicherungen auf Untertypbeziehungen im Detail verstehen. Holen Sie sich entsprechende Informationen aus Kapitel 2 des Skriptums. Folgende zusätzlichen Informationen könnten hilfreich sein:

- Konstruktoren werden in einer konkreten Klasse aufgerufen und sind daher vom Ersetzbarkeitsprinzip nicht betroffen.
- Zur Lösung der Aufgabe benötigen Sie keine Exceptions. Sollten Sie dennoch Exceptions verwenden, bedenken Sie, dass eine Instanz eines

Untertyps nur dann eine Exception werfen darf, wenn man auch von einer Instanz eines Obertyps in derselben Situation diese Exception erwarten würde.
- Mehrfachvererbung gibt es nur auf Interfaces. Sollte einer der verlangten Typen mehrere Obertypen haben, müssen alle Obertypen bis auf einen Interfaces sein. Die Obertypen sollen auch in diesem Fall so heißen wie in der Aufgabenstellung. Für die Klassen, welche diese Interfaces implementieren, sind dann andere Namen zu wählen.

Schalten Sie Warnungen des Compilers in keinem Fall aus. Viele Warnungen haben mit möglichen Verletzungen der Ersetzbarkeit zu tun.

Lassen Sie sich von der Form der Beschreibung der benötigten Typen nicht täuschen. Daraus, dass die Beschreibung eines Typs die Beschreibung eines anderen Typs teilweise wiederholt, folgt noch keine Ersetzbarkeit. Generell sind Sie wahrscheinlich auf dem falschen Weg, wenn es den Anschein hat, A könne Untertyp von B und B gleichzeitig Untertyp von A sein – außer wenn A und B gleich sind.

Achten Sie auf richtige Sichtbarkeit. Alle oben beschriebenen Typen und Methoden sollen überall verwendbar sein. Die Sichtbarkeit von Implementierungsdetails und insbesondere von Variablen soll aber so stark wie möglich eingeschränkt werden. Bedenken Sie, dass sichtbare Implementierungsdetails die Ersetzbarkeit beeinflussen.

Was man generell beachten sollte:

Es werden keinerlei Ausnahmen bezüglich des Abgabetermins gemacht. Was nicht rechtzeitig im Repository steht, wird bei der Beurteilung nicht berücksichtigt. Da es ab jetzt um 100 Punkte pro Aufgabe geht, wäre jede Ausnahme anderen Gruppen gegenüber ungerecht.

Schreiben Sie nicht mehr als eine Klasse in jede Datei (ausgenommen geschachtelte Klassen), halten Sie sich an übliche Namenskonventionen in Java (Großschreibung für Namen von Klassen und Interfaces, kleine Anfangsbuchstaben für Variablen und Methoden, etc.), und verwenden Sie die Namen, die in der Aufgabenstellung vorgegeben sind.

Übernehmen Sie das vorgegebene Interface *Artikel* unverändert, einschließlich der Kommentare im Wortlaut.

Warum die Aufgabe diese Form hat:

Diese Aufgabe ist ziemlich klar spezifiziert. Der Grund liegt darin, dass die Beschreibungen der Typen nur wenig Interpretationsspielraum bezüglich der Ersetzbarkeit bieten sollen. Die Aufgabe ist so formuliert, dass Untertypbeziehungen (abgesehen von Typen, die Sie vielleicht zusätzlich

einführen) eindeutig sind. Über Testfälle und Gegenbeispiele sollten Sie in der Lage sein, schwerwiegende Fehler selbst zu finden.