Objektorientierte Programmiertechniken
LVA 185.A01, VU, 3 Ects, 2012/2013 W

6. Übungsaufgabe

Themen:

kovariante Probleme, mehrfaches dynamisches Binden

Termine:

Ausgabe:	21.11.2012
reguläre Abgabe:	28.11.2012, 12:00 Uhr
nachträgliche Abgabe:	05.12.2012, 12:00 Uhr

Abgabeverzeichnis:

Gruppe/Aufgabe6

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf den Abschnitten 3.3.3 und 3.4

Aufgabe

Welche Aufgabe zu lösen ist:

Robies Robo Shop vertreibt und repariert Androiden aller Art. Folgende Grundtypen von Androiden werden angeboten:

- *Bediener* sind leichtgewichtige, flinke und ausdauernde Androiden, die normalerweise in der Nähe von und in direktem Kontakt zu Menschen eingesetzt werden. Man unterscheidet *Hilfskräfte* (beispielsweise für Reinigungsaufgaben) von *Gesellschaftern* (beispielsweise zur Unterhaltung oder Kinderbetreuung).
- *Schwerarbeiter* sind kräftige, ausdauernde Androiden, die mit schweren Lasten umgehen können und den Kontakt zu Menschen aus Sicherheitsgründen meiden. Unterschieden werden *Bauarbeiter*, *Service-*

Techniker und *Transportarbeiter*.
- *Beschützer* sind kräftige und flinke Androiden, die zum Schutz von Menschen und Objekten Einsatz finden. Außer in gefährlichen Situationen meiden sie den Kontakt zu Menschen. Unterschieden werden *Objektbewacher*, *LeibwÃ¤chter* und *Kämpfer*.

Zur Identifikation besitzt jeder Androide eine eindeutige Seriennummer. Vor dem Verkauf wird jeder Androide mit aufgabenspezifischen Sensoren und Aktoren, einer Skin sowie entsprechender Software ausgestattet, und alle Teile werden mit der Seriennummer des Andoiden codiert um unauthorisierte Manipulationen zu unterbinden. Entsprechend der Androide-Verordnung sind genaue Aufzeichnungen über die Anfangskonfiguration und alle Änderungen zu führen.

Implementieren Sie einen Teil der zum Führen dieser Aufzeichnungen nötigen Software. Diese muss auch dafür sorgen, dass die in der Androide-Verordnung festgelegten Einschränkungen eingehalten werden:

- Jeder Androide muss mit einer *berührungssensitiven*, *hochfesten* oder *gepanzerten* Skin ausgestattet sein. Bediener benötigen unbedingt eine berührungssensitive Skin, und nur Beschützer dürfen eine gepanzerte Skin haben.
- Die Software muss dem Einsatzgebiet des Androiden entsprechen. Analog zu den Androide-Typen gibt es *Hilfskraft-Software*, *Gesellschafter-Software*, *Bauarbeiter-Software* und so weiter.
- Die Software muss entsprechend der Sicherheitsstufen 1 bis 5 zertifiziert sein. Gesellschafter benötigen Software der Stufe 1, alle anderen Bediener Software der Stufen 1 oder 2. Schwerarbeiter können mit Software der Stufen 3 und 4 ausgestattet sein. Kämpfer müssen Software der Stufe 5 haben, alle anderen Beschützer Software der Stufe 4.
- Die Leistung aller Aktoren eines Bedieners darf zusammen die Grenze von 1 kW nicht überschreiten. Für Kämpfer ist die Leistung nicht begrenzt. Bei allen Androiden, die mit Software der Stufe 3 ausgestattet sind, darf die Grenze von 5 kW nicht überschritten werden, bei Androiden mit Software der Stufe 4 die Grenze von 10 kW.
- Nachträgliche Änderungen der Androiden sind nur auf eingeschränkte Weise erlaubt: Die Seriennummer, der Haupttyp (Bediener, Schwerarbeiter oder Beschützer) sowie die Sicherheitsstufe der Software dürfen nicht geändert werden.

Konkret soll eine Liste aller ausgelieferten Androiden geführt werden, die Folgendes erlaubt:

- Die Methode *insert* fügt einen Androiden mit eindeutiger Seriennummer und allen Ausstattungsdetails in die Liste ein und prüft die Bedingungen der Androide-Verordnung. Sind die Bedingungen nicht erfüllt, bleibt die Liste unverändert, und ein entsprechender Fehlercode wird

zurückgegeben. Kommt ein Androide mit derselben Seriennummer bereits in der Liste vor, so handelt es sich um eine Änderung, sonst um die Auslieferung eines neuen Androiden. Als Ausstattungsdetails bekommt jeder Android ein Sensoren-Aktoren-Kit, eine Skin und eine Software mit.
- Die Methode *find* liefert einen String mit der Beschreibung aller Ausstattungsdetails des Androiden mit der als Parameter gegebenen Seriennummer zurück (oder *null* falls kein Androide mit dieser Seriennummer existiert).
- Die Methode *iterator* erzeugt einen Iterator über den ausgelieferten Androiden in der Reihenfolge der ersten Auslieferung (das ist die Reihenfolge des Einfügens neuer Androiden).

Androiden sowie Sensoren-Aktoren-Kits, Skins und Software sollen wie Listen ausgelieferter Androiden jeweils als Objekte unterschiedlicher Typen dargestellt werden. Zur Unterscheidung zwischen verschiedenen Arten von Objekten soll nur die in jedem Objekt vorhandene dynamische Typinformation dienen.

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. Testfälle sind so zu gestalten, dass sich deklarierte Typen von Variablen im Allgemeinen von den dynamischen Typen ihrer Werte unterscheiden.

In der Lösung der Aufgabe dürfen Sie folgende Sprachkonzepte nicht verwenden:

- dynamische Typabfragen über *getClass* und *instanceof* sowie Typumwandlungen,
- bedingte Anweisungen wie *if*- und *switch*-Anweisungen sowie bedingte Ausdrücke (der Form x ? y : z), erlaubt sind solche Anweisungen und Ausdrücke jedoch für Vergleiche mit *null*,
- Schleifenkonstrukte, deren Zweck in der Emulation bedingter Anweisungen besteht,
- das Werfen und Abfangen von Ausnahmen.

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

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 das Verbot bedingter

Anweisungen und bedingter Ausdrücke 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 führt.

Was im Hinblick auf die Beurteilung zu beachten ist:

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 Ã¼berladener 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 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 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).

Ein vollständiger Verzicht auf bedingte Anweisungen kann ungeahnte Schwierigkeiten nach sich ziehen. Verzichten Sie trotzdem auf bedingte Anweisungen und umgehen Sie die Schwierigkeiten mittels kreativer Lösungen, vor allem durch dynamisches Binden. Bedingte Anweisungen, die nur auf *null* vergleichen, sind zwar erlaubt, sollten aber nur in begrenztem Umfang eingesetzt werden. Eine bedingte Anweisung lässt sich durch eine Schleife mit Abbruchbedingung simulieren, aber genau solche Schleifen sind nicht erlaubt. Verwenden Sie Schleifenkonstrukte nur in Fällen, in denen Schleifenrümpfe wiederholt ausgeführt werden.

Halten Sie die Anzahl der Klassen, Interfaces und Methoden möglichst klein und überschaubar. Durch die Aufgabenstellung ist eine große Anzahl davon ohnehin kaum vermeidbar, und durch weitere 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.

Was im Hinblick auf die Abgabe zu beachten ist:

Verzichten Sie wie üblich auf die Verwendung von packages und Verzeichnissen innerhalb des Abgabeverzeichnisses. 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.).