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

4. Übungsaufgabe

Themen:

Untertypbeziehungen, Zusicherungen

Termine:

Ausgabe:	07.11.2012
reguläre Abgabe:	14.11.2012, 12:00 Uhr
nachträgliche Abgabe:	21.11.2012, 12:00 Uhr

Abgabeverzeichnis:

Gruppe/Aufgabe4

Programmaufruf:

java Test

Grundlage:

Skriptum, Schwerpunkt auf Kapitel 2 sowie Teile von Abschnitt 3.1

Aufgabe

Welche Aufgabe zu lösen ist:

Folgendes Interface ist vorgegeben:

```
public interface Pict {
    // instances represent 2-dimensional pictures
    // consisting of printable characters
  String toString();
    // returns the picture as String
  void scale(double factor);
    // 0.1 <= factor <= 10.0; resize the picture
}
```

Es sollen folgende Typen (also Klassen, abstrakte Klassen oder Interfaces) als Untertypen von *Pict* erstellt werden:

- Instanzen von *Box* stellen aus druckbaren Zeichen geformte Rechtecke dar. Höhe und Breite (jeweils als Anzahl der Zeichen angegeben) stehen in einem fixen Verhältnis zueinander und werden im Konstruktor gesetzt. Die Ränder des Rechtecks können aus anderen Zeichen bestehen als der Inhalt. Als Zeichen für die Ränder ist kein Leerzeichen erlaubt, für den Inhalt schon. Die beiden zu verwendenden Zeichen werden im Konstruktor gesetzt und bleiben danach unverändert. Die Methode *scale* multipliziert die Seitenlängen mit dem als Parameter übergebenen Faktor. Erst bei Umformung in eine Zeichenkette durch *toString* werden Höhe und Breite auf ganze Zahlen aufgerundet (also in Richtung größerer Zahlen). Beispielsweise gibt *toString* bei Höhe 2.3 und Breite 3.7 mit den Zeichen "o" und "." folgende Zeilen zurück:

  ```
  oooo
  o..o
  oooo
  ```

  Sind Höhe oder Breite nicht größer als 2.0, besteht das Ergebnis nur aus Zeichen für die Ränder.
- Instanzen von *ClearBox* ähneln jenen von *Box*, jedoch wird als Zeichen für die Ränder immer "*" verwendet und für den Inhalt das Leerzeichen. Zusätzlich gibt es eine Möglichkeit, das Seitenverhältnis abzufragen (Breite dividiert durch Höhe).
- Auch Instanzen von *DarkBox* ähneln jenen von *Box*, jedoch entspricht das Zeichen für den Inhalt stets dem Zeichen für die Ränder. Zusätzlich gibt es eine Möglichkeit, dieses Zeichen jederzeit zu ändern.
- Instanzen von *FreeBox* stellen ebenfalls aus druckbaren Zeichen geformte Rechtecke mit fixem Seitenverhältnis dar, die nur durch *scale* geändert werden können. Alle Zeichen des gesamten Rechtecks werden durch Ãœbergabe eines (rechteckigen) Textes an den Konstruktor gesetzt. Die anfängliche Höhe und Breite ergeben sich aus diesem Text. Aufrufe von *scale* ändern die Seitenlängen, ohne jedoch den Text selbst zu ändern. Falls die aktuellen Seitenlängen kleiner sind als der Text, gibt *toString* nur das linke obere Eck des Textes in der entsprechenden Größe (aufgerundet) zurück. Bei größeren Seitenlängen wird der Text so oft wie nötig neben- bzw. übereinandergestellt. Liefert *toString* beispielsweise für die nicht skalierte Instanz von *FreeBox* diesen Text:

  ```
  1234
  5678
  ```

  so liefert *toString* nach Skalierung mit dem Faktor 1.5 Folgendes:

  ```
  123412
  567856
  123412
  ```

- Instanzen von *Repeated<P>* (wobei *P* ein beliebiger Typ ist) enthalten ein zweidimensionales, rechteckiges Array von Objekten des Typs *P*, das im Konstruktor gesetzt und nicht mehr verändert wird. Ein Aufruf von *toString* gibt (ohne vorherigen Aufruf von *scale*) die Elemente des Arrays

nebeneinander und übereinander liegend zurück (genauer gesagt die Ergebnisse der Aufrufe von *toString* in den Elementen). Haben die Elemente unterschiedliche Höhe bzw. Breite, werden die kleineren Elemente unterhalb bzw. rechts mit Leerzeichen aufgefüllt. Ähnlich wie bei *FreeBox* ändern Aufrufe von *scale* die Größe, ohne jedoch das Array zu verändern. Ist der durch (wiederholte) Aufrufe von *scale* gesetzte Skalierungsfaktor kleiner 1.0, so liefert *toString* nur die entsprechend große linke obere Ecke. Ist der Skalierungsfaktor größer als 1.0, so wird der Text entsprechend oft neben- bzw. übereinander wiederholt.
- Instanzen von *Scaled<P>* (wobei *P* ein Untertyp von *Pict* ist, kann auch *Pict* selbst sein) ähneln denen von *Repeated<P>*, jedoch funktioniert *scale* anders: Jeder Aufruf von *scale* wird mit demselben Argument direkt an die einzelnen Elemente des Arrays weitergeleitet. Ein Skalierungsfaktor ist in Instanzen von *Scaled<P>* daher nicht nötig (bzw. ist er immer gleich 1.0).

Versehen Sie (abstrakte) Klassen und Interfaces mit allen notwendigen Zusicherungen (entsprechend obigen Beschreibungen) und stellen Sie sicher, dass Sie nur dort eine Vererbungsbeziehung (*extends* oder *implements*) verwenden, wo tatsächlich eine Untertypbeziehung (auch hinsichtlich der Zusicherungen) besteht.

Untersuchen Sie, ob zwischen je zwei der folgenden Typen eine Untertypbeziehung besteht und dokumentieren Sie die Ergebnisse in der Datei *Test.java*:

- Box
- ClearBox
- DarkBox
- FreeBox
- Repeated<P> für unbekannte P
- Repeated<P> für jeden einzelnen Untertyp P von Pict
- Repeated<Char>
- Scaled<P> für unbekannte erlaubte P
- Scaled<P> für jeden einzelnen Untertyp P von Pict

Im Detail gilt Folgendes:

- Wenn zwei dieser Typen aufgrund der Schnittstellen und Zusicherungen äquivalent sein können (das heißt, ein Typ ist Untertyp des anderen und der andere gleichzeitig Untertyp des einen), muss es in *Test.java* einen entsprechenden Kommentar geben. Eine mögliche Untertypbeziehung kann durch *extends* oder *implements* im Programmcode eingeführt werden, muss aber nicht.
- Wenn zwei dieser Typen aufgrund der Schnittstellen und Zusicherungen in Untertypbeziehung zueinander stehen, soll dies in der Typstruktur des Programms (durch *extends* oder *implements*) sichtbar sein. Sollte dies aufgrund von Einschränkungen in Java nicht möglich sein, dann muss es

einen entsprechenden Kommentar dazu in *Test.java* geben.
- Wenn eine Untertypbeziehung zwischen zwei dieser Typen aufgrund der Schnittstellen und Zusicherungen nicht bestehen kann, dann ist auch dies in einem Kommentar in *Test.java* zu vermerken.

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 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.

Wie die Aufgabe zu lösen ist:

Die Schwierigkeit liegt darin, *alle* Untertypbeziehungen zu finden und Ersetzbarkeit sicherzustellen. Dieser Punkt ist wesentlich für die Beurteilung. Vererbungsbeziehungen, die nicht gleichzeitig auch Untertypbeziehungen sind, führen zu sehr hohen Punkteabzügen. Ebenso gibt es hohe Punkteabzüge für nicht wahrgenommene Gelegenheiten, Untertypbeziehungen zwischen den Untertypen von *Pict* herzustellen. Nennenswerte Abzüge gibt es auch für mangelhafte Zusicherungen. Die direkte Codewiederverwendung durch Vererbung spielt für die Beurteilung dagegen nur eine untergeordnete Rolle.

Eine Grundlage für das Auffinden der Untertypbeziehungen sind gute Zusicherungen. Wesentliche (aber nicht alle) Zusicherungen kommen bereits in obigen Beschreibungen der benötigten Typen vor. Sie brauchen diese Beschreibungsteile nur mehr richtig zuzuordnen. 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 direkt bei den betroffenen Methoden nochmals hinzuschreiben, da sie sonst leicht übersehen werden.

Vergewissern Sie sich der Korrektheit der Untertypbeziehungen zusätzlich über geeignete 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. Schreiben Sie die Gegenbeispiele als Kommentare in die Testklasse.

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.

Achten Sie besonders auf die Untertypbeziehungen zu generischen Klassen. Beispielsweise ist X zwar im Allgemeinen kein Untertyp von Y<A> für beliebige A, aber X könnte trotzdem Untertyp von Y<Z> fÃ¼r ein spezielles Z (vielleicht auch von Y<X>) sein.

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 und Abschnitt 3.1 des Skriptums. Folgende zusätzlichen Informationen könnten hilfreich sein:

- Konstruktoren werden in einer konkreten Klasse aufgerufen und sind daher vom Ersetzbarkeitsprinzip nicht betroffen. Konstruktoren haben wie statische Methoden keinen direkten Einfluss auf Untertypbeziehungen.
- 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.

Bedenken Sie, dass sich viele Warnungen des Compilers im Zusammenhang mit Generizität auf Verletzungen der Ersetzbarkeit beziehen und daher schwerwiegende Fehler im Sinne dieser Aufgabe darstellen. Vermeiden Sie daher diesbezügliche Meldungen des Compilers. Achtung: Übersetzen Sie die Klassen mittels *javac -Xlint:unchecked *.java*; dieses Compiler-Flag schaltet genaue Compiler-Meldungen im Zusammenhang mit Generizität ein. Andernfalls bekommen Sie auch bei schweren Fehlern vom Compiler nur eine harmlos aussehende Meldung (“Note: …”). Entsprechende Überprüfungen durch den Compiler dürfen nicht ausgeschaltet werden.

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, obwohl A und B ungleich 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.

Was man generell beachten sollte:

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

Bitte verwenden Sie in dieser und den folgenden Aufgaben keine Unterverzeichnisse im Abgabeverzeichnis (und damit auch keine Pakete). Das Verbot von Unterverzeichnissen hat sich zur Vermeidung vielfältiger Probleme bei der Abgabe und Beurteilung bewährt.

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. Damit erhöhen Sie die Lesbarkeit Ihrer Programme ganz wesentlich. Außerdem können derartige “Fehler” zu Punkteverlusten führen.

Übernehmen Sie das vorgegebene Interface *Pict* bitte unverändert. Sie müssen die Datei selbst erzeugen, da im Abgabeverzeichnis vorinstallierte Dateien leicht zu Problemen bei der Abgabe führen könnten.

Warum die Aufgabe diese Form hat:

Im Gegensatz zu vielen anderen Aufgaben ist diese Aufgabe 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 und Typäquivalenz (abgesehen von Typen, die Sie vielleicht zusätzlich einführen) eindeutig sind. Über Testfälle und Gegenbeispiele in Kommentaren sollten Sie in der Lage sein, große Fehler in der Struktur der Lösung selbst zu finden. Eine Voraussetzung für das Erkennen der richtigen Lösung und deren Eindeutigkeit ist aber ein gutes Verständnis der Ersetzbarkeit. Wenn Ihnen mehrere Lösungsmöglichkeiten (hinsichtlich der Struktur ohne eigene Typen) als gleichermaßen richtig erscheinen, sollten Sie nocheinmal ins Skriptum schauen.