Vorlesung „Java I“ SS 04 7. Übung (7. 6. 2004)

http://www.coli.uni-sb.de/cl/courses/java-04/

---

1 MutableInteger (3 Punkte)

Schreibe eine Version der Klasse `MutableInteger` aus der Vorlesung, die den repräsentierten Integer-Wert intern in einem Feld der Klasse `Integer` repräsentiert. Dies ist z.B. sinnvoll, wenn man `MutableInteger` als Implementierung eines generischen Interfaces `MutableNumber<Integer>` auffassen will, was wir hier aber nicht machen.

Die Klasse soll die Methoden `Integer get()` und `void set(Integer)` analog zur Vorlesung implementieren. Wie bisher soll garantiert sein, dass man die Werte eines `MutableInteger`-Objekts `obj` und seines Klons `obj.clone()` unabhängig ändern kann, ohne den Wert des anderen Objekts zu ändern. Dazu musst Du ggf. die `clone`-Methode überschreiben.

Außerdem sollen folgende Methoden definiert sein:

- eine `intValue`-Methode, die den repräsentierten Wert als `int`-Wert zurückgibt;
- eine `equals`-Methode, die zwei `MutableInteger`-Objekte als gleich betrachtet, wenn sie den gleichen Zahlenwert enthalten.

Schreibe ein Hauptprogramm, das Deine Klasse testet, insbesondere auf korrektes Verhalten von `clone`.

2 Typsicherheit ohne Generics (3+1 Punkte)

Schreibe eine Klasse `StringIntegerMap`, die das Interface `Map<String,Integer>` implementiert, indem sie intern ein privates Feld von Typ `Map<Object,Object>` verwendet und ihre Methodenaufrufe (mit geeigneten Casts) an das gekapselte `Map`-Objekt weiterreicht.

Der Hintergrund dieser Aufgabe ist, dass vor Java 1.5 keine Generics verfügbar waren, d.h. es gab nur ein Interface `Map`, das in etwa dem 1.5-Interface `Map<Object,Object>` entspricht. Damit waren Fehler möglich, die heute der Compiler per Typchecking abfangen kann. Klassen wie `StringIntegerMap` waren ein beliebtes Mittel, solche statische Typisierung zu ermöglichen.

Bonuspunkt: Unter Verwendung von `StringIntegerMap` könntest Du das Programm für die 1. Aufgabe der 5. Übung fast unverändert übernehmen. Was für eine grundsätzliche Änderung müsstest Du an allen Aufrufen der `get`-Methode der `Map` vornehmen, wenn Du einfach eine `Map<Object,Object>` verwenden würdest, um die Tabelle von Wortanzahlen darzustellen?

3 Bilderkennung (4+2 Punkte)

Eine der Aufgaben zur Vorbereitung des diesjährigen Internet Programming Solving Contest hatte mit der Erkennung von Gesichtern in einer Bitmap zu tun (`http://ipsc.ksp.sk/contests/ipsc2004/practice/problems/s.php`).

Ein Teilproblem dieser Aufgabe war es, in der Bitmap die *Gesichtselemente* wiederzufinden. Ein Gesichtselement ist eine zusammenhängende Menge von schwarzen Bildpunkten, d.h. von jedem Bildpunkt zu jedem anderen Bildpunkt gibt es einen Weg (aus waagerechten, senkrechten oder diagonalen Schritten), der nur durch Bildpunkte aus dem Gesichtselement läuft.

Schreibe ein Programm, das aus einer gegebenen Bitmap die Liste aller verschiedenen Gesichtselemente berechnet. Du kannst zum Einlesen der Datei die Klasse `LineByLine` aus einer der früheren Übungen verwenden. Anders als auf der IPSC-Webseite angegeben, wird in der ersten Zeile der Datei nur die Anzahl n der Bildzeilen stehen. Dann folgen n Zeilen, in denen die Ziffer 1 einen schwarzen und 0 einen weißen Punkt darstellt. Alle Bildzeilen sind gleich lang.

Hinweis: Ich würde das Problem so angehen, dass ich eine Klasse für Bildpunkte und eine Klasse für Gesichtselemente schreiben würde, die Mengen von Bildpunkten enthält. Dann würde ich ein zweidimensionales `boolean`-Array für die Original-Bitmap anlegen sowie ein zweidimensionales Array aus Referenzen auf Gesichtselemente. Ich würde zeilenweise von links nach rechts durch die Bitmap gehen und bei jedem schwarzen Punkt ein Gesichtselement anlegen. Dann würde ich dieses Gesichtselement mit seinen Nachbarn *verschmelzen*, indem ich alle Bildpunkte der Nachbarn in meine eigene Menge übernehme und alle Referenzen im Array auf meine Nachbarn durch Referenzen auf mich selbst ersetze.

Bonuspunkte: Schreibe zusätzlich eine Methode, die entscheidet, ob ein Gesichtselement im Sinne der IPSC-Aufgabenstellung in einem anderen enthalten ist.

4 Restrekursion (0+1 Punkte)

In den meisten funktionalen Programmiersprachen gibt es Optimierungen, die es dem Compiler erlauben, *restrekursive (tail-recursive)* Funktionen in Schleifen zu übersetzen. Eine restrekursive Funktion ist eine, in der alle Ergebnisse eines rekursiven Aufrufs ohne weitere Berechnung direkt als Ergebnis der Funktion zurückgegeben werden. Durch die Übersetzung in eine Schleife kann die Rekursion viel effizienter werden. Vor allem muss man sich nicht alle aktiven rekursiven Aufrufe der Funktion merken, sondern kann beliebige Rekursionstiefen in konstantem Speicher ausführen.

Warum ist es für einen Java-Compiler im allgemeinen nicht möglich, Restrekursion zu optimieren? (Es hat mit Vererbung und `super` zu tun.) Mit welchem Sprachkonstrukt aus der heutigen Vorlesung könnte man es ihm doch ermöglichen?

Hinweis: Manche Implementierungen der JVM, z.B. die Sun-Implementierung unter Windows, optimieren Restrekursion zur *Laufzeit* durch sog. Just-in-time-Compilierung.

Abgabe bis 14. 6. 2004, 9 Uhr

(java-uebungen@coli.uni-sb.de)