Informatik II (D-ITET)
FS 2010
Prof. Dr. F. Mattern
A. Bernauer (bernauer@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Gruppe für Verteilte Systeme
http://www.vs.inf.ethz.ch

Übungsserie Nr. 6

Ausgabe: 14. April 2010
Abgabe: 21. April 2010

Hinweise

Für diese Serie benötigen Sie das Archiv
http://www.vs.inf.ethz.ch/edu/I2/downloads/u6.zip.

1. Aufgabe: (10 Punkte) Schnittstellen, Klassen und Typumwandlungen

(1a) *(2 Punkte)* Zeichnen Sie ein Diagramm, das folgende Schnittstellen, Klassen und Vererbungsbeziehungen veranschaulicht:

```
interface A { }
abstract class B implements A { }
interface C extends A { }
class D extends B implements C { }
class E extends B { }
class F implements C { }
```

(1b) *(2 Punkte)* Für welche der obigen Typen *T* kann man mit *new T()* **kein** Objekt erzeugen? Begründen Sie jeweils Ihre Antwort.

(1c) *(3 Punkte)* Gegeben die obigen Typdefinitionen, welche Zeilen aus *u6a1/StaticCasts.txt* sind gültig bzw. ungültig?

(1d) *(3 Punkte)* Gegeben die obigen Typdefinitionen, welche Zeilen aus *u6a1/DynamicCasts.txt* führen zur Laufzeit zu einer *ClassCastException*.

2. Aufgabe: (5 Punkte) Schnittstellen und Implementierungen

In den bisherigen Übungen haben Sie Codegerüste erhalten, die sie ausfüllen mussten. Der Grund dafür ist, dass die genaue Signatur der Klassen so sein muss, wie die jeweiligen Tests es erwarten. Diese Signatur ist eine Schnittstelle, die der Test verwendet und die die Klasse anbietet. In Java kann man diese Tatsache mit einem *interface* explizit machen. Das Bindeglied zwischen Anbieter und Verwender einer Schnittstelle ist die Fabrikmethode, die ein konkretes Objekt erstellt und als Referenz vom Typ der Schnittstelle zurückgibt. So kann man die Implementierung einer Schnittstelle vollständig vom Verwender verstecken und damit unabhängig machen.

Das *interface u6a2.IStack* spezifiziert die Schnittstelle für einen Stack, der *int*-Werte verwalten kann. Die Klasse *u6a2.Stack* implementiert diese Schnittstelle, in dem sie alle darin aufgelisteten Methoden implementiert. Die Fabrikmethode heisst *u6a2.StackFactory.create* und soll ein neues *Stack*-Objekt als *IStack*-Referenz zurück geben.

(2a) *(1 Punkt)* Implementieren sie die Fabrikmethode. Alle Tests werden nun erfolgreich bestanden.

(2b) *(2 Punkte)* Ergänzen Sie die Schnittstelle um die uns bekannte Methode *empty* inklusive Dokumentation. Nun werden sie vom Java-Compiler aufgefordert, die Methode in der Klasse *Stack* auch zu implementieren. Tun Sie das.

(2c) *(2 Punkte)* Ergänzen Sie die Klasse *u6a2.Tests* um einen Test, der überprüft, dass *empty* korrekt implementiert wurde.

3. Aufgabe: (10 Punkte) Polymorphie

Schnittstellen und abstrakte Klassen erlauben es, mehrere verschiedenartige Dinge einheitlich zu behandeln. Dafür gibt es mehrere Anwendungmöglichkeiten von denen ein paar in dieser Aufgabe behandelt werden.

(3a) *(4 Punkte)* Datencontainer wie Listen und Stacks werden generisch, wenn Sie anstatt mit *int* oder andere konkreten Typen mit *Object*s arbeiten. Die Klasse *GenericList* implementiert eine solche generische Liste und wird in den Methoden *toString*, *add* und *size* der Schnittstelle *IListUtils* verwendet. Schreiben Sie eine Implementierung dazu, indem Sie analog zu Aufgabe 2 eine konkrete Klasse, die die Schnittstelle *IListUtils* implementiert, alsauch die Fabrikmethode *ListUtilsFactory.create* schreiben.

Hinweis: die Implementierungen der Methoden sind in grossen Teilen analog zu den Implementierungen aus der letzten Übungsserie. Orientieren Sie sich daran.

(3b) *(3 Punkte)* Die Klasse *GeometricObject* ist eine abstrakte Klasse für geometrische Objekte und definert die abstrakte Methode *area*. Die Klassen *Rectangle* und *Triangle* sind konkrete Klassen, die von *GeometricObject* ableiten. Daher müssen sie die Methode *area* implementieren. Tun Sie das für beide Klassen.

Die Klasse *GeometricObject* implementiert ausserdem die Schnittstelle *Comparable*, die die Methode *smallerThen* definiert. Implementieren Sie diese Methode in der Klasse *GeometricObject*, so dass geometrische Objekte an Hand Ihres Flächeninhaltes miteinander verglichen werden. Gehen Sie dabei davon aus, dass das übergebene *Comparable*-Objekt ein *GeometricObject* ist.

(3c) *(3 Punkte)* Alle Objekte, deren Klassen die Schnittstelle *Comparable* implementieren, können durch einen generischen Sortieralgorithmus sortiert werden. Die Schnittstelle *IListUtils* definiert eine solche Methode *sort*. Ergänzen Sie Ihre Implementierung aus Teilaufgabe *a* um diese Methode. Gehen Sie dabei davon aus, dass alle Objekte der Liste *Comparable*-Objekte sind.

Hinweis: die Implementierung von *sort* ist in grossen Teilen analog zur Implementierung aus der letzten Übungsserie. Orientieren Sie sich daran.

4. Aufgabe: (freiwillig) Ein effizienter, dynamisch wachsender Stack

Sie haben bisher zwei Varianten eines dynamisch wachsenden Stacks implementiert. Nun sollen Sie noch ein letztes mal die Implementierung verbessern.

Die Klasse *u6a4.ChunkedStack* speichert die Werte in einer Liste aus Arrays, implementiert durch die Klasse *ChunkList*. Beim Erreichen der aktuellen Kapazitätsgrenze wird ein neues Array an die Liste angehängt, so dass gleich für mehrere neue Werte Platz ist. Sollte durch das Entfernen von Werten das letzte Array vollständig ungenutzt zurück bleiben, wird es aus der Liste entfernt. So kann der Stack dynamisch wachsen und schrumpfen, ohne dass bei jedem Hinzufügen oder Entfernen von Werten ein grösserer Aufwand nötig ist. Die Grösse der Arrays kann durch die Konstante *ChunkLists.chunkSize* eingestellt werden und ermöglicht einen anwendungsabhängigen Kompromiss zwischen feingranularer Speichernutzung mit häufigem Anlegen und Löschen von Objekten und grober Speichernutzung mit seltenem Zusatzaufwand.

(4a) Die Klasse *u6a4.ChunkedStack* implementiert die Schnittstelle *u6a2.IStack* aus Teilaufgabe 2. Die Methode *toString* ist schon fertig. Implementieren Sie die anderen Methoden an Hand der Vorgaben der Schnittstelle. Verwenden Sie dabei falls nötig die angebotenen Hilfsmethoden der Klasse *ChunkList*.

(4b) Testen Sie Ihre Implementierung von *u6a4.ChunkedStack* mit den Tests aus *u6a2.Tests*. Passen Sie dazu die Fabrikmethode an, so dass diese nun ein *u6a4.ChunkedStack*-Objekt anlegt und zurückgibt. Lassen Sie dabei die bisherige Implementierung als Kommentar zurück.