Informatik II (D-ITET)
FS 2009
Prof. Dr. F. Mattern
Philipp Bolliger (bolligph@inf.ethz.ch)

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

Übungsserie Nr. 6

Ausgabe: 1. April 2009
Abgabe: 08./09. April 2009

1. Aufgabe: (4 Punkte) Interfaces und Klassen

Betrachten Sie die folgenden Interface- und Klassendefinitionen:

```
interface White { }
interface Black { }

abstract class Alpha implements White { }
class Delta implements Black { }
class Omega extends Alpha implements Black { }
```

Dazu seien folgende Variablen deklariert:

```
White w;  Black b;  Alpha a;  Delta d;  Omega o;
```

Welche der folgenden Code-Fragmente sind korrekt und welche fehlerhaft? *(Korrekte Antwort: jeweils 0.5 Punkte, falsche Antwort: jeweils 0.5 Punkte Abzug).*

	korrekt	Compiler-Fehler
(1a) `w = new White();`	○	○
(1b) `o = new Omega();`	○	○
(1c) `o = new Alpha();`	○	○
(1d) `o = new Omega(); b = (Black)o;`	○	○
(1e) `b = new Delta(); d = (Delta)b;`	○	○
(1f) `a = new Omega(); b = a;`	○	○
(1g) `b = new Delta(); o = b;`	○	○
(1h) `a = new Omega(); o = a;`	○	○

2. Aufgabe: (7 Punkte) Generische Listen

In Java sind alle Klassen Erben der Klasse `Object` aus dem Paket `java.lang`. Deshalb und auf Grund der Zuweisungskompatibilität (siehe Skript Folie 200) ist es möglich, einer Referenzvariablen vom Typ `Object` eine Referenz auf ein Objekt einer anderen Klasse zuzuweisen, z.B.:

```
Object obj = new String( "abc" );
```

Da die Referenzvariable `obj` vom Typ `Object` ist, können durch sie keine Aufrufe auf Methoden der Klasse `String` ausgeführt werden. So führt etwa die Anweisung

```
obj.toLowerCase();
```

zu einem Compiler-Fehler (die Methode `toLowerCase()` wandelt alle Zeichen des Strings in Kleinbuchstaben um). Um dennoch auf die Methoden von `String` zugreifen zu können, muss eine explizite Typkonvertierung durchgeführt werden:

```
String str = (String)obj;
str = str.toLowerCase();
```

Diese gelingt jedoch nur, wenn es sich bei dem zu konvertierenden Objekt wirklich um ein Objekt des Zieltyps handelt. Ist dies nicht der Fall, wird eine Ausnahme (`java.lang.ClassCastException`) ausgelöst.

(2a) *(3 Punkte)* Laden Sie sich die Datei `IntList.java` von der Vorlesungs-Webseite herunter, welche mit einigen Auslassungen der Lösung der Aufgaben des letzten Übungsblattes entspricht. Ändern Sie die Klassen `IntList` und `IntListElem` so ab, dass an Stelle von `int`-Werten beliebige Objekte verwaltet werden können. Benennen Sie die geänderten Klassen in `ObjectList` und `ObjectListElem` um.

(2b) *(1 Punkt)* Die Klasse `Object` besitzt eine Methode `toString()`, welche das Objekt als `String` zurückgibt und implizit aufgerufen wird, wenn ein Objekt mit `println()` ausgegeben wird. Die Methode `toString()` ist für alle im Standard-Sprachumfang von Java enthaltenen Typen implementiert. Für eigene Klassen sollte `toString()` entsprechend selbst implementiert werden. Tun Sie dies für ObjectList (angelehnt an die Version in IntList.java).

(2c) *(2 Punkte)* Erstellen Sie nun die Methode `toVerboseString()`, welche zu jedem Listenelement auch noch dessen Typ ausgibt. Benutzen Sie dazu den Operator `instanceof`. Beschränken Sie sich auf die Typen `Boolean`, `Integer`, `String` und `Character`[1].

(2d) *(1 Punkt)* Laden Sie sich die Datei `TestObjectList.java` von der Vorlesungs-Webseite herunter. Testen Sie die Methoden `toString()` und `toVerboseString()` in der Methode `main` dieser Datei.

[1] Ein Listenelement, das nicht einer dieser Typen ist, muss als *Unkown* markiert werden

3. Aufgabe: (14 Punkte) Generisches Sortieren von Listen

In der vorigen Aufgabe haben Sie eine generische Liste für beliebige Objekte implementiert, nun sollen Sie diese Liste mit einer *generischen* Sortiermethode erweitern, die beliebige Objekte nach *derer eigenen* Ordnung sortiert.

In Aufgabe 3 des letzten Übungsblatts haben Sie eine Methode `sort()` implementiert. Die Listenelemente wurden dabei bezüglich der totalen Ordnung *kleiner* sortiert, welche in Java bei `int`-Werten mit dem Operator < geprüft werden kann. Diese Ordnungsrelation existiert natürlich nicht für alle Objekte. Allerdings gibt es in Java ein vordefiniertes Interface `Comparable` (im Package java.lang), um Objekte zu markieren, für die eine solche Ordnungsrelation definiert ist, die also *vergleichbar* sind. Dieses Interface enthält als einzige Methode:

```
public int compareTo(Object obj);
```

um das gegenwärtige Objekt mit einem anderen zu vergleichen. Unter anderem implementieren eben die vor einiger Zeit in der Vorlesung besprochenen Hüllenklassen das Interface `Comparable`: Integer, Double, String (beim Sortieren von Strings haben Sie bereits die Methode `compareTo` kennengelernt), Character usw.

Wenn Sie für eigene Klassen das Interface `Comparable` implementieren wollen (sie also vergleichbar machen wollen), muss die Methode `compareTo` folgende Bedingungen erfüllen:

$$\texttt{obj1.compareTo(obj2)} = \begin{cases} \text{negativer } \texttt{int} & \text{wenn } \texttt{obj1} \textit{ kleiner } \texttt{obj2} \\ 0 & \text{wenn } \texttt{obj1} \textit{ gleich } \texttt{obj2} \\ \text{positiver } \texttt{int} & \text{wenn } \texttt{obj1} \textit{ grösser } \texttt{obj2} \end{cases}$$

Mehr Details dazu gibt es in der *Java-API*.[2]

(3a) *(2 Punkte)* Erweitern Sie nun die Klasse `Objectlist` um eine (möglicherweise `private boolean`) Methode `checkComparable()`, die überprüft, ob alle Elemente der Liste das Interface `Comparable` implementieren. Sie können die Tatsache, dass ein Objekt ein Interface implementiert, auch mit dem `instanceof`-Operator überprüfen.

(3b) *(2 Punkte)* Erweitern Sie nun die Klasse ObjectList um eine Methode `sort()`, die nun die Elemente der Liste mittels derer `compareTo`-Relation sortiert (Sie können sich an `IntList.java` orientieren). Überprüfen Sie vorher, ob alle Elemente dieses Interface implementieren (siehe oben). Tipp: Um die Methode `compareTo` zu benutzen, müssen Sie eine explizite Typumwandlung der Listenelemente von `Object` nach `Comparable` durchführen.

(3c) *(1 Punkt)* Testen Sie Ihre Sortierfunktion in der Methode main der Datei `TestObjectList.java`, indem Sie einer **leeren** Liste beliebige `Character`-Objekte hinzufügen. Diese können Sie z.B. aus einem längeren String generieren:

[2] http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Comparable.html

```
char[] myChars="Martinique".toCharArray();
for(int i = myChars.length -1 ; i >= 0; i--) {
    objlist3.addFirst(new Character(myChars[i]));
}
```

Sortieren Sie dann diese Liste und geben Sie die sortierte und unsortierte Listen aus[3].

(3d) *(2 Punkte)* Schreiben Sie nun eine eigene Klasse `Circle`, welche auch das Interface `Comparable` implementiert. Der Konstruktor von `Circle` soll ein Argument haben, das den Radius angibt. Die Ordnung auf Kreisen sei so definiert, dass ein Kreis a genau dann kleiner (oder gleich) als Kreis b ist, wenn die Fläche von a kleiner (oder gleich) der Fläche von b ist. Geben Sie die Datei `Circle.java` Ihrem Tutor ab.

(3e) *(1 Punkt)* Erzeugen Sie (im Programm `TestObjectList.java`) 5 Objekte vom Typ `Circle` mit zufälligem Radius, fügen Sie diese in eine neue (leere) `ObjectList` ein und sortieren Sie die Liste. Geben Sie die Liste vor und nach dem Sortieren aus.

(3f) *(2 Punkte)* Schreiben Sie nun eine Klasse `Rectangle`, welche ebenfalls das Interface `Comparable` implementiert. Der Konstruktor von `Rectangle` soll zwei Argumente haben, welche die Breite und die Länge des Rechtecks angeben. Wieder soll der Flächeninhalt die Ordnungsrelation definieren. Geben Sie die Datei `Rectangle.java` Ihrem Tutor ab.

(3g) *(1 Punkt)* Erzeugen Sie nun in der Methode `main` der Datei `TestObjectList.java` ein Rechteck und fügen Sie es zu der Liste mit den Kreisen hinzu. Falls Sie jetzt sortieren, wird ein Fehler auftreten (siehe nächste Aufgabe).

Falls Sie in Ihrer Implementierung der Methode `compareTo()` in den Klassen `Circle` und `Rectangle` eine explizite Typumwandlung vornehmen, z.B. in der Klasse `Circle`:

```
public class Circle implements Comparable { ...
    public int compareTo(Object obj) {
        Circle mycircle = (Circle)obj; // ClassCastException
        if(mycircle.radius * mycircle.radius * Math.PI >.... )
    }
}
```

wird in der dritten Zeile eine `ClassCastException` auftreten (da `sort` für `obj` auch ein `Rectangle` übergeben kann, wenn denn eins in der Liste ist).

(3h) *(3 Punkte)* Implementieren Sie nun eine abstrakte Basisklasse `Shape`, von der sowohl `Rectangle` als auch `Circle` abgeleitet werden und die deren gemeinsame Funktionalität (nämlich das Berechnen einer Fläche) als abstrakte Methode enthält. Orientieren Sie sich dabei an dem Konzept der Polymorphie (Folie 204ff). Implementieren Sie nun auch den Vergleich mit `compareTo` in der allgemeinen Basisklasse `Shape` (kommentieren Sie ggf. Ihre bisherigen Implementationen von `compareTo` aus). Testen Sie in der Methode `main` in `TestObjectList.java`, ob nun eine Liste aus Rechtecken und Kreisen korrekt anhand ihrer Flächen sortiert werden kann.

[3]Das Sortieren einer Liste, die unterschiedliche Typen enthält, wird nicht funktionieren!!