Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

SS 2006
Übungsblatt 8

Informatik II Musterlösung

Zu jeder Aufgabe sind Dateien abzugeben, deren Namen rechts in der Aufgabenüberschrift stehen. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 8-1 **Auswertung** (6 Punkte, `auswertung.txt`)

Geben Sie für jedes der folgenden Java-Fragmente an, welchen Wert es ausgewertet ergibt bzw. welche Ausgabe erzeugt wird. Erklären Sie ausserdem das Ergebnis.

Tipps:

- Mit dem folgenden Eclipse-Menübefehl können Sie eine sogenannte Scrapbook-Page-Datei anlegen. Der dazugehörige Editor erlaubt es einem, interaktiv Java-Befehle auszuführen.

  File → New → Other... → Java → Java Run/Debug → Scrapbook Page

- Wenn Sie bei einem der Sprachmechanismen nicht genau Bescheid wissen, können Sie in der „The Java Language Specification“ nachschlagen: `http://java.sun.com/docs/books/jls/`

a)
```
new Integer(3) == new Integer(3)
```

Lösung: `false`, da unterschiedliche Objekte

b)
```
new Integer(3).equals(new Integer(3))
```

Lösung: `true`, da equals() (in diesem Fall) auf Wertgleichheit und nicht auf Identität untersucht.

c)
```
int i=0;
System.out.println(i++);
System.out.println(++i);
System.out.println(i);
```

Lösung:

```
0
2
2
```

Erklärung: Der Inkrement-Operator hat unterschiedliche Versionen; im ersten Fall wird der Wert vor dem Inkrement zurückgegeben, im zweiten Fall danach.

d)
```
String s = null;
if (s==null || s.length() == 0) {
    System.out.println("String ist leer");
}
if (s==null | s.length() == 0) {
    System.out.println("String ist leer");
}
```

Lösung:

```
String ist leer
java.lang.NullPointerException
...
```

Erklärung: Im ersten Fall liegt Kurzschlussauswertung vor, die den zweiten Operand nur auswertet, wenn der erste `false` ist.

e)
```
Object obj = new Integer(3);
System.out.println(obj.getClass());
```

Lösung: `class java.lang.Integer`

Erklärung: Die Klasse eines Objekts wird quasi als Instanzvariable vermerkt. Damit wird der exakte Typ eines Objekts dynamisch entschieden, was fürs dynamische Binden benötigt wird.

f)
```
String s1 = "Hallo";
String s2 = s1;
s2 += " Welt!";
System.out.println(s1);
System.out.println(s2);
```

Lösung:

```
Hallo
Hallo Welt!
```

Erklärung: Strings werden niemals (destruktiv) verändert, sondern immer komplett neu erzeugt, wie in funktionalen Programmiersprachen.

Aufgabe 8-2 **Generische Paare** (4 Punkte, `Pair.java`)

Programmieren Sie generische Paare.

a) Implementieren Sie eine Klasse `Pair`, die die Instanzvariablen `key` und `value` hat, deren Typen Parameter der Klasse sind. Die Klasse soll zwei Konstruktoren haben, einer lässt die Instanzvariablen unitialisiert, der andere setzt beide gemäß seiner Parameter. Schreiben Sie weiterhin Getter und Setter für beide Instanzvariablen. Hinweis: Im Gegensatz zur Vorlesung ist hier eine einzige Klasse (ohne Vererbung) zu implementieren, in der die Getter und Setter standardkonforme Namen haben.

b) Verwenden Sie die folgende Klasse zum Testen Ihrer Implementierung:

`PairTester.java`

```
package pair;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class PairTester {
    public static void main(String[] args) {
        List<Pair<String, Double>> food = new ArrayList<Pair<String, Double>>();

        food.add(new Pair<String, Double>("Geschnetzeltes", 7.0));
        food.add(new Pair<String, Double>("Gemüselasagne", 5.5));
        food.add(new Pair<String, Double>("Apfelkompott", 6.0));

        Collections.sort(food, new NameComparator());
        System.out.println("Sortiert nach Name:\n"+food);

        Collections.sort(food, new PriceComparator());
        System.out.println("Sortiert nach Preis:\n"+food);
    }

    private static class NameComparator implements Comparator<Pair<String, Double>> {
        public int compare(Pair<String, Double> o1, Pair<String, Double> o2) {
            return o1.getKey().compareTo(o2.getKey());
        }
```

```
        }

        private static class PriceComparator implements Comparator<Pair<String, Double>> {
            public int compare(Pair<String, Double> o1, Pair<String, Double> o2) {
                return o1.getValue().compareTo(o2.getValue());
            }
        }
    }
```

Sorgen Sie dafür, dass hierbei folgende Ausgabe erzeugt wird:

```
Sortiert nach Name:
[Pair(Apfelkompott, 6.0), Pair(Gemüselasagne, 5.5), Pair(Geschnetzeltes, 7.0)]
Sortiert nach Preis:
[Pair(Gemüselasagne, 5.5), Pair(Apfelkompott, 6.0), Pair(Geschnetzeltes, 7.0)]
```

Lösung: Pair.java

```
package pair;

public class Pair<Key,Val> {
    private Key key;
    private Val value;
    public Pair() {
    }
    public Pair(Key key, Val value) {
        this.setKey(key);
        this.setValue(value);
    }
    public void setKey(Key key) {
        this.key = key;
    }
    public Key getKey() {
        return key;
    }
    public void setValue(Val value) {
        this.value = value;
    }
    public Val getValue() {
        return value;
    }
    @Override
    public String toString() {
        return String.format("Pair(%s, %s)", getKey(), getValue());
    }
    //--------------------
    // Für die Aufgabe ist es nicht zwingend, equals() zu implementieren;
    // würde man hingegen mit Mengen arbeiten, käme man um diesen Schritt
    // nicht herum, denn dort gilt es Duplikate zu eliminieren!

    /**
     * Whenever one overrides <code>equals()</code>, one also has to override
     * <code>hashCode()</code>!
     */
    @Override
    public boolean equals(Object obj) {
        if (obj == null || (! (obj instanceof Pair))) return false;
        Pair other = (Pair) obj;
```

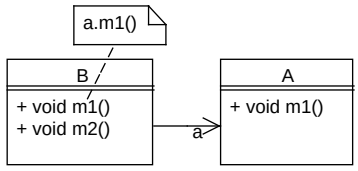
```
        return this.getKey().equals(other.getKey()) && this.getValue().equals(other.getValue());
    }
    @Override
    public int hashCode() {
       return 3*getKey().hashCode() + getValue().hashCode();
    }
}
```

Aufgabe 8-3 Delegation (6 Punkte, `SwimmingImpl.java, DrivingImpl.java, Car.java, Amphibious.java`)

Delegation ist ein Design-Pattern, das eng verwandt mit Vererbung ist: Wenn eine Klasse `B` die Methoden einer Klasse `A` übernehmen soll, setzt man bei Vererbung beide Klassen per `B extends A` in Beziehung. Bei Delegation werden ebenfalls (Instanzen) beide(r) Klassen in Beziehung gesetzt: `B` hat nun eine Instanzvariable `a` vom Typ `A`. Die in `a` enthaltene Instanz hat also sowohl den Zustand, als auch die Methoden, die man sich für Instanzen von `B` wünscht. Folglich muss man nur noch alle Methodenaufrufe, die `A` unterstützt, an `a` weiterleiten (engl. *forwarding*). Es gibt Programmiersprachen, die das automatisch erledigen. In Java muss man das von Hand machen, wird aber in Eclipse unterstützt über den Menübefehl

Source → Generate Delegate Methods...

Im folgenden sehen Sie ein Beispiel für Delegation:



`A.java`

```
package delegation;

public class A {
    public void m1() {
    }
}
```

`B.java`

```
package delegation;

public class B {
    private A a = new A();

    /** Übernommen von A per Delegation */
    public void m1() {
        a.m1();
    }

    /** Eigene Methode dieser Klasse */
    public void m2() {
    }
}
```

Zur Lösung des folgenden Problems ist (unter anderem) Delegation einzusetzen: Gegeben seien das leere Interface `Vehicle` und folgende Unter-Interfaces.

`Swimming.java`

```
package vehicle;

public interface Swimming extends Vehicle {
    public void swim();
}
```

Driving.java

```
package vehicle;

public interface Driving extends Vehicle {
    public void drive();
}
```

a) Erstellen Sie die Implementierungen `SwimmingImpl` von `Swimming` und `DrivingImpl` für `Driving`. Die Implementierungen der Methoden sollen einfach nur den Namen der aufgerufenen Methode ausgeben.

b) Implementieren Sie eine Klasse `Car`, die die Implementierung von `Driving` nutzt. Hierfür können Sie ganz normale Vererbung einsetzen.

c) Komplizierter wird es, wenn eine Klasse `Amphibious` für Amphibienfahrzeuge sowohl fahren als auch schwimmen können soll. Implementierungen kann man in Java aber nur einfach erben (im Gegensatz zu Schnittstellen). Glücklicherweise kann man per Delegation Mehrfachvererbung simulieren. Setzen Sie also Delegation ein, um `Amphibious` zu implementieren.

d) Stellen Sie sicher, dass Ihre Klassen `Car` und `Amphibious` wie folgt verwendet werden können:

VehicleTest.java

```
package vehicle;

public class VehicleTest {

    public static void testAbilities(Vehicle vehicle) {
        System.out.println("----> Object: "+vehicle);
        if (vehicle instanceof Driving) {
            System.out.println("Can drive:");
            ((Driving)vehicle).drive();
        }
        if (vehicle instanceof Swimming) {
            System.out.println("Can swim:");
            ((Swimming)vehicle).swim();
        }
    }

    public static void main(String[] args) {
        testAbilities(new Car());
        testAbilities(new Amphibious());
    }
}
```

Die `main`-Methode soll dabei folgende Ausgabe erzeugen:

```
----> Object: Car
Can drive:
drive
----> Object: Amphibious
Can drive:
drive
Can swim:
swim
```

Lösung:

`SwimmingImpl.java`

```
package vehicle;

public class SwimmingImpl implements Swimming {
    public void swim() {
        System.out.println("swim");
    }
}
```

`DrivingImpl.java`

```
package vehicle;

public class DrivingImpl implements Driving {
    public void drive() {
        System.out.println("drive");
    }
}
```

`Car.java`

```
package vehicle;

public class Car extends DrivingImpl {
    @Override
    public String toString() {
        return "Car";
    }
}
```

`Amphibious.java`

```
package vehicle;

public class Amphibious implements Driving, Swimming {
    private Driving driving = new DrivingImpl();
    private Swimming swimming = new SwimmingImpl();
    public void drive() {
        driving.drive();
    }
    public void swim() {
        swimming.swim();
    }
    @Override
    public String toString() {
        return "Amphibious";
    }
}
```

Abgabe: Per UniWorx, bis spätestens Montag, den 3.7.2006 um 9:00 Uhr.