Prof. Dr. Stefan Brass
Dipl.-Inform. Annett Thüring
Dipl.-Inform. Steffen Schiele
Dipl.-Inform. Andreas Bienert

Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik

Wintersemester 2014/15

# Objektorientierte Programmierung: Hausaufgabenblatt 10

Abgabe: 22.01.2015, 12:00 Übung: 26./27.01.2015

## Hausaufgabe 10: (4 Theoriepunkte)

Ein rätselwütiger König stellt seine Gefangenen vor die Wahl zwischen zwei Türen, die entweder in die Freiheit oder in den Tod führen. (In dem Buch "Dame oder Tiger" von Raymond Smullyan wartet hinter der einen Tür ein hungriger Tiger und hinter der anderen Tür eine heiratswillige Dame.)

An den Türen stehen Hinweise, von denen aber nur einer richtig ist, der andere ist falsch. Zum Beispiel steht an der linken Tür:

> "Diese Tür führt in die Freiheit, und die andere in den Tod."

An der rechten Tür steht:

> "Eine Tür führt in die Freiheit und eine Tür führt in den Tod."

Dies muß explizit gesagt werden, weil der König sich vorbehält, dass auch beide Türen in die Freiheit oder beide in den Tod führen können (tatsächlich ist er aber doch relativ fair).

Ziel dieser Aufgabe ist es, ein Programm zu schreiben, das Rätsel dieser Art löst. Dazu müssen die logischen Aussagen formalisiert werden. Definieren Sie dazu ein Interface "`Aussage`". Dieses Interface soll folgende Methode beinhalten:

```
boolean wahr(boolean links_frei, boolean rechts_frei)
```

Wenn also `a` eine Aussage ist, prüft man die vier möglichen Belegungen:

- `a.wahr(true, true)`: wenn dies `true` liefert, erfüllt die Situation ganz ohne Tiger die Aussage (d.h. beide Türen können aus dem Gefängnis heraus führen).
- `a.wahr(true, false)`: Wenn dies `true` liefert, ist es möglich, dass die linke Tür in die Freiheit führt, während rechts der hungrige Tiger wartet.
- `a.wahr(false, true)`: Entsprechend umgekehrt.
- `a.wahr(false, false)`: Wenn dies `true` liefert, ist es möglich, dass hinter beiden Türen der Tod wartet.

Es ist nicht ausgeschlossen, dass mehrere dieser Fälle `true` liefern. Dann enthält die Formel nicht genug Information, um die genaue Situation zu erkennen. Falls eine Tür sicher ist, würde dies ja auch reichen — man muss nicht genau wissen, ob hinter der anderen Tür ein Tiger ist.

Nun entwickeln Sie bitte Klassen, die das Interface implementieren, und zwar folgende:



- `LinksFrei`: Hier muss die Methode `wahr` das erste Argument liefern — das zweite ist ohne Belang. Der Konstruktor dieser Klasse hat keine Parameter (die Aussage "Die linke Tür führt in die Freiheit" ist atomar, d.h. enthält keine Teilaussagen).
- `RechtsFrei`: Entsprechend umgekehrt.
- `Nicht`: Der Konstruktor dieser Klasse hat einen Parameter. Man übergibt ein Objekt vom Typ "`Aussage`", das der zu negierenden Teilaussage entspricht. Z.B. könnte man eine Aussage folgendermaßen erzeugen:

  ```
  Aussage a = new Nicht(new LinksFrei());
  ```

  Die Teilaussage muss man sich natürlich in einem Attribut merken. Die Methode `wahr(boolean l, boolean r)` würde dann `wahr(l, r)` für die Teilaussage aufrufen, und das Ergebnis negieren (das geht in Java mit dem Operator "`!`").
- `Und`: Der Konstruktor dieser Klasse hat zwei Parameter vom Typ "Aussage". Man übergibt hier die beiden Teilaussagen. Beim Aufruf der Methode `wahr(l, r)` berechnet das Objekt entsprechend die Wahrheitswerte seiner beiden Teilaussagen (die der Konstruktor in zwei Attributen gespeichert hat), und verknüpft sie mit "`&`"/"`&&`".
- `Oder`: Entsprechend mit einer logischen "oder"-Verknüpfung.
- `EntwederOder`: Genauso, nur dass "exklusiv oder" verwendet wird ("`^`" in Java).

Für die drei binären Verknüpfungen ("`Und`", "`Oder`", "`EntwederOder`") definieren Sie bitte eine abstrakte Klasse "`BinAussage`", in der Sie Attribute für die beiden Teilaussagen einführen, sowie einen Konstruktor (mit der Zuweisung an die Attribute). Entweder verschieben Sie die Definition der Methode `wahr` auf die Subklassen, oder Sie führen eine abstrakte Hilfsmethode `op()` ein, die einen Code für den logischen Operator liefert (z.B. `'&'`, `'|'`, `'^'`). Dann können Sie `wahr` schon in der abstrakten Klasse definieren (mit Unterscheidung der drei Fälle). Die abstrakte Klasse kann auch schon das Interface "`Aussage`" implementieren, so dass Sie in den drei Unterklassen "`Und`", "`Oder`", "`EntwederOder`" nur noch einen Konstruktor (der den Konstruktor von "`BinAussage`" aufruft) und die Methode "`wahr`" (oder Ihre Hilfsmethode `op()`) schreiben müssen.

Zur Vereinfachung wird die Formel nicht aus einer Benutzereingabe eingelesen, sondern direkt im Programm mit folgender Testklasse codiert:

```
class Raetsel {
        // Folgende Hilfsmethoden ersparen "new" in der Formel:
        static Aussage links_frei() {
                return new LinksFrei();
        }

        static Aussage rechts_frei() {
                return new RechtsFrei();
        }

        static Aussage nicht(Aussage a) {
                return new Nicht(a);
        }
```

```
	static Aussage links_tod() {
		return nicht(links_frei());
	}

	static Aussage rechts_tod() {
		return nicht(rechts_frei());
	}

	static Aussage und(Aussage a, Aussage b) {
		return new Und(a, b);
	}

	static Aussage oder(Aussage a, Aussage b) {
		return new Oder(a, b);
	}

	static Aussage entweder_oder(Aussage a, Aussage b) {
		return new EntwederOder(a, b);
	}

	public static void main(String[] args) {

		// Inschriften der Schilder an den Tueren:
		Aussage linkes_schild =
			und(links_frei(), rechts_tod());
		Aussage rechtes_schild =
			entweder_oder(links_frei(),
					rechts_frei());

		// Ein Hinweis ist richtig, der andere falsch:
		Aussage gesamt = entweder_oder(linkes_schild,
					rechtes_schild);

		// Nun die vier Belegungen testen:
		System.out.print("Links Frei,  Rechts Frei:  ");
		System.out.println(gesamt.wahr(true,true));

		System.out.print("Links Frei,  Rechts Tiger: ");
		System.out.println(gesamt.wahr(true,false));

		System.out.print("Links Tiger, Rechts Frei:  ");
		System.out.println(gesamt.wahr(false,true));

		System.out.print("Links Tiger, Rechts Tiger: ");
		System.out.println(gesamt.wahr(false,false));
	}
}
```



Ihr Interface `Aussage`, die abstrakte Klasse `BinAussage`, und die sechs Klassen schreiben Sie bitte in eine eigene Datei “`Aussage.java`”, die Sie über die Übungsplattform abgeben. Die Datei “`Raetsel.java`” mit dem Testprogramm können Sie unter folgender URI herunterladen:

`http://www.informatik.uni-halle.de/~brass/oop14/homework/Raetsel.java`

Lassen Sie die Testklasse aber bitte in einer eigenen Datei, und fügen Sie sie nicht in Ihre Quelldatei ein. Es empfiehlt sich, für das Programm ein eigenes Verzeichnis anzulegen, denn es werden relativ viele `class`-Dateien angelegt. Sie starten das Programm dann mit

`java Raetsel`

Sie dürfen selbstverständlich in Ihren Klassen auch die Methode `toString()` definieren, was für das Debugging nützlich sein könnte. Das ist aber nicht verlangt. Wenn Sie es machen, wäre die einfachste Lösung eine vollständige Klammerung: Zum Beispiel liefern Sie bei den binären Operatoren eine Zeichenkette bestehend aus

- “(”,
- dann `toString()` für den linken Operanden,
- dann den Operator (ggf. mit Leerzeichen links und rechts),
- `toString()` für den rechten Operanden, und
- am Ende “)”.

## Übungsaufgabe 10A: (ohne Abgabe)

Bitte bearbeiten Sie die Übungsaufgaben auf dem Hausaufgabenblatt, aber geben Sie diese Aufgaben nicht ab. Diese Aufgaben werden in der Übung besprochen. Sie müssen Ihre Lösung eventuell in der Übung vorführen.

Das folgende Programm enthält mindestens 6 Fehler. Finden Sie möglichst viele davon. Das Programm besteht aus zwei Quelldateien in unterschiedlichen Verzeichnissen.

Quelldatei “`C.java`” im Verzeichnis “`P`”:

```
package P;
public class C {
        private int a = 1;
        int b = 2;
        protected int c = 3;
        public int d;

        C(int n) { d = n; }

        public  static C s() {
                return new C(0);
        }
}
```

Quelldatei “`D.java`” im Verzeichnis “`Q`”:

```
package Q;
class D extends P.C {
        public D() { super(4); }

        private void p() {
                this.a = 10;
                this.b = 20;
                this.c = 30;
                this.d = 40;
        }

        public  static void main(String[] cmdLineArgs) {
                P.C x = P.C.s();
                x.b = 50;
                x.c = 60;
                x.d = 70;
        }
}

public class C {
        public static void main(String[] args) {
                System.out.print("Hello, world!");
        }
}
```