MUSTERLÖSUNG

Softwaretechnik

14.03.2008

Prof. Dr. Walter F. Tichy
Dipl.-Inform. T. Gelhausen
Dipl.-Inform. A. Paar

Hier das Namensschild aufkleben.

Zur Klausur sind keine Hilfsmittel und kein eigenes Papier zugelassen.
Die Bearbeitungszeit beträgt 60 Minuten.
Die Klausur ist vollständig und geheftet abzugeben.
Mit Bleistift oder roter Farbe geschriebene Angaben werden nicht bewertet.

Aufgabe	**1**	**2**	**3**	**4**	∑
Maximal	18	13	17	12	**60**
K1					
K2					
K3					

Aufgabe 1: Aufwärmen (18P)

a.) Nennen Sie fünf in der Vorlesung genannte Kapitel eines Lastenheftes. (2,5P)

Zielbestimmung
Produkteinsatz
Produktfunktionen 5 x 0,5 = 2,5P
Produktdaten
Produktleistungen,
Qualitätsanforderungen, Ergänzungen, Glossar

b.) Nennen Sie drei Anforderungs-Erhebungstechniken. Geben Sie für jede Technik jeweils einen Vor- und einen Nachteil an. (3P)

z.B. Introspektion, Vorteil: einfach, schnell, billig Nachteil: oft nicht anwendbar

1 + 1 +1 = 3P

z.B. Umfragen, Fragebögen, Vorteil: kann über das Web durchgeführt werden, Nachteil: wenig Kontext

z.B. Ethnografie, Vorteil: zuverlässig, reichhaltiger Kontext, Nachteil: extrem zeitaufwendig

c.) In der Vorlesung wurden für den Vergleich zweier Objekte mehrere Stufen von Gleichheit definiert. Geben Sie die Definitionen für Gleichheit 0. und 1. Stufe an. (3P)

Gleichheit 0. Stufe:

Es handelt sich um dasselbe Objekt, die Objekte sind identisch. (1P)

Gleichheit 1. Stufe:

Es handelt sich entweder um dasselbe Objekt oder zwei verschiedene Objekte, die aber in allen Attributen/Assoziationen identische Werte besitzen (Gleichheit 0. Stufe oder paarweise Gleichheit 0. Stufe in allen Attributen). (2P)

d.) Gegeben sind die folgenden Definitionen der Java-Klassen A und B.

```
class A {
  public void print() {
    System.out.println("A");
  }
}

class B extends A {
  public void print() {
    System.out.println("B");
  }
}
```

Kreuzen Sie die Ausgabe der folgenden Java-Anweisungen an. (1P)

```
A a = new B();
a.print();
```

❑ A ✕ B ❑ AmbiguousCallException

e.) Erläutern Sie: Was ist Kontravarianz bei Eingabeparametern? Was ist Kovarianz bei Ausgabeparametern? (2P)

Kontravariante Eingabeparameter: Mit zunehmender Spezialisierung der Klasse in der eine Methode überschrieben wird nimmt die Spezialisierung des Eingabeparameters ab. (1P)

Kovariante Ausgabeparameter: Mit zunehmender Spezialisierung der Klasse in der eine Methode überschrieben wird nimmt die Spezialisierung des Ausgabeparameters zu. (1P)

f.) Wie wurde die „Benutzt“-Relation in der Vorlesung definiert? (1P)

Programmkomponente A benutzt Programmkomponente B genau dann, wenn…

A für den korrekten Ablauf die Verfügbarkeit einer korrekten Implementierung von B erfordert. (1P)

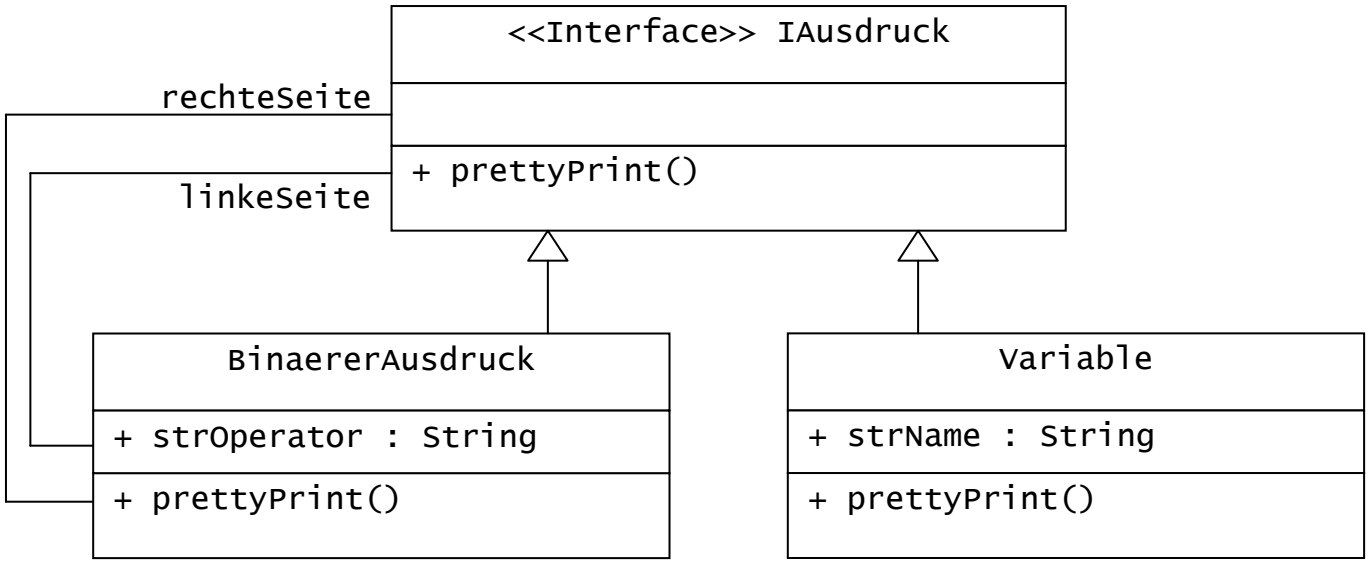
g.) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. (5,5P)

Hinweis: Jedes korrekte Kreuz zählt 0,5 Punkte, jedes falsche Kreuz bewirkt 0,5 Punkte Abzug. Die Teilaufgabe wird mindestens mit 0 Punkten bewertet.

Wahr	Falsch	Aussage
X		Ein Pflichtenheft spezifiziert die Anforderungen an eine Software in eindeutiger Weise, so dass sie implementiert werden können.
X		Ein Modul sollte ohne Kenntnis der späteren Nutzung entworfen, implementiert, getestet und überarbeitet werden können.
X		Kontravariante Eingabe-Parameter erfüllen das Substitutionsprinzip.
	X	In Java ist das Entwurfsmuster „Null-Objekt“ durch das Schüsselwort null realisiert.
	X	Das Test-Rahmenwerk JUnit erzeugt für ein gegebenes Programm automatisch Testfälle für kontrollflußorientierte Testverfahren.
X		Ein Laufzeitübersetzer (JIT) kann bestimmen, ob die ausgerollte oder die nicht ausgerollte Variante einer Schleife für die aktuelle Laufzeitumgebung besser geeignet ist.
X		Um ein Programm zu testen, können Regressionstests und Zusicherungen kombiniert werden.
X		Das Entwurfsmuster „Strategie“ bietet die Möglichkeit, eine Klasse mit einer von mehreren möglichen Verhaltensweisen zu konfigurieren.
X		Eine Fabrikmethode kann eine Einschubmethode bei einer Schablonenmethode für Objekterzeugung sein.
X		In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert.
	X	In UML-Aktivitätsdiagrammen wird nicht zwischen Objekt- und Kontrollflüssen unterschieden.

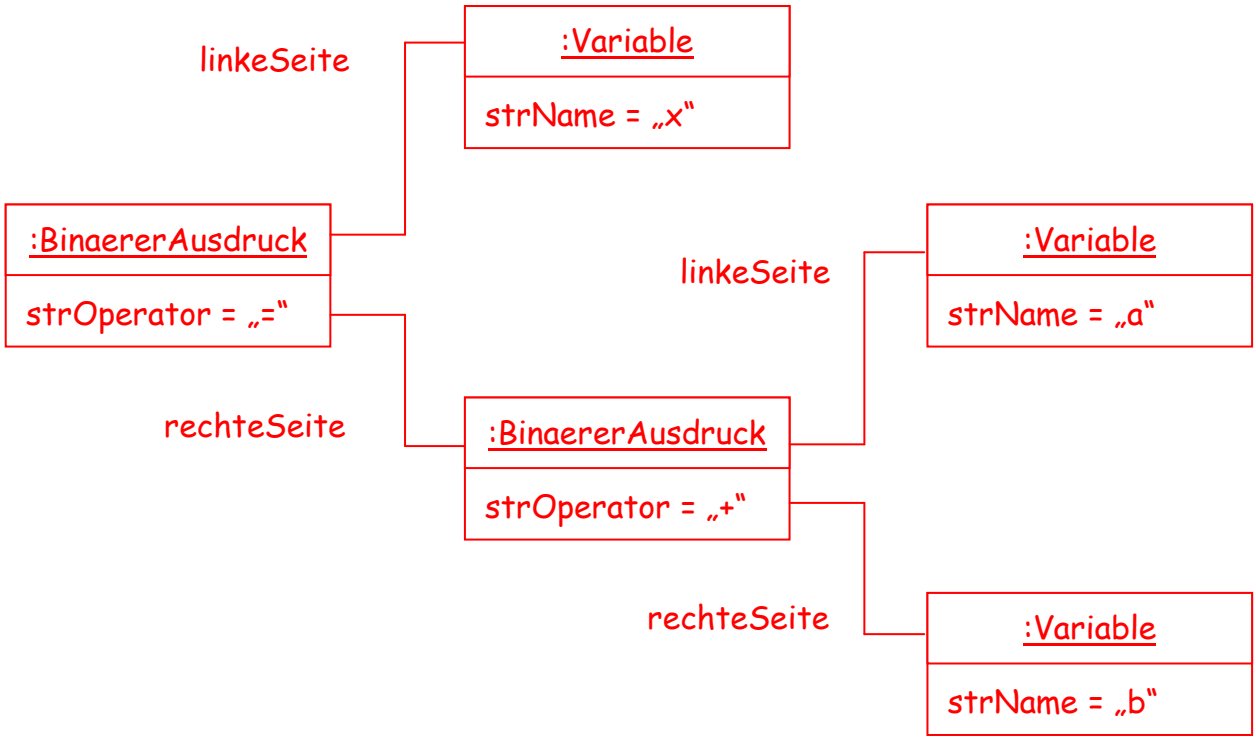
Aufgabe 2: Entwurfsmuster (13P)

Sie möchten einen Zerteiler (Parser) für arithmetische Ausdrücke programmieren. Die folgenden Java-Klassen repräsentieren die einzelnen Bestandteile solcher arithmetischer Ausdrücke. Konkrete arithmetische Ausdrücke können somit als Baum von Instanzen dieser Klassen im Speicher dargestellt werden.



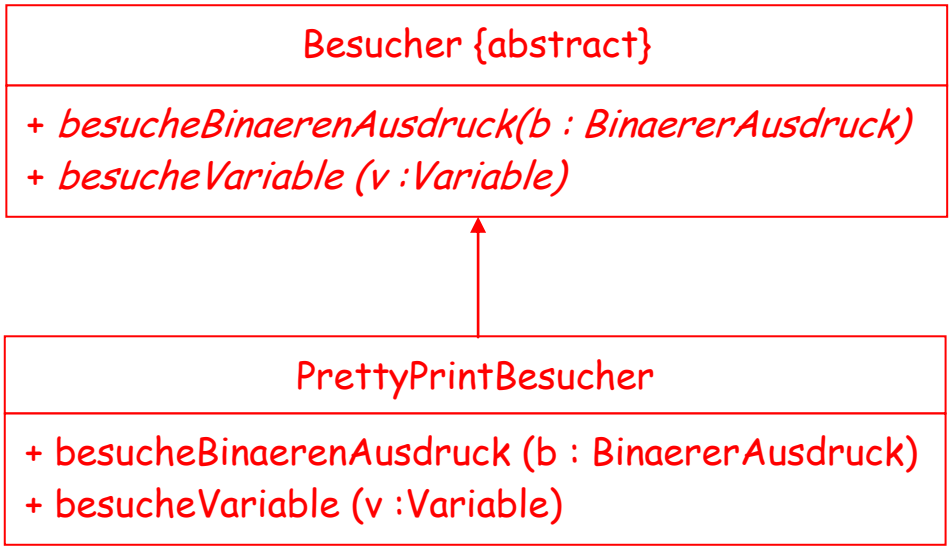
a.) Betrachten Sie den arithmetischen Ausdruck `x = a + b`. Geben Sie für diesen konkreten arithmetischen Ausdruck ein entsprechendes UML-Objektdiagramm an. Verwenden Sie die gegebenen Klassendefinitionen.

Hinweis: Sie brauchen die Objekte nicht zu benennen, sondern können anonyme Objekte verwenden. (3,5P)

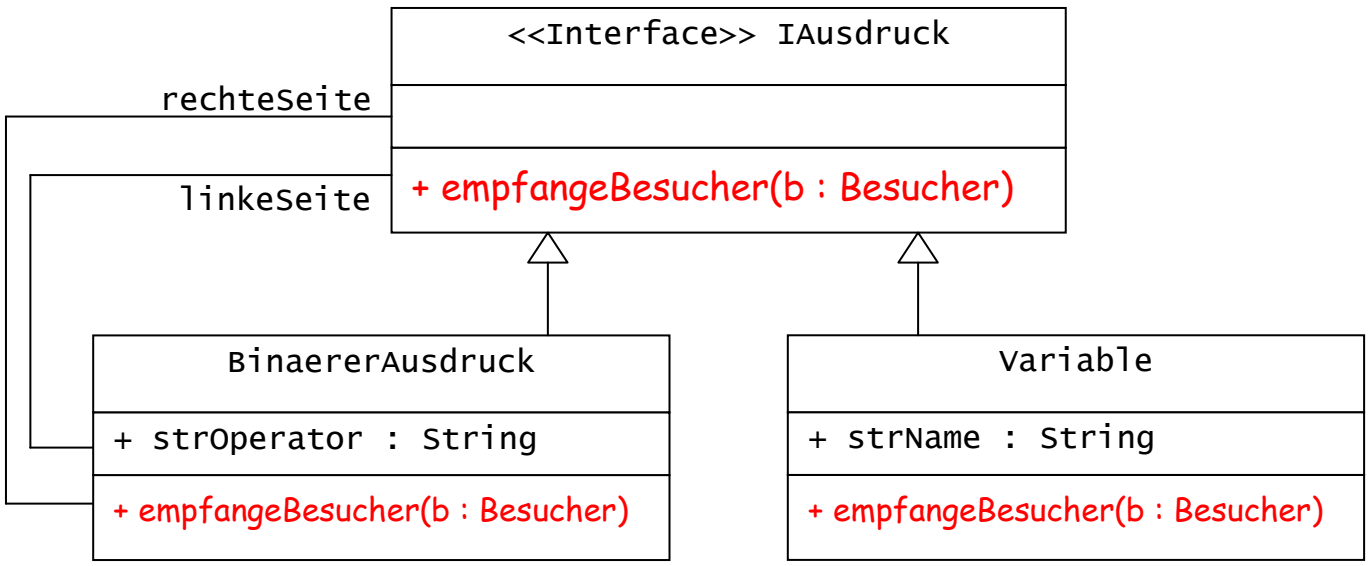


b.) Die Klassen **`BinaererAusdruck`** und **`Variable`** weisen beide die gemeinsame Funktion **`prettyPrint()`** auf. Während sich die Struktur des Klassenmodells in Zukunft nicht verändern wird, sind für die weitere Entwicklung eine Reihe von neuen Operationen für die Klassen **`BinaererAusdruck`** und **`Variable`** geplant. Sie entscheiden sich daher, vorausschauend das Entwurfsmuster *Besucher* zu verwenden und die Operation **`prettyPrint()`** in einer Besucherklasse zu kapseln.

Zeichnen Sie ein UML-Klassendiagramm, das für das gegebene Klassenmodell eine abstrakte Besucherklasse und einen konkreten **`PrettyPrintBesucher`** enthält. Deklarieren Sie alle Methoden als „public“. (3,5P)

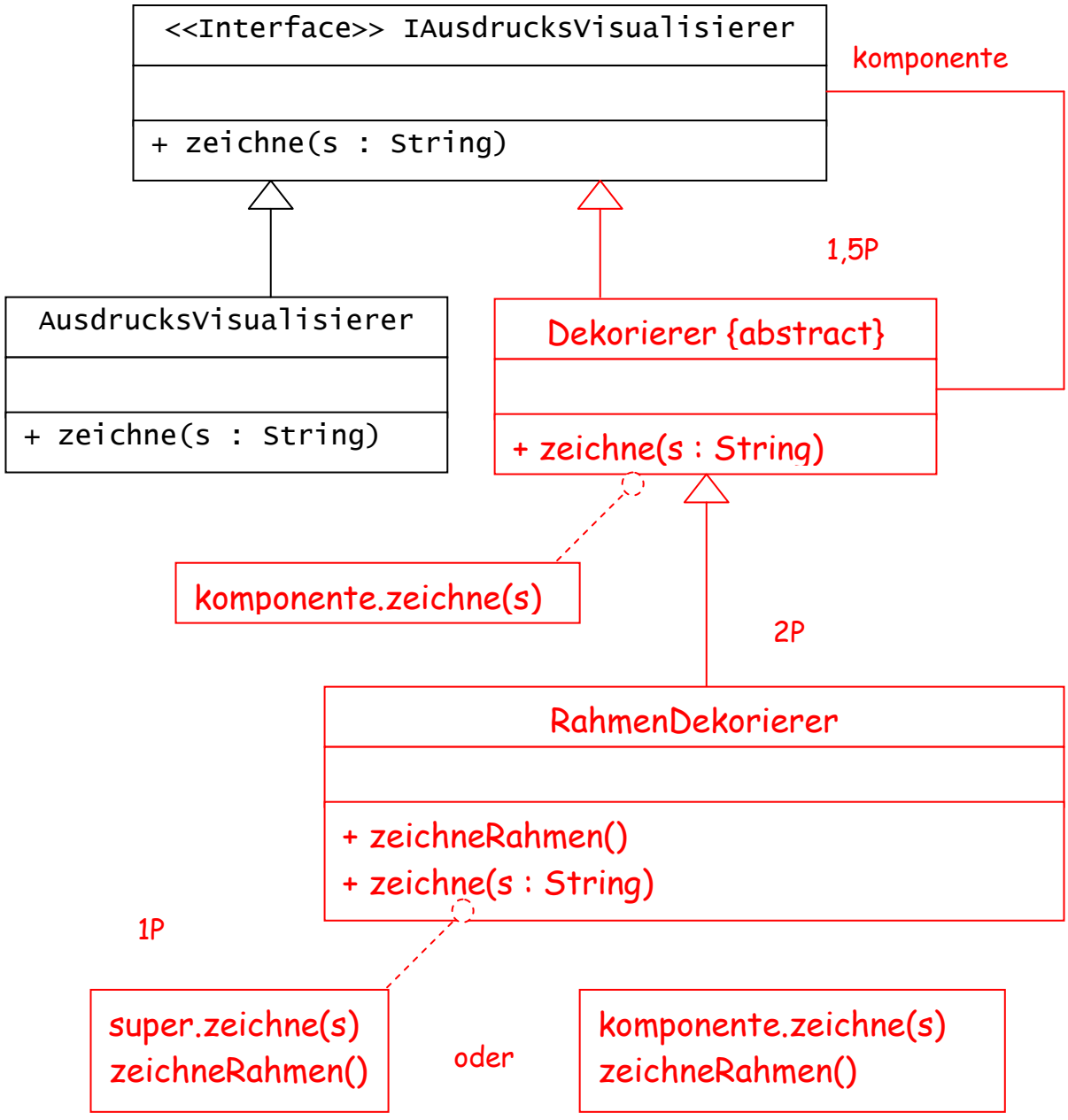


c.) Vervollständigen Sie das folgende Klassendiagramm um Funktionen, die für das Verwenden des in Aufgabenteil b.) entworfenen Besuchers notwendig sind. Deklarieren Sie alle Methoden als „public“. (1,5P)



d.) Die folgende Schnittstelle **`IAusdrucksVisualisierer`** wird von Ihrem Programm als visuelle Komponente verwendet, um arithmetische Ausdrücke auf dem Bildschirm auszugeben. Der Methode **`zeichne(s : String)`** übergibt man dazu den arithmetischen Ausdruck als Zeichenkette, die von dem konkreten **`AusdrucksVisualisierer`** dargestellt wird.

Verwenden Sie das Entwurfsmuster *Dekorierer* um einen **`RahmenDekorierer`** zu entwerfen. Dieser soll eine zusätzliche Methode **`zeichneRahmen()`** bieten, die die visuelle Komponente mit einem Rahmen dekoriert. Ergänzen Sie dazu das folgende UML-Klassendiagramm um einen abstrakten **`Dekorierer`** und um einen konkreten **`Rahmendekorierer`**. Geben Sie die Implementierung der **`zeichne(s : String)`** Methode in dem konkreten **`Dekorierer`** an (Die Implementierung der Methode **`zeichneRahmen()`** brauchen Sie nicht anzugeben). (4,5P)



Aufgabe 3: Objektorientierte Analyse & Entwurf (17P)

Gegeben sei folgender Auszug aus der Anwendungsdomänenbeschreibung von „HOAX“, einem Rechtemanagement-System für Videodateien:

> *[…]Alle Inhalte sind entweder mit dem Verfahren „AES-128“ oder „Triple-DES“ verschlüsselt. Der zu einem Inhalt gehörende Schlüssel wird vom Abspielgerät aus dem im Gerät gespeicherten und gesicherten DeviceKey und dem auf dem Medium gespeicherten MediaKey berechnet. Bei vorbespielten Medien haben alle Kopien eines Titels den gleichen MediaKey. Bei Leermedien hat jedes Medium einen eigenen, individuellen MediaKey; es gibt also keine zwei Leermedien mit dem gleichen MediaKey. Ein Aufzeichnungsgerät muss daher für jedes Leermedium einen eigenen, vom MediaKey des Mediums und vom DeviceKey abhängigen Schlüssel für den Inhalt erzeugen. […]*

a.) Nehmen wir an, HOAX soll in einer zukünftigen Version mit weiteren Verschlüsslungsalgorithmen arbeiten können. Welches Entwurfsmuster sollte dann verwendet werden? Begründen Sie Ihre Aussage. Geben Sie an, wer welche Rolle des Musters annimmt. (2P)

Hinweis 1: Das Entwurfsmuster *Oberklasse* ist nicht gemeint.

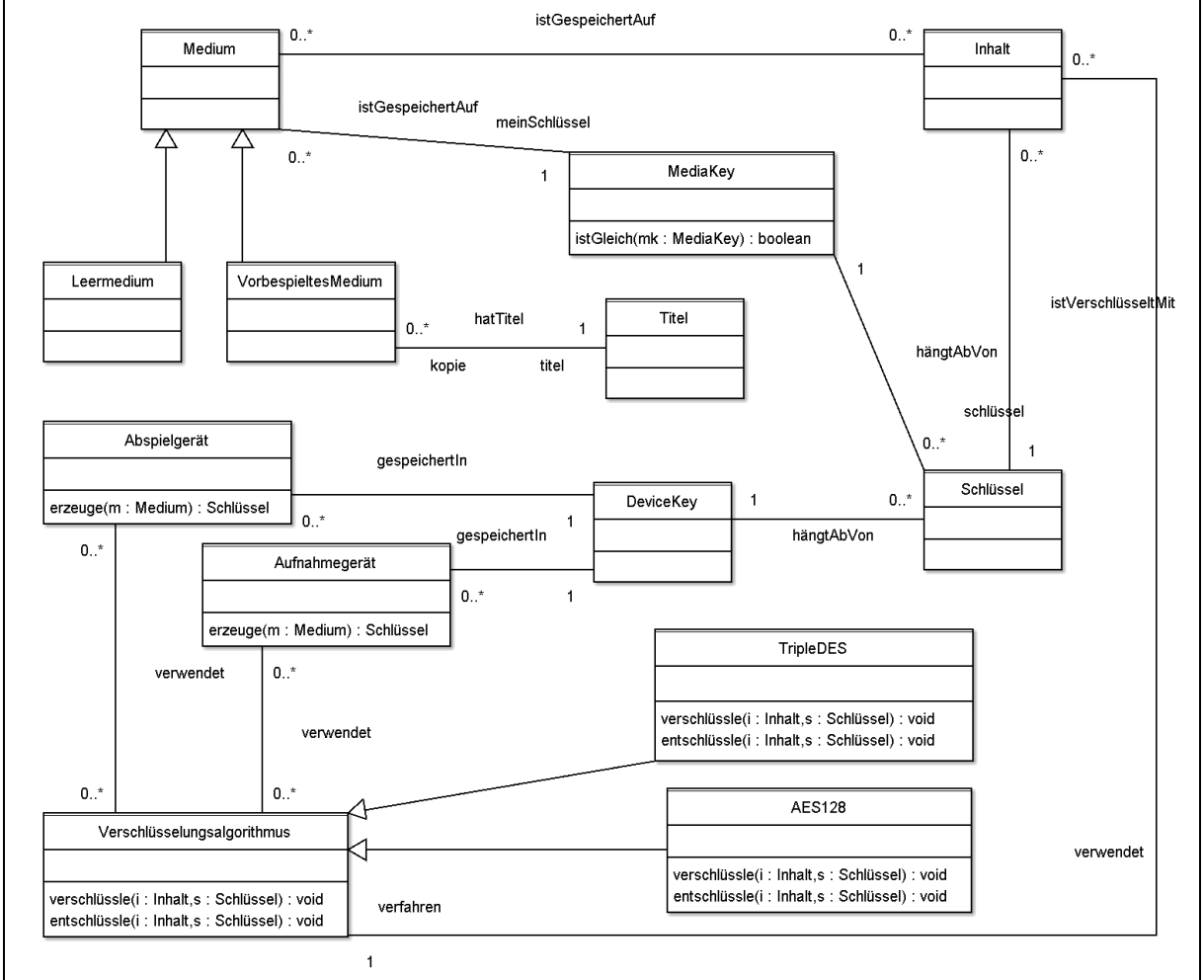
Hinweis 2: Sie sollen das Entwurfsmuster in Aufgabenteil b.) auch einsetzen.

Entwurfsmuster: Strategie. (0,5P) AES, Triple-DES und „weitere“ sind alle Verschlüsselungsalgorithmen und bilden eine Familie (0,5P). Der Klient (=Abspielgerät, 0,5P) möchte sie einheitlich verwenden und einfach austauschen (Kapselung, Zweck: Austausch/Variation, 0,5P) können.

b.) Modellieren Sie den beschriebenen Teil von HOAX möglichst genau in einem einzigen UML-Klassendiagramm. Sachverhalte, die man nicht mit dem Klassendiagramm ausdrücken kann, geben Sie bitte in OCL an. Verwenden Sie Ihr in Aufgabenteil a.) identifiziertes Entwurfsmuster und markieren Sie es deutlich. (15P)

Hinweise: Modellieren Sie nichts, das nicht im Text erwähnt wird oder für die Implementierung des Entwurfsmusters notwendig ist. Vergessen Sie insbesondere nicht die Methoden, die für das Entwurfsmuster notwendig sind. Modellieren Sie keine Sichtbarkeiten. Geben Sie aber überall Multiplizitäten und Assoziationsnamen an. Geben Sie Rollen wo nötig an.

OCL-Tipp: Die Kollektion `X.allInstances` enthält alle Instanzen der Klasse `X`.




```
context VorbespieltesMedium inv:
    VorbespieltesMedium.allInstances->forAll( other |
        (self.titel = other.titel)  implies
        self.meinSchlüssel.istGleich( other.meinSchlüssel )

context Leermedium inv:
    Leermedium.allInstances->forAll( other |
        self.meinSchlüssel.istGleich( other.meinSchlüssel )
        implies (self = other)
```

Aufgabe 4: Testen (12P)

```
public static enum Color { Cyan, Magenta, Yellow, Black };

public int applyGamma( int value, Color c, int[] gammaTable ) {
   switch( c ) {
    case Cyan:
     value *= 1.2;
     break;
    case Magenta:
     value *= value;
     // no break !!!
    case Yellow:
     value *= 1.7;
     break;
    case Black:
     value = (int)Math.sqrt( value );
     break;
   }
   value = gammaTable[value];
   return value;
}
```

a.) Beschreiben Sie ein Verfahren, wie man die in der Vorlesung nicht behandelte **`switch`**-Anweisung aus Java in die strukturerhaltende Zwischensprache aus der Vorlesung überführen kann. Geben Sie zur Veranschaulichung das entstehende Zwischensprachprogramm an. (insges. 9P)

Hinweis: Den **`default`**-Fall brauchen Sie nicht zu beachten.

Beschreibung der Umsetzungsvorschrift (5P):

Man verwandelt die switch-Anweisung in eine Kaskade von bedingten Sprüngen in einen Codeblock, der alle inneren Instruktionen der switch-Anweisung in der selben Reihenfolge enthält (1P). Jeder Fall wird zu einem bedingten Sprung (1P). Jedes „label" (case XY: …) wird zu einem Sprungziel (1P). Jedes break zu einem unbedingten Sprung an das Ende des ganzen Blocks (1P). Bei einem fehlenden break (1P) ist nichts zu tun („fall-through").

Zwischensprachprogramm (4P):

```
public int applyGamma( int value, Color c, int[] gammaTable ) {
```

```
10:   if (c==Cyan) goto 60;
20:   if (c==Magenta) goto 80;
30:   if (c==Yellow) goto 90;
40:   if (c==Black) goto 110;
50:   goto 130;       // falls kein Fall zutrifft
60:   value *= 1.2;
70:   goto 130;
80:   value *= value;
90:   value *= 1.7;
100:  goto 130;
110:  value = (int)Math.sqrt( value );
120:  goto 130;
130:  value = gammaTable[value];
      (1P pro korrekt umgesetztem Fall)
```

```
      return value;
}
```

b.) Übertragen Sie Ihre Erkenntnisse aus Teilaufgabe a.) auf den allgemeinen Fall: Was ist Voraussetzung, wenn man Zweigüberdeckung für eine beliebige **`switch`**-Anweisung erreichen will? Und unter welchen speziellen Voraussetzungen erfüllen Testdaten, die für eine **`switch`**-Anweisung Anweisungsüberdeckung erreichen, gleichzeitig auch die Kriterien der Zweigüberdeckung? (3P)

Hinweis: Den **`default`**-Fall brauchen Sie nicht zu beachten.

Um bei einem switch-Ausdruck Zweigüberdeckung zu erreichen, muss jeder Fall ein Mal angesprungen werden (1P). Testdaten für die Anweisungsüberdeckung erfüllen auch dann die Voraussetzungen für die Zweigüberdeckung, wenn kein „break" fehlt (kein „fall-through") (1P) und keine weiteren Verzweigungen in den Code-Blöcken vorkommen (1P). (Hinweis: Fehlende breaks führen dazu, dass Anweisungsüberdeckung leichter zu erfüllen ist, Zweigüberdeckung dann aber gerade nicht mehr erfüllt wird!)