Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

31. Januar 2011

Aufgaben zur Klausur **Softwaredesign** im WS 2010/11 (WI h253, MI h405, BInf v310, BMInf v300, BWInf v310)

Zeit: 75 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 10 Seiten.

Aufgabe 1:

Gegeben sei der folgende Java–Code

```
abstract class A {
    abstract void f();
}

class B extends A {
    void f() { }
}

class C extends A {
    A d;

    void f() { d.f(); }
}
```

Geben Sie das zugehörige OMT– oder UML–Diagramm an:

Ist in dieser Struktur ein Entwurfsmuster enthalten? Wenn ja, welches?

ja ☐ nein ☐

Begründung:

. . . . . .

Gegeben sei der folgende Java–Code

```
abstract class L {
    abstract void f();
}
class M extends L {
    void f() { }
}
class N extends L {
    L d;
    void f() { d.f(); }
}
class O extends N {
    void f() { d.f(); }
}
class P extends N {
    void f() { d.f(); }
}
```

Geben Sie das zugehörige OMT– oder UML–Diagramm an:

Ist in dieser Struktur ein Entwurfsmuster enthalten? Wenn ja, welches?

ja ☐ nein ☐

Begründung:

. . . . . .

Gegeben seien der folgenden Java–Code

```
class X {
    Object node;
    X [] children;
}
```

Geben Sie das zugehörige OMT– oder UML–Diagramm an:

Ist in dieser Struktur ein Entwurfsmuster enthalten? Wenn ja, welches?

ja ☐ nein ☐

Begründung:

. . . . . .

Dieses ist eine rekursive Struktur. Damit eine rekursive Struktur brauchbar wird, ist immer ein Baisfall nötig, für den die Rekursion abbricht. Ist dieses eine brauchbare Struktur?

ja ☐ nein ☐

Begründung:

. . . . . .

Aufgabe 2:

Entwickeln Sie ein Datenmodell zu Katalogisierung von Papierbildern und Dias. In diesem Katalog werden nicht die Bilder direkt gespeichert, sonder nur Information über die Bilder und wie und wo diese gelagert werden.

Ein Katalog kann einmal eine Menge von Diakästen beschreiben. Diese sind innerhalb eines Katalogs eindeutig durch einen Namen identifizierbar. Diakästen können um eine Menge von beschreibenden Attributen wie Titel, Datum, Ort, Reise, Kapazität des Kastens, ..., ergänzt werden.

Jedes Dia ist innerhalb eines Kastens durch eine Nummer identifiziert. Zu jedem Dia können wieder beliebige beschreibende Attribute existieren, wie Titel, Ort, oder auch technische Daten.

Ein Katalog kann weiter Photoalben enthalten. Diese werden wieder durch einen Namen eindeutig identifiziert. Wie bei Diakästen sind bei Alben auch beschreibende Attribute möglich.

Bilder in einem Album werden eindeutig durch ihre Seitenzahl und eine Position innerhalb einer Seite identifiziert. Sonst besteht zwischen Bildern und Dias kein Unterschied.

Kleine Kataloge, z.B. die Alben in einem Regal oder die in einem Raum gespeicherten Diakästen, sollen zu einem Gesamtkatakog zusammengefasst werden können. Innerhalb eines Kataloges werden die Teilkataloge durch einen Namen identifiziert. Einem Katalog selbst werden ebenfalls beschreibende Attribute zugeordnent. Die Zusammenfassung von Alben, Diakästen und Katalogen zu Gesamtkatalogen kann wiederholt werden.

Die Menge der beschreibenden Attribute für Kataloge, Alben, Kästen, Bilder und Dias soll eine einheitliche Struktur besitzen.

Es wird weiter angenommen, dass folgende einfache Wertebereiche geeignet vordefiniert sind: *AttrName, AttrWert, Nummer, Seite, Position, Id*. Verwenden Sie diese Bereiche als einfache Datentypen. Für die anderen Bereiche verwenden sie, wenn die Lösung es erfordert, unter anderem Namen wie *Katalog, Album, Kasten, Dia, Bild, Attribute*.

Entwickeln Sie ein Datenmodell in abstrakter Syntax in der Notation von Haskell, so wie diese in der Vorlesung genutzt worden ist. Verwenden Sie pro Datentypdefinition nur einen Typkonstruktor (also bitte keine geschachtelten Datentypdefinitionen).

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Entwickeln Sie zu einer gegebenen Anfragesprache für eine Freitextsuche in einem Dokumentenindex eine abstrakte Syntax. Die konkrete Syntax sei durch folgende in BNF–Notation gegebene kontextfreie Grammatik beschrieben. Dabei sind Terminalsymbole in ' gesetzt.

.0	$Query$	$::= Query\ 'OR'\ Query_1 \mid Query_1$
.1	$Query_1$	$::= Query_1\ 'AND'\ Query_2 \mid Query_2$
.2	$Query_2$	$::= Query_3\ 'NEAR'\ Query_2 \mid Query_3$
.3	$Query_3$	$::= Query_4\ Query_3 \mid Query_4$
.4	$Query_4$	$::= '('\ Query\ ')' \mid Query_5$
.5	$Query_5$	$::= Query_6 \mid Query_6\ '...'$
.6	$Query_6$	$::= Word$

In der Anfragesprache gibt es also Und– und Oder–Verknüpfungen, es gibt eine Kontext–Suche, in der ein Wort in der Nähe eines anderen gesucht werden soll (Operator $NEAR$), man kann nach Folgen von Wörtern suchen (1. Regel für $Query_3$), nach einfachen Wörtern und nach Präfixen (Operator ...). Es gibt also vier binäre Operatoren und einen unären Operator in der Sprache. Wie ein Wort dabei aufgebaut sein soll, bleibt hier offen, ein Wort wird durch eine beliebige Zeichenreihe repräsentiert.

Entwickeln Sie für diese Sprache eine abstrakte Syntax für die interne Verarbeitung von Anfragen. Versuchen Sie durch Abstraktion ein möglichst einfaches Modell zu entwickeln.

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....

Welche Strukturmuster findet man in diesem Datenmodell wieder?

1) .....

2) .....

3) .....

Aufgabe 4:

Gegeben sei das folgende Datenmodell in abstrakter Syntax nach Haskell:

.0 type M_1 $= \mathsf{Map}\ K_1\ (M_2, A_3)$

.1 type M_2 $= \mathsf{Map}\ K_2\ (A_2, S_3)$

.2 type S_3 $= \mathsf{Set}\ String$

.3 type K_1 $= Int$

.4 type K_2 $= String$

.5 type A_2 $= Float$

.6 type A_3 $= String$

Transformieren Sie dieses Modell in eines in 1.Normalform, also in eines, das nur noch aus einer Sammlung von Relationstypen besteht, bei denen alle Attribute unstrukturierte Wertebereiche haben. Eine n–stellige Relation ist eine Menge von n-Tupeln. *String* wird in dieser Aufgabe als unstrukturierter Typ betrachtet. In Haskell Syntax wird ein Relationstyp nach folgendem Muster definiert:

$Rel_n\ =\ \mathsf{Set}\ (T_1, \ldots, T_n)$

Beachten Sie, dass in dem Relationenmodell keine Redundanzen entstehen.

1) .....

2) .....

3) .....

4) .....

5) .....
