Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

17. August 2005

Aufgaben zur Klausur **Softwaredesign** im SS 2005 (WI h252, WI h253, II h752, MI h403, MI h404, MI h405)

Zeit: 90 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 17 Seiten

---

Aufgabe 1:

Gegeben sei der folgende Teil eines Datenmodells für einen CD–Katalog:

Das Modell in abstrakter Syntax in Haskell–Notation:

.0 data $Sammlung$ $= Sammlung\ [Album]\ \ Attribute$

.1 data $Album$ $= Album\ [CD]\ \ Attribute$

.2 data CD $= CD\ [Stueck]\ \ Attribute$

.3 type $Stueck$ $= Attribute$

.4 type $Attribute$ $=$ Map $Attr\ Wert$

.5 type $Attr$ $= String$

.6 type $Wert$ $= String$ mbox

Das Modell in abstrakter Syntax nach der alten VDM–Notation:

.0 $Sammlung$ $= Album^* \times Attribute$

.1 $Album$ $= CD^* \times Attribute$

.2 CD $= Stueck^* \times Attribute$

.3 $Stueck$ $= Attribute$

.4 $Attribute$ $=$ map $Attr$ to $Wert$

.5 $Attr$ $= ...$

.6 $Wert$ $= ...$

In diesem Datenmodell ist die Hierarchie streng festgelegt: Es gibt immer genau 4 Stufen in der Hierachie. Dieses ist für die Strukturierung eines Kataloges unflexibel. Zum Beispiel kann man Sammlungen nicht in Teilsamlungen aufgliedern, eine Sammlung kann auch keine CDs und Alben gleichzeitig enthalten.

Entwickeln Sie ein flexibleres Datenmodell, in dem beliebige Hierarchien von Sammlungen, Alben, CDs und Stücken möglich sind, es aber nur diese 4 Arten von Knoten gibt.

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

Welche zusätzlichen Konsistenzbedingungen (Datenstruktur–Invarianten) müssen für diese Datenstruktur eingehalten werden, damit ein CD–Katalog entsteht.

1) .....

2) .....

3) .....

4) .....

5) .....

Aufgabe 2:

Entwickeln Sie ein Datenmodell in Form einer abstraken Syntax für ein hierarchisches Dateisystem. Als Notationen sind die neue Haskell–Notation oder die alte VDM–Notation zugelassen. Geben Sie an, welche Notation Sie verwenden.

Ein Dateisystem in dieser Aufgabe ist ein Verzeichnis, in dem Namen Einträge zugeordnet sind. Einträge können von unterschiedlicher Art sein. Eine Ausprägung sind einfache Dateien. Eine zweite Form ist ein Verzeichnis, eine dritte Form ein symbolischer Verweis (symbolic link), der aus einem absoluten Pfad besteht.

Alle Einträge besitzen zusätzlich eine Menge von Attributen, wie zum Beispiel Zugriffsrechte und Zeitstempel. Diese Menge von Attributen soll nicht fest vorgegeben sein, sondern es soll eine variablen Anzahl von Attributen möglich sein.

Das Datenmodell in Form einer abstrakten Syntax (Haskell oder VDM):

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....

Das Datenmodell in Form eines OMT-Klassendiagramms.

Welche Strukturmuster kommen in diesem Modell vor? Geben Sie jeweils den Musternamen und die beteiligten Datentypen (Klassen) an.

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

Aufgabe 3:

Was versteht man unter einem *double dispatch*?

.....

.....

Welche Entwurfsmuster kann man für die Realisierung eines *double dispatch* nutzen?

1) .....

2) .....

3) .....

4) .....

5) .....

Aufgabe 4:

Worin besteht der softwaretechnische Nutzen bei der Verwendung von Erzeugungsmustern?

Präzise Stichworte (bitte keine Allgemeinplätze):

1) .....

2) .....

3) .....

4) .....

5) .....

Aufgabe 5:

Welche Entwurfsmuster dienen dazu, Methoden als Parameter oder in Daten zu verwenden?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

Aufgabe 6:

Gegeben seien die folgenden Java–Schnittstellen und Klassen

```
interface List {

    public void prepend(Value v);

    public void append(Value v);

    public int length();

    public Value get(int i);

    public boolean isEmpty();
}
```

```
class ListAsValueList implements List {
    private Value l;

    public ListAsValueList() {
        l = Value.nil();
    }

    public void prepend(Value v) {
        l = Value.pair(v,l);
    }

    public void append(Value v) {
        l = l.append(v);
    }

    public int length() {
        return
            l.length();
    }

    public Value get(int i) {
        Value l1 = l;
        while (i != 0) {
            l1 = l1.cdr();
            --i;
        }
        return
```

```
            l1.car();
    }

    public boolean isEmpty() {
        return
            l.isNil();
    }

    public String toString() {
        String res = "";
        int len = length();

        if (len != 0) {

            res = get(0).toString();

            for (int i = 1;
                  i < length();
                  ++i) {
                res += ", " + get(i).toString();
            }
        }
        return
            res;
    }
}
```

```
abstract public class MakeList {
    abstract public
        List newEmptyList();
}
```

```
public class MakeValueList extends MakeList {

    public List newEmptyList() {
        return
            new ListAsValueList();
    }
}
```

```
abstract class Value {

    public boolean isAtom() {
        return
            false;
    }
    public boolean isNil() {
        return
            false;
    }
    public boolean isPair() {
        return
            false;
    }
    public boolean isList() {
        return
            false;
    }
    public boolean isEqual(Value v2) {
        return
            false;
    }
    public Value car() {
        throw
            new RuntimeException("car not supported");
    }
    public Value cdr() {
        throw
            new RuntimeException("cdr not supported");
    }
    public Value append(Value v2) {
        return
            new Pair(v2,this);
    }
    public int length() {
        return
            0;
    }

    public static Value nil() {
        return
            Nil.nil;
    }
    public static Value pair(Value car, Value cdr) {
        return
            new Pair(car, cdr);
    }
```

```
    private static java.util.Dictionary atoms
        = new java.util.Hashtable();

    public static Value atom(String name) {
        Value a = (Atom)(atoms.get(name));

        if (a == null) {
            a = new Atom(name);
            atoms.put(name,a);
        }
        return
            a;
    }
}
```

```
final class Nil extends Value {
    static final Value nil = new Nil();

    private Nil() {}

    public boolean isAtom() {
        return
            true;
    }
    public boolean isNil() {
        return
            true;
    }
    public boolean isList() {
        return
            true;
    }
    public boolean isEqual(Value v2) {
        return
            v2 instanceof Nil;
    }
    public String toString() {
        return
            "nil";
    }
}
```

```
final class Atom extends Value {

    final String name;

    Atom(String name) {
        this.name = name;
    }

    public boolean isAtom() {
        return
            true;
    }

    public boolean isEqual(Value v2) {
        return
            (this == v2)
            ||
            (v2 instanceof Atom
             && name.equals(((Atom)v2).name));
    }

    public String toString() {
        return
            name;
    }
}
```

```
final class Pair extends Value {

    final Value car, cdr;

    Pair(Value car, Value cdr) {
        this.car = car;
        this.cdr = cdr;
    }

    public boolean isPair() {
        return
            true;
    }
    public boolean isList() {
        return
            cdr.isList();
    }
```

```
    public Value car() {
        return
            car;
    }

    public Value cdr() {
        return
            cdr;
    }
    public boolean isEqual(Value v2) {
        return
            (this == v2)
            ||
            (v2 instanceof Pair
             && car.isEqual(v2.car())
             && cdr.isEqual(v2.cdr()));
    }

    public Value append(Value v2) {
        return
            new Pair(car,
                     cdr.append(v2));
    }

    public int length() {
        return
            1 + cdr.length();
    }

    public String toString() {
        return
            "( " + car.toString() + "  .   " + cdr.toString() + " )";
    }

}
```

Welche Strukturmuster sind in dieser Ansammlung von Klassen zu erkennen? Nennen Sie den jeweiligen Musternamen, die beteiligten Klassen und die beteiligten Referenzen.

1. Mustername, beteiligte Klassen und Referenzen

   .....

   .....

   .....

2. Mustername, beteiligte Klassen und Referenzen

   .....

   .....

   .....

3. Mustername, beteiligte Klassen und Referenzen

   .....

   .....

   .....

4. Mustername, beteiligte Klassen und Referenzen

   .....

   .....

   .....

5. Mustername, beteiligte Klassen und Referenzen

   .....

   .....

   .....

Welche Erzeugungsmuster sind in dieser Ansammlung von Klassen zu erkennen? Nennen Sie den jeweiligen Musternamen, die beteiligten Klassen und die beteiligten Methoden und/oder Referenzen.

1. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

2. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

3. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

4. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

5. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

Welche Verhaltensmuster sind in dieser Ansammlung von Klassen zu erkennen? Nennen Sie den jeweiligen Musternamen, die beteiligten Klassen und die beteiligten Methoden und/oder Referenzen.

1. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

2. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

3. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

4. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....

5. Mustername, beteiligte Klassen, Methoden und Referenzen

   .....

   .....

   .....