Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

15. August 2011

Aufgaben zur Klausur **Softwaredesign** im SS 2011 (BInf v310, BMInf v300, BWInf v310, WI h253)

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:

Ein Präfix-Baum ist eine Datenstruktur zur effizienten Implementierung von Maps, bei denen die Schlüssel aus Zeichenreihen bestehen.

Ein Präfixbaum besteht aus einem (Wurzel-)Knoten. Dieser enthält möglicherweise eine Information, einen Attributwert, für einen Schlüssel. Außerdem besteht er aus einer Menge von Nachfolgern. Jeder Nachfolger wird durch ein Zeichen identifiziert und ist wieder ein Präfixbaum.

Das 1. Zeichen eines Schlüssels wird zur Markierung der Kanten auf der 1. Ebene des Baumes verwendet, das 2. Zeichen auf der 2. Ebene usw. Der Schlüssel für einen Eintrag setzt sich also zusammen aus den Zeichen des Pfades von der Wurzel bis zu dem Knoten, der das Attribut enthält.

Tipp: Veranschaulichen Sie sich diese Datenstruktur, indem Sie z.B. für die Menge der Schlüssel-Wert-Paare $\{(a, 5), (anna, 9), (ei, 7), (eto, 9)\}$ so einen Baum skizzieren.

Entwickeln Sie einen Datentyp in Haskell-Notation für einen Typ, der diese Datenstruktur 1-1 (ohne Optimierungen und Datenstrukturverfeinerungen) beschreibt. Die Attributkomponente soll dabei generisch sein.

. . . . . .

. . . . . .

. . . . . .

Gibt es für alle Mengen von Schlüssel-Wert-Paaren einen zugehörigen Präfixbaum?

ja ☐ nein ☐

Begründung:

. . . . . .

. . . . . .

Gibt es für alle Mengen von Schlüssel-Wert-Paaren **genau** einen zugehörigen Präfixbaum?

ja ☐ nein ☐

Begründung:

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben sei die folgende Datenstruktur in Haskell–Notation:

```
.0  data Tree a   = Leaf a
.1                | Chain a (Tree a)
.2                | Fork a (Tree a) (Tree a)
.3                | Triple a (Tree a) (Tree a) (Tree a)
```

Welche Strukturmuster sind in dieser Datenstruktur enthalten?

1) .....

2) .....

3) .....

4) .....

Vereinfachen (verallgemeinern) Sie diese Datenstruktur so, dass nicht mehr mit den vier verschiedenen Varianten (in OOP/OOD: Nicht mehr mit Vererbung) gearbeitet werden muss und dass mehr als 3 Nachfolger möglich werden. Bitte benutzen Sie pro Typdefinition nur einen Typkonstruktor.

Das vereinfachte Datenmodell:

.....

.....

.....

.....

Aufgabe 3:

Gegeben seien die folgenden Java Schnittstellen und Klassen:

```
public interface SimpleImage { }

public abstract class Image {
        public static final double black = 0.0;
        public static final double white = 1.0;
        public static final double grey = 0.5;

        public final int w;
        public final int h;

        protected Image(int w, int h){
                this.w = w; this.h = h;
        }
        protected abstract double at(int i, int j);

        public double pixelAt(int i,int j){
                return
                Math.max(Math.min(at(i,j),1.0),0.0);
        }
}

public class Uni extends Image implements SimpleImage {
        private final double color;

        public Uni(int w, int h, double color){
                super(w,h);
                this.color = color;
        }
        protected double at(int i, int j) {
                return color;
        }
}

public class GreyMap extends Image implements SimpleImage {

        private final double [][] pixels;

        public GreyMap(double [][] pixels){
                super(pixels.length,pixels[0].length);
                this.pixels = pixels;
        }
        protected double at(int i, int j) {
                return pixels[i][j];
```

```
        }
}

public class ImageCache extends Image implements SimpleImage {
        protected final Image img;

        public ImageCache(Image img){
                super(img.w,img.h);
                this.img = new GreyMap(eval(img));
        }

        private
        double [][] eval(Image img) {
                double [][] a = new double[img.w][img.h];
                for (int i=0; i<img.w;++i)
                        for(int j=0; j<img.h;++j)
                                a[i][j] = img.pixelAt(i,j);
                return a;
        }

        protected double at(int i, int j) {
                return img.at(i,j);
        }
}

public class Superimpose extends Image {
        protected final Image top,bottom;

        public Superimpose(Image top, Image bottom){
                super(Math.max(top.w,bottom.w),Math.max(top.h,bottom.h));
                this.top = top;
                this.bottom = bottom;
        }

        protected double at(int i, int j) {
                if (i<top.w && j<top.h)
                        return top.at(i,j);
                if (i<bottom.w && j<bottom.h)
                        return bottom.at(i,j);
                return black;
        }
}

public class SideBySide extends Image {
        protected final Image left, right;

        public SideBySide(Image left, Image right){
```

```
                super(left.w + right.w,Math.max(left.h,right.h));
                this.left = left;
                this.right = right;
        }

        protected double at(int i, int j) {
                if (i < left.w && j < left.h)
                        return left.at(i,j);
                if (i − left.w < right.w && j < right.h)
                        return right.at(i−left.w,j);
                return black;
        }
}

abstract class HalfImage extends Image {
        protected final Image img;

        protected HalfImage(int w, int h, Image img) {
                super(w,h);
                this.img = img;
        }
}

public class HalfWidth extends HalfImage {

        public HalfWidth(Image img){
                super((img.w +1)/2,img.h,img);
        }
        protected double at(int i, int j) {
                return
                (img.at(2∗i,j) + img.at(2∗i+1,j))/2.0;
        }
}

public class HalfHeight extends HalfImage {

        public HalfHeight(Image img){
                super(img.w,(img.h + 1)/2,img);
        }
        protected double at(int i, int j) {
                return
                (img.at(i,2∗j) + img.at(i,Math.min(2∗j+1,img.h)))/2.0;
        }
}

public class HalfSize extends HalfImage {
```

```
        public HalfSize(Image img){
                super((img.w + 1)/2,(img.h + 1)/2,
                                new HalfWidth(new HalfHeight(img)));
        }
        protected double at(int i, int j) {
                return
                img.at(i,j);
        }
}

public class ImageGenerator {
        public Image uni(int w, int h, double c) {
                return new Uni(w,h,c);
        }
        public Image imageMap(Image img) {
                return new ImageCache(img);
        }
        public Image halfSizeImage(Image img) {
                return new HalfSize(img);
        }
}

public class SmartImageGenerator extends ImageGenerator {
        public Image imageMap(Image img) {
                if (img instanceof SimpleImage)
                        return img;
                return super.imageMap(img);
        }
        public Image halfSizeImage(Image img) {
                return new HalfWidth(new HalfHeight(img));
        }
}
```

Welche Strukturmuster kommen in diesem Modell vor?
Geben Sie jeweils den Musternamen und die beteiligten Klassen, Schnittstellen und Referenzen an.

1) .....

2) .....

3) .....

4) .....

5) .....

Welche Erzeugungsmuster kommen in diesem Modell vor?
Geben Sie jeweils den Musternamen und die beteiligten Klassen, Schnittstellen, Referenzen und Methoden an.

1) .....

2) .....

3) .....

Welche Verhaltensmuster kommen in diesem Modell vor?
Geben Sie jeweils den Musternamen und die beteiligten Klassen, Schnittstellen, Referenzen und Methoden an.

1) .....

2) .....

3) .....

4) .....

5) .....

Aufgabe 4:

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

Das Modell in abstrakter Syntax in Haskell–Notation:

.0 data $Sammlung \quad = Sammlung\,[\,Album\,]\;\; Attribute$

.1 data $Album \quad = Album\,[\,CD\,]\;\; Attribute$

.2 data $CD \quad = CD\,[\,Stueck\,]\;\; Attribute$

.3 data $Stueck \quad = Stueck\; Attribute$

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

.5 type $Attr \quad = String$

.6 type $Wert \quad = String$

In diesem Datenmodell ist die Hierarchie streng festgelegt: Es gibt immer genau vier 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 vier 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 sinnvoller CD–Katalog entsteht.

1) .....

2) .....

3) .....

4) .....

5) .....