

Institut für Informatik  
Prof. Dr. Peter Thiemann  
Jochen Walter

Georges-Köhler-Allee 79  
D-79110 Freiburg i. Br.

Freiburg, den 9. Februar 2001

## Informatik 1, WiSe 2000/2001

### Übungsblatt 12

Die Aufgaben werden in den Übungs- und Programmiergruppen  
vom 5.2. bis zum 9.2. besprochen.

Die Lösungen müssen nicht abgegeben werden!

*Die Aufgaben auf diesem Blatt können in Teams bearbeitet werden. Java-Quelldateien können mit jedem beliebigen Texteditor wie pico oder xemacs eingegeben werden. Auch der in DrScheme eingebaute Editor läßt sich benutzen.*

*Um den Java-Compiler verwenden zu können, muß man setup java1.2 eingeben. Ein Java-Programm x.java läßt sich dann durch Eingabe von javac x.java mit dem Java-Compiler übersetzen. Die entstehende Bytecodedatei x.class kann danach mit dem Javainterpreter ausgeführt werden, indem man java x eingibt. Die Java API-Dokumentation findet sich auf den Web-Seiten zur Vorlesung.*

*Die Aufgaben sind mit einem bis drei Sternen versehen, wobei die Aufgaben mit einem Stern am einfachsten, die mit dreien am schwierigsten sind.*

#### Aufgabe 1 (\*\*\*):

In der Vorlesung haben Sie in Abschnitt B3.5.1 die Behauptung kennengelernt, daß jeder Listenterm  $\equiv_Q$ -äquivalent zu einem Konstruktorterm ist

Beweisen Sie diese Aussage für Terme der Form  $\text{app}(l_1, l_2)$ .

#### Lösung zu Aufgabe 1:

**Induktionsbasis:**  $\text{empty} \equiv_Q \text{empty}$  und  $\text{empty}$  ist ein Konstruktorterm.

**Induktionsschritt:** Die Induktionsannahme lautet  $l_1 \equiv_Q l'_1$  und  $l_2 \equiv_Q l'_2$ , wobei  $l'_1$  und  $l'_2$  Konstruktorterm sind. Man muß dann zeigen, daß auch  $\text{app}(l_1, l_2) \equiv_Q$ -äquivalent zu einem Konstruktorterm ist.

Zunächst gilt  $\text{app}(l_1, l_2) \equiv_Q \text{app}(l'_1, l'_2)$ . (Das liegt an der Definition von  $\equiv_Q$ .) Man muß dann beweisen, daß  $\text{app}(l'_1, l'_2) \equiv_Q$ -äquivalent zu einem Konstruktorterm ist. Das geschieht durch vollständige Induktion über  $l'_1$ :

**Induktionsbasis:** Falls  $l'_1 = \text{empty}$ , so gilt

$$\text{app}(\text{empty}, l'_2) \equiv_Q l'_2$$

wegen des Axioms  $\text{app}(\text{empty}, v) = v$ .

**Induktionsschritt:** Falls  $l'_1 = \text{cons}(a, l''_1)$  (mit  $l''_1 \in T_\Gamma$ ) ist, so gilt

$$\begin{aligned} \text{app}(l'_1, l'_2) &\equiv_Q \text{app}(\text{cons}(a, l''_1), l'_2) \\ &\equiv_Q \text{cons}(a, \text{app}(l''_1, l'_2)) \\ &\equiv_Q \text{cons}(a, l') \quad \text{für ein } l' \in T_\Gamma \end{aligned}$$

Man hat es hier also mit zwei Induktionen zu tun. Der Induktionsschritt der ersten Induktion lautet „Wenn zwei Terme  $l_1$  und  $l_2$  äquivalent zu einem Konstruktorterm sind, dann ist es auch  $\text{app}(l_1, l_2)$ “.

Der Induktionsschritt der Hilfsinduktion lautet „Wenn für zwei Konstruktorterm  $l''_1$  und  $l'_2$  der Term  $\text{app}(l''_1, l'_2)$  äquivalent zu einem Konstruktorterm ist, dann ist es auch (für alle  $a$ )  $\text{app}(\text{cons}(a, l''_1), l'_2)$ “.

### Aufgabe 2 (\*\*\*):

Geben Sie drei sinnlose Terme aus der Termmenge des ADT  $\text{list}(A)$  an, und zeigen Sie, daß sie in der Äquivalenzklasse  $[\perp]_{\equiv_Q}$  liegen.

### Lösung zu Aufgabe 2:

Drei sinnlose Terme sind z.B.  $\text{head}(\text{empty})$ ,  $\text{tail}(\text{empty})$  und  $\text{app}(\text{empty}, \text{tail}(\text{empty}))$ .

- (a) Zunächst gilt  $\text{head}(\text{empty}) \equiv_Q \perp$ , da  $\text{head}(\text{empty}) = \perp$  zu der Axiomenmenge  $Q$  gehört.
- (b) Weiter ist  $\text{tail}(\text{empty}) \equiv_Q \perp$ , da  $\text{tail}(\text{empty}) = \perp$  ebenfalls zu der Axiomenmenge  $Q$  gehört.
- (c) Schließlich gilt

$$\begin{aligned} \text{app}(\text{empty}, \text{tail}(\text{empty})) &\equiv_Q \text{app}(\text{empty}, \perp) && \text{(Siehe (a))} \\ &\equiv_Q \perp && \text{wegen } F(\dots, \perp, \dots) = \perp \end{aligned}$$

### Aufgabe 3 (\*\*\*):

Der ADT  $\text{Map}(A)$  soll Abbildungen von  $\text{Nat} \rightarrow A$  repräsentieren. Er unterscheidet sich von  $\text{Array}(A)$  dadurch, daß er statt  $\text{newArray}$  den Konstruktor

$$\text{newMap} : \rightarrow \text{Map}(A)$$

besitzt. Das Axiom R.1 wird ersetzt durch

$$\text{get}(\text{newMap}(), x) = \perp.$$

Das Axiom R.6 wird gestrichen.

Wir betrachten nun den Spezialfall  $\text{Map}(\{a, b\})$ , d.h. der Träger von  $A$  ist  $\{a, b\}$ . Geben Sie eine Menge  $M$  an, die für jede Äquivalenzklasse von  $T_{\Gamma}$  genau einen Repräsentanten enthält.

### Lösung zu Aufgabe 3:

- $\text{Map}() \in M$
- Für alle endlichen  $N = \{n_0, n_1, \dots, n_{|N|-1}\} \subset \mathbb{N}$  mit  $n_0 < n_1 < \dots < n_{|N|-1}$  und  $(e_0, e_1, \dots, e_{|N|-1}) \in \{a, b\}^{|N|}$  ist

$$\text{update}(\dots(\text{update}(\text{update}(\text{newMap}(), n_0, e_0), n_1, e_1) \dots), n_{|N|-1}, e_{|N|-1}) \in M.$$

- $M$  ist die kleinste Menge mit diesen Eigenschaften.

### Aufgabe 4 (Programmierung, \*\*\*):

Schreiben Sie ein Programm, das die größtmögliche Ellipse darstellt (`drawOval`), die in den sichtbaren Bereich des Fensters paßt. Das soll auch dann gelten, wenn die Größe des Fensters verändert wird. Um in diesem Fall beim Neuzeichnen der Ellipse die alte Ellipse zu löschen, muß zuvor ein Aufruf von `clearRect` erfolgen.

Die Breite bzw. Höhe des Fensters können Sie durch die Methoden `getWidth()` bzw. `getHeight()` erhalten. Da sich an den Rändern des Fensters ein Rahmen befindet, ist dies allerdings nicht der sichtbare Bereich. Der Rahmen hat links (oben, rechts, unten) eine Stärke von `getInsets().left` (`getInsets().top`, `getInsets().right`, `getInsets().bottom`) Bildpunkten. Daraus läßt sich der sichtbare Fensterbereich berechnen.

(Wenn man eine schöne Darstellung erhalten möchte, muß man beim Aufruf von `drawOval` die Werte für Breite und Höhe um eins verringern. Der Grund dafür steht in der Dokumentation zur Klasse `java.awt.Graphics`).

### Lösung zu Aufgabe 4:

```

// Programm      : Oval
// Beschreibung  : Oeffnet ein Fenster und zeichnet darin die
//                groesstmoeegliche Ellipse, die in den sichtbaren Bereich
//                passt.
// Programmierer : Jochen Walter
// Datei erzeugt am: Mon Nov 08 11:09:21 GMT+00:00 1999

```

```

import java.awt.*;
import javax.swing.*;

```

```

/** MyFrame erzeugt ein Fenster. */
class MyFrame extends JFrame

```

```

{
    /** Die Konstruktormethode MyFrame erzeugt ein Fenster und macht es
        auf dem Bildschirm sichtbar. */
    public MyFrame()
    {
        setTitle("Ellipse - Aufgabe 4");
        setBackground(Color.white);
        setSize(300,300);
        setVisible(true);
    }
}

```

```

/** paint zeichnet die groesstmoeegliche Ellipse, die in den
 * sichtbaren Bereich passt. */
public void paint(Graphics g)
{
    int linker_rand = 0 + getInsets().left,
        oberer_rand = 0 + getInsets().top,
        sichtbare_breite = getWidth()-getInsets().left-getInsets().right,
        sichtbare_hoehe = getHeight()-getInsets().top-getInsets().bottom;

    g.clearRect(0, 0, getWidth(), getHeight());
    g.drawOval(linker_rand, oberer_rand,
                sichtbare_breite-1, sichtbare_hoehe-1);
}
}

```

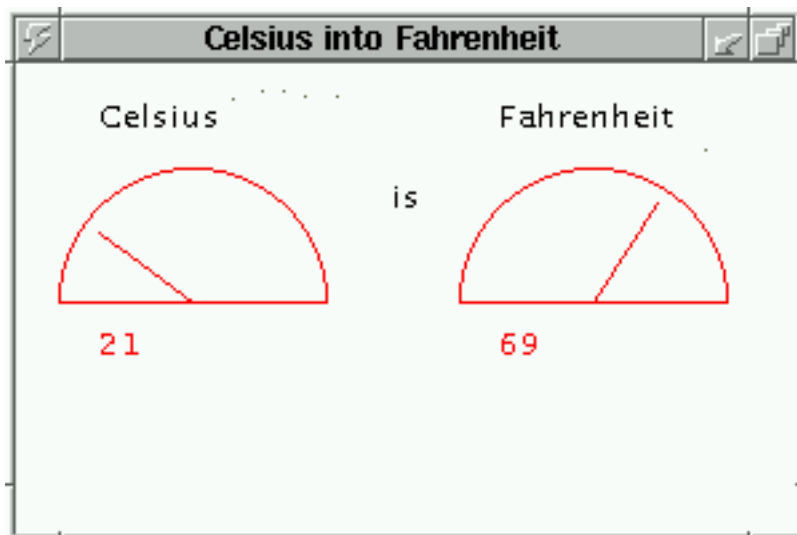
```

public class Oval
{
    public static void main(String[] args)
    {
        MyFrame f = new MyFrame();
        System.out.println("Das Fenster wurde dargestellt.");
    }
}

```

### Aufgabe 5 (\*\*\*):

In der Vorlesung wurden zwei Klassen vorgestellt, die Temperaturen auf verschiedene Weise anzeigen. Schreiben Sie eine Klasse `TemperaturesWriter2`, die die Temperatur wie ein Bimetallthermometer anzeigt sowie eine Klasse `CelsiusToFahrenheit5` mit einem Hauptprogramm. Das von der Klasse dargestellte Fenster sollte etwa so aussehen:



Bei Null Grad soll der Zeiger nach links zeigen. bei 100 Grad nach rechts. Falls der Mittelpunkt des Kreisbogens die Koordinaten `x_center` und `y_center` hat, lauten die Ausdrücke zur Berechnung der Koordinaten der Zeigerspitze

```
x_tip = x_center - (int) (radius * Math.cos(degrees * Math.PI/100.0));
y_tip = y_center - (int) (radius * Math.sin(degrees * Math.PI/100.0));
```

Den Quellcode für die bereits vorhandenen `Writer`-Klassen können sie auf der Web-Seite zur Vorlesung finden.

Die vorgegebene `TemperaturesWriter`-Klasse hat allerdings Designfehler:

- Die Temperatur wird auch dann angezeigt, wenn die Methoden `displayCelsius` und `displayFahrenheit` noch nicht aufgerufen wurden und die entsprechenden Datenfelder keinen sinnvollen Wert enthalten.
- Die graphische Thermometerdarstellung kann nur den Temperaturbereich von 0 bis 100 Grad richtig zeichnen. Wenn man eine Temperatur außerhalb dieses Bereichs angibt, erfolgt eine fehlerhafte Ausgabe. (Überzeugen Sie sich davon.)
- Die `paint`-Methode löscht das Fenster nicht, bevor sie einen neuen Temperaturwert ausgibt. Das führt vor allem bei der Klasse `TemperaturesWriter` zu einer unschönen Darstellung. (Überzeugen Sie sich davon, indem sie das entsprechende Fenster vergrößern.)

Machen Sie es besser:

- Definieren Sie in Ihrer Klasse zwei Datenfelder vom Typ `boolean`, die auf `true` gesetzt werden, wenn `celsius_degrees` bzw. `fahrenheit_degrees` einen sinnvollen Wert enthalten. Falls das nicht der Fall ist, soll in dem Fenster statt eines Thermometers eine Fehlermeldung ausgegeben werden.
- Überprüfen Sie vor der Darstellung des Thermometers, ob die Temperatur im zulässigen Bereich liegt. Wenn das nicht der Fall ist, soll in dem Fenster statt eines Thermometers eine Fehlermeldung ausgegeben werden.
- Rufen Sie in Ihrer `paint`-Methode `clearRect` mit geeigneten Parametern auf, bevor Sie irgendetwas in das Fenster zeichnen.

### Lösung zu Aufgabe 5:

```
/** CelsiusToFahrenheit converts an input Celsius value to Fahrenheit.
 *  output: the degrees Fahrenheit, a double */
public class CelsiusToFahrenheit5
{
    public static void main (String[] args)
    {
```

```

        TemperaturesWriter2 writer = new TemperaturesWriter2 ();
        DialogReader keyboard = new DialogReader ();
        int c = keyboard.readInt ("Degrees Celsius: ");
        writer.displayCelsius (c);
        double f = celsiusIntoFahrenheit (c);
        writer.displayFahrenheit ((int) f);
    }

    private static double celsiusIntoFahrenheit (double c)
    {
        return ((9.0/5.0) * c) + 32;
    }
}

import java.awt.*;
import javax.swing.*;

/** TemperaturesWriter2 creates a graphics window for displaying temperatures */
public class TemperaturesWriter2 extends JFrame
{ // these fields define the sizes and positions of the two thermometers:
    private int CELSIUS_POS = 20;    // x-position of the Celsius thermometer
    private int FAHRENHEIT_POS = 170; // x-position of the Fahrenheit thermometer
    private int TOP = 60;            // the y-position of the thermometers' tops
    private int LENGTH = 50;         // the length of the thermometer
    private int WIDTH = 100;         // the width of the thermometer
    private int MIN_TEMP = 0;        // the min. temp.
    private int MAX_TEMP = 100;      // the max. temp.
    private int TEXT_OFFSET = 15;    // where to position the text labels

    // these fields remember the two temperatures that are displayed:
    private int celsius_degrees;      // the degrees celsius
    private boolean celsius_degrees_contains_data = false;

    private int fahrenheit_degrees;   // the degrees fahrenheit
    private boolean fahrenheit_degrees_contains_data = false;

    /** Constructor TemperaturesWriter2 creates the window and makes it visible */
    public TemperaturesWriter2()
    { setTitle("Celsius into Fahrenheit");
      setSize(300,200);
      setBackground(Color.white);
      setVisible(true);
    }

    /** displayCelsius draws the Celsius thermometer
     * @param degrees - temperature to display */
    public void displayCelsius(int degrees)
    {
        celsius_degrees = degrees;
        celsius_degrees_contains_data = true;
        repaint(); // repaint the window with the new celsius_degrees
    }

    /** displayFahrenheit draws the Fahrenheit thermometer
     * @param degrees - temperature to display */

```

```

public void displayFahrenheit(int degrees)
{
    fahrenheit_degrees = degrees;
    fahrenheit_degrees_contains_data = true;
    repaint(); // repaint the window with the new fahrenheit_degrees
}

/** paint fills the window with two thermometers
 * @param g - the 'graphics pen' that draws the items onto the window */
public void paint(Graphics g)
{
    g.clearRect(0, 0, getWidth(), getHeight());

    g.setColor(Color.black);
    g.drawString("Celsius", CELSIUS_POS + TEXT_OFFSET, TOP - TEXT_OFFSET);
    g.drawString("is", (CELSIUS_POS + WIDTH + FAHRENHEIT_POS)/2, TOP + TEXT_OFFSET);
    g.drawString("Fahrenheit", FAHRENHEIT_POS + TEXT_OFFSET, TOP - TEXT_OFFSET);
    drawThermometer(CELSIUS_POS, celsius_degrees,
                    celsius_degrees_contains_data, g);

    drawThermometer(FAHRENHEIT_POS, fahrenheit_degrees,
                    fahrenheit_degrees_contains_data, g);
}

/** drawThermometer paints a thermometer
 * @param where - the horizontal position of the thermometer
 * @param degrees - how high to fill the thermometer
 * @param data_available - contains degrees a reasonable value?
 * @param g - the graphics pen */
private void drawThermometer(int where, int degrees,
                             boolean data_available, Graphics g)
{
    if (data_available)
    {
        if (MIN_TEMP <= degrees && degrees <= MAX_TEMP)
        {
            g.setColor(Color.red);
            g.drawArc(where, TOP, WIDTH, WIDTH, 0, 180);
            g.drawLine(where, TOP+LENGTH, where+WIDTH, TOP+LENGTH);

            paintTemp(where, degrees, g); // show temperature
        }
        else
        {
            g.setColor(Color.red);
            g.drawString("Out of range", where, TOP+LENGTH);
        }
        g.drawString(degrees + "", where + TEXT_OFFSET, TOP+LENGTH+20);
    }
    else
    {
        g.drawString("No data", where, TOP+LENGTH);
        System.out.println("No data" );
    }
}

```

```

/** paintTemp fills a thermometer with the appropriate temperature
 * @param where - the location of the thermometer
 * @param degrees - how high to fill it
 * @param g - the graphics pen */
private void paintTemp(int where, int degrees, Graphics g)
{
    int x_center = where + WIDTH/2,    // Mittelpunkt des Kreisbogens
        y_center = TOP + LENGTH;

    int radius = WIDTH/2 - 5;          // Radius des Kreisbogens

    int x_tip = x_center                // Spitze des Zeigers
        - (int) (radius * Math.cos(degrees * Math.PI/MAX_TEMP)),
        y_tip = y_center
        - (int) (radius * Math.sin(degrees * Math.PI/MAX_TEMP));

    g.drawLine(x_center, y_center, x_tip, y_tip);
}
}

```