Technische Universität Darmstadt
Fachbereich Informatik
Prof. Johannes Fürnkranz

Allgemeine Informatik II im SS 2007

Übungsblatt 9

Bearbeitungszeit: 27.06. bis 03.07.2006

Aufgabe 1: Expresszüge

In der letzten Übung haben Sie Eigenschaften eines Zugs (**`Train`**) mit Lok und Wagen als Java-Klasse implementiert. Expresszüge (beispielsweise der ICE) haben meistens zwei Loks – am Anfang und am Ende des Zuges.

Die **`Train`**-Klasse erlaubt aber nur, Züge mit einer Lok zu konstruieren. Um diese Einschränkung aufzuheben und Expresszüge mit zwei Loks zu bilden können, empfiehlt es sich eine neue Klasse einzuführen, die die Klasse **`Train`** erweitert.

Schreiben Sie eine Klasse **`ExpressTrain`**, die von **`Train`** erbt (**`extends`**). Gehen Sie dabei folgendermaßen vor:

a) Legen Sie die Klasse im Package **train** ab.

b) Stellen Sie sicher, dass die Klasse eine zweite Lok speichern kann.

c) Definieren Sie zwei Konstruktoren:
   1. einen, der einen Parameter erwartet und einen **`ExpressTrain`** mit zwei gleichen Loks konstruiert;
   2. einen, der zwei Parameter erwartet und einen **`ExpressTrain`** mit zwei verschiedenen Loks konstruiert.

   Verwenden Sie bei beiden Konstruktoren sinnvoll den Konstruktor der Klasse **`Train`**.

d) Überschreiben Sie die Methode **`getLength()`**, so dass die korrekte Zuglänge mit beiden Loks zurückgegeben wird. Nutzen Sie auch die **`getLength()`**-Methode der Superklasse **`Train`** (**`super.getLength()`**).

e) Überschreiben Sie die Methode **`toString()`**. Die zweite Lok soll am Ende des Strings erscheinen.

Ist es auch erforderlich, die anderen Methoden der **`Train`**-Klasse zu überschreiben? Überlegen Sie, warum oder warum nicht und diskutieren Sie darüber im Forum.

Aufgabe 2: Operationsverstärker

In dieser Aufgabe beschäftigen Sie sich mit einem Baustein aus der Elektrotechnik: dem Operationsverstärker. Sie können diese Aufgabe aber selbstverständlich auch lösen, wenn Sie nicht Elektrotechnik studieren.

Schreiben Sie eine Klasse **`IdealOpAmp`**, die einen Operationsverstärker mit den Attributen **`float inputResistor`** und **`float outputResistor`** für Eingangs- und Ausgangswiderstand enthält. Kennzeichnen Sie die Attribute mit dem Schlüsselwort **`protected`** (statt **`public`** oder **`private`**), damit auch spätere Erben der Klasse darauf zugreifen können. Außerdem soll Ihre Klasse folgende Konstruktoren und Methoden haben:

- einen Konstruktor, der keine Parameter erwartet und die Werte der Attribute auf **`1.0f`** setzt (Bei **`float`**-Werten wird ein kleines **`f`** an die Zahl gehängt);

- einen Konstruktor, dem Eingangs- und Ausgangswiderstand übergeben werden.

- eine Methode **`public String toString()`**, die den Namen der Klasse und die Werte der Attribute als String zurückgibt.

- eine Methode **`public float computeOutput(float input)`**, die abhängig von einer Eingangsspannung (**`input`**) und den beiden Widerständen eine Ausgangsspannung berechnet und als **`float`** zurückgibt. Ausgehend von einem Inverter-Schaltbild lautet die Formel für die Ausgangsspannung $\mathbf{U_o}$ wie folgt: $\mathbf{U_o = -(R_i / R_o) * U_i}$ (mit Eingangsspannung $\mathbf{U_i}$, Eingangswiderstand $\mathbf{R_i}$ und Ausgangswiderstand $\mathbf{R_o}$).

Schreiben Sie nun eine Klasse **`AdderOpAmp`**, die von **`IdealOpAmp`** erbt. Die Klasse **`AdderOpAmp`** soll zusätzlich zu den geerbten Eigenschaften einen weiteren Eingangswiderstand float inputResistor2 erhalten. Überschreiben bzw. ergänzen Sie folgende Methoden:

- die Konstruktoren, damit sie den zusätzlichen Widerstand unterstützen.

- **`toString()`**, um die Ausgabe an die neue Klasse anzupassen (z. B. der zusätzliche Widerstand)

- **`computeOutput(…)`**, die nun zwei Eingangsspannungen übergeben bekommt und die Ausgangsspannung berechnet. Formel: $\mathbf{U_o = -((R_{i1} / R_o) * U_{i1} + (R_{i2} / R_o) * U_{i2})}$.

- **`boolean adderMode()`**, die den Wert **`true`** zurückgibt, wenn alle drei Widerstände denselben Wert haben.

Schreiben Sie zum Testen eine Klasse **`OpAmpTest`**, wie üblich bestückt mit einer **`main`**-Methode. In dieser Methode soll folgendes geschehen:

a) Initialisieren Sie ein **`IdealOpAmp`**-Array mit **`IdealOpAmp`**- und **`AdderOpAmp`**-Objekten.

b) Gehen Sie nun das Array durch und geben Sie mit der **`toString()`**-Methode die Operationsverstärker-Informationen aus. In Abhängigkeit davon, ob ein Array-Element den **aktualen bzw. dynamischen Typ `AdderOpAmp`** hat (der **formale bzw. statische Typ** jedes Array-Elements ist **`IdealOpAmp`**), soll an das Ende des Textes der **`toString()`**-Methode noch angehängt werden, welchen Status die Methode **`adderMode()`** zurückgibt. Um diese Methode ansprechen zu können, müssen Sie einen **TypeCast** nach **`AdderOpAmp`** vornehmen.

c) Führen Sie einige beliebige Berechnungen mit **`computeOutput(…)`** durch und geben diese auf dem Bildschirm aus. Auch hier müssen Sie gegebenenfalls einen **TypeCast** vornehmen.

Die Ausgabe der Programmausführung könnte z.B. wie folgt aussehen:

```
IdealOpAmp - RI: 582.22, RO: 34.12
AdderOpAmp - RI1: 111.123 , RI2: 232.54, RO: 321.33, AdderMode: false
IdealOpAmp - RI: 1.0, RO: 1.0
AdderOpAmp - RI1: 1.0 , RI2: 1.0 , RO: 1.0 , AdderMode: true
Compute UO: -53.0
```

Viel Spaß!