Technische Universität Darmstadt

Fachbereich Informatik

Prof. Dr. Andreas Koch

Allgemeine Informatik 2 im SS 2008

Übungsblatt 9

Bearbeitungszeit: 09.06. bis 15.06.2008

Aufgabe 1: Wechselseitige Rekursion

Schreiben Sie eine Klasse **`EvenOdd`**, die zwei statische **`boolean`**-Methoden **`even(int num)`** und **`odd(int num)`** enthält.

Diese sollen **zusammenarbeiten**, um für beliebige Ganzzahlen ≥ 0 festzustellen, ob diese gerade bzw. ungerade sind. Wenn der Parameter **`num`** gleich 0 ist, ist die Frage für beide Methoden leicht zu beantworten. Anderenfalls ist eine Zahl genau dann gerade, wenn die nächstkleinere Zahl ungerade ist – bzw. genau dann ungerade, wenn die nächstkleinere Zahl gerade ist.

Aufgabe 2: Vererbung und Polymorphie

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 **`double inputResistor`** und **`double 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 **`1.0`** setzt.
- 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 double computeOutput(double input)`**, die abhängig von einer Eingangsspannung (**`input`**) und den beiden Widerständen eine Ausgangsspannung berechnet und als **`double`** zurückgibt. Ausgehend von einem Inverter-Schaltbild lautet die Formel für die Ausgangsspannung U_o wie folgt: $U_o = -(R_i / R_o) * U_i$ (mit Eingangsspannung U_i, Eingangswiderstand R_i und Ausgangswiderstand R_o).

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

- die Konstruktoren, damit sie den zusätzlichen Widerstand unterstützen.
- **`toString()`**, um den zurückgegebenen String an die neue Klasse anzupassen (also auch den zusätzlichen Widerstand in den String aufzunehmen)

- **`computeOutput(…)`**, die nun zwei Eingangsspannungen übergeben bekommt und die Ausgangsspannung berechnet. Formel: $U_o = -((R_{i1} / R_o) * U_{i1} + (R_{i2} / R_o) * U_{i2})$.
- **`boolean adderMode()`**, die **`true`** zurückgibt, wenn alle drei Widerstände denselben Wert haben.

Schreiben Sie zum Testen eine Klasse **`OpAmpTest`** mit einer statischen **`void`**-Methode **`test()`**, in der Sie folgendes tun:

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/dynamischen Typ `AdderOpAmp`** hat (der **formale/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
```

Aufgabe 3: FREIWILLIGE FLEISSAUFGABE: Strings durchsuchen

In Übung 8 haben Sie die Methode **`indexOf(…)`** der Klasse **`String`** kennen gelernt, um in einem String einen Buchstaben zu suchen. Es gibt in der Klasse **`String`** zwar auch schon eine Methode **`indexOf(…)`**, um in einem String einen anderen String zu suchen, aber das wäre ja langweilig…

Schreiben Sie eine Klasse **`String2`**, die eine private Objektvariable String **`s`** besitzt. Es soll auch einen Konstruktor geben, der einen **`String`** übergeben bekommt und **`s`** damit initialisiert.

Schreiben Sie dann eine Methode **`public int indexOf(String s2)`**, die im String **`s`** nach dem String **`s2`** sucht. Wenn **`s2`** gefunden wird, soll die Methode den Index des ersten Zeichens des ersten Vorkommens in **`s`** zurückgeben, sonst **`-1`**. Eine Suche in **”teststring“** nach **”st“** soll also **2** zurückgeben (nicht **4**!), eine Suche nach **”test“** oder **”teststring“ 0** und eine Suche nach **”atlantis“** oder **”teststring2“ -1**.

Sie werden zwei verschachtelte Schleifen mit entsprechenden Abbruchbedingungen benötigen, um ab jeder möglichen Position(*) in **`s`** Zeichen für Zeichen nach **`s2`** zu suchen. Außerdem temporäre Variablen, damit Sie verwalten können, ob und wo Sie **`s2`** gefunden haben.

(*) Die möglichen Positionen bei einer Suche in **”geheim“** nach einem **`s2`** mit drei Buchstaben:

geheim / geheim / geheim / geheim – Achten Sie darauf, nicht über **`s`** hinaus zu suchen!

Viel Spaß!