technische universität dortmund

fakultät für informatik

Amer Krivošija
Denis Kurz
Eugen Rusakov

Dortmund, den 19. Januar 2017

Übung zur Vorlesung EidP (WS 2016/17)

Blatt 10

Block grün

Es können 4 Punkte erreicht werden.

Abgabedatum: 26. Januar 2017, 23:59 Uhr

Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep1617uebung/`

- In der optionalen Datei `Anmerkungen.txt` können Sie allgemeine Anmerkungen bezüglich Ihrer Lösungen notieren.
- Stellen Sie sicher, dass alle von Ihnen abgegebenen Dateien reine Textdateien im UTF-8-Format sind.

Aufgaben

Aufgabe 0: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Text-Datei `Aufgabe_10_0.txt` an.

a) Was gilt, wenn `A` eine Oberklasse von `B` ist? (0.3 Punkte)

b) Was versteht man unter Polymorphie? (0.1 Punkte)

c) Wie werden virtuelle Methoden gekennzeichnet und wann werden sie gebunden? (0.2 Punkte)

d) Wie werden rein virtuelle Methoden definiert? (0.1 Punkte)

e) Wann nennt man eine Klasse abstrakt? (0.1 Punkte)

f) Erklären Sie den konzeptionellen Ablauf bei der Ausnahmebehandlung mit Exceptions. (0.2 Punkte)

Aufgabe 1: Virtuelle Methoden (3 Punkte)

In dieser Aufgabe wollen wir eine Klassenhierarchie für mathematische Funktionen entwickeln. Die Basisklasse `Function` soll über die rein virtuelle Methode `double evaluate(double x)` und keine Attribute verfügen. Von der Klasse `Function` sollen drei Unterklassen abgeleitet werden, die lineare Funktionen (`LinearFunction`), quadratische Funktionen (`QuadraticFunction`) und exponentiell wachsende Funktionen (`ExponentialFunction`) modellieren. Legen Sie dazu die Datei `Aufgabe_10_1.cpp` an.

Hinweis: Durch `#include <cmath>` können Sie die Funktionen `exp`, `sqrt` und `acos` verwenden. Es gilt $acos(-1) = \pi$.

a) Legen Sie die Klassenhierarchie entsprechend der Vorgaben an. (0.4 Punkte)

b) Implementieren Sie in den abgeleiteten Klassen die Methode `evaluate`, sodass

- für `LinearFunction` der Wert $a \cdot x + b$,
- für `QuadraticFunction` der Wert $a \cdot x^2 + b \cdot x + c$ und
- für `ExponentialFunction` der Wert $a \cdot e^x + b$

berechnet wird. Erweitern Sie die Klassen um Attribute mit der Zugriffsspezifikation `protected`, welche die benötigten Konstanten `a`, `b` und `c` bereitstellen. Die Attribute sollen jeweils über einen Konstruktor initialisiert werden. (0.6 Punkte)

c) Ergänzen Sie die Klasse `Function` um eine Methode `print`, mit der die Funktionswerte für $x \in \{0, 0.1, \ldots, 0.9, 1.0\}$ auf der Konsole ausgegeben werden. (0.5 Punkte)

d) Ergänzen Sie eine von `Function` abgeleitete Klasse `Compose`, deren `evaluate`-Methode die Berechnungen zweier im Konstruktor als Referenz übergebener Funktionen sequenziell miteinander verknüpft:

$$(f \circ g)(x) = f(g(x))$$

(0.8 Punkte)

e) Schreiben Sie eine `main`-Funktion, welche die Werte der Dichtefunktion der Gauß-Normalverteilung

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\cdot(\frac{x-\mu}{\sigma})^2}$$

für $\sigma = 0.5$ und $\mu = 0.5$ mittels `print` eines geeignet initialisierten `Compose`-Objekts ausgibt. (0.6 Punkte)

f) Kompilieren Sie Ihr Programm, überprüfen Sie Ihre Ergebnisse und kopieren Sie diese als Block-Kommentar an das Ende der Datei `Aufgabe_10_1.cpp`. (0.1 Punkte)

Präsenzaufgabe 2: Ausnahmebehandlung (0 Punkte)

In dem unten angegebenen Programm wird in der Funktion `Formel` die Summe zweier Zahlen x und y durch ihre Quadratwurzel geteilt. Folglich darf die Summe weder Null noch negativ sein, da dann aus unterschiedlichen Gründen Fehler auftreten können. Falls ein Fehler auftritt, dann enthält die Variable `fehler` einen positiven Fehlercode, der nach jedem Aufruf abgeprüft werden muss.

```
1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5  double Formel(double x, double y, int &fehler) {
6      double sum = x + y;
7      if (sum < 0) {
8          fehler = 1;
9      } else if (sum == 0) {
10         fehler = 2;
11     } else {
12         fehler = 0;
13         return sum / sqrt(sum);
14     }
15     return 0;
16 }
17
18 int main() {
19     int fehler;
20     double x[3] = { -2, -4, 0 };
21     double y[5] = { 1, 2, 0, 3, 1 };
22     for (unsigned int i = 0; i < 3; ++i) {
23         for (unsigned int j = 0; j < 5; ++j){
24             double z = Formel(x[i], y[j], fehler);
25             switch (fehler) {
26             case 1:
27                 cerr << "negative Summe" << endl;
28                 break;
29             case 2:
30                 cerr << "Division durch Null" << endl;
31                 break;
32             default:
33                 cout << z << endl;
34             }
35         }
36     }
37     return 0;
38 }
```

Schreiben Sie die Funktion und das Hauptprogramm um, indem Sie den Mechanismus der Ausnahmebehandlung aus C++ mit `try`, `throw` und `catch` verwenden. Die Ausgabe des Programms soll sich nicht ändern. Verwenden Sie für jede Fehlerart eine eigene Fehlerklasse und einen eigenen `catch`-Handler.