Dirk Praetorius,
Marcus Page

Sommersemester 2012
05.06.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 12

Aufgabe 12.1*. (Redefinieren & überladen)
Worin besteht der Unterschied zwischen Redefinition und Überlagerung von Methoden? Was gilt es hierbei zu beachten? Machen Sie sich den Unterschied deutlich, indem Sie eine Klasse `myReal` zur Speicherung von reellen Zahlen schreiben, von der Sie eine Klasse `myComplex` zur Speicherung komplexer Zahlen ableiten. Beide Klassen sollen über verschiedene `print`-Methoden verfügen, durch die sich das Ausgabeformat beeinflussen lässt.

Aufgabe 12.2*. (Polymorphie und abstrakte Klassen)
Den `Minsort`-Algorithmus kennen Sie bereits aus der C-Vorlesung. Schreiben Sie eine Klasse `MinSort` die Sie von `Sortierverfahren` (vgl. Folie 104) aus der Vorlesung ableiten. Rufen Sie die `sort`-Methode von `MinSort` anschließend mittels `sortMe` aus der Vorlesung polymorph auf. Was ist hierbei zu beachten? Was sind die Vor- und Nachteile abstrakter Datentypen? Wann würden Sie abstrakte Datentypen verwenden?

Aufgabe 12.3*. (Effiziente Speicherung 1)
Ändern Sie ihre `Matrix`-Klasse aus der letzten Woche dahingehend ab, dass die Einträge als ein langer Vektor (und nicht als `vector<vector<double> >`) gespeichert werden. Offensichtlich müssen Sie nun auch die Zugriffsfunktionen entsprechend umschreiben. Erweitern Sie die `Matrix`-Klassen außerdem um eine `print`-Methode zur Ausgabe der Einträge am Bildschirm.

Aufgabe 12.4*. (Effiziente Speicherung 2, virtuelle Methoden)
Schreiben Sie eine Klasse `UpperTriangular` zur Speicherung von oberen Dreiecksmatrizen. Diese sollen (wie schon in C) effizient gespeichert werden, d.h. die Nulleinträge unterhalb der Diagonalen sollen nicht gespeichert werden. Um möglichst viel Programmierarbeit zu sparen, soll die Klasse von `SquareMatrix` aus der letzten Woche abgeleitet werden. Legen Sie nun eine obere Dreiecksmatrix an und lassen diese mit der `print`-Routine aus der letzen Aufgabe ausgeben. Was müssen Sie hierbei beachten?

Aufgabe 12.5*. (Polymorphie)
Gegeben seien die Klassen CRectangle und CTriangle, die wie folgt von CPolygon abgeleitet sind:

```
#include <iostream>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void setValues (int a, int b)
      { width=a; height=b; }
  };
```

```
class CRectangle: public CPolygon {
  public:
    int getArea ()
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int getArea ()
      { return (width * height / 2); }
  };
```

Schreiben Sie ein aufrufendes Hauptprogramm, in dem Sie ein Array von 5 Polygonen anlegen. Drei der Polygone sollen hierbei Dreiecke und zwei Rechtecke sein. Befüllen Sie Ihre Polygone mittels `setValues` mit Daten.

Aufgabe 12.6. (Polymorphie)
Ihr Array aus Aufgabe 11.4 soll nun in einer Schleife durchlaufen werden. Hierbei soll die Fläche jedes einzelnen Polygons ausgegeben werden. Wie können Sie das bewerkstelligen?

Aufgabe 12.7. (Polymorphie und abstrakte Klassen)
Von `CPolygon` werden überhaupt keine echten Objekte instanziiert. Das ist auch nicht beabsichtigt. Diese Klasse wird nur zur Vererbung und Ausnutzung der Polymorphie verwendet. Was sollten Sie als C++ Programmierer also tun, damit sich dieser Umstand auch in Ihrem Code widerspiegelt?
Die Ausgabe aus der vorangegangenen Aufgabe ist noch immer etwas armselig. Zusätzlich zur Gesamtfläche des Polygons, soll jedes Polygon nun auch seinen Typ mitteilen. Durchlaufen Sie ihr Array wieder in einer Schleife und stellen Sie sicher, dass jede einzelne Ausgabe etwa wie folgt lautet:

```
Ich bin ein Rechteck und mein Flächeninhalt ist 24.
```

Erweitern Sie alle nötigen Klassen um diese Funktionalität sicherzustellen.

Aufgabe 12.8. (Mehrfachvererbung 1)
Schreiben Sie eine Klasse `PosDefMatrix` zur Speicherung positiv definiter Matrizen, die Sie von `SquareMatrix` ableiten. Eine Matrix $A \in \mathbb{R}^{n\times n}$ bezeichnet man als *positiv definit*, falls für alle reellwertigen Vektoren $x \in \mathbb{R}^n$ gilt $x^T Ax > 0$. Positiv definite Matrizen sind immer invertierbar. Erweitern Sie ihre Klasse um eine Funktion `computeInv` welche die Inverse A^{-1} von A berechnet und zurückgibt, falls $n \leq 2$ ist. Ansonsten soll die Meldung

```
das ist mir noch zu schwierig
```

ausgegeben, und der NULL-Pointer zurückgeliefert werden.

Aufgabe 12.9. (Mehrfachvererbung 2, Diamantvererbung)
Schreiben Sie eine Klasse `SymmMatrix` zur Speicherung symmetrischer Matrizen, die Sie von `SquareMatrix` ableiten. Achten Sie darauf die Einträge effizient zu speichern, d.h. doppelte Einträge sollen nur einmal gespeichert werden. Schreiben Sie sich entsprechende Zugriffsfunktionen.
Schreiben Sie nun eine Klasse `SPDMatrix` zur Speicherung symmetrisch positiv definiter Matrizen. Als *SPD* bezeichnet man Matrizen die sowohl symmetrisch, als auch positiv definit sind. Dementsprechend soll die Klasse `SPDMatrix` sowohl von `SymmMatrix` als auch von `PosDefMatrix` abgeleitet werden. Beide Klassen sind hierbei von `SquareMatrix` abgeleitet, so dass das *Diamantproblem* vorliegt. Welche Möglichkeiten kennen Sie, um die gegebenen Unklarheiten aufzulösen? Welche würden Sie bevorzugen und warum?

Aufgabe 12.10. Stellen Sie Ihre gesamten `Matrix`-Klassen übersichtlich als *UML*-Diagramm dar. Wieso ist es sinnvoll, solche Übersichten zu haben (besonders bei größeren Projekten)?