Übungsblatt 3

(27. April 2007)

(2 Punkte)

Aufgabe 1 Machen Sie jetzt Ihre Klasse `GeometricObject` zu einer Unterklasse der Klasse `ToString` aus dem Skript. Implementieren Sie möglichst aussagekräftig die Methode `toString` in den einzelnen Unterklassen.

Aufgabe 2 Fügen Sie jetzt folgende Schnittstelle Ihrem Projekt hinzu:

Paintable.hpp

```
1  #ifndef PAINTABLE_H_
2  #define PAINTABLE_H_
3  #include <QPainter>
4
5  class Paintable{
6  public:
7    virtual void paintMe(QPainter* p)=0;
8  };
9
10 #endif
```

Diese Klasse, wie die übrigen dieser Aufgabe, ist auch im Netz herunterzuladen:
`http://panitz.name/cpp/student/src/name/panitz/paintable/Paintable.hpp`

Ihre Klasse `GeometricObject` soll jetzt zusätzlich die Schnittstelle `Paintable` erweitern. Hierzu muß die Methode `paintMe` in den entsprechenden Unterklassen implementiert werden. Hier soll sich mit dem `QPainter`-Objekt die geometrische Figur zeichnen. Eine Dokumentation zu `QPainter` finden Sie auf:
`http://doc.trolltech.com/4.2/qpainter.html`

Jetzt sollten Sie in der Lage sein, einzelne geometrische Figuren in einem Fenster graphisch mit der nachfolgenden Klasse in einem Fenster anzuzeigen.

Board.hpp

```
1  #ifndef BOARD_H_
2  #define BOARD_H_
3
4  #include <QWidget>
```

```
5  #include "Paintable.hpp"
6
7  class Board:public QWidget {
8   public:
9    Board(Paintable* pt);
10   Paintable* pt;
11   QSize minimumSizeHint();
12   QSize sizeHint() ;
13   void paintEvent(QPaintEvent *event);
14   static int showPaintable(Paintable* pt,int argc,char **argv);
15 };
16 #endif /*BOARD_H_*/
```

Board.cpp

```
1  #include "Board.hpp"
2  #include <QtGui>
3
4  Board::Board(Paintable* pt): QWidget(0){
5    this->pt=pt;
6    setBackgroundRole(QPalette::Base);
7  }
8
9  QSize Board::minimumSizeHint(){return QSize(100, 100);}
10 QSize Board::sizeHint(){return QSize(400, 300);}
11
12 void Board::paintEvent(QPaintEvent *){
13   QPen pen;
14   QPainter painter(this);
15   painter.setPen(pen);
16   pt->paintMe(&painter);
17 }
18
19 int Board::showPaintable(Paintable* pt,int argc,char **argv){
20   QApplication application(argc,argv);
21   Board* b = new Board(pt);
22   b->show();
23   return application.exec();
24 }
```

Hierzu brauchen Sie eine `main`-Funktion der folgenden Art nur die statische Methde `showPaintable` Aufrufen:

```
1 int main(int argc,char **argv){
2   return Board::showPaintable
3    (new EquilateralTriangle(new Vertex(300,100),50),argc,argv);
4 }
```

Übersetzen Sie das so entstandene Qt-Programm am besten über die Kommandozeile mit `qmake -project`, `qmake` und `make`.

Es ist allerdings mit ein wenig Geschick auch möglich mit einem standard-make-Projekt in Eclipse Qt-Anwendungen zu kompilieren.

Aufgabe 3 Nehmen Sie jetzt die Klasse `Li1` aus dem Skript, die eine einfache verkettete Liste ganzer Zahlen modelliert.

a) Ändern Sie die Klasse so ab, dass Sie nicht `int`-Werte als Elemente hat, sondern Zeiger auf `Paintable`-Objekte. Nennen Sie diese geänderte Klasse `PaintableList`.

b) Dann ändern Sie die Klasse Board so ab, dass jetzt nicht ein einzelnes `Paintable`-Objekt darin abgespeichert ist, sondern ein `PaintableList`-Objekt.

c) In der Methode `Board::paintEvent` soll jetzt nicht nur eine geometrische Figur gezeichnet werden, sondern alle in der Liste gespeicherten Objekte.

d) Fügen Sie in `Board` eine Methode `addPaintable(Paintable*)` hinzu.

e) Schreiben Sie einen Destruktor für die Klasse `Board`, die auch die Liste komplett aus dem Speicher löscht.

f) Stellen Sie in einem Fenster verschiedene geometrische Figuren dar.