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} #define BOARD}
2  #include <QWidget>
3  #include "Paintable.hpp"
4
```

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

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:

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

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.

Lösung

Vertex.hpp

```
#ifndef VERTEX_HPP
#define VERTEX_HPP

#include "../../../ToString.hpp"

class Vertex:public ToString{
 public:
  double x;
  double y;

  Vertex(double x, double y);
  virtual std::string toString();

  virtual Vertex* add(Vertex* that);
  virtual Vertex* sub(Vertex* that);
  virtual Vertex* mult(double d);
  virtual double length();

  virtual void addMod(Vertex* that);
  virtual void subMod(Vertex* that);
  virtual void multMod(double d);
};
#endif
```

ToStringPaintable.hpp

```
#ifndef TO_STRINGPAINTABLE_HPP
#define TO_STRINGPAINTABLE_HPP
```

```
3  #include "Paintable.hpp"
4  #include "../../../ToString.hpp"
5
6  class ToStringPaintable:public ToString, public Paintable{};
7  #endif
```

Vertex.cpp

```
1  #include "Vertex.hpp"
2  #include <cmath>
3
4  Vertex::Vertex(double x, double y){
5    this->x=x;
6    this->y=y;
7  }
8
9  std::string Vertex::toString(){
10   return "("+doubleToString(x)+","+doubleToString(y)+")";
11 }
12
13 Vertex* Vertex::add(Vertex* that){
14   return new Vertex(this->x+that->x,this->y+that->y);
15 }
16 Vertex* Vertex::sub(Vertex* that){
17   return new Vertex(this->x-that->x,this->y-that->y);
18 }
19 Vertex* Vertex::mult(double d){
20   return new Vertex(this->x*d,this->y*d);
21 }
22 double Vertex::length(){
23   return sqrt(this->x*this->x +this->y*this->y);
24 }
25 void Vertex::addMod(Vertex* that){
26   this->x=this->x+that->x;
27   this->y=this->y+that->y;
28 }
29 void Vertex::subMod(Vertex* that){
30   this->x=this->x-that->x;
31   this->y=this->y-that->y;
32 }
33 void Vertex::multMod(double d){
```

```
34   this->x=this->x*d;
35   this->y=this->y*d;
36 }
```

GeometricObject.hpp

```
1  #ifndef GEOMETRIC_OBJECT_HPP
2  #define GEOMETRIC_OBJECT_HPP
3
4  #include "ToStringPaintable.hpp"
5  #include "Vertex.hpp"
6
7  class GeometricObject:public ToStringPaintable {
8   public:
9    Vertex* pos;
10   double width;
11   double height;
12
13   GeometricObject(Vertex* v,double width,double height);
14
15   virtual ~GeometricObject();
16   virtual bool hasWithin(Vertex* p);
17   virtual bool touches(GeometricObject* that);
18   virtual std::string toString();
19   virtual double area()=0;
20 } ;
21 #endif
```

GeometricObject.cpp

```
1  #include "GeometricObject.hpp"
2
3  GeometricObject::GeometricObject
4                     (Vertex* pos,double width,double height){
5    this->pos=pos;
6    this->width=width;
7    this->height=height;
8  }
9
10 GeometricObject::~GeometricObject(){
11   delete pos;
12 }
```

```
13
14 std::string GeometricObject::toString(){
15   return "position = "+pos->toString()+" width = "
16       +doubleToString(width)+" height = "+doubleToString(height)
17       +" area = "+ doubleToString(area());
18 }
19 bool GeometricObject::hasWithin(Vertex* p){
20   return p->x >= this->pos->x
21             && p->x <= this->pos->x+ this->width
22       && p->y >= this->pos->y
23             && p->y <= this->pos->y+ this->height;
24 }
25 bool GeometricObject::touches(GeometricObject* that){
26   if (this->pos->x > that->pos->x + that -> width)
27     return false;
28   if (this->pos->x + this->width < that->pos->x)
29     return false;
30   if (this->pos->y > that->pos->y + that -> height)
31     return false;
32   if (this->pos->y + this->height < that->pos->y)
33     return false;
34   return true;
35 }
```

Rectangle.hpp

```
1  #ifndef RECTANGLE_HPP
2  #define RECTANGLE_HPP
3
4  #include "GeometricObject.hpp"
5
6  class Rectangle: public GeometricObject{
7   public:
8    Rectangle(Vertex* p,double width,double height);
9    virtual double area();
10   virtual std::string toString();
11   virtual void paintMe(QPainter* p);
12 };
13 #endif
```

Rectangle.cpp

```
#include "Rectangle.hpp"
Rectangle::Rectangle(Vertex* p,double width,double height)
  :GeometricObject(p,width,height){}

double Rectangle::area(){return width*height;}

std::string Rectangle::toString(){
  return "Rectangle ->"+GeometricObject::toString();
}

void Rectangle::paintMe(QPainter* p){
  p->drawRect((int)pos->x,(int)pos->y,(int)width,(int)height);
}
```

Square.hpp

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include "Rectangle.hpp"

class Square: public Rectangle{
 public:
  Square(Vertex* p,double width);
  virtual std::string toString();
};

#endif
```

Square.cpp

```
#include "Square.hpp"
Square::Square(Vertex* p,double width)
 :Rectangle(p,width,width){}

std::string Square::toString(){
  return "Square ->"+GeometricObject::toString();
}
```

Oval.hpp

```
#ifndef OVAL_HPP
#define OVAL_HPP

#include "GeometricObject.hpp"

class Oval: public GeometricObject{
 public:
  Oval(Vertex* p,double width,double height);
  virtual double area();
  virtual std::string toString();
  virtual bool hasWithin(Vertex* p);
  virtual void paintMe(QPainter* p);
};
#endif
```

Oval.cpp

```
#include "Oval.hpp"

Oval::Oval(Vertex* p,double width,double height)
  :GeometricObject(p,width,height){}

double Oval::area(){return M_PI*width*height/4;}

std::string Oval::toString(){
  return "Oval ->"+GeometricObject::toString();
}

bool Oval::hasWithin(Vertex* p){
  return pow(p->x-pos->x-width/2,2)/pow(width/2,2)
        +pow(p->y-pos->y-height/2,2)/pow(height/2,2)
     <=1;
}

void Oval::paintMe(QPainter* p){
  p->drawEllipse((int)pos->x,(int)pos->y,(int)width,(int)height);
}
```

Circle.hpp

```
#ifndef CIRCLE_HPP
#define CIRCLE_HPP
```

```
3
4  #include "Oval.hpp"
5
6  class Circle: public Oval{
7   public:
8    Circle(Vertex* p,double width);
9    virtual std::string toString();
10 };
11
12 #endif
```

Circle.cpp

```
1  #include "Circle.hpp"
2  Circle::Circle(Vertex* p,double width):Oval(p,width,width){}
3
4  std::string Circle::toString(){
5    return "Circle ->"+GeometricObject::toString();
6  }
```

EquilateralTriangle.hpp

```
1  #ifndef EQUILATERAL_TRIANGLE_HPP
2  #define EQUILATERAL_TRIANGLE_HPP
3
4  #include "GeometricObject.hpp"
5
6  class EquilateralTriangle: public  GeometricObject{
7   public:
8    EquilateralTriangle(Vertex* p,double width);
9    virtual double area();
10   virtual std::string toString();
11   virtual void paintMe(QPainter* p);
12 };
13 #endif
```

EquilateralTriangle.cpp

```
1  #include "EquilateralTriangle.hpp"
2
3  EquilateralTriangle::EquilateralTriangle(Vertex* p,double width)
4    :GeometricObject(p,width,sqrt(3)*width/2){}
```

```
5
6  std::string EquilateralTriangle::toString(){
7    return "EquilateralTriangle ->"+GeometricObject::toString();
8  }
9
10 double EquilateralTriangle::area(){return width*height/2;};
11
12 void EquilateralTriangle::paintMe(QPainter* p){
13   int x = (int)pos->x;
14   int y = (int)pos->y;
15   int h = (int)height;
16   int w = (int)width;
17   p->drawLine(x,y+h,x+w,y+h);
18   p->drawLine(x+w,y+h,x+(w/2),y);
19   p->drawLine(x+(w/2),y,x,y+h);
20 }
```

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.

Lösung

PaintableList.hpp

```
#ifndef PAINTABLE_LIST_HPP
#define PAINTABLE_LIST_HPP
#include "ToStringPaintable.hpp"

class PaintableList:public ToStringPaintable{
public:
  ToStringPaintable* head;
  PaintableList* tail;
  PaintableList(ToStringPaintable* head=0,PaintableList* tail=0);
  virtual ~PaintableList();

  virtual bool isEmpty();
  virtual std::string toString();
  virtual void paintMe(QPainter* p);
};
#endif
```

PaintableList.cpp

```
#include "PaintableList.hpp"

PaintableList::PaintableList(ToStringPaintable* head,PaintableList* tail)
  this->head=head;
  this->tail=tail;
}

bool PaintableList::isEmpty(){
  return this->tail==NULL;
}

PaintableList::~PaintableList(){
  if (!this->isEmpty()) delete this->tail;
}

std::string PaintableList::toString(){
  if (this->isEmpty()) return "[]";
  std::string result  = "["+head->toString();
  for (PaintableList* tmp=this->tail;!tmp->isEmpty();tmp=tmp->tail){
      result=result+","+tmp->head->toString();
```

```
21    }
22    return result+"]";
23  }
24
25  void PaintableList::paintMe(QPainter* p){
26    for (PaintableList* tmp=this;!tmp->isEmpty();tmp=tmp->tail){
27      tmp->head->paintMe(p);
28    }
29  }
```

FirstPaints.cpp

```
1   #include "Board.hpp"
2   #include "Star.hpp"
3   #include "Circle.hpp"
4   #include "Square.hpp"
5
6   #include "PaintableList.hpp"
7   #include <iostream>
8
9   int main(int argc,char **argv){
10    PaintableList* xs
11        = new PaintableList(new Star(new Vertex(0,0),100,100,2)
12         ,new PaintableList(new Star(new Vertex(100,100),20,50,10)
13         ,new PaintableList(new Rectangle(new Vertex(0,0),100,100)
14         ,new PaintableList(new Circle(new Vertex(0,0),100)
15         ,new PaintableList(new Star(new Vertex(10,10),50,50,1000)
16         ,new PaintableList())))));
17
18    xs->println();
19
20    int res = Board::showPaintable(xs,argc,argv);
21
22    for (PaintableList* tmp=xs;!tmp->isEmpty();tmp=tmp->tail){
23      delete (tmp->head);
24    }
25
26    std::cout <<"everything deletet"<< std::endl;
27    return res;
28  }
```