Übungsblatt 2

(20. April 2007)

Aufgabe 1 Sie sollen auf diesem Übungsblatt die Aufgaben des ersten Übungsblattes weiterführen und die dort programmierten Klassen benutzen und gegebenenfalls verändern. Allerdings soll die C-Version der Lösung des ersten Übungsblattes nicht weiter verfolgt werden.

a) Machen Sie `GeometricObject` zu einer abstrakten Klassen und fügen ihr eine abstrakte Methode zur Berechnung des Flächeninhalts zu. Benutzen Sie diese abstrakte Methoden in der Methode `print`.

b) Implementieren Sie eine konkrete Unterklasse `Rectangle` der Klasse `GeometricObject`, die Rechtecke darstellt. Schreiben Sie geeignete Konstruktoren und überschreiben Sie die Methoden `print`.

c) Implementieren Sie eine Unterklasse `Square` der Klasse `Rectangle`, die Quadrate darstellt. Schreiben Sie geeignete Konstruktoren.

d) Implementieren Sie eine konkrete Unterklasse `Oval` der Klasse `GeometricObject`, die Ellipsen darstellt. Schreiben Sie geeignete Konstruktoren und überschreiben Sie die Methoden `print` und `hasWithin`.

e) Implementieren Sie eine Unterklasse `Circle` der Klasse `Oval`, die Kreise darstellt. Schreiben Sie geeignete Konstruktoren.

f) Implementieren Sie eine konkrete Unterklasse `EquilateralTriangle` der Klasse `GeometricObject`, die gleichseitige Dreiecke darstellt. Schreiben Sie geeignete Konstruktoren und überschreiben Sie die Methoden `print`.

g) Schreiben Sie Tests, in denen von jeder Klasse Objekte erzeugt und benutzt werden.

Lösung

Geos.hpp

```
1 class Vertex{
2  public:
3   double x;
4   double y;
5
```

```
6    Vertex(double x, double y);
7    virtual void print();
8
9    virtual Vertex* add(Vertex* that);
10   virtual Vertex* sub(Vertex* that);
11   virtual Vertex* mult(double d);
12   virtual double length();
13
14   virtual void addMod(Vertex* that);
15   virtual void subMod(Vertex* that);
16   virtual void multMod(double d);
17 };
18
19 class GeometricObject{
20  public:
21   Vertex* pos;
22   double width;
23   double height;
24
25   GeometricObject(Vertex* v,double width,double height);
26   virtual void print();
27
28   virtual bool hasWithin(Vertex* p);
29   virtual bool touches(GeometricObject* that);
30
31   virtual double area()=0;
32 } ;
33
34 class Rectangle: public GeometricObject{
35  public:
36   Rectangle(Vertex* p,double width,double height);
37   virtual double area();
38   virtual void print();
39 };
40 class Square: public Rectangle{
41  public:
42   Square(Vertex* p,double width);
43   virtual void print();
44 };
45 class Oval: public GeometricObject{
46  public:
```

```
47    Oval(Vertex* p,double width,double height);
48    virtual double area();
49    virtual void print();
50    virtual bool hasWithin(Vertex* p);
51  };
52
53  class Circle: public Oval{
54   public:
55    Circle(Vertex* p,double width);
56    virtual void print();
57  };
58
59  class EquilateralTriangle: public  GeometricObject{
60   public:
61    EquilateralTriangle(Vertex* p,double width);
62    virtual void print();
63    virtual double area();
64  };
```

Geos.cpp

```
1   #include "Geos.hpp"
2   #include <iostream>
3   #include <cmath>
4
5   Vertex::Vertex(double x, double y){
6     this->x=x;
7     this->y=y;
8   }
9
10  void Vertex::print(){
11    std::cout<<"("<<this->x<<","<<this->y<<")";
12  }
13
14  Vertex* Vertex::add(Vertex* that){
15    return new Vertex(this->x+that->x,this->y+that->y);
16  }
17  Vertex* Vertex::sub(Vertex* that){
18    return new Vertex(this->x-that->x,this->y-that->y);
19  }
20  Vertex* Vertex::mult(double d){
```

```
21    return new Vertex(this->x*d,this->y*d);
22  }
23  double Vertex::length(){
24    return sqrt(this->x*this->x +this->y*this->y);
25  }
26  void Vertex::addMod(Vertex* that){
27    this->x=this->x+that->x;
28    this->y=this->y+that->y;
29  }
30  void Vertex::subMod(Vertex* that){
31    this->x=this->x-that->x;
32    this->y=this->y-that->y;
33  }
34  void Vertex::multMod(double d){
35    this->x=this->x*d;
36    this->y=this->y*d;
37  }
38  GeometricObject::GeometricObject
39                      (Vertex* pos,double width,double height){
40    this->pos=pos;
41    this->width=width;
42    this->height=height;
43  }
44  void GeometricObject::print(){
45    std::cout<<"GeometricObject(";
46    this->pos->print();
47    std::cout<<","<<this->width<<","<<this->height<<")"<<"->area = "<<area(
48  }
49  bool GeometricObject::hasWithin(Vertex* p){
50    return p->x >= this->pos->x
51               && p->x <= this->pos->x+ this->width
52        && p->y >= this->pos->y
53               && p->y <= this->pos->y+ this->height;
54  }
55  bool GeometricObject::touches(GeometricObject* that){
56    if (this->pos->x > that->pos->x + that -> width)
57      return false;
58    if (this->pos->x + this->width < that->pos->x)
59      return false;
60    if (this->pos->y > that->pos->y + that -> height)
61      return false;
```

```
62    if (this->pos->y + this->height < that->pos->y)
63      return false;
64    return true;
65  }
66
67  Rectangle::Rectangle(Vertex* p,double width,double height)
68    :GeometricObject(p,width,height){}
69
70  double Rectangle::area(){return width*height;}
71  void Rectangle::print(){
72    std::cout<<"Rectangle ->";
73    GeometricObject::print();
74  }
75
76  Square::Square(Vertex* p,double width)
77   :Rectangle(p,width,width){}
78
79  void Square::print(){
80    std::cout<<"Square ->";
81    GeometricObject::print();
82  }
83
84  Oval::Oval(Vertex* p,double width,double height)
85    :GeometricObject(p,width,height){}
86
87  double Oval::area(){return M_PI*width*height/4;}
88
89  void Oval::print(){
90    std::cout<<"Oval ->";
91    GeometricObject::print();
92  }
93  bool Oval::hasWithin(Vertex* p){
94    return pow(p->x-pos->x-width/2,2)/pow(width/2,2)
95          +pow(p->y-pos->y-height/2,2)/pow(height/2,2)
96       <=1;
97  }
98
99
100 Circle::Circle(Vertex* p,double width):Oval(p,width,width){}
101
102 void Circle::print(){
```

```
103   std::cout<<"Circle ->";
104   GeometricObject::print();
105 }
106
107
108 EquilateralTriangle::EquilateralTriangle(Vertex* p,double width)
109   :GeometricObject(p,width,sqrt(3)*width/2){}
110
111 void EquilateralTriangle::print(){
112   std::cout<<"EquilateralTriangle ->";
113   GeometricObject::print();
114 }
115 double EquilateralTriangle::area(){return width*height/2;};
```