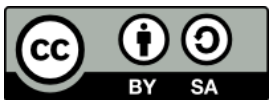
Наследяване (упражнение)

Любомир Чорбаджиев[1]
lchorbadjiev@elsys-bg.org

[1]Технологическо училище "Електронни системи"
Технически университет, София

29 март 2009 г.



Съдържание

Point.hpp

```
1 #ifndef POINT_HPP__
2 #define POINT_HPP__
3
4 class Point {
5   double x_, y_;
6 public:
7   Point(double x=0.0, double y=0.0)
8     : x_(x), y_(y)
9   {}
10  double get_x(void) const {return x_;}
11  double get_y(void) const {return y_;}
```

Point.hpp

```
1   Point& set_x(double x) {
2     x_=x;
3     return *this;
4   }
5   Point& set_y(double y) {
6     y_=y;
7     return *this;
8   }
9   void print(void) const;
10 };
11
12 #endif
```

Point.cpp

```
1 #include <iostream>
2 #include "Point.hpp"
3 using namespace std;
4
5 void Point::print(void) const {
6   cout << "(" << x_ << ",␣" << y_ << ")";
7 }
```

Shape.hpp

```
1 #ifndef SHAPE_HPP__
2 #define SHAPE_HPP__
3
4 class Shape {
5 public:
6   virtual void print(void) const=0;
7 };
8
9 #endif
```

Rectangle.hpp

```
1 #ifndef RECTANGLE_HPP__
2 #define RECTANGLE_HPP__
3
4 #include "Point.hpp"
5 #include "Shape.hpp"
6
7 class Rectangle: public Shape {
8   Point ul_;
9   Point br_;
10 public:
11   Rectangle(const Point& ul, const Point& br);
12   void print(void) const;
13 };
14 #endif
```

Rectangle.cpp

```
1 #include <iostream>
2 #include "Rectangle.hpp"
3 using namespace std;
4
5 Rectangle::Rectangle(const Point& ul,
6                      const Point& br)
7   : ul_(ul), br_(br)
8 {}
9 void Rectangle::print(void) const {
10   cout << "Rectangle(";
11   ul_.print();
12   cout << ", ";
13   br_.print();
14   cout << ")" << endl;
15 }
```

Circle.hpp

```
1  #ifndef CIRCLE_HPP__
2  #define CIRCLE_HPP__
3
4  #include "Point.hpp"
5  #include "Shape.hpp"
6
7  class Circle: public Shape {
8    Point center_;
9    double radius_;
10 public:
11   Circle(const Point& center, double radius);
12   void print(void) const;
13 };
14 #endif
```

Circle.cpp

```
1 #include <iostream>
2 #include "Circle.hpp"
3 using namespace std;
4
5 Circle::Circle(const Point& center,
6                double radius)
7   : center_(center), radius_(radius)
8 {}
9 void Circle::print(void) const {
10   cout << "Circle(";
11   center_.print();
12   cout << ",␣" << radius_ << ")" << endl;
13 }
```

Drawing.hpp

```
1 #ifndef DRAWING_HPP__
2 #define DRAWING_HPP__
3
4 #include <list>
5 #include "Shape.hpp"
6 using namespace std;
7
8 class Drawing {
9   list<const Shape*> shapes_;
10 public:
11   void add(const Shape* shape);
12   void print(void) const;
13 };
14 #endif
```

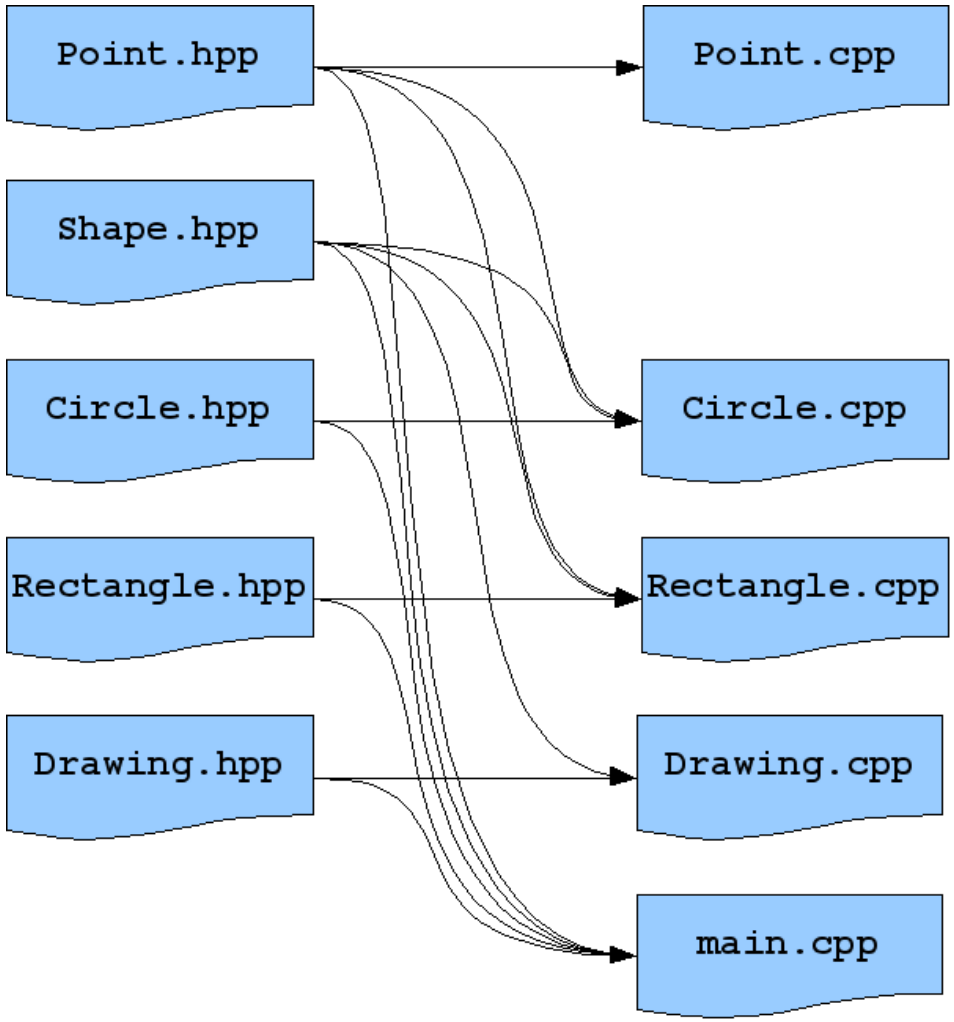
Drawing.cpp

```
1 #include <iostream>
2 #include "Drawing.hpp"
3 using namespace std;
4 void Drawing::add(const Shape* shape) {
5   shapes_.push_back(shape);
6 }
7 void Drawing::print(void) const {
8   cout << "Drawing{" << endl;
9   for(list<const Shape*>::const_iterator
10        it=shapes_.begin();it!=shapes_.end();++it){
11     cout << "\t";
12     (*it)->print();
13   }
14   cout << "}" << endl;
15 }
```

main.cpp

```
1 #include "Drawing.hpp"
2 #include "Rectangle.hpp"
3 #include "Circle.hpp"
4 #include "Point.hpp"
5 int main(void) {
6   Rectangle r1(Point(0,0),Point(10,10));
7   Circle c1(Point(0,0),10), c2(Point(10,10),10);
8
9   Drawing d1;
10  d1.add(&r1);
11  d1.add(&c1);
12  d1.add(&c2);
13  d1.print();
14  return 0;
15 }
```

Зависимост между файловете



make

- За управление на зависимости между файлове се използва програмата `make`.
- Зависимостите между файловете и правилата по-които се създават нови файлове от изходните се описват във файл. Традиционно този файл се нарича `Makefile`.
- Програмата `make` чете този файл и използвайки правилата, описани в него създава производните файлове.
- Най-често програмата `make` се използва за компилиране на приложения.
- http://www.gnu.org/software/make/manual/make.html
- Въведение в работата с GNU Make

Синтаксис на Makefile

- Синтаксисът на Makefile е следния:

```
<target>: <dependencies>
  <rule 1>
  <rule 2>
  ...
```

- Например, ако искаме да опишем как файлът Point.cpp се компилира до обектен файл, трябва да напишем следното:

```
Point.o: Point.cpp Point.hpp
  g++ -c Point.cpp -o Point.o
```

Makefile

```
1  all: drawing
2
3  Point.o: Point.cpp Point.hpp
4  Circle.o: Circle.cpp Circle.hpp Shape.hpp \
5    Point.hpp
6  Rectangle.o: Rectangle.cpp Rectangle.hpp \
7    Shape.hpp Point.hpp
8  Drawing.o: Drawing.cpp Drawing.hpp Shape.hpp
9  main.o: main.cpp Drawing.hpp Rectangle.hpp \
10   Circle.hpp Shape.hpp Point.hpp
```

Makefile

```
1 drawing: main.o Drawing.o Rectangle.o \
2   Circle.o Point.o
3   g++ main.o Drawing.o Rectangle.o Circle.o \
4   Point.o -o drawing
5
6 clean:
7   rm -f *.o *~ drawing
```

Задачи

1. Проверете какъв ще бъде ефекта, ако функция `Shape::print()` **const** не е виртуална.
2. Добавете нова фигура `Line` в йерархията от фигури.
   1. Фигурата `Line` трябва да представя линия (отсечка) между две точки. В метода `Line::print()` **const** се отпечатват двете точки, които дефинират линията (отсечката).
   2. Добавете необходимите за компилирането на новите файлове `Line.hpp` и `Line.cpp` правила в `Makefile`.
   3. Добавете в `main.cpp` използване на новата фигура.