AlexOne

七、桥接模式

目录

1 传统方法

```
#include <iostream>

using namespace std;

// 继承机制将抽象部分与它实现部分固定在一起,
// 使得难以对抽象部分和实现部分(SolidLine/DotLine)独立地进行修改、扩充和重用
# if 0
          Shape
      Circle   Rect
#endif

class Shape{
public:
    Shape(int x=0,int y=0)
        :_x(x),_y(y){}
    virtual void drawShape()=0;
protected:
    int _x,_y;
};

class Circle : public Shape{
public:
    Circle(int x=0,int y=0,int ridus=0):Shape(x,y),_ridus(ridus){}
    void drawShape(){
        cout<<"draw from "<<"("<<_x<<","<<_y<<")"<<" ridus : "<<_ridus<<endl;
    }
protected:
    int _ridus;
};

class Rect:public Shape{
public:
    Rect(int x=0,int y=0,int len=0,int wid=0):Shape(x,y),_len(len),_wid(wid){}
    void drawShape(){
        cout<<"draw from "<<"("<<_x<<","<<_y<<")"<<" len : "<<_len<<" wid : "
<<_wid<<endl;
    }
protected:
    int _len;
    int _wid;
```

```
};


int main(){
    // 这是未加Line * pl指针的时候的用法

    Circle  c(1,2,3);
    c.drawShape();
    Rect  r(1,2,3,4);
    r.drawShape();

    return 0;
}
```

2 桥接模式

```
#include <iostream>

using namespace std;

// 继承机制将抽象部分与它实现部分固定在一起，
// 使得难以对抽象部分和实现部分独立地进行修改、扩充和重用

// 解决:
// 1.将实现部分封装到一个到一个抽象类中。
// 2.在被实现的抽象基类中，包含一个实现部分的基类引用。
# if 0
           Line
          / | \
    DotLine |  SolidLine     //现在可以这样设计了。可以画实线&圆、虚线&圆、实线&矩形、虚线&矩形
          Shape
          /    \
      Circle   Rect
#endif

class Line{
public:
    virtual void drawLine()=0;
};

class DotLine:public Line{
public:
    void drawLine(){
        cout<<"我可以画虚线"<<endl;
    }
};

class SolidLine:public Line{
public:
    void drawLine(){
        cout<<"我可以画实线"<<endl;
    }
};

class Shape{        // 此时，Shape就是一个桥，由Circle、Rect去继承
public:
    Shape(int x=0,int y=0,Line *pl=NULL)
        :_x(x),_y(y),_pl(pl){}
```

```
    virtual void drawShape()=0;
protected:
    int _x,_y;
    Line * _pl;     // 在被实现的抽象基类中，包含一个实现部分的基类引用。
};

class Circle : public Shape{
public:
    Circle(int x=0,int y=0,int ridus=0,Line *pl=NULL): Shape(x,y,pl),_ridus(ridus){}
    void drawShape(){
        cout<<"draw from "<<"("<<_x<<","<<_y<<")"<<"ridus"<<_ridus<<endl;
        _pl->drawLine();
    }
protected:
    int _ridus;
};

class Rect:public Shape{
public:
    Rect(int x=0,int y=0,int len=0,int wid=0,Line
*pl=NULL):Shape(x,y,pl),_len(len),_wid(wid){}
    void drawShape(){
        cout<<"draw from "<<"("<<_x<<","<<_y<<")"<<"len"<<_len<<"wide "<<_wid<<endl;
        _pl->drawLine();
    }
protected:
    int _len;
    int _wid;
};


int main(){
    // 这是未加Line * pl指针的时候的用法

    Circle  c(1,2,3);
    c.drawShape();
    Rect  r(1,2,3,4);
    r.drawShape();
/*
    DotLine  dl;
    Circle  c(1,2,3,&dl);
    c.drawShape();

    SolidLine sl;
    Rect  r(1,2,3,4,&sl);
    r.drawShape();
*/
    return 0;
}
```