# C++ Programming

## Chapter 12 Polymorphism 多态性

Zheng Guibin
（郑贵滨）

# 12.1 What is polymorphism（什么是多态）

## 多态性(Polymorphism)

- 源于希腊语，是“多种形式”的意思。
- 多态性是面向对象编程最重要的特征之一，是指不同的对象对同一命令(在面向对象编程术语中，命令又称为“消息”)做出不同响应的能力。
- 多态性在我们日常用语中被广泛使用。例如：
  同一词“打开(Open)”，用在不同的对象上表示不同的含义：
  - 开银行账户
  - 开窗户
  - 电脑屏幕上打开窗口又有区别。

# 12.1 What is polymorphism

```
class advanced_computer
{
public:
      void hello()
      {
        cout << "Hello from the Advanced Computer" << endl ;
      }
} ;

class simple_computer
{
public:
      void hello()
      {
        cout << "Hello from the Simple Computer" << endl ;
      }
} ;
```

```
main()
{
  advanced_computer HAL ;
  simple_computer PC ;
  HAL.hello() ;
  PC.hello() ;
}
```

# 12.1 What is polymorphism（什么是多态）

- 多态性主要分为两大类
  - 静态(编译时)多态性/ *static* (or compile-time) *polymorphism*
    发生在程序被编译的时候
  - 动态(运行时)多态性/*dynamic* (or run-time) *polymorphism*
    发生在程序运行的时候。
- C++有3种静态多态性机制:
  - 函数重载(第7章)
    函数具有相同的名字，不同的参数列表，就意味着函数重载。
  - 运算符重载 (第10章)
  - 模板 (第13章).
  - Example: P12B

# 12.2 Virtual functions(虚函数)

- **静态联编(早期联编、静态绑定static or early binding)**
  哪个函数被调用是在编译时被确定的，
  编译器根据调用函数的对象来确定调用哪个函数。
- **动态联编/后期联编（ dynamic or late binding）**
  程序运行时才能决定哪个函数被调用，即根据执行调用的对象确定哪个函数被调用。
- **C++ 使用虚函数来实现动态联编**
  - 父类用关键字virtual声明虚成员函数，子类的同名函数也默认是虚函数
  - 为清楚起见，在子类中最好使用关键字virtual来声明虚函数，尽管关键字virtual是可选的。
  - 虚函数使得多态性在所有情况下都能起作用

# 12.2 Virtual functions(虚函数)

```
class circle
{
public:
  circle( double r ) ;
  virtual double area() ;
  void area_message( string message ) ;
protected:
  double radius ;
} ;
// circle member functions.
circle::circle( double r ) : radius( r )
{}

double circle::area()
{
  return pi * radius * radius ;
}
```

# 12.2 Virtual functions

```
// circle member functions.
circle::circle( double r ) : radius( r )
{}

double circle::area()
{
  return pi * radius * radius ;
}

void circle::area_message( string message )
{
  cout << message <<  area() << endl ;
}
```

## 12.2 Virtual functions

```
// class cylinder derived from class circle.
class cylinder : public circle
{
public:
  cylinder( double r, double l ) ;
  virtual double area() ;
private:
  double length ;
} ;

// cylinder member functions.
cylinder::cylinder( double r, double l ) : circle( r ), length( l )
{}

double cylinder::area()
{
  return  2 * pi * radius * ( radius + length ) ;
}
```

## 12.2 Virtual functions

```
// class sphere derived from class circle.
class sphere : public circle
{
public:
  sphere( double r ) ;
  virtual double area() ;
} ;
// sphere member functions.
sphere::sphere( double r ) : circle( r )
{}

double sphere::area()
{
  return  4 * pi * radius * radius ;
}
```

## 12.2 Virtual functions

```
main()
{
  char shape ;
  double radius, height ;
  circle* ptr ;

  cout << "Enter a shape (1=circle, 2=cylinder 3=sphere) " ;
  cin >> shape ;

  if ( shape > '0' && shape < '4' )
  {
    cout << "Enter Radius " ;
    cin >> radius ;

    if ( shape == '1' )
    {
      ptr = new circle( radius ) ;
    }
```

# 12.2 Virtual functions

```
//main() …

   else if ( shape == '2' )
    {
      cout << "Enter Height " ;
      cin >> height ;
      ptr = new cylinder( radius, height ) ;
    }
    else if ( shape == '3' )
    {
      ptr = new sphere( radius ) ;
    }
    ptr -> area_message( "The area is: " ) ;
  }
  else
    cout << "Invalid input" << endl ;
}
```

# 12.2 Virtual functions

## 12.2.1 When to use virtual functions（何时使用虚函数）

- 使用虚函数会涉及内存和执行时间开销的问题，所以在选择一个基类函数是否声明为虚函数时要慎重考虑。
- 一般而言，如果一个基类的成员函数可能在派生类中被覆盖，那么就应将其声明为虚函数。

## 12.2.2 Overriding and overloading（覆盖和重载）

- 重载是一项编译器技术，用来区分函数名字相同但是参数列表不同的函数。函数重载已在第7章介绍。
- 覆盖发生在继承的时候，当派生类的成员函数与基类的成员函数的函数名和参数列表都相同时便会发生覆盖。

# 12.3 Abstract base classes（抽象基类）

- **纯虚函数（pure virtual function）**
  - 基类成员函数display_details()内没有代码，并且被赋值为0。

    **virtual void display_details() = 0 ;**
  - 用上面这种方式定义的类的成员函数，称为纯虚函数。
  - 纯虚函数只是为了编译器的需要而存在，实际上并不做任何事情。
  - **纯虚函数在所有的派生类中都被覆盖，**因为没有必要实现。
- **抽象基类（Abstract base classes，ABCs）**
  - 包含纯虚函数的基类。

# 12.3 Abstract base classes（抽象基类）

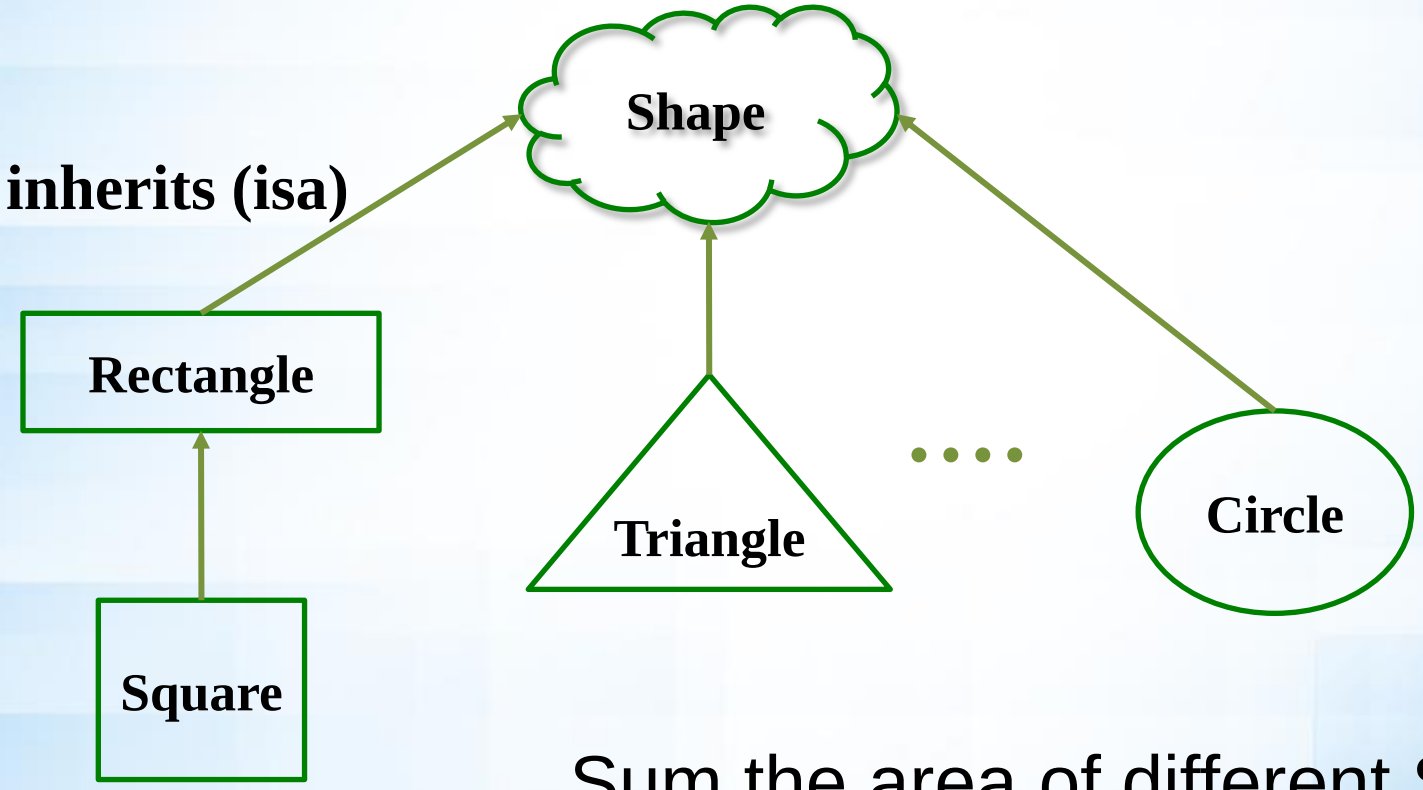
抽象基类(ABCs)不是用来创建对象的，他们的用途仅仅是作为基类去派生其它的类。

定义一个抽象基类的对象是不可能的，也就是说，抽象基类的对象不能实例化。

虽然一个类层次结构不必包含抽象基类，但在形成类层次结构的第一阶段通常是定义一个抽象基类。

# 12.3 Abstract base classes

- **Example: Shape class hierarchy**




Sum the area of different Shapes?

# 12.3 Abstract base classes（抽象基类）

```
//抽象类& 纯虚函数
class Shape    //Abstract
{
  public :
  virtual double get_area() = 0; //Pure virtual Function
};
```

- 具有一个或者多个纯虚函数的类称为抽象类
- Shape只是作为其他派生类的一个基类才有意义，作为一个“category”
- 不能创建抽象类的对象,但可以声明指向抽象类的指针变量和引用变量。

```
Shape s; // error : variable of an abstract class
```

# 12.3 Abstract base classes（抽象基类）

## Derived Class Circle

```
class Circle: public Shape
{
   public:
            Circle( double r );
            double get_area();
   private:
            double radius;
};
```

```
Circle:: Circle( double r )
{
    radius = r;
}
```

```
double Circle:: get_area()
{
    const double pi=3.14;
    return pi*radius*radius;
}
```

# 12.3 Abstract base classes（抽象基类）

## Derived Class Rectangle

```
class Rectangle: public Shape
{
    public:
            Rectangle(double w, double l);
            double get_area();
     private:
            double width, length;
};
```

```
Rectangle::Rectangle(double w, double l)
 {
    width = w;
    length = l;
 }
```

```
double  Rectangle::get_area()
 {
    return width*length;
 }
```

# 12.3 Abstract base classes（抽象基类）

## 如何计算不同形状?——静态绑定

```
int main()
{
    Circle circle(3);
    Rectangle rect(4, 5);

    double sum = 0;
    sum += circle.get_area();
    sum += rect.get_area();

  return 0;
}
```

# 12.3 Abstract base classes（抽象基类）

如何计算不同形状？——动态绑定

```
int main()
{
    Shape *shapes[2] = {new Circle(3), new Rectangle(4, 5)};
    double sum = 0;
    for(int i = 0; i < 2; i++)
    {
        sum += shapes[i]->get_area();
    }
    cout << sum << endl;
    return 0;
}
```

# Programming pitfalls

1.虚函数和虚基类很容易让人混淆。虽然他们都是用在继承的环境下，但是实际上它们并没有什么关系。

2.抽象基类必须至少有一个派生类。定义一个抽象基类的对象将会引发错误。

3.只有在派生类中的成员函数和基类的成员函数的函数原型相同时，基类中的这个虚函数才能被派生类中的函数覆盖。

# Programming pitfalls

4.基类中的虚函数能在一个或者多个派生类中被覆盖。只能在基类中声明一个函数为虚函数，派生类中不能定义虚函数。遗憾的是，这将削弱重用基类的能力，要想重用基类，必须先对其进行修改。

5.关键字virtual不能在类声明之外使用。在定义一个非内联成员函数时，使用关键字virtual是一个很常见的错误。

```
class b
   {
        public:
        virtual void mf() ; // virtual is valid here.
   } ;
   virtual void b::mf() // error: virtual is not valid here.
   {
       ...
   }
```

# Q & A

Thank You!