第6章　多态

C++语言程序设计教程—大连理工大学软件学院

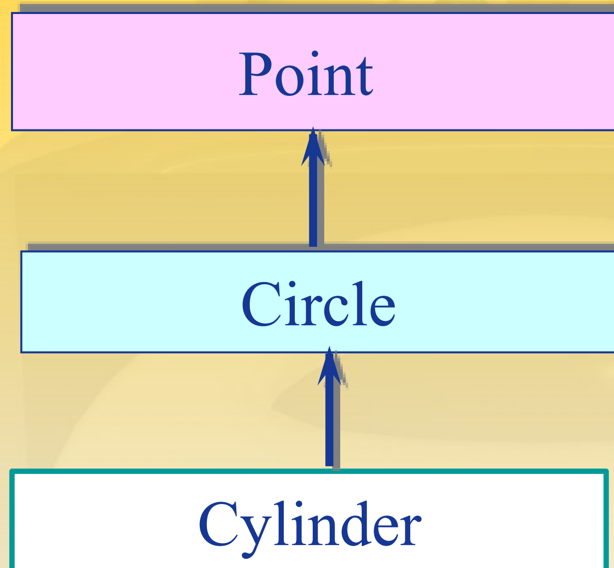
面向对象 OOP

抽象(abstraction)

- 封装(encapsulate)
- 继承 (inheritance)
- 多态(polymorphism)

◆课堂练习

定义如下继承体系，能否使用一种容器，存储各种类对象？



第6章　　多态

6.1 理解多态

6.2 多态的实现

6.3 虚函数

6.4 虚析构函数

6.5 纯虚函数与抽象类

6.1 理解多态

6.1 理解多态

程序

中多态现象比比皆是。

不同消息接收者产生
不同行为

6.1 理解多态

- 多态性（Polymorphism）

所谓多态性是指同样的消息被不同类型的对象接收时产生不同行为的现象。

例如，鼠标单击，响应不同

- 向不同的对象发送同一个消息，
  不同的对象在接收时会产生不同的行为

同一名字，多种语义；

同个接口，多种方法；

```
Complex c1(1,2), c2(3,4), c3;
int i, j = 6;
c3 = c1 + c2 ;
i =  j + 2 ;
```

6.1 理解多态

多态性（Polymorphism）

不同的类可以有同名的方法，
但其具体的实现和结果可以各不相同。

bush.roar()

orang.roar()

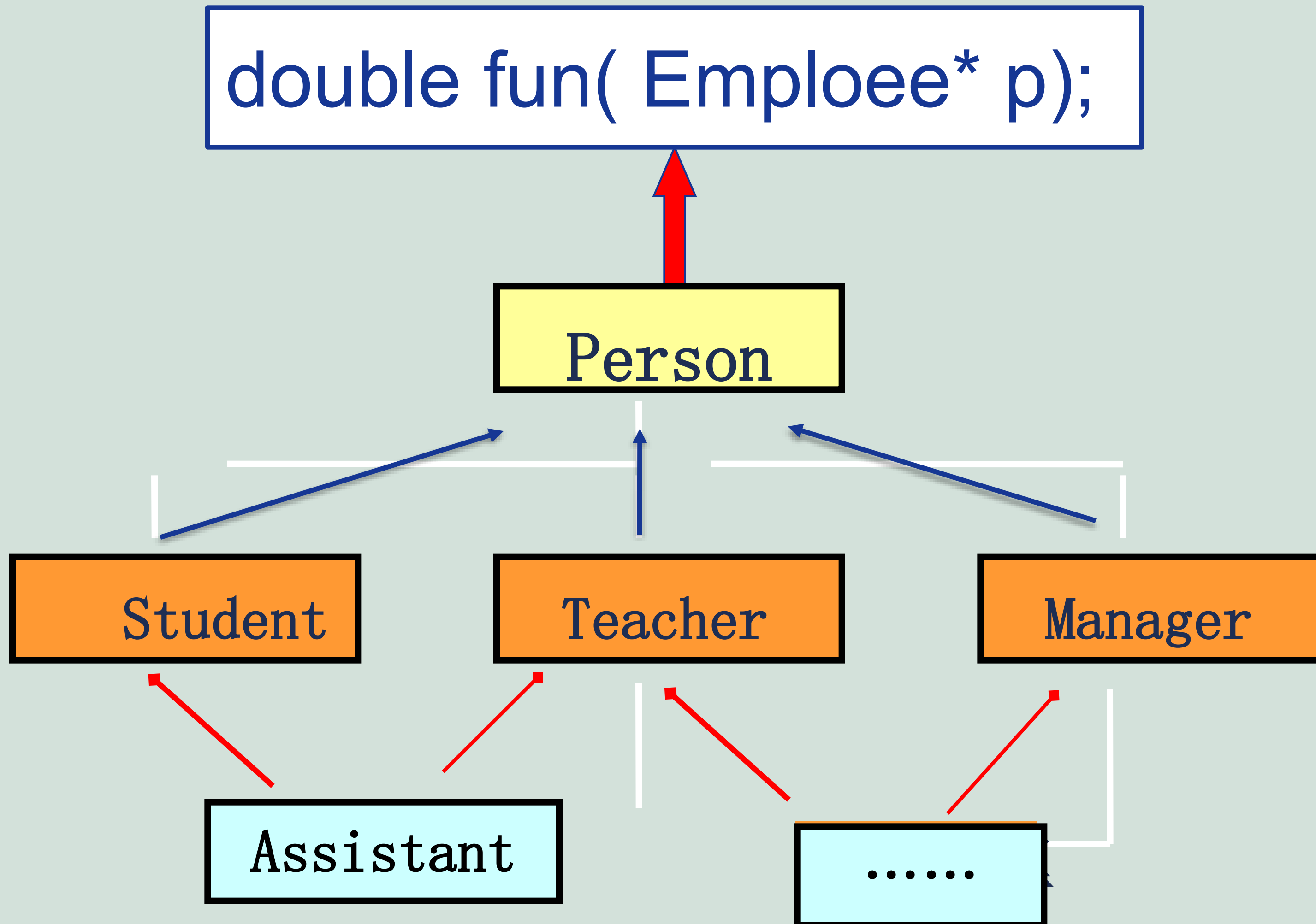
donald.roar()

客户以相同的方法请求，同一消息为不同的对象接受时可产生完全不同的结果。

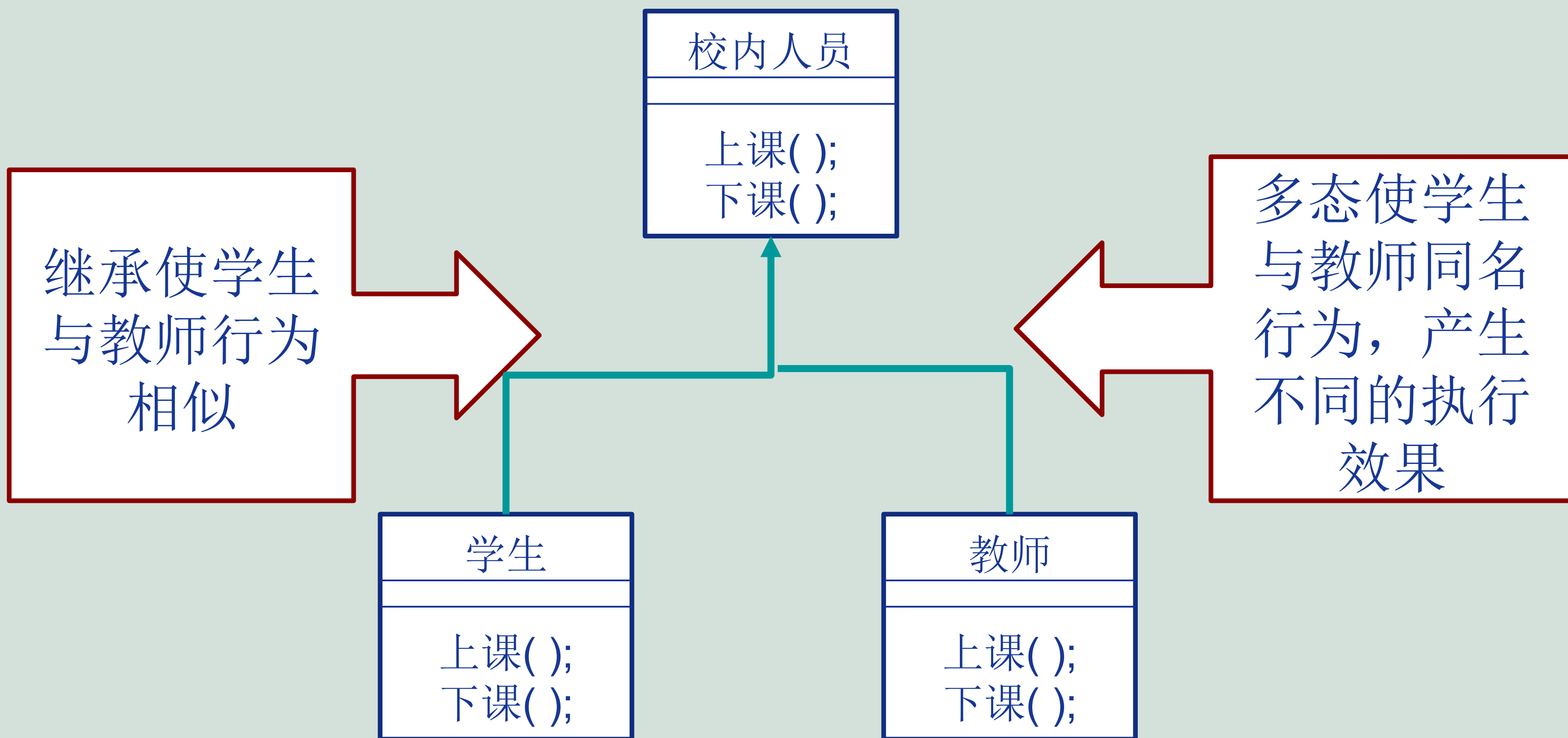
6.1 理解多态

第5章 继承

- 多态性（Polymorphism）



6.1 理解多态



本章介绍的多态性与第五章介绍的继承性相结合，
可以生成一系列虽彼此相似却又独一无二的类和对象。

6.2 多态的实现

- 多态分两类：**静态多态与动态多态**。
  - **静态多态**是指在程序编译时系统就能够确定要调用的是哪个函数，也被称为**编译时多态**。
  - 通过**函数的重载**来实现。

class A
int data void set (char) void set (int) void show();

A objA;

objA.set('a');

objA.set (5);

函数重载注意事项

- 不能仅靠函数的返回值来区别重载函数，必须从形式参数上区别开来。
- 不同参数传递方式也无法区别重载函数

```
void func(int value);
void func(int &value);
```

- 由typedef定义的类型别名并没有真正创建一个新的类型

```
typedef long size_t;
long function(long num);
size_t  function(size_t num);
```

静态多态

1. 类中重载成员函数

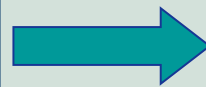
根据参数的特征加以区分

2. 派生类中的同名成员函数

- 使用“ :: ”加以区分
- 使用对象加以区分

静态关联（static binding在运行前进行的关联，又称为**早期关联**。

class B :public A
char name[10];
A::Show ( )
Show ( )



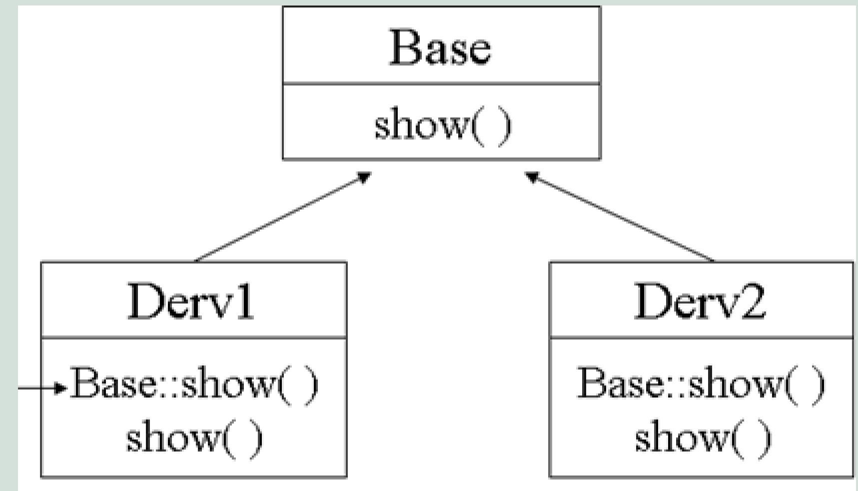
```
Aobj . Show ( );
Bobj . Show ( );
Bobj . A :: Show ( );
```

例　基类指针中调用同名成员函数

```
class Base
{   public:
      void show()
     { cout << "Base"<<endl ; }
};

class Derv1: public Base
{  public:
      void show(){ cout << "Derv1"<<endl ; }
};

class Derv2: public Base
{  public:
      void show(){ cout << "Derv2"<<endl ; }
};
```



例　基类指针中调用同名成员函数

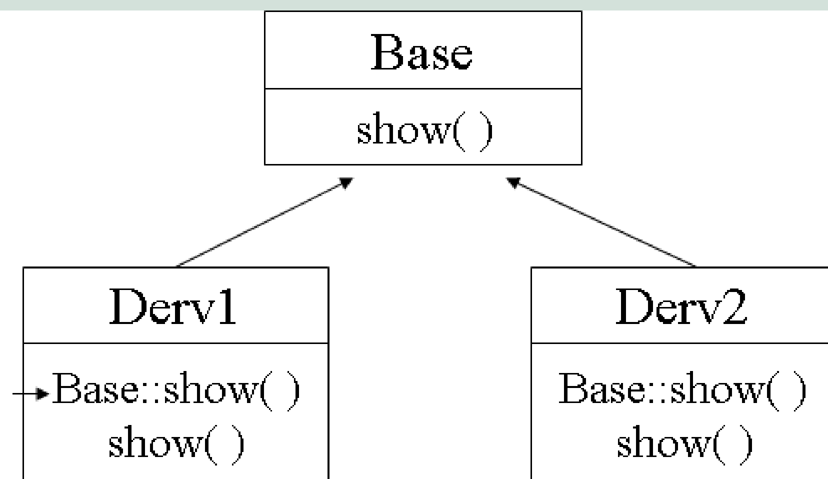
```
//eg1.cpp

Derv1 dv1;
Derv2 dv2;
dv1.show() ;
dv2.show() ;

Base* pBase;

pBase = &dv1;
pBase->show();

pBase = new Derv2();
pBase->show();
```



通过基类指针
只能访问从基类继承的成员

例　基类指针中调用虚函数

```
class Base
{   public:
      virtual   void show();
};

class Derv1: public Base
{  public:
      void show();
};

class Derv2: public Base
{  public:
      void show();
};
```

pBase

&Derv1

pBase->show()

pBase

new Derv2

pBase->show()

基类Base

virtual
show()

Derv1

Base::show()
show()

Derv2

Base::show()
show()

- **动态多态性**
  指程序在编译时并不能确定要调用的函数，直到运行时系统才能动态地确定操作所针对的具体对象，它又被称为**运行时多态。**
- 动态多态是通过**虚函数**（virtual function）来实现的。

6.3 虚函数

- C++中的虚函数的作用是允许在派生类中重新定义与基类同名的函数，并且可以**通过基类指针或者基类引用来访问这个同名函数**。虚函数成员声明的语法为：

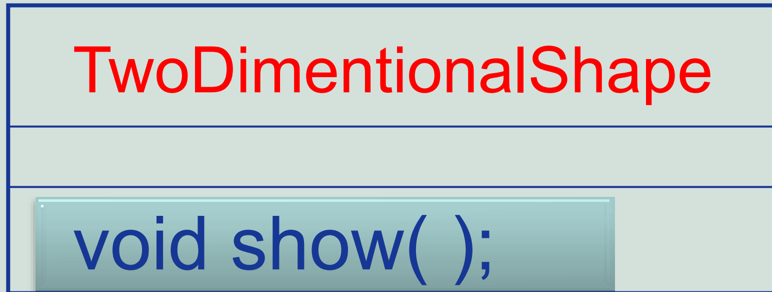
```
virtual 函数类型 函数名（参数列表）；
```

注意以下两点：

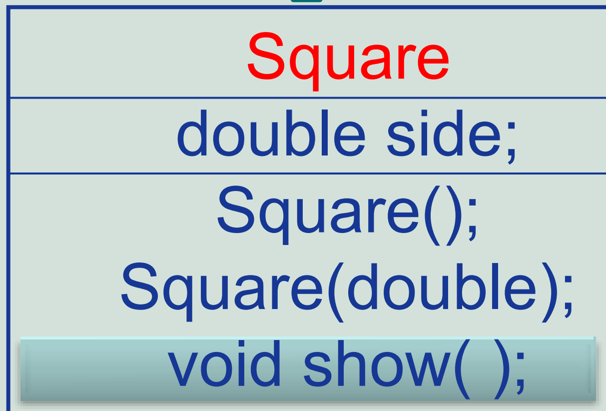
1. virtual只能使用在类定义的函数**原型声明**中，不能在成员函数实现的时候使用，也不能用来限定类外的普通函数。
2. virtual具有继承性，在派生类覆盖基类虚成员函数时，既可以使用virtual，也可以不用virtual来限定，二者没有差别，默认派生类中的重写函数是具有virtual的。

6.3 虚函数

例6.1



继承




```
// 二维图形输出函数
void TwoDimensionalShape::show(){
  cout<<"这是二维图形"<<endl;
}

// 正方形输出函数
void Square::show(){
 cout<<"这是边长为"<<side
        <<"的正方形" <<endl;
}
```

6.3 虚函数

```
TwoDimensionalShape t;              // 创建二维图形对象t
Square s(3);                        // 创建正方形对象s

 TwoDimensionalShape * members[2];  //二维图形指针数组
  members[0]=&t;
  members[1]=&s;

  for(int i=0;i<2;i++)
     members[i]->show();            //调用的是哪一个函数呢？
```

程序运行结果：
这是一个二维图形
这是一个二维图形
这是边长为3的正方形

无论指向的是哪种类型的对象，通过基类指针调用的都是基类中定义的show函数

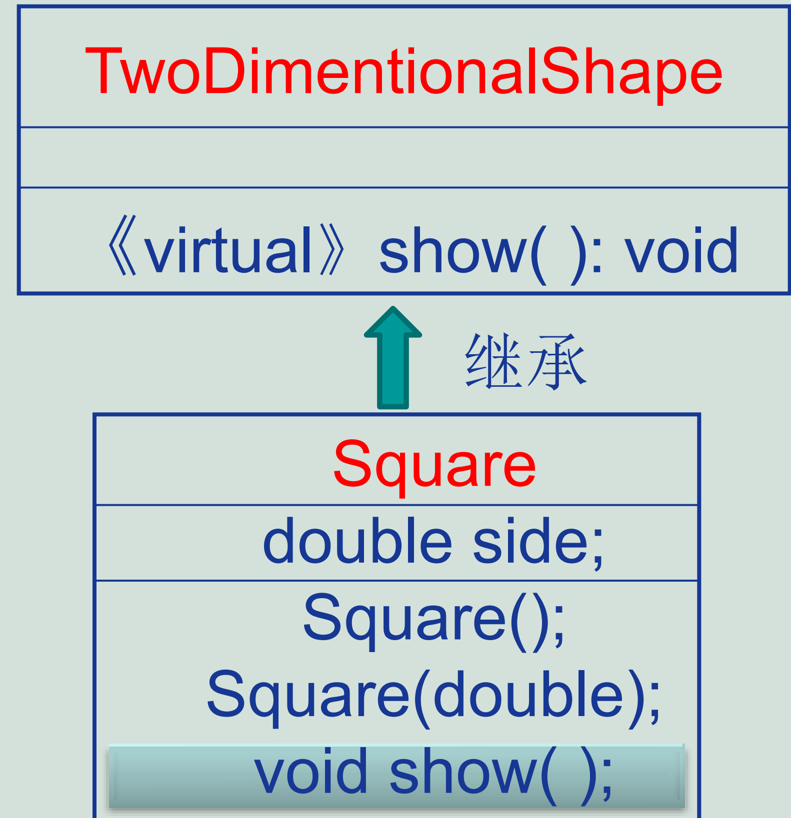
6.3 虚函数

```
class TwoDimensionalShape {            //二维图形类
public:
    virtual void show();
        ……
}
```

程序运行结果产生了变化：

这是一个二维图形

这是边长为3的正方形

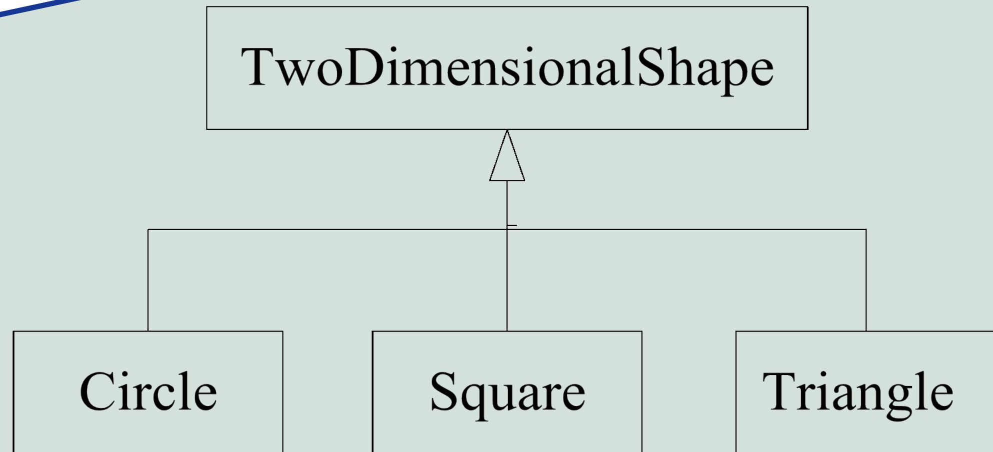


6.3 虚函数

```
TwoDimensionalShape * members[10];
  members[0]= new Circle();
  members[1]= new Square()
  members[2]= new Triangle(

  for(int i=0;i<3;i++)
      members[i]->show();
```

动态关联（dynamic binding）
在编译后执行该过程，
也被称为**滞后关联**



6.3 虚函数

使用虚函数的几点**说明**：

- 能用**virtual**声明类的非静态的成员函数，只能用于类的继承层次结构中。不能将类外的普通函数(友员)和静态成员函数声明成虚函数。

- 在派生类中重新定义虚函数，要求函数名、函数类型、函数参数个数和类型全部与基类的虚函数完全相同。否则不能实现多态性, 为函数重载.

```
class  base
{ public :
      virtual  void  vf1 ( ) ;
      virtual  void  vf2 ( ) ;
      virtual  void  vf3 ( ) ;
      void  f ( ) ;
 } ;
```

```
derived   d ;
  base  * bp = & d ;
  bp -> vf1 ( ) ;          // derived :: vf1 ( )
  bp -> vf2 ( ) ;          // base :: vf2 ( )
  bp -> f ( ) ;            // base :: f ( )
```

```
class  derived : public  base
{ public :
      void  vf1 ( ) ;                // 虚函数
      void  vf2 ( int ) ;          // 重载函数
      char  vf3 ( ) ;              // error，仅返回类型不同
      virtual void f ( ) ;
 } ;
```

使用虚函数

使用虚函数的几点**说明：**

- **一个成员函数被声明为虚函数后，在同一类族中的不能再定义与该虚函数具有相同的参数和返回类型的非virtual同名函数。**

- **派生类中可以根据需要重新定义虚函数，保持虚函数特性;如果没有对基类的虚函数重新定义，则派生类简单地继承其直接基类的虚函数。**

❖实现动态多态的条件

1. 声明虚函数

   公有继承类族中，顶层基类

2. 派生类重写虚函数

   派生类的函数覆盖基类的虚函数

   相同的函数名，参数列表，函数类型

3. 调用形式

   通过基类的指针或者引用调用虚函数。

假设创建一个派生类对象 Square s(5,6);

下面的哪些情况会发生动态绑定？

A
```
TwoDimensionalShape t =  s;
t.show();
```

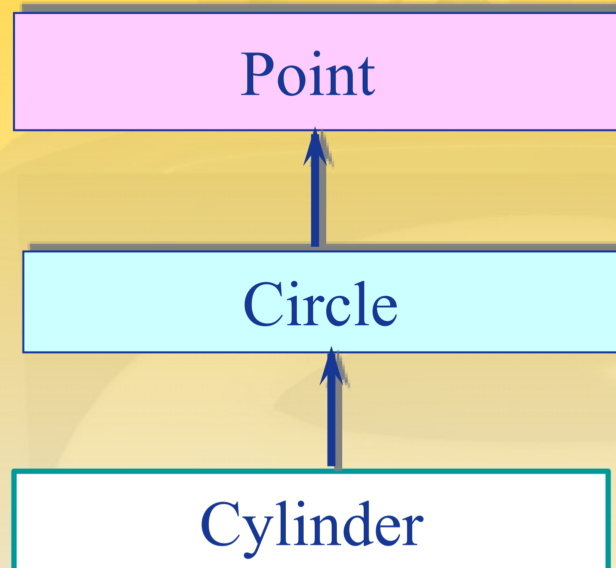
B
```
void test 1(TwoDimensionalShape & t){  t.show();  }
test(s);
```

C
```
void test 2(TwoDimensionalShape  t){  t.show();  }
test(s);
```

D
```
void test 3(TwoDimensionalShape *p)
{  p->show();  }
test( &s);
test( new Square(2,3));
```

◆课堂练习1

定义如下继承体系，能否使用一种容器，存储各种类对象？



6.4 虚析构函数

- **构造函数不能是虚函数**

  建立一个派生类对象时，必须从类层次的根开始，

  沿着继承路径逐个调用基类的构造函数

- **析构函数可以是虚函数**

  虚析构函数用于指引 delete 运算符正确析构动态对象

6.4 虚析构函数

❖虚析构函数

- 当基类的析构函数为虚函数时，无论指针指的是同一类族的哪一个类对象，对象撤销时，系统会采用动态关联，调用相应的析构函数，完成该对象的清理工作。

- 习惯把析构函数声明为虚函数，即使基类并不需要析构函数，以确保撤销动态存储空间时能够得到正确的处理。

eg2.cpp

例 普通析构函数在删除动态派生类对象的调用情况

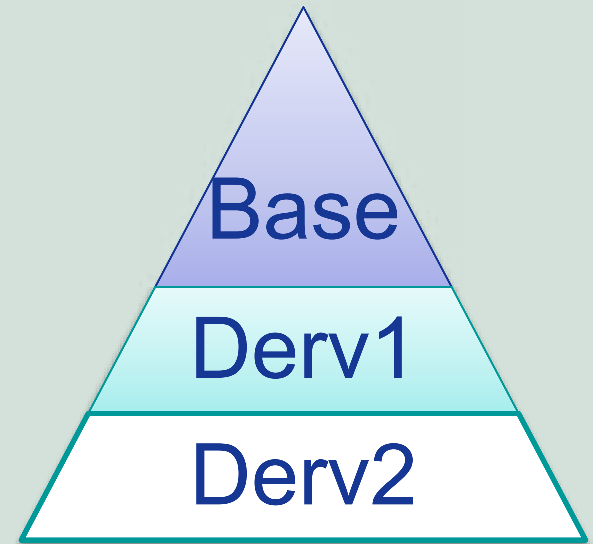
```
class Base
{ public:
        Base( ){ cout << "Base" << endl; }
        ~Base( ){ cout << "Desconstruct Base" << endl; }
};
class Derv1: public Base
{ public:
   Derv1( ){ cout << "Derv1" << endl; }
   ~Derv1( ){ cout << "Desconstruct Derv1" << endl; }
};
class Derv2: public Derv1
{  public:
        Derv2( ){ cout << "Derv2" << endl; }
        ~Derv2( ){ cout << "Desconstruct Derv2" << endl; }
};
```

例 普通析构函数在删除动态派生类对象的调用情况

```
Base* pBase =  new Base;
delete pBase;

 Derv1* pDerv1 =  new Derv1;
 delete pDerv1;

pBase = new Derv2;
delete pBase;
```



析构由基类指针建立的派生类对象

没有调用派生类析构函数

例 普通析构函数在删除动态派生类对象的调用情况

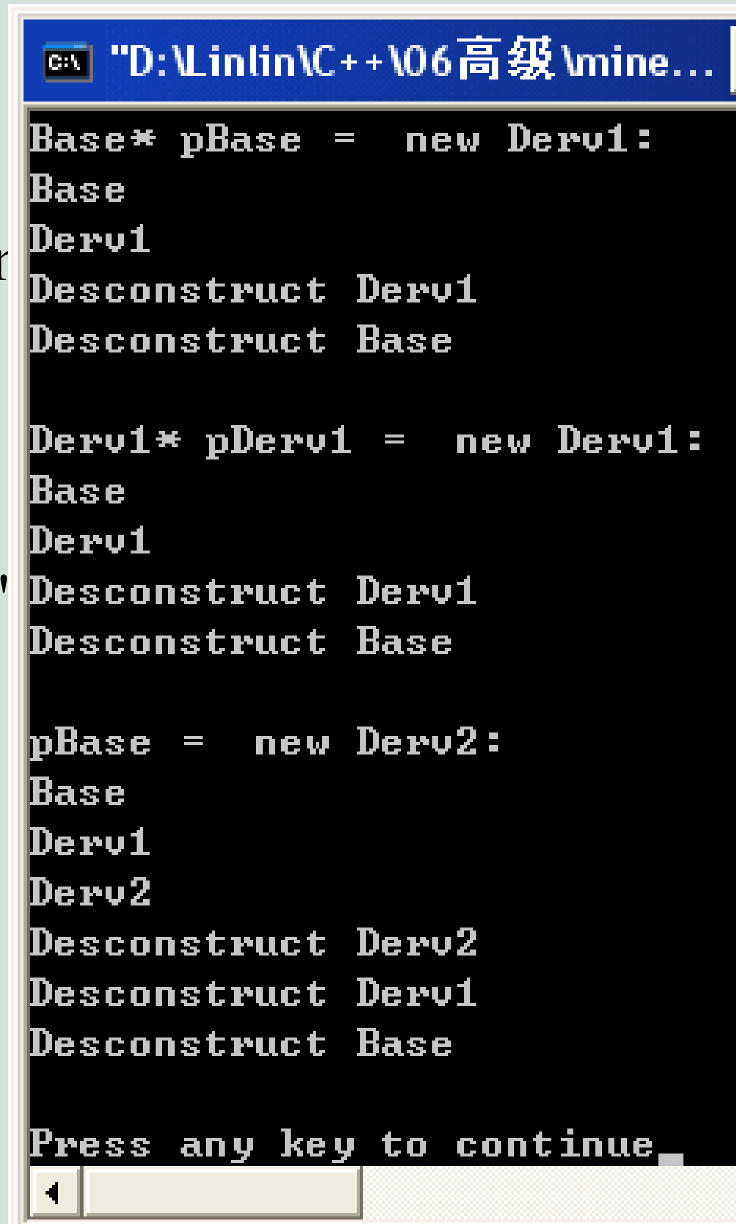
```
class Base
{ public:
        Base( ){ cout << "Base" << endl; }
        virtual ~Base( ){ cout << "Desconstruct Base" << endl; }
};
class Derv1: public Base
{ public:
   Derv1( ){ cout << "Derv1" << endl; }
   ~Derv1( ){ cout << "Desconstruct Derv1" << endl; }
};
class Derv2: public Derv1
{  public:
        Derv2( ){ cout << "Derv2" << endl; }
        ~Derv2( ){ cout << "Desconstruct Derv2" << endl; }
};
```

例 普通析构函数在删除动态派生类对象的调用情况

```
void  main( )
{
    cout<<"Base* pBase =  new Derv1:"<<er
    Base* pBase =  new Derv1;
    delete pBase;

    cout<<"\nDerv1* pDerv1 =  new Derv1:"
     Derv1* pDerv1 =  new Derv1;
    delete pDerv1;

    cout<<"\npBase =  new Derv2:"<<endl;
    pBase =  new Derv2;
    delete  pBase;
}
```



第6章　多态

1 理解多态

2 静态多态

3 动态多态

4 虚函数

5 纯虚函数与抽象类

6 多态案例

上节回顾

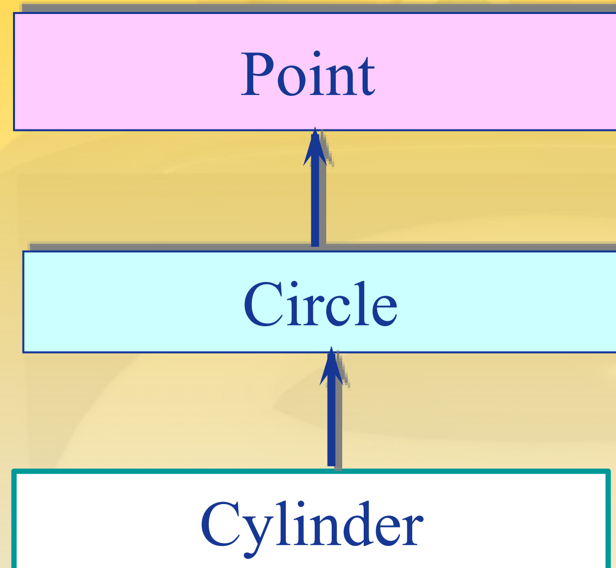
❖ 动态多态的实现

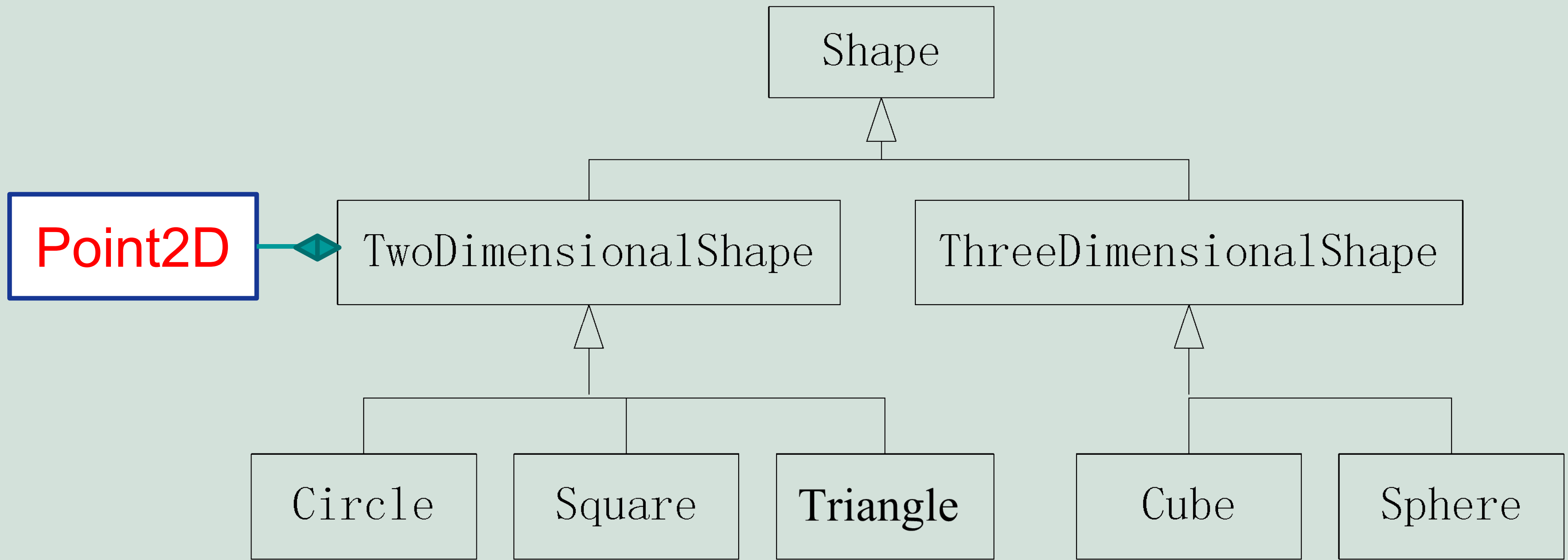
虚函数是在基类中冠以关键字 virtual 的非静态成员函数。

- 继承体系：判断成员函数所在的类是否会作为基类；
  虚函数为类族提供了一种公共接口。
- 重写函数：该函数在类被继承后有无可能被更改功能；
  允许在派生类中对基类的虚函数重新定义
- 调用形式：是否通过基类指针或引用调用该虚函数；
  赋值兼容性原则

◆课堂练习1

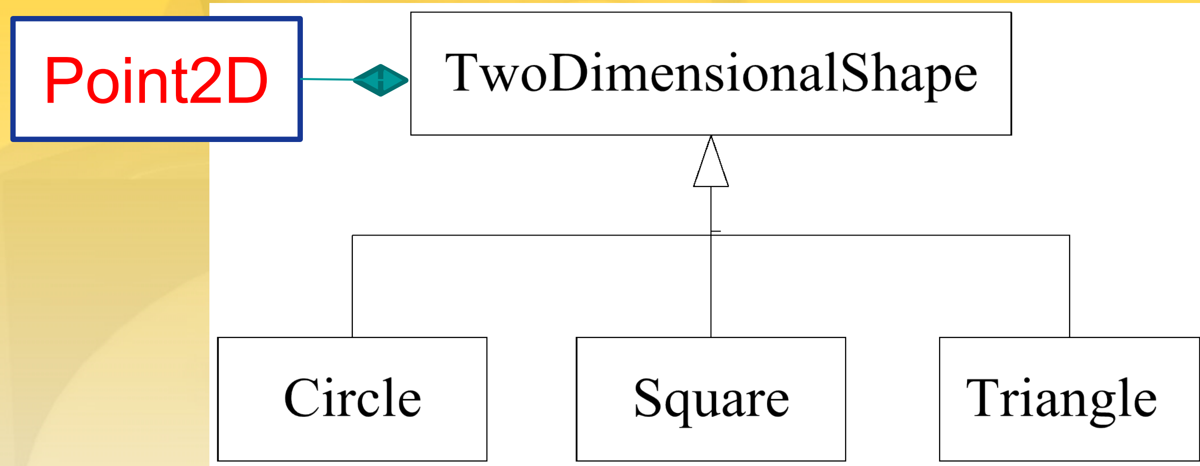
定义如下继承体系，能否使用一种容器，存储各种类对象？




Shape
Point2D
TwoDimensionalShape
ThreeDimensionalShape
Circle
Square
Triangle
Cube
Sphere

设计练习2：

使用虚函数实现动态多态。



6.5 纯虚函数与抽象类

TwoDimensionalShape

posistion : Point
name: String

+ show(): void
? ? ?

Square

~~double side;~~
~~void show();~~
void draw ();
double
getArea();

Circle

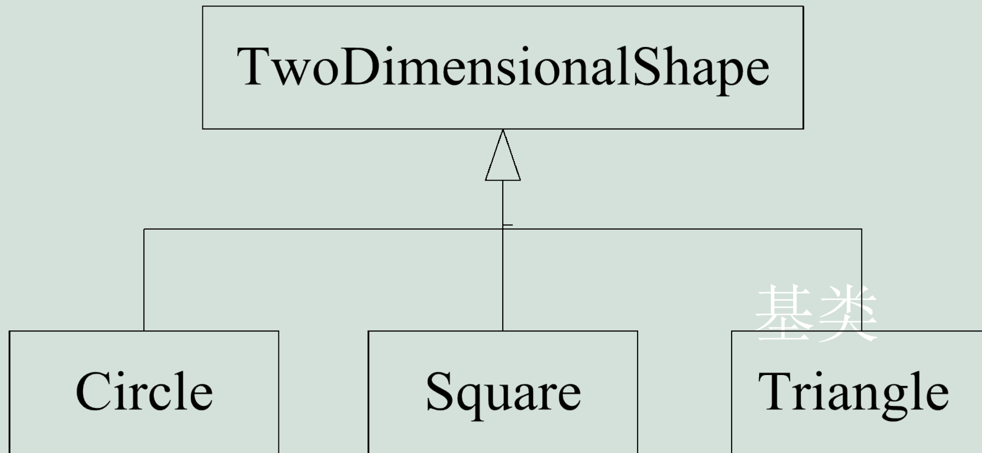
~~double side;~~
double PI;
~~void show();~~
void draw();
double
getArea();

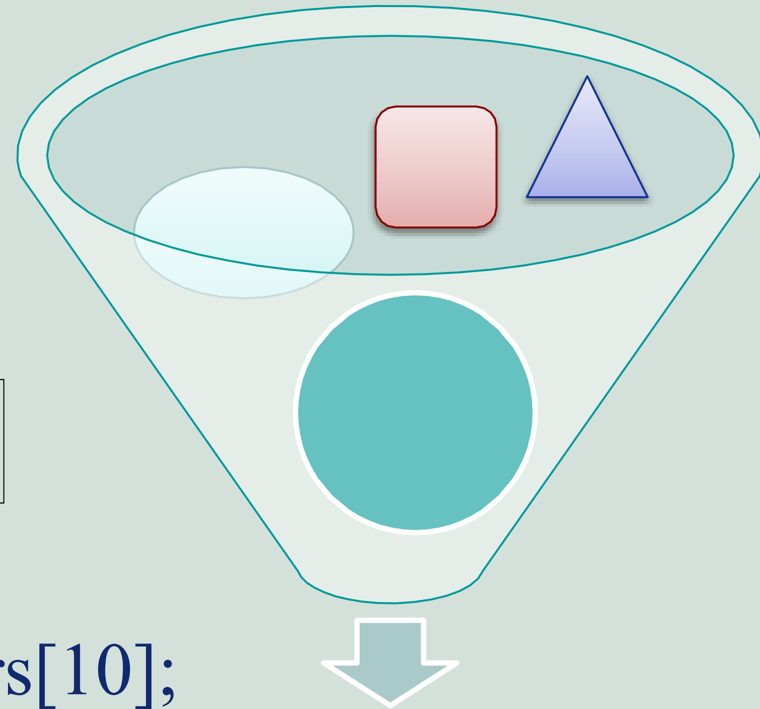
Triangle

double side1;
double side2;
double side3;

void show();
void draw();
double getArea();

6.5 纯虚函数与抽象类





图形容器

TwoDimensionalShape * members[10];

members[0]= new Circle(0,1,0.5);

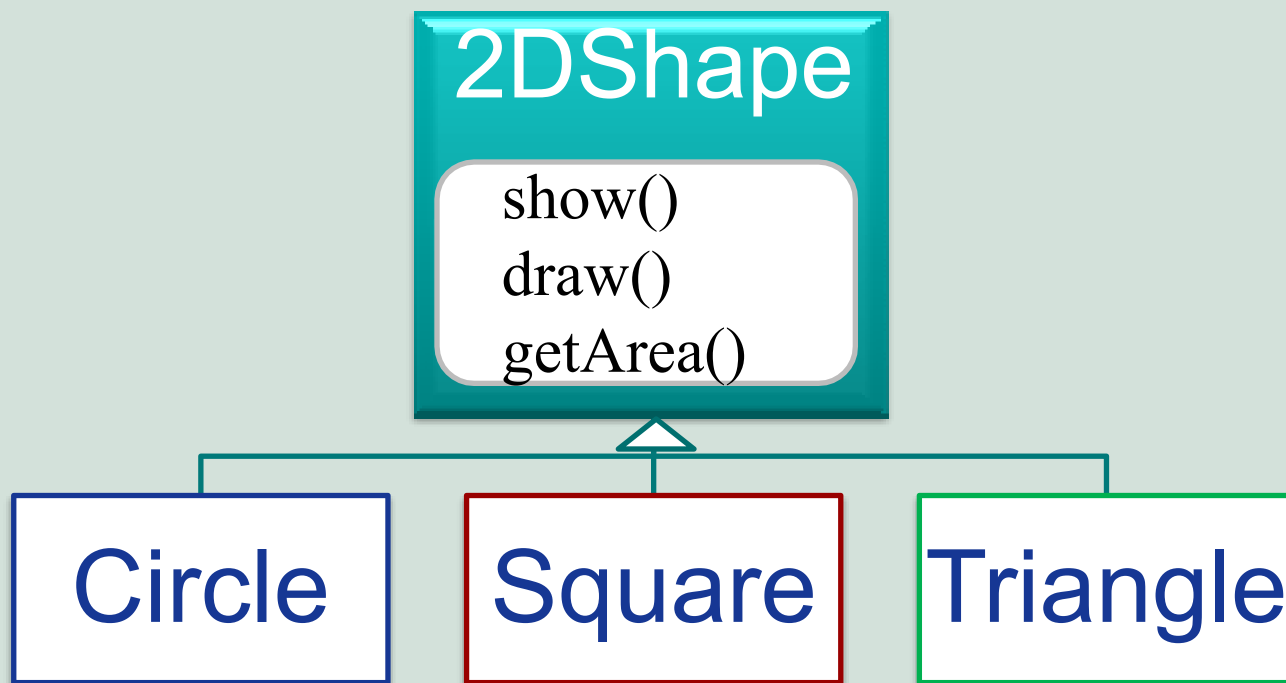
members[1]= new Square(1,1, 2.5);

members[2]= new Triangle(2,2,3,5,4);

。。。。。。

for(int i=0;i<10;i++)   members[i]->draw();

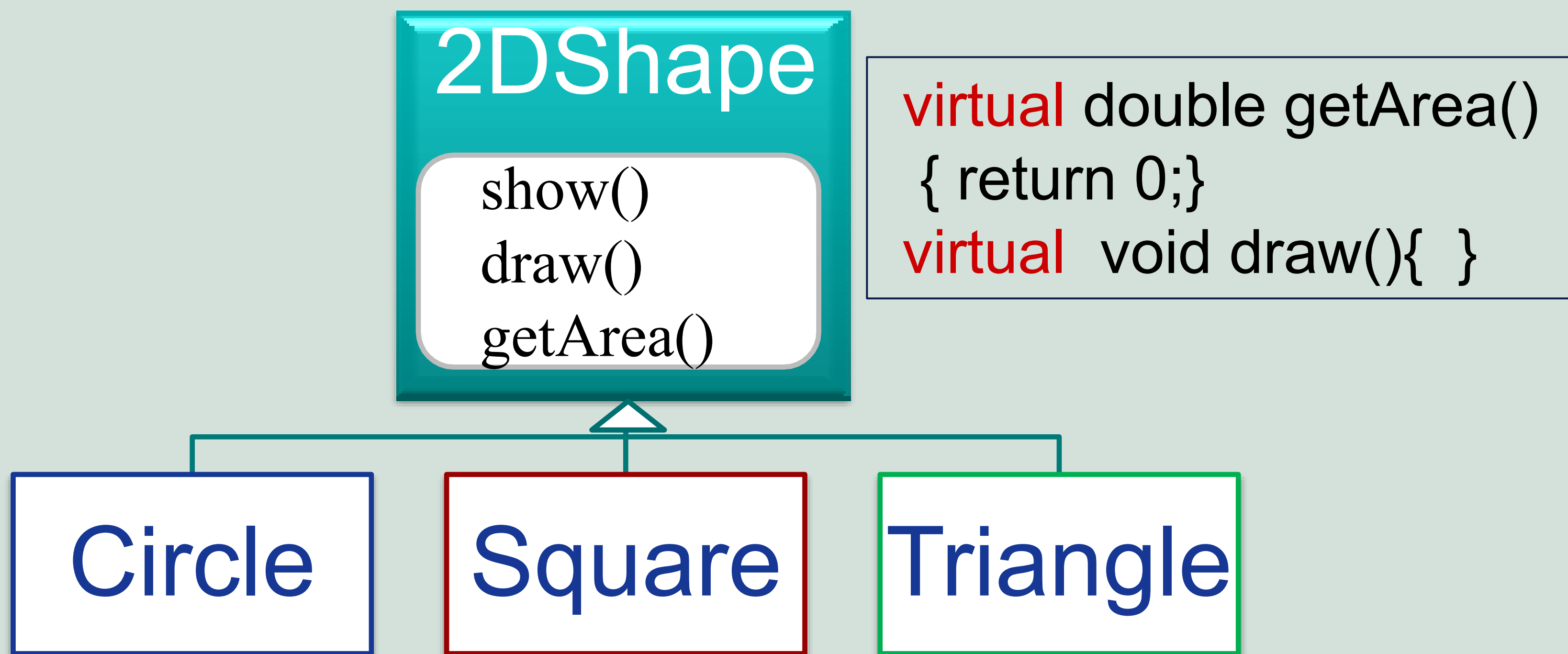
6.5 纯虚函数与抽象类




```
TwoDimensionalShape * members[10];
members[3] = new Shape2D(3,3,"2D");
for(int i=0;i<10;i++)      members[i]->draw();
```

如何画一个
二维图形对象？

6.5 纯虚函数与抽象类




```
void test( Shape2D & t ){
    t. show();                                  // 输出平面图形信息
    cout<<"面积为"<<t.getArea();      // 如何计算二维图形面积
}
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/796030004222010145