面向对象程序设计

组合与继承

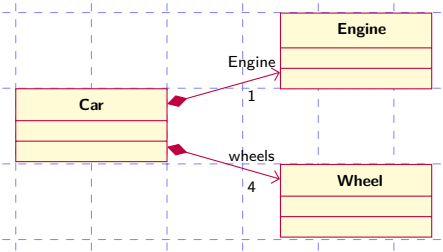
2020 年 春

耿楠

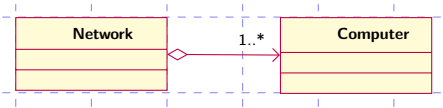
计算机科学系
信息工程学院

西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国 · 杨凌

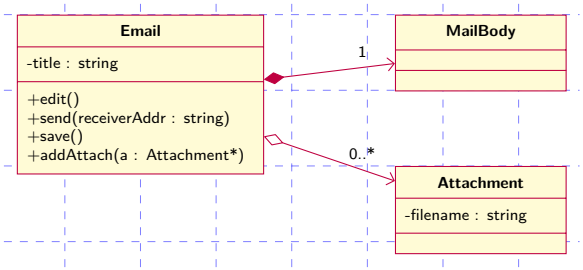
- 组合—composition(整体与部分的关系)



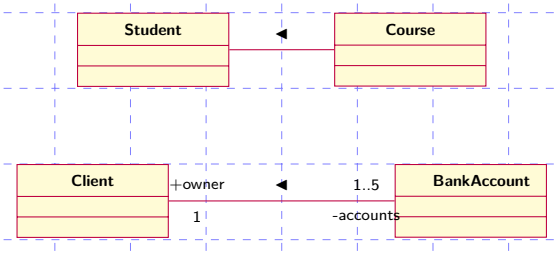
- 聚合—aggregation(松散关系)



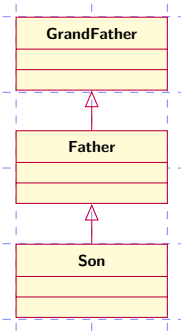
复合关系



- 拥有关系

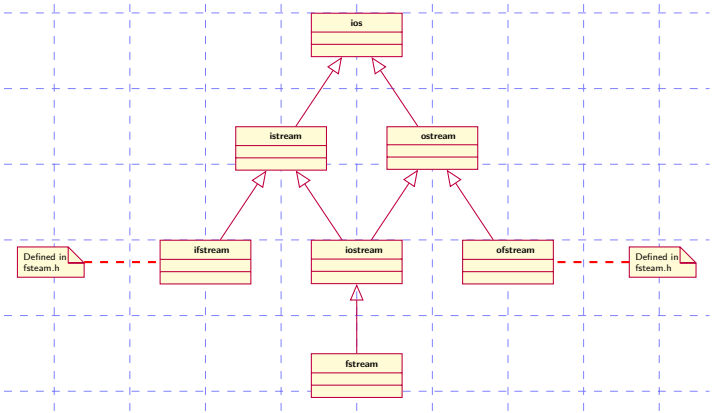


- 继承与派生类

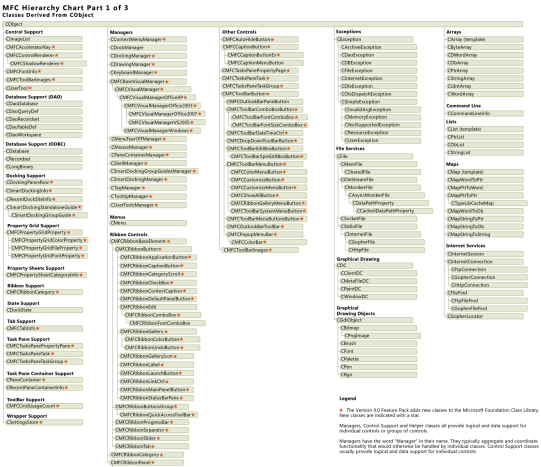


- 用已有类定义新类，新类拥有原有类的全部特征
  - 原有类 ⇒基类（父类）
  - 新类 ⇒派生类
- 可以多继承（一个派生类有多个基类）和多层派生（多层继承）
- 特点：新类可以继承原有类的属性和行为，并且可以 添加新的属性和行为，或更新原有类的成员
- 优点：代码重用

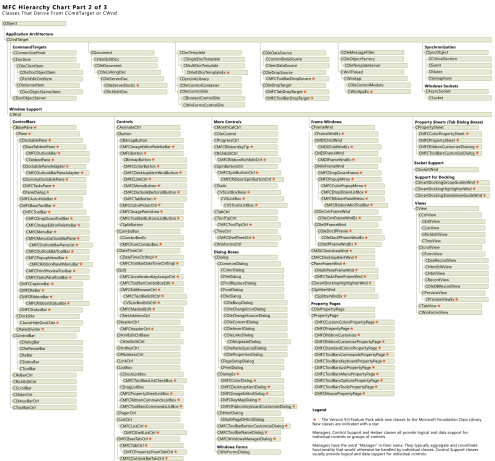
- 例子:C++ 输入输出流类


ios
istream
ostream
ifstream
iostream
ofstream
Defined in fsteam.h
Defined in fsteam.h
fstream

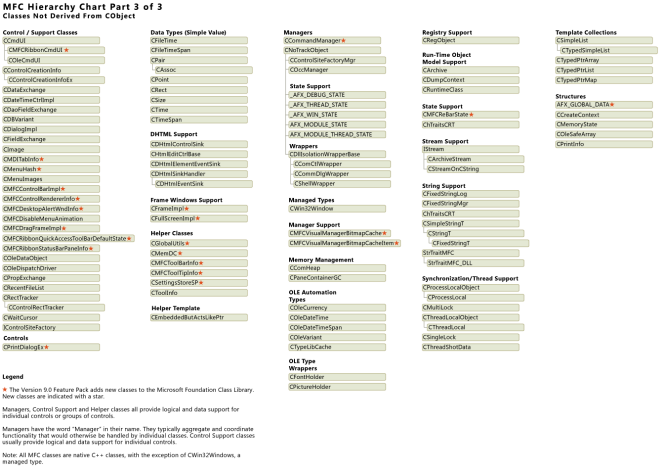
▶ 例子:MFC 类层次


MFC Hierarchy Chart Part 1 of 3
Classes Derived From CObject
CObject
Control Support
CImageList
CMFCAcceleratorKey
CMFCControlRenderer
CMFCShadowRenderer
CMFCPointsArray
CMFCToolBarImages
CUserTool
Database Support (DAO)
CDaoDatabase
CDaoQueryDef
CDaoRecordset
CDaoTableDef
CDaoWorkspace
Database Support (ODBC)
CDatabase
CRecordset
CLongBinary
Docking Support
CDockingPaneRow
CSmartDockingInfo
CRecentDockSiteInfo
CSmartDockingStandaloneGuide
CSmartDockingGroupGuide
Property Grid Support
CMFCPropertyGridProperty
CMFCPropertyGridColorProperty
CMFCPropertyGridFileProperty
CMFCPropertyGridFontProperty
Property Sheets Support
CMFCPropertySheetCategoryInfo
Ribbon Support
CMFCRibbonCategory
State Support
CDockState
Tab Support
CMFCTabInfo
Task Pane Support
CMFCTaskPanePropertyPane
CMFCTasksPaneTask
CMFCTasksPaneTaskGroup
Task Pane Container Support
CPaneContainer
CRecentPaneContainerInfo
Toolbar Support
CMFCToolBarImages
Wrapper Support
CSettingsStore
Managers
CContextMenuManager
CDockManager
CDockingManager
CDrawingManager
CImageManager
CMFCMenuImages
CMFCVisualManager
CMFCVisualManagerOffice2
CMFCVisualManagerOffice2003
CMFCVisualManagerOffice2007
CMFCVisualManagerVS2005
CMFCVisualManagerWindows
CMenuTearOffManager
CMouseManager
CPaneContainerManager
CPaneFrameWnd
CSmartDockingGroupGuideManager
CSmartDockingManager
CTagManager
CToolTipManager
CUserToolsManager
Menus
CMenu
Ribbon Controls
CMFCRibbonBaseElement
CMFCRibbonButtons
CMFCRibbonApplicationButton
CMFCRibbonCaptionButton
CMFCRibbonCategoryScroll
CMFCRibbonCheckBox
CMFCRibbonContextCaption
CMFCRibbonDefaultPanelButton
CMFCRibbonEdit
CMFCRibbonComboBox
CMFCRibbonFontComboBox
CMFCRibbonGallery
CMFCRibbonColorButton
CMFCRibbonUndoButton
CMFCRibbonGalleryIcon
CMFCRibbonLabel
CMFCRibbonLaunchButton
CMFCRibbonLinkCtrl
CMFCRibbonMainPanelButton
CMFCRibbonStatusBarPane
CMFCRibbonButtonsGroup
CMFCRibbonQuickAccessToolBar
CMFCRibbonProgressBar
CMFCRibbonSeparator
CMFCRibbonSlider
CMFCRibbonTab
CMFCRibbonCategory
CMFCRibbonPanel
CMFCRibbonMainPanel
Other Controls
CMFCAutoHideButtons
CMFCCaptionButton
CMFCCaptionButtonEx
CMFCCaptionMenuButton
CMFCTasksPanePropertyPage
CMFCTasksPaneTask
CMFCTasksPaneTaskGroup
CMFCToolBarButton
CMFCToolBarEditBoxButton
CMFCToolBarFontComboBox
CMFCToolBarFontSizeComboBox
CMFCToolBarMenuButton
CMFCToolBarSpinEditBoxButton
CMFCToolBarMenuButtons
CMFCColorMenuButton
CMFCColorsListBoxItem
CMFCCustomizeMenuButton
CMFCDrawMiButton
CMFCToolBarSystemMenuButton
CMFCToolBarMenuFontsButton
CMFCOutlookBarTearOffBar
CMFCShapeMenuBar
CMFCColorBar
CMFCToolBarImages
Exceptions
CException
CArchiveException
CDaoException
CDBException
CFileException
CInternetException
COleException
COleDispatchException
CSimpleException
CHtmlHelpException
CMemoryException
CNotSupportedException
CResourceException
CUserException
File Services
CFile
CMemFile
CSharedFile
CWrappedFile
CMonikerFile
CAsyncMonikerFile
CDataPathProperty
CCachedDataPathProperty
CSocketFile
CStdioFile
CInternetFile
CGopherFile
CHttpFile
Graphical Drawing
CDC
CClientDC
CMetaFileDC
CPaintDC
CWindowDC
Graphical Drawing Objects
CGdiObject
CBitmap
CPenBrush
CBrush
CFont
CPalette
CPen
CRgn
Arrays
CArray template
CByteArray
CDWordArray
CObArray
CPtrArray
CStringArray
CUintArray
CWordArray
Command Line
CCommandLineInfo
Lists
CList template
CPtrList
CObList
CStringList
Maps
CMap template
CMapWordToPtr
CMapPtrToWord
CMapPtrToPtr
CTypedCacheMap
CMapWordToOb
CMapStringToPtr
CMapStringToOb
CMapStringToString
Internet Services
CInternetSession
CInternetConnection
CFtpConnection
CGopherConnection
CHttpConnection
CFtpFind
CFtpFileFind
CGopherFileFind
CGopherLocator
Legend
The Version 9.0 feature Pack adds new classes to the Microsoft Foundation Class Library. New classes are indicated with a star.
Managers, Control Support and Helper classes all provide logical and data support for individual controls or groups of controls.
Managers have the word "Manager" in their name. They typically aggregate and coordinate functionality that would otherwise be handled by individual classes. Control Support classes usually provide logical and data support for individual controls.

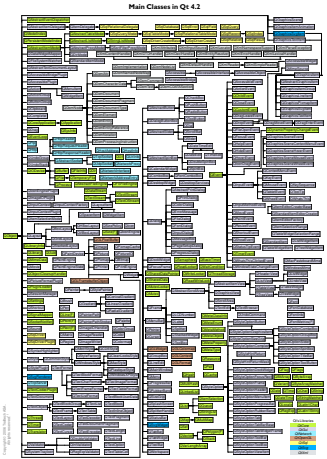
- 例子:MFC 类层次


MFC Hierarchy Chart Part 2 of 3
Classes That Derive From CCmdTarget or CWnd
CObject
Application Architecture
CCmdTarget
Window Support
CWnd
Controls
Micro Controls
Frame Windows
Property Sheets (Tab Dialog Boxes)
Dialog Boxes
Property Pages
Legend

▶ 例子:MFC 类层次


MFC Hierarchy Chart Part 3 of 3
Classes Not Derived From CObject
Control / Support Classes
CCmdUI
CMFCRibbonCmdUI *
COleCmdUI
CControlCreationInfo
CControlCreationInfoEx
CDataExchange
CDateTimeCtrlImpl
CDaoFieldExchange
CDBVariant
CDialogImpl
CFieldExchange
CImage
CMDITabInfo *
CMenuHash *
CMenuImages
CMFCControlBarImpl *
CMFCControlRendererInfo *
CMFCDesktopAlertWndInfo *
CMFCDisableMenuAnimation
CMFCDragFrameImpl *
CMFCRibbonQuickAccessToolBarDefaultState *
CMFCRibbonStatusBarPaneInfo *
COleDataObject
COleDispatchDriver
CProjExchange
CRecentFileList
CRectTracker
CControlRectTracker
CWaitCursor
CControlSiteFactory
Controls
CPrintDialogEx *
Data Types (Simple Value)
CFileTime
CFileTimeSpan
CPair
CAlsoc
CPoint
CRect
CSize
CTime
CTimeSpan
DHTML Support
CDHtmlControlSink
CHtmlEditCtrlBase
CDHtmlElementEventSink
CDHtmlSinkHandler
CDHtmlEventSink
Frame Windows Support
CFrameImpl *
CFullScreenImpl *
Helper Classes
CGlobalUtils *
CMemDC *
CMFCToolBarInfo *
CMFCToolTipInfo *
CSettingsStoreSP *
CToolInfo
Helper Template
CEmbeddedButActsLikePtr
Managers
CCommandManager *
CNetTrackObject
CControlSiteFactoryMgr
COccManager
State Support
_AFX_DEBUG_STATE
_AFX_THREAD_STATE
_AFX_WIN_STATE
AFX_MODULE_STATE
AFX_MODULE_THREAD_STATE
Wrappers
CDBIsolationWrapperBase
CComCtlWrapper
CCommDlgWrapper
CShellWrapper
Managed Types
CWin32Window
Manager Support
CMFCVisualManagerBitmapCache *
CMFCVisualManagerBitmapCacheItem *
Memory Management
CComHeap
CPaneContainerGC
OLE Automation Types
COleCurrency
COleDateTime
COleDateTimeSpan
COleVariant
CTypeLibCache
OLE Type Wrappers
CFontHolder
CPictureHolder
Registry Support
CRegObject
Run-Time Object Model Support
CArchive
CDumpContext
CRuntimeClass
State Support
CMFCReBarState *
CInTraitsCRT
Stream Support
IStream
CArchiveStream
CStreamOnCString
String Support
CFixedStringLog
CFixedStringMgr
CInTraitsCRT
CSimpleStringT
CStringT
CFixedStringT
StrTraitMFC
StrTraitMFC_DLL
Synchronization/Thread Support
CProcessLocalObject
CProcessLocal
CMutiLock
CThreadLocalObject
CThreadLocal
CSingleLock
CThreadSlotData
Template Collections
CSimpleList
CTypedSimpleList
CTypedPtrArray
CTypedPtrList
CTypedPtrMap
Structures
AFX_GLOBAL_DATA *
CCreateContext
CMemoryState
COleSafeArray
CPrintInfo
Legend
* The Version 9.0 Feature Pack adds new classes to the Microsoft Foundation Class Library. New classes are indicated with a star.
Managers, Control Support and Helper classes all provide logical and data support for individual controls or groups of controls.
Managers have the word "Manager" in their name. They typically aggregate and coordinate functionality that would otherwise be handled by individual classes. Control Support classes usually provide logical and data support for individual controls.
Note: All MFC classes are native C++ classes, with the exception of CWin32Windows, a managed type.

- 例子:QT 类层次



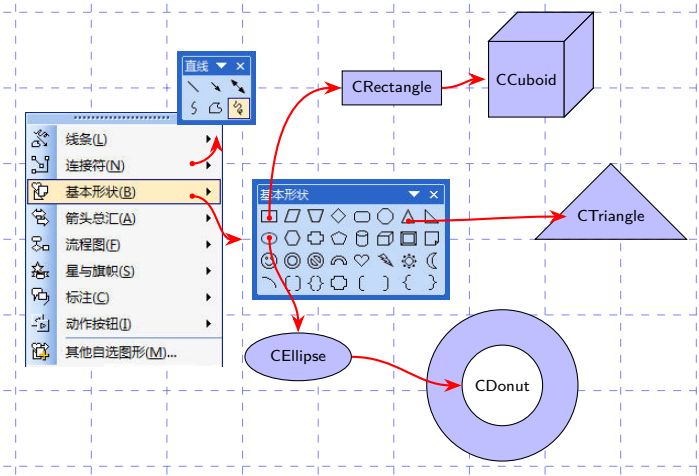
▶ 派生类的定义

```
class 派生类名: 继承方式1 基类名1,
      继承方式2 基类名2, ...
{
private:
  派生类的私有数据和函数;
public:
  派生类的公有数据和函数;
protected:
  派生类的保护数据和函数;
};
```

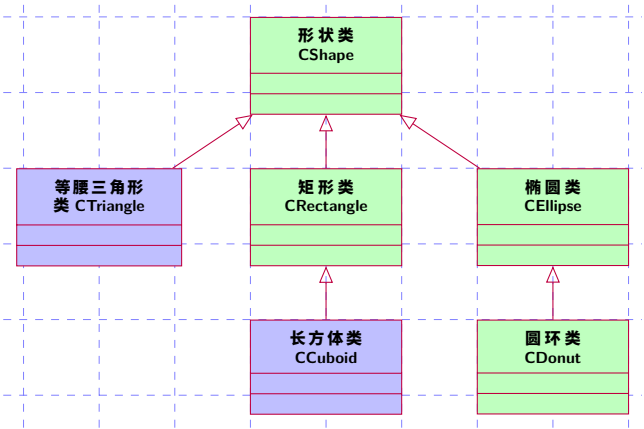
继承方式:

```
public:     公有继承
private:    私有继承
protected:  保护继承
```

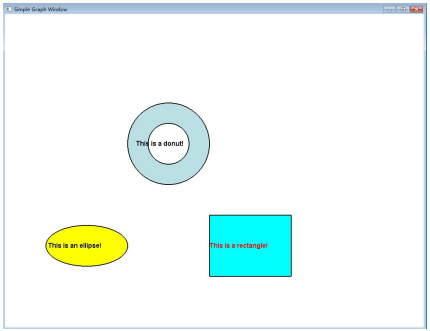
- 用继承方式实现基本形状类



- 基类：形状类（CShape）



- 用继承方式实现基本形状类
  - 功能描述
    - 带文本的基本形状绘制
    - 可更改文本和形状的颜色
    - 可更改形状的大小
    - 可上下左右移动形状



用继承方式实现基本形状类

```
// 例 05-01: ex05-01.cpp
//定义一个表示二维平面的点的类

class CPoint2D{
   float x, y;
public:
   CPoint2D(){
      x = y = 0;
   }
   CPoint2D(float x, float y){
      this->x = x;
      this->y = y;
   }
   void Translate(float x, float y);
   void Scale(float r);
   void Rotate(float angle);
   friend class CShape;
   friend class CRectangle;
   friend class CEllipse;
   friend class CDonut;
};
```

```
// 例 05-02: ex05-02.cpp

class CShape{
   ULONG textColor;
   char strText[256];
protected:
   CPoint2D wPos;
   ULONG objColor;
public:
   CShape();
   CShape(CPoint2D w, char *strText,
        ULONG objColor = 0xBBE0E3,
        ULONG textColor = 0): wPos(w);
   void Translate(float x, float y);
   void DrawText();
   void ShowPos();
};
```

文本颜色

文本内容

全局坐标

对象颜色

所有基本形状共用数据成员

用继承方式实现基本形状类

```
// 例 05-01: ex05-01.cpp
//定义一个表示二维平面的点的类

class CPoint2D{
   float x, y;
public:
   CPoint2D(){
      x = y = 0;
   }
   CPoint2D(float x, float y){
      this->x = x;
      this->y = y;
   }
   void Translate(float x, float y);
   void Scale(float r);
   void Rotate(float angle);
   friend class CShape;
   friend class CRectangle;
   friend class CEllipse;
   friend class CDonut;
};
```

友元类

```
// 例 05-02: ex05-02.cpp

class CShape{
   ULONG textColor;
   char strText[256];
protected:
   CPoint2D wPos;
   ULONG objColor;
public:
   CShape();
   CShape(CPoint2D w, char *strText,
        ULONG objColor = 0xBBE0E3,
        ULONG textColor = 0): wPos(w);
   void Translate(float x, float y);
   void DrawText();
   void ShowPos();
};
```

文本颜色
文本内容
全局坐标
对象颜色

所有基本形状共用数据成员

用继承方式实现基本形状类

```
// 例 05-01: ex05-01.cpp
//定义一个表示二维平面的点的类

class CPoint2D{
   float x, y;
public:
   CPoint2D(){
      x = y = 0;
   }
   CPoint2D(float x, float y){
      this->x = x;
      this->y = y;
   }
   void Translate(float x, float y);
   void Scale(float r);
   void Rotate(float angle);
   friend class CShape;
   friend class CRectangle;
   friend class CEllipse;
   friend class CDonut;
};
```

友元类

```
// 例 05-02: ex05-02.cpp

class CShape{
   ULONG textColor;
   char strText[256];
protected:
   CPoint2D wPos;
   ULONG objColor;
public:
   CShape();
   CShape(CPoint2D w, char *strText,
        ULONG objColor = 0xBBE0E3,
        ULONG textColor = 0): wPos(w);
   void Translate(float x, float y);
   void DrawText();
   void ShowPos();
};
```

所有基本形状共用函数成员

平移操作

文本显示

位置输出

- 用继承方式实现基本形状类–矩形类

```
// 例 05-03: ex05-03.cpp
//定义类 CRectangle，该类继承 CShape

#include "Shape.h"
class CRectangle: public CShape{
   CPoint2D lv1, lv2, lv3, lv4;
public:
   CRectangle(): lv1(CPoint2D(-50, -30)), lv2(CPoint2D(50, -30)),
      lv3(CPoint2D(50, 30)), lv4(CPoint2D(-50, 30)) {}
   CRectangle(float length, float width, CPoint2D w, char *strText,
          ULONG objColor = 0xBBE0E3,
          ULONG textColor = 0):
          CShape(w, strText, objColor, textColor){
      lv1.x = lv4.x = -0.5 * length;
      lv1.y = lv2.y = -0.5 * width;
      lv2.x = lv3.x = 0.5 * length;
      lv3.y = lv4.y = 0.5 * width;
   }
   void Draw();
   void ShowPos();
};
```

基类

初始化列表初始化基类数据

用继承方式实现基本形状类–椭圆类

```
// 例 05-04: ex05-04.cpp
//定义类 CEllipse，该类继承 CShape

#include "Shape.h"
class CEllipse: public CShape{
protected:
   float x_radius, y_radius;
public:
   CEllipse(){
      x_radius = y_radius = 50;
   }
   CEllipse(float rx, float ry, CPoint2D w, char *strText,
         ULONG objColor = 0xBBE0E3,
         ULONG textColor = 0):
      CShape(w, strText, objColor, textColor){
      x_radius = rx;
      y_radius = ry;
   }
   void Draw();
   void ShowPos();
};
```

基类

初始化列表初始化基类数据

用继承方式实现基本形状类–圆环类

```
// 例 05-05: ex05-05.cpp
//定义类 CDonut，该类继承 CEllipse

#include "Ellipse.h"
class CDonut: public CEllipse
   float ratio;
public:
   CDonut()
   {
      ratio = 0.5;
   }
   CDonut( float r, float rx, float ry, CPoint2D w, char *strText,
         ULONG objColor = 0xBBE0E3,
         ULONG textColor = 0):
      CEllipse(rx, ry, w, strText, objColor, textColor)
      ratio = r;
   }
   void Draw();
   void ShowPos();
};
```

基类

初始化列表初始化基类数据

吸收基类成员

```
// 例 05-06: ex05-06.cpp
// 定义矩形的 Draw 方法

void CRectangle::Draw()
{
   setColor(CShape::objColor);
   fillRectangle(lv1.x + CShape::wPos.x, lv1.y + CShape::wPos.y,
            lv3.x + CShape::wPos.x, lv3.y + CShape::wPos.y);
   setColor(0x000000);

   setLineWidth(2);
   rectangle(lv1.x + CShape::wPos.x, lv1.y + CShape::wPos.y,
          lv3.x + CShape::wPos.x, lv3.y + CShape::wPos.y);
   setLineWidth(1);

   CShape::DrawText() ;
}
```

吸收数据成员

吸收成员函数

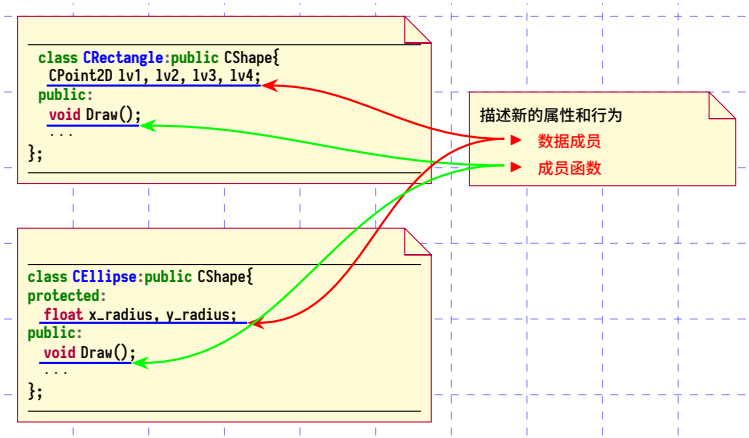
改造基类成员

```
// 例 05-07: ex05-07.cpp
// 成员函数实现

void CShape::ShowPos()
{
   cout << strText << endl;
   cout << "CShape: (" << wPos.x << ","
        << wPos.y << ")" << endl;
}

void CRectangle::ShowPos()
{
   CShape::ShowPos();
   cout << "CRectangle: (" << lv1.x << ","
      << lv1.y << "), (" << lv3.x << ","
      << lv3.y << ")" << endl;
}
```

```
CRectangle myRect;
myRect.ShowPos();
```

同名覆盖:
当通过派生类对象调用 ShowPos() 时，将自动调用成员函数CRectangle::ShowPos()

- 添加新成员


class CRectangle:public CShape{
CPoint2D lv1, lv2, lv3, lv4;
public:
void Draw();
...
};
描述新的属性和行为
数据成员
成员函数
class CEllipse:public CShape{
protected:
float x_radius, y_radius;
public:
void Draw();
...
};

- 继承关系是可以传递的
  - 如类 A 派生出类 B，类 B 又派生出类 C，则类 B 是类 C 的直接基类，类 A 是类 B 的直接基类，而类 A 称为类 C 的间接基类
- 继承关系不允许循环
  - 在派生中，不允许类 A 派生出类 B，类 B 又派生出类 C，而类 C 又派生出类 A

- 公有继承 (public)
  - 基类的公有成员在派生类中仍然为公有成员，可以由派生类对象和派生类成员函数 直接访问
  - 基类的私有成员在派生类中，无论是派生类的成员还是派生类的对象都 无法直接访问
  - 保护成员在派生类中仍是保护成员，可以通过派生类的成员函数访问，但 不能由派生类的对象直接访问

▶ 实例

```
class CShape
{
 ULONG textColor;
 char strText[256];
protected:
 CPoint2D wPos;
public:
 ULONG objColor;
};
```

```
class CEllipse:public CShape
{
 float x_radius, y_radius;
public:
 void Draw(){
   ULONG color1 = CShape::textColor; ✘
   CPoint2D pos = CShape::wPos; ✔
   ULONG color2 = CShape::objColor; ✔
 }
};
```

This is an ellipse!

```
CEllipse myEllip;
ULONG color1 = myEllip.textColor; ✘
CPoint2D pos = myEllip.wPos; ✘
ULONG color2 = myEllip.objColor; ✔
```

- 私有继承 (Private)
  - 基类的公有成员和保护成员被继承后成为派生类的私有成员
  - 基类的私有成员在派生类中不能被直接访问
  - 经过私有继承，所有基类的成员都成为了派生类的私有成员，如进一步派生，基类的全部成员将无法在新的派生类中被访问

实例

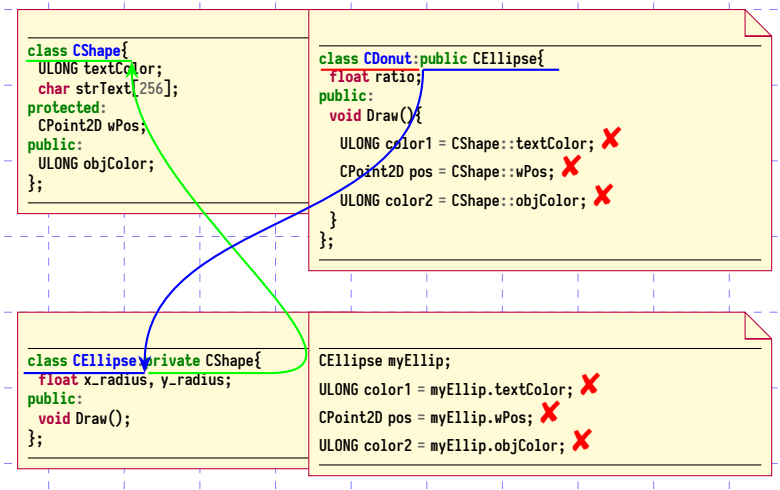
```
class CShape
{
 ULONG textColor;
 char strText[256];
protected:
 CPoint2D wPos;
public:
 ULONG objColor;
};
```

```
class CEllipse:private CShape
{
 float x_radius, y_radius;
public:
 void Draw(){
   ULONG color1 = CShape::textColor; ✘
   CPoint2D pos = CShape::wPos; ✔
   ULONG color2 = CShape::objColor; ✔
 }
};
```

This is an ellipse!

```
CEllipse myEllip;
ULONG color1 = myEllip.textColor; ✘
CPoint2D pos = myEllip.wPos; ✘
ULONG color2 = myEllip.objColor; ✘
```

- 实例


class CShape{
ULONG textColor;
char strText[256];
protected:
CPoint2D wPos;
public:
ULONG objColor;
};
class CDonut:public CEllipse{
float ratio;
public:
void Draw(){
ULONG color1 = CShape::textColor;
CPoint2D pos = CShape::wPos;
ULONG color2 = CShape::objColor;
}
};
class CEllipse:private CShape{
float x_radius, y_radius;
public:
void Draw();
};
CEllipse myEllip;
ULONG color1 = myEllip.textColor;
CPoint2D pos = myEllip.wPos;
ULONG color2 = myEllip.objColor;

- 保护继承 (Protected)
  - 基类的公有成员和保护成员被继承后作为派生类的保护成员
  - 基类的私有成员在派生类中不能被直接访问
  - 如果将派生类作为新的基类继续派生时，基类成员可以沿继承树继续传播

- 实例

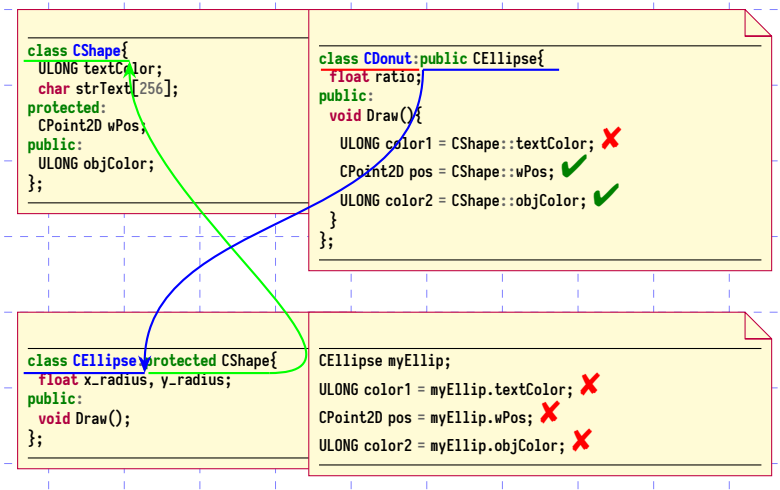
```
class CShape
{
 ULONG textColor;
 char strText[256];
protected:
 CPoint2D wPos;
public:
 ULONG objColor;
};
```

```
class CEllipse:protected CShape
{
 float x_radius, y_radius;
public:
 void Draw(){
   ULONG color1 = CShape::textColor; ✘
   CPoint2D pos = CShape::wPos; ✔
   ULONG color2 = CShape::objColor; ✔
 }
};
```

This is an ellipse!

```
CEllipse myEllip;
ULONG color1 = myEllip.textColor; ✘
CPoint2D pos = myEllip.wPos; ✘
ULONG color2 = myEllip.objColor; ✘
```

- 实例


class CShape{
ULONG textColor;
char strText[256];
protected:
CPoint2D wPos;
public:
ULONG objColor;
};
class CDonut:public CEllipse{
float ratio;
public:
void Draw(){
ULONG color1 = CShape::textColor;
CPoint2D pos = CShape::wPos;
ULONG color2 = CShape::objColor;
}
};
class CEllipse:protected CShape{
float x_radius, y_radius;
public:
void Draw();
};
CEllipse myEllip;
ULONG color1 = myEllip.textColor;
CPoint2D pos = myEllip.wPos;
ULONG color2 = myEllip.objColor;

- 基类成员在派生类中的访问控制属性

基类属性 \ 继承方式	public	protected	private
public	public	protected	不可访问
protected	protected	protected	不可访问
private	private	private	不可访问

派生类构造函数的定义

```
派生类名（参数总表):基类名1(参数表1), ..., 基类名 m(参数表 m),
                成员对象名1(参数表1), ..., 成员对象名 n(参数表 n)
{
    派生类新增成员的初始化;
}
```

初始化列表

形状类派生椭圆类

```
class CEllipse:private CShape
{
  float x_radius, y_radius;
public:
  CEllipse() {x_radius = y_radius = 50;}
  CEllipse(float rx, float ry, CPoint2D w, char *strText, ULONG objColor=0xBBE0E3,
           ULONG textColor=0):CShape(w, strText,objColor, textColor)
  {
     x_radius = rx;
     y_radius = ry;
  }
  void Draw();
  void ShowPos();
};
```

初始化列表

椭圆类派生圆环类

```
class CDonut:public CEllipse
{
  float ratio;
public:
  CDonut() {ratio = 0.5;}
  CDonut(float r, float rx, float ry, CPoint2D w, char *strText,
         ULONG objColor=0xBBE0E3,
         ULONG textColor=0):CEllipse(rx, ry, w, strText, objColor, textColor)
  {
    ratio = r;
  }
  void Draw();
  void ShowPos();
};
```

初始化列表

单继承的构造与析构

- 首先调用基类成员类构造函数
- 然后调用基类构造函数
- 再调用派生类成员类的构造函数
- 最后调用派生类构造函数
- 当派生类对象析构时，各析构函数调用顺序正好相反

单继承的构造与析构

```
// 例 05-08-01: ex05-08-01.cpp
// 构造函数和析构函数的演示


#include <iostream>

using namespace std;
class memObj
{
   int a;
public:
   memObj(int x)
   {
      a = x;
      cout << "Constructing member object " << a << endl;
   }
   ~memObj()
   {
      cout << "Destructing member object" << a << endl;
   }
};
```

单继承的构造与析构

```
// 例 05-08-02: ex05-08-02.cpp
// 定义一个简单的类，包含一个属性，
// 构造函数和析构函数

class base
{
   memObj obj1;
public:
   //该构造函数有成员初始化列表
   base():obj1(1)
   {
      cout << "Constructing base\n";
   }
   ~base()
   {
      cout << "Destructing base\n";
   }
};
```

```
// 例 05-08-03: ex05-08-03.cpp
//定义类 derived，该类继承 base 类

class derived: public base
{
   memObj obj2;
public:
   derived():obj2(2)
   {
      cout << "Constructing derived\n";
   }
   ~derived()
   {
      cout << "Destructing derived\n";
   }
};
```

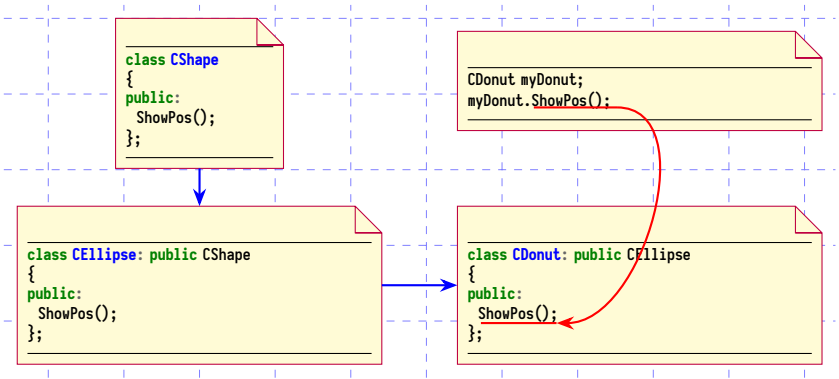
▶ 单继承的构造与析构

```
// 例 05-08-04: ex05-08-04.cpp

int main()
{
   derived ob;
   return 0;
}
```

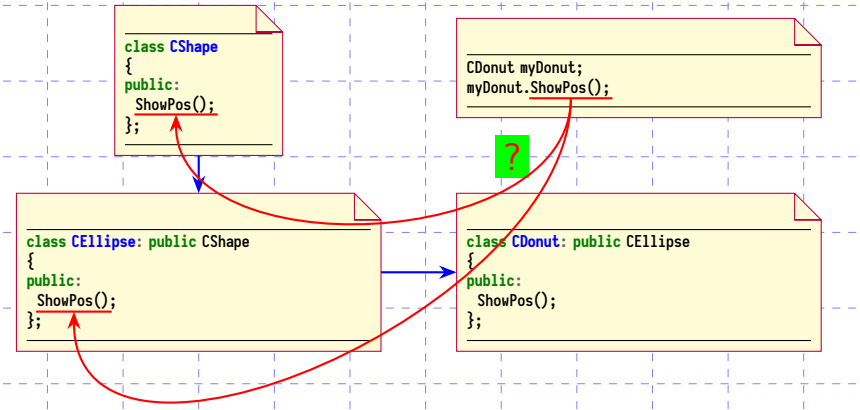
```
testcpp
Constructing member object 1
Constructing base
Constructing member object 2
Constructing derived
Destructing derived
Destructing member object2
Destructing base
Destructing member object1

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

- **类型兼容**：在公有派生的情况下，一个派生类对象可作为基类的对象来使用
  - 派生类对象可以赋值给基类对象
  - 派生类对象可以初始化基类的引用
  - 派生类对象的地址可赋给指向基类的指针

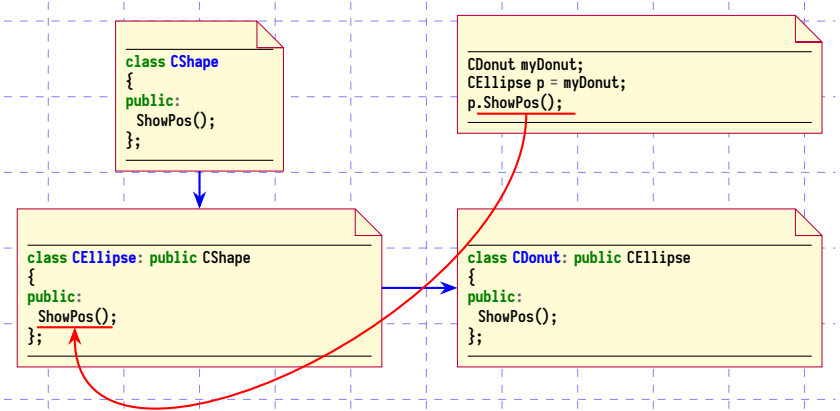
- 如何通过派生类对象调用基类中被覆盖的成员函数



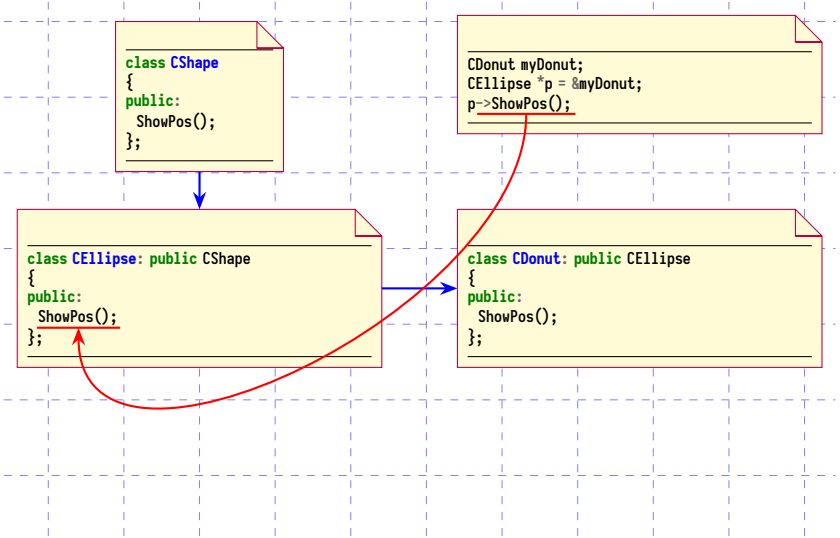
- 如何通过派生类对象调用基类中被覆盖的成员函数



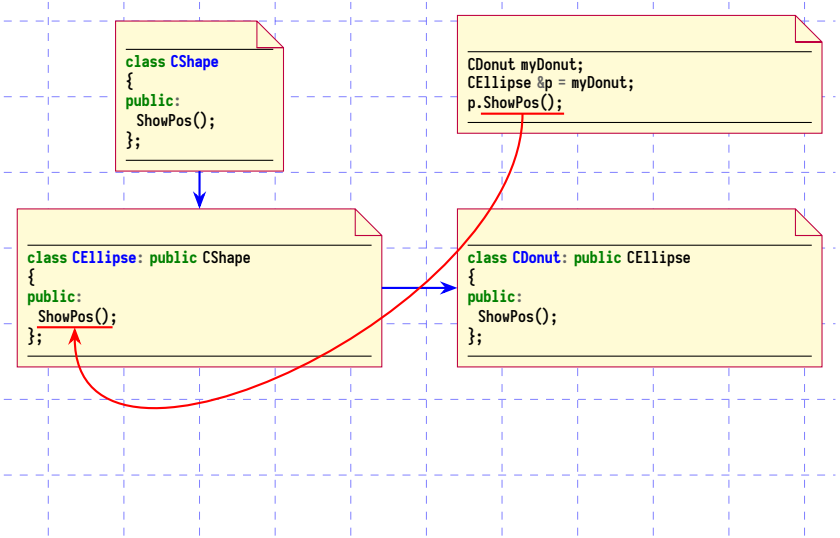
- 如何通过派生类对象调用基类中被覆盖的成员函数



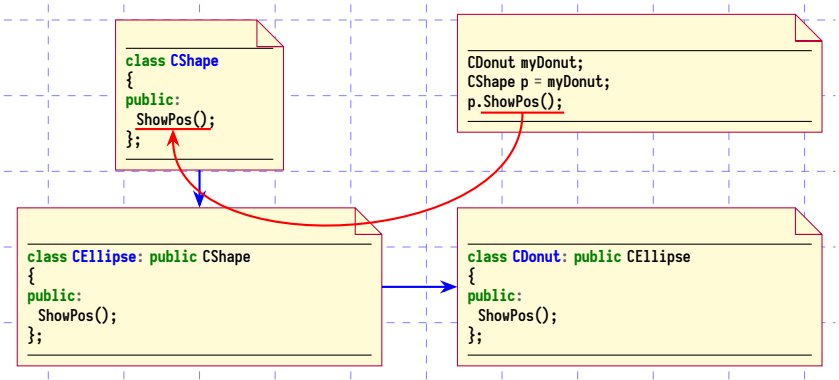
- 如何通过派生类对象调用基类中被覆盖的成员函数



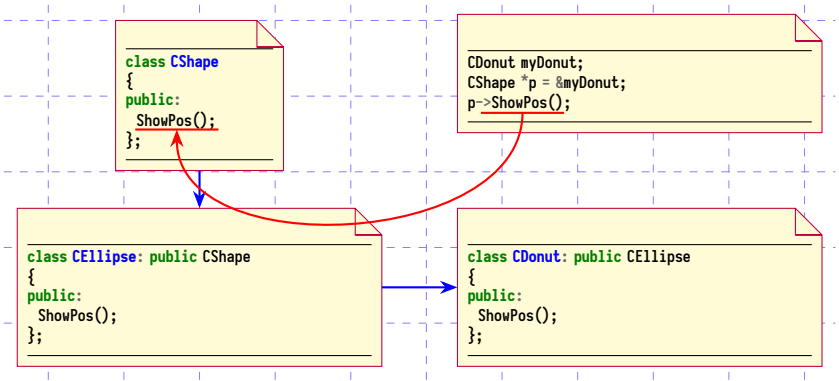
- 如何通过派生类对象调用基类中被覆盖的成员函数



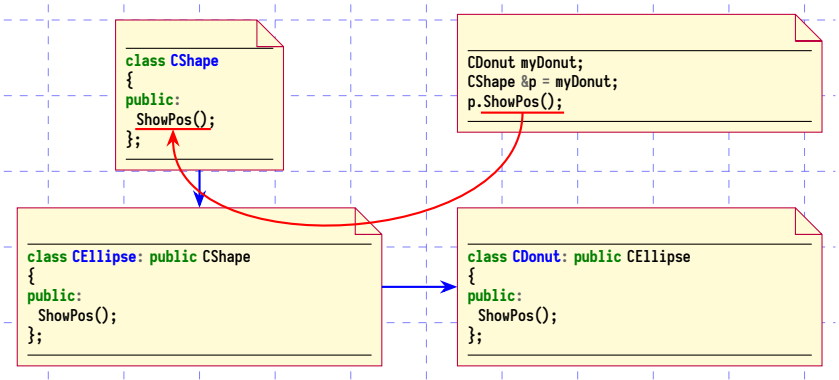
- 如何通过派生类对象调用基类中被覆盖的成员函数



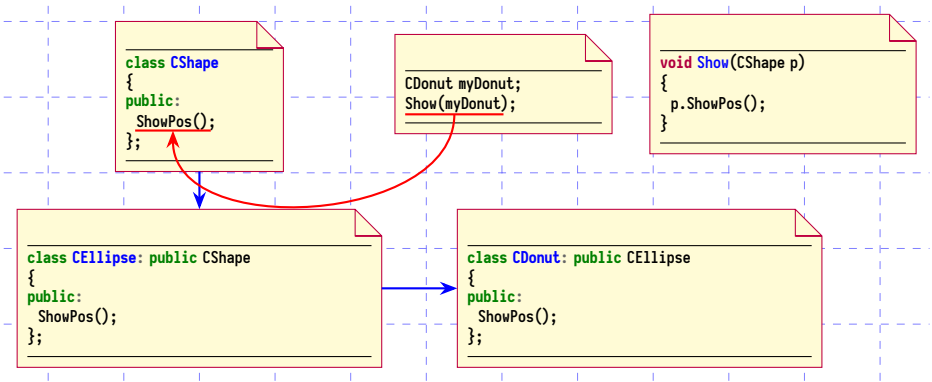
- 如何通过派生类对象调用基类中被覆盖的成员函数



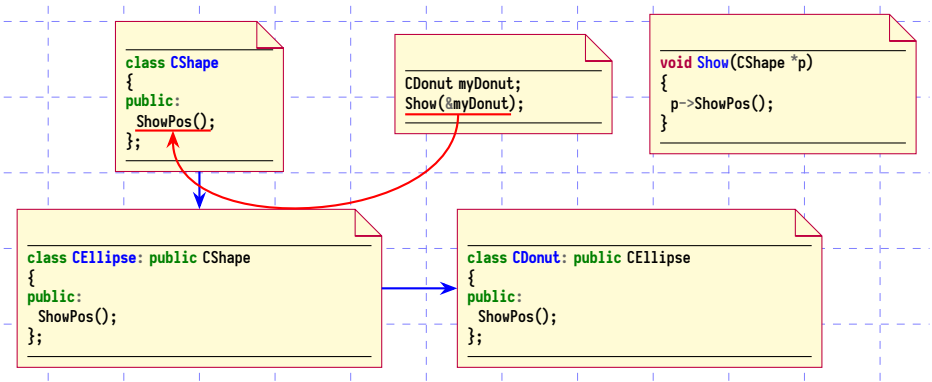
- 如何通过派生类对象调用基类中被覆盖的成员函数



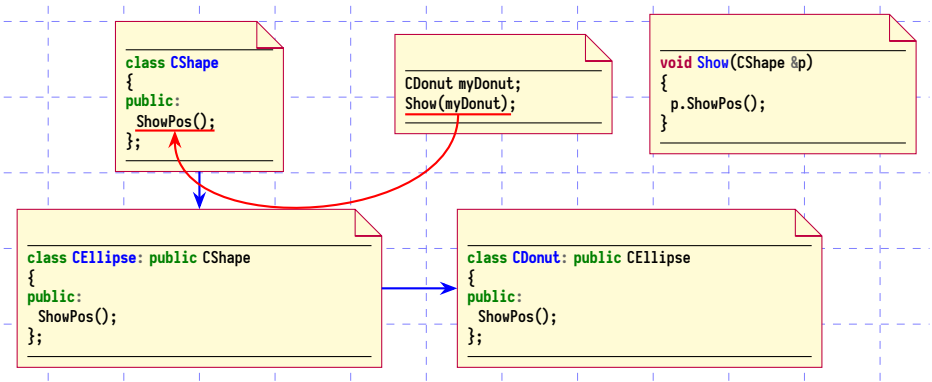
- 如何通过派生类对象调用基类中被覆盖的成员函数



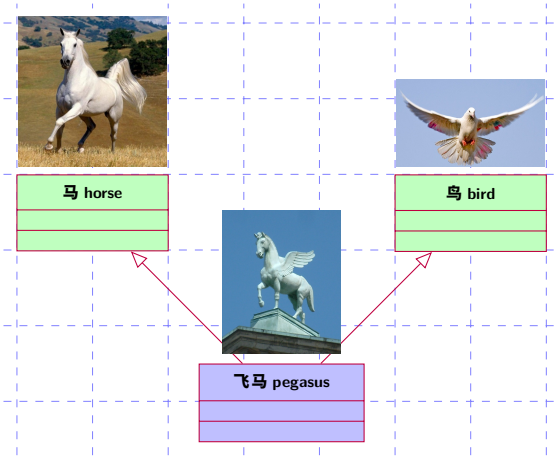
- 如何通过派生类对象调用基类中被覆盖的成员函数



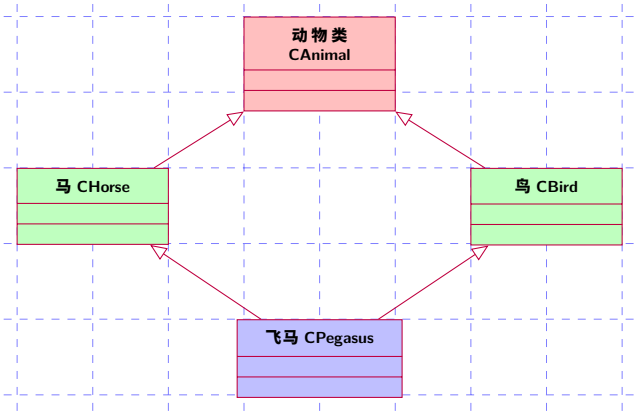
- 如何通过派生类对象调用基类中被覆盖的成员函数



- 飞马 (Pegasus)



- 飞马 (Pegasus)



代码示例

```
// 例 05-09: ex05-09.cpp
// 类 protect 属性 和
// public 方法 的演示
#include <iostream>

using namespace std;
class base1
{
protected:
   int x;
public:
   void showx()
   {
      cout << x << "\n";
   }
};
class base2
{
protected:
   int y;
public:
   void showy()
   {
      cout << y << "\n";
   }
};
```

```
// 例 05-09: ex05-09.cpp
//定义类 derived，该类继承了 base1 和 base2

class derived: public base1, public base2
{
public:
   void set(int i, int j)
   {
      x=i;
      y=j;
   }
};
```

```
// 例 05-09: ex05-09.cpp

int main()
{
   derived ob;
   ob.set(10, 20);
   ob.showx();
   ob.showy();
}
```

多继承的构造与析构

- 调用各基类构造函数：调用顺序按基类被继承时声明的顺序，从左向右依次进行
- 调用派生类成员对象构造函数：调用顺序按其在类中定义的顺序依次执行
- 调用派生类构造函数

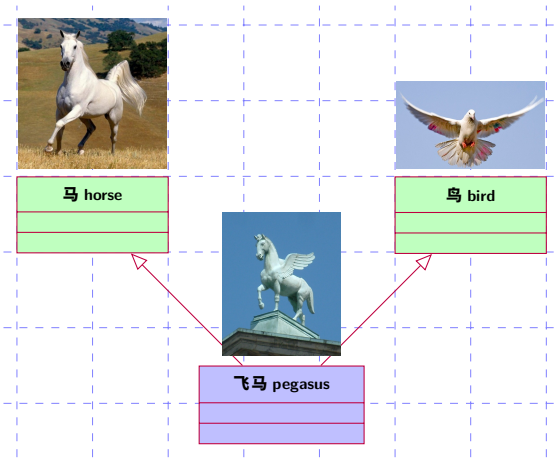
代码示例

```cpp
// 例 05-10: ex05-10.cpp
// 构造函数和析构函数的演示

#include <iostream>

using namespace std;

class base1{
public:
   base1()   {
      cout << "Constructing base1\n";
   }
   ~base1()   {
      cout << "Destructing base1\n";
   }
};

class base2{
public:
   base2()   {
      cout << "Constructing base2\n";
   }
   ~base2()   {
      cout << "Destructing base2\n";
   }
};
```

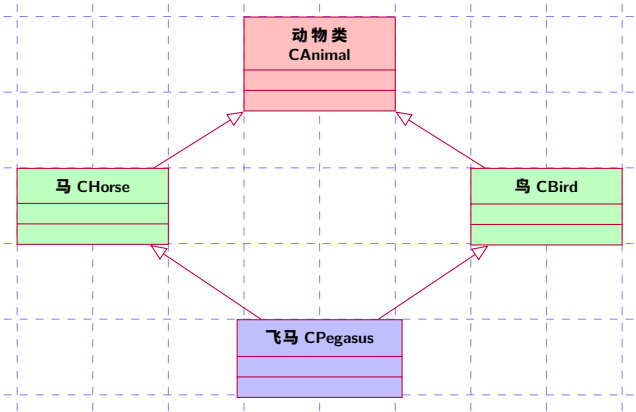
```cpp
// 例 05-10: ex05-10.cpp
//定义类 derived，该类继承了 base1 和 base2

class derived: public base1, public base2{
public:
   derived()   {
      cout << "Constructing derived\n";
   }
   ~derived()   {
      cout << "Destructing derived\n";
   }
};

int main(){
   derived ob;
}
```

```
testcpp
Constructing base1
Constructing base2
Constructing derived
Destructing derived
Destructing base2
Destructing base1

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

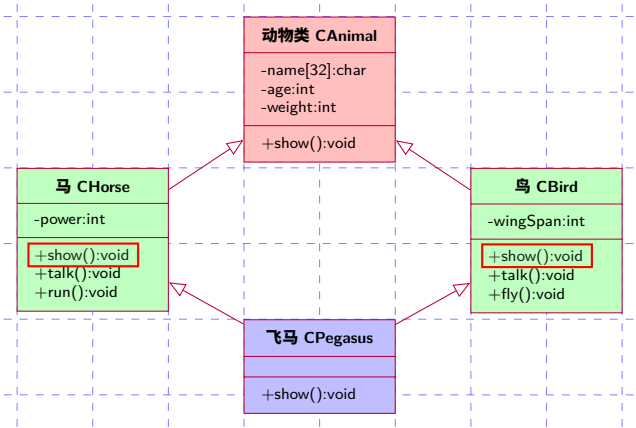
- 飞马 (Pegasus)



- 飞马 (Pegasus)



- 飞马 (Pegasus)


动物类 CAnimal
-name[32]:char
-age:int
-weight:int
+show():void
马 CHorse
-power:int
+show():void
+talk():void
+run():void
鸟 CBird
-wingSpan:int
+show():void
+talk():void
+fly():void
飞马 CPegasus
+show():void

动物类

```cpp
// 例 05-11: ex05-11.cpp
// 定义一个类 CAnimal

#include <iostream>
#include <cstring>

using namespace std;

class CAnimal
{
   char name[32];
   int age;
   int weight;
public:
   CAnimal(const char *strName="", int a=0, int w=0){
      strcpy(name, strName);
      age = a;
      weight = w;
      cout << "Animal constructor " << name << endl;
   }
   void Show(){
      cout << name << " " << age << " " << weight << endl;
   }
   ~CAnimal(){
      cout << "Animal destructor " << name << endl;
   }
};
```

鸟类

```
// 例 05-11: ex05-11.cpp
//定义一个类 CBird，该类继承 CAnimal 类，并且增加了 wingSpan 属性和一些方法

class CBird: public CAnimal
{
   int wingSpan;
public:
   CBird(int ws=0, const char *strName="", int a=0, int w=0):
       CAnimal(strName, a, w)
   {
      wingSpan = ws;
      cout << "Bird constructor " << endl;
   }
   void Show(){
      CAnimal::Show();
      cout << "Wingspan:" << wingSpan << endl;
   }
   void Fly(){
      cout << "I can fly! I can fly!!" << endl;
   }
   void Talk(){
      cout << "Chirp..." << endl;
   }
   ~CBird(){
      cout << "Bird destructor " << endl;
   }
};
```

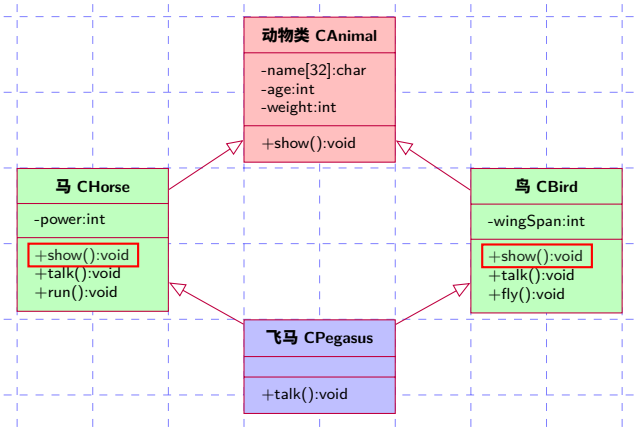
马类

```
// 例 05-11: ex05-11.cpp
//定义一个类 CHorse，该类继承 CAnimal 类，并且增加了 power 属性和一些方法

class CHorse: public CAnimal
{
   int power;
public:
   CHorse(int pow=0, const char *strName="", int a=0, int w=0):
        CAnimal(strName, a, w)
   {
      power = pow;
      cout << "Horse constructor " << endl;
   }
   void Show(){
      CAnimal::Show();
      cout << "Power:" << power << endl;
   }
   void Run(){
      cout << "I can run! I run because I love to!!" << endl;
   }
   void Talk(){
      cout << "Whinny!..." << endl;
   }
   ~CHorse(){
      cout << "Horse destructor "  << endl;
   }
};
```

飞马类

```cpp
// 例 05-11: ex05-11.cpp
//定义类 CPegasus ， 该类继承 CHorse 和 CBird

class CPegasus : public CHorse, public CBird
{
public:
   CPegasus(const char *strName="", int a=0, int w=0, int ws=0, int pow=0):
      CHorse(pow, strName, a, w), CBird(ws, strName, a, w)
   {
      cout << "Pegasus constructor" << endl;
   }
   void Talk(){
      CHorse::Talk();
   }
   ~CPegasus(){
      cout << "Pegasus destructor" << endl;
   }
};

int main(){
   CPegasus pegObj("Eagle", 5, 100, 2, 500);

   return 0;
}
```

- 多继承的二义性问题


动物类 CAnimal
-name[32]:char
-age:int
-weight:int
+show():void
马 CHorse
-power:int
+show():void
+talk():void
+run():void
鸟 CBird
-wingSpan:int
+show():void
+talk():void
+fly():void
飞马 CPegasus
+talk():void

- 派生类的多个基类中拥有同名成员时，继承后通过对象调用同名成员将出现二义性

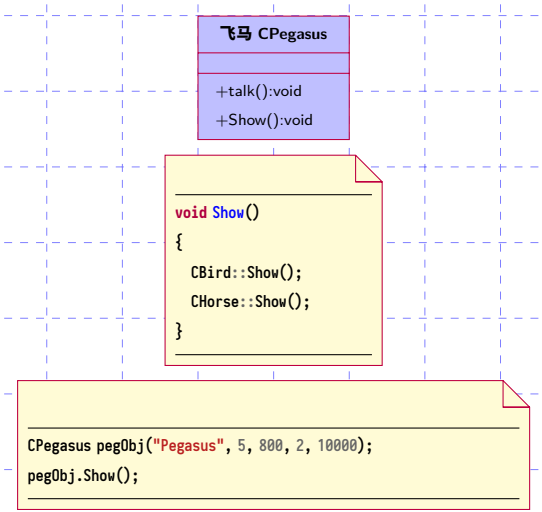
```
CPegasus pegObj("Pegasus", 5, 800, 2, 10000);
pegObj.Show(); ✘
```

```
F:\CPP\example... 112    error: request for member 'Show' is ambiguous
F:\CPP\example... 19     note: candidates are: void CAnimal::Show()
F:\CPP\example... 39     note:                 void CBird::Show()
F:\CPP\example... 67     note:                 void CHorse::Show()
                         === Build failed: 1 error(s), 8 warning(s) (0
```

- 解决方法 1：类型兼容

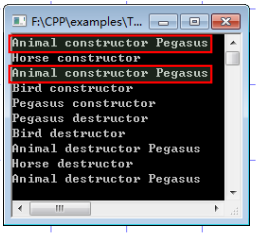
```
CPegasus pegObj("Pegasus", 5, 800, 2, 10000);
CBird birdObj = pegObj;
birdObj.Show();
CHorse horObj = pegObj;
horObj.Show();
```

- 解决方法 2：成员重定义

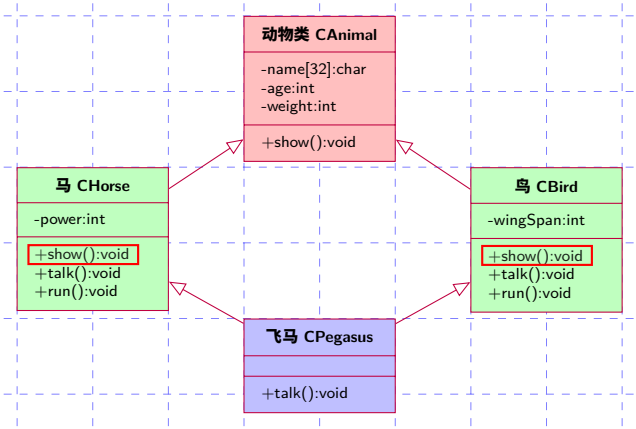

飞马 CPegasus
+talk():void
+Show():void
void Show()
{
CBird::Show();
CHorse::Show();
}
CPegasus pegObj("Pegasus", 5, 800, 2, 10000);
pegObj.Show();

- 间接二义性：基类构造函数两次被调用

```
CPegasus(char *strName="", int a=0, int w=0, int ws=0,intpow=0):
            CHorse(pow, strName, a,w), CBird(ws, strName, a, w)
{
    cout << "Pegasus constructor" << endl;
}
:
:
int main()
{
   CPegasus pegObj("Pegasus", 5, 100, 2, 500);
   return 0;
}
```



- 同名数据成员在内存中同时拥有多个拷贝



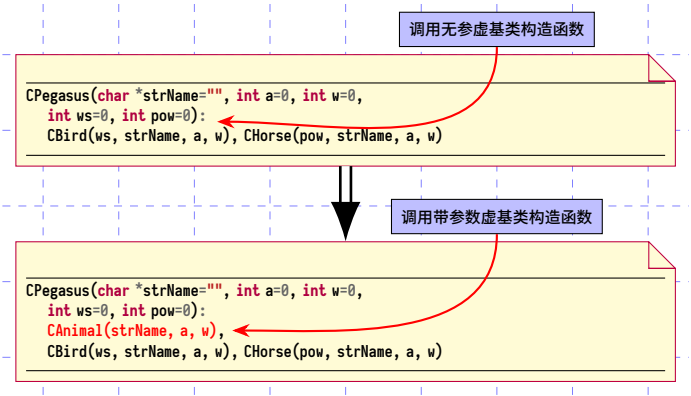
- 间接二义性
  - 如何解决从不同途径继承来的同名的数据成员在内存中有不同的拷贝问题（调用一次构造函数）？

- 定义

```
class 派生类名:virtual 继承方式 基类名
class CHorse: virtual public CAnimal
class CBird: virtual public CAnimal
```

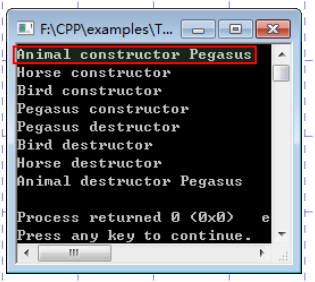
- 作用
  - 虚基类构造函数只被调用一次

- 间接二义性：基类构造函数两次被调用



构造函数调用

```
CPegasus pegObj("Pegasus", 5, 800, 2, 10000);
```

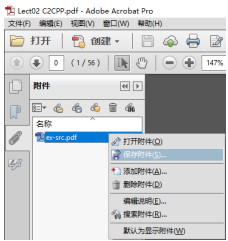


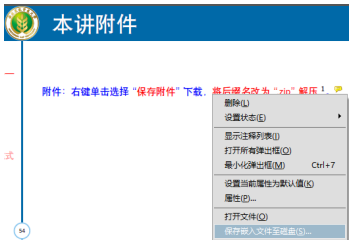
包含与继承的选择

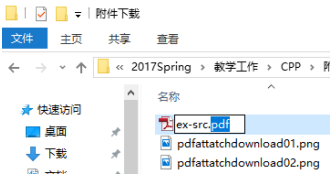
- 接口的多继承有一定价值，但应避免实现多继承。在决定使用多继承之前，先仔细考虑其他替代方案。
- 继承是面向对象提供的另外一种复用代码的重要机制，继承使得派生类与基类之间具有接口的相似性。派生类可看作是基类的特定子类型，派生类对象可替代基类对象。
- 与包含相比，继承需要更多的技巧，而且易出错，包含是面向对象编程中的主要技术之一。

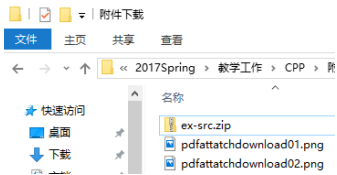
- 何时使用继承，何时使用包含？
  - 如果多个类共享数据而非行为，应该创建这些类可以包含的共用对象。
  - 如果多个类共享行为而非数据，应该让它们从共同的基类继承而来，并在基类里定义共用的操作。
  - 如果多个类既共享数据也共享行为，应该让它们从一个共同的基类继承而来，并在基类里定义共用的数据和操作。
  - 如果想由基类控制接口，使用继承；如果想自己控制接口，使用包含。

附件：右键单击该链接，选择“保存附件”下载，将后缀名改为“.zip”解压[1][2]。









[1]请退出全屏模式后点击该链接。

[2]以 Adobe Acrobat Reader 为例。

本讲结束，谢谢!
欢迎多提宝贵意见和建议

西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国 · 杨凌