10 对象和类

VCG

前言

- 面向对象编程(OOP）是一种特殊的、设计程序的概念性方法。最重要的OOP 特性
  - 抽象
  - 封装和数据隐藏
  - 多态
  - 继承
  - 代码的可重用性
- 为了实现这些特性并将它们组合在一起， C++所做的最重要的改进是提供了类

1 过程性编程和面向对象编程

任务/问题

- 记录Genre Giants垒球队队的统计数据
  - 输入每名选手的姓名、击球次数、击中次数、命中率以及其他重要的基本统计数据
  - 显示这些结果

过程性编程

- 过程性编程(面向过程)方法
  - 首先，考虑要遵循的步骤
    - 将程序分解为一系列按顺序执行的过程或函数
      - 过程或函数为具有特定功能的代码块（逻辑单元）
        - 相对独立，功能完整。是对具有相关性语句的归类和对某过程的抽象
        - 利用过程/函数名，接收输入参数，执行特别操作，并有返回值
  - 然后，考虑如何表示数据
    - 需要的和必要的数据
- 注意，过程性编程
  - 步骤和数据的操作顺序并没有那么严格，且可能需要多次迭代
  - 强调程序的操作过程和流程控制，而不太关注数据的状态和对象的交互
    - 不适合需要更高级抽象的场景

记录垒球队的统计数据

- 任务/问题
  - 要输入每名选手的姓名、击球次数、击中次数、命中率以及其他重要的基本统计数据
  - 还希望程序能够显示这些结果
- 首先，考虑要遵循的步骤
  - 【主要过程】main()调用一个函数获取输入，调用另一个函数计算，再调用第三个函数显示结果
  - 【细化和扩展】获得下一场比赛的数据后，【复用】，可以添加一个函数来更新统计数据。需要在main()中提供一个菜单，选择是输入、计算、更新显示数据
- 其次，表示数据
  - 用一个字符串数组来存储选手姓名，一个数组存储选手击球数，一个数组存储击中数目等
  - 【细化和扩展】以上方法不灵活，可设计一个结构来存储每位选手的所有信息，然后用该结构组成的数组来表示整个球队

面向对象编程

- 面向对象编程(Object Oriented Programming, OOP)
  - 首先考虑数据【数据的描述，设计抽象数据类型（和实现语言无关）】
    - 如何表示数据【数据属性】
    - 如何使用数据【接口】
  - 完成接口和数据表示的细节
  - 使用接口设计方案创建程序

什么是接口

- 接口
  - 是一个共享框架
  - 是一个交互/通讯机制，多个系统(用户、设备或者模块)交互时使用
  - 只强调交互的形式，而隐藏系统的内部处理细节
  - 接口一般定义为函数方式，其核心：完成的功能，接口名(函数名)，接口形式(参数，返回值)

面向对象编程

- 不仅要考虑如何表示数据， 还要考虑如何使用数据
- 选手
  - 有一个对象表示整个选手的各个方面
    - 基本数据单元，一个表示选手的姓名和统计数据的对象
  - 一些处理该对象的方法
    - 首先，需要一种将基本信息加入到该单元中的方法
    - 其次，应计算一些东西，如命中率，因此需要添加一些执行计算的方法
    - 另外，还需要一些更新和显示信息的方法
    - 总之，用户与数据交互的方式有：初始化、更新和报告，即为用户接口

面向对象编程

- 总之，OOP
  - 首先，从用户角度考虑问题【描述接口】
    - 描述对象所需要的数据
    - 描述用户与数据交互所需要的操作【定义接口】
  - 然后，确定如何实现接口和数据存储
    - 接口实现【实现具体的函数】
    - 数据存储【完善具体的数据表达】
  - 最后，使用设计方案创建程序【应用】
    - 用写好的类接口完成应用任务

抽象和类

2 抽象和类

- 简化和抽象，是处理复杂问题的原则之一
  - 将问题的本质特征抽象出来（而忽略其它部分）
    - 忽略干扰，聚焦注意力是人类高效处理信息的基础机制
  - 并根据特征来描述解决方案

- 程序设计问题求解时
  - 首先，抽象出**信息与用户之间的接口**
    - 如，在垒球统计数据示例中，用接口来描述了用户如何初始化、更新和显示数据
  - 然后，用接口来实现问题求解

- 程序设计语言中，户定义类型用于实现抽象接口，具体C++中，类

2.1 类型是什么

- 指定基本类型，意味着
  - 决定数据对象需要的内存数量
  - 决定如何解释内存中的位
    - 如，long和float在内存中占用的位数相同，但将它们转换为数值的方法不同
      - 即，内存中的二进制表示规则不同
  - 决定可使用数据对象执行的操作或方法

【注】

- 1，2决定了数据在内存中表示形式，3决定了该数据类型可以进行的操作
- 内置数据类型，其内存表示和操作信息被内置到编译器中

2.2 C++中的类

- C++中，类是抽象数据类型的实现，一种用户自定义类型
  - 类将数据表示和操纵数据的方法组合成一个整体

- C++类
  - 以数据成员的方式描述数据部分
  - 以public成员函数(称为方法)的方式描述操作数据的方法：接口

什么是接口

- 接口
  - 是一个共享框架
  - 是一个交互/通讯机制，多个系统(用户、设备或者模块)交互时使用
  - 只强调交互的形式，而隐藏系统的内部处理细节
  - 接口一般定义为函数方式
    - 完成的功能，接口名(函数名)，接口形式(参数，返回值)

C++类的接口

- 在类中，以公共接口的说法替代接口
  - 交互系统由类对象组成
  - 接口由编写类的人【类的开发者】提供的方法【类里的public函数】组成
  - 接口让程序员【类的使用者】能够编写与类对象交互的代码，从而让程序能够使用类对象
    - 如，程序员写好std::cin对象的>>接口，之后可以类似std::cin>>n；完成数据输入

- 程序员可以同时是类的开发者和类的使用者

设计一个表示股票的类

- 设计股票stock类【数据表示+操作数据方法】
  - 首先，考虑如何表示股票？
    - 数据对象对应什么， stock类应该包含哪些信息
  - 其次，定义股票数据的操作方法，定义出stock类的公有接口；为支持接口，还需要数据信息
  - 最后，完成接口实现
- C++具体实现，定义类
  - 类声明
    - 以数据成员的方式描述数据部分
    - 以成员函数(称为方法)的方式描述公有接口
  - 类方法定义
    - 描述如何实现类成员函数
- 使用类

P10.1 tock00.h

```
1.  #include <string>
2.  class Stock // class declaration
3.  {
4.  private:
5.    std::string company;
6.    long shares;
7.    double share_val;
8.    double total_val;
9.    void set_tot() { total_val = shares * share_val; }
10. public:
11.   void acquire(const std::string &c, long n, double p);
12.   void buy(long num, double price);
13.   void sell(long num, double price);
14.   void update(double price);
15.   void show();
16. }; // note semicolon at the end
```

类的定义(声明、类方法实现)、类的使用

```
1. // 类声明, .h文件中
2. class Stock {
3. private:
4.     std::string company;
5.     //…
6.     void set_tot() {total_val = shares * share_val;}
7. public:
8.     void acquire(const std::string & c, long n,
   double p);
9. };    // note semicolon at the end

10. //类方法实现, .cpp中
11. void Stock::acquire(const std::string &c, long n,
   double p)
12. {
13.     company = c;
14.     share_val = p;
15.     //…
16.     set_tot();
17. }
```

```
1. //类的应用, 另一个.cpp中
2. #include "stock00.h"
3. int main()
4. {
5.     Stock fluffy_the_cat;
6.     fluffy_the_cat.acquire("NanoSmart", 20, 12.50);
7.     fluffy_the_cat.show();
8.     return 0;
9. }
```

类定义

- Stock00.h(P10.1 tock00.h)
- 类定义
  - 第一步，提供类声明
  - 第二步，实现类成员函数
- 类声明包括
  - 类数据成员【完整的表示】
  - 类成员函数/方法【可以只是原型】
  - .h文件
- 部分类成员函数的完整定义
  - 在类声明外，其它文件中实现
  - .cpp文件

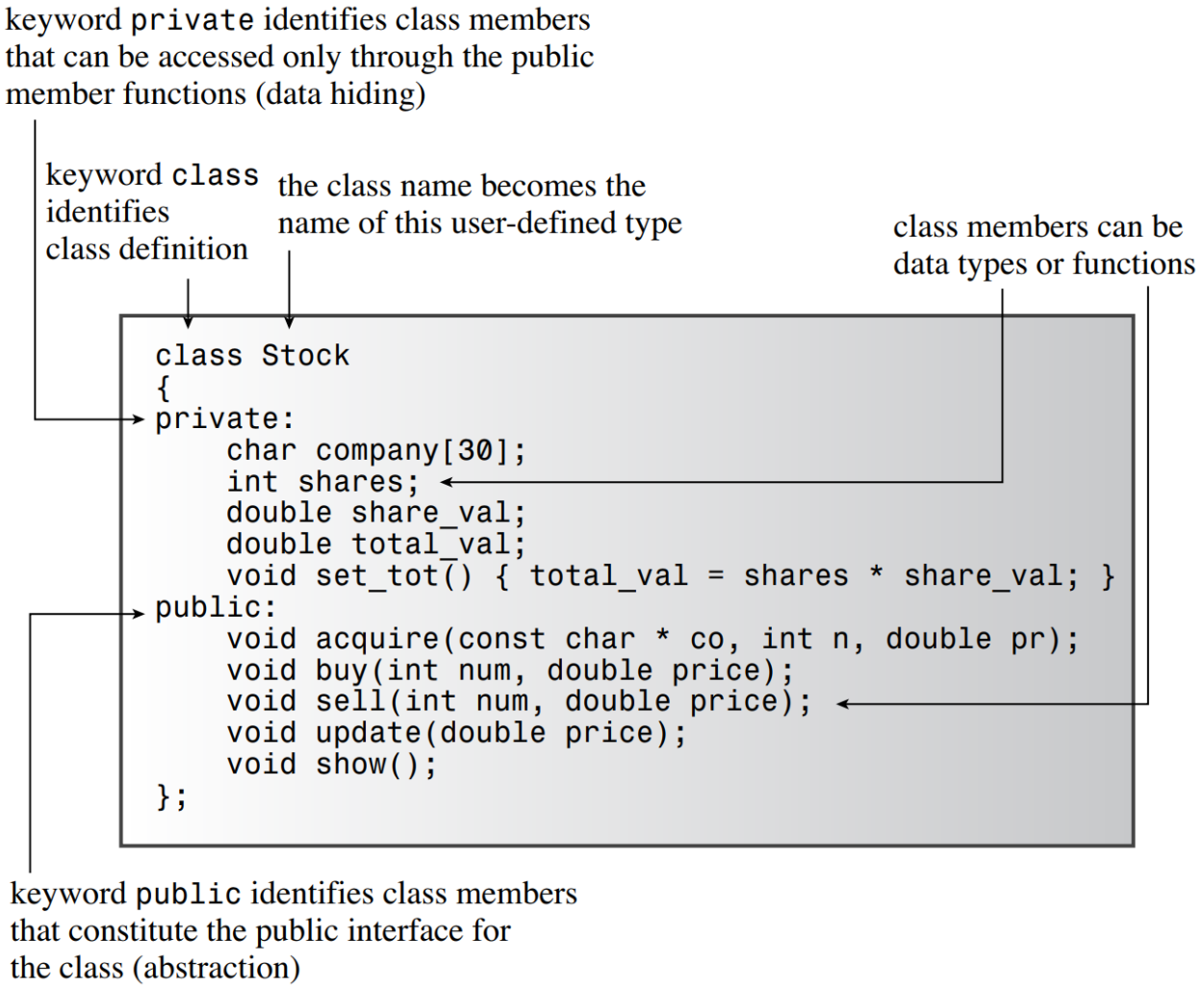
```
1.  //  01-stock00.h
2.  class Stock // class declaration
3.  {
4.  private:
5.      std::string company;
6.      void set_tot() { total_val = shares * share_val; }
7.  public:
8.      void acquire(const std::string &c, long n, double p)
9.  }; // note semicolon at the end

10. // 01-stock00.cpp
11. #include "01-stock00.h“
12. // 类成员函数（类方法）实现
13. void Stock::acquire(const std::string & co, long n,
    double pr)
14. {
15. }
```

类声明

- 类声明包括
  - 类数据成员【完整的表示】
  - 类成员函数/方法【可以只是原型】

- 访问控制，指对类成员(数据，函数)的访问
  - 关键字：private/public/protected
  - 类内访问
    - 类的成员函数内部，访问类的数据成员或者成员函数
    - 访问不受限
  - 类外访问
    - 该类的实例，去访问类的数据成员或者成员函数
    - 只能访问类中的public成员



类声明 - 访问控制

- 访问控制，指对类成员(数据，函数)的访问
  - 类内访问
    - 类的成员函数内部，访问类的数据成员或者成员函数
    - 访问不受限
  - 类外访问
    - 该类的实例，去访问类的数据成员或者成员函数
    - 只能访问类中的public成员
    - Stock sto; sto.show();
- private：类外不能访问
  - 用于数据隐藏，是一种封装机制的核心
  - sto.set_tot();//不可以！
- public：类外可以访问
  - 用于提供对外的接口

```
1.  class Stock {
2.  private:
3.      std::string company;
4.      void set_tot() { total_val; } //类内，可
5.  public:
6.      void acquire(const std::string & c, long n, double
    p);
7.  };

8.  void Stock::acquire(const std::string & co, long n,
    double pr){//类内，都可以访问，数据成员或者成员函数
9.      company = co; //类内，数据成员，可以
10.     set_tot(); //类内，成员函数，可以
11. }

12. int main(){
13.     Stock fluffy_the_cat;
14.     fluffy_the_cat.show( ); //类外，公有，可以
15.     fluffy_the_cat.company; //类外，私有，不可以！
16. }
```

封装

- 类设计尽可能将公有接口与实现细节分开
  - 公有接口表示设计的抽象组件
  - 将实现细节放在一起并将它们与抽象分开被称为封装
- 封装
  - 数据隐藏（将数据放在类的私有部分中）【能看到但不能访问数据成员】
  - 将（函数）实现的细节隐藏在私有部分中【能看到但不能访问成员函数的实现】
  - 将类函数定义和类声明放在不同的文件中【看不到成员函数实现代码】
- 数据隐藏不仅可以防止直接访问数据，还让开发者（类的用户）无需了解数据是如何被表示的

数据隐藏的好处

- 数据隐藏不仅可以防止直接访问数据， 还让开发者（ 类的用户 ） 无需了解数据是如何被表示的
  - 例如，show()成员将显示某支股票的总价格，这个值
    - 可以存储在对象中
    - 也可以在需要时通过计算得到
    - 从使用类的角度看，使用哪种方法没有什么区别。所需要知道的只是各种成员函数的功能；
    - 也就是说，需要知道成员函数接受什么样的参数以及返回什么类型的值
- 将实现细节从接口设计中分离出来
  - 如果以后找到了更好的、实现数据表示或成员函数细节的方法，可以对这些细节进行修改，而无需修改程序接口，这使程序维护起来更容易

OOP 和 C++

- OOP 是一种编程风格
  - 从某种程度说，它用于任何一种语言中，包括C
- C++中包括了许多专门用来实现 OOP 方法的特性
  - 首先，将数据表示和函数原型放在一个类声明中（而不是放在一个文件中），通过将所有内容放在一个类声明中，来使描述成为一个整体。
  - 其次，让数据表示成为私有，使得数据只能被授权的函数访问。

类成员函数公有还是私有?

- 隐藏数据是OOP主要的目标之一
  - 数据项，通常放在私有部分
    - 只在类成员函数中使用
- 作为公有接口的函数
  - 公有，public
- 不提供访问控制时，
  - 类声明中，默认private
  - 结构声明中，默认public

2.3 实现类成员函数

P10.2 stock00.cpp

- 类定义第二步
  - 为类声明中的原型表示的成员函数提供代码

- 成员函数有函数头和函数体，可以有返回类型和参数。此外：
  - 定义成员函数时，使用**作用域解析运算符(::)**来标识函数所属的类
  - 类方法可以访问类的private成员【属于类内】

```
1. #include <iostream>
2. #include "01-stock00.h“

3. void Stock::show()
4. {
5.     std::cout << "Company: " << company
6.               << "  Shares: " << shares << '\n'
7.               << "  Price: $" << share_val
8.               << "  Total: $" << total_val << '\n';
9. }
```

函数成员说明

- 类的定义中，使数据私有，并限于对公有函数访问，使得
  - 允许控制数据如何被使用【数据不能被直接访问，而只能通过接口访问】
  - 在这个例子中，允许加入安全防护措施，避免不适当的交易【用户只能调用接口】
- set_tot()，内部的，私有的。为什么？
  - 类的开发者调用，但不是给类的使用者使用
- 内联函数(为了加速函数调用，不追求运行效率可忽略)
  - 如函数写在声明中，默认为inline
  - inline void，而不是 void inline
  - Inline未必会被编译器执行

调用方法时，使用哪个对象？

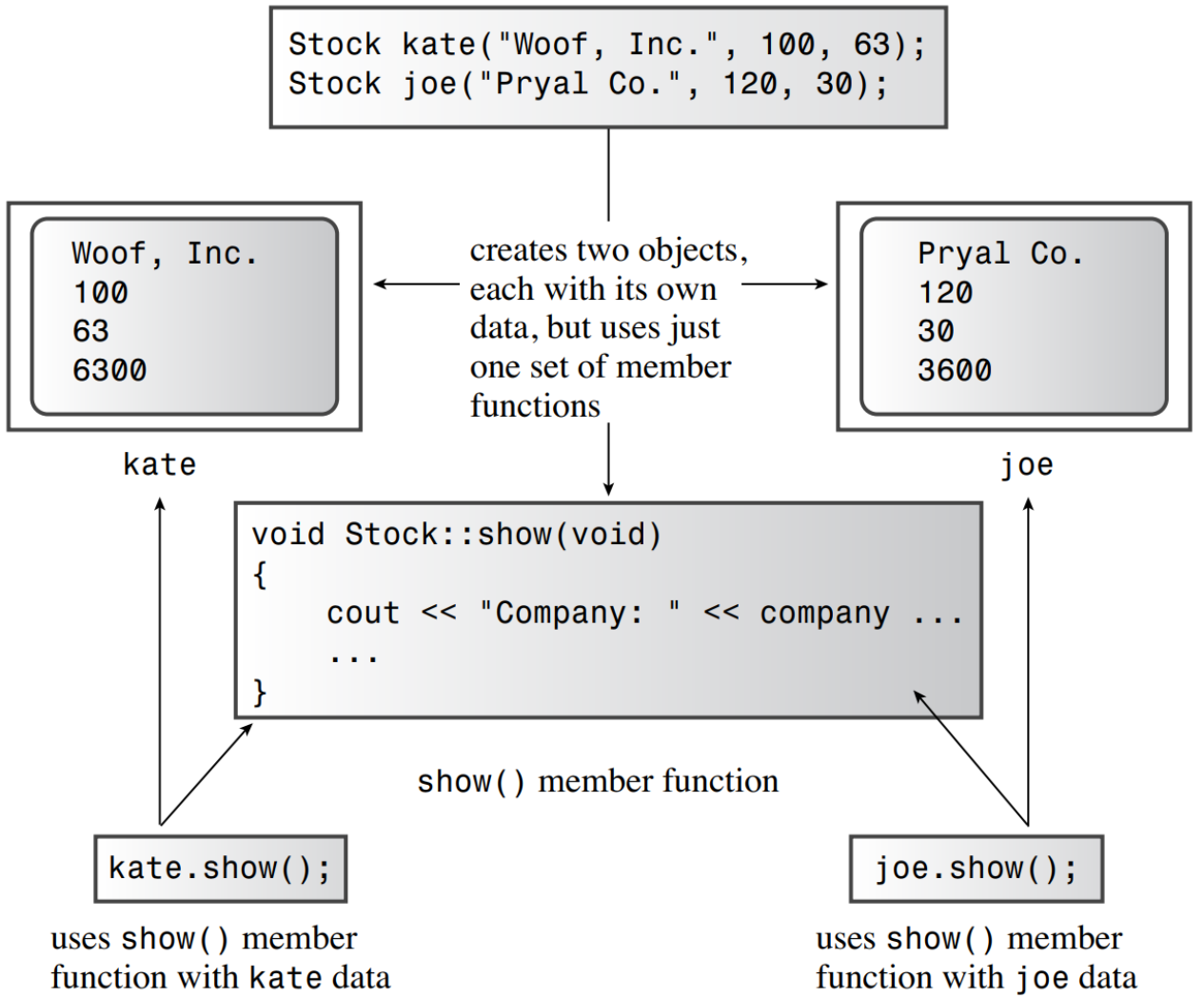
- 如何将类方法应用于对象？

```
shares += num;
```

  - 即，该语句使用的是哪个对象的shares ？

```
Stock kate, joe;
kate.show();
joe.show();
obj.show(); // 函数show中的类成员，是obj的
```

- 类成员的存储方式
  - 数据成员。不同对象，具有不同的存储空间。即每个对象都有自己的数据成员
  - 类方法/函数。同一个类的所有对象，共享同一组类方法/函数，即每种方法只有一个副本



2.4 使用类

P10.3, usestock0.cpp

- 引入类声明的头文件
  - #include …
- 实例化对象
  - Stock fluffy_the_cat;
- 通过对象，调用接口
  - fluffy_the_cat.show();

```
1. #include <iostream>
2. #include "01-stock00.h"

3. int main(){
4.     Stock fluffy_the_cat;
5.     fluffy_the_cat.acquire("Nano", 20, 12.50);
6.     fluffy_the_cat.show();
7.     fluffy_the_cat.buy(15, 18.125);
8.     fluffy_the_cat.show();
9.     fluffy_the_cat.sell(400, 20.00);
10.    fluffy_the_cat.show();
11.    fluffy_the_cat.buy(300000,40.125);
12.    fluffy_the_cat.show();
13.    fluffy_the_cat.sell(300000,0.125);
14.    fluffy_the_cat.show();
15.    return 0;
16.}
```

类比：服务器-客户端模型

- 服务器-客户端模型
  - 服务器：类声明(包括类方法)构成了服务器，它是程序可以使用的资源
  - 客户：使用类的程序
  - 接口：客户与服务器交互的方式
- 交互机制
  - 客户只能通过以公有方式定义的接口使用服务器
    - 意味着客户唯一的责任是，了解该接口。
  - 服务器的责任是，确保服务器根据该接口可靠并准确地执行
    - 服务器设计人员只能修改类设计的实现细节，而不能修改接口
    - 这样，对客户和服务器进行改进可以独立进行，对服务器的修改不会对客户的行为造成意外的影响

2.5 修改实现

- 修改show的内部实现
- 对cout不做过多要求
  - `std::cout.precision(3); // save preceding value for precision`

2.6 小结

- 类设计的第一步，提供类声明
- 类设计的第二步，实现类成员函数实现类成员函数

```
class className{
private:
    data member declarations
public:
    member function prototypes
};
```

类的构造函数和析构函数

3 类的构造函数和析构函数

- C++的目标之一是让使用类对象就像使用标准类型一样
- 问题之一：没有标准类型的初始化机制
  - 如，int year= 2001;
  - 原因
    - 数据的部分访问是私有
  - 解决方法
    - 设计合适的成员函数
    - 变量/对象创建时，(自动)进行
- 构造函数
  - 一个特殊的成员函数：类构造函数。专门用于构造新对象、将值赋给它们的数据成员
  - 更准确地说， C++实现了一种机制(包括成员函数命名在内的语法)，程序员根据需要去使用

3.1 声明和定义构造函数

- 构造函数定义
  - 函数名：类名
  - 返回值：没有(void也没有)
- 构造函数可以重载。其参数，由初始化数据成员的需求决定
  - 形参名最好不要和数据成员名一样
    - 要求数据成员命名有特点
      - 如，前缀m_，后缀_
    - 或者，this->
  - 否则，参数名和数据成员名混淆

```
1.  class Stock{
2.  public:
3.      Stock(); // default constructor
4.      Stock(const std::string &c, long n = 0, double p
    = 0.0);
5.      ~Stock(); // noisy destructor
6.  };

7.  // constructors (verbose versions)
8.  Stock::Stock() // default constructor
9.  {
10.     std::cout << "Default constructor called\n";
11. }

12. Stock::Stock(const std::string &c, long n, double p)
13. {
14.     std::cout << "Constructor using " << c << "\n";
15. }
```

3.2 使用构造函数

- 构造函数的调用(创建时自动调用)
  - 显示地调用
    - `Stock garment = Stock("Furry Mason", 50, 2.5);`
  - 隐式地调用
    - `Stock garment("Furry Mason", 50, 2.5);`
  - 动态创建，new
    - `Stock *pstock = new Stock("Electroshock Games", 18, 19.0);`

- 如果没有创建对象，不能调用构造函数！

```
Stock::Stock(“Furry Mason”, 50, 2.5); // 不能直接调用构造函数，错误！！
Stock *garment;                       // 没有创建对象
```

3.3 默认构造函数

- 如果类中没有定义任何构造函数，编译器将提供默认构造函数：
  - 没有参数
  - 不做任何事情

- 特别提醒
  - 当且仅当没有定义任何构造函数时，编译器才会提供默认构造函数
  - 如果有构造函数，则不会提供默认构造函数

- 良好编程习惯
  - 定义类时，要包括默认构造函数

3.4 析构函数

- 析构函数
  - 对象过期(撤销/删除/释放)时，自动调用的特殊的成员函数
  - 用于完成对象的清理工作
- 析构函数形式
  - ~类名称()
  - 没有参数
  - 没有返回值
- 析构函数一般自动调用
- 如果程序没有提供析构函数，系统将自动补上默认析构函数
  - 如果类中存在内存的动态申请，则析构函数最好不能省略！

3.5 改进stock类

（P10.4，stock10.h，P10.5，stock10.cpp，P10.6，usestock.cpp）

- 增加了构造函数和析构函数
- 初始化和赋值语句的区别
  - 初始化：创建对象，同时赋初值
  - 赋值语句：已经有了对象，之后的赋值

- const对象只能调用const方法

3.6 构造函数和析构函数小结

- 构造函数，在创建类对象时被调用
  - 用于初始化对象
  - 可以重载
  - 自动调用
- 析构函数，当对象被删除时被调用
  - 用于对象数据清理
  - 自动调用

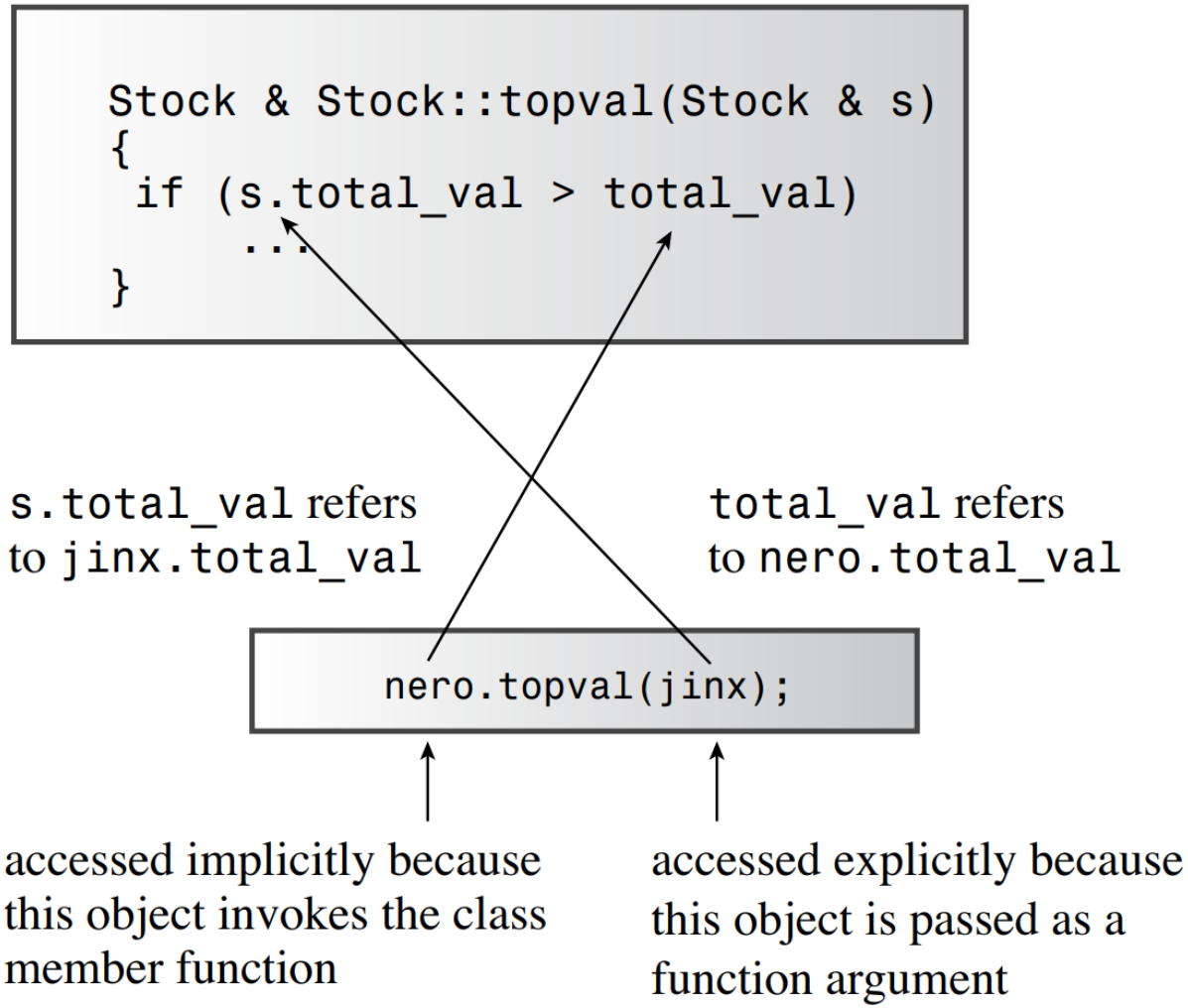
this指针

4 this指针

- 成员函数只涉及一个对象
  - 该函数中的类成员都是发起调用的对象的成员

```
1. top = stock1.topval(stock2);
2. const Stock &Stock::topval(const Stock &s) const
3. {
4.     if (s.total_val > total_val)
5.         return s;     // argument object
6.     else
7.         return ?????; // invoking object
8. }
```

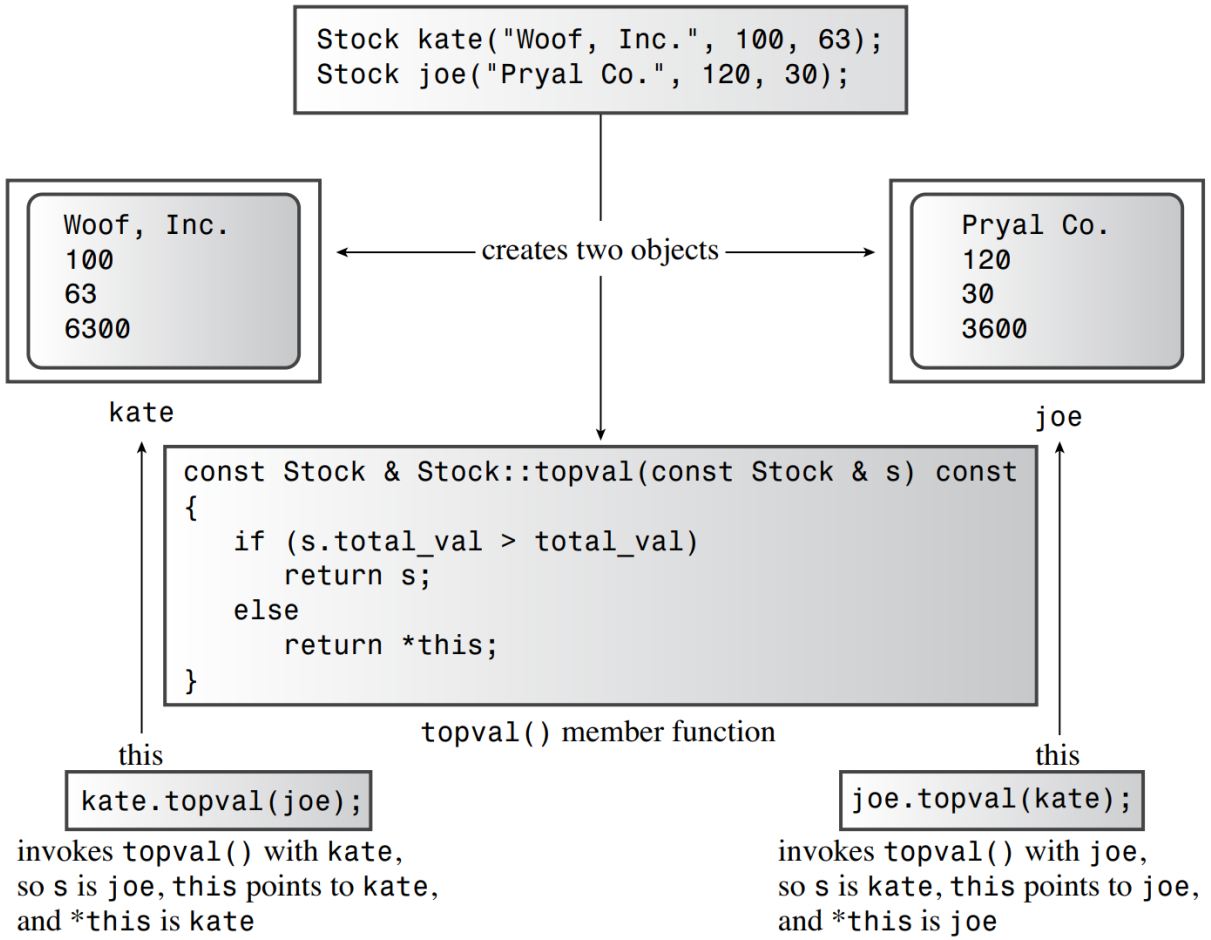
- 返回两个对象中股价总值最高的那个
  - 涉及多个对象
  - 在类成员函数topval内部，需要返回一个指针
    - 此处的?????表示，一个发起该函数调用的对象(stock1)的指针【在类内，看不到创建的对象stock1的】



this指针

- C++提供了this指针机制
  - this 指针指向用来调用成员函数的对象
    - this被作为隐藏参数传递给方法
    - stock1.topval(stock2)将this设置为stock1对象的地址
  - 编译器在编译类成员函数时，自动为该函数添加一个隐含形参this
    - 在该类成员函数中，有关类成员的访问均通过this指针进行，即this->成员

```
1. const Stock &Stock::topval(const Stock &s) const
2. {
3.     if (s.total_val > this->total_val)
4.         return s; // argument object
5.     else
6.         return *this; // invoking object
7. }
```



对象数组

5 对象数组

声明对象数组的方法与声明标准类型数组相同
(P10.9 usestock2.cpp)

- 数组初始化
  - 和一般数组声明类似
- 初始化
  - 必须为每个元素调用构造函数

```
1.  const int STKS = 4;
2.  int main(){
3.      // create an array of initialized objects
4.      Stock stocks[STKS] = {
5.          Stock("NanoSmart", 12, 20.0),
6.          Stock("Fleep Enterprises", 60, 6.5)
7.          };
8.      std::cout << "Stock holdings:\n";
9.      for (auto st = 0; st < STKS; st++)
10.         stocks[st].show();

11.    // set pointer to first element
12.     const Stock * top = &stocks[0];
13.     for (auto st = 1; st < STKS; st++)
14.         top = &top->topval(stocks[st]);

15.     return 0;
16. }
```

类作用域

6 类作用域

- 作用域：名称在文件多大范围内可见/使用
  - 全局(文件)作用域
    - 在全局变量所属文件的任何地方
  - 局部(代码块)作用域
    - 局部变量只能在其所属的代码块中
  - 函数名称的作用域
    - 可以是全局的，但不能是局部的
- C++引入类作用域
  - 类中定义的名称(变量/函数)，作用域为整个类
    - 即，类成员要带上类名的限定才有意义
  - 必须通过对象来访问类的成员
    - 直接成员运算符.
    - 间接成员运算符->
    - 作用域解析运算符::

```
1.  class Ik{
2.  private:
3.      int fuss; // fuss has class scope
4.  public:
5.      Ik(int f = 9) { fuss = f; } // fuss is in scope
6.      void ViewIk() const;  // ViewIk has class scope
7.  };
8.  void Ik::ViewIk() const // places ViewIk into Ik scope
9.  {
10.     cout << fuss << endl; //fuss in scope within class
11. }

12. int main()
13. {
14.     Ik *pik = new Ik;
15.     Ik ee = Ik(8);  // constructor in scope
16.     ee.ViewIk();    // brings ViewIk into scope
17.     pik->ViewIk();  // brings ViewIk into scope
18. }
```

6.1 作用域为类的常量

- 在类中
  - 不可以const
  - 可以enum
  - 可以static const int Months = 12;

```
1.  class Bakery{
2.  private:
3.      const int Months = 12;
4.      double costs[Months];
5.  };

6.  class Bakery{
7.  private:
8.      enum{Months = 12};
9.      double costs[Months];
10. };

11. class Bakery{
12. private:
13.     static const int Months = 12;
14.     double costs[Months];
15. };
```

6.2 作用域内枚举 - PASS

抽象和类

7 抽象数据类型

- 抽象数据类型(abstract data type, ADT)
  - 在抽象层次上，分析和描述数据类型
  - 可以认为是一种逻辑上的类型，和具体编程语言无关
  - 可以当成一种逻辑上的客观存在，与怎么用具体的方式描述无关
- 堆栈，应包含的操作
  - 可创建空栈
  - 可将数据项添加到堆顶(压入)
  - 可从栈顶删除数据项(弹出)
  - 可查看栈否填满。 可查看栈是否为空
- 然后，在具体的程序设计语言中实现堆栈

示例

- (P10.10 stack.h)
- 为了支持多类型，可使用typdef，但
  - 一个程序不能同时支持两种类型
  - 换不同类型时，要重新编译
  - 更好的解决方案，模板！
- 实现细节
  - 容器用固定长度数组实现
- 极端情况
  - 空的时候的pop
  - 满的时候的push

```
1.  typedef unsigned long Item;
2.  class Stack{
3.  private:
4.      enum {MAX = 10};     // constant specific to class
5.      Item items[MAX];     // holds stack items
6.      int top;             // index for top stack item
7.  public:
8.      Stack();
9.      bool isempty() const;
10.     bool isfull() const;
11.     // push() returns false if stack already is full,
    true otherwise
12.     bool push(const Item & item); // add item to stack
13.     // pop() returns false if stack already is empty
14.     bool pop(Item & item);  // pop top into item
15. };
```

8 总结

- 面向对象编程强调程序如何表示数据。使用OOP方法
  - 首先，根据它与程序之间的接口来描述数据，从而指定如何使用数据
  - 然后，设计一个类来实现该接口
- 通常将类声明分成两部分组成，这两部分通常保存在不同的文件中
  - 类声明(包括由函数原型表示的方法)应放到头文件中。定义成员函数的源代码放在方法文件中
- 类是用户定义的类型，对象是类的实例
- 每个对象都存储自己的数据，而共享类方法
- 如果希望成员函数对多个对象进行操作，可以将额外的对象作为参数传递给它
- 类很适合用于描述ADT
  - 公有成员函数接口提供了ADT 描述的服务，
  - 类的私有部分和类方法的代码提供了实现，这些实现对类的客户隐藏