CS 1502H 程序设计思想与方法 (C++)
Chapter 10. 创建新的类型 (3)

陈东尧
上海交通大学
2025年秋季

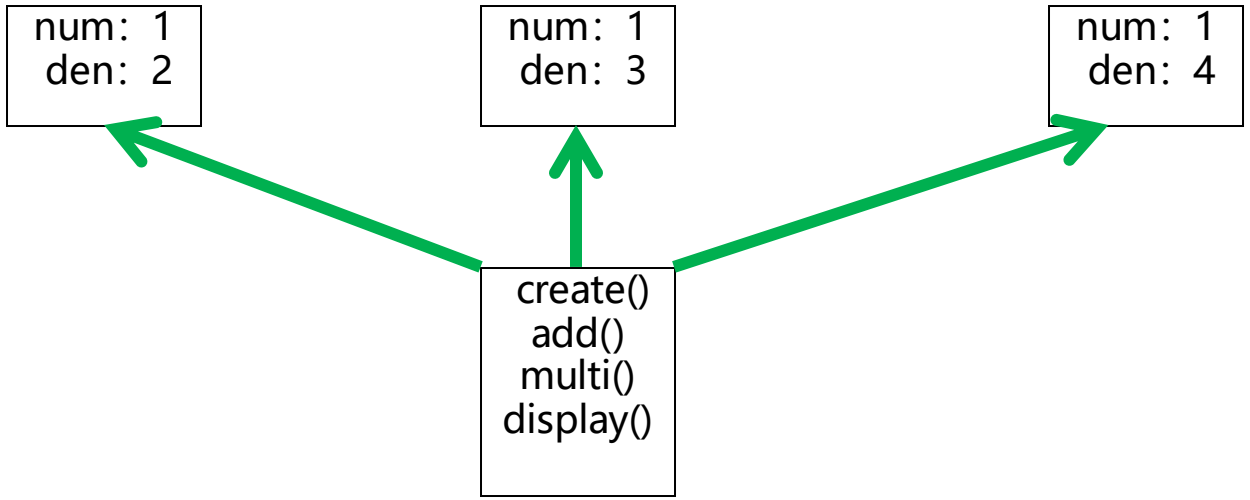
上节课的回顾

公有和私有成员

- 私有成员(private)：只能由类的成员函数调用
- 公有成员(public)：外部函数可以调用，类对外的接口
- 私有成员被封装在一个类中，类的用户是看不见的

this指针

•为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



如何实现这种链接？

对象的构造与析构

- 变量可以在定义时赋初值，对象能赋初值吗？怎么赋？

将赋初值的过程写成一个函数，让计算机在定义对象时执行该函数

这个函数称为构造函数

同理，有时在对象消亡前也有些工作要做，可将该过程写成一个函数，让计算机在对象消亡时执行该函数

这个函数称为析构函数

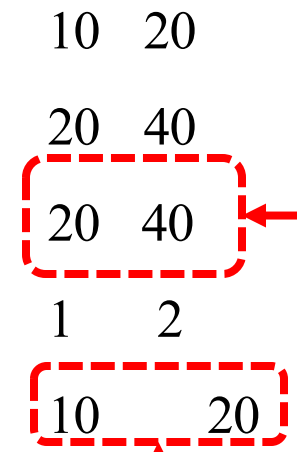
构造函数和析构函数是特殊的成员函数

拷贝构造的应用场合（1）：对象定义时

```
#include <iostream>
using namespace std;

class point{
int x, y;
public:
point(int a, int b) { x = a; y = b; }
point(const point &p) { x = 2 * p.x; y = 2 * p.y; }
void print() { cout << x << " " << y << endl; }
};

int main()
{
point p1(10, 20), p2(p1), p3 = p1, p4(1, 2);
p1.print(); p2.print(); p3.print(); p4.print();
p4 = p1; p4.print();
return 0;
}
```



通过赋值的定义默认调用拷贝构造函数

此处仅是赋值，没有拷贝过程。
因为p4已经初始化完毕了。

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

const 对象

const对象

- 定义后值不能被修改的对象

const对象的定义

- const MyClass obj（参数表）；

const 对象

const对象

- 定义后值不能被修改的对象

const对象的定义

- const MyClass obj（参数表）；

const 对象

const对象

- 定义后值不能被修改的对象

const对象的定义

- const MyClass obj（参数表）；

如何保证数据成员不被修改？

- 对const对象只能调用const成员函数

const成员函数

原型：返回类型　函数名（形式参数表）　const

一旦被定义成const成员函数，编译器会检查是否修改数据成员

- 如：

```
class A {
    int x;
public:
    A(int i) { x = i; }
    int getx() const  {  return x;  }
};
```

const是函数原型的一部分，
在类外定义时也必须加上

```
int A::getx() const
{
      return x;
 }
```

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

从问题（对象）出发，思考需求

- 关于固有变量的思考

从问题（对象）出发，思考需求

- 关于固有变量的思考
  1. 对象的哪些变量是固有的不能更改？

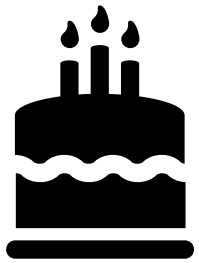
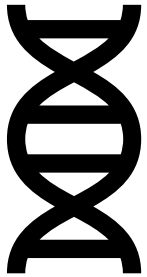
从问题（对象）出发，思考需求

- 关于固有变量的思考
  1. 对象的哪些变量是固有的不能更改？
  2. 哪些变量对于所属对象是固有的，但是不同对象间又不同

从问题（对象）出发，思考需求

- 关于固有变量的思考
  1. 对象的哪些变量是固有的不能更改？
  2. 哪些变量对于所属对象是固有的，但是不同对象间又不同

一个例子：生日

常量成员

const数据成员

在定义时赋初值，然后在整个对象生存期中值不能被修改

常量成员的声明

在该成员声明前加上const。如

```
class  People {
        const  int birthday ;
         …
};
```

常量成员

const数据成员

在定义时赋初值，然后在整个对象生存期中值不能被修改

常量成员的声明

在该成员声明前加上const。如

```
class  People {
        const  int birthday ;
         …
};
```

同一类的不同对象的const数据成员的值可以不同

（即：一个人的生日不会变，每个人的生日很可能不同）

常量数据成员

- 不能在类声明中初始化const数据成员。应在对象产生时初始化

```
class A
{
  //错误，企图在类声明中初始化const数据成员
  const int SIZE = 200;
};
```

const数据成员的初始化

- 只能在类构造函数的初始化表中进行
- 例：

```
class A
{       A(int size);                //构造函数
        const int SIZE;    }
A::A(int size) : SIZE(size)         //构造函数的初始化表
{...}
A a(100);   //对象a的SIZE的值为100
A b(200);   //对象b的SIZE的值为200
```

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

从问题（对象）出发，思考需求

- 在一些应用中，有没有哪些数据成员需要**所有对象需要共用**？

从问题（对象）出发，思考需求

- 在一些应用中，有没有哪些数据成员需要所有对象需要共用？
- 例子：存款利率

静态数据成员

类的所有对象共享的数据成员

假设类SavingAccount专门用于存放存款帐户，它包括存户的姓名、地址、存款额、利率等成员变量：

```
class SavingAccount
{
    char  name[20]; //存户姓名
    char  addr[60];   //存户地址
    double  total;     //存款额
    double  rate;     //利率
     …
};
```

修改利率必须修改所有帐户的rate的值

静态数据成员

类的所有对象共享的数据成员

假设类SavingAccount专门用于存放存款帐户，它包括存户的姓名、地址、存款额、利率等成员变量：

```
class SavingAccount
{
    char  name[20]; //存户姓名
    char  addr[60];   //存户地址
    double  total;     //存款额
    double  rate;     //利率
     …
};
```

修改利率必须修改所有帐户的rate的值

解决思路

rate应该由各对象共享，用全局变量。**但全局变量有安全问题和变量名冲突问题**

限定为只能由类中对象共享

静态数据成员

- **声明格式**
- 在rate前面加上static

```
class SavingAccount
{
   char  name[20];          //存户姓名
   char  addr[60];          //存户地址
   double  total;           //存款额
    static double  rate;    //利率
   …
}
```

静态数据成员注意事项

静态数据成员不属于对象的一部分，而是**类中所有对象**共享；

定义对象时，不分配静态成员的空间

静态数据成员必须单独分配空间，称静态数据成员的定义

静态数据成员的定义

- **定义在类的实现文件中**
- 如在SavingAccount类的实现文件中，有如下的定义：
- double SavingAccount::rate = 0.05;
- 该定义为rate分配了空间，并给它赋了一个初值0.05。
- **如果没有这个定义，compiler会报告一个错误**

静态数据成员的使用

- **通过作用域操作符直接调用**
- 如： SavingAccount::rate
- 须是公有的

- **可以从对象引用它**
- 如有个SavingAccount类的对象obj，则可以用：obj.rate

静态数据成员的使用

- **通过作用域操作符直接调用**
- 如： SavingAccount::rate
- 必须是公有的

- **可以从对象引用它**
- 如有个SavingAccount类的对象obj，则可以用：obj.rate

- **由于是整个类共享的，因此不管用哪种调用方式，得到的值都是相同的**

静态成员函数

专门处理静态数据成员、不能处理其他数据成员的函数

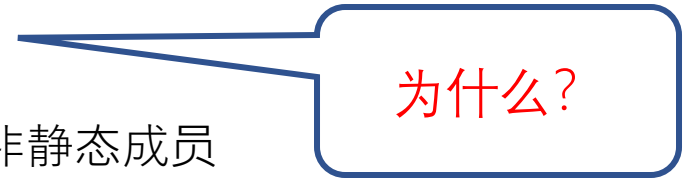
静态成员函数的声明

- 在类定义中的函数原型前加上保留词static
- 如在SavingAccount类中，当利率发生变化时，必须修改这个静态数据成员的值。为此可以设置一个静态的成员函数：

```
static void SetRate(double newRate)    { rate = newRate; }
```

特点

- 没有this指针
- 无法处理类中的非静态成员



静态成员函数

•专门处理静态数据成员、不能处理其他数据成员的函数

•静态成员函数的声明

•在类定义中的函数原型前加上保留词static

- 如在SavingAccount类中，当利率发生变化时，必须修改这个静态数据成员的值。为此可以设置一个静态的成员函数
- static void SetRate(double newRate)　{ rate = newRate; }

•特点

•没有this指针

因为静态成员函数不为对象服务，不需指明相关的对象

•无法处理类中的非静态成员

静态成员函数使用

- **通过类作用域限定符**
- 类名::静态成员函数名（实际参数）
- **通过对象**
- 访问对象名.静态成员函数名（实际参数）
- 注意：依然不能访问非静态成员

静态成员实例

- 在程序执行的某个时刻，有时需要知道某个类已创建的对象个数，现在仍存活的对象个数。

- **类设计**

数据成员

定义两个静态的数据成员：obj_count和obj_living

成员函数

构造函数：对这两个数各加1

析构函数：obj_living减1

显示函数：静态的成员函数返回这两个值

类定义

```
class StaticSample {
private:
   static int obj_count;
   static int obj_living;
public:
   StaticSample()  {  ++obj_count;    ++obj_living;     }
   ~StaticSample()  {  --obj_living;  }
   static void display( ) {
            cout << "总对象数：" << obj_count << "¥t存活对象数：”<< obj_living << endl;}
};

int StaticSample::obj_count = 0;          //静态数据成员的定义及初始化
int StaticSample::obj_living = 0;         //静态数据成员的定义及初始化
```

StaticSample的应用

```
int main()
{
    StaticSample::display();
    StaticSample s1, s2;
    StaticSample::display();
    StaticSample *p1 = new StaticSample,
                 *p2 = new StaticSample;
    s1.display();
    delete p1;
    p2->display();
    delete p2;
    StaticSample::display();

    return 0;
}
```

StaticSample的应用

```
int main()
{
    StaticSample::display();
    StaticSample s1, s2;
    StaticSample::display();
    StaticSample *p1 = new StaticSample,
                 *p2 = new StaticSample;
    s1.display();
    delete p1;
    p2->display();
    delete p2;
    StaticSample::display();

    return 0;
}
```

StaticSample的应用

```
int main()
{
    StaticSample::display();
    StaticSample s1, s2;
    StaticSample::display();
    StaticSample *p1 = new StaticSample,
                 *p2 = new StaticSample;
    s1.display();
    delete p1;
    p2->display();
    delete p2;
    StaticSample::display();

    return 0;
}
```

总对象数：0　　存活对象数：0
总对象数：2　　存活对象数：2
总对象数：4　　存活对象数：4
总对象数：4　　存活对象数：3
总对象数：4　　存活对象数：2

静态常量数据成员

- **静态的常量成员**
- 整个类的所有对象的共享常量

- **静态常量成员的声明**
- 如果成员的类型是整型或非标准整型

  static const int 数据成员名 = 常量表达式;
- 其他类型，在类中只是声明，在类外给初值
- 类声明： static const double st;

  类外：const double 类名::st = 1.5;

静态常量成员实例

•例如，某个类中需要用到一个数组，而该数组的大小对所有对象都是相同的，则在类中可指定一个数组规模，并创建一个该规模的数组。如下所示：

```
class sample {
    static const int SIZE = 10;
    int storage[SIZE];
    …
};
```

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

友元

- 私有成员只能让它的成员函数来访问

友元

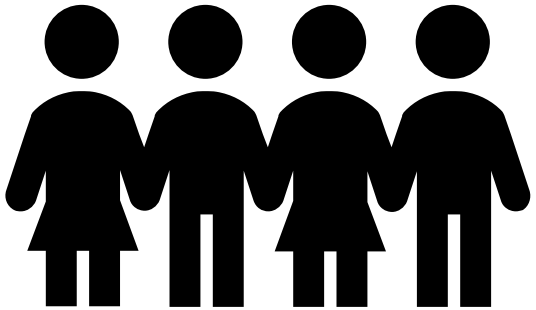
- 私有成员只能让它的成员函数来访问

如何灵活变化私有成员的访问？

友元

- 私有成员只能让它的成员函数来访问

如何灵活变化私有成员的访问？



友元

- 友元是一条通往私有成员的通道

友元分类

- 全局函数（友元函数）
- 另一个类的成员函数（友元成员）
- 整个类（友元类）

友元

- 友元是一条通往私有成员的通道

友元分类

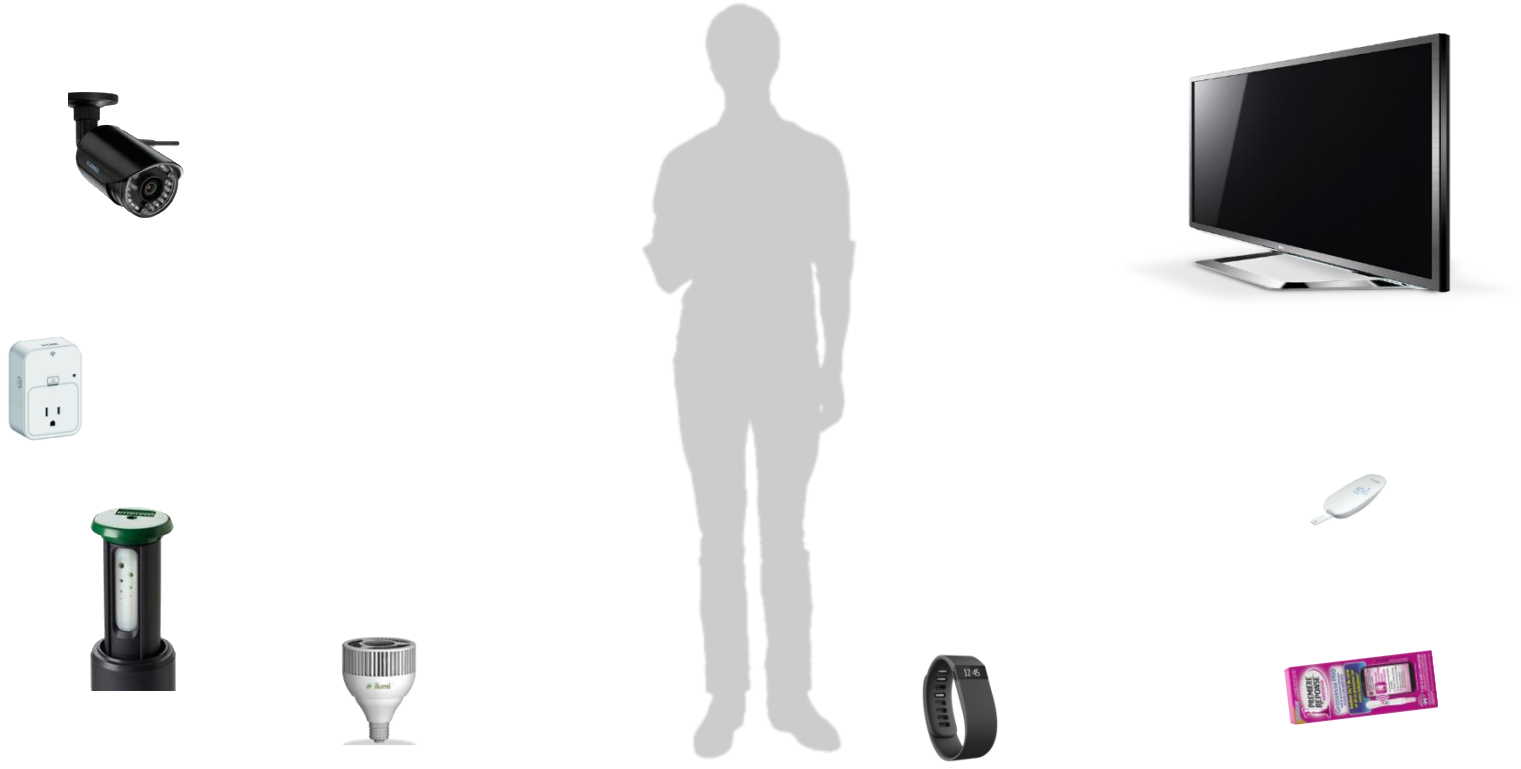
- 全局函数（友元函数）
- 另一个类的成员函数（友元成员）
- 整个类（友元类）

友元用途

- 为某个类量身定做的一些工具
- 如电视机的遥控器：遥控器可以控制电视的内部开关，光电控制单元等等

友元的其他实际例子：

- 智能设备



友元的其他实际例子：

- 智能设备：帮助链接不同的实体设备，大大增加生活的便捷

友元特点

- **友元关系是授予的而不是索取的**
- 如果函数 f 要成为类A的友元，类A必须显式声明函数f是他的友元，而不是函数f自称是类A的友元。
- **友元关系不是对称关系**
- 如果类A声明了类B是它的友元，并不意味着类A也是类B的友元。
- **友元关系不是传递关系**
- 如果类A是类B的友元，类B是类C的友元，并不意味着类A是类C的友元。

友元函数

- 友元函数是一个全局函数
- 函数定义可以在类中，也可以在类外
- 不管定义在哪里，都不是类的成员

- **友元函数声明**
- friend 返回类型 函数名(参数表);

友元函数实例

为DoubleArray增加一个输出函数

```
void Print(const DoubleArray &arr)
{
    for (int k = 0; k <= arr.high – arr.low + 1; ++k))
        cout << arr.storage[k] << endl;
}

class DoubleArray{
   friend void Print(const DoubleArray &);
   ……
};
```

友元函数实例

为DoubleArray增加一个输出函数

一般都有个所在类的对象参数

```
void Print(const DoubleArray &arr)
{
    for (int k = 0; k <= arr.high – arr.low + 1; ++k))
        cout << arr.storage[k] << endl;
}
```

友元函数可以定义在类中

```
class DoubleArray{
    friend void Print(const DoubleArray &);
    ……
};
```

友元成员

- 其他某个类的成员函数
- 可以访问friend声明语句所在类的私有成员

- **格式**
- friend 函数返回类型 类名标识符::函数名(参数列表)；

类声明

- 使用友元成员或友元类时，需要声明类

```
class B;

class A {
public:
  int Func1( B& b );
};

class B {
 // A::Func1 is a friend function to class B, so A::Func1 has access to all members of B
  friend int A::Func1( B& );
private:
  int _b;
};

int A::Func1( B& b ) { return b._b; }   // OK
```

友元类

- 整个类作为另一个类的友元
- 当A类被说明为类B的友元时，类A的所有函数都是类B的友元函数

- **声明方法**
- friend class 类名；
- 如：

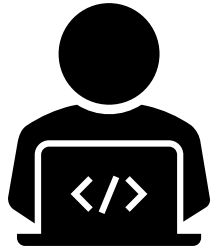
```
class Y;
class X{
        ….
    friend Y;          //    或friend class Y
       ……
};
```

友元类实例：电视机和遥控器

```
class  Remote;
class Tv
{
    friend class Remote;

    enum {Off, On};
    enum {MinVal,  MaxVal = 20};

    int state;              // on or off
    int volume;
    int maxchannel;
    int channel;
```



代码演示

remote_tv-example.cpp

友元类实例：电视机和遥控器

```
public:
    Tv(int s = Off, int mc = 125) : state(s),  volume(5), maxchannel(mc),  channel(2)  {  }
    void onoff() {state = (state == On)? Off : On;}
   void volup() { if (volume < MaxVal) volume++;   }
    void voldown() {   if (volume > MinVal)  volume--;  }
    void chanup()  {    if (channel < maxchannel)   channel++;    else    channel = 1;    }
    void chandown()  {    if (channel > 1)   channel--;    else channel =  maxchannel;    }
    void settings() const
    {
        cout << " 电视机是:" << (state == Off? "Off" : "On") << endl;
        if (state == On)   {
            cout << "  音量 = " << volume << endl;
            cout << "  频道 = " << channel << endl;
        }
    }
};
```

友元类实例：电视机和遥控器

```
class Remote
{
public:
        void volup(Tv & t) { if (t.volume < t.MaxVal) t.volume++;   }
        void voldown(Tv & t) {  if (t.volume > t.MinVal) t.volume-- ;   }
       void onoff(Tv & t) { t.onoff(); }
       void chanup(Tv & t) {  t.chanup();  }
       void chandown(Tv & t) {  t.chandown();  }
        void set_chan(Tv & t, int c) {  t.channel = c;  }    // 设置频道
};
```

友元类实例：电视机和遥控器的使用

```
int main()
{
    Tv tv1;
    cout << " tv1的初始状态是: ¥n";
    tv1.settings();
    tv1.onoff();
    tv1.chanup();
   cout << "¥n  调整设置后tv1的状态是:¥n";
   tv1.settings();
   Remote remote;
  remote.set_chan(tv1, 10);
   remote.volup(tv1);
   cout << "¥n  用遥控器调整后tv1的状态是:¥n";
   tv1.settings();

   return 0;
}
```

友元类实例：电视机和遥控器的使用

```
int main()
{
    Tv tv1;
    cout << " tv1的初始状态是: ¥n";
    tv1.settings();
    tv1.onoff();
    tv1.chanup();
   cout << "¥n  调整设置后tv1的状态是:¥n";
   tv1.settings();
   Remote remote;
  remote.set_chan(tv1, 10);
   remote.volup(tv1);
   cout << "¥n  用遥控器调整后tv1的状态是:¥n";
   tv1.settings();

   return 0;
}
```

```
tv1的初始状态是:
电视机是:Off
调整设置后tv1的状态是:
电视机是:On
音量 = 5
频道 = 3
用遥控器调整后tv1的状态是:
电视机是:On
音量 = 6
频道 = 10
```

友元声明说明

- 友元关系可以写在类定义中的任何地方
- 但一个较好的程序设计的习惯是将所有的友元关系的声明放在最前面的位置，并且不要在他的前面添加任何访问控制说明

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

用面向对象似乎更加复杂了？

- 通过工具数据和数据的处理封装起来了（怎么求的面积，**调用工具的人不知道**）
- 工具并不只针对本问题，是一个**普适的工具**
- 工具分成了截然不同的两部份：
  造工具和用工具，分工更容易，
  更有利于从软件工程管理

面向对象的程序设计的特点

- **代码重用：**
  - 例子：圆类型也可以被那些也需要处理圆的其他程序员使用
- **实现隐藏：**
  - 类的创建者创造新的工具
  - 类的使用者则收集已有的工具快速解决所需解决的问题
  - 这些工具是如何实现的，类的使用者**不需要知道**

总结

- 本章介绍了面向对象的程序设计的基本思想。
- 面向对象程序设计的基本思想是如何根据应用的需求创造合适的类型，用该类型的对象来解决特定的问题。
- 本章主要介绍了如何定义一个类，如何通过访问控制实现信息的封装。如何定义类的对象，如何实现对象的初始化，如何操作对象。