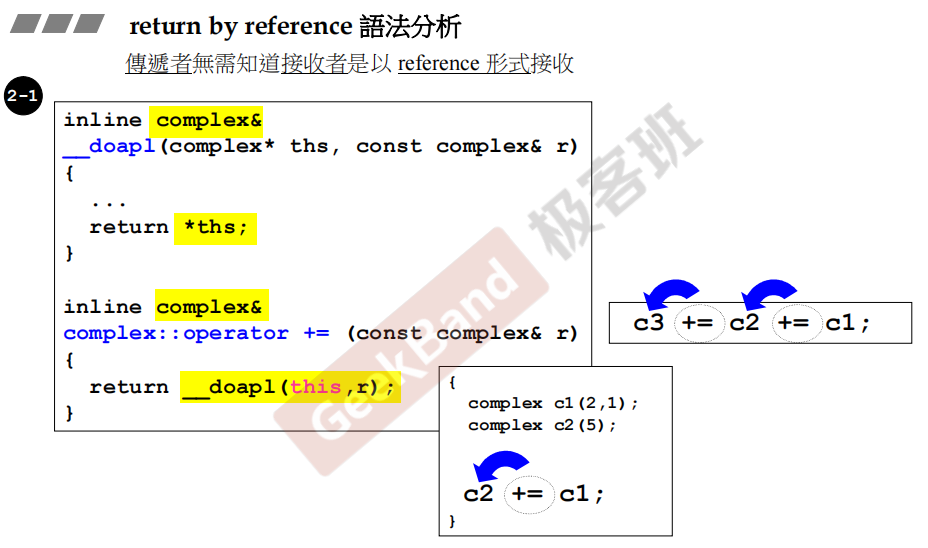
C++面向对象高级编程 上

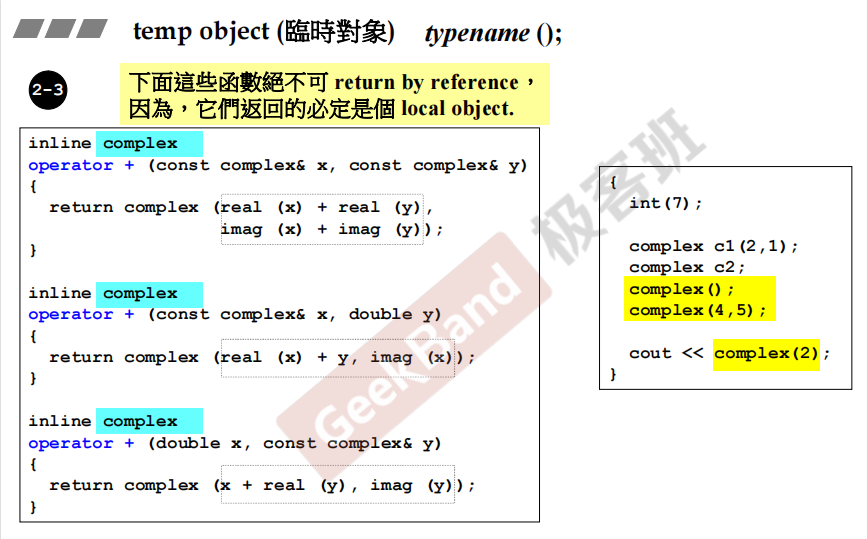
**5、操作符重载与临时对象**

**(1)、return by reference**



**注意：传递者无需知道接收者是by reference还是by value的形式进行接收；就如上例中第一个函数返回值是一个值传递，而函数的接收者却是一个引用；**

**(2)、临时对象**



**解释：1、语法：typename() --> 类型名+()，它们在程序的下一行就会得到释放，而得等到一个函数的结束；如上右图中的complex() 和 complex(4, 5) 都是临时对象，会在其下一行就被释放掉；**

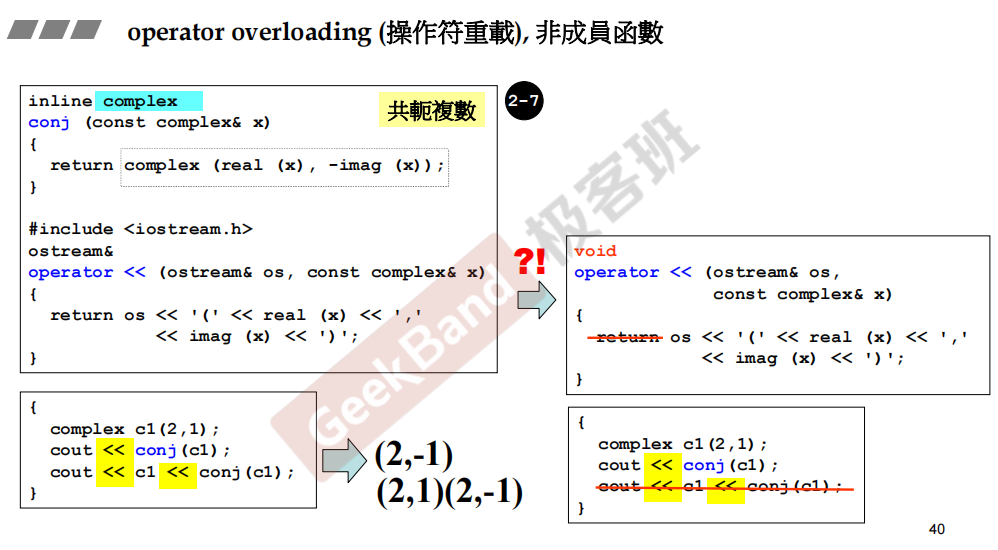
**2、要注意，临时对象作为返回值时，函数接收者绝对不能以引用的形式接收，因为临时对象在程序**

**的下一行已经被释放了，已经不存在了，无法对其进行引用，会报错；**

**3、以上的操作符重载都不是在类的内部，而是global全局函数，只是它的返回值是complex类类型，**

**这样的全局重载操作符函数，通过返回一个类的临时对象同样可以实现一个类类型的相加等的运算操作；**

**(3)、操作符重载 -- 非成员函数（全局函数）**

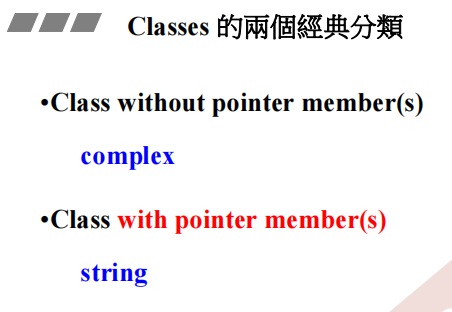


**注意：1、像上图中的 << 操作符，是不能写成成员函数的，而要写成全局函数，因为如上图中**

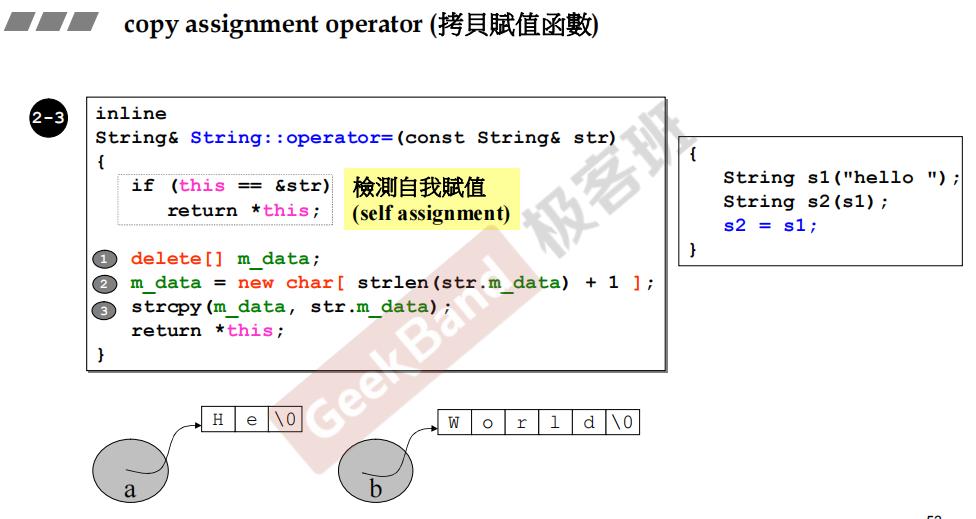
**是cout作为<<的调用对象，而cout是无法调用类型的成员函数的，所以要将<<的重载定义为全局函数；**

**7、三大函数：拷贝构造，拷贝复制，析构**

**(1)、接下来要讲的是类带指针的情况**



**(2)、拷贝赋值(复制)函数的实现**



**拷贝赋值的步骤：1、从代码可看出，先检测自我赋值的情况，如 s1=s1，如果是自我赋值，则直接**

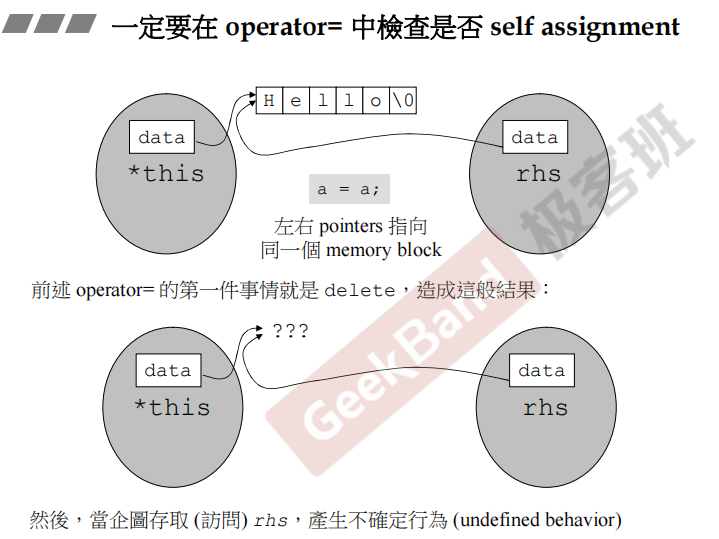
**返回自身对象，即对this指针解引用：\*this；**

**2、如果不是自我赋值，则先将调用者自身的内存释放掉，然后按照对方的数据内存大小开辟新的空间；**

**3、将对方数据的值复制到自身新开辟的空间中，最后返回自身对象，即\*this；**

**注意：**

**如果没有自我赋值检测的代码，可能出现未定义行为：**

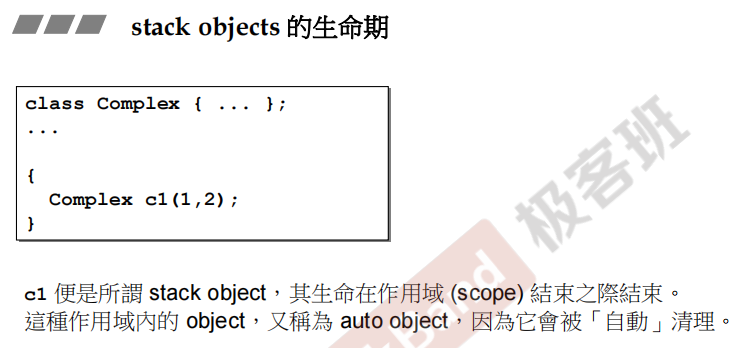


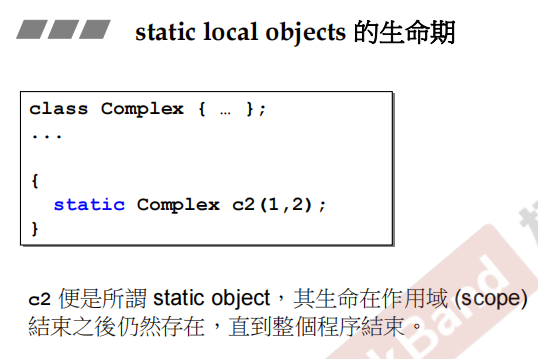
**如上图，当将自身对象的空间delete掉之后，还要根据自身对象的大小new出一个新的空间是不可能的，这时就会出现这种未定义的行为报错；**

**8、堆、栈与内存管理**

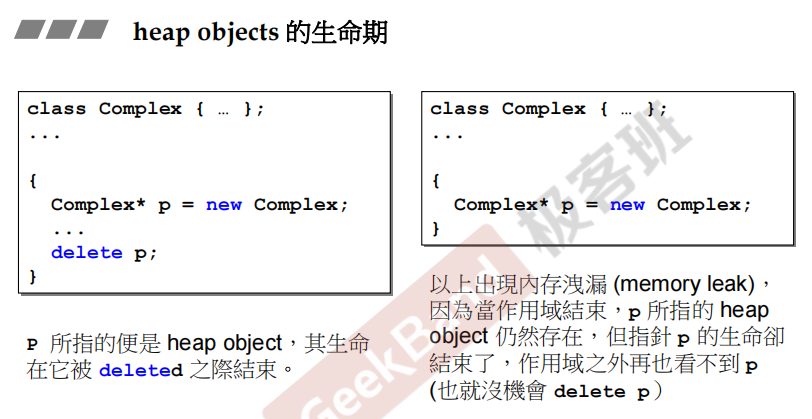
**(1)、stack、heap**



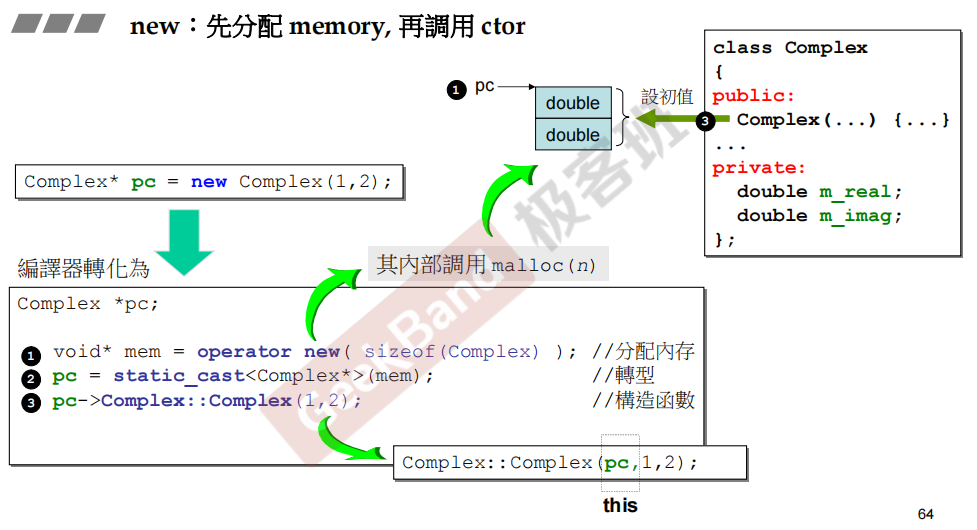








**(2)、new在一般编译器中的实现过程：三个动作**



**注意：**

**第一步：进行内存分配，上图中的operator new其实是一个函数的名称，其内部调用了malloc(n)，**

**其中的sizeof(complex)，计算的内存大小就是Complex类中的两个double成员的大小；**

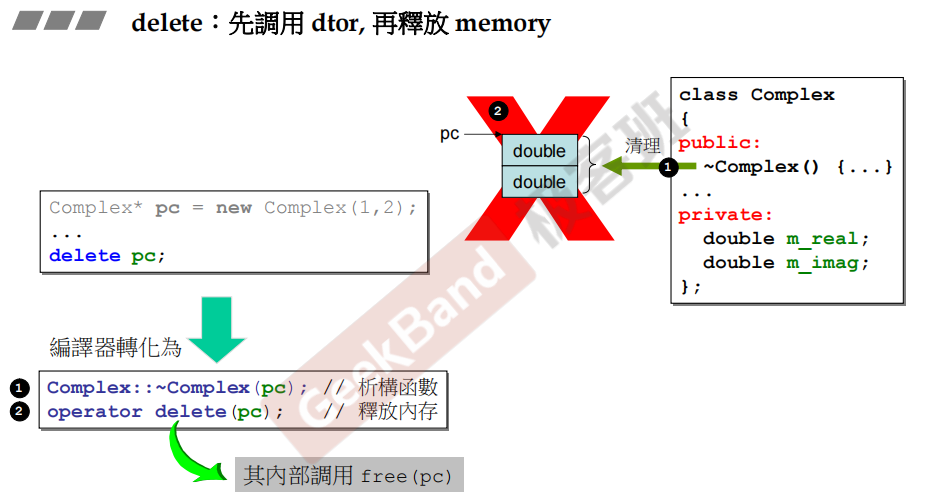
**第二步：比较不主要，是将void\* 类型的mem通过static\_cast转换为Complex\*类型，**

**第三步：在开辟出来的空间中调用构造函数，将对象的内容存入进去，这个构造函数内的对该对象**

**中的成员的赋值，是通过其构造函数自身自带的其所属对象的this指针实现的，这个调用者就是pc，**

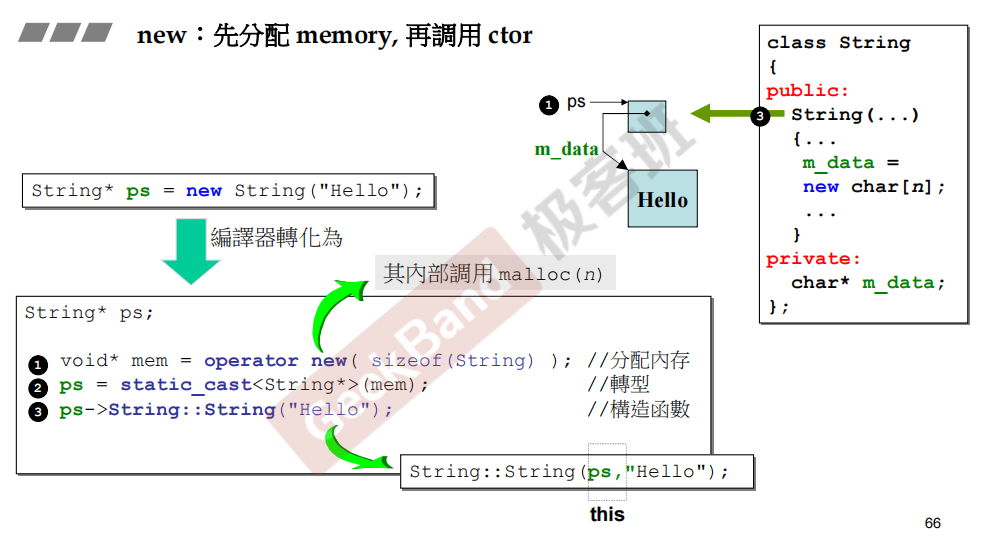
**所以this指针也是指向pc对象的；**

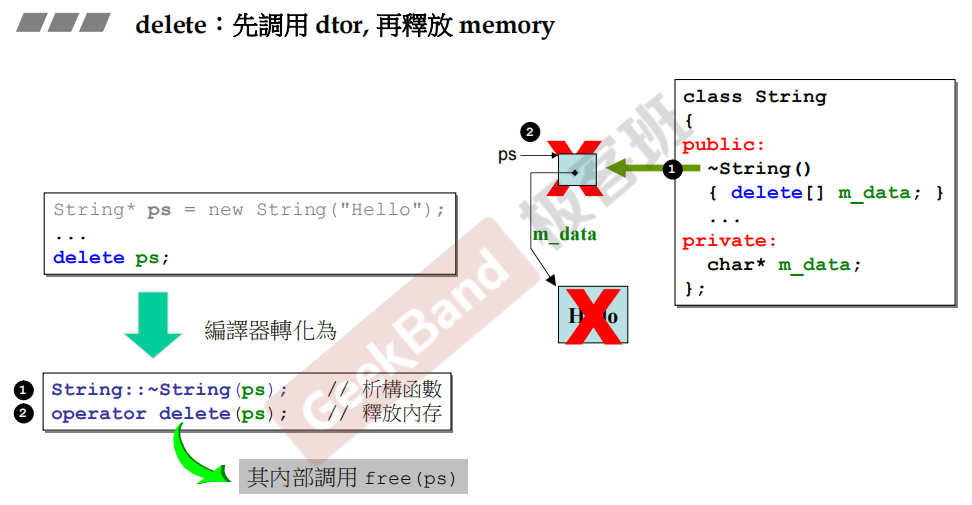
**(3)、delete的两个动作：先调用对象的析构函数，再释放内存 ----不带指针成员的类**



**注意：编译器中对第二步 operator delete函数的调用，实际上其内部调用了free(pc)；**

**(4)、带指针成员的类String： new 和 delete在编译器中的行为：**



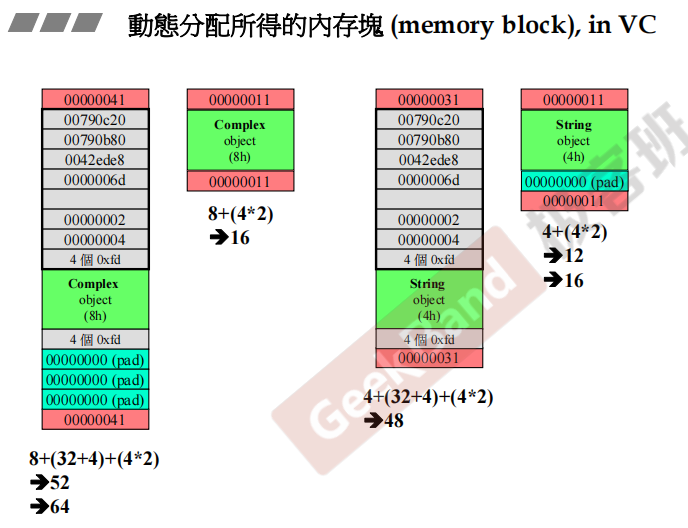


**上图中的delete ps分为两个步骤：**

**第一个是调用该对象的析构函数，将class string内部为指针成员m\_data分配的内存空间给释放掉；**

**第二个是将在外部new String("Hello")所为ps分配的空间，通过调用operator delete(ps)释放掉该字符串的内存；**

**(5)、在VC中动态分配的内存**

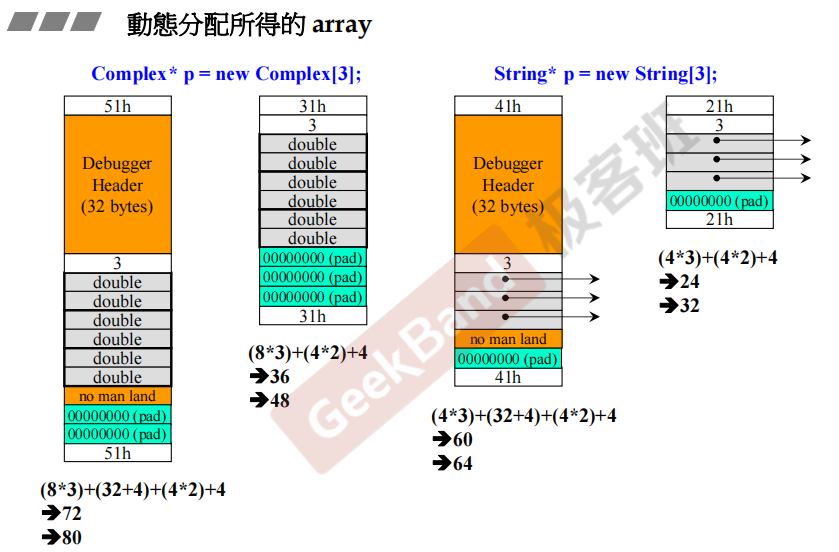


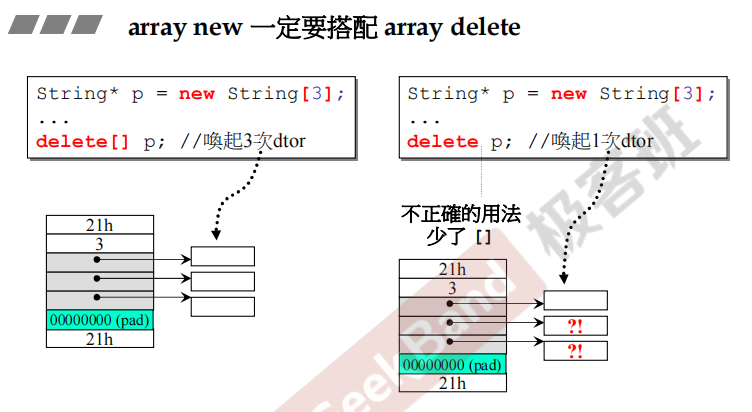
**上图中的灰色部分是在调试模式下多给的内存，为了方便回收，内存必须是16的倍数，所以第一个**

**的内存中的蓝青色部分是为了凑够64的pad内存，而最上面和最下面的红色部分是为了辨识指针所指**

**的内存到底是多大而设置的cookie饼干内存，如果去处了调试模式，就没有了灰色部分的内存；**

**(6)、arry new 一定要搭配 arry delete，就是要保证在delete时有 [] 符号**

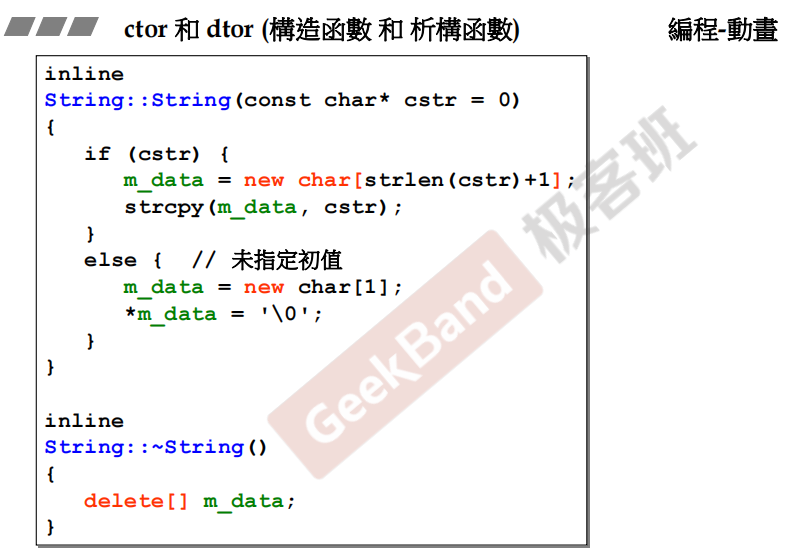


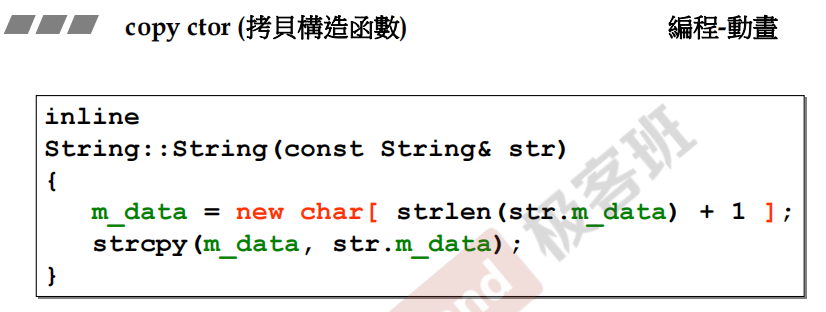


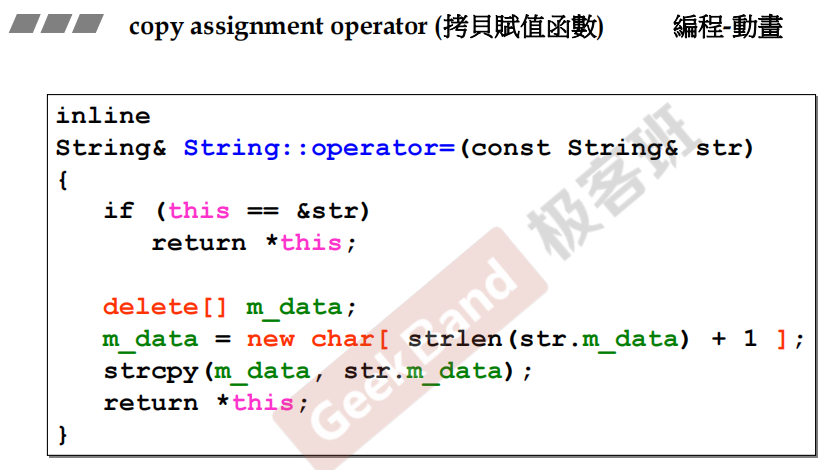
**9、复习String类的实现过程**



**通过动态分配的方式来存储字符串：**

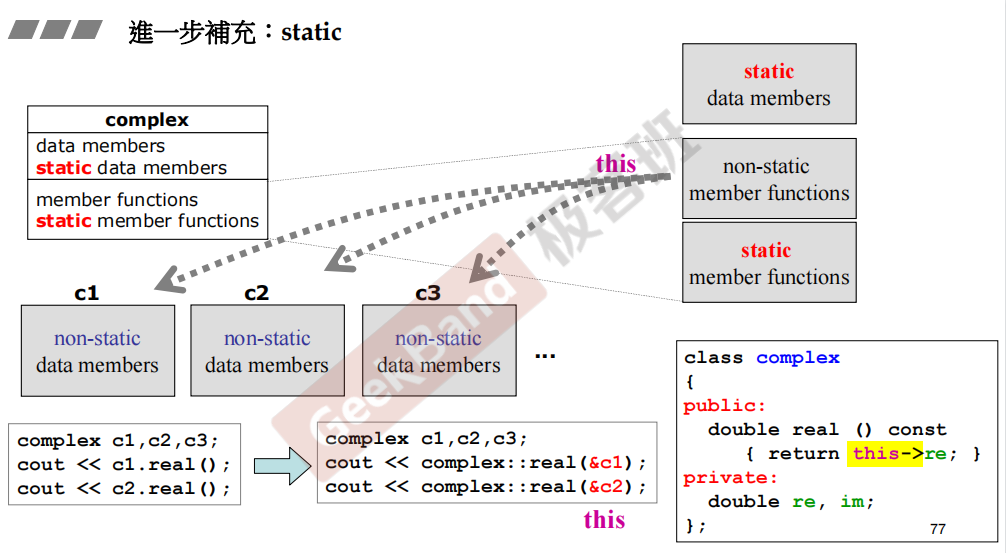






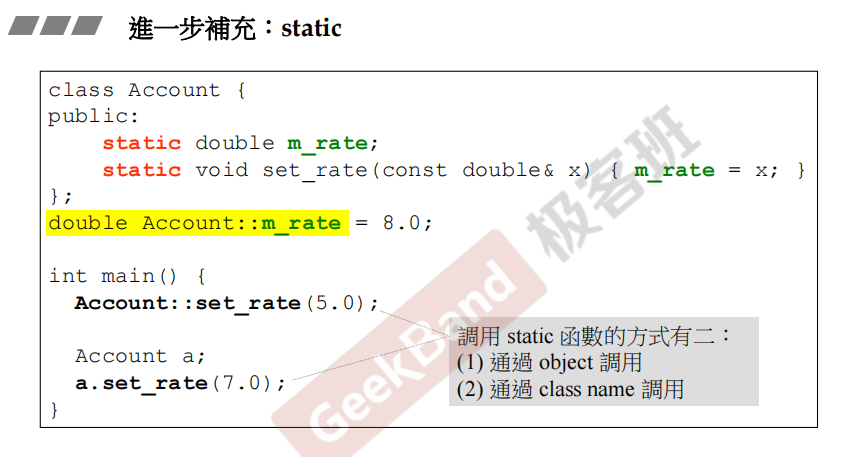
**10、扩展补充：类模板、函数模板及其他**

**(1)、static**



**注意：静态成员函数没有自带this指针，静态成员函数只能存取静态成员数据；**

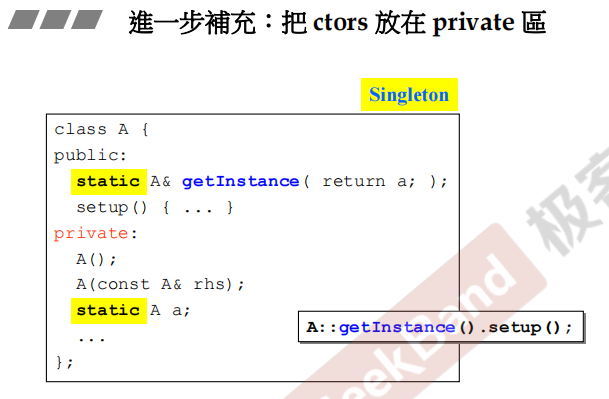
**调用static函数的两种方式：**



**单例模式：**

**只希望这个Class类产生一个对象，把它的构造函数放在private区后，在private区直接设定**

**一个自己的静态对象，同时为了让外界获得这个唯一的对象，需要对外提供一个接口getInstance()函数，内部直接返回在private区里创建的唯一对象a：**



**上面的写法仍然有缺陷，如果外界都不需要这个对象，而这个类的private区内却又一个作用在全局区域的 static A a 对象，会造成空间的浪费，应该是外界需要的时候再去设定这个对象，因此将声明这个**

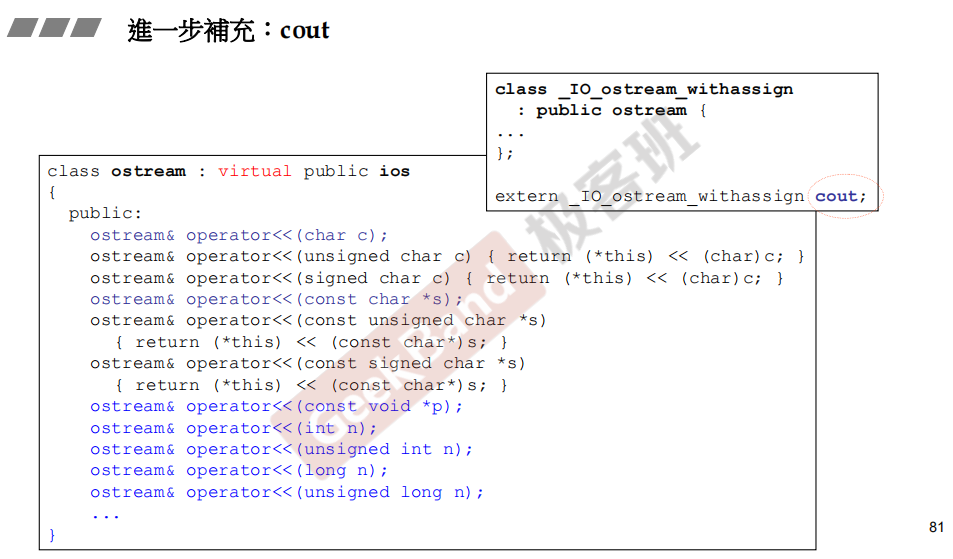
**静态对象的代码放在唯一的对外接口函数会更加的合适，实现外界需要时再去创建：**



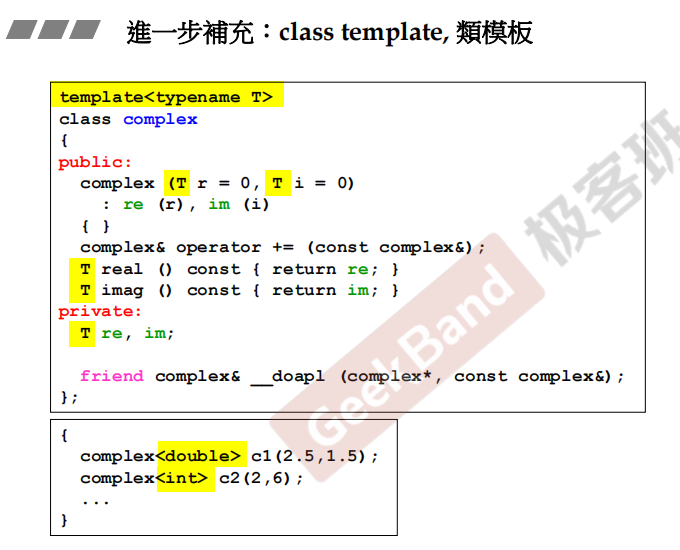
**注意：外界一旦需要这个单例对象，调用getInstance后这个静态对象就会一直存在于全局区，直到整个**

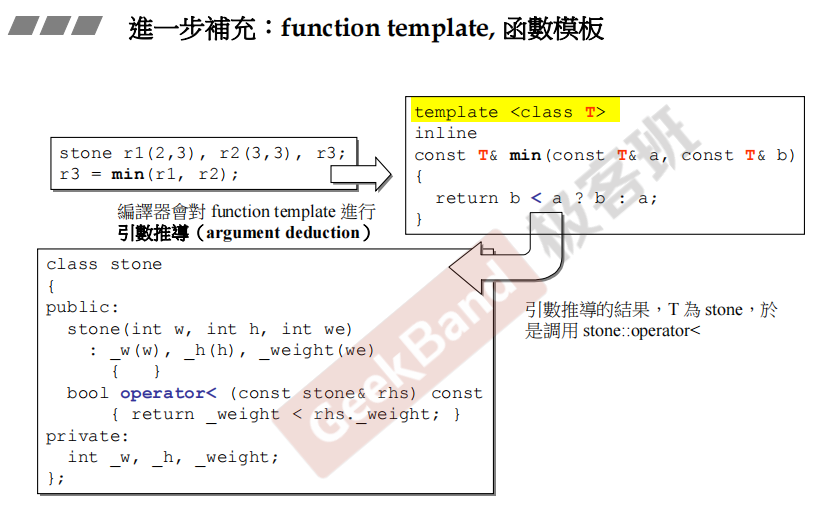
**程序结束才被释放；**

**（2）、对cout的补充**

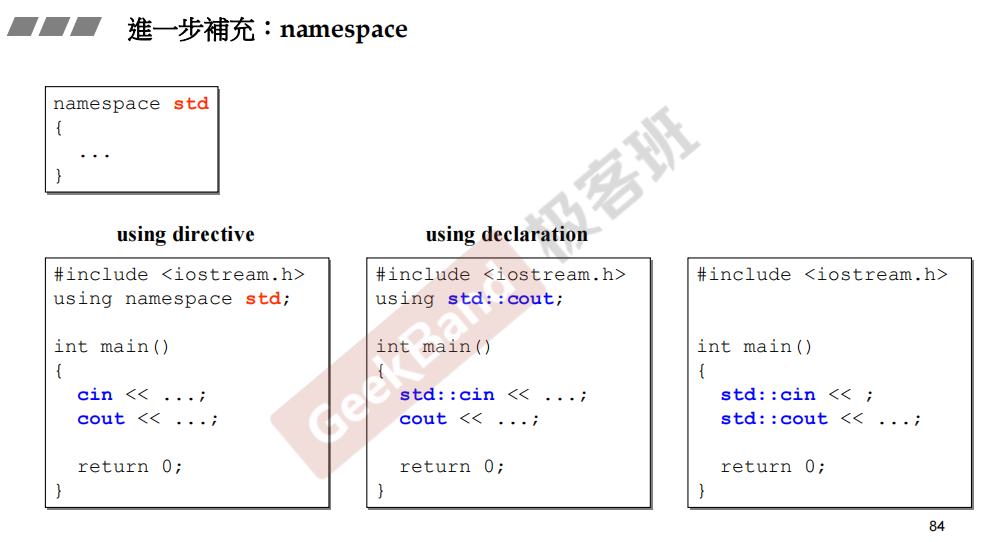


**（3）、类模板 、函数模板**



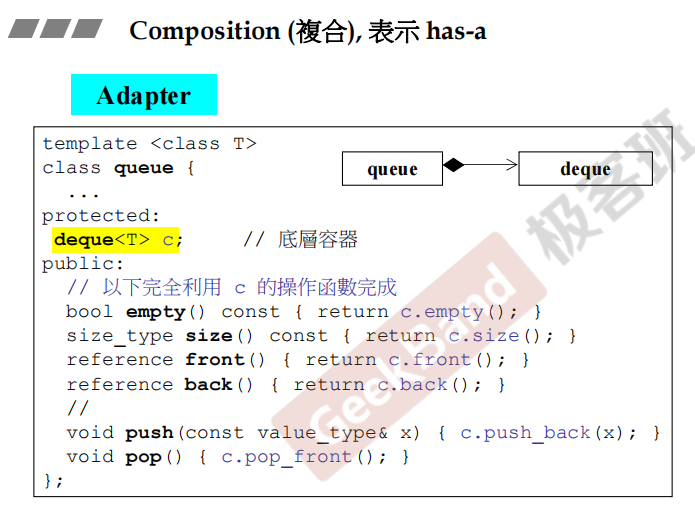


**（4）、namespace**



**11、组合与继承**

**（1）、复合Composition释义**

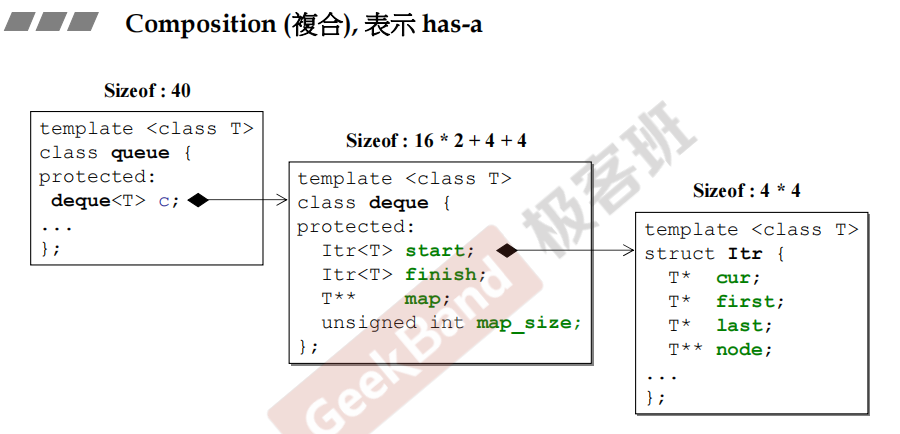


**解释：1、复合就表示自身内部有另一个完整的类结构类型，也可以有n个，如上图，queue类中含有deque, 用示意图来表示的话，就是一个黑色菱形指向包含的类结构类型；**

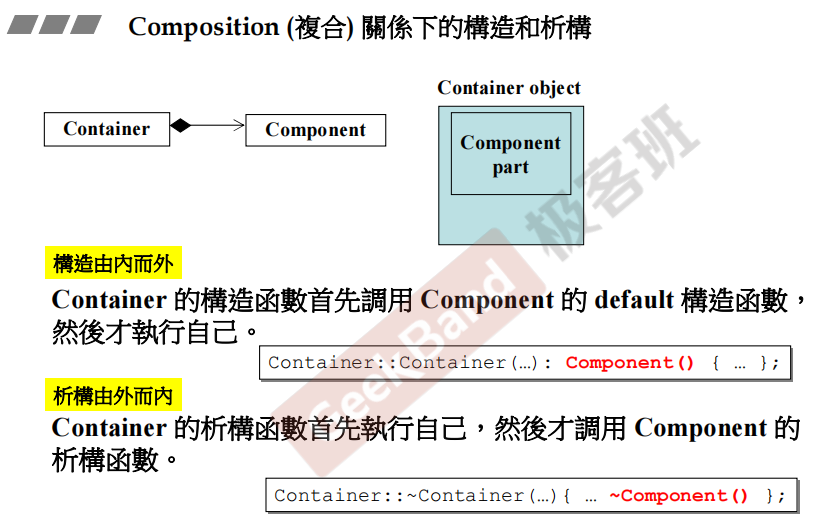
**2、上图中的Adapter是表示适配器模式，因为实际上deque拥有的功能比queue更加地强大且全面，**

**但是queue只需要其中的一部分功能，则queue复合了deque后在类中只是开放设定了deque的一部分**

**功能，deque可以很轻松地满足queue所需的功能；**



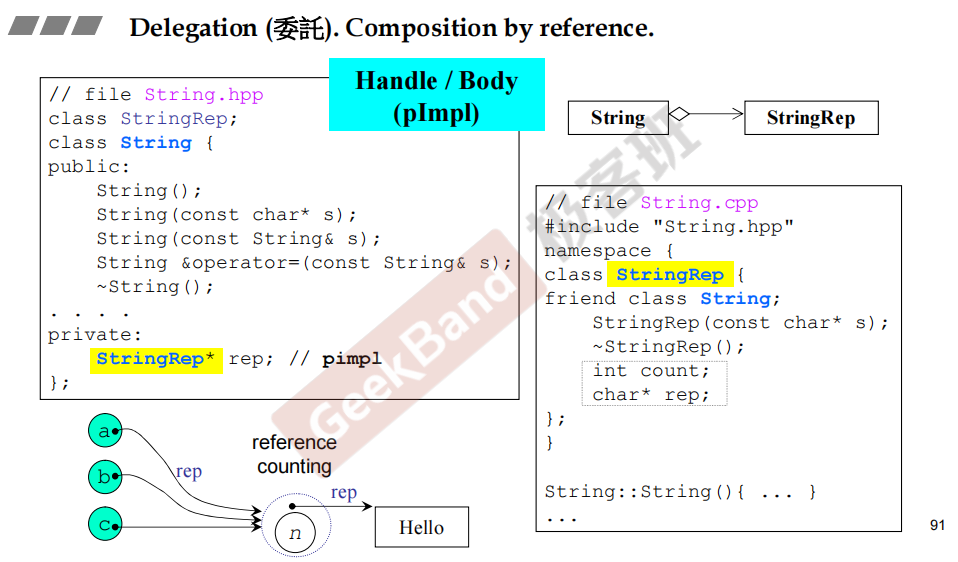
**（2）、Composition(复合)关系下的构造和析构**



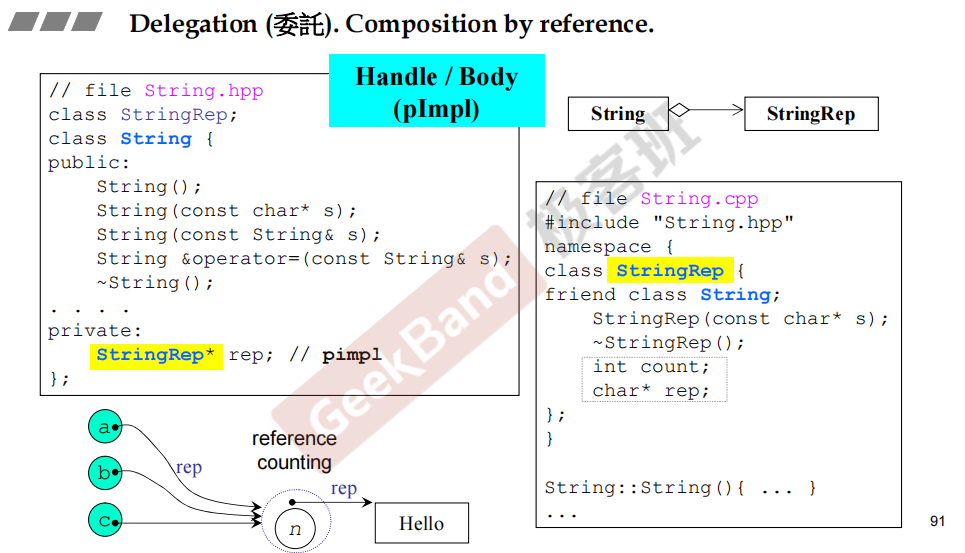
**注意：上述的红色部分都是C++编译器自动执行的，不需要我们调用，它的底层调用是如上所示而已；**

**同时因为编译器不知道要执行哪个构造函数，因此是调用默认的构造函数，但析构不用，因为析构**

**不能重载，一个类只有一个；**



**（3）、委托Delegation**



**解释；1、如果一个类中有另一个类的指针作为成员，则表示这个类委托了里面的成员类，这个类要**

**什么时候需要这个成员类时，再去new出来；把自身的一些功能委托给内部的成员类去完成；**

**2、是String类先创建出来再有内部的成员指针类的，String类的内部仅仅是提供了另一个类的指针接口，这样的话，无论图中右边的类内部怎么去修改和实现其功能，都不会影响左边的String类，很好**

**地实现了封装隔离性；**

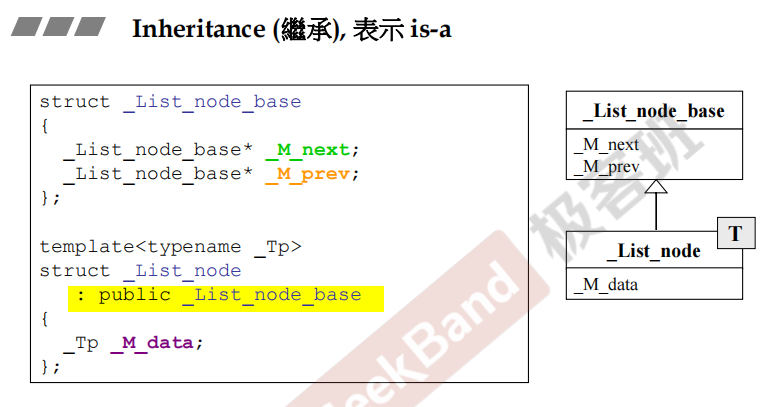
**3、上图中的a、b、c表示的是三个StringRep\* ，它们指向StringRep类的同一内容，即count和char\* rep，**

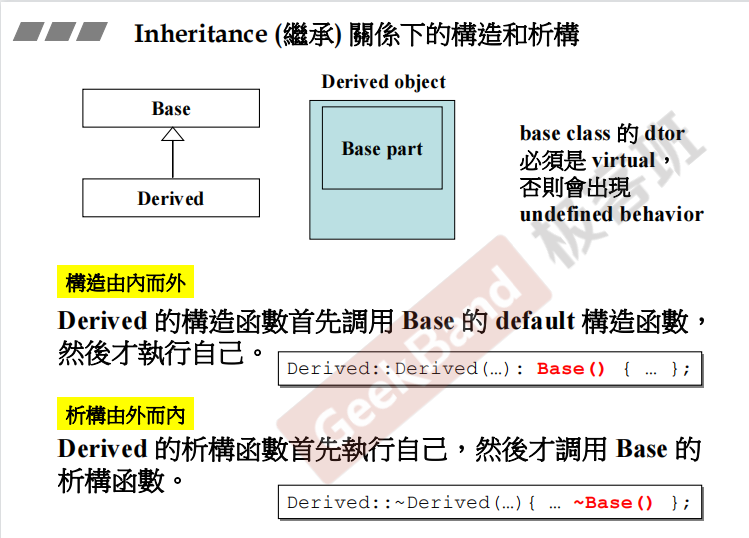
**实现了共享计数，可以节省内存空间，但共享的内容必须是同一个内容，且是共享读，如果有一个**

**指针想要更改其中的内容，则会触发copy on write的写时复制的机制，将内容复制一份给这个指针去**

**修改，而不会影响共享内容，同时计数count也要减一；**

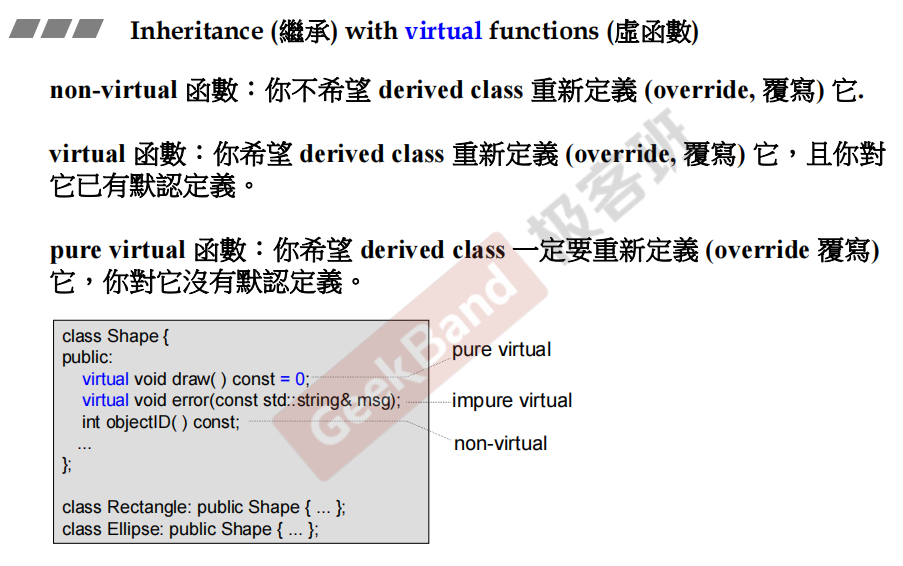
**（4）、继承**



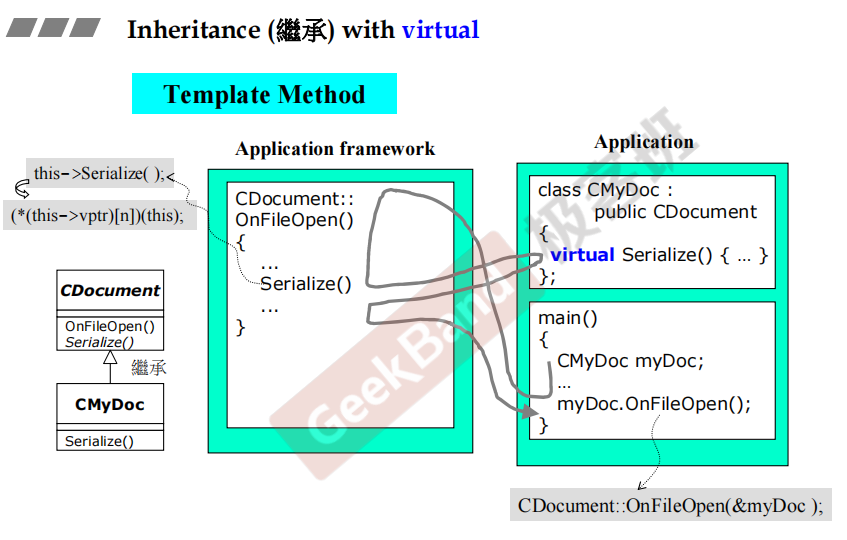


**12、虚函数和多态**

**（1）、继承和虚函数**



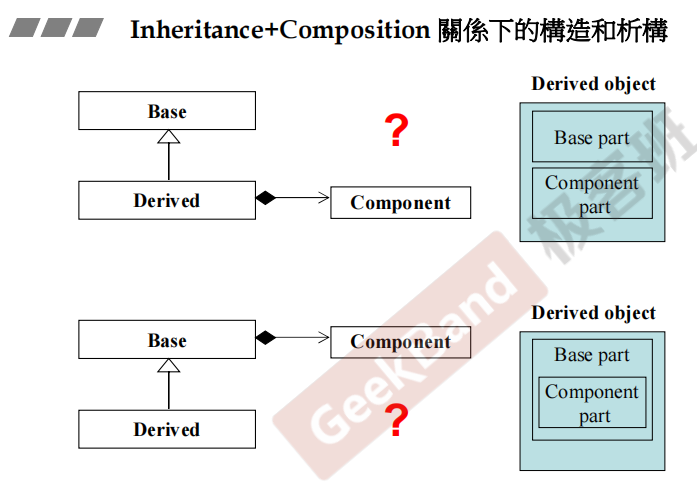
**通过子类对象调用从父类那继承的虚函数：**



**子类重写父类的虚函数，实现自己的功能：**



**（2）、继承和复合关系下的构造和析构**



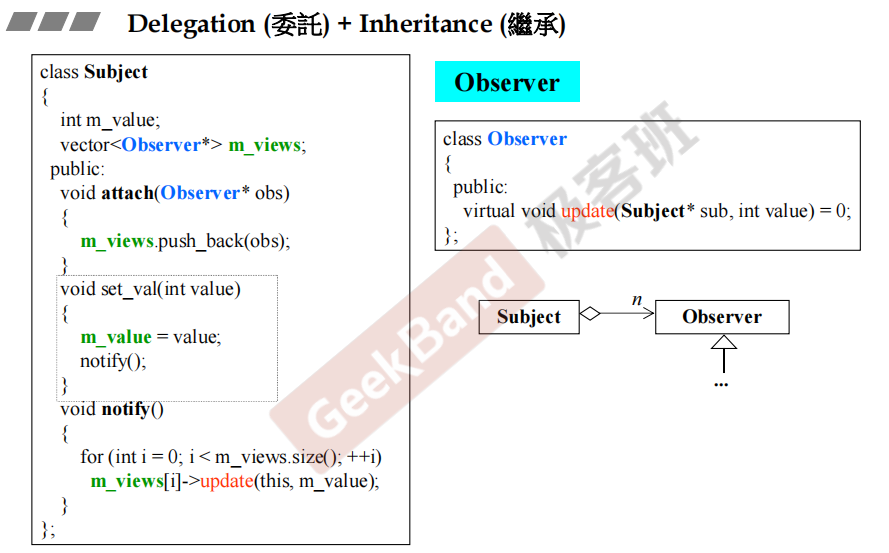
**解释：1、子类既拥有父类的数据成员，也拥有自身复合的类成员，但是父类数据部分和复合部分并不**

**清楚是谁先被构造出来的？实际上是父类的构造函数先被调用，然后才是子类自身的Component部分；**

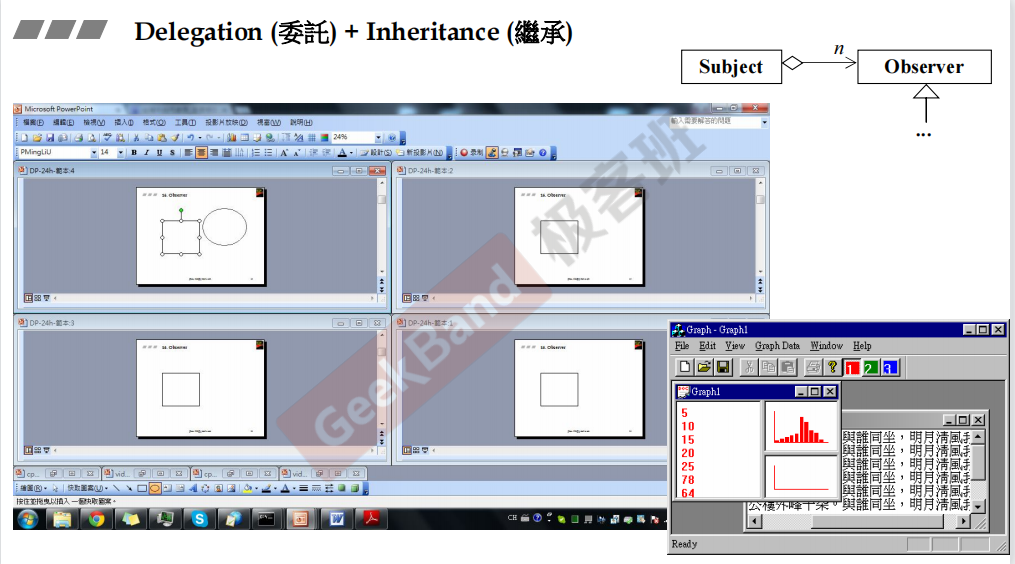
**2、上图中的下面部分表示父类中有复合的类，那么子类继承自父类后也会拥有该复合的类成员，但**

**在子类中这两部分谁先被构造出来呢？实际上先有Component part再有Base part，最后构造子类自身，而析构的顺序刚好相反；**

**（3）、委托Delegation 和 Inheritance继承**

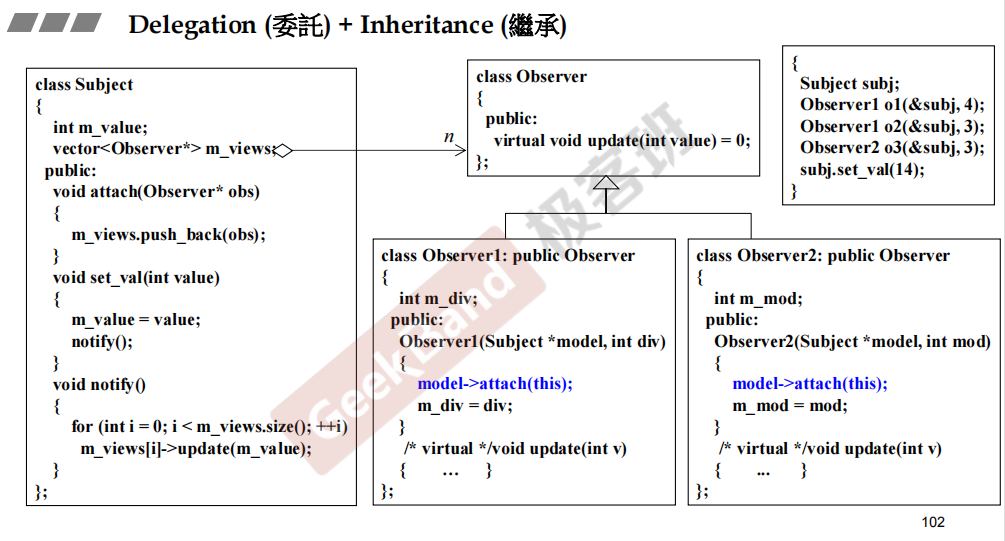


**上图中对类的设计就是为了解决下面：同时能观察多个窗口的变化，一个发生变化，其他也跟着变化**



**详细的关系和代码图如下：将类subject要实现的功能委托给一个指针类数组，这个数组中存储多个**

**观察窗口的Observer指针类，且这个Observer类是一个基类，要派生出多个不同的窗口观察子类**

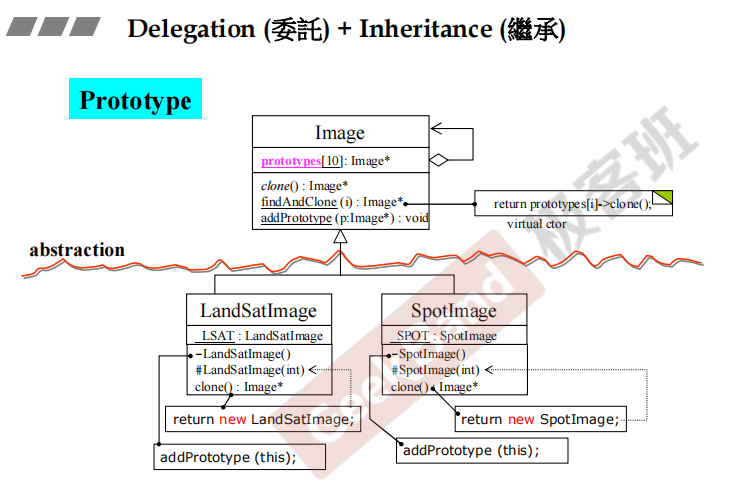


**13、委托先关的设计**

**（1）、组合Composite设计模式**



**（2）、原型Prototype设计模式**



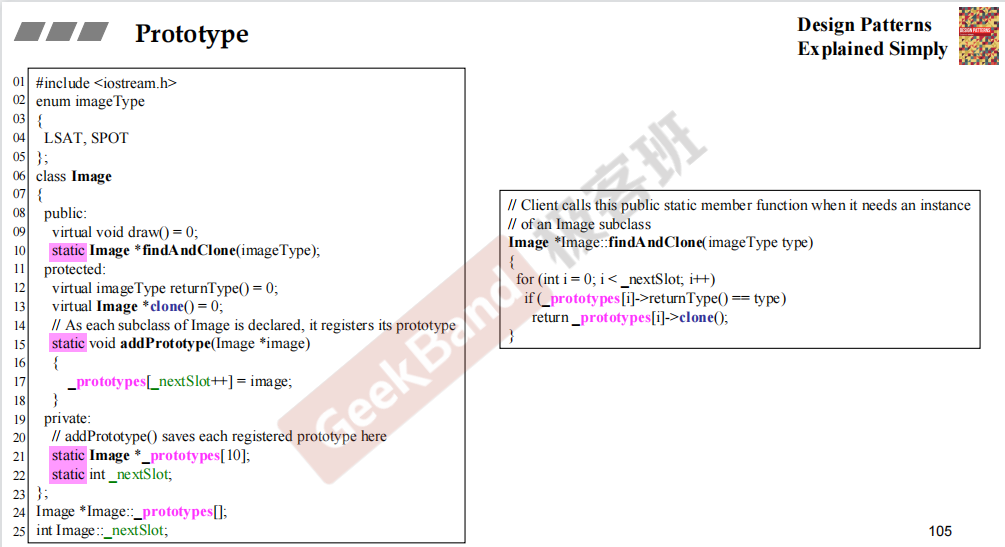
**解释：1、上图中的派生类中的 - LandSatImage()表示这是一个私有的构造函数，而#号表示是一个protected权限的函数，没有符号的第三个函数就是public的函数，LSAT表示是LandSatImage类**

**的成员名，并且有一个下划线，表示它是静态的，同时这个成员是私有的；**

**2、在派生类的构造函数中有一个addPrototype(this)函数，当构造这个派生类的对象时，就要将这个**

**类对象的原型加到父类的prototype[]数组中；**

**下图是父类的具体代码：**



**下图是子类的具体代码：**

