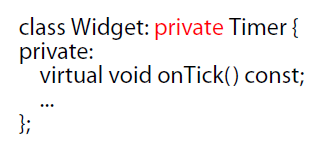
Item 39 : Use Private inheritance judiciously

Consider Example1.cpp

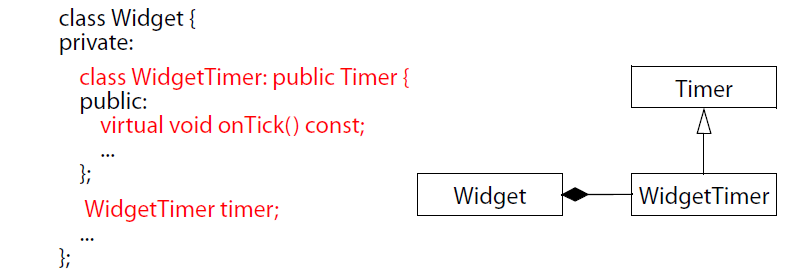
In private inheritance, you’ve just seen in action: in contrast to public inheritance, compilers will generally *not* convert a derived class object (such as Student) into a base class object (such as Person) if the inheritance relationship between the classes is private. That’s why the call to eat fails for the object s.

Private inheritance means is-implemented-in-terms-of. If you make a class D privately inherit from a class B, you do so because you are interested in taking advantage of some of the features available in class B, not because there is any conceptual relationship between objects of types B and D.

Widget to redefine a virtual function in Timer, Widget must inherit from Timer, but not public inheritace since OnTick is not conceptual part of widget. In this case we can use private inheritance but again there is a problem.

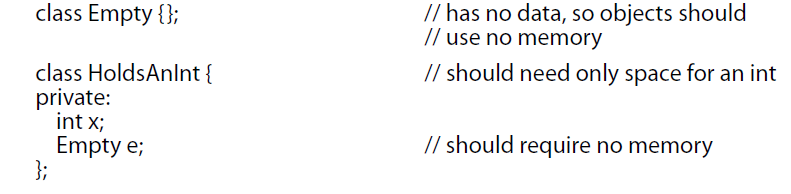


If we privately inherit from base class we will have to include header file of the base class which is an overhead. We can do the same thing with composition and public inheritance minimizing compilation dependency.

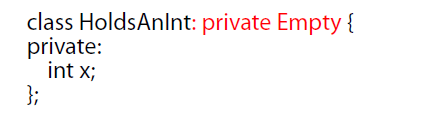


In this we can use just forward declare class WidgetTimer;

But private Inheritance is useful in case of Empty Class



But sizeof(HoldsAnInt) > sizeof(int); because compiler will give size to Empty object e as 1. Another way is use private Inheritance



Now sizeof(HoldsAnInt) == sizeof(int) since no object is created

But class Empty need not be empty, it can have non-virtual member functions and data members which we can use in HoldsAnInt.