11.8 — Virtual base classes

BY ALEX ON JANUARY 28TH, 2008 | LAST MODIFIED BY ALEX ON JANUARY 19TH, 2016

Note: This section is an advanced topic and can be skipped or skimmed if desired.

In the previous section on multiple inheritance, we left off talking about the “diamond problem”. In this section, we will resume this discussion.

Virtual base classes

Here is our example from the previous lesson, with some constructors:

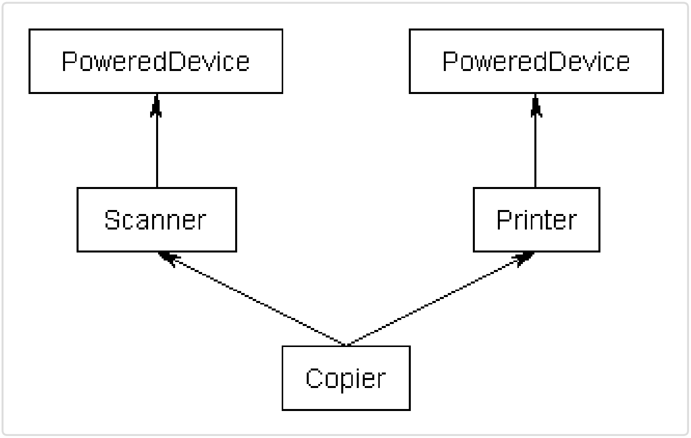
```
class PoweredDevice
{
public:
    PoweredDevice(int nPower)
    {
        cout << "PoweredDevice: " << nPower << endl;
    }
};

class Scanner: public PoweredDevice
{
public:
    Scanner(int nScanner, int nPower)
        : PoweredDevice(nPower)
    {
        cout << "Scanner: " << nScanner << endl;
    }
};

class Printer: public PoweredDevice
{
public:
    Printer(int nPrinter, int nPower)
        : PoweredDevice(nPower)
    {
        cout << "Printer: " << nPrinter << endl;
    }
};

class Copier: public Scanner, public Printer
{
public:
    Copier(int nScanner, int nPrinter, int nPower)
        : Scanner(nScanner, nPower), Printer(nPrinter, nPower)
    {
    }
};
```

If you were to create a Copier class object, by default you would end up with two copies of the PoweredDevice class -- one from Printer, and one from Scanner. This has the following structure:



We can create a short example that will show this in action:

```
int main()
{
    Copier cCopier(1, 2, 3);
}
```

This produces the result:

```
PoweredDevice: 3
Scanner: 1
PoweredDevice: 3
Printer: 2
```

As you can see, PoweredDevice got constructed twice.

While this is sometimes what you want, other times you may want only one copy of PoweredDevice to be shared by both Scanner and Printer. To share a base class, simply insert the “virtual” keyword in the inheritance list of the derived class. This creates what is called a **virtual base class**, which means there is only one base object that is shared. Here is the an example (without constructors for simplicity) showing how to use to virtual keyword to create a shared base class:

```
class PoweredDevice
{
};

class Scanner: virtual public PoweredDevice
{
};

class Printer: virtual public PoweredDevice
{
};

class Copier: public Scanner, public Printer
{
};
```

Now, when you create a Copier class, you will get only one copy of PoweredDevice that will be shared by both Scanner and Printer.

However, this leads to one more problem: if Scanner and Printer share a PoweredDevice base class, who is responsible for creating it? The answer, as it turns out, is Copier. The Copier constructor is responsible for creating PoweredDevice. Consequently, this is one time when Copier is allowed to call a non-immediate-parent constructor directly:

```
class PoweredDevice
{
public:
    PoweredDevice(int nPower)
```

```cpp
    {
        cout << "PoweredDevice: " << nPower << endl;
    }
};

class Scanner: virtual public PoweredDevice
{
public:
    Scanner(int nScanner, int nPower)
        : PoweredDevice(nPower)
    {
        cout << "Scanner: " << nScanner << endl;
    }
};

class Printer: virtual public PoweredDevice
{
public:
    Printer(int nPrinter, int nPower)
        : PoweredDevice(nPower)
    {
        cout << "Printer: " << nPrinter << endl;
    }
};

class Copier: public Scanner, public Printer
{
public:
    Copier(int nScanner, int nPrinter, int nPower)
        : Scanner(nScanner, nPower), Printer(nPrinter, nPower), PoweredDevice(nPower)
    {
    }
};
```

This time, our previous example:

```cpp
int main()
{
    Copier cCopier(1, 2, 3);
}
```

produces the result:

```
PoweredDevice: 3
Scanner: 1
Printer: 2
```

As you can see, PoweredDevice only gets constructed once.

There are a few details that we would be remiss if we did not mention.

First, virtual base classes are created before non-virtual base classes, which ensures all bases get created before their derived classes.

Second, note that the Scanner and Printer constructors still have calls to the PoweredDevice constructor. If we are creating an instance of Copier, these constructor calls are simply ignored because Copier is responsible for creating the PoweredDevice, not Scanner or Printer. However, if we were to create an instance of Scanner or Printer, the virtual keyword is ignored, those constructor calls would be used, and normal inheritance rules apply.

Third, if a class inherits one or more classes that have virtual parents, the most derived class is responsible for constructing the virtual base class. In this case, Copier inherits Printer and Scanner, both of which have a PoweredDevice virtual base class. Copier, the most derived class, is responsible for creation of PoweredDevice. Note that this is true even in a single inheritance case: if Copier was singly inherited from Printer, and Printer was virtually inherited from PoweredDevice, Copier is still responsible for creating PoweredDevice.

12.1 -- Pointers and references to the base class of derived objects

Index

11.7 -- Multiple inheritance

Share this:

Email | Facebook 5 | Twitter | Google | Pinterest

C++ TUTORIAL | PRINT THIS POST

30 comments to 11.8 — Virtual base classes

prABU
June 11, 2008 at 9:18 pm · Reply

i was not satisfied with ur explian, i need more definitions for vitual base classes

bookworm
August 10, 2009 at 3:27 pm · Reply

You should not read advanced topics before you fully understand fundamental stuff.

sandhya
July 8, 2008 at 2:05 am · Reply

Hi Alex,