Inheritance

Multiple Inheritance?

- A class can be defined based on more than one existing class
  - Members from all base classes are inherited
- Class to represent an invoice

```
class Invoice
{
private:
    int  number;
    int  amount;
    bool paid;
    string customer;
public:
    Invoice( int numberVal, int amountVal,  const char * pCustomer );
    void PrintDetails();
};
```

Example

- InvoicedPlumbJob
  - Based on PlumbJob and Invoice

```
class InvoicedPlumbJob : public PlumbJob, public Invoice
{
public:
  InvoicedPlumbJob(   int numberVal, int amountVal,  const char * pCustomer,
                      int hoursVal, int rateVal, int jobNumVal, int calloutVal );
};
```

```
InvoicedPlumbJob::InvoicedPlumbJob(  int numberVal, int amountVal,  const char * pCustomer,
                                     int hoursVal, int rateVal, int jobNumVal, int calloutVal )
 : PlumbJob( hoursVal, rateVal, jobNumVal, calloutVal ),
   Invoice( numberVal, amountVal, pCustomer )
{...
}
```

The constructor calls both base initializers

This example is straightforward as there are no common base classes below PlumbJob or Invoice, and no conflicting member names

Example

- A Decorating job with an associated materials cost

```
class DecorJob : public Job
{
private:
    int materials;
public:
    void SetMaterials( int materialsVal ) { materials = materialsVal; }
    int  GetAmount() const { return Job::GetAmount() + materials; }
};
```

- A job with a decorating component and a plumbing component

```
class PlumbDecorJob : public PlumbJob, public DecorJob
{
    ...
};
```

Using the Class

- Implement a member function GetAmount()

```
class PlumbDecorJob : public PlumbJob, public DecorJob
{
public:
 int GetAmount() const { return PlumbJob::GetAmount() + DecorJob::GetAmount(); }
};
```

Calls the GetAmount member of each of the base classes and sums the result

Implementing GetAmount this way looks good, and compiles correctly, but is wrong

The GetAmount functions in the base classes both include hours * rate, which will be added in twice

Conclusions - Multiple Inheritance needs careful thought

More Problems

- Using the SetRate() member function
  - Defined in Job class

```
PlumbDecorJob pdjob;
pdjob.SetRate( 16 );
```

Gives the compiler error (MS Visual C++):

'PlumbDecorJob::SetRate' is ambiguous. Could be the 'SetRate' in base 'Job' of base 'PlumbJob' of class 'PlumbDecorJob' or the 'SetRate' in base 'Job' of base 'DecorJob' of class 'PlumbDecorJob'

We have a problem because we are inheriting from other classes which have a common base class

This happens often in practice, for example most MFC classes are derived from CObject

Resolving Ambiguity

- Use scoping operator :: to determine which base class should implement the function

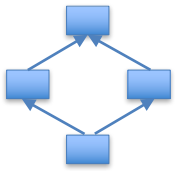
```
PlumbDecorJob pdjob;
pdjob.PlumbJob::SetRate( 16 );
```

This could be achieved more automatically by creating an override for SetRate in PlumbDecorJob which just calls PlumbJob::SetRate

Although the compiler accepts these solutions, they won't work in practice as we still have a second, uninitialized copy of the rate under the DecorJob class.

Resolving Ambiguity

- This problem is caused by deriving from 2 base classes which derive from a common base class
  - Two copies of the common base class
  - "Diamond of Death"
  - May be solved using "virtual inheritance"



```
class PlumbJob : public virtual Job
{ ...

class DecorJob : public virtual Job
{ ...

class PlumbDecorJob : public PlumbJob, public DecorJob
{ ...
```

Because PlumbJob and DecorJob use Job as a virtual base class, they both share the same copy of Job when used together in PlumbDecorJob

This is very rarely a good solution

Remember we still haven't solved the GetAmount problem!

Conclusions

- Multiple inheritance may be useful
  - Rarely essential
- Requires careful design consideration, as hidden complications often arise
  - As shown
- Common base classes cause problems
  - Makes multiple inheritance difficult to use with many class libraries
- Solutions often require knowledge of ancestor classes and their derivation
  - Should only really be interested in immediate base classes