subtleties of implicit and explicit interface implementations. C# does not support multiple inheritance, but a class has the option of implementing one or more interfaces. One challenge with interfaces is that they may include methods that have the same signatures as existing class members or members of other interfaces. Explicit interface implementations can be used to disambiguate class and interface methods that would otherwise conflict. Explicit interfaces can also be used to hide the details of an interface that the class developer considers private.

**Disambiguating Methods**

Let's look at an example of method disambiguation. In Listing 1 we have started to write a class called **C** that implements interfaces **I1** and **I2**, each of which defines a method **A()**.

**Listing 1. Class C implements interfaces I1 and I2.**

interface I1

{

void A();

}

interface I2

{

void A();

}

class C : I1, I2

{

public void A()

{

Console.WriteLine("C.A()");

}

}

In this case, **A()** is a public class member that *implicitly* implements a member of *both* interfaces. **A()** can be invoked through either interface or through the class itself as follows:

**Listing 2. A() can be invoked from I1, I2, or C.**

C c = new C();

I1 i1 = (I1)c; //even if do not type cast will it work???? yes

I2 i2 = (I2)c;

i1.A();

i2.A();

c.A();

The output from this code is

C.A()

C.A()

C.A()

This works fine if you want **A()** to do the same thing in both interfaces. In most cases, however, methods in different interfaces have distinct purposes requiring wholly different implementations. This is where explicit interface implementations come in handy. To explicitly implement an interface member, just use its fully qualified name in the declaration. A fully qualified interface name takes the form

InterfaceName.MemberName

In Listing 3 we add an explicit implementation of **I1**'s **A()** method.

**Listing 3. Class C explicitly implements I1.A().**

class C : I1, I2

{

public void A() //observe it is public and member of class

{

Console.WriteLine("C.A()");

}

void I1.A() //observe private and not member of class

{

Console.WriteLine("I1.A()");

}

}

C c = new C();

I1 i1 = (I1)c; //even if do not type cast will it work???? yes

I2 i2 = (I2)c;

i1.A();

i2.A();

c.A();

Now when we run the statements from Listing 2 we get

I1.A()

C.A()

C.A()

When an interface method is explicitly implemented, it is no longer visible as a public member of the class. The only way to access it is through the interface. As an example, suppose we deleted the implicit implementation of **A()** as shown in Listing 4:

**Listing 4. Class C does not implicitly implement A()**

class C : I1, I2

{

void I1.A()

{

Console.WriteLine("I2.A()");

}

}

In this case we would get a compile error saying that **C** fails to implement **I2.A()**. We could fix this error by changing the first line to

class C : I1

but we'd get another compile error when trying to invoke **A()** as a member of **C**:

C c = new C();

c.A();

This time the compiler would report that class **C** does not contain a definition for method **A()**. We get the error because the **explicit implementation of I1.A() hides A() from the class**. The only way to call **I1.A()** now is through **C**'s **I1** interface:

C c = new C();

I1 i1 = (I1)c; //even if I do not typecast will it work???

i1.A();