C# nested classes are like C++ nested classes, not Java inner classes

devblogs.microsoft.com/oldnewthing/20060801-19

August 1, 2006

Raymond Chen

When you declare a class inside another class, the inner class still acts like a regular class. The nesting controls access and visibility, but not behavior. In other words, all the rules you learned about regular classes also apply to nested classes.

The `this` keyword in an instance methods of a class (nested or not) can be used to access members of that class and only those members. It cannot be used to access members of other classes, at least not directly. (And the `this` can be omitted when it would not result in ambiguity.) You create an instance of a class (nested or not) by saying `new ClassName(...)` where `...` are the parameters to an applicable class constructor.

Java nested classes behave the same way, but Java also has the concept of inner classes. To construct an instance of an inner class in Java, you write `new o.InnerClass(...)` where `...` as before are the parameters to an applicable class constructor. The `o` in front is an expression that evaluates to an object whose type is that of the outer class. The inner class can then use the `this` keyword to access its own members as well as those of the instance of the outer class to which it was bound.

In C++ and C#, you will have to implement this effect manually. It's not hard, though:

```
// Java
class OuterClass {
 string s;
 // ...
 class InnerClass {
  public InnerClass() { }
  public string GetOuterString() {
return s; }
 }
 void SomeFunction() {
  InnerClass i = new this.InnerClass();
  i.GetOuterString();
 }
}
```

```
// C#
class OuterClass {
 string s;
 // ...
 class InnerClass {
  OuterClass o_;
  public InnerClass(OuterClass o) { o_ =
o; }
  public string GetOuterString() { return
o_.s; }
 }
 void SomeFunction() {
  InnerClass i = new InnerClass(this);
  i.GetOuterString();
 }
}
```

In Java, the inner class has a secret `this$0` member which remembers the instance of the outer class to which it was bound. Creating an instance of an inner class via the `new o.InnerClass(...)` notation is treated as if you had written `new InnerClass(o, ...)`, where `o` is automatically assigned to the secret `this$0` member, and attempts to access members of the outer class are automatically treated as if they were written `this$0.outermember`. (This description of how inner classes are implemented is not just conceptual. It is spelled out in the language specification.)

The C# equivalent to this code merely makes explicit the transformation that in Java was implicit. We give the inner class a reference to the outer class (here, we called it `o_`) and pass it as an explicit parameter to the inner class's constructor. And when we want to access a member of that outer class, we use `o_` to do it.

In other words, Java inner classes are syntactic sugar that is not available to C#. In C#, you have to do it manually.

If you want, you can create your own sugar:

```
class OuterClass {
 ...
 InnerClass NewInnerClass() {
  return new InnerClass(this);
 }
 void SomeFunction() {
  InnerClass i = this.NewInnerClass();
  i.GetOuterString();
 }
}
```

Where you would want to write in Java `new o.InnerClass(...)` you can write in C# either `o.NewInnerClass(...)` or `new InnerClass(o, ...)`. Yes, it's just a bunch of moving the word `new` around. Like I said, it's just sugar.

Now, I'm not saying that the Java way of representing inner classes isn't useful. It's a very nice piece of sugar if you access the outer class's members frequently from the inner class. However, it's not the type of transformation that makes you say, "Well, if a language doesn't support this, it's too hard for me to implement it manually, so I'll just give up." The conversion is not that complicated and consists entirely of local changes that can be performed without requiring a lot of thought.

As a postscript, my colleague Eric Lippert points out that JScript.NET does have instance-bound inner classes.

```
class Outer {
 var s;
 class Inner {
  function GetOuterString() {
   return s;
  }
 }
}
var o = new Outer();
o.s = "hi";
var i = new o.Inner();
i.GetOuterString();
```

Raymond Chen

Follow