**Attributes**

* C# allows you to add declarative information to a program in the form of an attribute.
* An attribute defines additional information (metadata) that is associated with a class, structure, method, and so on.
* For example, you might define an attribute that determines the type of button that a class will display.
* Attributes are specified between square brackets, preceding the item to which they apply. Thus, an attribute is not a member of a class. Rather, an attribute specifies supplemental information that is attached to an item.

**Attribute Basics :**

* An attribute is supported by a class that inherits System.Attribute.
* Thus, all attribute classes must be subclasses of Attribute.
* Although Attribute defines substantial functionality, this functionality is not always needed when working with attributes. By convention, attribute
* classes often use the suffix Attribute.
* For example, ErrorAttribute would be a name for an attribute class that described an error.
* When an attribute class is declared, it is preceded by an attribute called AttributeUsage.
* This built-in attribute specifies the types of items to which the attribute can be applied. Thus, the usage of an attribute can be restricted to methods, for example.

**Creating an Attribute:**

* In an attribute class, you will define the members that support the attribute.
* Often attribute classes are quite simple, containing just a small number of fields or properties.
* For example, an attribute might define a remark that describes the item to which the attribute is being attached. Such an attribute might look like this:

[AttributeUsage(AttributeTargets.All)]

public class RemarkAttribute : Attribute

{

string pri\_remark; // underlies Remark property

public RemarkAttribute(string comment)

{

pri\_remark = comment;

}

public string Remark

{

get

{

return pri\_remark;

}

}

}

**Attaching an Attribute:**

* Once you have defined an attribute class, you can attach the attribute to an item.
* An attribute precedes the item to which it is attached and is specified by enclosing its constructor inside square brackets.
* For example, here is how RemarkAttribute can be associated with a class:

[RemarkAttribute("This class uses an attribute.")]

class UseAttrib

{

// ...

}

* This constructs a RemarkAttribute that contains the comment, “This class uses an attribute.”
* This attribute is then associated with UseAttrib.
* When attaching an attribute, it is not actually necessary to specify the Attribute suffix.
* For example, the preceding class could be declared this way:

[Remark("This class uses an attribute.")]

class UseAttrib

{,

// ...

}

**Obtaining an Object’s Attributes**

* Once an attribute has been attached to an item, other parts of the program can retrieve the attribute.
* To retrieve an attribute, you will usually use one of two methods.
* The first is GetCustomAttributes( ), which is defined by MemberInfo and inherited by Type. It retrieves a list of all attributes attached to an item.
* Here is one of its forms:
  + object[ ] GetCustomAttributes(bool searchBases)
* If searchBases is true, then the attributes of all base classes through the inheritance chain will be included. Otherwise, only those classes defined by the specified type will be found.
* The second method is GetCustomAttribute( ), which is defined by Attribute. One of its forms is shown here:
  + static Attribute GetCustomAttribute(MemberInfo mi, Type attribtype)
* Here, mi is a MemberInfo object that describes the item for which the attributes are being obtained.
* The attribute desired is specified by attribtype.
* You will use this method when you know the name of the attribute you want to obtain, which is often the case.
* For example, assuming that the UseAttrib class has the RemarkAttribute, to obtain a reference to the RemarkAttribute, you can use a sequence like this:

// Get a MemberInfo instance associated with a

// class that has the RemarkAttribute.

Type t = typeof(UseAttrib);

// Retrieve the RemarkAttribute.

Type tRemAtt = typeof(RemarkAttribute);

RemarkAttributera = (RemarkAttribute)

Attribute.GetCustomAttribute(t, tRemAtt);

* This sequence works because MemberInfo is a base class of Type. Thus, t is a MemberInfoinstance.
* Once you have a reference to an attribute, you can access its members.
* This makes information associated with an attribute available to a program that uses an element to which an attribute is attached.

For example, the following statement displays the Remark

property:

Console.WriteLine(ra.Remark);

The following program puts together all of the pieces and demonstrates the use of

RemarkAttribute:

// A simple attribute example.

using System;

using System.Reflection;

[AttributeUsage(AttributeTargets.All)]

public class RemarkAttribute : Attribute

{

string pri\_remark; // underlies Remark property

public RemarkAttribute(string comment)

{

pri\_remark = comment;

}

public string Remark

{

get

{

return pri\_remark;

}

}

}

[RemarkAttribute("This class uses an attribute.")]

class UseAttrib {

// ...

}

class AttribDemo

{

static void Main()

{

Type t = typeof(UseAttrib);

Console.Write("Attributes in " + t.Name + ": ");

object[] attribs = t.GetCustomAttributes(false);

foreach(object o in attribs)

{

Console.WriteLine(o);

}

Console.Write("Remark: ");

// Retrieve the RemarkAttribute.

Type tRemAtt = typeof(RemarkAttribute);

RemarkAttributera = (RemarkAttribute)

Attribute.GetCustomAttribute(t, tRemAtt);

Console.WriteLine(ra.Remark);

}

}