**Using Reflection**

* Using **Type**’s methods and properties, it is possible to obtain detailed information about a type at runtime. This is an extremely powerful feature, because once you have obtained information about a type, you can invoke its constructors, call its methods, and use its properties. Thus, reflection enables you to use code that was not available at compile time.
* reflection techniques: obtaining information about methods, invoking methods, constructing objects, and loading types from assemblies.

**Obtaining Information About Methods**

* Once you have a Type object, you can obtain a list of methods supported by the type by using GetMethods( ). One form is shown here:
  + - MethodInfo[ ] GetMethods( )
* It returns an array of MethodInfo objects that describe the methods supported by the invoking type.
* MethodInfo is in the System.Reflection namespace.
* MethodInfo is derived from the abstract class MethodBase, which inherits MemberInfo.
* Thus, the properties and methods defined by all three of these classes are available for your use.
* For example, to obtain the name of a method, use the Name property. Two members that are of particular interest at this time are ReturnType and GetParameters( ).
* The return type of a method is found in the read-only ReturnType property, which is an object of Type.
* The method GetParameters( ) returns a list of the parameters associated with a method.
* It has this general form:
  + - ParameterInfo[ ] GetParameters( );
* The parameter information is held in a ParameterInfo object.
* ParameterInfo defines a large number of properties and methods that describe the parameter.
* Two properties that are of particular value are Name, which is a string that contains the name of the parameter, and ParameterType, which describes the parameter’s type.
* The parameter’s type is encapsulated within a Type object.
* Here is a program that uses reflection to obtain the methods supported by a class called MyClass.
* For each method, it displays the return type and name of the method, and the name and type of any parameters that each method may have.

// Analyze methods using reflection.

using System;

using System.Reflection;

class MyClass

{

int x;

int y;

public MyClass(inti, int j)

{

x = i;

y = j;

}

public int Sum()

{

return x+y;

}

public bool IsBetween(inti)

{

if(x <i&&i< y)

return true;

else

return false;

}

public void Set(int a, int b)

{

x = a;

y = b;

}

public void Set(double a, double b)

{

x = (int) a;

y = (int) b;

}

public void Show()

{

Console.WriteLine(" x: {0}, y: {1}", x, y);

}

}

class ReflectDemo

{

public static void Main(String[] args)

{

Type t = typeof(MyClass); // get a Type object representing MyClass

Console.WriteLine("Analyzing methods in " + t.Name);

Console.WriteLine();

Console.WriteLine("Methods supported: ");

MethodInfo[] mi = t.GetMethods();

// Display methods supported by MyClass.

foreach(MethodInfo m in mi)

{

// Display return type and name.

Console.Write(" " + m.ReturnType.Name +" " + m.Name + "(");

// Display parameters.

ParameterInfo[] pi = m.GetParameters();

for(inti=0; i<pi.Length; i++) {

Console.Write(pi[i].ParameterType.Name +" " + pi[i].Name);

if(i+1 <pi.Length) Console.Write(", ");

Console.WriteLine(")");

Console.WriteLine();

}

}

}

***A Second Form of GetMethods( )***

A second form of GetMethods( ) lets you specify various flags that filter the methods that are retrieved. It has this general form:

MethodInfo[ ] GetMethods(BindingFlagsflags)

This version obtains only those methods that match the criteria you specify. BindingFlags is an enumeration. Here are several commonly used values:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| DeclaredOnly | Retrieves only those methods defined by the specified class. Inherited methods are not included |
| Instance | Retrieves instance methods |
| NonPublic | Retrieves nonpublic methods |
| Public | Retrieves public methods |
| Static | Retrieves static methods. |

* You can OR together two or more flags.
* In fact, minimally you must include either Instance or Static with Public or NonPublic. Failure to do so will result in no methods being retrieved.
* One of the main uses of the BindingFlags form of GetMethods( ) is to enable you to obtain a list of the methods defined by a class without also retrieving the inherited methods.
* This is especially useful for preventing the methods defined by object from being obtained.
* For example, try substituting this call to GetMethods( ) into the preceding program:

// Now, only methods declared by MyClass are obtained.

MethodInfo[] mi = t.GetMethods(BindingFlags.DeclaredOnly |

BindingFlags.Instance |

BindingFlags.Public) ;