**Dynamic dispatch**

When a method is invoked on an object, the object itself determines what code gets executed by looking up the method at run time in a table associated with the object. This feature distinguishes an object from an abstract data type (or module), which has a fixed (static) implementation of the operations for all instances. It is a programming methodology that gives modular component development while at the same time being very efficient.

**C#**

The **dynamic** type enables the operations in which it occurs to bypass compile-time type checking. Instead, these operations are resolved at run time. The **dynamic** type simplifies access to COM APIs such as the Office Automation APIs, and also to dynamic APIs such as IronPython libraries, and to the HTML Document Object Model (DOM).

Type **dynamic** behaves like type **object** in most circumstances. However, operations that contain expressions of type **dynamic** are not resolved or type checked by the compiler. The compiler packages together information about the operation, and that information is later used to evaluate the operation at run time. As part of the process, variables of type **dynamic** are compiled into variables of type **object**. Therefore, type **dynamic** exists only at compile time, not at run time.

using System;

class MyClass

{

public int MyMethod()

{

return 100;

}

}

class Program

{

static void Main(string[] args)

{

object o = new MyClass();

Type t = o.GetType();

object result = t.InvokeMember("MyMethod", System.Reflection.BindingFlags.InvokeMethod, null, o, new object[] { });

int i = Convert.ToInt32(result);

Console.WriteLine("i = " + i);

/\*

Dynamic Dispatch

C# 4.0 supports dynamic late-binding.

With the dynamic keyword, instead of calling a method MyMethod

on some object using reflection in this manner,

you can now tell the compiler to please treat

o i.e., d as dynamic and delay all analysis until run time.

\*/

dynamic d = new MyClass();

int j = d.MyMethod();

Console.WriteLine("j = " + j);

Console.ReadKey();

}

}