**Method Overloading:**

* In C#, two or more methods within the same class can share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be *overloaded,* and the process is referred to as *method overloading.*
* Method overloading is one of the ways that C# implements polymorphism.
* In general, to overload a method, simply declare different versions of it.
* The compiler takes care of the rest.
* You must observe one important restriction: The type and/or number of the parameters of each overloaded method must differ. It is not sufficient for two methods to differ only in their return types. They must differ in the types or number of their parameters. method to use.) Of course, overloaded methods *may* differ in their return types, too.
* When an overloaded method is called, the version of the method executed is the one whose parameters match the arguments.
* Here is a simple example that illustrates method overloading:

// Demonstrate method overloading.

using System;

class Overload

{

public void OvlDemo()

{

Console.WriteLine("No parameters");

}

// Overload OvlDemo for one integer parameter.

public void OvlDemo(int a)

{

Console.WriteLine("One parameter: " + a);

}

// Overload OvlDemo for two integer parameters.

public int OvlDemo(int a, int b)

{

Console.WriteLine("Two parameters: " + a + " " + b);

return a + b;

}

// Overload OvlDemo for two double parameters.

public double OvlDemo(double a, double b)

{

Console.WriteLine("Two double parameters: " + a + " "+ b);

return a + b;

}

}

class OverloadDemo

{

public static void Main()

{

Overload ob = new Overload();

int resI;

double resD;

// Call all versions of OvlDemo().

ob.OvlDemo();

Console.WriteLine();

ob.OvlDemo(2);

Console.WriteLine();

resI = ob.OvlDemo(4, 6);

Console.WriteLine("Result of ob.OvlDemo(4, 6): " + resI);

Console.WriteLine();

resD = ob.OvlDemo(1.1, 2.32);

Console.WriteLine("Result of ob.OvlDemo(1.1, 2.32): " + resD);

}

}

**This program generates the following output:**

No parameters

One parameter: 2

Two parameters: 4 6

Result of ob.OvlDemo(4, 6): 10

Two double parameters: 1.1 2.32

Result of ob.OvlDemo(1.1, 2.32): 3.42

* As you can see, **OvlDemo( )** is overloaded four times. The first version takes no parameters; the second takes one integer parameter; the third takes two integer parameters; and the fourth takes two **double** parameters.
* Notice that the first two versions of **OvlDemo( )** return **void** and the second two return a value.
* This is perfectly valid, but as explained, overloading is not affected one way or the other by the return type of a method.
* Thus, attempting to use these two versions of **OvlDemo( )** will cause an error:

// One OvlDemo(int) is OK.

public void OvlDemo(int a)

{

Console.WriteLine("One parameter: " + a);

}

/\* Error! Two OvlDemo(int)s are not OK even though return types differ. \*/

public int OvlDemo(int a)

{

Console.WriteLine("One parameter: " + a);

return a \* a;

}

* As the comments suggest, the difference in their return types is an insufficient difference for the purposes of overloading.
* As you will recall from Chapter 3, C# provides certain implicit (i.e., automatic) type conversions.
* These conversions also apply to parameters of overloaded methods.
* For example, consider the following:

// Implicit type conversions can affect overloaded method resolution.

using System;

class Overload2

{

public void MyMeth(int x)

{

Console.WriteLine("Inside MyMeth(int): " + x);

}

public void MyMeth(double x)

{

Console.WriteLine("Inside MyMeth(double): " + x);

}

}

class TypeConv

{

public static void Main()

{