Métodos

Un método es un bloque de código que contiene una serie de instrucciones. Un programa hace que se ejecuten las instrucciones al llamar al método y especificando los argumentos de método necesarios. En C#, todas las instrucciones ejecutadas se realizan en el contexto de un método. El método Main es el punto de entrada para cada aplicación de C# y se llama mediante Common Language Runtime (CLR) cuando se inicia el programa.

Firmas de método

Los métodos se declaran en una clase o struct especificando el nivel de acceso, como public o private, modificadores opcionales como abstract o sealed, el valor de retorno, el nombre del método y cualquier parámetro de método. Todas estas partes forman la firma del método.

Un tipo de valor devuelto de un método no forma parte de la firma del método con el objetivo de sobrecargar el método. Sin embargo, forma parte de la firma del método al determinar la compatibilidad entre un delegado y el método que señala.

Los parámetros de método se encierran entre paréntesis y se separan por comas. Los paréntesis vacíos indican que el método no requiere parámetros. Esta clase contiene cuatro métodos:

```
abstract class Motorcycle
{
    // Anyone can call this.
    public void StartEngine() {/* Method statements here */ }

    // Only derived classes can call this.
    protected void AddGas(int gallons) { /* Method statements here */ }

    // Derived classes can override the base class implementation.
    public virtual int Drive(int miles, int speed) { /* Method statements
here */ return 1; }

    // Derived classes must implement this.
    public abstract double GetTopSpeed();
}
```

Acceso a métodos

Llamar a un método en un objeto es como acceder a un campo. Después del nombre del objeto, agregue un punto, el nombre del método y paréntesis. Los argumentos se enumeran entre paréntesis y están separados por comas. Los métodos de la clase `Motorcycle` se pueden llamar como en el ejemplo siguiente:

```
class TestMotorcycle : Motorcycle
{

    public override double GetTopSpeed()
    {
        return 108.4;
    }

    static void Main()
    {

        TestMotorcycle moto = new TestMotorcycle();

        moto.StartEngine();
        moto.AddGas(15);
        moto.Drive(5, 20);
        double speed = moto.GetTopSpeed();
        Console.WriteLine("My top speed is {0}", speed);
    }
}
```

Parámetros de métodos frente a Argumentos

La definición del método especifica los nombres y tipos de todos los parámetros necesarios. Si el código de llamada llama al método, proporciona valores concretos denominados argumentos para cada parámetro. Los argumentos deben ser compatibles con el tipo de parámetro, pero el nombre del argumento (si existe) utilizado en el código de llamada no tiene que ser el mismo que el parámetro con nombre definido en el método. Por ejemplo:

```
 public void Caller()
 {
     int numA = 4;
     // Call with an int variable.
     int productA = Square(numA);

     int numB = 32;
     // Call with another int variable.
     int productB = Square(numB);

     // Call with an integer literal.
     int productC = Square(12);
```

```
    // Call with an expression that evaulates to int.
    productC = Square(productA * 3);
}

int Square(int i)
{
    // Store input argument in a local variable.
    int input = i;
    return input * input;
}
```

Valores devueltos

Los métodos pueden devolver un valor al autor de llamada. Si el tipo de valor devuelto, el tipo enumerado antes del nombre de método, no es `void`, el método puede devolver el valor mediante la utilización de la palabra clave `return` . Una instrucción con la palabra clave `return` seguida de un valor que coincide con el tipo de valor devuelto devolverá este valor al autor de llamada del método.

El valor puede devolverse al autor de la llamada mediante valor o, a partir de C# 7.0, mediante referencia. Los valores se devuelven al autor de la llamada mediante referencia si la palabra clave ref se usa en la firma del método y sigue cada palabra clave return. Por ejemplo, la siguiente firma del método y la instrucción return indican que el método devuelve nombres de variable estDistance mediante referencia al autor de la llamada.

```
public ref double GetEstimatedDistance()
{
   return ref estDistance;
}
```

La palabra clave `return` también detiene la ejecución del método. Si el tipo de valor devuelto es `void`, una instrucción `return` sin un valor también es útil para detener la ejecución del método. Sin la palabra clave `return` , el método dejará de ejecutarse cuando alcance el final del bloque de código. Los métodos con un tipo de valor devuelto no nulo son necesarios para usar la palabra clave `return` para devolver un valor. Por ejemplo, estos dos métodos utilizan la palabra clave `return` para devolver enteros:

```
class SimpleMath
{
    public int AddTwoNumbers(int number1, int number2)
    {
```

```
        return number1 + number2;
    }

    public int SquareANumber(int number)
    {
        return number * number;
    }
}
```

Para utilizar un valor devuelto de un método, el método de llamada puede usar la llamada de método en cualquier lugar; un valor del mismo tipo sería suficiente. También puede asignar el valor devuelto a una variable. Por ejemplo, los dos siguientes ejemplos de código logran el mismo objetivo:

```
int result = obj.AddTwoNumbers(1, 2);
result = obj.SquareANumber(result);
// The result is 9.
Console.WriteLine(result);

result = obj.SquareANumber(obj.AddTwoNumbers(1, 2));
// The result is 9.
Console.WriteLine(result);
```

Usar una variable local, en este caso, `result`, para almacenar un valor es opcional. La legibilidad del código puede ser útil, o puede ser necesaria si debe almacenar el valor original del argumento para todo el ámbito del método.

Para usar un valor devuelto mediante referencia de un método, debe declarar una variable local de tipo ref si pretende modificar su valor. Por ejemplo, si el método Planet.GetEstimatedDistancedevuelve un valor Double mediante referencia, puede definirlo como una variable local de tipo ref con código como el siguiente:

```
ref int distance = plant
```

Devolver una matriz multidimensional de un método, `M`, que modifica el contenido de la matriz no es necesario si la función de llamada ha pasado la matriz a `M`. Puede devolver la matriz resultante de `M` para obtener un estilo correcto o un flujo funcional de valores, pero no es necesario porque C# pasa todos los tipos de referencia mediante valor, y el valor de una referencia de matriz es el puntero de la matriz. En el método `M`, los cambios en el contenido de la matriz los puede observar cualquier código que tenga una referencia a la matriz, como se muestra en el ejemplo siguiente.

```
static void Main(string[] args)
{
```

```
    int[,] matrix = new int[2, 2];
    FillMatrix(matrix);
    // matrix is now full of -1
}

public static void FillMatrix(int[,] matrix)
{
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        for (int j = 0; j < matrix.GetLength(1); j++)
        {
            matrix[i, j] = -1;
        }
    }
}
```