Reflection – Ejemplo Completo

A continuación, vamos a ver un ejemplo completo que nos ayudará a entender mejor la Reflexión y para que la podemos utilizar.

Para realizar este ejemplo se ha creado una aplicación de Consola con Visual Studio con sus valores por defecto, siendo su nombre **ConsoleApplication9** y su clase principal **Program.cs**.

A continuación, se van a agregar las siguientes clases al programa con su correspondiente código fuente:

Agregar -> ICoche.cs

```
// Interface ICoche.cs
namespace ConsoleApplication9
{
    interface ICoche
    {
        bool EnMovimiento();
    }


}
```

Agregar -> Coche.cs

```
// Clase Coche.cs
namespace ConsoleApplication9
{
    internal class Coche : ICoche
    {
        // variable públicas
        public string Color;

        // variable privada
        private int _velocidad;

        //Velocidad – propiedad de solo lectura que devuelve la velocidad
        public int Velocidad
        {
            get { return _velocidad; }
        }

        //Acelerar – añade KM/h a la velocidad
        public void Acelerar(int acelerarPor)
        {
            // Ajustando la velocidad
            _velocidad += acelerarPor;
        }

        // ¿se está moviendo el coche?
        public bool EnMovimiento()
        {
```

```
            // ¿El coche está parado, es decir, con velocidad = 0?
            if (Velocidad == 0)
            {
                return false;
            }
            else
            {
                return true;
            }
        }

        //Constructor
        public Coche()
        {
            // Establece los valores por defecto
            Color = "Blanco";
            _velocidad = 0;
        }

        //sobrecarga de constructor
        public Coche(string color, int velocidad)
        {
            Color = color;
            _velocidad = velocidad;
        }

        //metodos
        public double calcularConsumo(int inicioKM, int finKM, double litros)
        {
            return (finKM - inicioKM) / litros;
        }

    }

}
```

Agregar -> CochesDeportivos.cs

```
// Clase CochesDeportivos.cs
namespace ConsoleApplication9
{
    internal class CochesDeportivos : Coche
    {
        //Constructor
        public CochesDeportivos()
        {
            //Cambia los valores por defecto
            Color = "Azul";
        }
    }


}
```

Utilizamos los miembros de **Type** para obtener información sobre una declaración de tipo, como los constructores, métodos, campos, propiedades y eventos de una clase, así como el módulo y el ensamblaje en el que se implementa la clase.

En este ejemplo vamos a usar la forma de obtener información de tipo es usar el operador **typeof**. Este operador toma el nombre del tipo como un parámetro.

Para ello en la clase **Program.cs**, en el método principal **Main** añadimos el siguiente código fuente:

```
using System;
namespace ConsoleApplication9
{
    public class Program
    {
        static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine(t.FullName);
            Console.ReadLine();
        }

    }

}
```

Resultado

```
ConsoleApplication9.Coche
```

La clase **System.Type** define un número de miembros que se pueden usar para examinar los metadatos de un tipo, un gran número de los cuales devuelven tipos desde el espacio de nombres **System.Reflection**.

Puedes dividir las propiedades implementadas por **Type** en tres categorías:

- Tenemos una serie de propiedades que recuperan las cadenas que contienen varios nombres asociados con la clase, como, por ejemplo:
    - **Name**: El nombre del tipo de datos.
    - **FullName**: El nombre completo del tipo de datos, incluido el nombre del espacio de nombres.
    - **Namespace**: El nombre del espacio de nombres en el que se define el tipo de datos.
- También es posible recuperar referencias a otros tipos de objetos que representan clases relacionadas, como, por ejemplo:
    - **BaseType**: Tipo de base inmediata de este tipo.
    - **UnderlyingSystemType**: El tipo al que este tipo se asigna en el tiempo de ejecución de .NET (tipos predefinidos reconocidos por IL).
- También tenemos propiedades booleanas que indican si este tipo es, por ejemplo, una clase, una enumeración, etc. Estas propiedades nos permiten descubrir una serie de rasgos básicos sobre el tipo al que se refiere: **IsAbstract,**

IsArray, IsClass, IsCOMObject, IsEnum, IsGenericTypeDefinition, IsGenericParameter, IsInterface, IsPrimitive, IsPublic, IsNestedPrivate, IsNestedPublic, IsSealed. IsValueType, IsPointer.

En este ejemplo mostramos información de tipo usando las propiedades de la clase **System.Type**:

A continuación, añadimos el siguiente método en la clase **Program.cs** que nos servirá para obtener las propiedades.

```
public static void ObtenerPropiedades(Type t)
 {
     StringBuilder TextoSalida = new StringBuilder();

     // propiedades para recuperar las cadenas
     TextoSalida.AppendLine("Detalles del Tipo " + t.Name);
     Console.WriteLine("");
     Console.WriteLine("**************************");
     TextoSalida.AppendLine("Nombre del Tipo: " + t.Name);
     TextoSalida.AppendLine("Nombre Completo: " + t.FullName);
     TextoSalida.AppendLine("Espacio de Nombres: " + t.Namespace);

     // propiedades para recuperar las referencias
     Type TipoBase = t.BaseType;

     if (TipoBase != null)
     {
         TextoSalida.AppendLine("Tipo Base: " + TipoBase.Name);
     }

     Type TipoUnderlyingSystem = t.UnderlyingSystemType;

     if (TipoUnderlyingSystem != null)
     {
         TextoSalida.AppendLine("Tipo UnderlyingSystem: " +
             TipoUnderlyingSystem.Name);
     }

     //Propiedades para recuperar booleanos
     TextoSalida.AppendLine("¿Es una Clase Abstracta?: " + t.IsAbstract);
     TextoSalida.AppendLine("¿Es un Array?: " + t.IsArray);
     TextoSalida.AppendLine("¿Es una Clase?: " + t.IsClass);
     TextoSalida.AppendLine("¿Es un Objeto COM? : " + t.IsCOMObject);

     TextoSalida.AppendLine("\nMIEMBROS PÚBLICOS:");
     MemberInfo[] Miembros = t.GetMembers();

     foreach (MemberInfo ProximoMiembro in Miembros)
     {
         TextoSalida.AppendLine(ProximoMiembro.DeclaringType + " " +
         ProximoMiembro.MemberType + "  " + ProximoMiembro.Name);
     }
     Console.WriteLine(TextoSalida);


 }
```

Y agregamos en el método **Main** el método **ObtenerPropiedades()** y las mostramos por pantalla:

```
static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("**************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);
            Console.ReadLine();

        }
```

Resultado

```
**************************

El tipo de este objeto es: ConsoleApplication9.Coche



**************************

Detalles del Tipo Coche

Nombre del Tipo: Coche

Nombre Completo: ConsoleApplication9.Coche

Espacio de Nombres: ConsoleApplication9

Tipo Base: Object

Tipo UnderlyingSystem: Coche

¿Es una Clase Abstracta?: False

¿Es un Array?: False

¿Es una Clase?: True

¿Es un Objeto COM? : False
```

MIEMBROS PÚBLICOS:

ConsoleApplication9.Coche Method get_Velocidad

ConsoleApplication9.Coche Method Acelerar

ConsoleApplication9.Coche Method EnMovimiento

ConsoleApplication9.Coche Method calcularConsumo

System.Object Method ToString

System.Object Method Equals

System.Object Method GetHashCode

System.Object Method GetType

ConsoleApplication9.Coche Constructor .ctor

ConsoleApplication9.Coche Constructor .ctor

ConsoleApplication9.Coche Property Velocidad

ConsoleApplication9.Coche Field Color

La mayoría de los métodos **System.Type** se utilizan para obtener detalles de los miembros del tipo de datos correspondiente: los constructores, las propiedades, los métodos, los eventos, etc. Existe una gran cantidad de métodos, pero todos siguen el mismo patrón. Por ejemplo, dos métodos recuperan detalles de los métodos del tipo de datos son: **ObtenerMetodo()** y **ObtenerMetodos().**

ObtenerMetodo() devuelve una referencia a un objeto **System.Reflection.MethodInfo**, que contiene detalles de un método. Busca el método público con el nombre especificado.

ObtenerMetodo() devuelve una matriz de dichas referencias. La diferencia es que **ObtenerMetodos()** devuelve detalles de todos los métodos, mientras que **ObtenerMetodo()** devuelve detalles de solo un método con una lista de parámetros especificada.

Ambos métodos tienen sobrecargas que toman un parámetro adicional, un valor **BindingFlags** enumerado que indica qué miembros deben devolverse, por ejemplo, si se deben devolver miembros públicos, miembros de instancia, miembros estáticos, etc.

MethodInfo se deriva de la clase abstracta **MetodoBase**, que hereda **MemberInfo**. Por lo tanto, las propiedades y los métodos definidos por estas tres clases están disponibles para su uso.

A continuación, vamos a agregar estos dos métodos en la clase **Program.cs**:

```
       // muestra los nombres de los métodos del tipo.
        public static void ObtenerMetodos(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Metodos *****");
            MethodInfo[] metodos = t.GetMethods();
            foreach (MethodInfo metodo in metodos)
                Console.WriteLine("->{0}", metodo.Name);
            Console.WriteLine("");
        }

        // muestra el nombre del método del tipo.
        public static void ObtenerMetodo(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Método *****");
            //Busca un nombre, es case-sensitive.
            //La búsqueda incluye instancias a métodos publicos estáticos y
públicos
            MethodInfo metodo = t.GetMethod("EnMovimiento");
            Console.WriteLine("->{0}", metodo.Name);
            Console.WriteLine("");

        }
```

Y a mayores, agregamos estos métodos en el método **Main**:

```
       static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("**************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            Console.ReadLine();

        }
```

Resultado

```
**************************

***** Método *****
```

```
->EnMovimiento




**************************

***** Metodos *****

->get_Velocidad

->Acelerar

->EnMovimiento

->calcularConsumo

->ToString

->Equals

->GetHashCode

->GetType
```

Aquí, simplemente está imprimiendo el nombre del método usando la propiedad **MethodInfo.Name**. Como se puede adivinar, **MethodInfo** tiene muchos miembros adicionales que le permiten determinar si el método es **static**, **virtual** o **abstract**. Además, el tipo **MethodInfo** le permite obtener el valor de retorno del método y el conjunto de parámetros.

Vamos a continuar viendo cómo podemos obtener los campos y propiedades de un tipo. El comportamiento de **Type.GetField()** y **Type.GetFields()** es exactamente similar a los dos métodos anteriores, excepto que **Type.GetField()** devuelve una referencia de **System.Reflection.MethodInfo** y **Type.GetFields()** devuelve una referencia de una matriz **System.Reflection.MethodInfo**. Del mismo modo, **Type.GetProperty()** y **Type.GetProperties()**.

A continuación, vamos a agregar los métodos **ObtenerCampos()** y **ObtenerNombrePropiedades()** en la clase **Program.cs**:

```
        // Mostrar los nombres de los campos del tipo.
        public static void ObtenerCampos(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Campos *****");
            FieldInfo[] campos = t.GetFields();
            foreach (FieldInfo campo in campos)
```

```
                Console.WriteLine("->{0}", campo.Name);
            Console.WriteLine("");
        }

        // Mostrar los nombres de las propiedades del tipo.
        public static void ObtenerNombrePropiedades(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Propiedades *****");
            PropertyInfo[] propiedades = t.GetProperties();
            foreach (PropertyInfo propiedad in propiedades)
                Console.WriteLine("->{0}", propiedad.Name);
            Console.WriteLine("");

        }
```

A continuación, agregamos estos métodos en el método principal **Main**:

```
       static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("**************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            // Obtenemos los campos del tipo
            ObtenerCampos(t);

            // Obtenemos los nombres de las propiedades del tipo
            ObtenerNombrePropiedades(t);

            Console.ReadLine();

        }
```

Resultado

```
**************************

***** Campos *****

->Color

**************************
```

```
***** Propiedades *****

->Velocidad
```

También podemos obtener las interfaces del tipo con **GetInterfaces()**, que nos devuelve una matriz de **System.Types**. Esto debería tener sentido dado que las interfaces son, de hecho, tipos.

A continuación, vamos a agregar el método **ObtenerInterfaces** en la clase **Program.cs**:

```
        // Mostrar interfaces implementadas.
        public static void ObtenerInterfaces(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Interfaces *****");
            Type[] interfaces = t.GetInterfaces();
            foreach (Type i in interfaces)
                Console.WriteLine("->{0}", i.Name);

        }
```

Luego, agregamos este método en el método principal **Main**:

```
       static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("**************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            // Obtenemos los campos del tipo
            ObtenerCampos(t);

            // Obtenemos los nombres de las propiedades del tipo
            ObtenerNombrePropiedades(t);

            // Obtenemos las interfaces del tipo
            ObtenerInterfaces(t);

            Console.ReadLine();

        }
```

Resultado

```
**************************

***** Interfaces *****

->ICoche
```

Vamos a continuar con los parámetros del método y los tipos de retorno, primero debemos crear una matriz **MethodInfo[]** utilizando la función **GetMethods()**.

El tipo **MethodInfo** proporciona la propiedad **ReturnType** y el método **GetParameters()** para estas mismas tareas. A continuación, vamos a implementarlos dentro de la clase **Programa.cs**:

```
        // Mostrar métodos de retorno del Tipo y lista de parámetros.
        public static void ObtenerInformacionParametros(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Obtener Información de los Parámetros
*****");
            MethodInfo[] metodos = t.GetMethods();
            foreach (MethodInfo metodo in metodos)
            {
                // Obtener valor retornado.
                string ValoresRetornados = metodo.ReturnType.FullName;
                StringBuilder paramInfo = new StringBuilder();
                paramInfo.Append("(");

                // Obtener Parámetros.
                foreach (ParameterInfo parametro in metodo.GetParameters())
                {
                    paramInfo.Append(string.Format("{0} {1} ",
parametro.ParameterType, parametro.Name));
                }
                paramInfo.Append(")");

                // Mostrar firma básica del método
                Console.WriteLine("->{0} {1} {2}", ValoresRetornados,
metodo.Name, paramInfo);
            }
            Console.WriteLine("");

        }
```

Luego añadimos este método en el método principal **Main**:

```
       static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("**************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);
```

```
            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            // Obtenemos los campos del tipo
            ObtenerCampos(t);

            // Obtenemos los nombres de las propiedades del tipo
            ObtenerNombrePropiedades(t);

            // Obtenemos las interfaces del tipo
            ObtenerInterfaces(t);

            // Obtenemos los métodos de retorno del Tipo y lista de parámetros
            ObtenerInformacionParametros(t);

            Console.ReadLine();

        }
```

Resultado

```
**************************

***** Obtener Información de los Parámetros *****

->System.Int32 get_Velocidad ()

->System.Void Acelerar (System.Int32 acelerarPor )

->System.Boolean EnMovimiento ()

->System.Double calcularConsumo (System.Int32 inicioKM System.Int32 finKM
System.Double litros)

->System.String ToString ()

->System.Boolean Equals (System.Object obj )

->System.Int32 GetHashCode ()

->System.Type GetType ()
```

También podemos obtener los constructores del usando la función **GetConstractors()**, que nos devuelve una matriz de elementos **ConstractorInfo** que podemos usar para obtener más información del constructor de clases.

Vamos a crear un método para obtener los constructores en la clase **Program.cs**:

```
        // Obtenemos información de los constructores
        public static void ObtenerConstructores(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("***************************");
            Console.WriteLine("***** Constructores *****");
            ConstructorInfo[] constructores = t.GetConstructors();
            foreach (ConstructorInfo constructor in constructores)
                Console.WriteLine(constructor.ToString());
            Console.WriteLine("");

        }
```

Luego añadimos este método en el método principal **Main**:

```
       static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("***************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            // Obtenemos los campos del tipo
            ObtenerCampos(t);

            // Obtenemos los nombres de las propiedades del tipo
            ObtenerNombrePropiedades(t);

            // Obtenemos las interfaces del tipo
            ObtenerInterfaces(t);

            // Obtenemos los métodos de retorno del Tipo y lista de parámetros
            ObtenerInformacionParametros(t);

            // Obtenemos información de los constructores
            ObtenerConstructores(t);

            Console.ReadLine();

        }
```

Resultado

```
***** Constructores *****

Void .ctor()

Void .ctor(System.String, Int32)
```

Y finalizamos este ejemplo viendo el Enlace Tardío o **Late Binding** que es una tecnología poderosa en Reflection que nos permite crear una instancia de un tipo dado e invocar a sus miembros en tiempo de ejecución sin tener conocimiento en tiempo de compilación de su existencia.

Esta técnica también se llama invocación dinámica y es útil solo cuando se trabaja con un objeto que no está disponible en tiempo de compilación. En esta técnica, es responsabilidad del desarrollador pasar la firma correcta de los métodos antes de invocar, de lo contrario, producirá un error, mientras que, en el enlace inicial, el compilador verifica la firma del método antes de llamar al método. Es muy importante tomar la decisión correcta sobre cuándo llamar y usar y cuándo no usar esto debido a los problemas de rendimiento que genera. El uso de esta técnica tiene un impacto en el rendimiento de la aplicación.

Para ello en el método **Main**, finalizamos este ejemplo con este código:

```
       static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("****************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            // Obtenemos los campos del tipo
            ObtenerCampos(t);

            // Obtenemos los nombres de las propiedades del tipo
            ObtenerNombrePropiedades(t);

            // Obtenemos las interfaces del tipo
            ObtenerInterfaces(t);

            // Obtenemos los métodos de retorno del Tipo y lista de parámetros
            ObtenerInformacionParametros(t);

            // Obtenemos información de los constructores
            ObtenerConstructores(t);

            Assembly objetoEnsamblado;
            // Carga un Ensamblado
```

```
            objetoEnsamblado = Assembly.GetExecutingAssembly();

            // obtiene la información de tipo de clase en la que se aplicó el
enlace tardío
            Type TipoClase =
objetoEnsamblado.GetType("ConsoleApplication9.Coche");

            // crea la instancia de la clase usando la clase System.Activator
            object objeto = Activator.CreateInstance(TipoClase);

            // obtiene el método del tipo
            MethodInfo metodo = TipoClase.GetMethod("EnMovimiento");

            // Enlace tardío utilizando el método Invoke sin parámetros
            bool EstaCocheMovimiento;
            EstaCocheMovimiento = (bool)metodo.Invoke(objeto, null);
            if (EstaCocheMovimiento)
            {
                Console.WriteLine("El estado del Coche es: En Movimiento");
            }
            else
            {
                Console.WriteLine("El estado del Coche es: Parado");
            }

            // Enlace Tardío con Parámetros
            object[] parametros = new object[3];
            parametros[0] = 32456; //parametro 1 inicioKM
            parametros[1] = 32810; // parametro 2 FinKM
            parametros[2] = 10.6; // parametro 3 litros
            metodo = TipoClase.GetMethod("calcularConsumo");
            double MilesPorKM;
            MilesPorKM = (double)metodo.Invoke(objeto, parametros);
            Console.WriteLine("Miles por Kilómetro son: " + MilesPorKM);

            Console.ReadLine();

        }
```

Resultado

```
El estado del Coche es: Parado

Miles por Kilómetro son: 33,3962264150943
```

Con esto finalizamos este ejemplo donde hemos implementado varios métodos por medio de los cuales hemos obtenido usando la reflexión sobre el tipo de una clase que hemos creado para este ejemplo.

CÓDIGO FUENTE COMPLETO

```
using System.Reflection;
using System;
using System.Text;

namespace ConsoleApplication9
{
    public class Program
    {
        public static void ObtenerPropiedades(Type t)
        {
            StringBuilder TextoSalida = new StringBuilder();

            // propiedades para recuperar las cadenas
            TextoSalida.AppendLine("Detalles del Tipo " + t.Name);
            Console.WriteLine("");
            Console.WriteLine("**************************");
            TextoSalida.AppendLine("Nombre del Tipo: " + t.Name);
            TextoSalida.AppendLine("Nombre Completo: " + t.FullName);
            TextoSalida.AppendLine("Espacio de Nombres: " + t.Namespace);

            // propiedades para recuperar las referencias
            Type TipoBase = t.BaseType;

            if (TipoBase != null)
            {
                TextoSalida.AppendLine("Tipo Base: " + TipoBase.Name);
            }

            Type TipoUnderlyingSystem = t.UnderlyingSystemType;

            if (TipoUnderlyingSystem != null)
            {
                TextoSalida.AppendLine("Tipo UnderlyingSystem: " +
                    TipoUnderlyingSystem.Name);
            }

            //Propiedades para recuperar booleanos
            TextoSalida.AppendLine("¿Es una Clase Abstracta?: " + t.IsAbstract);
            TextoSalida.AppendLine("¿Es un Array?: " + t.IsArray);
            TextoSalida.AppendLine("¿Es una Clase?: " + t.IsClass);
            TextoSalida.AppendLine("¿Es un Objeto COM? : " + t.IsCOMObject);

            TextoSalida.AppendLine("\nMIEMBROS PÚBLICOS:");
            MemberInfo[] Miembros = t.GetMembers();

            foreach (MemberInfo ProximoMiembro in Miembros)
            {
                TextoSalida.AppendLine(ProximoMiembro.DeclaringType + " " +
                ProximoMiembro.MemberType + "  " + ProximoMiembro.Name);
            }
            Console.WriteLine(TextoSalida);
        }

        // muestra los nombres de los métodos del tipo.
        public static void ObtenerMetodos(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Metodos *****");
```

```
            MethodInfo[] metodos = t.GetMethods();
            foreach (MethodInfo metodo in metodos)
                Console.WriteLine("->{0}", metodo.Name);
            Console.WriteLine("");
        }

        // muestra el nombre del método del tipo.
        public static void ObtenerMetodo(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Método *****");
            //Busca un nombre, es case-sensitive.
            //La búsqueda incluye instancias a métodos publicos estáticos y
públicos
            MethodInfo metodo = t.GetMethod("EnMovimiento");
            Console.WriteLine("->{0}", metodo.Name);
            Console.WriteLine("");
        }

        // Mostrar los nombres de los campos del tipo.
        public static void ObtenerCampos(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Campos *****");
            FieldInfo[] campos = t.GetFields();
            foreach (FieldInfo campo in campos)
                Console.WriteLine("->{0}", campo.Name);
            Console.WriteLine("");
        }

        // Mostrar los nombres de las propiedades del tipo.
        public static void ObtenerNombrePropiedades(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Propiedades *****");
            PropertyInfo[] propiedades = t.GetProperties();
            foreach (PropertyInfo propiedad in propiedades)
                Console.WriteLine("->{0}", propiedad.Name);
            Console.WriteLine("");
        }

        // Mostrar interfaces implementadas.
        public static void ObtenerInterfaces(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Interfaces *****");
            Type[] interfaces = t.GetInterfaces();
            foreach (Type i in interfaces)
                Console.WriteLine("->{0}", i.Name);
        }

        // Mostrar métodos de retorno del Tipo y lista de parámetros.
        public static void ObtenerInformacionParametros(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("**************************");
            Console.WriteLine("***** Obtener Información de los Parámetros
*****");
```

```
            MethodInfo[] metodos = t.GetMethods();
            foreach (MethodInfo metodo in metodos)
            {
                // Obtener valor retornado.
                string ValoresRetornados = metodo.ReturnType.FullName;
                StringBuilder paramInfo = new StringBuilder();
                paramInfo.Append("(");

                // Obtener Parámetros.
                foreach (ParameterInfo parametro in metodo.GetParameters())
                {
                    paramInfo.Append(string.Format("{0} {1} ",
parametro.ParameterType, parametro.Name));
                }
                paramInfo.Append(")");

                // Mostrar firma básica del método
                Console.WriteLine("->{0} {1} {2}", ValoresRetornados,
metodo.Name, paramInfo);
            }
            Console.WriteLine("");
        }

        // Obtenemos información de los constructores
        public static void ObtenerConstructores(Type t)
        {
            Console.WriteLine("");
            Console.WriteLine("****************************");
            Console.WriteLine("***** Constructores *****");
            ConstructorInfo[] constructores = t.GetConstructors();
            foreach (ConstructorInfo constructor in constructores)
                Console.WriteLine(constructor.ToString());
            Console.WriteLine("");
        }


        static void Main()
        {
            Coche c = new Coche();
            // Obtener el tipo usando typeof
            Type t = typeof(Coche);
            Console.WriteLine("****************************");
            Console.WriteLine("El tipo de este objeto es: {0}", t.FullName);

            // modificamos esta línea para recuperar detalles de cualquier otro
tipo de datos
            ObtenerPropiedades(t);

            // Obtenemos los métodos del tipo
            ObtenerMetodo(t);
            ObtenerMetodos(t);

            // Obtenemos los campos del tipo
            ObtenerCampos(t);

            // Obtenemos los nombres de las propiedades del tipo
            ObtenerNombrePropiedades(t);

            // Obtenemos las interfaces del tipo
            ObtenerInterfaces(t);

            // Obtenemos los métodos de retorno del Tipo y lista de parámetros
```

```
            ObtenerInformacionParametros(t);

            // Obtenemos información de los constructores
            ObtenerConstructores(t);

            Assembly objetoEnsamblado;
            // Carga un Ensamblado
            objetoEnsamblado = Assembly.GetExecutingAssembly();

            // obtiene la información de tipo de clase en la que se aplicó el
enlace tardío
            Type TipoClase =
objetoEnsamblado.GetType("ConsoleApplication9.Coche");

            // crea la instancia de la clase usando la clase System.Activator
            object objeto = Activator.CreateInstance(TipoClase);

            // obtiene el método del tipo
            MethodInfo metodo = TipoClase.GetMethod("EnMovimiento");

            // Enlace tardío utilizando el método Invoke sin parámetros
            bool EstaCocheMovimiento;
            EstaCocheMovimiento = (bool)metodo.Invoke(objeto, null);
            if (EstaCocheMovimiento)
            {
                Console.WriteLine("El estado del Coche es: En Movimiento");
            }
            else
            {
                Console.WriteLine("El estado del Coche es: Parado");
            }

            // Enlace Tardío con Parámetros
            object[] parametros = new object[3];
            parametros[0] = 32456; //parametro 1 inicioKM
            parametros[1] = 32810; // parametro 2 FinKM
            parametros[2] = 10.6; // parametro 3 litros
            metodo = TipoClase.GetMethod("calcularConsumo");
            double MilesPorKM;
            MilesPorKM = (double)metodo.Invoke(objeto, parametros);
            Console.WriteLine("Miles por Kilómetro son: " + MilesPorKM);

            Console.ReadLine();
        }

    }

}
```

FIN