Рефлексия

{C# - Reflection}

В мире .NET **рефлексией (reflection)** называется процесс обнаружения типов во время выполнения.

С применением рефлексии метаданные можно получать программно в виде удобной объектной модели.

Классы, обеспечивающие рефлексию, входят в состав .NET Reflection API, относящегося к пространству имен **System.Reflection**.

Основные классы System.Reflection

Имя	Описание
Assembly	В классе содержатся статические методы, которые позволяют загружать сборку, исследовать ее и производить с ней различные манипуляции
AssemblyName	Этот класс позволяет выяснить различные детали, связанные с идентификацией сборки (номер версии, информация о культуре и т.д.)
FieldInfo	содержит информацию о заданном поле
PropertyInfo	содержит информацию по заданному свойству
MethodInfo	содержит информацию по заданному методу
ParameterInfo	содержит информацию о параметре метода
EventInfo	содержит информацию о заданном событии
MemberInfo	определяет общее поведение для типов EventInfo, FieldInfo, MethodInfo и PropertyInfo

Класс System.Type

- составляет ядро подсистемы рефлексии, инкапсулирует тип данных.
- содержит многие свойства и методы, которыми можно пользоваться для получения информации о типе данных во время выполнения.
- производный от абстрактного класса System.Reflection.MemberInfo.

Имя	Описание
IsAbstract IsArray IsClass IsEnum IsGenericParameter IsInterface IsSealed IsValueType и т.д.	Эти свойства позволяют выяснять ряд основных деталей об интересующем типе (например, является ли он абстрактной сущностью, массивом, вложенным классом и т.д.)
GetConstructors() GetEvents() GetFields() GetInterfaces() GetMembers() GetMethods() GetNestedTypes() GetProperties()	Эти методы позволяют получать массив представляющих интерес элементов (интерфейсов, методов, свойств и т.д.). Каждый из этих методов возвращает соответствующий массив (например, GetFields() возвращает массив FieldInfo, GetMethods () — массив MethodInfo и тд.). Имеются методы для получения одного элемента.
FindMembers()	возвращает массив объектов типа MemberInfo на основе указанных критериев поиска
GetType()	возвращает экземпляр Type, обладающий указанным строковым именем
InvokeMember()	Реализация "позднее связывание" для заданного элемента

Исходный класс Car

```
class Car
    {
        public int Num;  // поле
        public string  Name {get; set;}  // свойство

        public Car(int Num, string Name)  // конструктор
        {
            this.Num = Num; this.Name = Name;
        }
        public override string ToString() // метод
        {
            return String.Format("{0}. {1}",Num, Name);
        }
        public event EventHandler Stop; // событие

        public static Car  operator +(Car car, int N)
        {
            return new Car(N++, "BMW");
        }
    }
```

Класс System.Type

```
// Способы получения экземпляра класса Type.
Type t1 = typeof(Car);
Console.WriteLine("typeof: " + t1);
// Полное имя типа в строковом представлении.
Type t2 = Type.GetType("ExampleType.Car");
Console.WriteLine("Type.GetType: " + t2);
Car car = new Car(1, "BMW");
Type t3 = car.GetType();
Console.WriteLine("car.GetType(): " + t3);

Console.WriteLine("Полное Имя:              {0}", t1.FullName);
Console.WriteLine("Базовый класс:           {0}", t1.BaseType);
Console.WriteLine("Абстрактный:             {0}", t1.IsAbstract);
Console.WriteLine("Запрещено наследование: {0}", t1.IsSealed);
Console.WriteLine("Это class:               {0}", t1.IsClass);
Console.WriteLine("Информация о членах класса Car");
foreach (MemberInfo mi in t1.GetMembers())
{   // DeclaringType -  член данного класса или базового
    // MemberType - тип члена (метод, конструктор и т.д.)
    // Name - имя члена
 Console.WriteLine(mi.DeclaringType + mi.MemberType + mi.Name);
}
```

Класс System.Type

```
typeof: ExampleType.Car
Type.GetType: ExampleType.Car
car.GetType(): ExampleType.Car
Информация о  классе Car
Полное Имя:                 ExampleType.Car
Базовый класс:              System.Object
Абстрактный:                False
Это COM объект:             False
Запрещено наследование: False
Это class:                  True
Информация о членах класса Car
ExampleType.Car Method get_Name
ExampleType.Car Method set_Name
ExampleType.Car Method ToString
ExampleType.Car Method add_Stop
ExampleType.Car Method remove_Stop
ExampleType.Car Method op_Addition
System.Object Method Equals
System.Object Method GetHashCode
System.Object Method GetType
ExampleType.Car Constructor .ctor
ExampleType.Car Property Name
ExampleType.Car Event Stop
ExampleType.Car Field Num
```

Рефлексия полей и свойств

```
Type t1 = typeof(Car);
// получение информации о всех полях
Console.WriteLine("{0,-30}{1,-20}{2,-20}",
                  "Принадлежность", "Имя","Тип", "Статический");
foreach (FieldInfo fi in t1.GetFields())
{
Console.WriteLine("{0,-30}{1,-20}{2,-20}{3,-20}",
          fi.DeclaringType, fi.Name, fi.FieldType, fi.IsStatic);
}
Console.WriteLine("-----------------------------------------");
// получение информации о всех свойствах
Console.WriteLine("{0,-20}{1,-20}{2,-20}{3,-20}", "Принадлежность",
"Имя", "Чтение (Get)", "Запись (Set)");
foreach (PropertyInfo pi in t1.GetProperties())
{    Console.WriteLine("{0,-20}{1,-20}{2,-20}{3,-20}",
               pi.DeclaringType, pi.Name, pi.CanRead, pi.CanWrite);

 foreach (MethodInfo mi in pi.GetAccessors())
  {    // ReturnType - тип возвращаемого значения
        Console.WriteLine("{0,-20}{1,-20}",mi.ReturnType, mi.Name);
  }
}
```

Рефлексия полей и свойств

```
Принадлежность                  Имя                     Тип
Example2.Car                    Name                    System.String       False

Example2.Car                    Fuel                    System.Double       True

-------------------------------------------------------------------------------
Принадлежность      Имя                     Чтение (Get)        Запись (Set)

Example2.Car        Num                     True                True

System.Int32        get_Num
System.Void         set_Num
Example2.Car        Price                   True                True

System.Double       get_Price
System.Void         set_Price
```

```
class Car
{
public string Name; // открытое поле
public static double Fuel; // открытое статическое поле
private int num; // закрытое поле
public int Num  // свойство
{  get { return num; }   set { num = value; }
}
// автоматическое свойство
public double Price { get; set; }
}
```

Рефлексия конструкторов. Создание объекта

```
Type t1 = typeof(Car);
// получение информации о всех конструкторах
Console.WriteLine("{0,-30}{1,-20}", "Принадлежность", "Имя");
foreach (ConstructorInfo mi in t1.GetConstructors())
{
    Console.WriteLine("{0,-30}{1,-20}", mi.DeclaringType, mi.Name);
    foreach (ParameterInfo pi in mi.GetParameters())
    {
        Console.WriteLine("{0,10}. {1,-20}{2,-20}",
                          pi.Position, pi.Name, pi.ParameterType);
    }
}
Console.WriteLine("----------------------------------------");

Type CarType = Type.GetType("ExampleCreateObjects.Car");
//получение инф-ии о конструкторе - с параметрами:  int и  string
ConstructorInfo ci = CarType.GetConstructor
                     (new Type[] { typeof(int), typeof(string)});

//вызов конструктора с передачей значений - создание  объекта!
object Obj = ci.Invoke(new object[] { 1, "BMW" });
Console.WriteLine(Obj.ToString());
```

Рефлексия конструкторов. Создание объекта

```
Принадлежность                  Имя
ExampleCreateObjects.Car        .ctor
          0. Num                  System.Int32
          1. Name                 System.String
-------------------------------------------------
1. BMW
-------------------------------------------------
```

```
class Car
{  int Num;
    string Name;
    double Fuel;

    public Car(int Num, string Name)
    {   Fuel = 0;
        this.Num = Num; this.Name = Name;
    }
    public override string ToString()
    {  return String.Format("{0}. {1}", Num, Name);
    }
}
```

Рефлексия методов

BindingFlags

Имя	Описание
DeclaredOnly	Извлекаются только те методы, которые определены в заданном классе. Унаследованные методы в извлекаемые сведения не включаются
Instance	Извлекаются методы экземпляра
Nonpublic	Извлекаются методы, не являющиеся открытыми
Public	Извлекаются открытые методы
Static	Извлекаются статические методы

Для вызова конструктора или метода используется метод Invoke():

object Invoke(object[] parameters)

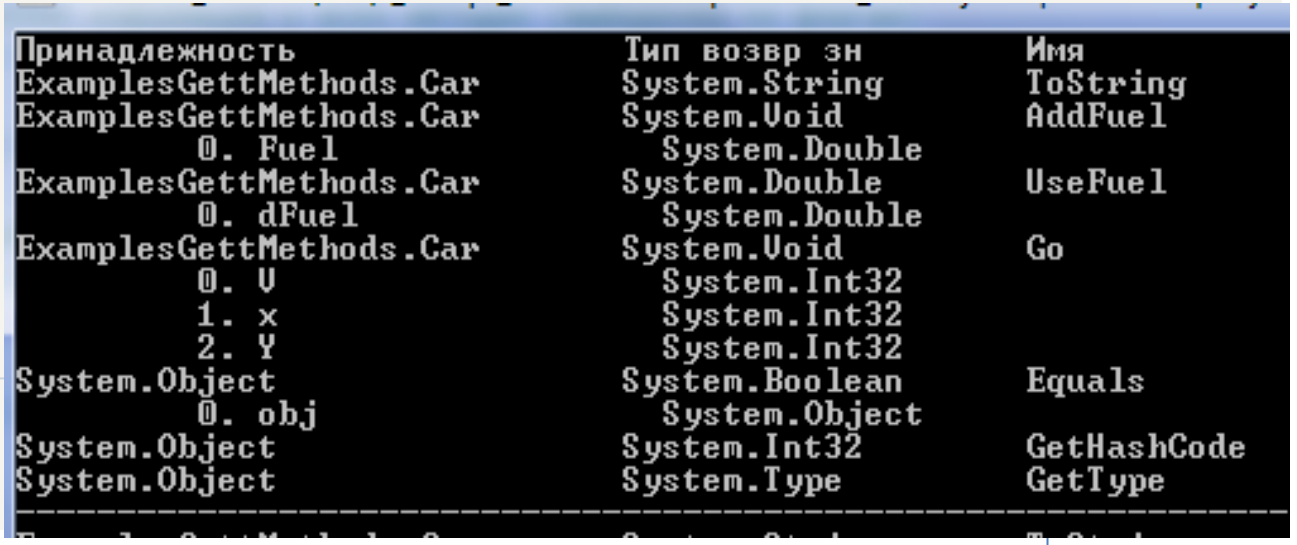
parameters - аргументы, которые требуется передать методу (массив).

- количество элементов массива parameters должно совпадать с количеством передаваемых аргументов, а типы аргументов - с типами параметров.
- метод Invoke() возвращает ссылку на сконструированный объект.

Рефлексия методов

```
Type t1 = typeof(Car);
// получение информации о всех методах
Console.WriteLine("{0,-30}{1,-20}{2,-20}",
                                  "Принадлежность","Тип возвр зн", "Имя");
foreach (MethodInfo mi in t1.GetMethods())
{
// ReturnType - тип возвращаемого значения
Console.WriteLine("{0,-30}{1,-20}{2,-20}",
                               mi.DeclaringType, mi.ReturnType, mi.Name);
      foreach (ParameterInfo pi in mi.GetParameters())
        {
           Console.WriteLine("{0,10}. {1,-20}{2,-20}",
                                pi.Position, pi.Name, pi.ParameterType);
         }
}
```

Рефлексия методов



```
Принадлежность                  Тип возвр зн        Имя
ExamplesGettMethods.Car         System.String       ToString
ExamplesGettMethods.Car         System.Void         AddFuel
        0. Fuel                   System.Double
ExamplesGettMethods.Car         System.Double       UseFuel
        0. dFuel                  System.Double
ExamplesGettMethods.Car         System.Void         Go
        0. V                      System.Int32
        1. x                      System.Int32
        2. Y                      System.Int32
System.Object                   System.Boolean      Equals
        0. obj                    System.Object
System.Object                   System.Int32        GetHashCode
System.Object                   System.Type         GetType
```

```
abstract class Car
{    int Num;
     string Name;
     double Fuel;
     public Car(int Num, string Name)
     {    Fuel = 0;
          this.Num = Num; this.Name = Name;     }

     public override string ToString()
     {  return String.Format("{0}. {1}", Num, Name); }
     public void AddFuel(double Fuel)
     {  this.Fuel = Fuel;  }

     public double UseFuel(double dFuel)
     {    Fuel = Fuel - dFuel;      return Fuel;
     }
     public abstract void Go(int V, int x, int Y);
}
```

Рефлексия методов

```
// получение информации о всех методах (выборочно по флагам)
MethodInfo[] InfoMethods = t1.GetMethods(BindingFlags.DeclaredOnly |
                           BindingFlags.Instance | BindingFlags.Public);
foreach (MethodInfo mi in InfoMethods)
{   Console.WriteLine("{0,-30}{1,-20}{2,-20}",
                          mi.DeclaringType, mi.ReturnType, mi.Name);
    foreach (ParameterInfo pi in mi.GetParameters())
    {    Console.WriteLine("{0,10}. {1,-20}{2,-20}",
                            pi.Position, pi.Name, pi.ParameterType);
    }
}
// получение информации о конкретном методе по имени
MethodInfo info = t1.GetMethod("ToString");
Console.WriteLine("{0,-30}{1,-20}{2,-20}",
           info.DeclaringType, info.ReturnType, info.Name);

if (info.IsStatic) Console.WriteLine("Метод  статический);
if (info.IsPublic) Console.WriteLine("Метод открытый");
if (info.IsPrivate) Console.WriteLine("Метод закрытый");
```

Рефлексия методов

```
ExamplesGettMethods.Car       System.String       ToString
ExamplesGettMethods.Car       System.Void         AddFuel
        0. Fuel                 System.Double
ExamplesGettMethods.Car       System.Double       UseFuel
        0. dFuel                System.Double
ExamplesGettMethods.Car       System.Void         Go
        0. V                    System.Int32
        1. x                    System.Int32
        2. Y                    System.Int32
-------------------------------------------------------------
ExamplesGettMethods.Car       System.String       ToString
Метод ToString - открытый
Метод ToString - виртуальный
```

```
abstract class Car
{    int Num;
     string Name;
     double Fuel;
     public Car(int Num, string Name)
     {    Fuel = 0;
          this.Num = Num; this.Name = Name;     }

     public override string ToString()
     {  return String.Format("{0}. {1}", Num, Name); }

     public void AddFuel(double Fuel)
     {  this.Fuel = Fuel;  }

     public double UseFuel(double dFuel)
     {    Fuel = Fuel - dFuel;      return Fuel;
     }
     public abstract void Go(int V, int x, int Y);
}
```

Рефлексия методов

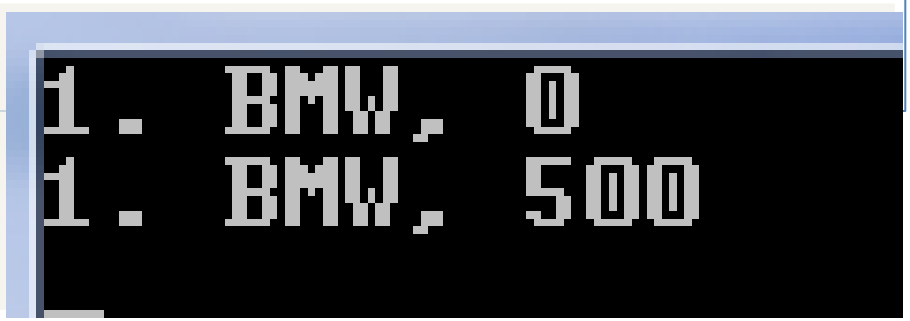
```
Type CarType = typeof(Car);
// Получение конструктора
ConstructorInfo ci = CarType.GetConstructor(new Type[]
                                        { typeof(int), typeof(string) });

//вызов конструктора с передачей значений - создание  объекта!
object Obj = ci.Invoke(new object[] { 1, "BMW" });

Console.WriteLine(Obj.ToString());

// получение информации о методе по имени
MethodInfo info = CarType.GetMethod("AddFuel");
// Вызов метода AddFuel объекта Obj с передачей значения "500"
info.Invoke(Obj, new object[] {500});

Console.WriteLine(Obj.ToString());
```



Рефлексия интерфейсов

```
Type t1=typeof(Car);

// получение списка интерфейсов, реализованных в классе Car
foreach (Type type in t1.GetInterfaces())
{
    Console.WriteLine("{0,-30}{1,-20}",
                                    type.DeclaringType, type.Name);
    // вывод списка методов интерфейса
    foreach (MethodInfo pi in type.GetMethods())
    {
        Console.WriteLine("{0,-10}. {1,-20}{2,-20}",
                            pi.Name, pi.ReturnType, pi.DeclaringType);
    }
}
```

Рефлексия интерфейсов

```
                                    IMove
Stop        . System.Void             ExampleInterface.IMove
Move        . System.Void             ExampleInterface.IMove
```

```
interface IMove
{
  void Stop();
  void Move(int dX);
 }
```

```
class Car:IMove
{
  int Num;
  string Name;
  double Fuel;
  public Car(int Num, string Name)
  {    Fuel = 0;
       this.Num = Num; this.Name = Name;
  }
  public override string ToString()
  {  return String.Format("{0}. {1}", Num, Name);  }
  public void Stop()
  {      }
  public void Move(int dX)
  {       }
}
```

Динамически загружаемые сборки

Процесс загрузки внешних сборок по требованию называется **динамической загрузкой**.

Класс Assembly позволяет:

- динамически загружать приватные и разделяемые сборки,
- просматривать собственные свойства сборок.

Assembly.Load()	**Загрузка сборки. Передается только дружественное имя сборки. Assembly.Load(“MyClassLib”)**
Assembly.LoadFrom()	**Загрузка сборки. Передается полный путь к сборке. Assembly.LoadFrom(“D:\MyClassLib.dll”)**

Динамически загружаемые сборки

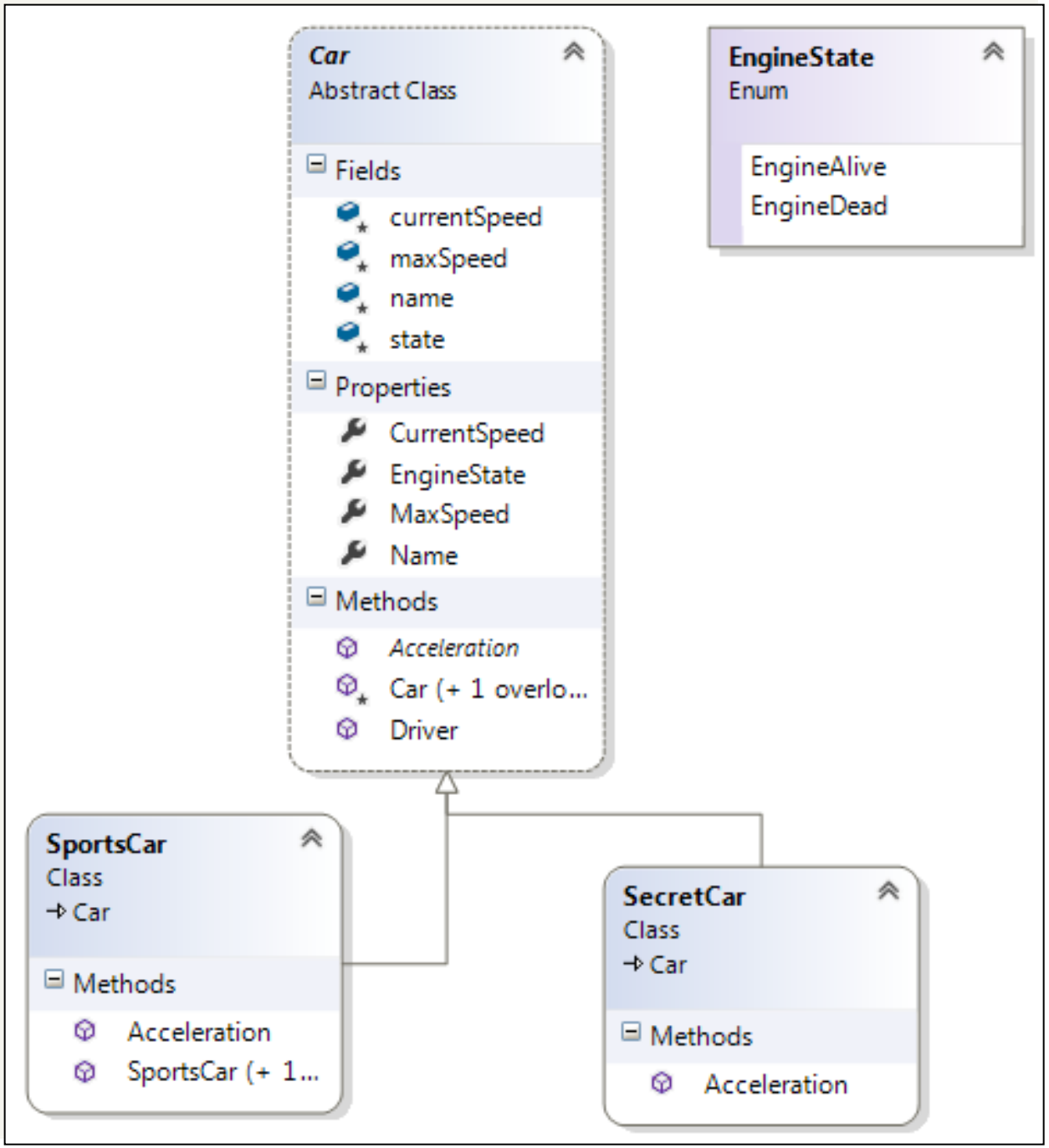
```
Assembly assembly = null;
try
{   assembly = Assembly.Load("TestClassLib");
    Console.WriteLine("Сборка TestClassLib - успешно загружена.");
}
catch (FileNotFoundException ex)
{   Console.WriteLine(ex.Message);  }

// Выводим информацию о всех типах в сборке.
Console.WriteLine(new string('_', 80));
Console.WriteLine("\nТипы в: {0} \n", assembly.FullName);

Type[] types = assembly.GetTypes();
foreach (Type t in types)
    Console.WriteLine("Тип: {0}", t);

// Выводим информацию о членах выбранного типа.
Type type = assembly.GetType("TestClassLib.SportsCar");

MemberInfo[] members = type.GetMembers();
foreach (MemberInfo element in members)
     Console.WriteLine("{0,-15}:  {1}", element.MemberType, element);
```


Car
Abstract Class
Fields
currentSpeed
maxSpeed
name
state
Properties
CurrentSpeed
EngineState
MaxSpeed
Name
Methods
Acceleration
Car (+ 1 overlo...
Driver
EngineState
Enum
EngineAlive
EngineDead
SportsCar
Class
Car
Methods
Acceleration
SportsCar (+ 1...
SecretCar
Class
Car
Methods
Acceleration

Динамически загружаемые сборки

```
Сборка TestClassLib - успешно загружена.
-------------------------------------------------------------------------------


Типы в: TestClassLib, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

Тип: TestClassLib.Car
Тип: TestClassLib.EngineState
Тип: TestClassLib.SportsCar
Тип: TestClassLib.SecretCar
-------------------------------------------------------------------------------


Члены класса: TestClassLib.SportsCar

Method          :  Void Acceleration()
Method          :  Void Driver(System.String, Int32)
Method          :  System.String get_Name()
Method          :  Void set_Name(System.String)
Method          :  Int16 get_CurrentSpeed()
Method          :  Void set_CurrentSpeed(Int16)
Method          :  Int16 get_MaxSpeed()
Method          :  TestClassLib.EngineState get_EngineState()
Method          :  System.String ToString()
Method          :  Boolean Equals(System.Object)
Method          :  Int32 GetHashCode()
Method          :  System.Type GetType()
Constructor     :  Void .ctor()
Constructor     :  Void .ctor(System.String, Int16, Int16)
Property        :  System.String Name
Property        :  Int16 CurrentSpeed
Property        :  Int16 MaxSpeed
Property        :  TestClassLib.EngineState EngineState
```

Позднее связывание

Поздним связыванием (late binding) называется технология, которая позволяет создавать экземпляр определенного типа и вызывать его члены во время выполнения без кодирования факта его существования жестким образом на этапе компиляции.

Широко используется в расширяемых приложениях.

Класс System.Activator

- Класс **System.Activator** играет ключевую роль в процессе позднего связывания в .NET.
- Основной метод **Activator.CreateInstance()**, который позволят создавать экземпляр подлежащего позднему связыванию типа.

Раннее связывание - все указанные выше операции выполняются во время компиляции.

Позднее связывание

```
SPORTCAR:  Быстрая скорость!
Имя водителя: Иванов И.И.!. Возраст: 45
```

```
Assembly assembly = null;
try
{
   assembly = Assembly.Load("TestClassLib");
}
catch (FileNotFoundException e)
{   Console.WriteLine(e.Message);                }

// создание экземпляра типа во время выполнения.
Type type = assembly.GetType("TestClassLib.SportsCar");
object instance = Activator.CreateInstance(type);
// Получаем экземпляр класса MethodInfo для метода Acceleration().
MethodInfo method = type.GetMethod("Acceleration");

method.Invoke(instance, null);

method = type.GetMethod("Driver");

// Массив параметров для метода Driver("Водитель", 45).
object[] parameters = { "Иванов И.И.!", 45 };
method.Invoke(instance, parameters);
```

Позднее связывание

```
Assembly assembly = null;

try
{
    // загрузка сборки
    assembly = Assembly.Load("TestClassLib");
    // получение указанного типа из сборки!
    Type type = assembly.GetType("TestClassLib.SportsCar");
    // создание объекта класса
    dynamic carInstance = Activator.CreateInstance(type);
    // вызов метода!
    carInstance.Acceleration();
    carInstance.Driver("Иванов И.И.!", 50);

}
catch (FileNotFoundException e)
{
    Console.WriteLine(e.Message);
}
```

```
SPORTCAR:  Быстрая скорость!
Имя водителя: Иванов И.И.!. Возраст: 50
```