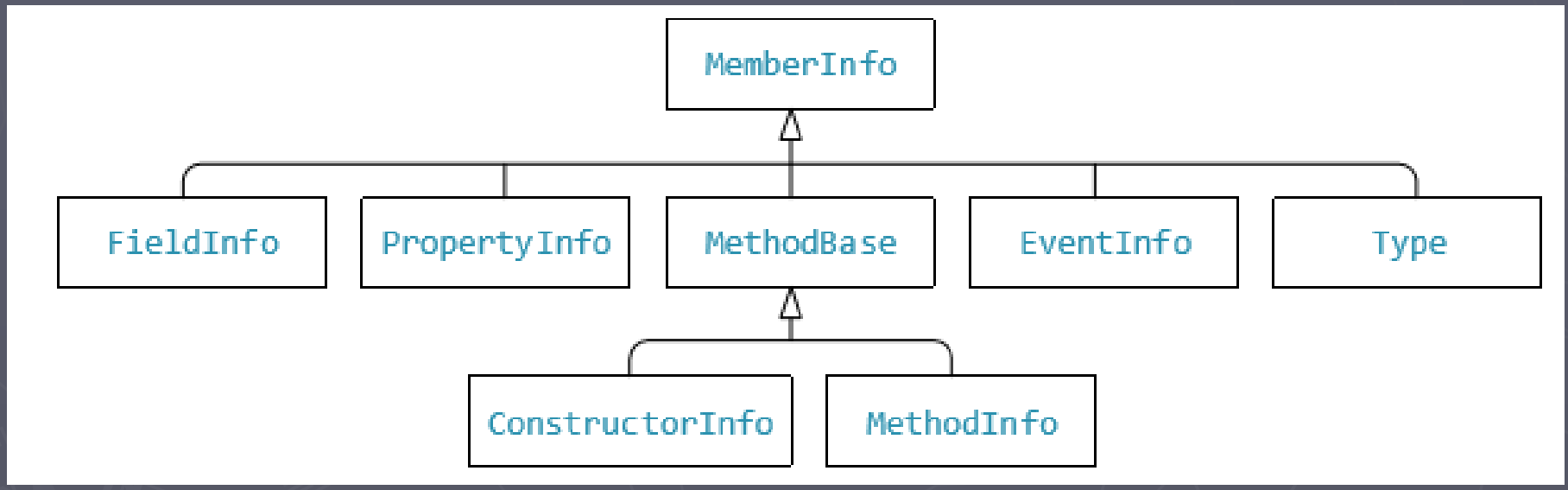
Рефлексия

- Процесс выявления типов во время выполнения приложения

System.Reflection

При создании сборки в неё помещаются метаданные, которые являются описанием всех типов в сборке и их элементов

Класс	Назначение
Assembly	сборка манипулирование этой сборкой
AssemblyName	информация о сборке
EventInfo	информация о событии
FieldInfo	информация о поле
MethodInfo	информация о методе
PropertyInfo **ConstructorInfo**	информация о свойстве, конструкторе
Module	доступ к определенному модулю внутри сборки


MemberInfo
FieldInfo
PropertyInfo
MethodBase
EventInfo
Type
ConstructorInfo
MethodInfo

System.Type - класс, позволяет получить информацию о членах типа – методы:

- **FindMembers()** - воз. массив объектов MemberInfo данного типа
- **GetConstructors()** - конструкторы данного типа в виде набора объектов ConstructorInfo
- **GetEvents()** - события данного типа в виде массива объектов EventInfo
- **GetFields()** - поля данного типа в виде массива объектов FieldInfo
- **GetInterfaces()** - реализуемые данным типом интерфейсы в виде массива объектов Type
- **GetMembers()** -члены типа в виде массива объектов MemberInfo
- **GetMethods()** - методы типа в виде массива объектов MethodInfo
- **GetProperties()** - свойства в виде массива объектов PropertyInfo
- Свойство **IsAbstract** возвращает true, если тип является абстрактным
- Свойство **IsArray** возвращает true, если тип является массивом
- Свойство **IsClass** возвращает true, если тип представляет класс
- Свойство **IsEnum** возвращает true, если тип является перечислением
- Свойство **IsInterface** возвращает true, если тип представляет интерфейс

Получить информацию о типе

```
User admin = new  User();
 // User – это некий класс

 Type t = admin.GetType();
```

```
 Type t1 = Type.GetType("System.Int32");
```

```
System.Int32
```

```
 Type t2 = typeof(Point);
 Type t3 = typeof(List<int>);
 Type t4 = typeof(List<>);
```

Применение рефлексии

```
using System.Reflection;
```

```
Type t = typeof(Int32);
 Console.WriteLine("Full name = " + t.FullName);
 Console.WriteLine("Base type is = " + t.BaseType);
 Console.WriteLine("Is sealed = " + t.IsSealed);
 Console.WriteLine("Is class = " + t.IsClass);
 foreach (Type iType in t.GetInterfaces()) {
     Console.WriteLine(iType.Name);
 }
 foreach (FieldInfo fi in t.GetFields()) {
     Console.WriteLine("Field = " + fi.Name);
 }
```

```
Full name = System.Int32
Base type is = System.ValueType
Is sealed = True
Is class = False
IComparable
IFormattable
IConvertible
IComparable`1
IEquatable`1
Field = MaxValue
```

```
foreach (PropertyInfo pi in t.GetProperties()) {
        Console.WriteLine("Property = " + pi.Name);
    }
    foreach (MethodInfo mi in t.GetMethods()) {
        Console.WriteLine("Method Name = " + mi.Name);
        Console.WriteLine("Method Return Type = " + mi.ReturnType);
        foreach (ParameterInfo pr in mi.GetParameters()) {
            Console.WriteLine("Parameter Name = " + pr.Name);
            Console.WriteLine("Type = " + pr.ParameterType);
        }
    }
```

покажет данные только об открытых элементах типа

```
Method Name = TryParse
Method Return Type = System.Boolean
Parameter Name = s
Type = System.String
Parameter Name = result
Type = System.Int32&
Method Name = TryParse
Method Return Type = System.Boolean
Parameter Name = s
Type = System.String
Parameter Name = style
Type = System.Globalization.NumberStyles
Parameter Name = provider
Type = System.IFormatProvider
Parameter Name = result
Type = System.Int32&
Method Name = GetTypeCode
Method Return Type = System.TypeCode
Method Name = GetType
Method Return Type = System.Type
```

```
Method Name = CompareTo
Method Return Type = System.Int32
Parameter Name = value
Type = System.Object
Method Name = CompareTo
Method Return Type = System.Int32
Parameter Name = value
Type = System.Int32
Method Name = Equals
Method Return Type = System.Boolean
Parameter Name = obj
Type = System.Object
Method Name = Equals
Method Return Type = System.Boolean
Parameter Name = obj
Type = System.Int32
Method Name = GetHashCode
Method Return Type = System.Int32
Method Name = ToString
Method Return Type = System.String
Method Name = ToString
Method Return Type = System.String
Parameter Name = format
Type = System.String
Method Name = ToString
Method Return Type = System.String
```

System.Reflection.BindingFlags

- Флаги BindingFlags, связанные с получением

Флаг	Описание
Default	Отсутствие специальных флагов
IgnoreCase	Игнорировать регистр имён получаемых элементов
DeclaredOnly	Получить элементы, объявленные непосредственно в типе (игнорировать унаследованные элементы)
Instance	Получить экземплярные элементы
Static	Получить статические элементы
Public	Получить открытые элементы
NonPublic	Получить закрытые элементы
FlattenHierarchy	Получить public и protected элементы у типа и у всех его предков

```
        Type tt = typeof(Int32);
        var bf =  BindingFlags.Public |
                  BindingFlags.NonPublic |
                  BindingFlags.Static |
                  BindingFlags.Instance;

FieldInfo[] fi = tt.GetFields(bf);
```

Основные элементы класса Assembly

Имя элемента	Описание
`CreateInstance()`	Находит по имени тип в сборке и создаёт его экземпляр
`FullName`	Строковое свойство с полным именем сборки
`GetAssembly()`	Ищет в памяти и возвращает объект `Assembly`, который содержит указанный тип (статический метод)
`GetCustomAttributes()`	Получает атрибуты сборки
`GetExecutingAssembly()`	Возвращает сборку, которая содержит выполняемый в текущий момент код (статический метод)
`GetExportedTypes()`	Возвращает `public`-типы, определённые в сборке
`GetFiles()`	Возвращает файлы, из которых состоит сборка
`GetLoadedModules()`	Возвращает все загруженные в память модули сборки
`GetModule()`	Получает указанный модуль сборки
`GetModules()`	Возвращает все модули, являющиеся частью сборки
`GetName()`	Возвращает объект `AssemblyName` для сборки
`GetReferencedAssemblies()`	Возвращает объекты `AssemblyName` для всех сборок, на которые ссылается данная сборка
`GetTypes()`	Возвращает типы, определённые в сборке
`Load()`	Статический метод, который загружает сборку по имени
`LoadFrom()`	Статический метод; загружает сборку из указанного файла
`LoadModule()`	Загружает внутренний модуль сборки в память

Основные элементы класса Module

Имя элемента	Описание
`Assembly`	Свойство с указанием на сборку (объект `Assembly`) модуля
`FindTypes()`	Получает массив классов, удовлетворяющих заданному фильтру
`FullyQualifiedName`	Строка, содержащая полное имя и путь к модулю
`GetType()`	Пытается выполнить поиск указанного типа в модуле
`GetTypes()`	Возвращает все типы, определённые в модуле
`Name`	Строка с коротким именем модуля

```
Assembly assembly = Assembly.GetExecutingAssembly();
            Console.WriteLine(assembly.FullName);
            foreach (Module module in assembly.GetModules())
            {
                Console.WriteLine(module.FullyQualifiedName);

              foreach (Type type in module.GetTypes())
                {
                    Console.WriteLine(type.FullName);
                }
            }
```

```
OOP_Lect, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
--------------
C:\NATALLIA\лекции\ООПС#\OOP_1\лекции\OOP_Lect\OOP_Lect\bin\Debug\OOP_Lect.exe
<>f__AnonymousType0`4
OOP_Lect.Polimorf
OOP_Lect.Polimorf+Person
OOP_Lect.Polimorf+Employee
OOP_Lect.Polimorf+SuperArray
OOP_Lect.Polimorf+SuperArray`1
OOP_Lect.Polimorf+Transaction`2
OOP_Lect.Polimorf+Point
OOP_Lect.Polimorf+ColorPoint
OOP_Lect.Polimorf+<>c
```

Позднее связывание

- Механизм отражения позволяет реализовать на платформе .NET позднее связывание (late binding).
- Обозначает процесс динамической загрузки сборок и типов при работе приложения, создание экземпляров типов и работу с их элементами.
- позволяет создавать расширяемые приложения, когда дополнительный функционал программы неизвестен, и его могут подключить сторонние разработчики

```
Assembly asm = Assembly.LoadFrom("Data.exe");
```

- System.Activator
- Activator.CreateInstance()

```
Assembly asm = Assembly.LoadFrom("Data.exe");

Type typ = asm.GetType("Data.Program");

// создаем экземпляр класса Program
object obj = Activator.CreateInstance(typ);

// получаем метод GetArray
MethodInfo method = typ.GetMethod("GetArray");

// вызываем метод, передаем значения для параметров и получаем
object result = method.Invoke(obj, new object[] { 6, 100, 3 });
    Console.WriteLine((result));
```

получаем тип

создаем его экземпляр

получаем сам метод

вызываем его

объект, для которого вызывается метод + набор параметров

```
static class Reflector
    {


        static public void Method(object obj)
        {}

        static public void Field(object obj)
        {}

        static public void Interfece(object obj)
        {}

        static public void MethodForType(object obj, Type parametr)
        {}

        static public void Voke(object obj, string methode)
        {}
    }
```