# ЛЕКЦІЯ № 4

Рефлексія та атрибути

# Поняття рефлексії

**Рефлексія** – це механізм отримання інформації про типи під час виконання програми.

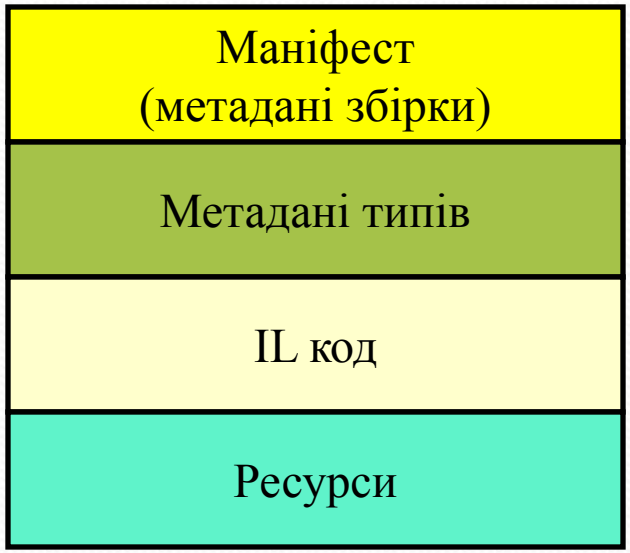
**Метадані** (у C#) – це дані про виконуваний модуль. До таких даних належать і дані про всі типи збірки.

Кожен тип (в тому числі і простий) в середовищі .NET пов'язаний зі спеціальним **об'єктом-типом**. Клас **Type** простору імен **System** грає ключову роль для отримання даних про типи збірки.

# Застосування рефлексії

1. Отримання даних про типи збірки у процесі виконання програми;
2. Пізнє зв'язування – забезпечує можливість підключення бібліотек на етапі виконання програми;
3. Створення динамічних збірок.

# Метадані у збірці




***Маніфест*** – це набір метаданих про саму збірку, включаючи інформацію про всі файли, які входять у збірку, версію збірки, а також відомості про всі зовнішні збірки, на які вона посилається.

***Метадані типів*** – це відомості про типи даних, які використані у збірці. Компілятор створює метадані типів автоматично. У них міститься інформація про кожний тип, наявний у програмі, та про кожний його екземпляр.

# Отримання екземпляру типу Type

1. Отримання екземпляра **Type** за іменем типу

   *Type t = Type.**GetType**("System.Int32");*
2. Отримання екземпляра **Type** з об’єкта класу

   *Class1 app = new Class1();*

   *Type type = app.**GetType**();*
3. Отримання екземпляра Type за допомогою методу **typeof**

   *Type myType = **typeof**(MyClass1);*
4. Отримання масиву типів поточної збірки

   *Type[] types =**Assembly.GetExecutingAssembly**().GetTypes();*
5. Отримання масиву типів зі збірки на жорсткому диску

   *Type[] types =**Assembly. Load**("C:\MyAssembly.dll").GetTypes();*

# Властивості класу Type

| Властивість | Призначення |
|---|---|
| IsAbstract | Чи є тип абстрактним об'єктом |
| IsArray | Чи є тип масивом |
| IsClass | Чи є тип класом |
| IsCOMObject | Чи є тип COM-об'єктом |
| IsEnum | Чи є тип перерахуванням |
| IsValueType | Чи є тип структурою |
| IsInterface | Чи є тип інтерфейсом |
| IsGenericTypeDefinition | Чи є тип визначенням універсального типу |
| IsGenericParameter | Чи є тип типом-параметром шаблону |
| IsPublic | Чи є поле відкритим |
| IsNotPublic | Чи не є поле відкритим |

# Методи класу Type

| Методи | Призначення |
|---|---|
| GetConstructors() | Повертає всі відкриті конструктори поточного об'єкта Type |
| GetEvents() | Повертає всі відкриті події, оголошені або успадковані в поточному об'єкті Type |
| GetFields() | Повертає всі відкриті поля поточного об'єкта Type |
| GetInterfaces() | При перевизначенні в успадкованому класі повертає всі інтерфейси, реалізовані або успадковані поточним об'єктом Type |
| GetMembers() | Повертає всі відкриті елементи поточного об'єкта Type |
| GetMethods() | Повертає всі відкриті методи поточного об'єкта Type |
| GetProperties() | Повертає всі відкриті поля поточного об'єкта Type |
| GetField(String) | Виконує пошук відкритого поля із заданим ім'ям |
| GetMember(String) | Виконує пошук відкритого елемента із заданим ім'ям |

# Отримання закритих членів класу

```
static void Main(string[] args){
    Assembly assembly =
    Assembly.LoadWithPartialName("System.Drawing");
    Type type = assembly.GetType(
        "System.Drawing.SystemFonts");
    MemberInfo[] members = type.GetMembers(
        BindingFlags.Instance | BindingFlags.NonPublic);
    foreach (MemberInfo member in members){
        Console.WriteLine( member.Name + "\n" );
    }
    Console.ReadLine();
}
```

# BindingFlags

| Ім’я прапорця | Призначення |
|---|---|
| BindingFlags.Instance | Повертає відкриті нестатичні члени класу |
| BindingFlags.Static | Повертає статичні члени класу |
| BindingFlags.Public | Повертає відкриті члени класу |
| BindingFlags.NonPublic | Повертає закриті члени класу |

# Приклад отримання вкладених у збірку типів

```
class Program{
    static void Main(string[] args){
        Type[] types = Assembly.LoadWithPartialName("System.Drawing").GetTypes();
        foreach (Type type in types){
            string s = type.ToString();
            if (type.IsClass){
                s = "клас " + s;
                Console.WriteLine(s);
            }else if (type.IsInterface){
                s = "інтерфейс " + s;
                Console.WriteLine(s);
            }else if (type.IsEnum){
                s = "перерахування " + s;
                Console.WriteLine(s);
            }else if (type.IsValueType){
                s = "структура " + s;
                Console.WriteLine(s);
            }
        }
        Console.ReadLine();
    }
}
```

# Приклад отримання конструкторів класу

```
class Program{
    static void Main(string[] args){
        Assembly assembly = Assembly.LoadWithPartialName("System.Drawing");
        Type type = assembly.GetType("System.Drawing.SystemFonts");
        ConstructorInfo[] constructors = type.GetConstructors();
        foreach (ConstructorInfo constructor in constructors){
      String s = "";
      ParameterInfo[] parametrs = constructor.GetParameters();
      foreach (ParameterInfo parametr in parametrs){
            s = s + parametr.ParameterType.Name + ", ";
      }
      s = s.Trim();
      s = s.TrimEnd(',');
      string res  = type.Name + "(" + s + ")";
      Console.WriteLine(res);
        }
        Console.ReadLine();
    }
}
```

# Динамічний виклик методу без параметрів

```
public class Program{
  static void Main(string[] args){
    Assembly a = null;
    try{
      a = Assembly.Load("CarLibrary");
    }catch( FileNotFoundException e){
      Console.WriteLine( e.Message );
      Console.ReadLine();
      return;
    }
    Type miniVan =a.GetType("CarLibrary.MiniVan");
    object obj = Activator.CreateInstance(miniVan);
    MethodInfo mi = miniVan.GetMethod("TurboBoost");
    mi.Invoke(obj, null);
  }
}
```

# Динамічний виклик методу з параметрами

```
public class Program{
  static void Main(string[] args){
    Assembly a = null;
    try{
      a = Assembly.Load("CarLibrary");
    }catch( FileNotFoundException e){
      Console.WriteLine( e.Message );
      Console.ReadLine();
      return;
    }
    Type miniVan =a.GetType("CarLibrary.MiniVan");
    object obj = Activator.CreateInstance(miniVan);
    object[] paramArray = new object[2];
    paramArray[0] = "Ford";
    paramArray[1] = 4;
    MethodInfo mi = miniVan.GetMethod("TurboBoost");
    mi.Invoke(obj, paramArray);
  }
}
```

# Алгоритм динамічного створення збірок

1. Згенерувати збірку;
2. На основі цієї збірки згенерувати модуль;
3. На основі цього модуля згенерувати тип (наприклад, клас);
4. На основі цього типу (класу) згенерувати його елементи (конструктори, методи і т.п.);
5. Виконати безпосереднє створення типу.

# Динамічне створення збірки (1)

```
// створення імені збірки
AssemblyName an = new AssemblyName("MyAssembly");
an.Version = new Version("1.0.0.0");

// створення збірки
AssemblyBuilder ab = AppDomain.CurrentDomain.DefineDynamicAssembly(an,
    AssemblyBuilderAccess.Save);

// створення модуля у збірці
ModuleBuilder mb = ab.DefineDynamicModule("MyModule", "My.dll");

// створення типу у збірці
TypeBuilder tb = mb.DefineType("MyClass", TypeAttributes.Public);

// створення конструктора без параметрів
ConstructorBuilder cb0 = tb.DefineConstructor(MethodAttributes.Public,
    CallingConventions.Standard, null);

// додавання коду для конструктора
ILGenerator il0 = cb0.GetILGenerator();
il0.Emit(OpCodes.Ret);
```

# Динамічне створення збірки (2)

```
// створення конструктора з параметром типу string
ConstructorBuilder cb = tb.DefineConstructor(MethodAttributes.Public, CallingConventions.Standard,
     new Type[] { typeof(string) });

// додавання коду для конструктора
ILGenerator il = cb.GetILGenerator();
il.EmitWriteLine("Constructor");
il.Emit(OpCodes.Ret)

// безпосереднє створення типу
tb.CreateType();

// збереження типу у файл
ab.Save("My1.dll");
```

# Визначення атрибуту

**Атрибут** – деяка додаткова інформація про будь-який елемент програми.

Ця інформація може відноситися до різних елементів мови програмування – типів (класів, перерахувань та ін.), полів, конструкторів, методів та деяких інших конструкцій мови програмування.

**Атрибут** є спеціальним видом класу, що є нащадком класу ***System.Attribute***.

# Атрибути у програмі

| Часто використовувані атрибути | |
|---|---|
| [Obsolete] | Вказує на застарілий тип або елемент |
| [Serializable] | Вказує на можливість серіалізації класу або структури |
| [NonSerializable] | Поле класу або структури не зберігається у результаті серіалізації |
| [WebMethod] | Вказує на доступність методу через запити HTTP |

Приклад

```
[Serializable]
[Obsolete("Клас є застарілим!!!")]
public class Motorcycle{
   [NonSerializable]
   float weightOfCurrentPassengers;

   bool hasRadioSystem;
   bool hasHeadSet;
   bool hasBar;
}
```

# Користувацькі атрибути

```
public sealed class VehicleDescriptionAttribute : System.Attribute{
    private string msgData;

    public VehicleDescriptionAttribute(string msgData){
       this. msgData = msgData;
    }

    public VehicleDescriptionAttribute(){}

    public string Description{
       get{return msgData;}
       set{msgData = value;}
    }
}
```

# Отримання користувацьких атрибутів

```
[VehicleDescription("Дуже довге та повільне авто, але занадто дороге")]
class Winnebago
{
}

static void Main(string[] args){
    // отримуємо об’єкт Type для класу Winnebago
    Type t = typeof(Winnebago);
    // отримуємо усі атрибути даної збірки
    object[] customAtts = t.GetCustomAttributes(false);
    // виводимо інформацію про кожний атрибут на консоль
    foreach(VehicleDescriptionAttribute v in customAtts)
        Console.WriteLine(v.Desc);
    }
}
```