Reflection in .Net

WEEK 10 LECTURE 01

MURTAZA MUNAWAR FAZAL

What is Reflection

Reflection is the feature in .Net, which enables us to get some information about object in runtime.

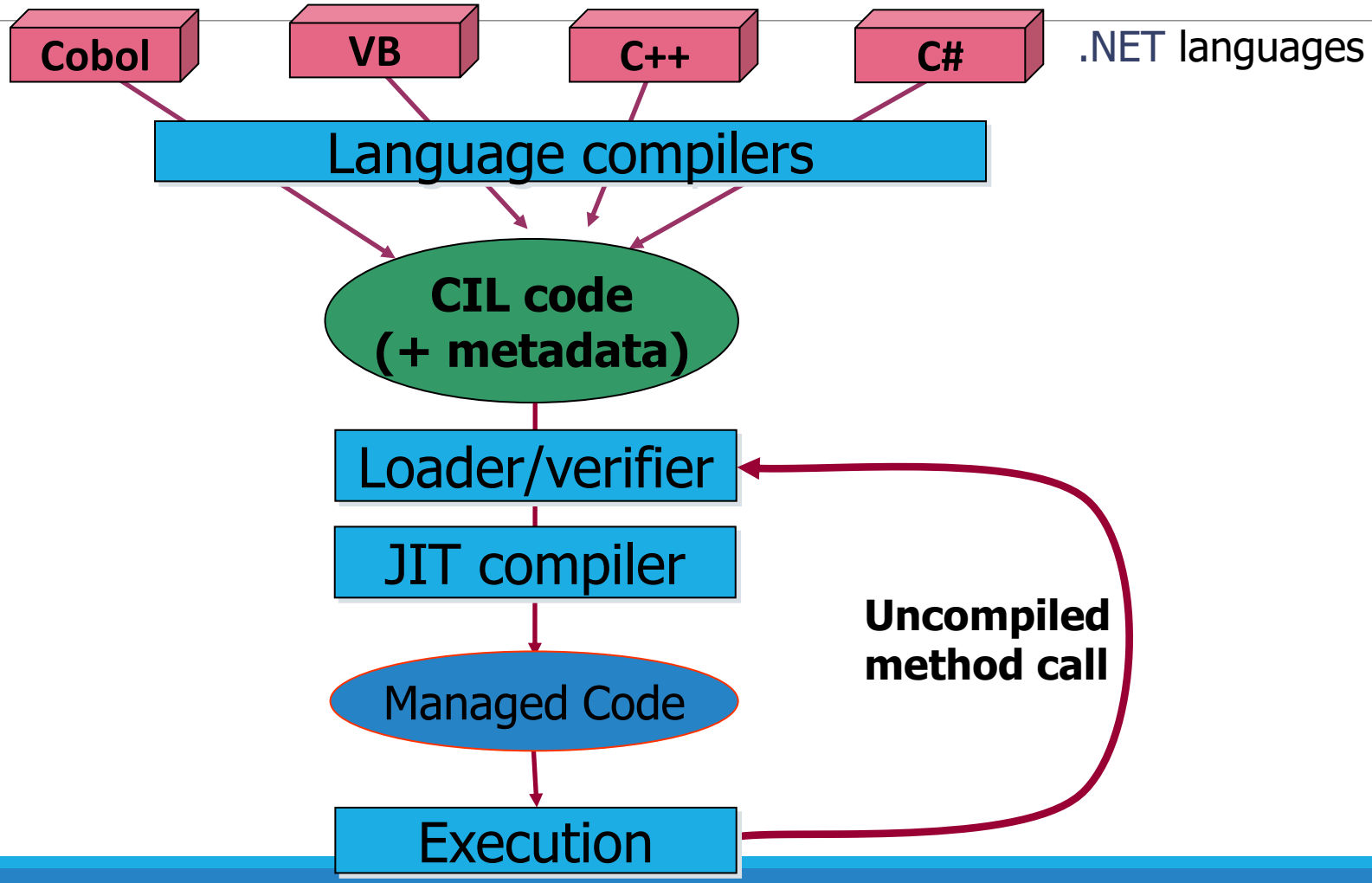
Information can be:

- Data of the class
- Names of the methods
- Constructors of that object

Uses of Reflection

- explore assembly metadata
- creating objects dynamically
- invoking methods dynamically
- write “generic” code that works with different types
- implement sophisticated programming techniques

.Net Execution Model



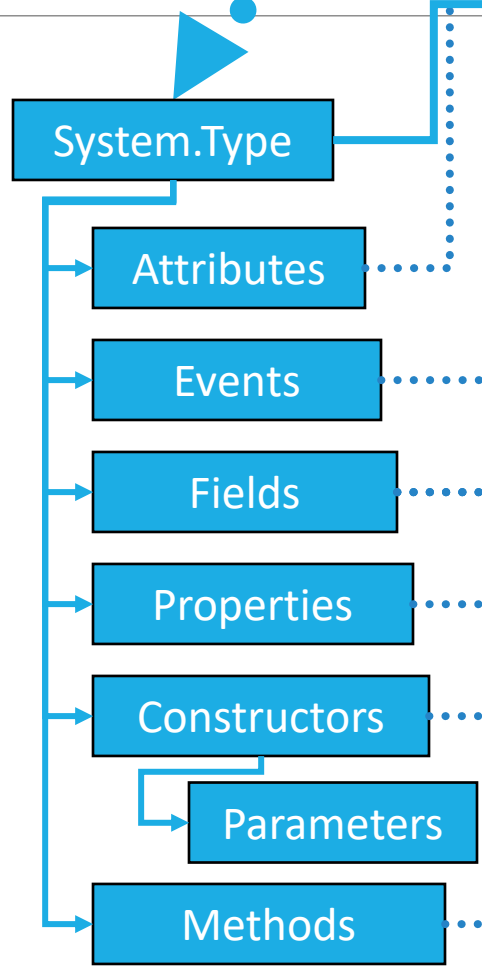
Metadata

- Metadata
  - Single location for type information and code
  - Types' metadata can be explored with Reflection
  - Code is literally contained within type information
  - Every .NET object can be queried for its type

Uses of Metadata

- Dynamic Type System
  - Highly dynamic and language independent
  - Types may be extended and built at run time
  - Allows on-the-fly creation of assemblies
  - .NET Compilers use .NET to emit .NET code

MetaData: Type Info at Runtime




```
[serializable]
public class Person :
{
    public event OnSaveChange onsv;
    public Date DOB;
    public string FirstName;
    public string LastName;
    public string Name {
        get {
            return FirstName + " " + LastName;
            }
    }
    public Person(string First,string Last)
    {
        FirstName=First;LastName=Last;
    }
    public bool Save()
    {
        System.Type t = this.GetType();
        foreach( FieldInfo f in t.GetFields() )
        { ... }
    }
```

Accessing meta-data: System.Object.GetType()

- All .NET classes (implicitly) inherit System.Object
- Available on every .NET class; simple types too

Explicit language support for type meta-data

- C#, JScript.NET: typeof(…)
- VB.NET: If TypeOf … Is … Then …

Determining Type Identity

- Types have unique identity across any assembly
- Types can be compared for identity
  - if (a.GetType() == b.GetType()) { … };

Viewing metadata

Add custom attributes to a compiled executable’s metadata

Why/When to use this?

Reflection System.Type

Provides access to metadata for any .NET type

Returned by `System.Object.GetType()`

Allows drilling down into all facets of a type

- Category: Simple, Enum, Struct or Class
- Methods and Constructors, Parameters and Return
- Fields and Properties, Arguments and Attributes
- Events, Delegates, and Namespaces

Example 1 (Reflection)

file:///c:/users/mfazal/documents/visual studio 2015/Project...

```
Void .ctor()
Void .ctor(Int32)

Int32 Inc()
Int32 Dec()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()
```

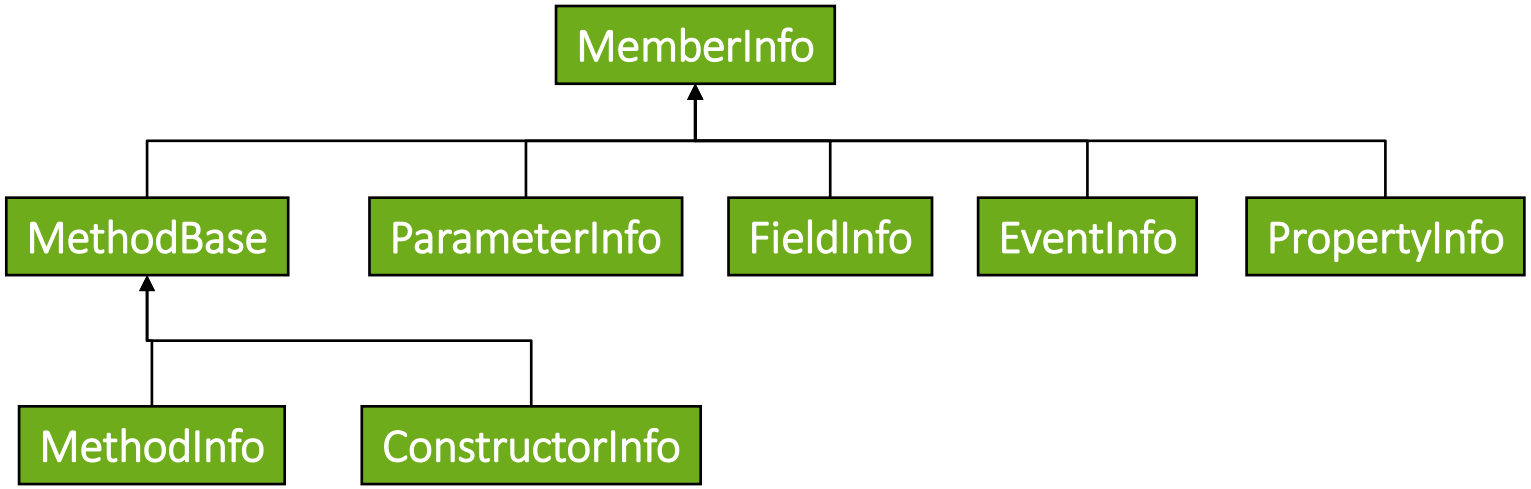
Reflection

MemberInfo

Base class for all "member" element descriptions

- Fields, Properties, Methods, etc.

Provides member kind, name, and declaring class



Reflection Attributes

Custom attributes are the killer-app for Reflection!

Attributes enable declarative behavior

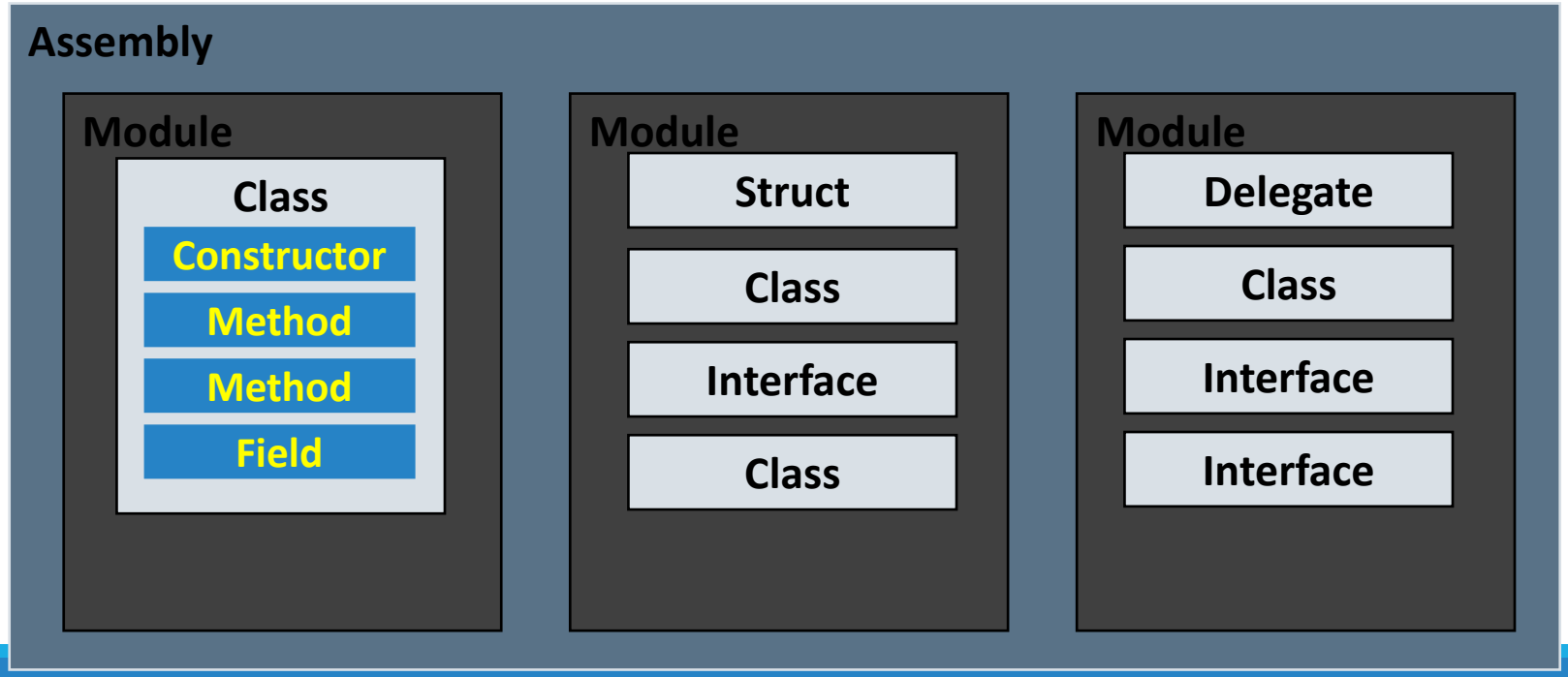
Attributes allow data augmentation

Reflection The Bigger Picture

Types know their module; modules know their types

Modules know their assembly and vice versa

Code can browse and search its entire context



Scenario

Suppose that your organization wants to keep track of bug fixes. You already keep a database of all your bugs, but you'd like to tie your bug reports to the fixes in the code.

How do implement this?

Code (Reflection Example 2)

Reflecting on a Type

```
public static void Main()
    {
        // examine a single object
        Type theType =
          Type.GetType(
            "System.Reflection.Assembly");
        Console.WriteLine(
          "\nSingle Type is {0}\n", theType);
    }
```

Late binding to methods and properties

Performing late binding by dynamically instantiating and invoking methods on types.

Creating types at runtime

The ultimate use of reflection is to create new objects/types at runtime and then to use those objects to perform tasks. You might do this when a new class, created at runtime, will run significantly faster than more generic code created at compile time.

Problems

Reflection APIs are known to cause problem on obfuscated assemblies.

Summary

Reflection = System.Type + GetType()

Explore Type Information at Runtime

Enables Attribute-Driven Programming

Use Emit Classes to Produce .NET Assemblies

Bottom Line: Fully Self-Contained Structural Model