



# .Net Reflection

Taipan Tamsare

# Overview

- Reflection core concepts
- Exploring metadata
- Detail information
- Attributes
- Building Types at runtime

# 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

# Why reflection

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

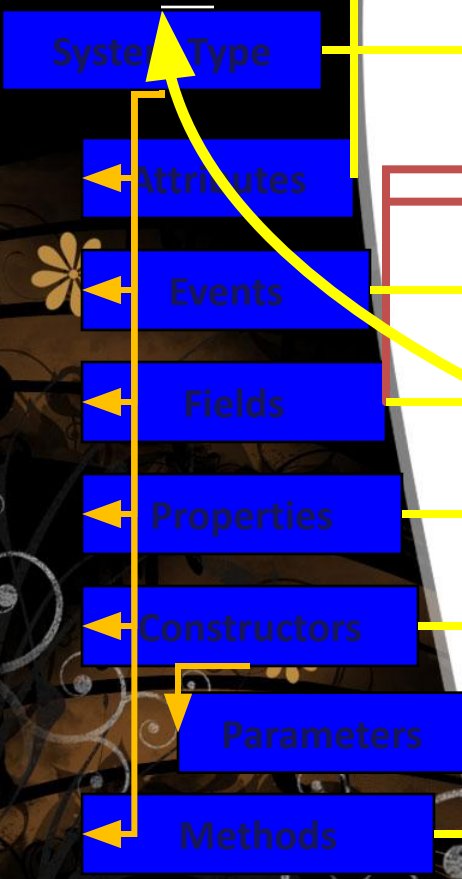



# Reflection Core Concepts

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

# Exploring Metadata

```
[serializable]
public class Person :
{
    public event OnSaveChange onsv;
    public Date DOB;
    public string FirstName;
    public string LastName;
    public string Name {
        get {
            return F..+ " " + L..;
        }
    }
    public void Person(string F.,string
    L.)
    {
        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() ) { ... };`

# 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

```
public class TestDataType
{
    public TestDataType()
    {
        counter = 1;
    }

    public TestDataType(int c)
    {
        counter = c;
    }

    private int counter;

    public int Inc()
    {
        return counter++;
    }
    public int Dec()
    {
        return counter--;
    }
}
```

```
Type objectType = testObject.GetType();
```

```
    ConstructorInfo [] info = objectType.GetConstructors();
    MethodInfo [] methods = objectType.GetMethods();
```

```
    // get all the constructors
```

```
    Console.WriteLine("Constructors:");
    foreach( ConstructorInfo cf in info )
    {
        Console.WriteLine(cf);
    }
```

```
    Console.WriteLine();
```

```
    // get all the methods
```

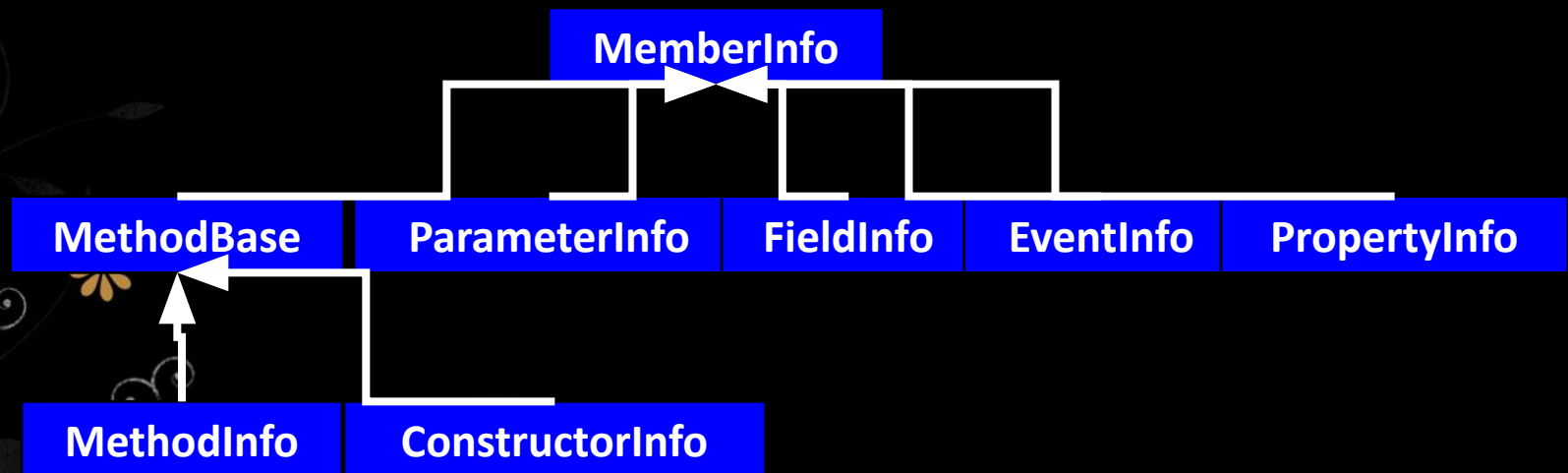
```
    Console.WriteLine("Methods:");
    foreach( MethodInfo mf in methods )
    {
        Console.WriteLine(mf);
    }
```

More [example](#)

# Reflection

## MemberInfo

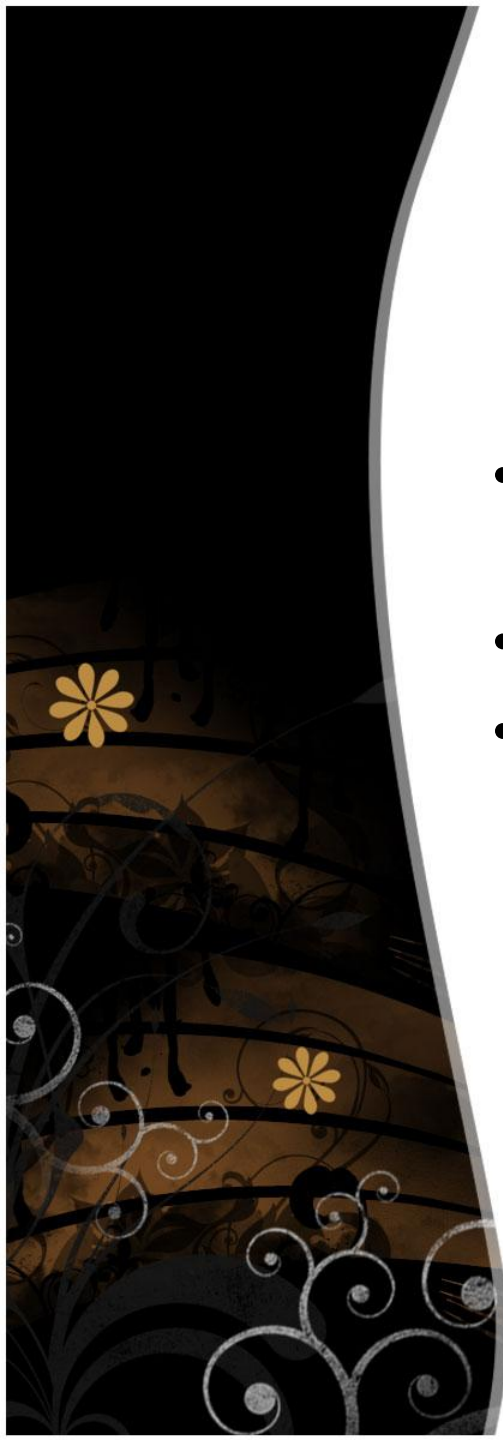
- Base class for all "member" element descriptions
  - Fields, Properties, Methods, etc.
- Provides member kind, name, and declaring class



# Reflection

## Attributes

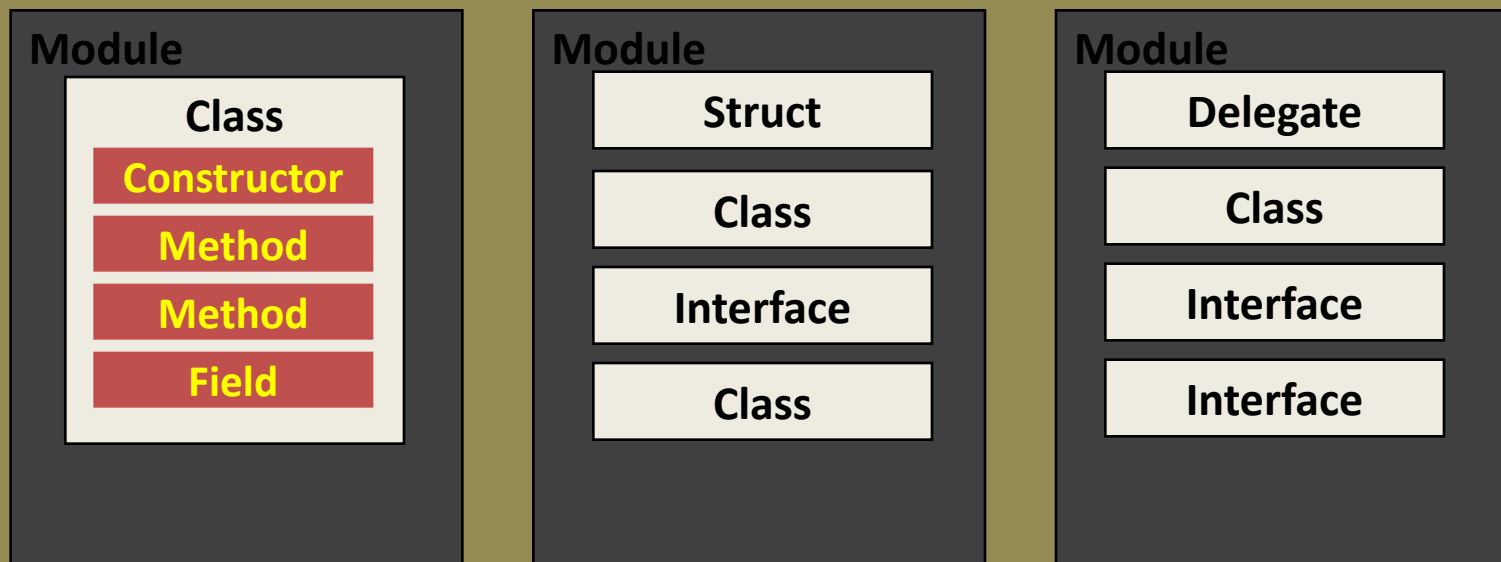
- Custom attributes are the killer-app for Reflection!
- Attributes enable declarative behavior
- Attributes allow data augmentation



# 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

## Assembly



# Building Types at Runtime

`System.Reflection.Emit`

- Full representation of physical structure
- Allows building modules and assemblies at run time
- ✿ – Transient code only used at run time
- Persistent code for reuse
- Create classes and types, and emit IL
- Used by .NET compilers to build .NET apps
- [Example](#)

# 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

# References

- Microsoft
- Assoc. Prof. Pan Wuming
- [http://www.codersource.net/csharp\\_tutorial\\_reflection.html](http://www.codersource.net/csharp_tutorial_reflection.html)
- <http://www.csharp-examples.net/reflection-examples/>