Components Based Design and Development

Computer Engineering Studies

Universidad Carlos III de Madrid

Components Practice

Reflection

Juan Llorens

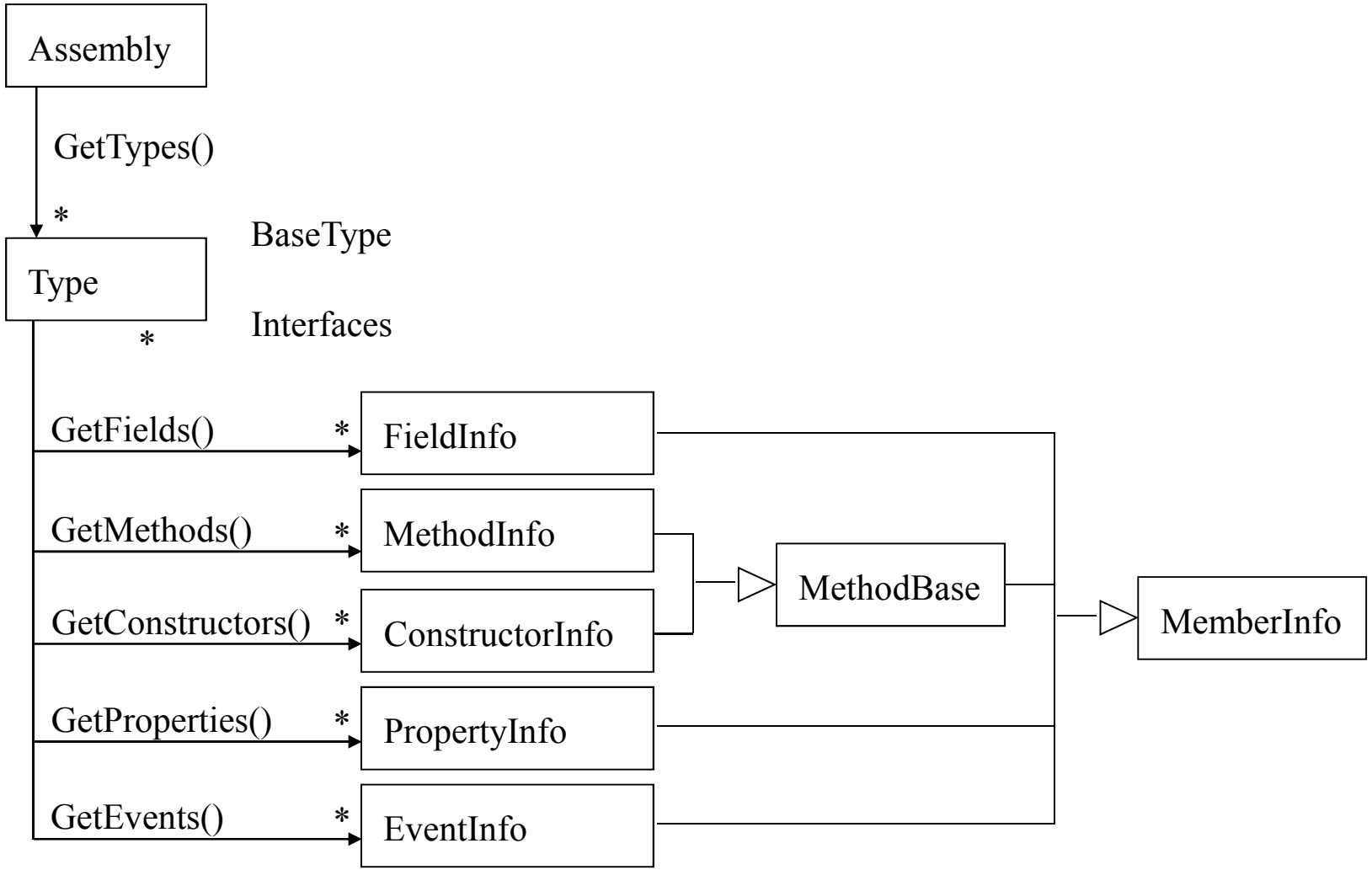
Högskolan på Åland – Finland / Universidad Carlos III de Madrid - Spain

Juan.llorens@uc3m.es

Reflection

- Permits access to meta-information of types at run-time
- **System.Reflection** allows:
  - Getting meta-information about assemblies, modules and types
  - Getting meta-information about the members of a type
  - Dynamic creation of instances of a type at run-time
  - Search for methods and their dynamic invocation at run-time
  - Accessing values of properties and fields of an object
  - Design of new types at run time - namespace **System.Reflection.Emit**

Reflection Class Hierarchy



Class Assembly

- Class Assembly loads assemblies and their meta-data
- Provides access to its meta-data

```
public class Assembly {

public static Assembly Load(string name);

public virtual string FullName {get;}
public virtual string Location {get;}
public virtual MethodInfo EntryPoint {get;}

public Module[] GetModules();
public virtual Type[] GetTypes();
public virtual Type GetType(string typeName);
```

- Loading an assembly
- Name, storage location, entry point of the assembly
- Getting modules and all in the assembly defined types
- Getting type with name typeName
- Creation of an object of type typeName

Class Type

- Type used for meta-description of all types in the run-time system
- Provides access to the meta-information about its members

```
public abstract class Type : MemberInfo, IReflect {

    public abstract string FullName {get;};
    public abstract Type BaseType {get;};
    public Type[] GetInterfaces();

    public bool IsAbstract {get;};
    public bool IsClass {get;};
    public bool IsPublic {get;};
    …

    public ConstructorInfo[] GetConstructors();
    public virtual EventInfo[] GetEvents();
```

- Type name
- Direct base type
- List of implemented interfaces

- Properties of type

- Getting constructors,
  events,
  fields,
  methods,
  properties
- Optionally parameterized by **BindingFlags**

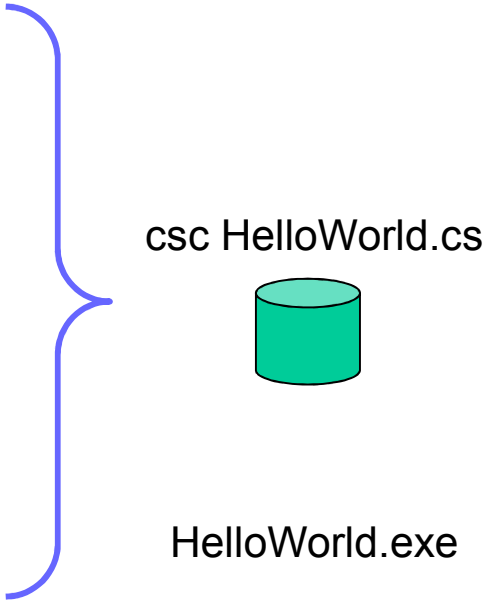
Example Reflection

- C# program "HelloWorld"

```
namespace Hello {
    using System;
    public class HelloWorld {

        public static void Main(string[] args) {
            Console.WriteLine("HelloWorld");
        }

        public override string ToString() {
            return "Example HelloWorld";
        }
    }
}
```



- Loading the assembly "HelloWorld.exe":

```
Assembly a = Assembly.Load("HelloWorld");
```

Example Reflection

- Print all existing types in a given assembly

```
Type[] types = a.GetTypes();
foreach (Type t in types)
    Console.WriteLine(t.FullName);
```

Hello.HelloWorld

- Print all existing methods of a given type

```
Type hw = a.GetType("Hello.HelloWorld");
MethodInfo[] methods = hw.GetMethods();
foreach (MethodInfo m in methods)
    Console.WriteLine(m.Name);
```

GetType

ToString

Equals

GetHashCode

Attributes with Parameters

Example

positional parameter

name parameters
come after pos. parameters

```
[Obsolete("Use class C1 instead", IsError=true)]   // causes compiler message saying
public class C {...}                               // that C is obsolete
```

Positional parameter = parameter of the attribute's constructor
Name parameter = a property of the attribute

Attributes are declared as classes

```
public class ObsoleteAttribute : Attribute {          // class name ends with "Attribute"
    public string Message { get; }                    // but can be used as "Obsolete"
    public bool IsError { get; set; }
    public ObsoleteAttribute() {...}
    public ObsoleteAttribute(string msg) {...}
    public ObsoleteAttribute(string msg, bool error) {...}
}
```

Valid variants:

```
[Obsolete]                                        // Message == "", IsError == false
[Obsolete("some Message")]                        // IsError == false
[Obsolete("some Message", false)]
[Obsolete("some Message", IsError=false)]
```

values must be constants

AttributeUsage

***AttributeUsage* describes how user-defined attributes are to be used**

```
public class AttributeUsageAttribute : Attribute {
    public AttributeUsageAttribute (AttributeTargets validOn) {...}
    public bool AllowMultiple { get; set; }          // default: false
    public bool Inherited { get; set; }              // default: false
}
```

Usage

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Interface, AllowMultiple=false)]
public class MyAttribute : Attribute {
    ...
}
```

Defining Your Own Attributes

Declaration

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Interface, Inherited=true)]
class CommentAttribute : Attribute {
    string text, author;
    public string Text { get {return text;} }
    public string Author { get {return author;} set {author = value;} }
    public Comment (string text) { this.text = text; author ="HM"; }
}
```

Usage

```
[Comment("This is a demo class for Attributes", Author="XX")]
class C { ... }
```

Querying the attribute at runtime

```
class Attributes {

static void Main() {
  Type t = typeof(C);
object[] a = t.GetCustomAttributes(typeof(Comment), true);
foreach (Comment ca = a) {
```

Example Reflection

- Create a new instance of a given type

```
Assembly a = Assembly.Load("HelloWorld");
object o = a.CreateInstance("Hello.HelloWorld");
```

- Get method ToString(), which has no parameters
- Invoke the method

```
Type hw = a.GetType("Hello.HelloWorld");
MethodInfo mi = hw.GetMethod("ToString");
object retVal = mi.Invoke(o, null);
```

Example: Handling plug-ins

- "Plug-in" is any class implementing IMyPlugin interface

```
string[] files = Directory.GetFiles(path, "*.dll");          // returns full paths
foreach (string f in files) {
    Assembly a = Assembly.LoadFile(f);
    Type[] types = a.GetTypes();
    foreach (Type t in types) {
      if (t.IsClass) {
            if (t.GetInterface("IMyPlugin") != null) {
                  IMyPlugin p = (IMyPlugin) Activator.CreateInstance(t);
                  // add p to list of all installed plug-ins
            }
      }
    }
}
```