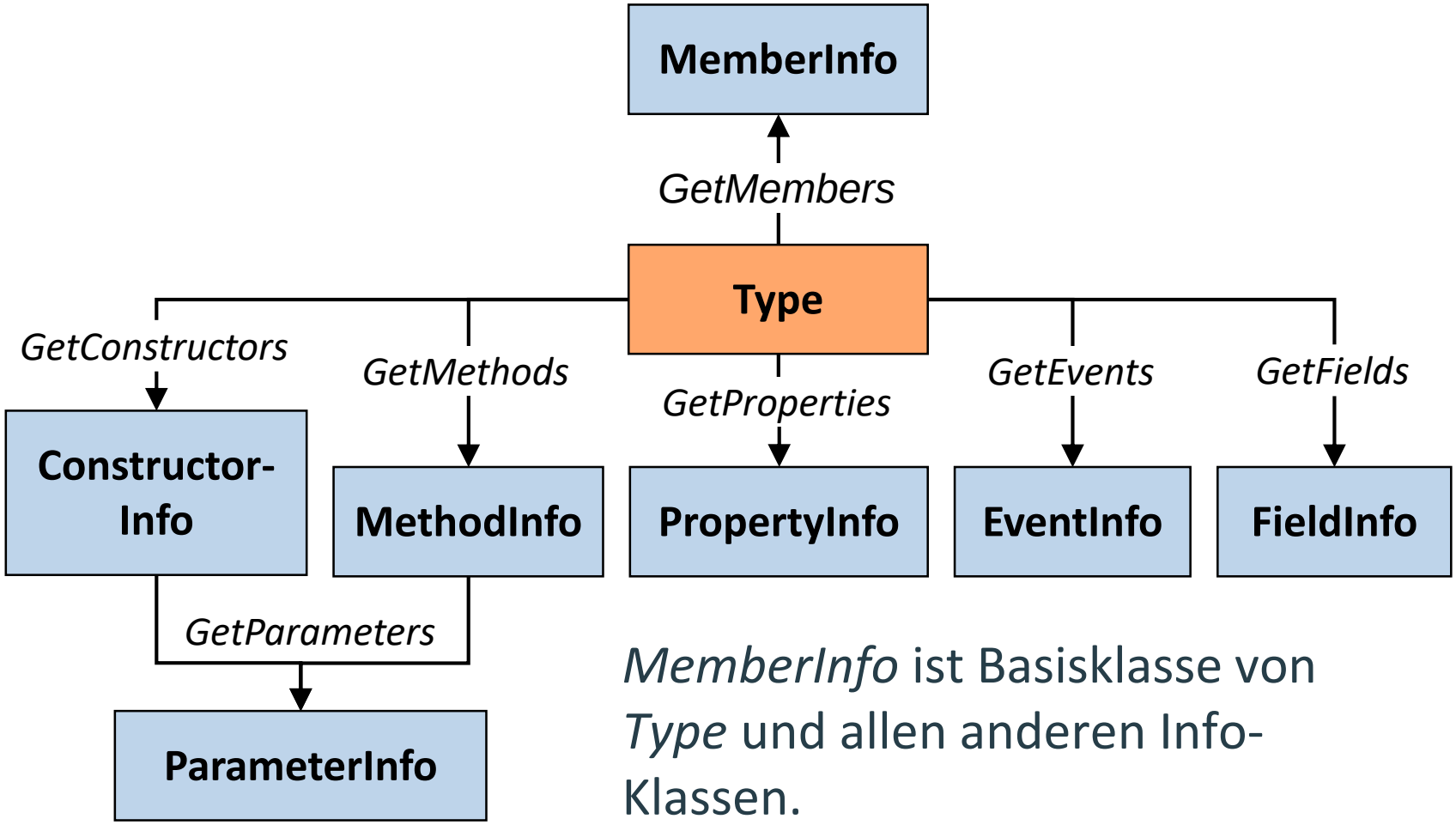
.NET: Reflection und Attribute

Version 5.0

Metadaten, Reflection und Attribute

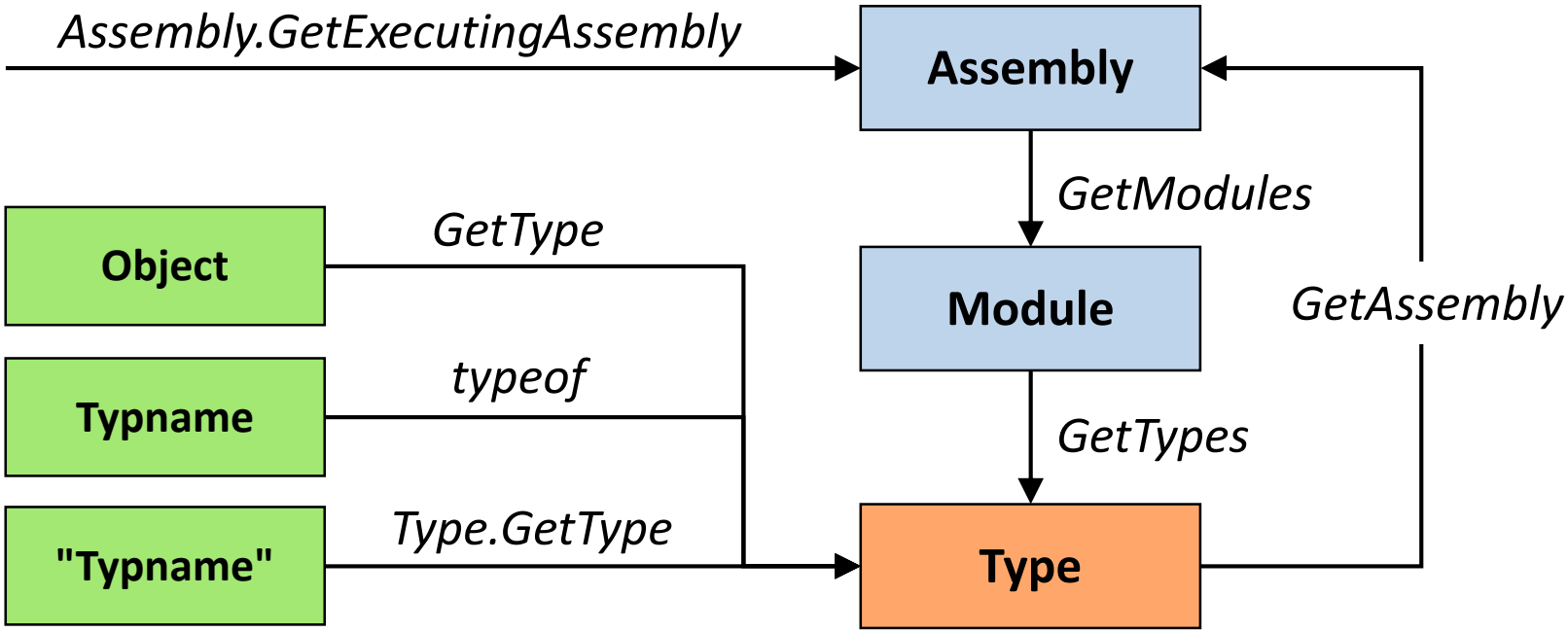
- *Metadaten* enthalten eine detaillierte Beschreibung aller Bestandteile eines Assemblies:
  - *Assembly:* exportierte Typen eines Assemblies (Manifest).
  - *Modul:* definierte Typen (Klassen, Strukturen, Delegates, …)
  - *Typ:* Felder, Konstruktoren, Methoden, Properties, Events.
- Mit *Reflection*
  - kann auf Metadaten zugegriffen werden,
  - können Typen zur Laufzeit erzeugt und instanziert werden.
- Mit Attributen können Metadaten erweitert werden.
  - In der FCL sind bereits viele Attribute definiert (*DllImport*, *MarshalAs*, *STAThread*, *Serializable*, *CLSCompliant, WebMethod, …*).
  - Entwickler können neue Attribute definieren.

Reflection (1)



MemberInfo ist Basisklasse von *Type* und allen anderen Info-Klassen.

Reflection (2)



Beispiel:

```
string s = "abc";
Type t1 = s.GetType();
Type t2 = typeof(string);
Type t3 = Type.GetType("System.String");
```

Reflection: Beispiel

```
Type t = typeof(string);
MemberInfo[] members = t.GetMembers();
foreach(MemberInfo mi in members)
  Console.WriteLine("{0} {1}",
    mi.MemberType, mi.Name);
```

```
Field Empty
Method ToString
Method Equals
Method ToString
…
Constructor .ctor
Property Length
```

Anwendung von Attributen

- Konstrukte, denen Attribute zugeordnet werden können:

(1) Assembly
(2) Modul
(3) Typ
(4) Feld
(5) Methode
(6) Rückgabewert
(7) Parameter
(8) Property
(9) Event

```
1 [assembly: AssemblyVersion("1.0.0.0")]
2 [module: CLSComplient(true)]
3 [Serializable]
  public class MyClass {
    4 [NonSerialized]
      private int field;
    5 [DllImport("msvcrt.dll")]
    6 [return: MarshalAs(UnmanagedType.I4)]
      public static extern int puts(
    7   [MarshalAs(UnmanagedType.LPStr)]
        string m);
    8 [Browsable(true), Category("Misc")]
      public string MyProperty { … }
    9 [Browsable(true)]
      public event EventHandler MyEvent;
  }
```

Definition von Attributen

- Vorgehensweise
  - Klasse von System.Attribute ableiten.
  - Festlegung, worauf Attribut angewendet werden kann (AttributeTargets).
  - Konvention: Klassenname endet mit Postfix Attribute.
  - Definition von Konstruktoren und Properties.
  - Konstruktoren und Properties definieren Attributparameter.
- Beispiel:

```
[AttributeUsage(AttributeTargets.Method |
                AttributeTargets.Property]
public class MyAttributeAttribute : System.Attribute {
  public MyAttributeAttribute(string param1) { … }
  public string Param1 { get { … }; }
  public int    Param2 { get { … }; set { … } }
}
```

Zuweisung von Attributen

- Bei der Zuweisung eines Attributs
  - müssen die Parameter des Konstruktors als *Positionsparameter* angegeben werden,
  - können die Properties als benannte Parameter gesetzt werden.

```
class MyClass {
  [MyAttribute("param1")]
  void MyFirstMethod() { … }
  [MyAttribute("param1", Param2=5)]
  void MySecondMethod() { … }
}
```

- Es können auch mehrere Attribute zugewiesen werden.

```
[WebMethod,CLSCompliant][MyAttribute("param1")]
void MyMethod();
```

Zugriff auf Attribute

- Mit `MemberInfo.IsDefined(`*`attributeType`*`, `*`inherit`*`)` kann fest-gestellt werden, ob ein bestimmtes Attribut zugewiesen wurde.

```
Type t = typeof(MyClass);
if (t.IsDefined(typeof(MyAttributeAttribute), true) { … }
```

- `MemberInfo.GetCustomAttributes(attributeType, inherit)` liefert alle zugewiesenen Attributinstanzen.

```
Type t = typeof(MyClass);
MethodInfo mInfo = t.GetMethod("MyMethod");

object[] attr = mInfo.GetCustomAttribues(
                    typeof(MyAttributeAttribute), true);
if (attr.Length >= 1) {
  MyAttributeAttribute myAttr =
                    (MyAttributeAttribute)attr[0];
  string p1 = myAttr.Param1;
  int    p2 = myAttr.Param2;
}
```