Worksheet – Reflection

Lernziele:

Sie können die Konzepte des Reflection für die Analyse von Assemblies und für die dynamische Instanzierung von Objekten einsetzen.

Aufgabe 1 – Assembly Analyzer

Mit Hilfe von Reflection lassen sich beliebige Assemblies analysieren und beispielsweise sämtliche darin definierten Typen ermitteln. Sie sollen nun schrittweise einen einfachen Assembly-Analyzer entwickeln.

Sie finden auf dem AD das Zip-File AssemblyAnalyzerStud.zip. Es enthält das Projekt AssemblyAnalyzer. Die enthaltene Methode PrintTypes ist noch leer, und soll in den folgenden Aufgaben um die beschriebenen Analysen erweitert werden.

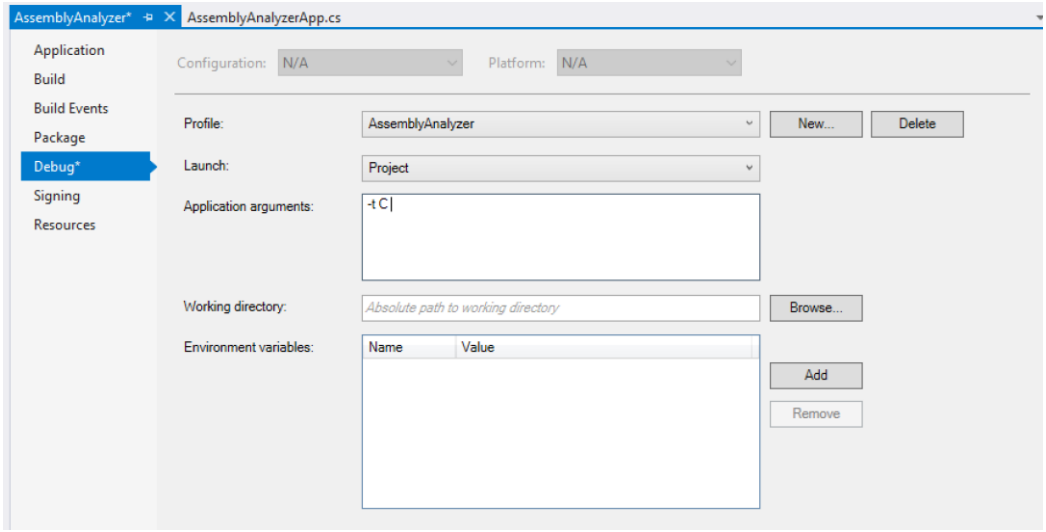
a) Implementieren Sie die PrintTypes-Methode der Applikation `AssemblyAnalyzer`, so dass alle Typen einer Assembly auf der Konsole ausgegeben werden. Verwenden Sie für Ihre Tests `MysteryAssembly.dll` vom Netzwerkshare.

b) Ergänzen Sie die PrintTypes-Methode so, dass Sie flexibel mittel Kommandozeilen Argument bestimmen können, ob alle Typen, nur Klassen oder nur Interfaces ausgegeben werden.

```
Syntax: AssemblyAnalyzer -t <Option>
<Option>:
A: alle Typen (default)
C: nur Klassen
I: nur Interfaces
```

Der Parameter wird der PrintTypes-Methode als typesToPrint bereits übergeben.

Das übergeben des Parameters an die Main-Methode, können Sie dabei wie folgt im Visual Studio simulieren:



c) Erweitern Sie die PrintTypes-Methode so, dass Sie als weiteres Argument (Parameter printMethods) angeben können, ob auch Methoden-Namen eines Typs ausgeben werden.

Hinweis: alle Methoden, unabhängig von ihrer Sichtbarkeit

```
Syntax: AssemblyAnalyzer -m true|false
false: Methodennamen werden nicht ausgegeben (default)
true: Methodennamen werden ausgegeben
```

Für das Auslesen des übergebenen Wertes steht Ihnen im Programm die Methode GetMethodArg() zur Verfügung. Falls kein Wert übergeben wird, wird ein Default-Wert übergeben.

d) Erweitern Sie nun den Analyzer in der Main-Methode so, dass er alle Assemblies einliest, die in einem Verzeichnis abgelegt sind. Den Pfad erhalten Sie dabei als Konsolen-Parameter:
`Syntax: AssemblyAnalyzer -f <directoryname>`

Der Parameter für die Pfadangabe wird Ihnen in der Main-Methode als Variable `assemblyDir` bereitgestellt.

Aufgabe 2 – Objekt-Instanzierung

Mit Reflection können Sie nicht nur Meta-Informationen aus Assemblies lesen, sondern auch beliebige Typen dynamisch instanziieren und Methoden ausführen.

Sie finden auf dem Netzwerkshare den SimpleCalculator und den ScientificCalculator. Beide DLL's beinhalten eine Klasse welche das ICalculator Interface implementiert. Das ICalculator Interface befindet sich dabei in der CalculatorInterface.dll.

```
namespace Calculators
{
    public interface ICalculator
    {
        double Add(double a, double b);
        double Sub(double a, double b);
        double Mult(double a, double b);
        double Div(double a, double b);
    }
}
```

Sie sollen in dieser Übung eine Applikation schreiben, die eine beliebige Implementation dieses Interfaces verwenden kann. Die Applikation soll die Klasse `SimpleCalculator` aus der Assembly laden, instanziieren und anschliessend die Methode `Add` zur Ausführung bringen, ohne dass der Code etwas über die Klasse `SimpleCalculator` wissen muss. So könnte man später `SimpleCalculator.dll` durch eine beliebige, andere `ICalculator`-Implementation von Drittentwicklern ersetzen.

a) Legen Sie eine neue Applikation an. Welche der beiden Assemblies vom Netzwerkshare müssen Sie im Projekt referenzieren, und wieso?

b) Schreiben Sie zunächst eine Methode `Type GetTypeFromAssembly()`, die den Typ mit dem Namen «`Calculators.SimpleCalculator`» aus der Assembly `SimpleCalculator.dll` lädt. Überprüfen Sie, dass der geladene Typ tatsächlich das Interface `ICalculator` implementiert, sonst soll eine entsprechende Fehlermeldung ausgegeben werden.

c) Instanzieren Sie ein Objekt dieses Typs.

d) Führen Sie die `Add(…)`-Methode mittels `MethodBase.Invoke(…)` oder `Type.InvokeMember(…)` aus. Das Ergebnis soll auf der Konsole ausgegeben werden.

e) Erweitern Sie ihren Code so, dass Sie anstelle der SimpleCalculator.dll die ScientificCalculator.dll laden und ausführen können.

f) Erweitern Sie ihre Applikation so, dass der Name oder der Pfad der zu ladenden Assembly übergeben werden kann (Analog Aufgabe 1d). Damit können verschiedenste Implementationen Ihres Calculators abhängig vom übergebenen Wert beim Starten der Applikation geladen werden.

   Damit sollte Ihre Applikation komplett unabhängig von einem der beiden Typen sein und keine Referenz auf die konkreten Calculator-Implementierungs-Assemblies mehr enthalten!

Gratulation - damit haben Sie einen Plugin-Mechanismus implementiert!