REFLECTION

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

REFLECTION

- Reflection ermöglicht es einem Programm, zur Laufzeit Informationen von Klassen zu sammeln und diese zu manipulieren
- Ermöglicht z.B. Lesen von Attributen, das Erstellen von Instanzen, das Aufrufen von Methoden oder die Manipulation von Feldern zur Laufzeit
- Namespace: System.Reflection & System.Type

WOFÜR WIRD REFLECTION GENUTZT

- Reflection bietet dynamische Möglichkeiten zur Interaktion mit Assemblies, Typen und deren Mitgliedern, die zur Kompilierzeit nicht bekannt sein müssen

- Es kann zum Beispiel für die Dynamische Typenerstellung genutzt werden
- Das heißt es können Objekte erstellt und manipuliert werden ohne Kenntnis des Typs zur Kompilierzeit

ASSEMBLY

- Ein Assembly ist eine Sammlung von Typen und Ressourcen, die zusammen eine logische Funktionseinheit bilden
- Assemblys bilden die Grundlage für die Bereitstellung, die Versionskontrolle, die Wiederverwendung, die Festlegung des Aktivierungsumfangs und die Sicherheitsberechtigungen für .NET-basierte Anwendungen
- Assemblys können ausführbare Dateien (.exe) oder Dynamic Link Library Dateien (.dll) sein
- Sie bestehen z.B. aus Metadaten, dem IL-Code und Ressourcen

TYPE

- Type ist eine Klasse, die ein zentrales Element der Reflection in .NET darstellt
- Sie enthält Informationen über den Typ eines Objekts
- Jeder Datentyp (Klasse, Interface, Array ...) hat eine zugehörige Type-Instanz
- Der Type kann über typeof abgefragt werden

```
Type type = typeof(Program);
```

WICHTIGE METHODEN UND EIGENSCHAFTEN

- **GetMethod(string):** Ruft eine MethodInfo Instanz für eine spezifische Methode ab
- **GetMethods():** Gibt ein Array von MethodInfo für alle im Typen enthaltenen Methoden zurück, die auf die BindingFlags Parameter zutreffen
- **GetProperties():** Gibt ein Array von PropertyInfo Instanzen zurück, die alle Eigenschaften des Typs darstellen
- **GetCustomAttributes():** Liefert Attribute zurück, die auf den Typ angewendet werden
- **Name:** Gibt den vollständigen Namen der Klasse aus
- ...

METHODINFO

- MethodInfo beinhaltet Informationen über eine Methode eines Typs
- Sie ermöglicht es, die Methode zu untersuchen und zur Laufzeit aufzurufen (invoken)

- **ReturnType:** Gibt den Typ zurück, den die Methode liefer
- **IsGenericMethod, IsStatic, IsPublic:** Flags, die angeben, ob die Methode generisch, statisch oder öffentlich ist

```
var typeInfo = typeof(MauiApp)
var methods = typeInfo.GetMethods();
var firstMethod = methods[0];
Console.WriteLine(firstMethod.IsGenericMethod);
```

PROPERTYINFO

- PropertyInfo beinhaltet Informationen über eine Eigenschaft eines Typs
- Sie ermöglicht es z.B. den Wert der Eigenschaft abzufragen oder zu schreiben

- **PropertyType:** Gibt den Typ der Eigenschaft an
- **CanRead, CanWrite:** Zeigt an, ob die Eigenschaft gelesen oder geschrieben werden kann

```
var typeInfo = typeof(MauiProgram);
var properties = typeInfo.GetProperties();
var firstProperty = properties[0];
Console.WriteLine(firstProperty.CanRead);
```

GRUNDLEGENDE REFLECTION-OPERATIONEN

- Typinformationen abfragen

```
Type type = typeof(AccountSetupViewModel);
Console.WriteLine("Methoden:");
foreach(var method in type.GetMethods())
{
    Console.WriteLine(method.Name);
}
```

GRUNDLEGENDE REFLECTION-OPERATIONEN

- Reflection zum Auslesen von Attributen

```
Type type = typeof(AccountSetupViewModel);
Attribute[] attrs = Attribute.GetCustomAttributes(type);
foreach(Attribute attr in attrs)
{
    Console.WriteLine(attr.ToString());
}
```

GRUNDLEGENDE REFLECTION-OPERATIONEN

- Zugriff auf private Felder und Methoden von Objekten

```
Type type = typeof(AccountSetupViewModel);
var instance = Activator.CreateInstance(type);
var privateFieldInfo = type.GetField("foo", BindingFlags.NonPublic | BindingFlags.Instance);
var fieldValue = privateFieldInfo.GetValue(instance);
```

GRUNDLEGENDE REFLECTION-OPERATIONEN

- Objekte dynamisch erstellen

```
Type type = typeof(AccountSetupViewModel);
object? instance = Activator.CreateInstance(type);
```