DELEGATES

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

DELEGATE

- Typ der eine Referenz auf eine Methode speichert
- Methoden können als Parameter an eine andere Methode übergeben werden
- Mehrere Methoden können zu einem Delegaten hinzugefügt werden, sodass alle aufgerufen werden, wenn der Delegate selbst aufgerufen wird (Multicast-Delegate)

SYNTAX

- Die Deklaration eines Delegaten erfolgt über das Schlüsselwort delegate
- Signatur und Rückgabetyp sind entscheidend
- Delegaten **können** außerhalb von Klassen definiert werden
- Nützlich wenn der Delegate z.B. von mehreren Klassen genutzt werden soll

```
delegate void CallbackDelegate(string msg);

internal class Program
{
    static CallbackDelegate callbackDelegate = PrintNumber;

    static void Main(string[] args)
    {
        callbackDelegate("Hallo");
    }

    private static void PrintNumber(string msg)
    {
        Console.WriteLine(msg);
    }
}
```

MULTICAST-DELEGATE

- Mittels += Operator werden weitere Delegaten einem Delegaten hinzugefügt
- Mittels -= Operator werden diese entfernt

```
private delegate void CallbackDelegate(string msg);

static void Main(string[] args)
{
    CallbackDelegate callbackDelegate = PrintNumber;
    callbackDelegate += WriteToFile;
    callbackDelegate("Hallo");
}

private static void PrintNumber(string msg)
{
    Console.WriteLine(msg);
}

private static void WriteToFile(string msg)
{
    File.AppendAllText("C:\\Users\\vu\\file.txt", msg);
}
```

SYNTAX

- Methoden können als Parameter an andere Methoden übergeben werden, indem sie einem Delegaten zugewiesen werden
- Die Foo-Methode ist eine Funktion höherer Ordnung (Higher-Order Function)
- Funktionen höherer Ordnung sind Funktionen, die eine andere Funktion als Parameter erhalten oder eine Funktion zurückgeben

```
delegate void MyDelegate(string s);

internal class Program
{
    static void Main(string[] args)
    {
        Foo(Method);
    }

    private static void Foo(MyDelegate dele)
    {
        dele("Hallo");
    }

    public static void Method(string s)
    {
        Console.WriteLine(s);
    }
}
```

ANONYME METHODEN

- Anonyme Methoden sind Methoden ohne expliziten Namen
- Sie ermöglichen die Definition von Methoden direkt dort, wo sie benötigt werden
- Vermeidung der Notwendigkeit, separate Methoden zu definieren

```
delegate int SumDelegate(int x, int y);

internal class Program
{
    static void Main()
    {
        SumDelegate sumDelegate = delegate (int x, int y)
        {
            return x + y;
        };

        Test(sumDelegate);
    }

    public static void Test(SumDelegate dele)
    {
        int result = dele(10, 10);
        Console.WriteLine(result);
    }
}
```

GENERISCHE DELEGATEN

- Generische Delegaten sind vordefinierte Delegattypen, die Parameter- und Rückgabetypen generisch akzeptieren.

```
delegate void MyDelegate<T>(T s);

internal class Program
{
    static void Main(string[] args)
    {
        Foo(Method);
    }

    private static void Foo(MyDelegate<int> dele)
    {
        dele(5);
    }

    public static void Method(int s)
    {
        Console.WriteLine(s);
    }
}
```

ACTION DELEGATE

- Action ist ein vordefinierter generischer Delegattyp in C#
- Es repräsentiert eine Methode, die keine Rückgabe hat (void) und optional Parameter akzeptieren kann
- Action wird häufig für Methoden verwendet, die Aktionen ausführen oder Effekte erzeugen, aber keine Werte zurückgeben

```
internal class Program
{
    static void Main(string[] args)
    {
        Foo(Method);
    }

    private static void Foo(Action<int, string> a)
    {
        a(5, "Hallo");
    }

    public static void Method(int s, string x)
    {
        Console.WriteLine(s);
    }
}
```

FUNC DELEGATE

- Func ist ein weiterer vordefinierter generischer Delegattyp in C#
- Im Gegensatz zu Action gibt Func einen Rückgabewert zurück und kann ebenfalls Parameter akzeptieren
- Der letzte Parameter definiert den Rückgabetypen, und die vorherigen definieren die Parameter
- Func wird häufig verwendet, wenn Methoden einen Wert berechnen oder eine Transformation auf Daten durchführen und einen Rückgabewert haben

```
internal class Program
{
    static void Main(string[] args)
    {
        Test(Sum);
    }

    public static void Test(Func<int, int, int> func)
    {
        int result = func(10, 10);
        Console.WriteLine(result);
    }

    static int Sum(int x, int y)
    {
        return x + y;
    }
}
```

PREDICATE DELEGATE

- Predicate ist ein vordefinierter generischer Delegattyp
- Sie stellt eine Methode dar, die einen booleschen Wert zurückgibt und einen Parameter vom Typ T akzeptiert
- Wird häufig verwendet, um eine Bedingung zu definieren, die auf Elemente in einer Sammlung angewendet wird

```
internal class Program
{
    static void Main(string[] args)
    {
        Test(Sum);
    }

    public static void Test(Predicate<int> func)
    {
        bool result = func(10);
        Console.WriteLine(result);
    }

    static bool Sum(int x)
    {
        return x % 2 == 0;
    }
}
```

DELEGATE METHODEN

- GetInvocationList() Methode
- Gibt ein Array von Delegaten zurück, die in der Aufrufliste des Delegaten enthalten sind
- Jeder Delegat in der Liste entspricht einer Methode, die im ursprünglichen Delegaten enthalten ist
- Syntax: delegatename.GetInvocationList();

```
delegate int CalculateDelegate(int x, int y);

public class Program
{
    public static void Main()
    {
        CalculateDelegate dele = Add;

        foreach(var del in dele.GetInvocationList())
        {
            Console.WriteLine(del.Method.Name);
        }
    }

    private static int Add(int x, int y)
    {
        return x + y;
    }
}
```