POLYMORPHIE

WAS BEDEUTET POLYMORPHIE?

In den letzten Videos haben wir uns mit dem Thema Polymorphie befasst. Dieses Thema ist eigentlich recht einfach und dementsprechend fassen wir das Ganze in diesem Dokument mal kurz zusammen. Fangen wir also an mit der Frage: Was bedeutet Polymorphie eigentlich?

Polymorphie ist ein griechisches Wort und bedeutet „Vielseitigkeit“ (also „verschiedene Formen“). In der Objektorientierten Programmierung sagt man auch: „Es gibt eine Schnittstelle aber dennoch verschiedene Formen.“

Man unterscheidet dabei zwischen 2 verschiedenen Arten der Polymorphie.

- **Die Statische Polymorphie**
- **Die Dynamische Polymorphie**

DIE STATISCHE POLYMORPHIE

Mit der statischen Polymorphie ist das sogenannte „Überladen“ von Methoden gemeint. Mit der Methodenüberladung ist es möglich, verschiedene Ausführungen von einer Methode zu erstellen. Es ist also mögliche für eine einzige Methode verschiedene Signaturen zu erstellen. Mit Signatur ist die Kopfzeile der Methode gemeint, also die Definition der Parameter und deren Datentypen.

```
static void Ausgabe(int zahl)
{
    Console.WriteLine(zahl);
}

static void Ausgabe(string text)
{
    Console.WriteLine(text);
}
```

Abbildung 1. Eine überladene Methode

Auf Abbildung 1 siehst du ein Codebeispiel das eine einfache Methodenüberladung zeigt. Wir haben die Methode „Ausgabe“ zweimal mit demselben Bezeichner. Das Besondere daran ist, dass beide Methoden jeweils eine unterschiedliche Signatur besitzen. Das heißt man kann die Methode auf zwei verschiedene Arten aufrufen. Einmal mit der Übergabe einer Zahl und einmal mit der Übergabe eines Strings.

```
static void Main(string[] args)
{
    Ausgabe("Ich bin ein Text");
    Ausgabe(1000);
}
```

DIE DYNAMISCHE POLYMORPHIE

Als dynamische Polymorphie wird das Überschreiben von virtuellen Methoden bezeichnet. Man verwendet dafür das Schlüsselwort „**virtual**“.

Eine als „**virtual**“ markierte Methode (also eine virtuelle Methode) erlaubt es den erbenden Klassen, sich zu überschreiben. Klingt vielleicht kompliziert ist es aber nicht. Hier mal ein einfaches Beispiel:

```
abstract class Tier
{
    //Virtuelle Methode
    public virtual void GibLaut()
    {
        Console.WriteLine("Das Tier macht ein Geräusch!");
    }
}
```

Abbildung 2. Eine virtuelle Methode

Auf Abbildung 2 siehst du eine abstrakte Klasse namens „Tier“, die als Basisklasse für unterschiedliche Tiere dienen soll. Sie besitzt eine virtuelle Methode namens „GibLaut()“, die das Tier ein Geräusch machen lässt. Nun ist es so, dass jedes Tier sein eigenes spezielles Geräusch macht. Die Katze miaut, der Hund bellt, die Kuh macht muh usw. Aus diesem Grund muss es den erbenden und spezifischeren Tierklassen möglich sein, eine eigene Version der „GibLaut()-Methode“ zu implementieren und genau für diesen Zweck gibt es das Schlüsselwort „**virtual**“.

Mit dem Schlüsselwort „virtual“ ermöglicht man das überschreiben einer bereits implementierten Methode.

Jetzt stellt sich uns nur noch die Frage, wie man eine Methode überhaupt überschreiben kann. Das ist eigentlich ganz einfach! Um eine virtuelle Methode zu überschreiben, verwenden wir das Schlüsselwort „**override**“. Mit „**override**“ können wir eine virtuelle Methode überschreiben. Hier ein Beispiel:

```
class Hund : Tier
{
    //Überschriebene Methode
    public override void GibLaut()
    {
        Console.WriteLine("Der Hund bellt...");
    }
}
```

Abbildung 3. Eine überschriebene Methode

In der Klasse „Hund“ überschreiben wir die zur Klasse „Tier“ gehörende Methode „GibLaut()“ mit dem „override-Schlüsselwort“. Das Funktioniert, weil wir die Methode in der Basisklasse als „virtual“ markiert haben.