# STRUKTUREN UND ENUMS

## EINLEITUNG

In den letzten Videos haben wir uns die Strukturen und Enumeratoren (kurz enums) angeschaut. In diesem Dokument möchte ich dir diese beiden Themen nochmal jeweils zusammenfassen, damit du das Ganze noch wiederholen kannst.

## STRUKTUREN

Fangen wir an mit den sogenannten Strukturen. Mithilfe von Strukturen können wir kompakte Objekte erstellen, welche eine zusammengehörende Gruppe von Variablen zusammenfassen. Strukturen definieren (wie Klassen auch) einen eigenen Datentyp. Hier mal ein Beispiel:

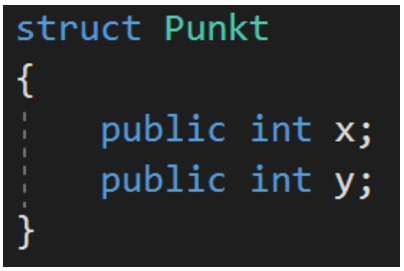


**Abbildung 1. Eine Struktur die einen Punkt darstellt**

Auf Abbildung 1 siehst du eine sehr einfache Struktur, die einen Punkt in einem Koordinatensystem darstellt. Ein solches „Punkt“ Objekt soll lediglich aus 2 Variablen und zwar den Koordinaten des Punktes bestehen. Somit eignen sich Structs an dieser Stelle sehr gut. Doch was ist jetzt eigentlich der Vorteil von Structs und wann sollte man sie anstelle einer normalen Klasse verwenden?

### STRUCTS VS KLASSEN

Zuerst einmal ist es wichtig das man versteht, dass Structs nicht dasselbe wie Klassen sind. Klassen sind nämlich **Referenztypen** und werden auf den sogenannten Heap des Speichers verwaltet, während Structs **Wertetypen** sind und auf dem Stack abgelegt werden. Wir müssen an dieser Stelle nicht zu sehr ins Detail gehen. Es reicht zu wissen, dass Wertetypen weitaus Speicherfreundlicher sind als Referenztypen und dementsprechend stellen Structs eine sehr speicherfreundliche „Alternative“ zur Klasse dar. Wir benutzen Structs dennoch nicht sehr häufig, denn es gibt nicht sehr oft Szenarien in welchen die Nutzung tatsächlich sinnvoll ist.

Benutze Structs nur dann, wenn die Objekte wirklich nur ein paar zusammengehörende Variablen bündeln. Diese Variablen sollten dabei auch wirklich logisch zusammengehören und einen logischen Wert darstellen. Wie zum Beispiel einen Punkt im Koordinatensystem, welcher eine x- und y-Koordinate hat.

### EIGENHEITEN VON STRUCTS

Du hast bereits gelernt, dass Structs nicht dasselbe sind wie Klassen und dementsprechend gestaltet sich deren Nutzung auch ein wenig anders. Schauen wir uns die Eigenheiten von Structs mal kurz an!

Die erste Eigenheit ist die, dass man in der Strukturen-Definition keine Variablen initialisieren kann. Die Initialisierung sollte über den Konstruktor geschehen.

Die zweite wichtige Eigenheit ist die, dass man für Strukturen keinen parameterlosen Konstruktor definieren kann. Man muss dem Konstruktor Parameter übergeben können, ansonsten gibt Visual Studio einen Fehler aus.

## STRUCTS DEFINIEREN UND INSTANTIIEREN

Einen Struct definiert und instantiiert man im Grunde wie eine normale Klasse auch. Um einen Struct zu definieren schreiben wir anstelle von „class“ einfach das „struct“-Schlüsselwort.

```
struct MeinStruct
{

}
```

Abbildung 2. Der Grundaufbau eines Structs

Innerhalb des Codeblocks können wir nun unsere Variablen deklarieren und einen Konstruktor definieren.

Um einen Struct zu instantiieren (um Objekte aus dem Struct zu erstellen) muss man wie bei Klassen auch das „new“-Schlüsselwort verwenden.

```
MeinStruct objekt = new MeinStruct();
```

Abbildung 3. Ein Struct instantiieren

## ENUMERATOREN

Schauen wir uns jetzt mal die sogenannten Enumeratoren (kurz enums) an. Diese sind ein sehr nützliches Werkzeug bei der Programmierung mit C# und jeder Entwickler sollte sie verwenden können. Stellen wir uns also erst einmal die Frage: Was sind Enums eigentlich?

Enums ermöglichen das Definieren von eigenen Datentypen. Sie sind eine Sammlung von benannten ganzzahligen Konstanten (Variablen die immer denselben Wert haben). Die Nutzung von Enums macht unseren Code weitaus lesbarer und sauberer. Ich möchte dir das Ganze mal anhand eines einfachen Beispiels erklären.

Stell dir einmal vor, du hast ein Programm in welchen du eine Variable für den aktuellen Wochentag definieren möchtest. Da man in diese Variable nur einen Wert von 1-7 speichern kann (die Woche hat nämlich nur 7 Tage), eignet sich ein Enum hier sehr gut, denn mit einem Enum können wir jedem dieser Werte eine benannte Konstante verleihen. Wenn wir der eigentlichen Variable später einen Wochentag zuweisen wollen, müssen wir nicht mehr die Zahl, sondern den Name der entsprechenden Enum-Konstante schreiben. Hier mal ein Beispiel:

```
enum Wochentage
{
    Montag = 1,
    Dienstag = 2,
    Mittwoch = 3,
    Donnerstag = 4,
    Freitag = 5,
    Samstag = 6,
    Sonntag = 7
}
```

Abbildung 4. Die Definition eines Enums

Auf Abbildung 4 siehst du wie ein Enum definiert wird. Der Enum namens „Wochentage“ enthält dabei 7 nach den Wochentagen benannte Konstanten, mit den jeweils passenden Werten.

Jetzt wo der Enum definiert wurde kann man auch Objekte davon erstellen:

```
static void Main(string[] args)
{
    Wochentage tag = Wochentage.Montag;

    Console.WriteLine(tag);
}
```

**Abbildung 5. Ein Objekt vom Typ "Wochentage" erstellen**

Wie du siehst können wir den Enum wie einen normalen Datentyp verwenden. Um einen Wert zuzuweisen muss man nur den Bezeichner des Enums schreiben und mit einem Punkt getrennt die gewünschte Konstante auswählen. Das ist weitaus lesbarer als das einfache zuweisen von nichtssagenden Zahlen.