**Statische Klassen & Exkurs „Generische Methoden“**

**Statische Klassen**

Dynamische Typen (public class Abc) können Sie unendlich viele Male neu instanziieren. Statische Objekte (public static class Abc) **instanziieren sich selbst** (impliziert) und das nur **ein einziges Mal**. Dafür sind sie mit all ihren Möglichkeiten eine wahre Schatzkiste.

Mit dem **static**-Modifizierer kann man **eine ganze Klasse** oder auch **nur ihre Mitglieder** (Properties, Fields…) als statisch markieren.

Eine static-class kann nicht von außen instanziiert werden, sie kann nur statische Mitglieder haben (siehe: static void Main einer Konsolenanwendung).

Wann brauchen wir statische Klassen?

* Beispiel: Sammlung von **Hilfsmethoden**
* Globale **Konfigurationen** und **Konstanten**
* **Performance**-Vorteile durch nicht benötigte Instanzen

Hier ein Antipattern für eine nicht-statische Klasse und warum so etwas in dem Fall nicht ideal ist:

*public class* ***Rechner***

*{*

*public int Addiere(int a, int b)*

*{*

*return a + b;*

*}*

*}*

Um die Addiere – Methode aufrufen zu können, brauche ich eine Instanz des Types Rechner (var rechner = new Rechner()), was eigentlich über das Anwendungsziel hinausschießt.

Besser setze ich so etwas um mit „static“:

**Beispiel**

public **static** class MatheHelfer

{

public static int Addiere(int a, int b)

{

return a + b;

}

}

**Aufruf**:

int summe = MatheHelfer.Addiere(3, 5);

Wie oben bereits erwähnt: Eine statische Klasse ist **von außen nicht instanziierbar**. Überzeugen Sie sich davon selbst:

MatheHelfer helfer = new MatheHelfer(); // Fehler

**Eigenschaften statischer Klassen**

* Statische Klassen enthalten nur statische Methoden und Felder.
* Kein public-Instanz-Konstruktor erlaubt (nur statischer Konstruktor)

Beispiel für **„statische Felder“**:

public static class Einstellungen

{

public static string Anwendungsversion = "1.0.0";

}

**Anwendungskonfiguration** (Vorbedingung zum nächsten Schritt)

Falls nicht vorhanden, installieren Sie das NuGet-Paket:

dotnet add package Microsoft.Extensions.Configuration.Json

Erstellen Sie eine Datei **appsettings.json** im Projektverzeichnis und befüllen Sie sie mit gewünschten Konfig-Informationen.

**Statische Konstruktoren**

Obwohl eine statische Klasse von außen nicht instanziiert werden kann, kann sie statische Felder oder andere Initialisierungen enthalten. Dazu wird ein **statischer Konstruktor** aufgerufen, um eben solche Initialisierungen einmalig auszuführen.

public static class DatenbankVerbindung

{

public static string VerbindungString;

static DatenbankVerbindung()

{

VerbindungString = "Server=localhost;Database=test;";

}

}

Der statische Konstruktor wird **nur einmal** aufgerufen, bevor eine statische Methode oder ein statisches Feld der Klasse verwendet wird.

* Er hat keinen Zugriffsmodifikator (also kein public, private usw.).
* Er wird automatisch vom **.NET-Laufzeitsystem** ausgeführt.
* Er kann keine Parameter haben.
* Er wird genau **einmal pro App-Domain** ausgeführt.
* Er wird **lazy** aufgerufen, d. h. erst **beim ersten Zugriff auf die Klasse**.

**Merke:**

Der statische Konstruktor ist der ideale Ort, um **ressourcen-problematische** und sich selten ändernde Werte abzurufen und diese der ganzen Anwendung via public-Properties zur Verfügung zu stellen. Diese Werte bleiben auf die Laufdauer der Anwendung erhalten. Es ist der „Geheimtipp“, wie man schnell und einfach eine Art Zwischenspeicherung (**Caching**) umsetzen kann.

Weitere Caching-Methoden werden Sie zu einem späteren Zeitpunkt noch kennen lernen können.

**Beispiel:**

// Statischer Konstruktor zur Initialisierung

static Einstellungen()

{

IConfiguration config = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)

.Build();

DbConnectionString = config["Datenbank:Verbindungszeichenkette"];

SetWertAusDerDatenbank();

}

**Übung**

Implementieren Sie eine statische Klasse Konverter, die Methoden zur Umrechnung von Celsius nach Fahrenheit und umgekehrt enthält und wenden Sie diese an. Fügen Sie dieser Klasse auch einen statischen Konstruktor hinzu. In diesem lesen Sie in eine public Property den Inhalt einer Textdatei ein. Geben Sie diese Drei (2 Konverter-Methoden und die „String- Property“ in der Console aus.

**Statische Klassen vs. Statische Methoden in nicht-statischen Klassen**

Es besteht auch die Möglichkeit der Unterbringung von statischen Methoden in einer „normalen“ (= nicht statischen) Klasse.

Im .NET Framework gibt es zahlreiche Implementierungen statischer Methoden in nicht statischen Klassen.

**Beispiel**

var cultureInfo = new CultureInfo("de-AT");

var cultureInfo2 = CultureInfo.GetCultureInfo("de-AT");

**Beispiel**:

public class Rechner

{

public static int Multipliziere(int a, int b)

{

return a \* b;

}

}

**Merke**

Statische Felder, Properties und Methoden sind **keine Member** des durch eine nicht statische Klasse modellierten Objektes.

**Wussten Sie?**

Sie können auch in einer nicht-statischen Klasse einen statischen Konstruktor implementieren.

**Erweiterungsmethoden mit statischen Klassen**

Extension Methods

public static class StringErweiterungen

{

public static string Verdoppeln(this string text)

{

return text + text;

}

}

Anwendung:

string text = "Hallo";

Console.WriteLine(text.Verdoppeln());

**Übung**

Erstellen Sie in der obigen Klasse mindestens eine weitere Erweiterungsmethode Ihrer Wahl.

**Exkurs „Generische Methoden“**

Generische Methoden ermöglichen es, **mit verschiedenen Datentypen** arbeiten zu können, ohne dass der Datentyp im Voraus festgelegt werden muss. Stattdessen wird ein Typparameter verwendet, der zur Laufzeit durch einen spezifischen Datentyp ersetzt wird. Dies führt zu **wiederverwendbarem Code** und einer stärkeren Typsicherheit zur Kompilierzeit.

Der Typparameter wird in spitzen Klammern (<T>) nach dem Methodennamen angegeben und kann innerhalb der Methode wie ein regulärer Datentyp verwendet werden.

**Beispiel**

static bool TryParse<T>(object input, out T? result) where T : struct

{

try

{

result = (T)Convert.ChangeType(input, typeof(T));

return true;

}

catch

{

result = null;

return false;

}

}

Anwendung:

if (TryParse<int>("12", out int? resultInt))

Console.WriteLine(resultInt);

if (TryParse<bool>("true", out bool? resultBool))

Console.WriteLine(resultBool);

if (TryParse<decimal>("12,56", out decimal? resultDecimal))

Console.WriteLine(resultDecimal);

if (TryParse<char>('e', out char? resultChar))

Console.WriteLine(resultChar);