# C#

## Wiederholung

Du solltest aus dem 2. Jahrgang folgende Begriffe kennen und erklären können, sowie über deren praktischen Einsatz Bescheid wissen:

* Interpreter – Compiler
* Datentypen
* Arrays
* Verzweigung (if)
* Schleifen
* Klassen
* Objekte
* Datenkomponenten
* Methoden
* Properties
* Sichtbarkeit
* UML
* Vererbung
* Versionsverwaltung (GIT)
* Einfache GUI mit Windows Forms
* Lesen und Schreiben aus einer Textdatei

## GIT-Konzept

Ein Repository mit dem Namen “**2022\_SEW3\_<nachname>”** erstellen.   
Wir arbeiten heuer auf einer lokalen Festplatte: **C:\DEV\<name>**   
**Am Ende jeder Einheit pushen. Dateien auf einer lokalen Festplatte sind niemals sicher!**

## Fehlerbehandlung

Exception Handling (Außnahmbehandlung) ist ein Prinzip der objektorientierten Programmierung um auf Programmfehler zu reagieren.   
Während der Laufzeit können im Programm auftreten. Z.b.:

* Nullreference 🡪 Zugriff auf eine uninitalisierte Vaiable (null)
* Division durch 0
* Cast 🡪 z.B.: int.Parse mit einem Text als Parameter
* Index-out-of-range 🡪 bei Verwendung von Arrays
* Eine Datei wird nicht gefunden
* Ein bestimmter Server ist nicht erreichbar.

Aus diesem Grund sollen Programmteile, wo man mit einem Fehler rechnet (Murphys Law) mit einem try umspannt werden. In einem catch-Block kann der Fehler behandelt werden (nochmals probieren, aussagekräftige Fehlermeldung, …). In einem optionalen finally-Block können noch zusätzliche Operationen ausgeführt werden. Dieser finally-Block wird immer ausgeführt, auch wenn keine Exception auftritt (z.B.: Datei schließen oder von einem Server abmelden, …).

try {

// Hier ist Code, der einen Fehler verursachen kann.

In x = 7 / 0;

}

Catch (Excepion) {

// Hier ist die Fehlerbehandlung

Console.WrieLineEine("Division durch 0 ist einfach keine gute Idee");

}

Finally {

// Optionaler Teil, wird jedoch immer aufgerufen, egal ob Fehler oder nicht.

Console.WriteLine("Ich wünsche dir noch einen schönen Tag.");

}

Es gibt verschiedene Arten von Exceptions, jede dieser "Unterarten" ist von einer Basisklasse Exception abgeleitet.

Mithilfe des Schlüsselwortes throw kann selbst eine Exception geworfen werden. Dazu muss lediglich ein neues Exception Objekt erzeugt werden. Es empfiehlt sich (um der C# Konvention zu folgen) bei Bedarf eigene Exceptions von den Basisklasse Exception abzuleiten (oder eine bereits vordefinierte zu verwenden, wenn passend).

public class JosephinumException : Excpetion { }

throw new JosephinumException();

Jede abgeleitete Exception soll mit dem Wort Exception enden.

Exceptions nicht für normalen Programmablauf verwenden, denn Exceptions sind langsam. Z.B.: lässt sich eine DivisionByZeroExcepion einfach durch eine if-Anweisung, welche den Divisor überprüft vermeiden.

## Static

Statische Elemente (Methoden, Variablen, …) hängen am Datentyp. Nichtstatische Elemente hängen am konkreten Objekt.

Static: Die Variable oder Methode gehör zur Klasse. Es gibt diese Variable oder Methode nur einmal pro Klasse. Z.B.: Main Methode in jedem Programm

* static void Main(strin[] args)
* int.Parse(…)
* Console.WriteLine(…)
* Math.Cos(…)
* …

statische Variable:

* Math.PI

In einer statischen Methode können von der eigenen Klasse nur statische Methoden aufgerufen werden.

Statische Elemente können immer dann verwendet werden, wenn eine Klasse keinen inneren Zustand hat.

Kein static: Die Variable (Datenkomponente) oder Methode gehört zu einem konkreten Objekt. Die entspricht unserer bisherigen Verwendung. In diesem Fall haben die Objekte einen inneren Zustand (z.B.: Objekt mit Vornamen, Zwischenergebnis, …).

## Generische Listen

Bei generischen Listen handelt es sich um eine typsichere Liste. Eine Liste ein verwaltetes Array, das automatisch um das Doppelte vergrößert, sobald es vollläuft. Es ist somit bei der Initialisierung einer Liste keine Angabe über die Länge der Liste notwendig (das erleichtert uns das Leben erheblich 😊).

Die Typsicherheit entsteht durch Angabe des Datentyps in spitzen Klammern.

List<int> meineListe = new List<int>();

Obriges Beispiel erzeugt eine typsichere Liste vom Datentyp int.

Die generische Liste ist im Namensraum System.Collections.Generic definiert.

Die Verwendung einer generischen Liste entspricht im Wesentlichen dem eines Arrays. Die Anzahl der enthaltenen Elemente kann über das Property Count abgefragt werden.