# Nullable Datentypen

Ingo Köster

Diplom-Informatiker (FH)

# Nullable Datentypen

› Wert `null` für Wertetypen verwenden

› Kennzeichnung durch `?` am Datentypennamen
  › `int? zahl = null;`
  › `bool? einWert = null;`
  › etc.

› Mittels Eigenschaft `HasValue` überprüfen ob ein Wert gespeichert wurde

```
if (zahl.HasValue == true) {
   Console.WriteLine(zahl);
}
```

# Nullable Datentypen

› Bei der Deklaration von Nullable Variablen ohne Initialisierung haben die Variablen den Wert `null`

› Als Member von Objekten

```
class MyClass {
  private int? zahl; // hat den Wert null
}
```

› Bei lokalen Variablen ist der Wert nicht definiert

```
void Test() {
  int? zahl; // zahl ist nicht zugewiesen
}
```

# Nullable Datentypen

- Umwandlungen (`float` nach `float?`)
  - `float y = 5f;`
  - `float? z = y;`

- Andersherum nicht implizit möglich, nur explizit
  - `int? zahl = 5;`
  - `byte b = (byte)zahl;`

- Explizite Umwandlung zwischen Nullable Datentypen
  - `double? x = 5;`
  - `int? z = (int?)x;`

# Nullable Datentypen

- Operatoren (+, -, etc.)
- Liefern als Ergebnis `null` falls einer der beiden Operanden `null` ist

- Vergleichsoperatoren (>, >=, etc.) liefern `false` falls einer der beiden Operanden `null` ist
  - == liefert `true` falls beide `null` sind

- Nullable Variablen sind mit `null` vergleichbar

# ?? Operator (null-coalescing)

- Ersetzt den ersten Operanden durch den zweiten Operanden, falls der erste `null` ist

- Beispiel:
  - `double? x = null;`
  - `float? z = 5.0f;`
  - `Console.WriteLine(x ?? z);  // 5.0`

# ?? Operator

› Mittels des ?? Operators sind auch Zuweisungen möglich

```
› long? x = 123;
› int? y = 512;
› long? z = x ?? y;        // Okay, z = 123

› long z = x ?? y;         // Fehler
```

# Null Coalescing Assignment

- Mit C# 8.0 gibt im Bezug auf die Handhabung von `null` einen neuen Operator
- **??=**
- Mit diesem Zuweisungsoperator kann eine Zuweisung ausgeführt werden, wenn eine Variable den Wert `null` hat
- Anstatt

```
if (pers == null) { pers = new Person() { Id = 1, … }; }
```

  - oder

```
pers = pers ?? new Person() { Id = 1, … };
```

- Mittels Null Coalescing Assignment

```
pers ??= new Person() { Id = 1, … };
```

# Null-Bedingungsoperator

› Es soll ein StringBuilder Objekt zum Erzeugen eines Strings verwendet werden

```
StringBuilder sb = null;
```

› Ist der StringBuilder jedoch noch nicht angelegt, führt der folgende Aufruf zu einer Exception

```
string s = sb.ToString();
```

› Es ist notwendig zu prüfen, ob der `StringBuilder` angelegt wurde

```
string s;
if (sb != null) {
   s = sb.ToString();
}
```

# Null-Bedingungsoperator

- Mit dem Null-Bedingungsoperator **?.** kann diese Art der Prüfung vereinfacht werden
  - `string s = sb?.ToString();`

- Stellt der Operator links von ihm `null` fest, werden die Anweisungen rechts nach dem Operator nicht mehr ausgeführt
  - `string s = sb?.ToString().ToUpper();`

# Null-Bedingungsoperator

- Der Null-Bedingungsoperator arbeitet nur mit Typen die `null` zulassen
  - `int length = sb?.ToString().Length; // Nicht möglich!`

- Soll jedoch in einen Nullable Typ zugewiesen werden, kann der Null-Bedingungsoperator verwendet werden
  - `int? length = sb?.ToString().Length;`