C#
Wert- und Referenztypen

BITLC | Tom Selig
12. Dezember 2025

Grundlagen

- In C# wird zwischen sogenannten Wertetypen und Referenztypen unterschieden. Diese unterscheiden sich grundlegend in der Art der Speicherung und damit auch in der Art der Handhabung.
- Fast alle der einfachen (primitiven) Datentypen die C# mitbringt, sind Wertetypen (z.B. `int`, `double`, `bool`, `char`, …), lediglich `string` und `object` sind Referenztypen.
- Zu den Referenztypen zählen unter anderem alle Arrays und auch alle Klassen.
- Zur Speicherung von Wert- und Referenztypen gibt es zwei unterschiedliche Speicherbereiche, den **Stack** und den **Heap**.

Wertetypen

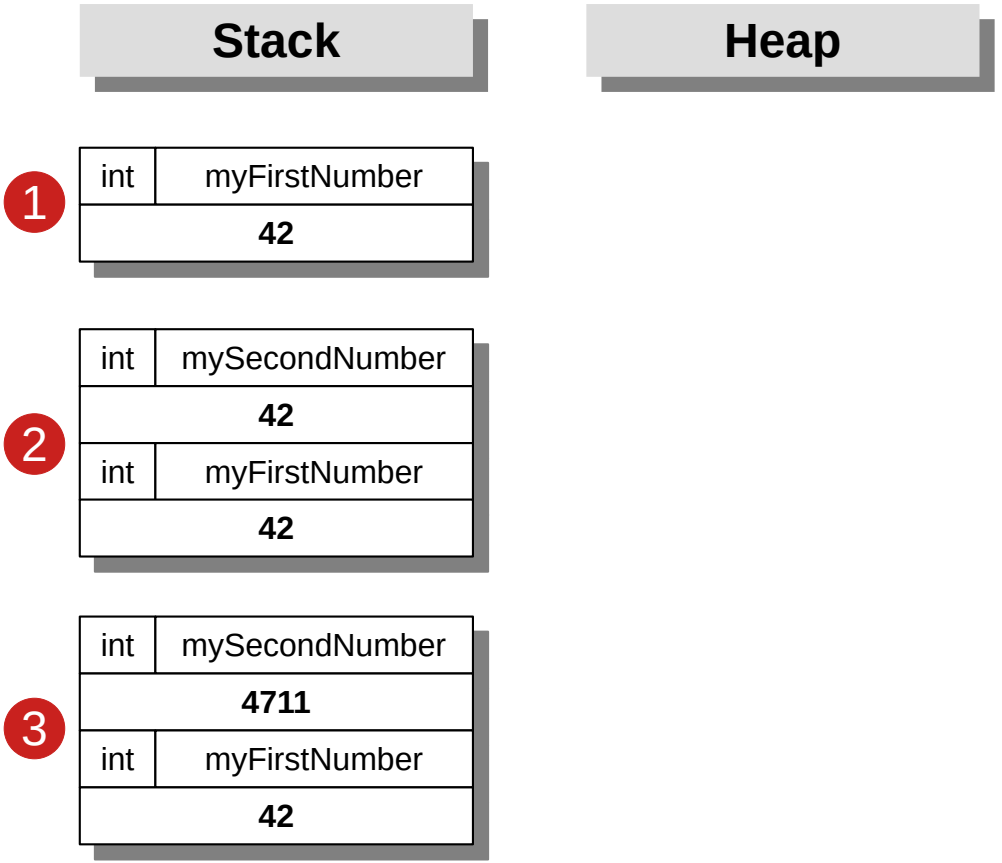
- Wertetypen werden über ein Literal erzeugt, z.B.:

```
int myNumber;
myNumber = 42;
```

- Wenn eine Variable eines Wertetypen erzeugt wird, so wird auf dem **Stack** ein je nach Datentyp entsprechend großer Speicherbereich dafür reserviert (allokiert).
- Im Beispiel oben sind das 32 Bit (4 Byte) für eine Integer-Variable.
- Wird der Variablen dann ein Wert zugewiesen, wird dieser direkt in den allokierten Speicherbereich auf dem **Stack** geschrieben.

Wertetypen

```
static void Main(string[] args) {

    // Variable eines Wertetypen wird erzeugt
1   int myFirstNumber = 42;

    // Inhalt der Variablen wird in eine
    // zweite Variable kopiert
2   int mySecondNumber = myFirstNumber;

    // Beide Variablen haben den gleichen Wert
    Console.WriteLine(myFirstNumber);  // 42
    Console.WriteLine(mySecondNumber); // 42

    // Änderung des Werts von mySecondNumber
3   mySecondNumber = 4711;

    // Variablen haben unterschiedliche Werte
    // Die Änderung hatte keine Auswirkung
    // auf myFirstNumber
    Console.WriteLine(myFirstNumber);  // 42
    Console.WriteLine(mySecondNumber); // 4711
}
```



Referenztypen

- Referenztypen werden in der Regel mit dem `new`-Operator erzeugt:

```
int[] myArray;
myArray = new int[4];
```

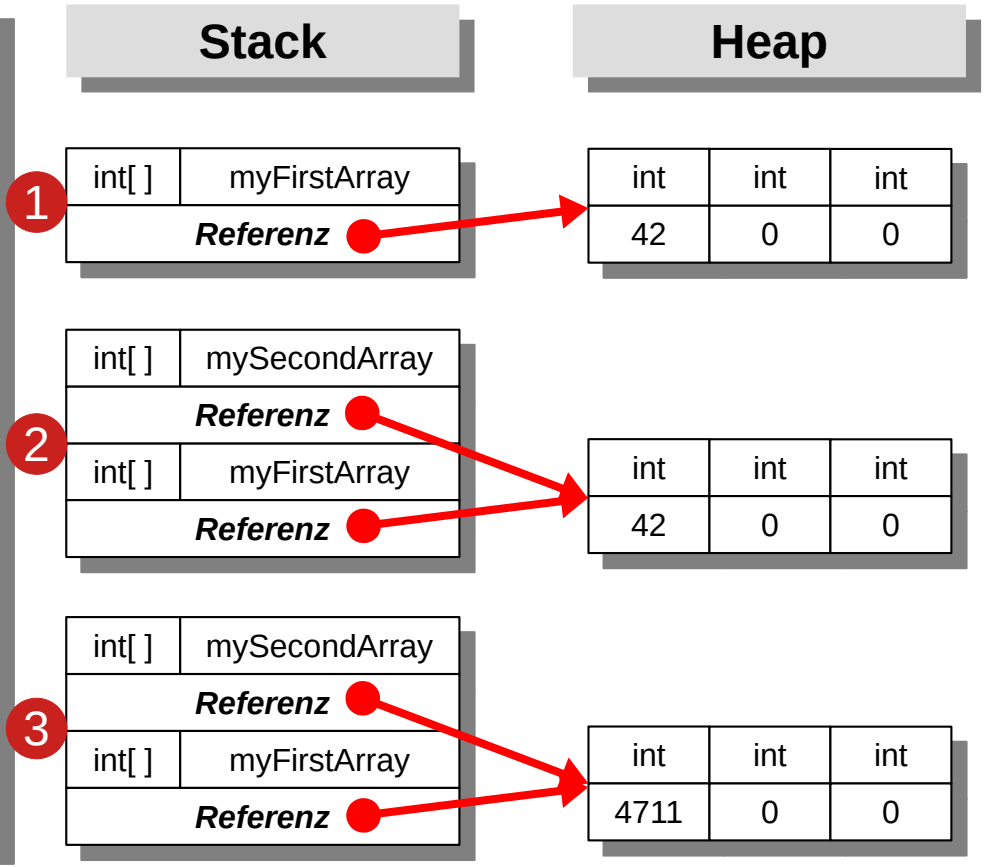
- Wird eine Variable eines Referenztypen deklariert, so wird auf dem **Stack** ein 4 Byte oder 8 Byte großer Speicherbereich allokiert.
- Wird dann mit `new` ein Objekt erzeugt und der Variablen zugewiesen, dann passieren zwei Dinge:

  Das Objekt wird erzeugt und auf dem **Heap** abgelegt.

  Die Speicheradresse, an der das Objekt auf dem **Heap** liegt, wird in der Variablen auf dem **Stack** hinterlegt (eine Referenz).

Referenztypen

```
static void Main(string[] args) {

    // Variable eines Referenztypen wird
    // erzeugt und mit Wert belegt
1   int[] myFirstArray = new int[3];
    myFirstArray[0] = 42;

    // Variable wird in eine zweite kopiert
2   int[] mySecondArray = myFirstArray;

    // Beide haben den gleichen Inhalt
    Console.WriteLine(myFirstArray[0]);  // 42
    Console.WriteLine(mySecondArray[0]); // 42

    // Änderung des Inhalts bei mySecondArray
3   mySecondArray[0] = 4711;

    // Inhalt ist immer noch gleich, Änderung
    // hat auch Auswirkung auf MyFirstArray
    Console.WriteLine(myFirstArray[0]);  // 4711
    Console.WriteLine(mySecondArray[0]); // 4711
}
```



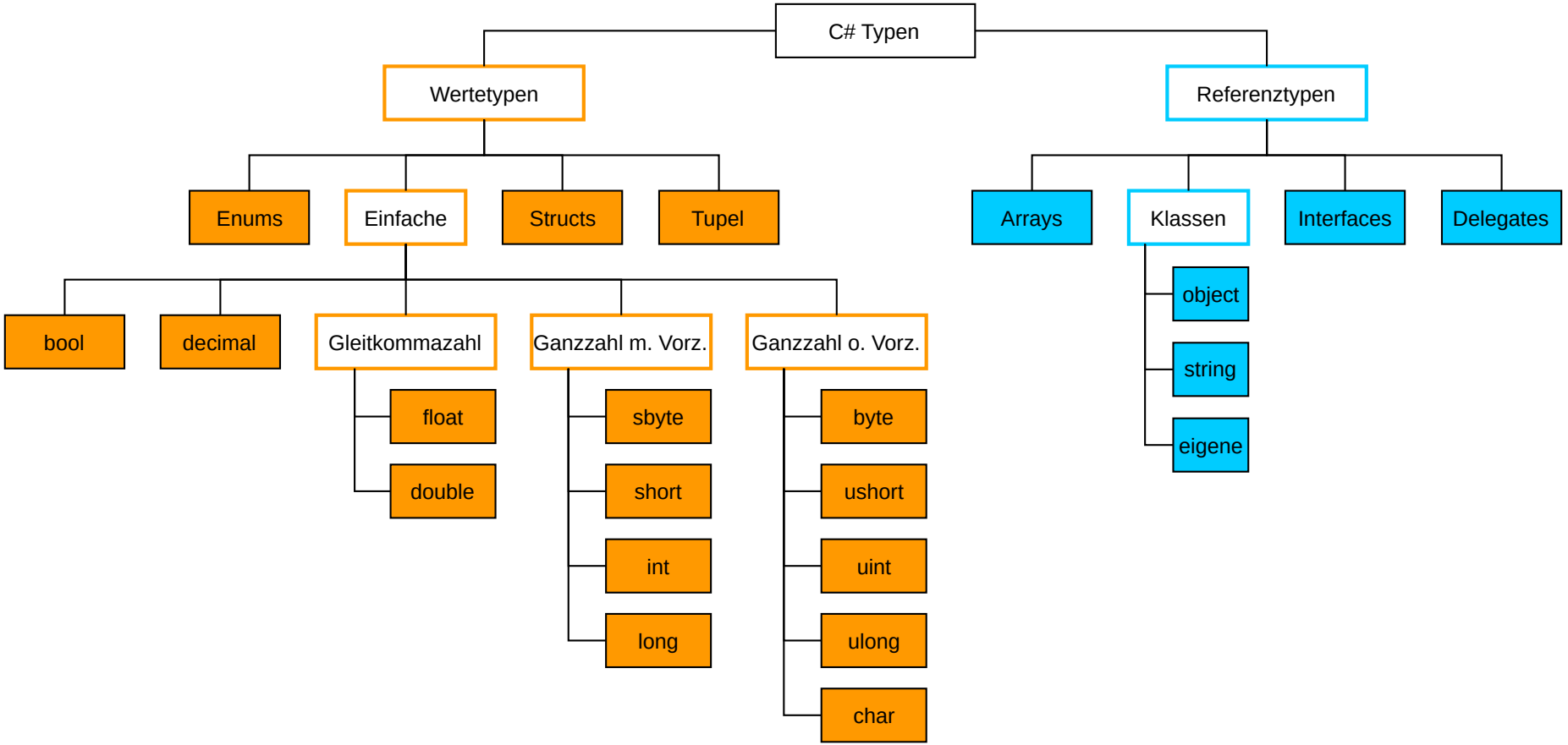
Zusammenfassung

- Der Inhalt eines **Wertetypen** ist der Wert selber, also z.B. 42.
- Der Inhalt eines **Referenztypen** ist eine Referenz auf ein Objekt (also eine Speicheradresse), aber niemals das Objekt selbst.
- **Referenztypen** „leben“ immer auf dem **Heap**.
- **Wertetypen** „leben“ da, wo sie deklariert werden:

  Ein **Wertetyp** als lokale Variable oder Methodenparameter wird auf dem **Stack** erzeugt.

  Ein **Wertetyp**, der Teil eines Referenztyps (z.B. eines Arrays) ist, wird auf dem **Heap** erzeugt.

Typen


C# Typen
Wertetypen
Referenztypen
Enums
Einfache
Structs
Tupel
Arrays
Klassen
Interfaces
Delegates
bool
decimal
Gleitkommazahl
Ganzzahl m. Vorz.
Ganzzahl o. Vorz.
object
string
eigene
float
double
sbyte
short
int
long
byte
ushort
uint
ulong
char

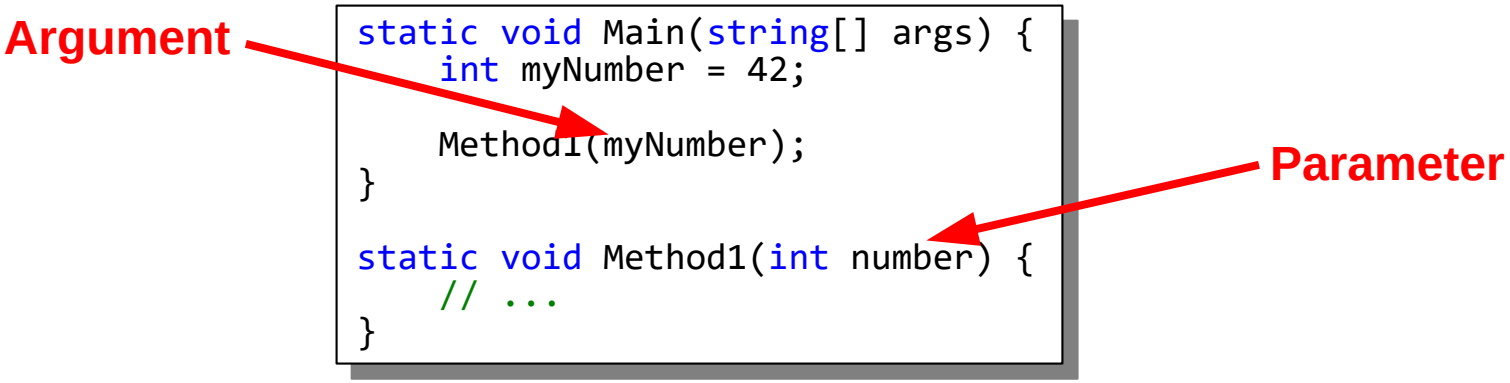
`null`

- Referenztypen können den Wert `null` annehmen.
- `null` bedeutet, dass der Referenztyp auf kein Objekt zeigt.
- Wenn man versucht, auf ein Member einer `null`-Referenz zuzugreifen, erhält man einen Laufzeitfehler:

```
int[] myArray = null;
Console.WriteLine(myArray[0]);  // NullReferenceException
```

- `null` ist der Standardwert für Referenztypen.
- Wertetypen kann auf normalem Weg nicht der Wert `null` zugewiesen werden.

Argument vs. Parameter



- Ein Argument ist der Wert, der an eine Methode übergeben wird.
- Ein Parameter ist der Wert, den die Methode von außen erwartet.
- Beim Aufruf einer Methode wird also das übergebene Argument in den Parameter der Methode kopiert.

call by value

- Standardmäßig werden in C# Argumente an Methoden als Wert übergeben (**call by value**).
- Das ist auch bei Referenztypen der Fall.
- Es wird dabei eine Kopie des Arguments erstellt, die dann an die Methode übergeben wird.
- Änderungen am Parameter innerhalb der Methode haben keine Auswirkung auf das Argument außerhalb der Methode.
- Bei Referenztypen haben lediglich Änderungen am Inhalt des referenzierten Objekts Auswirkungen außerhalb der Methode.

call by value

```
static void Main(string[] args) {

    // Wertetyp wird deklariert
    // und mit 42 initialisiert
    int myNumber;
    myNumber = 42;

    // Übergabe an Methode
    // (call by value)
    Method1(myNumber);

    // Ausgabe des Wertes
    // Immer noch 42
    Console.WriteLine(myNumber);
}

static void Method1(int number) {

    // Änderung des übergebenen
    // Parameters
    number = 4711;
}
```

```
static void Main(string[] args) {

    // Referenztyp wird deklariert
    // und mit 42 initialisiert
    int[] myArray = new int[1];
    myArray[0] = 42;

    // Übergabe an Methode
    // (call by value)
    Method1(myArray);

    // Ausgabe des Wertes
    // Immer noch 42
    Console.WriteLine(myArray[0]);
}

static void Method1(int[] array) {

    // Änderung des übergebenen
    // Parameters
    array = null;
}
```

call by reference

- Um ein Argument an eine Methode als Referenz zu übergeben (**call by reference**), muss der Modifikator `ref` benutzt werden.
- Der Modifikator muss sowohl in der Methodensignatur als auch beim Aufruf der Methode geschrieben werden.
- Dabei wird beim Aufruf der Methode nicht der Wert des Arguments in den Parameter kopiert, sondern eine Referenz auf das Argument übergeben.
- Änderungen am Parameter in der Methode haben somit auch Auswirkungen auf das Argument außerhalb der Methode, da beide auf den selben Speicherbereich „zeigen“.

call by reference

```
static void Main(string[] args) {

    // Wertetyp wird deklariert
    // und mit 42 initialisiert
    int myNumber;
    myNumber = 42;

    // Übergabe an Methode
    // (call by reference)
    Method1(ref myNumber);

    // Ausgabe des Wertes
    // Ist jetzt 4711
    Console.WriteLine(myNumber);
}

static void Method1(ref int number) {

    // Änderung des übergebenen
    // Parameters
    number = 4711;
}
```

```
static void Main(string[] args) {

    // Referenztyp wird deklariert
    // und mit 42 initialisiert
    int[] myArray = new int[1];
    myArray[0] = 42;

    // Übergabe an Methode
    // (call by reference)
    Method1(ref myArray);

    // Ausgabe des Wertes
    // ! NullReferenceException !
    Console.WriteLine(myArray[0]);
}

static void Method1(ref int[] array) {

    // Änderung des übergebenen
    // Parameters
    array = null;
}
```