

# HoGent

BEDRIJF  
EN  
ORGANISATIE

## Hoofdstuk 2 : C#

# Hoofdstuk 2 : C#

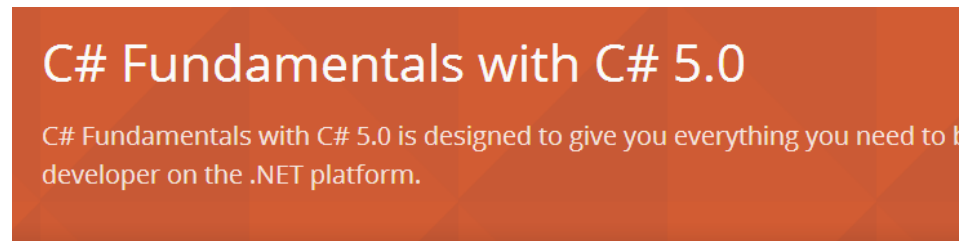
---

Huidige versie C# 8

1. Symbolen
2. Datatypes
3. Operatoren
4. Controle structuren
5. Array
6. Exceptions
7. What's new in C# 8

- ▶ Kennismaking met de C# taal
  - Bekijk de slides
  - Of overloop een tutorial
    - <https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/intro-to-csharp/index> (interactieve tutorials)
    - [http://msdn.microsoft.com/en-us/library/ms228602\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/ms228602(v=VS.90).aspx) (C# for Java developers)
    - PluralSight (zie volgende slide)

- ▶ Kennismaking met de C# taal
  - Of op Pluralsight :



by Scott Allen

## Table of Contents [Expand All](#)

▶ Introduction to C#	⌵	⌵
▶ Classes and Objects in C#	⌵	⌵
▶ Types and Assemblies	⌵	⌵
▶ Members: Methods, Events, and Properties	⌵	⌵
▶ Flow Control	⌵	⌵
▶ Object Oriented Programming	⌵	⌵

hoofdstuk 1

hoofdstuk 3

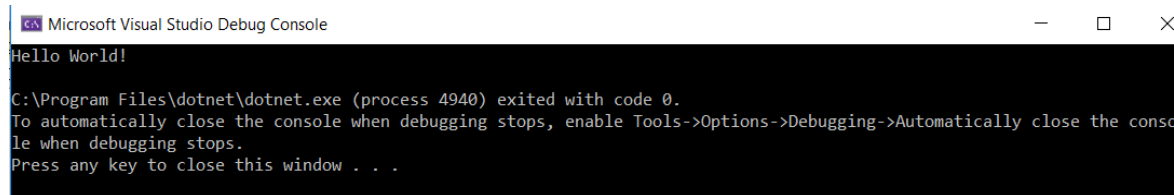
- ▶ Wens je de code in de slides uit te proberen
  - In VS : New Project > Language =C#, project type=Console. Kies Console App(.Net Core). Klik Next. Geef een naam in, bvb Demo en een locatie in
  - Open Program.cs, dit bevat de klasse **Program** met static methode **Main** methode. Hier plaats je de code in
  - Schrijven naar de Console kan via **Console.WriteLine**
  - Een voorbeeldje

```
using System;

namespace Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

- ▶ Wens je de code in de slides uit te proberen

- Run de code, F5



```
Microsoft Visual Studio Debug Console
Hello World!
C:\Program Files\dotnet\dotnet.exe (process 4940) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

- Press any key om de console af te sluiten

- Meer op :

<https://www.microsoft.com/net/tutorials/csharp/getting-started/hello-world>

# Symbolen

# 1. Symbolen

---

- ▶ C# is opgebouwd uit namen, keywords, cijfers, karakters, strings, operatoren, commentaar, ...
- ▶ Namen:
  - mogen letters, cijfers en \_ bevatten
  - beginnen steeds met een letter of \_
  - C# is case-sensitive
- ▶ Keywords: <http://msdn.microsoft.com/en-us/library/x53a06bb.aspx>



# 1. Symbolen

---

- ▶ Commentaar:

- single line: //
- delimited: /\* .....  
.....  
..... \*/

# Data types

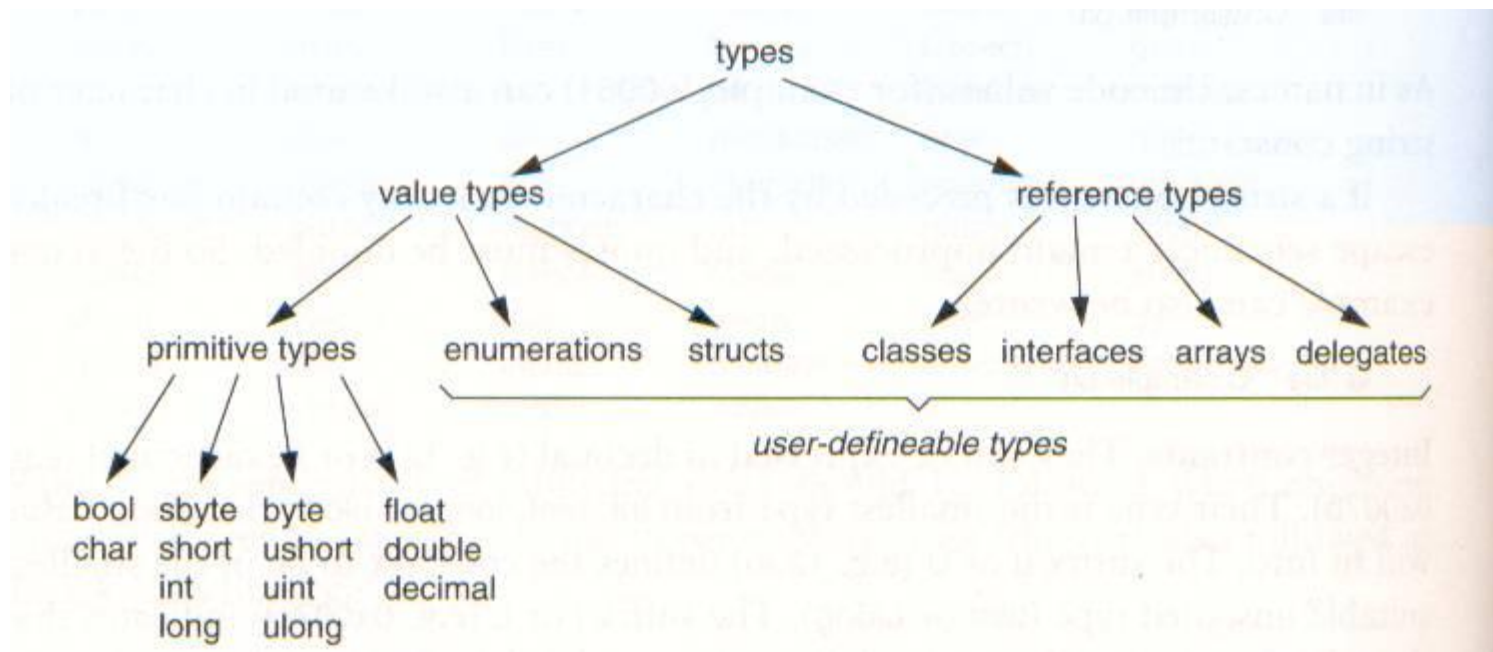
## 2. Data types

---

- ▶ C# is strongly typed
  - Elke variabele en object krijgt een type bij declaratie
- ▶ Een data type is:
  - een ingebouwd data type, zoals int of char
  - of een user-defined data type, zoals een **class** of een **interface** of **delegate**.
- ▶ Data types zijn :
  - Value Types die de actuele waarde bevatten, of
  - Reference Types, die een reference bevatten naar de actuele data.

## 2. Data types

- ▶ **Common Type System (CTS) :**
  - Bevat de built-in data-types met bijhorende methodes die binnen alle .NET programmeertalen gebruikt kunnen worden.



# 2. Data types

## ► CTS : de (belangrijkste) built-in data types

Category	System Type	Description	C# data type (shorthand notation)
Integer	Byte	An 8-bit unsigned integer. (-128 ... 127)	<b>byte</b>
	Int16	A 16-bit signed integer. (-32.768 .. 32.767)	<b>short</b>
	Int32	A 32-bit signed integer. (-2.147.483.648.. 2.147.483.647)	<b>int</b>
	Int64	A 64-bit signed integer. ( $-2^{63}..2^{63}-1$ )	<b>long</b>
Floating point	Single	A single-precision: 32-bit floating-point number. ( $\pm 1.4\text{E}-45..\pm 3.4\text{E}38$ ) (7 cijfers)	<b>float</b>
	Double	A double-precision: 64-bit floating-point number. ( $\pm 5\text{E}-324..\pm 1.7\text{E}308$ ) (15 cijfers)	<b>double</b>
Logical	Boolean	A Boolean value (true or false).	<b>bool</b>
Other	Char	A Unicode (16-bit) character.	<b>char</b>
	Decimal	A decimal value. 128 bit signed number (29 cijfers – 10delig talstelsel)	<b>decimal</b>
Class objects	Object	The root of the object hierarchy.	<b>object</b>
	String	An immutable, fixed-length string of Unicode characters. Limited by system memory	<b>string</b>

## 2. Data types

### ► Common Type System

- Value-type : bevatten de actuele waarde
  - Simple types : voorgedefinieerde value types (zijn structs)
  - structs
  - enum
- Reference-type : bevat een verwijzing naar een object

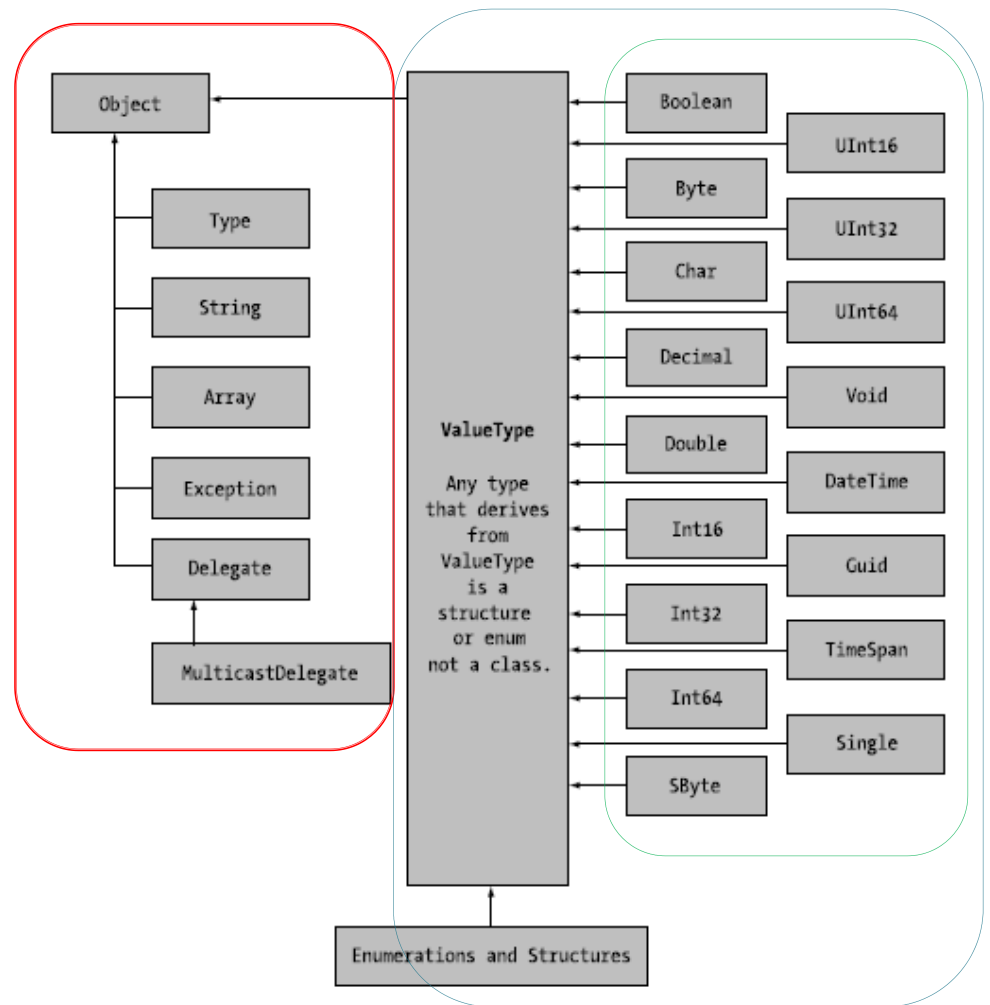


Figure 3-6. The class hierarchy of system types

## 2. Data types : Value types

### ► Value types

- Value types bevatten de *actuele* waarde. Toekennen van een value type aan een andere *kopieert* de waarde.
- Erven van [System.ValueType](#), die op zijn beurt erft van Object
- Kunnen geen null waarde bevatten.
  - Wordt opgelost door Nullable types (zie verder)

### ValueType Class

.NET Framework 4 | Other Versions ▾ 2 out of 2 rated this helpful [Rate this topic](#)

Updated: August 2011

Provides the base class for value types.

#### ▲ Inheritance Hierarchy

```
System.Object
  System.ValueType
    System.Enum
```

Namespace: [System](#)  
Assembly: mscorlib (in mscorlib.dll)

#### ▲ Syntax

C# C++ F# VB

```
[SerializableAttribute]  
[ComVisibleAttribute(true)]  
public abstract class ValueType
```

## 2. Data types : Value types

### ► Simple types

- int, bool,...
- Declaratie

```
datatype naam = initiele waarde;
```

- bepaalt naam variabele, datatype (benodigde opslagruimte) en scope
- Declareer en initialiseer een variabele vóór gebruik

```
int i, d;  
int k = 5;  
d=0;  
d+=i + k; //genereert een compiler fout "use of unassigned local variable i"
```

- int,... zijn aliases voor C# data types System.Int32,... (zie structs)

```
System.Int32 i = 0;  
Is hetzelfde als int i=0;
```



## 2. Data types : Value types

### ► Simple types

- Voor instantiatie maak je gebruik van constructor of literals

```
int i = new int();  
int j = 0;
```

- De default waarden zijn (en meteen ook de schrijfwijze voor literals)
  - bool : false of true
  - char : '\0' (1 lege karakter)
  - int : 0
  - decimal : 0.0M
  - double : 0.0 of 0.0D
  - float : 0.0F
  - long : 0L
  - DateTime : 1/1/0001 12:00:00 AM

## 2. Data types : Value types

### ▶ struct type

- *Voor de creatie van een value type bestaande uit 1 waarde.*
- Is vergelijkbaar met een klasse, maar is een *VALUE* type (geen reference type). Een struct type kan fields, methods en constructors hebben, *maar wordt gemanaged op de stack en dus doorgegeven by value en representeert 1 waarde*
- Concreet wordt binnen .Net veel gebruik gemaakt van structs: int, long en float bvb zijn eigenlijk niks anders dan verkorte schrijfwijzen voor de structs System.Int32, System.Int64 en System.Single. Deze structs hebben fields, constructors (int i = new int();) en methods (zoals ToString(), vb 2.ToString()).
- Een struct wordt voornamelijk gebruikt voor entiteiten die belangrijk zijn om hun waarde, zoals een getal, een datum, ...
- Meer info : <http://msdn.microsoft.com/en-us/library/ah19swz4.aspx>
- We gaan zelf geen structs bouwen maar gaan er wel veel gebruiken

## 2. Data types : Value types

### ► enum types

- Is ook een *value* type
- Lijst van constante namen en de overeenkomstige numerische waarde (default van type `System.Int32`). Numerisch waarde wordt op de stack gezet.

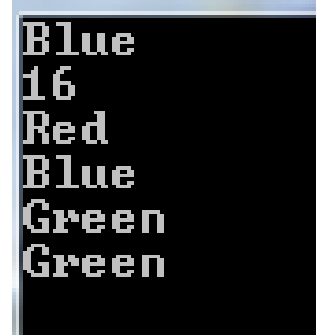
//definitie enum

```
public enum ColorType
{
    Red = 0,
    Blue = 16,
    Green = 256
};
```

## 2. Data types : Value types

### ► enum types

```
static void Main(string[] args) {  
    ColorType c = ColorType.Blue; //declaratie + instantiatie variabele van dit type  
    Console.WriteLine(c); // de constante naam nl. Blue  
    Console.WriteLine((int)c); // bijhorende waarde nl. 16  
    //overlopen van de enumeratiewaarden  
    foreach (string s in Enum.GetNames(typeof(ColorType)))  
        Console.WriteLine(s);  
    // Parsen : omzetten van een string of numerische waarde naar een enum datatype  
    c = (ColorType)Enum.Parse(typeof(ColorType), "Green");  
    // Enum.Parse retourneert een object. Vergeet cast niet!!  
    Console.WriteLine(c); // Green  
    Console.ReadLine();  
}
```



```
Blue  
16  
Red  
Blue  
Green  
Green
```

## 2. Data types : Reference types

---

### ▶ Reference types

- Variabelen van het reference type, bevatten een verwijzing naar de actuele data.
- Built in reference types
  - Object
  - String
- Zelf reference types definiëren, via de keywords
  - class
  - interface
  - delegate

## 2. Data types : Reference types

### ► System.Object

- de root van type hiërarchie
- Alle types zijn hiermee compatibel, zowel value als reference
- Methodes

Name	Description
<a href="#">Equals(Object)</a>	Determines whether the specified <b>Object</b> is equal to the current <b>Object</b> .
<a href="#">Equals(Object, Object)</a>	Determines whether the specified object instances are considered equal.
<a href="#">Finalize</a>	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
<a href="#">GetHashCode</a>	Serves as a hash function for a particular type.
<a href="#">GetType</a>	Gets the <a href="#">Type</a> of the current instance.
<a href="#">MemberwiseClone</a>	Creates a shallow copy of the current <b>Object</b> .
<a href="#">ReferenceEquals</a>	Determines whether the specified <b>Object</b> instances are the same instance.
<a href="#">ToString</a>	Returns a string that represents the current object.

## 2. Data types : Reference types

### ► String

- Is in feite een character-array (maar niet volledig als array te benaderen)
- Immutable type : éénmaal een waarde toegekend kan je waarde niet meer wijzigen.
- Vergelijken van string-waarde
  - == ( vergelijkt de inhoud! En dit in tegenstelling tot == bij objecten waar gekeken wordt of beiden naar zelfde object wijzen.)
  - Equals

```
string s1 = "abc";  
string s2 = "abc";  
  
Console.WriteLine ( s1 == s2) ; // resultaat is true  
Console.WriteLine (s1.Equals(s2)); //resultaat is true
```

## 2. Data types : Reference types

- Methodes

String Member	Meaning in Life
Length	This property returns the length of the current string.
Compare()	This method compares two strings.
Contains()	This method determines whether a string contains a specific substring.
Equals()	This method tests whether two string objects contain identical character data.
Format()	This method formats a string using other primitives (e.g., numerical data, other strings) and the {0} notation examined earlier in this chapter.
Insert()	This method inserts a string within a given string.
PadLeft() PadRight()	These methods are used to pad a string with some characters.
Remove() Replace()	Use these methods to receive a copy of a string, with modifications (characters removed or replaced).
Split()	This method returns a String array containing the substrings in this instance that are delimited by elements of a specified Char or String array.
Trim()	This method removes all occurrences of a set of specified characters from the beginning and end of the current string.
ToUpper() ToLower()	These methods create a copy of the current string in uppercase or lowercase format, respectively.



## 2. Data types : Reference types

- Enkele voorbeeldjes

- Trimmen

```
string input = " Steve "; // has a space at the start and end.  
string clean1 = input.TrimStart(); // "Steve "  
string clean2 = input.TrimEnd(); // " Steve"  
string clean3 = input.Trim(); // "Steve"  
string shortversion = input.Trim().Substring(0,3); // "Ste"
```

- Formatteren

```
string original = "Test string";  
string capital = original.ToUpper(); // TEST STRING  
string lower = original.ToLower(); // test  
string string lower2 = "Another Test".ToLower(); // another test
```

## 2. Data types : Reference types

- Opmerking : Immutable
  - Alle value types en het type string zijn immutable
    - Onderstaand voorbeeld geeft niet het juiste resultaat

```
string name = "    Hallo world";  
name.Trim(); => daar immutable, geeft string een nieuwe string terug  
Console.WriteLine(name); => "    Hallo world"
```

- Oplossing

```
string name = "    Hallo world";  
name = name.Trim();  => plaats de nieuwe waarde in een variabele  
Console.WriteLine(name); => "Hallo world"
```

- Een ander voorbeeld van een valueType

```
DateTime creationDate = new DateTime(2000, 1, 1);  
creationDate = creationDate.AddYears(1);  
Console.WriteLine(creationDate);
```

## 2. Data types : Reference types

- String concatenatie
  - Je kan gebruik maken van + operator

```
string name = "Steve";  
string greet1 = "Hello " + name + "!"; // Hello Steve!
```

- Of **String interpolatie**.
  - String literal voorafgegaan door een \$
  - De variabele plaats je tussen {}, na : gebruik je string formatter ([https://msdn.microsoft.com/en-us/library/dwhawy9k\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dwhawy9k(v=vs.110).aspx))

```
string name = "Steve"; decimal totalPrice=10M;  
string greet = $"Hello {name}, the total price is {totalPrice:C2}!"; //Hello Steve,...
```

- Of **String.Format**(*te formatteren string, variabele0, variabele1,...*)
  - Verwijzen naar variabele in de te formatteren string {volgnummer}

```
string name = "Steve";  
string greet3 = String.Format("Hello {0}!", name); // Hello Steve!
```

## 2. Data types : Reference types

- Escape literals

**Table 3-6.** *String Literal Escape Characters*

Character	Meaning in Life
\'	Inserts a single quote into a string literal.
\"	Inserts a double quote into a string literal.
\\	Inserts a backslash into a string literal. This can be quite helpful when defining file paths.
\a	Triggers a system alert (beep). For console programs, this can be an audio clue to the user.
\n	Inserts a new line (on Win32 platforms).
\r	Inserts a carriage return.
\t	Inserts a horizontal tab into the string literal.

- Maak je in een string gebruik van een escape character dan \ verdubbelen ofwel laat je string voorafgaan door @-sign.
  - @"c:\Docs\Source\a.txt" of "c:\\Docs\\Source\\a.txt"

## 2. Data types : Nullable types

### ▶ Nullable Types

- Is een value type dat een waarde van het gedefinieerde datatype kan bevatten + de 'waarde' null.
- Declaratie (gebruikt het ?)
  - `int? x = 10; int? y = null;`
- Toekenning
  - `x = null; y = 10;`
- Properties
  - `HasValue` : bool, true als de variabele geen null waarde bevat
  - `Value` : van hetzelfde type als het onderliggende value type. Als `HasValue` is true bevat dit de waarde. Indien `HasValue` false wordt een `InvalidOperationException` gethrowed.
- Casting naar datatype
  - `int? x = null; int y = 10;`
  - `if (x.HasValue) y = x.Value;`

## 2. Data types : Constanten

### ► Constanten

```
const long size = ((long)int.MaxValue + 1)/4;
```

- Moeten geïnitieerd worden bij declaratie
- Kunnen niet meer van waarde wijzigen
- Zijn impliciet altijd static (static modifier is niet toegelaten in de declaratie)

## 2. Data types : Conversies

### ► Casting

- Impliciete casting
  - Enkel indien de waarde hierdoor niet wordt aangepast
  - Volgorde : byte, short, int, long, float, double, decimal
- Expliciete casting
  - Checked : controleert of cast safe is, anders Overflow Exception

```
long val = 30000;  
int i = (int)val;  
int j = checked ((int)val);
```

- Van string naar numerieke waarde en omgekeerd

```
string s = "100";  
int i = int.Parse(s);
```

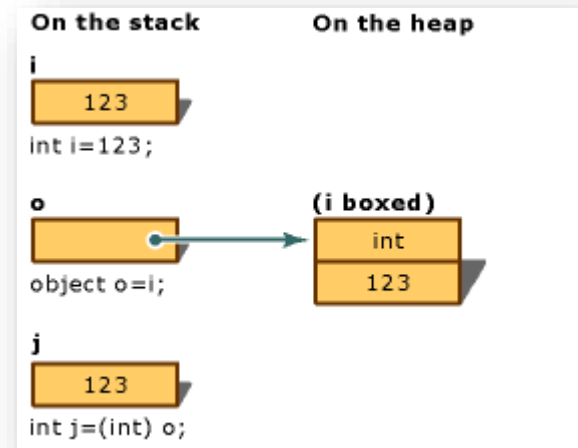
```
int i = 10;  
string s = i.ToString();
```

- Verschil tussen Convert en Parse : Convert.ToInt32 kan overweg met null waarden, int.Parse throwt ArgumentNullException

## 2. Data types : Conversies

- ▶ Boxing en unboxing
  - Boxing = opslag van value types in de garbage-collected heap. Boxing is alloceren van een object instance op de heap en het kopiëren van de waarde in het nieuwe object
  - Unboxing is het omgekeerde

```
int i = 123;  
Object o = i; //implicit boxing  
Object o = (Object)i; //Explicit boxing  
int j = (int)o; //unboxing
```





## 2. Data types : Date en Time

- ▶ System.DateTime en System.TimeSpan (= struct)
  - DateTime : maand, dag, jaar en tijd (initieel 01/01/0001 12:00 AM)
  - TimeSpan : uren, min, sec

```
// De constructor met params (year, month, day). De datum wordt  
ingesteld op 12:00AM
```

```
DateTime dt = new DateTime(2011, 10, 17);
```

```
//Vandaag
```

```
DateTime dt = DateTime.Today of DateTime.Now (= met tijdstip)
```

```
// Welke dag in de week?
```

```
Console.WriteLine("The day of {0} is {1}", dt.Date, dt.DayOfWeek);
```

```
//De constructor met params (uren, minuten, seconden)
```

```
TimeSpan ts = new TimeSpan(4, 30, 0);
```

```
//+ operator niet gedefinieerd tussen 2 DateTimes. Wel mogelijk :
```

```
DateTime dt = DateTime.Now + new TimeSpan(5,0,0);
```

```
dt.AddDays(5) of dt.AddYears(5) of dt.AddMonths(5)
```

## 2. Data types : Date en Time

### ► System.DateTime

Properties	Description
Date	Gets the <b>date</b> component of this instance.
DayOfWeek	Gets the <b>day of the week</b> represented by this instance.
Now	Gets a DateTime object that is set to the current date and time on this computer, expressed as the local time.
Today	Gets the current date.

Methods	Description
AddDays	Returns a new DateTime that adds the specified number of days to the value of this instance.
Parse(String)	Converts the specified string representation of a date and time to its DateTime equivalent.
ToString()	Converts the value of the current DateTime object to its equivalent string representation. (Overrides <a href="#">ValueType.ToString()</a> .)
ToShortDateString()	Converts the value of the current DateTime object to its equivalent short date string representation (dd/mm/yyyy)

## 2. Data types : Date en Time

### ► Adding/Subtracting from dates and times

Method	Description
Add	Adds/Subtracts the value of the specified TimeSpan object instance.
AddDays	Adds/Subtracts the specified number of days
AddHours	Adds/Subtracts the specified number of hours
AddMilliseconds	Adds/Subtracts the specified number of Milliseconds
AddMinutes	Adds/Subtracts the specified number of minutes
AddMonths	Adds/Subtracts the specified number of months
AddSeconds	Adds/Subtracts the specified number of seconds
AddYears	Adds/Subtracts the specified number of years

## 2. Data types : Date en Time

- ▶ Deel van een DateTime opvragen

```
DateTime meetingAppt = new DateTime(2008, 9, 22, 14, 30, 0);  
  
System.Console.WriteLine (meetingAppt.Day);  
System.Console.WriteLine (meetingAppt.Month);  
System.Console.WriteLine (meetingAppt.Year);  
System.Console.WriteLine (meetingAppt.Hour);  
System.Console.WriteLine (meetingAppt.Minute);  
System.Console.WriteLine (meetingAppt.Second);  
System.Console.WriteLine (meetingAppt.Millisecond);
```

## 2. Data types : Date en Time

### ► Formatteren van een DateTime

```
DateTime meetingAppt = new DateTime(2008, 9, 22, 14, 30, 0);

System.Console.WriteLine(meetingAppt.ToLongDateString()); // Monday, September 22, 2008
System.Console.WriteLine(meetingAppt.ToShortDateString()); // 9/22/2008
System.Console.WriteLine(meetingAppt.ToShortTimeString()); // 2:30 PM

DateTime meetingAppt = new DateTime(2008, 9, 22, 14, 30, 0);

System.Console.WriteLine(meetingAppt.ToString("MM/dd/yy")); // 09/22/08

System.Console.WriteLine(meetingAppt.ToString("MM - dd - yyyy")); // 09 - 22 - 2008

System.Console.WriteLine(meetingAppt.ToString("ddd dd MMM yyyy")); // Mon 22 Sep 2008

System.Console.WriteLine(meetingAppt.ToString("dddd dd MMMM yyyy")); // Monday September 22 2008
```

- Meer op <https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings>

## 2. Data types : Time

### ► System.TimeSpan

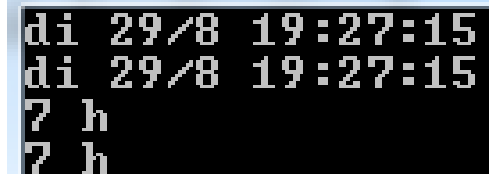
Methods	Description
<a href="#">Add</a>	Returns a new TimeSpan object whose value is the sum of the specified TimeSpan object and this instance.
<a href="#">FromHours</a>	Returns a TimeSpan that represents a specified number of hours, accurate to the nearest millisecond.
<a href="#">FromMinutes</a>	Returns a TimeSpan that represents a specified number of minutes, accurate to the nearest millisecond.
<a href="#">FromSeconds</a>	Returns a TimeSpan that represents a specified number of seconds, accurate to the nearest millisecond.
<a href="#">Parse(String)</a>	Converts the string representation of a time interval to its TimeSpan equivalent.
<a href="#">Subtract</a>	Returns a new TimeSpan whose value is the difference between the specified TimeSpan object and this instance.
<a href="#">ToString()</a>	Converts the value of the current TimeSpan object to its equivalent string representation

Properties	Description
<a href="#">Days</a>	Gets the days component of the time interval represented by the current TimeSpan structure.
<a href="#">Hours</a>	Gets the hours component of the time interval represented by the current TimeSpan structure.
<a href="#">Minutes</a>	Gets the minutes component of the time interval represented by the current TimeSpan structure.
<a href="#">Seconds</a>	Gets the seconds component of the time interval represented by the current TimeSpan structure.
<a href="#">TotalHours</a>	Gets the value of the current TimeSpan structure expressed in whole and fractional hours.
<a href="#">TotalMinutes</a>	Gets the value of the current TimeSpan structure expressed in whole and fractional minutes.

## 2. Data Types : Time

- Format van System.DateTime en System.TimeSpan
  - `date.ToString(string)` methode => *string* is de format specifier (zie volgende slide)
  - `String.Format("{0:string}", date)` => *string* is de format specifier
  - Maak je in een string gebruik van een escape character dan \ verdubbelen ofwel laat je string voorafgaan door @-sign.

```
static void Main(string[] args)
{
    DateTime date1 = new DateTime(2000, 8, 29, 19, 27, 15);
    Console.WriteLine(date1.ToString("ddd d/M H:mm:ss"));
    Console.WriteLine(String.Format("{0:ddd d/M H:mm:ss}", date1));
    Console.WriteLine(date1.ToString("h \\h") );
    Console.WriteLine(date1.ToString(@"h \h"));
    Console.ReadLine();
}
```



```
di 29/8 19:27:15
di 29/8 19:27:15
7 h
7 h
```

Format	Description
"d"	The day of the month, from 1 to 31.
"dd"	The day of the month, from 01 to 31.
"ddd"	The abbreviated name of the day of the week.
"dddd"	The full name of the day of the week.
"m"	The minute, from 0 through 59.
"mm"	The minute, from 00 through 59.
"M"	The month, from 1 through 12.
"MM"	The month, from 01 through 12.
"MMM"	The abbreviated name of the month.
"MMMM"	The full name of the month.
"yy"	The year, from 00 to 99.
"yyy"	The year, with a minimum of 3 digits.
"yyyy"	The year as a four-digit number.
"yyyyy"	The year as a five-digit number.
"H"	The hour, using a 24-hour clock from 0 to 23.
"h"	The hour, using a 12-hour clock from 1 to 12.

Format	Description
"."	The time separator (as defined in regional settings)
"/"	The date separator (as defined in regional settings).
"\"	The escape character. Ex. <code>DateTime.Now.ToString("h \\h")</code> -> 1 h Or <code>DateTime.Now.ToString("@\"h \\h")</code> -> 1 h



## 2. Data types : StringBuilder

- ▶ System.Text.StringBuilder
  - Kan wel benaderd worden als een array (indexed).
  - Betere performantie indien de string objecten vaak wijzigen.
  - Slaat een string op als een array van characters. Editeren betekent geen nieuw string object
  - Properties

Capacity	Gets or sets the maximum number of characters that can be contained in the memory allocated by the current instance.
Chars	Gets or sets the character at the specified character position in this instance.
Length	Gets or sets the length of this instance.

## 2. Data types : StringBuilder

- ▶ System.Text.StringBuilder (vervolg)
  - Methodes

AppendFormat	Appends a formatted string, which contains zero or more format specifications, to this instance. Each format
Equals	Returns a value indicating whether this instance is equal to a specified object.
Insert	Overloaded. Inserts the string representation of a specified object into this instance at a specified character position.
Remove	Removes the specified range of characters from this instance.
Replace	Replaces all occurrences of a specified character or string in this instance with another specified character or string.
ToString	Coverts a StringBuilder to a String

# Operatoren

# 3. Operatoren

- ▶ Rekenkundige operatoren
  - $+, -, *, /$ , %(rest na deling)
- ▶ Toekenningsoperatoren
  - $=, +=, -=, \dots$
- ▶ Vergelijkingsoperatoren
  - $<, >, ==, >=, <=, !=$
- ▶ Logische operatoren
  - $\&, |$  : bitwise AND, OR
  - $\&\&, ||$  : conditionele AND, OR
  - $\wedge$  : XOR

# 3. Operatoren

## ▶ ?? Operator

- retourneert de linkerkant van de operand als niet null, anders de rechterkant.

```
int? x = null;  
  
// Set y to the value of x if x is NOT null; otherwise,  
// if x = null, set y to -1.  
int y = x ?? -1;
```

## ▶ Null conditional operator

- Test op null alvorens een member access te doen

```
int? length = customers?.Length; // null if customers is null  
Customer first = customers?[0]; // null if customers is null
```

# Controle structuren

# 4. Controle structuren

## ▶ Selectie structuren

- if (conditie) {statements} else {statements}
- switch (conditie) {  
    case waarde : {Statements} ...  
    default:{statements}  
}

## ▶ Iteratie structuren

- while (conditie) {statements};
- do {statements} while (conditie) ;
- for (initialization; condition; increment) {Statements}
- foreach (ElementVarDecl in Collection) {Statements}
- break; continue;
- return;

## 4. Controle structuren

```
int hour = DateTime.Now.Hour;
if (hour < 12)
    Console.WriteLine( "Good morning");
else if (hour < 18)
    Console.WriteLine( "Good afternoon");
else
    Console.WriteLine( "Good evening");
```

Of verkort (?:)

`a > 0 ? 5 : 10;`

Of null conditional operator

`p?.Name ?? "no name";`


```
Console.WriteLine("Do you enjoy C# ? (yes/no/maybe)");
string input = Console.ReadLine();
switch (input.ToLower())
{
    case "yes":
    case "maybe":
        Console.WriteLine("Great!"); break;
    case "no":
        Console.WriteLine("Too bad!"); break;
    default:
        Console.WriteLine("I'm sorry, I don't understand that!"); break;
}
```



# 4. Controle structuren

- ▶ Nieuw in C# 7 : pattern matching
  - Laat matching toe o.b.v. type

```
// ----- Assume that spaceltem is of type SpaceType, and that Planet and Star derive from SpaceType.
switch (spaceltem)
{
    case Planet p:
        if (p.Type != PlanetType.GasGiant)
            LandSpacecraft(p);
        break;
    case Star s when (s.Id > 1000):
        AvoidHeatSource(s);
        break;
    case null:
        // ----- If spaceltem is null, processing falls here, even if it is a Planet or Star null instance.
        break;
    default:
        // ----- Anything else that is not Planet, Star, or null.
        break;
}
```



A diagram with a light blue box containing the text "Pattern matching variable". Two blue arrows point from this box to the code. The first arrow points to the variable "p" in the line "case Planet p:". The second arrow points to the variable "p" in the line "if (p.Type != PlanetType.GasGiant)".

## 7. 4. Controle structuren

- ▶ New in C# 7.0 Pattern matching :
  - Type pattern : test of een uitdrukking kan worden geconverteerd naar een opgegeven type en, indien mogelijk, naar een variabele van dat type.

```
static string IsEmployee(Object o)
{
    if (o is Employee e) //is hetzelfde als var e = o as Employee; if (e!=null)
    {
        return "o is an Employee object";
    }
    return "o is not an Employee object.";
}
```

- Constant pattern: test of een expressie evalueert naar een opgegeven constante waarde.

```
static void IsHigh(Object o)
{
    const int HIGH_ROLL = 6;
    if (o is Dice d && d.Roll() is HIGH_ROLL)
    {
        Console.WriteLine($"The value is {HIGH_ROLL}");
    }
    Console.WriteLine($"The dice roll is not a {HIGH_ROLL}!");
}
```

# 7. What's new in c# 8

- ▶ New in C# 7.0 Pattern matching : Pattern matching
  - var pattern : idem aan type pattern maar de waarde van de variabele kan alle waarden bevatten, ook null
    - In onderstaand vb test de 4de case op null, the empty string, of any string that contains only white space.

```
static object CreateShape(string shapeDescription)
{
    switch (shapeDescription)
    {
        case "circle":
            return new Circle(2);

        case "square":
            return new Square(4);

        case "large-circle":
            return new Circle(12);

        case var o when (o?.Trim().Length ?? 0) == 0:
            // white space
            return null;
        default:
            return "invalid shape description";
    }
}
```

## 4. Controle structuren

```
int i = 2;
while (i <= 100) {
    i = i * 2;
    Console.WriteLine(i.ToString() );
}
```

```
int i = 2;
do {
    i = i * 2;
    Console.WriteLine(i.ToString() );
} while (i <= 100);
```

```
for (int i = 0; i < 10; i++){
    Console.WriteLine(i.ToString() );
}
```

# 4. Controle structuren

```
foreach (int age in ages){  
    if (age==21) break;    //verlaat de foreach  
    if (age==2) continue; //voer vervolg code niet uit, ga naar volgende age  
    Console.WriteLine(age.ToString() );  
}
```

# Arrays

# 5. Arrays

## ► Array

### ◦ 1 dimensioneel

- `int[ ] a; //declareert een array van integers`
- `int[ ] b = new int[3]; //initialiseert een array van 3 integers`
- `int[ ] c = new int[3]{3,4,5}; //initialiseert c met de waarden 3,4,5`
- `int[ ] d = {3,4,5}; //idem`
- `int[ ] e;`  
`e = new int[3];`

```
int[] rij = new int[3] { 3, 4, 5 };

for (int i = 0; i < rij.Length; i++)
    rij[i]++;

foreach (int getal in rij)
    Console.WriteLine(getal);
```

# 5. Arrays

- Meerdimensioneel
  - Rectangular:
    - `int[,] a = new int[2,3];` //twee rijen – drie kolommen
    - `int x=a[0,1]`
  - Jagged: rijen kunnen verschillende lengte hebben
    - `int[][] a = new int [2][];` //twee rijen – aantal kolommen onbepaald
    - `a[0]=new int[3];` // 3 kolommen in rij 1
    - `a[1]=new int[4];` // 4 kolommen in rij 2
    - `int x = a[0][1];`
- Operaties
  - `a.Length` //totaal aantal elementen
  - `a.GetLength(0)` //aantal elementen in dimensie 1
  - `Array.Copy(b,a,2)` //kopieert `b[0..1]` naar `a`
  - `Array.Sort(b);` //sorteert `b` in oplopende volgorde



# Exceptions

# 6. Exceptions

---

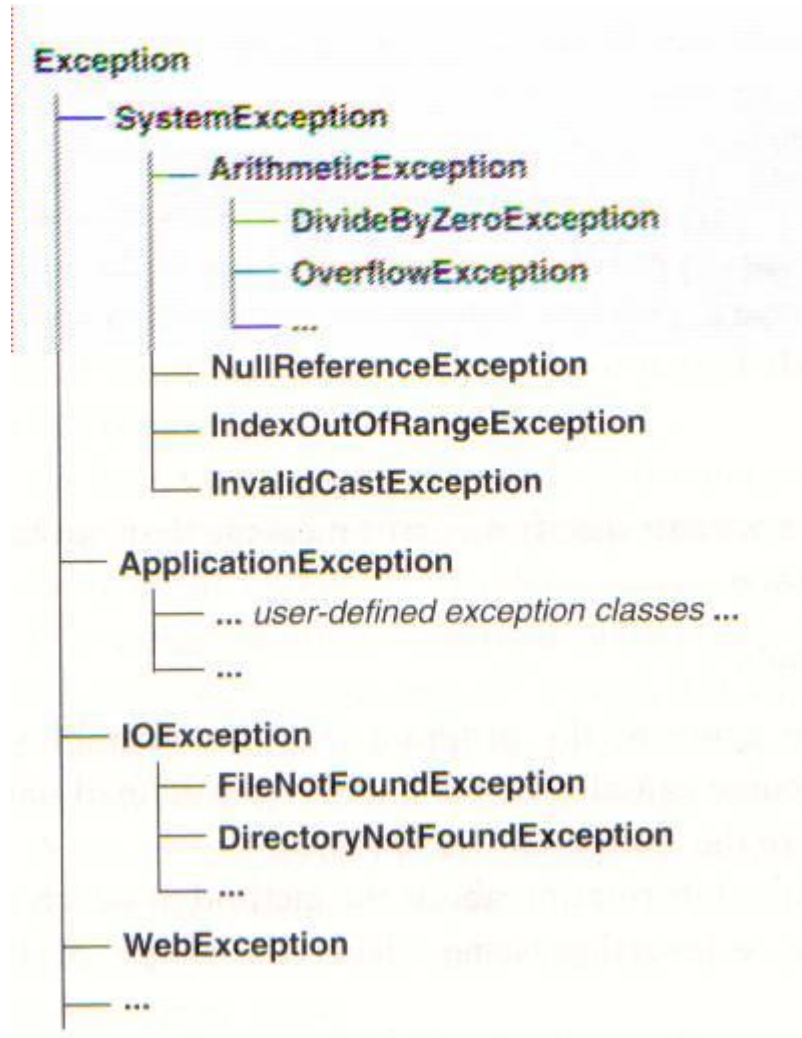
- ▶ Definitie
- ▶ System.Exception
- ▶ Throwen van een exception
- ▶ Afhandelen van een exception

# 6. Exceptions

## ► Definitie

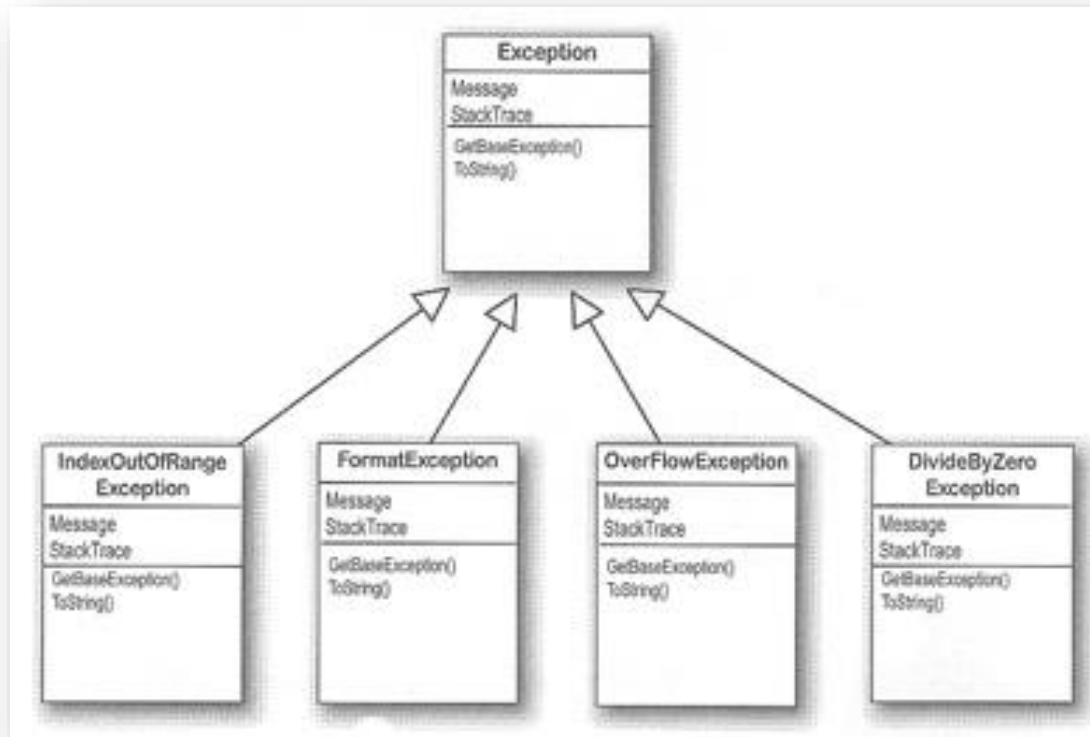
- Exception
  - Indicatie van een onverwacht probleem at run-time
  - Zonder foutafhandeling wordt programma afgesloten met een run-time error.
- Exception classes zorgen voor een gestructureerde foutafhandeling (error-handling)
- Errorhandling gebeurt dmv van error-objecten. Deze kunnen geworpen (thrown) worden op de plaats waar de fout zich voordoet en opgevangen (caught) op de plaats waar ze moet verwerkt worden.
- De exception class en subclasses hebben een hiërarchische structuur

# 6. Exceptions



# 6. Exceptions

- ▶ Fouten genereren uitzonderingen (exceptions)
  - Als er een fout optreedt, wordt een Exception object gecreëerd, met informatie over de uitzondering.



Exception is een basisklasse. Zo kan je je eigen Exception klassen schrijven.

# 6. Exceptions

- ▶ Fouten genereren uitzonderingen (exceptions)
  - Het Exception-object bevat informatie over de fout, en terwijl de gebeurtenis door de lagen heen naar boven borrelt, wordt de gebeurtenis ingepakt in steeds meer details. Het komt er ruwweg op neer dat de `Application_Error`-uitzondering de `Page_Error`-uitzondering bevat, die weer voortvloeit uit de basis-Exception, die het omhoog borrelen van de gebeurtenis in gang heeft gezet.
- ▶ Fouten dien je zoveel mogelijk op te vangen
- ▶ Igv niet opgevangen fouten, leidt de gebruiker om naar een fout pagina
- ▶ Indien gewenst kan je niet opgevangen fouten loggen op applicatie niveau

# 6. Exceptions

- ▶ **System.Exception properties : info over fout**
  - Message : foutmelding gekoppeld aan uitzondering
    - Default boodschap : geassocieerd met exception type
    - Gecustomiseerde boodschap : doorgegeven aan constructor van object
  - Source : naam van applicatie of object die fout veroorzaakt heeft
  - TargetSite : naam methode die fout gegenereerd heeft
  - StackTrace : exception historiek.1 string die een sequentiële lijst van methodes bevat, die op moment dat de exception zich voordoet, nog niet volledig zijn uitgevoerd
  - InnerException : voor geneste exceptions
  - HelpLink : link naar Help file geassocieerd met fout
  - ToString : retourneert een string met de naam van de exception, de exception message, de naam van de inner exception en de stack.

# 6. Exceptions

- ▶ Throwen van een exception
  - impliciet: door de CLR, ten gevolge van ongeldige operations: delen door nul, array access met een foute index, member access met een null reference .... Deze maakt een Exception object aan, met informatie over de fout
  - expliciet: door de programmeur zelf...

```
public int BerekenGemiddelde(int totaal , int aantal)
{
    if (aantal <= 0)
        throw new ArgumentException("Aantal moet groter zijn dan nul");
    else
        return totaal/aantal;
}
```



# 6. Exceptions

---

## ▶ Throwing

- 2 strekkingen
  - Enkel in uitzonderlijke gevallen. Werk anders verder met defaultwaarden
  - Van zodra het gewenste gedrag niet kan worden uitgevoerd.
    - Foutieve inputwaarden
      - throw een `ArgumentException`
    - Iets lukt niet en je kan het ook niet verhelpen
      - bvb opslaan van een gebruiker maar database niet beschikbaar, creatie gebruiker en gebruiker bestaat reeds.

# 6. Exceptions

## ► Afhandelen van exceptions

```
try
{
    //protected statement sequence
    ....
}
catch (type of exception ex)
{
    ...
}
catch (...)
{...}
finally
{
    ...
}
```

# 6. Exceptions

---

- ▶ Afhandelen van exceptions (meerdere exceptionhandlers)
  - .NET bevat tal van voorgedefinieerde exception klassen die afgeleid zijn van de basis klasse Exception in de System Namespace
  - In foutafhandeling kan je meerdere catch blokken voorzien, die elk een specifieke fout afhandelen
  - Volgorde van catch blokken is belangrijk! Runtime gaat op zoek naar eerste catch blok met een type fout waarvoor de “is een “ regel geldt.
  - Zoek op in help : Elke methode die exceptions throwt heeft een sectie exceptions die de mogelijke exceptions beschrijft

# 6. Exceptions

- ▶ Afhandelen van exceptions
  - Hoe gaat runtime op zoek naar handlers indien je werkt met geneste Try blokken ?
    - Bij fout wordt de uitvoering van code onmiddellijk afgebroken
    - De runtime gaat op zoek naar fouthandler in bijhorend catch blokken. Als bijhorende handler gevonden wordt, wordt fout afgehandeld en wordt verdergegaan met uitvoering van programma
    - Als er geen geschikte handler is voor fout in de bijhorende catch blokken, dan wordt de verwerking van de omvattende methodes (try blokken) afgebroken tot een corresponderende handler gevonden
    - Als er geen omvattende try blokken zijn met een passende handler, dan handelt de runtime zelf de fout af met een runtime error

# 6. Exceptionhandling

## ► Opvangen van fouten

- try ... catch...finally
  - Voorbeeld : delen door 0, index out of range,...
  - In de Help kan je voor elke methode terugvinden wat de mogelijke exceptions zijn
    - Zoek in Help naar division operator [http://msdn.microsoft.com/en-us/library/aa691373\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/aa691373(v=vs.71).aspx)
    - Onderaan of in aparte Exceptions Sectie worden de exceptions beschreven

If the value of the right operand is zero, a `System.DivideByZeroException` is thrown. If the resulting value is too large to represent in the `decimal` format, a `System.OverflowException` is thrown. If the result value is too small to represent in the `decimal` format, the result is zero. The scale of the result is the smallest scale that will preserve a result equal to the nearest representable decimal value to the true mathematical result.

# 6. Exceptionhandling

## ► try ... catch...finally

```
class Program
{
    static double SafeDivision(double x, double y)
    {
        if (y == 0)
            throw new System.DivideByZeroException();
        return x / y;
    }

    static void Main()
    {
        double a = 100, b = 0;
        try
        {
            double result = SafeDivision(a, b);
            Console.WriteLine(String.Format("{0} divided by {1} = {2}", a, b, result));
        }
        catch (DivideByZeroException e)
        {
            Console.WriteLine("Attempted divide by zero.");
        }
    }
}
```

# 6. Exceptionhandling

- try ... catch...finally
  - Enkele tips
    - Ontwerp code met exceptionhandling
    - Als je een exception moet opvangen, gebruik steeds de meest specifieke exception (niet gewoon Exception)
    - Maak NOOIT een leeg catch blok aan
    - Geef aan gebruikers bruikbare foutmeldingen, zodat gebruiker weet hoe hij fout kan aanpassen
    - Werp waar mogelijk .Net Exceptions. Werp enkel custom exceptions als je meer informatie wenst te geven. Geef dan ook de innerexception mee!
    - Vergeet het finally blok niet waar nodig. Zorg voor clean-up. (using is vaak een goed alternatief)
    - Meer op : <http://msdn.microsoft.com/en-us/library/5b2yeyab.aspx>

# 6. Exceptions

## ▶ Afhandelen van exceptions



**Gotta catch 'em all!**

- 2 gouden regels
  - Handel enkel exceptions af als je er echt iets kan aan doen
  - Aan de meeste exceptions kan je niets doen
- Zie later : wat met runtime exceptions in webapplicaties :  
loggen en gebruiksvriendelijke fout tonen aan gebruiker



# 6. Exceptions

## ► Custom Exception

- Maak een nieuwe klasse aan, erft van een base exception
- Gebruik suffix Exception voor de klassenaam
- Maak exception serializable
- Aanmaken : voeg een nieuwe klasse toe, verwijder inhoud file, rechtsklik > Add snippet > Visual C# > Exception. Pas de naam van de exception aan en klik tab.

```
[Serializable]
public class InvalidAccountValueException : Exception
{
    public InvalidAccountValueException() { }
    public InvalidAccountValueException(string message) : base(message) { }
    public InvalidAccountValueException(string message, Exception inner) : base(message, inner) { }
    protected InvalidAccountValueException(
        System.Runtime.Serialization.SerializationInfo info,
        System.Runtime.Serialization.StreamingContext context)
        : base(info, context) { }
}
```

# 6. Exceptions

- ▶ Exception filters
  - Voegen condities toe aan een catch block

```
static void Main()
{
    try
    {
        Foo.DoSomethingThatMightFail(null);
    }
    catch (MyException ex) when (ex.Code == 42)
    {
        Console.WriteLine("Error 42 occurred");
    }
}
```

- <https://www.thomaslevesque.com/2015/06/21/exception-filters-in-c-6/>

# 7. What's new in c# 8

- ▶ <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8>
- ▶ Om C#8.0 te kunnen gebruiken
  - Rechtsklik op project > Edit Project File
  - Voeg onderstaande toe

```
<PropertyGroup>  
  <OutputType>Exe</OutputType>  
  <TargetFramework>netcoreapp2.2</TargetFramework>  
  <Nullable>enable</Nullable>  
  <LangVersion>8.0</LangVersion>  
</PropertyGroup>
```

- We bespreken de belangrijkste nieuwigheden

# 7. What's new in c# 8

- ▶ Switch expressions (ter vervanging van switch case)
  - Case en : worden vervangen door =>
  - De default case wordt vervangen door \_
  - Geen break statement

```
string input = Console.ReadLine();  
Console.WriteLine(input.ToLower() switch {  
    "yes" => "Fantastic",  
    "maybe" => "Great",  
    "no" => "Too bad!",  
    _ => "I'm sorry, I don't understand that!"  
});
```

# 7. What's new in c# 8

- ▶ Switch expressions (ter vervanging van switch case)
  - Case en : worden vervangen door =>
  - De default case wordt vervangen door \_
  - Geen break
  - Voorbeeld

```
string input = Console.ReadLine();  
Console.WriteLine(input.ToLower() switch {  
    "yes" => "Fantastic",  
    "maybe" => "Great",  
    "no" => "Too bad!",  
    _ => "I'm sorry, I don't understand that!"  
});
```

# 7. What's new in c# 8

## ► Switch expressions

- Die gebruik maken van property patterns : test op het type en de waarde van een property

```
Object vehicle = new Car() { Passengers = 5 };  
var toll = vehicle switch  
{  
    Car { Passengers: 0 } => 2.00m + 0.50m,  
    Car { Passengers: 1 } => 2.0m,  
    Car { Passengers: 2 } => 2.0m - 0.50m,  
    Car c => 2.00m - 1.0m,  
  
    Bus b when ((double)b.Riders / (double)b.Capacity) < 0.50 => 5.00m + 2.00m,  
    Bus b when ((double)b.Riders / (double)b.Capacity) > 0.90 => 5.00m - 1.00m,  
    Bus b => 5.00m,  
  
    _ => 10.0m  
};
```

# 7. What's new in c# 8

- ▶ Switch expressions
  - Tuple matching

```
public static string RockPaperScissors(string first, string second)
{
    return (first, second) switch
    {
        ("rock", "paper") => "rock is covered by paper. Paper wins.",
        ("rock", "scissors") => "rock breaks scissors. Rock wins.",
        ("paper", "rock") => "paper covers rock. Paper wins.",
        ("paper", "scissors") => "paper is cut by scissors. Scissors wins.",
        ("scissors", "rock") => "scissors is broken by rock. Rock wins.",
        ("scissors", "paper") => "scissors cuts paper. Scissors wins.",
        (_, _) => "tie"
    };
}
```

- ▶ Nog meer voorbeelden op  
: <https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/pattern-matching>

# 7. What's new in c# 8

## ► Null reference

- Bad code => gives a runtime error

```
static void Main(string[] args)
{
    string s = null;
    Console.WriteLine($"The first letter of {s} is {s[0]}");
}
```

- Oplossing : null reference. Voeg volgende lijn toe aan .csproj :  
<Nullable>enable</Nullable>

- 2 warnings.

⚠ CS8600 Converting null literal or possible null value to non-nullable type.  
⚠ CS8602 Dereference of a possibly null reference.

- Wegwerken van de warnings, kan op 2 manieren

De variabele s kan geen null waarde bevatten

De compiler verzekert dat een not null waarde wordt toegekend bij declaratie of in constructor

```
static void Main(string[] args)
{
    string s = "abc";
    Console.WriteLine($"The first letter of {s}");
}
```

De variabele s kan null waarde bevatten

```
static void Main(string[] args)
{
    string? s = null;
    Console.WriteLine($"The first letter of {s} is {s?[0]?'?'}");
}
```

Een mooi voorbeeld: <https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/nullable-reference-types>



# 7. What's new in c# 8

## ► Indices and ranges (vanaf .net core 3.0)

- System.Index: staat voor een index in een reeks.
  - De operator ^: index relatief aan het einde van een reeks.
- System.Range: een subbereik van een reeks.
  - De operator Range (..) : geeft begin en einde van een bereik op als operanden.

- Meer op

<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/ranges-indexes>

```
var words = new string[]{  
    // index from start  index from end  
    "The",    // 0        ^9  
    "quick",  // 1        ^8  
    "brown",  // 2        ^7  
    "fox",    // 3        ^6  
    "jumped", // 4        ^5  
    "over",   // 5        ^4  
    "the",    // 6        ^3  
    "lazy",   // 7        ^2  
    "dog"     // 8        ^1  
};  
  
string[] allWords = words[..]; // contains "The" through "dog".  
string[] quickBrownFox = words[1..4]; //contains "The" through "fox"  
string[] first4Words = words[..4]; // contains "The" through "fox"  
string[] lastWords = words[6..]; // contains "the", "lazy" and "dog"  
string lastWord = words[^1]; //dog  
string lazyDog = words[^2..^0]; //end index ^0 is not included. contains "lazy" and "dog"  
Range phrase = 1..4;  
string[] wordsFromRange = words[phrase];
```

# 7. What's new in c# 8

- ▶ Null-coalescing assignment operator ??= (vanaf .net core 3.0)
  - wijst de waarde van de rechteroperand alleen toe aan de linkeroperand als de linkeroperand nul oplevert.

```
List<int> numbers = null;  
int? i = null;  
numbers ??= new List<int>();  
numbers.Add(i ??= 17);  
numbers.Add(i ??= 20);
```

```
Console.WriteLine(string.Join(' ', numbers)); // output: 17 17  
Console.WriteLine(i); // output: 17
```

# Appendix

# Appendix: Documentatie en Tutorials

---

- ▶ Tutorials :
  - <https://www.microsoftvirtualacademy.com/>
  - <http://www.csharp-station.com/Tutorial.aspx>
- ▶ C# Programming Guide :
  - <http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx>
- ▶ C# Reference :
  - <http://msdn.microsoft.com/en-us/library/618ayhy6.aspx>
- ▶ Pluralsight : C# Fundamentals