

Hoofdstuk 2 : C# - Model – TDD

HoGent

Hoofdstuk 2 : C# - Model – TDD

1. Symbolen
2. Datatypes
3. Operatoren
4. Controle structuren
5. Array
6. Collections
7. Exceptions
8. Klassen
9. Unit testen
10. Associaties
11. Overerving
12. Polymorfisme
13. Abstracte klasse
14. Interface
15. Statische members
16. Generic klasse
17. Naming Conventions
18. Documenteren C# code via XML
19. Oefening

HoGent

C#

- ▶ Deel 1 – deel 7 is zelfstudie
 - Kennismaking met de C# taal
 - Bekijk de slides
 - Of Neem de tutorial [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) door
 - Of op Pluralsight, bekijk C# Fundamentals – Part 1
 - An introduction to C#
 - C# types
 - C# Flow controls - Exceptions

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: /*
.....
..... */

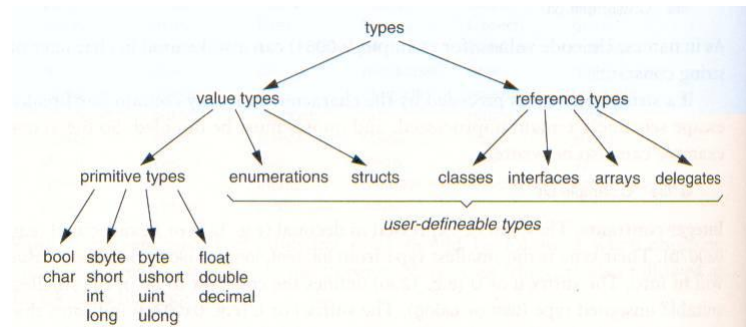
2. Data types

- ▶ C# is strongly typed
 - Elke variable en object is van een gedeclareerd type
- ▶ Een data type is:
 - een built-in 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)	byte
	Int16	A 16-bit signed integer. (-32.768 .. 32.767)	short
	Int32	A 32-bit signed integer. (-2.147.483.648.. 2.147.483.647)	int
	Int64	A 64-bit signed integer. ($-2^{63} .. 2^{63} - 1$)	long
Floating point	Single	A single-precision: 32-bit floating-point number. ($\pm 1.4E-45 .. \pm 3.4E38$)	float
	Double	A double-precision: 64-bit floating-point number. ($\pm 5E-324 .. \pm 1.7E308$)	double
Logical	Boolean	A Boolean value (true or false).	bool
Other	Char	A Unicode (16-bit) character.	char
	Decimal	A decimal value. 96 bit signed number	decimal
Class objects	Object	The root of the object hierarchy.	object
	String	An immutable, fixed-length string of Unicode characters. Limited by system memory	string

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

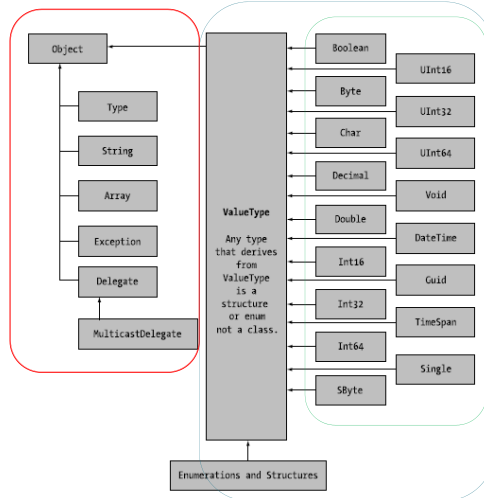


Figure 3-6. The class hierarchy of system types

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 : 0.0D
- float : 0.0F
- long : 0L
- DateTime : 1/1/0001 12:00:00 AM

2. Data types : Value types

▸ struct type

- Is vergelijkbaar met een klasse. Een struct type kan fields, methods en constructors hebben, *maar wordt gemanaged op de stack en dus doorgegeven by value.*
- 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

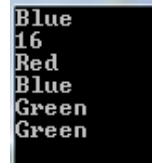
- Lijst van constante namen en de overeenkomstige numerische waarde (default van type System.Int32). Numerisch waarde wordt op de stack gezet.

```
//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. 116  
    //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();  
}
```



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
Equals(Object)	Determines whether the specified Object is equal to the current Object .
Equals(Object, Object)	Determines whether the specified object instances are considered equal.
Finalize	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.
GetHashCode	Serves as a hash function for a particular type.
GetType	Gets the Type of the current instance.
MemberwiseClone	Creates a shallow copy of the current Object .
ReferenceEquals	Determines whether the specified Object instances are the same instance.
ToString	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 <code>String</code> array containing the substrings in this instance that are delimited by elements of a specified <code>Char</code> or <code>String</code> 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

- 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;`

HoGent

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)

HoGent

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();
```

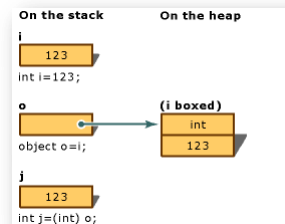
- Verschil tussen Convert en Parse : Convert.ToInt32 kan overweg met null waarden, int.Parse throwt ArgumentNullException

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)
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 date component of this instance.
DayOfWeek	Gets the day of the week 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 ValueType.ToString() .)
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

► System.TimeSpan

Methods	Description
Add	Returns a new TimeSpan object whose value is the sum of the specified TimeSpan object and this instance.
FromHours	Returns a TimeSpan that represents a specified number of hours, accurate to the nearest millisecond.
FromMinutes	Returns a TimeSpan that represents a specified number of minutes, accurate to the nearest millisecond.
FromSeconds	Returns a TimeSpan that represents a specified number of seconds, accurate to the nearest millisecond.
Parse(String)	Converts the string representation of a time interval to its TimeSpan equivalent.
Subtract	Returns a new TimeSpan whose value is the difference between the specified TimeSpan object and this instance.
ToString()	Converts the value of the current TimeSpan object to its equivalent string representation
Properties	Description
Days	Gets the days component of the time interval represented by the current TimeSpan structure.
Hours	Gets the hours component of the time interval represented by the current TimeSpan structure.
Minutes	Gets the minutes component of the time interval represented by the current TimeSpan structure.
Seconds	Gets the seconds component of the time interval represented by the current TimeSpan structure.
TotalHours	Gets the value of the current TimeSpan structure expressed in whole and fractional hours.
TotalMinutes	Gets the value of the current TimeSpan structure expressed in whole and fractional minutes.

Pag. 27

2. Data Types : Date en 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
? h
? h
```

Format	Description	Format	Description
"d"	The day of the month, from 1 to 31.	","	The time separator (as defined in regional settings)
"dd"	The day of the month, from 01 to 31.	"/"	The date separator (as defined in regional settings).
"ddd"	The abbreviated name of the day of the week.	"\"	The escape character. Ex. <code>DateTime.Now.ToString("h \\h") -> 1 h</code> Or <code>DateTime.Now.ToString(@"h \h") -> 1 h</code>
"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.		

Pag. 29

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

3. Operatoren

► Rekenkundige operatoren

- +, -, *, /, %(rest na deling)

► Toekenningsoperatoren

- =, +=, -=, ...

► Vergelijkingsoperatoren

- <, >, ==, >=, <=, !=

► Logische operatoren

- &, | : bitwise AND, OR
- &&, || : conditionele AND, OR
- ^ : XOR

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 (?:)

b = a > 0 ? 5 : 10;

```
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

```
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() );
}
```

5. Arrays

► Array

◦ 1 dimensioneel

- `int[] a; //declareert een array van integers`
- `int[] b = new int[3]; //initializeert 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

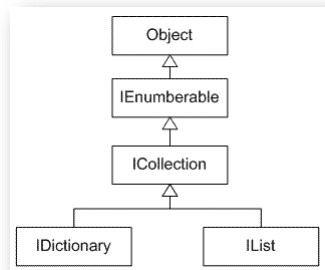
- `int[][] a = new int [2][]; //twee rijen – aantal kolommen onbepaald`
- `a[0]=new int[3]; // 3 kolommen in rij 1`
- `a[1]=new int[4]; // 4 kolommen in rij 2`
- `int x = a[0][1];`

◦ 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`

6. Collections

- ▶ Namespace : System.Collections.Generic
- ▶ Een generische collection is strongly typed (type safe) : dit betekent dat het maar 1 type van object kan bevatten
- ▶ Collections worden ook generics genoemd in .Net



6. Collections

- ▶ IEnumerable<T>
 - Interface : Biedt enumerator om door een collectie te lopen.

```
IEnumerable<int> ints = new List<int> { 1, 2, 3 };  
foreach (int i in ints)  
    Console.WriteLine(i.ToString());  
Console.ReadLine();
```

- Opmerking : T is een generic type parameter die je bij definitie van lijst moet opgeven.

6. Collections

► ICollection<T>

- Interface : Implementeert IEnumerable<T>
- Voegt **Count property** toe aan een collectie
 - Retourneert aantal objecten in collectie

```
ICollection<int> ints = new List<int> { 1, 2, 3 };  
foreach (int i in ints)  
    Console.WriteLine(i.ToString());  
Console.WriteLine(ints.Count);  
Console.ReadLine();
```

6. Collections

► IList<T>

- Interface : Implementeert ICollection<T>
- Voegt manipulatie methodes toe aan collectie
- Property
 - Item ([i]) : retourneert i^{de} element. Index start vanaf 0

Name	Description
Add	Adds an item to the IList .
Clear	Removes all items from the IList .
Contains	Determines whether the IList contains a specific value.
CopyTo	Copies the elements of the ICollection to an Array , starting at a particular Array index. (Inherited from ICollection .)
GetEnumerator	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable .)
IndexOf	Determines the index of a specific item in the IList .
Insert	Inserts an item to the IList at the specified index.
Remove	Removes the first occurrence of a specific object from the IList .
RemoveAt	Removes the IList item at the specified index.

6. Collections

► IList<T>

```
IList<int> ints = new List<int> ();
ints.Add(1);
ints.Add(2);
ints.Add(3);
foreach (int i in ints)
    Console.WriteLine(i.ToString());
Console.WriteLine(ints[1].ToString());
Console.WriteLine(ints.Count);
Console.ReadLine();
```

6. Collections

- List<T>
 - Concrete strongly typed lijst van elementen van type T
 - Mogelijkheid tot manipuleren, sorteren en zoeken
- HashSet<T>
 - Idem List, maar laat geen dubbels toe (werkt met hashkey voor T)
- Queue<T>
 - first-in, first-out collectie van objecten
- Stack<T>
 - Represents a variable size last-in-first-out (LIFO) collection of instances of the same arbitrary type.
- LinkedList<T>
 - Represents a doubly linked list.
- SortedSet<T>
 - Represents a collection of objects that is maintained in sorted order.

6. Collections

- ▶ **IDictionary<Tkey, TValue>**
 - Verzameling van naam/waarde paren
 - Tkey = type van de sleutel in de dictionair, TValue = type van de waarde in de dictionair
 - Properties
 - Keys : lijst van de sleutels
 - Values : lijst van de waarden
 - Item [key] : retourneert value van item met specifieke key
 - Methodes

6. Collections

- ▶ **IDictionary<Tkey, TValue>**
 - Methodes

Name	Description
Add(T)	Adds an item to the ICollection<T> . (Inherited from ICollection<T> .)
Add(TKey, TValue)	Adds an element with the provided key and value to the IDictionary<TKey, TValue> .
Clear	Removes all items from the ICollection<T> . (Inherited from ICollection<T> .)
Contains	Determines whether the ICollection<T> contains a specific value. (Inherited from ICollection<T> .)
ContainsKey	Determines whether the IDictionary<TKey, TValue> contains an element with the specified key.
CopyTo	Copies the elements of the ICollection<T> to an Array , starting at a particular Array index. (Inherited from ICollection<T> .)
GetEnumerator()	Returns an enumerator that iterates through a collection. (Inherited from IEnumerable .)
GetEnumerator()	Returns an enumerator that iterates through the collection. (Inherited from IEnumerable<T> .)
Remove(T)	Removes the first occurrence of a specific object from the ICollection<T> . (Inherited from ICollection<T> .)
Remove(TKey)	Removes the element with the specified key from the IDictionary<TKey, TValue> .
TryGetValue	Gets the value associated with the specified key.

6. Collections

▸ IDictionary<Tkey, TValue>

```
IDictionary<int, string> klasLijst = new Dictionary<int, string> ();  
klasLijst.Add(1, "Jan Peterson");  
klasLijst.Add(2, "Steven Spielberg");  
foreach (int i in klasLijst.Keys)  
    Console.WriteLine(i.ToString() + " : " + klasLijst[i]);  
foreach (string s in klasLijst.Values)  
    Console.WriteLine(s);  
Console.WriteLine(klasLijst[1]);  
Console.WriteLine(klasLijst.Count);  
Console.ReadLine();
```

```
1 : Jan Peterson  
2 : Steven Spielberg  
Jan Peterson  
Steven Spielberg  
Jan Peterson  
2
```

6. Collections

▸ Dictionary<Tkey, TValue>

- Concrete collectie met key/waarde paren

▸ SortedDictionary<Tkey, TValue>

- Represents a collection of key/value pairs that are sorted on the key.

6. Collections

- ▶ yield return
 - Voor het bouwen van een IEnumerable

```
private IEnumerable<int> ComputeAges()  
{  
    yield return 21;  
    yield return 22;  
    for (int i = 23; i < 32; i++)  
        yield return i;  
}
```

```
foreach(int age in ComputeAges())  
    Console.WriteLine(age.ToString());
```

De eerste iteratie in **foreach** loop zorgt voor de uitvoering van de ComputeAges t.e.m. het eerste **yield return** statement. Deze iteratie retourneert 21, en de huidige locatie in de ComputeAges methode wordt behouden.

Bij de volgende iteratie in foreach gaat de uitvoering in de iteratie methode ComputeAges verder en wordt 22 geretourneerd,... tot einde iteratie methode bereikt

7. Exceptions

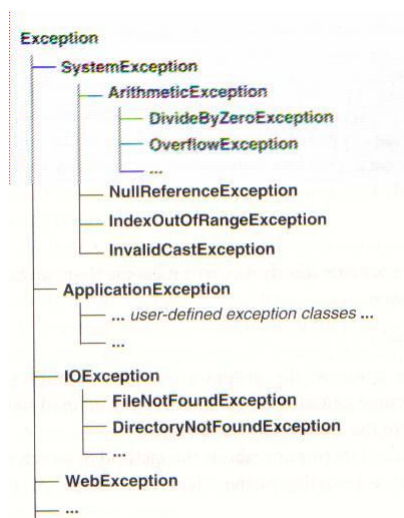
- ▶ Definitie
- ▶ System.Exception
- ▶ Throwen van een exception
- ▶ Afhandelen van een exception

7. 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

7. Exceptions



7. 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 sekventiële 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.

7. 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;
}
```

7. Exceptions

► Throwing

- 2 strekkingen
 - Enkel in uitzonderlijke gevallen. Werk anders verder met defaultwaarden
 - Van zodra het gewenste gedrag niet kan worden uitgevoerd.
 - Foutieve inputwaarden
 - 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.

7. Exceptions

► Afhandelen van exceptions

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

7. 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
 - Enkele voorbeelden : Exceptionklassen igv openen file
 - SecurityException : gebruiker heeft niet de vereiste permissies
 - ArgumentException : path is leeg, bevat enkel blanco's of illegale chars
 - FileNotFoundException : bestand bestaat niet
 - ArgumentNullException : path of mode is een NULL verwijzing
 - UnauthorizedAccessException : file is read-only of file is een pad
 - DirectoryNotFoundException : directory niet gevonden

7. 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

7. Exceptions

► Afhandelen van exceptions

◦ Voorbeeld

```
protected void buttonBerekenGemiddelde_Click(object sender, EventArgs e)
{
    try
    {
        int gemiddelde =
            berekenGemiddelde(int.Parse(textboxTotaal.Text),int.Parse(textBoxAantal.Text));
    }
    catch (ArgumentException ex) {
        labelMessage.Text = ex.Message;
    }
}
```

7. 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

7. Exceptions

► Custom Exception

- Maak een nieuwe klasse aan, erf 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) { }
}
```

H

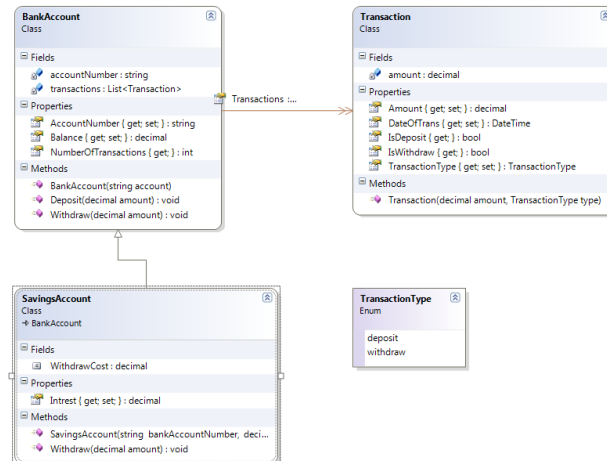
61

8. Klassen

- De banking solution : ontwerp
- Aanmaken van de solution Banking
- Aanmaken van een klasse
- Members van een klasse
 - Fields
 - Methods
 - Constructor
 - Destructor
 - Properties
 - Region
- Aanmaken members van een klasse
- Gebruiken van een klasse
- Class View/Object Browser

8. Klassen

- De banking solution : het ontwerp van de domein laag

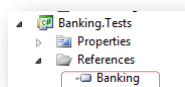
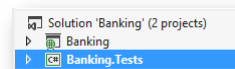


HoGent

Pag. 63

8. Klassen

- Aanmaken van de solution Banking
 - File > New Project > C# > Web > ASP.NET MVC4 Application
 - Geef als naam "Banking" in en kies een locatie. Klik OK
 - Kies een "Internet application", "Razor" als View Engine EN vink "Create a unit test project" aan (name = Banking.Tests)
 - Visual Studio creëert solution "Banking" met 2 projecten. Open de Solution Explorer
 - MVC project "Banking"
 - Banking wordt de namespace!
 - Alle klassen die we aanmaken zullen tot deze namespace behoren
 - Compileert naar assembly banking.dll
 - Het Unit test project "Banking.Tests"
 - Bevat een Reference naar het Banking project (banking.dll in bin folder)



HoGent

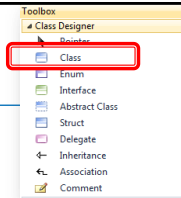
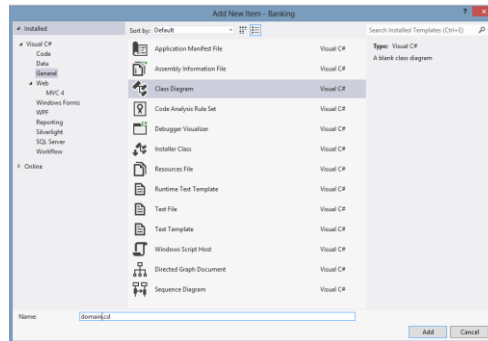
Pag. 64

8. Klassen

► Aanmaken van een klasse

1. Klasse ontwerpen a.d.h.v. een **klassen diagram**

- Voeg onder de folder Models van Banking project een class diagram toe. Rechtsklik op Models > Add > New Item ... > Visual C# > General > **Class diagram** > Noem dit Domain.cd

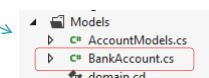
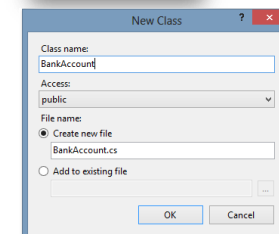
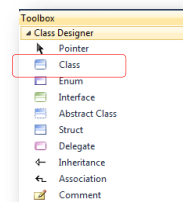
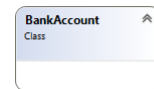


8. Klassen

► Aanmaken van een klasse

1. Klasse ontwerpen a.d.h.v. een **klassen diagram**

- Ga naar Toolbox (View > Toolbox) en sleep "Class" op editor.
- Visual Studio opent popup venster. Geef class name BankAccount in.
- De bijhorende .cs file wordt automatisch gegenereerd (omgekeerd zijn aanpassing in de code ook direct in design zichtbaar)
- VS maakt de file BankAccount.cs aan in de root van het project. Versleep dit bestand naar de Models folder.

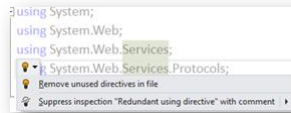


8. Klassen

► Aanmaken van een klasse

1. Klasse ontwerpen a.d.h.v. een **klassen diagram**

- Dubbelklik BankAccount.cs in Solution explorer
 - using statements : de gebruikte assemblies
 - Maak je de klasse aan via een class diagram dan staan er niet gebruikte using statements. Ga er over met de muis, het Resharper lampje verschijnt en klik Remove unused directives in file



- Namespaces = Logische groepering van gerelateerde klassen (packages in Java). Meerdere klassen kunnen tot dezelfde namespace behoren

namespace Banking.Models

- Pas de namespace aan in Banking.Models

```
{  
    public class BankAccount  
    {  
    }  
}
```

HoGent

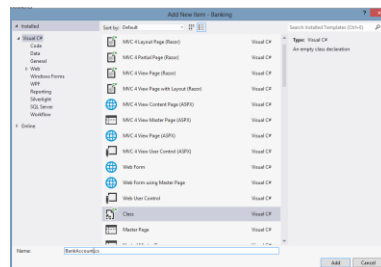
8. Klassen

► Aanmaken van een klasse

1. Klasse ontwerpen a.d.h.v. een **klassen diagram**

2. Of rechtstreek via code

- Rechtsklik op de folder Models > Add > New Item > Visual C# > Class. Geef de klasse de naam BankAccount
- Je kan dan achteraf een class diagram aanmaken en deze file hierop droppen



HoGent

8. Klassen

► Members van een klasse

- Fields (Attributen)
- Constructor – destructor
- Properties
- Methods
- Events

- Access modifiers
 - public : toegankelijk binnen en buiten de klasse.
 - private : enkel toegankelijk binnen de klasse.
 - internal : toegankelijk binnen assembly.
 - protected : toegankelijk voor klassen die erven van deze klasse

8. Klassen : Members

1. Fields (Attributen)

[modifier] datatype variablename

- Kapselen data in
- Kunnen variabelen zijn of constanten
- Enkel private. Access gebeurt door (public) properties. Zie verder.
- Static fields: zijn gekoppeld aan de klasse en niet aan een instantie (object) van de klasse. Ze bestaan 1 maal per klasse.
- Namingconventie : camelCase (beginnen met kleine letter)

```
public class BankAccount
{
    private string accountNumber;
    private decimal balance;
}
```

8. Klassen : Members

2. Methods

```
[modifier] return_type MethodeName ([parameters]) { ...}
```

- Operaties die een object kan uitvoeren.
- Kunnen al dan niet (void) een waarde retourneren.
- Kunnen static gedeclareerd worden.
- Bevatten parameter lijst : parameters gescheiden door een komma, parameters hebben type en naam, gebruik () indien geen parameters.
- Je kan meerdere methodes hebben met dezelfde naam. Ze verschillen in aantal argumenten en/of type van argumenten method overloading
- Naming conventions : beginnen met een hoofdletter!!!

```
public class BankAccount {  
    public void Deposit(decimal amount) {  
        throw new NotImplementedException();  
    }  
}
```

8. Klassen : Members

2. Methodes (vervolg)

- Je kan ook return type opgeven

```
public decimal GetBalance()  
{  
    return balance;  
}
```

- return statement
 - Kan om het even waar staan in de code van de methode en kan meerdere malen voorkomen
 - Retourneert de waarde van de methode
 - Uitvoering methode wordt onmiddellijk gestopt (eventueel na uitvoering finally bij exception handling of Dispose bij using) en de controle wordt teruggegeven aan oproepend programma.

8. Klassen : Members

2. Methodes (vervolg)

- 3 type parameters
 - Value parameters : input parameter
 - Ref parameters : input/output parameters
 - Output parameters : output parameter

```
public void Test (int x) {x+=1;}  
public void Test(ref int x) {x+=1;}  
public void Test(out int x) {x = 1;}
```

- Je kan ook een defaultwaarde meegeven aan 1 of meerdere parameters (startend bij de laatste parameter in de rij)

```
public void Test (int x =1) {x+=1;}
```

- Kan je oproepen :

```
Test();  
Test(5);
```

8. Klassen : Members

3. Constructor

- Een constructor heeft steeds dezelfde naam als de klasse, en heeft nooit een return type.
- Een klasse hoeft geen constructor te hebben. In dat geval maakt de compiler zelf een default constructor (public naamklasse()) aan.
- Een klasse kan 1 of meerdere constructors hebben. Ze verschillen in aantal argumenten en/of type van argumenten. In dat geval hoeft de klasse geen default constructor te hebben en maakt de compiler ook geen default constructor aan.

```
public class BankAccount { ...  
    public BankAccount(string accountNumber) {  
        AccountNumber = accountNumber;  
        Balance = 0;  
    }  
    public BankAccount(string accountNumber, decimal balance) : this(accountNumber) {  
        Balance = balance;  
    }  
}
```

Snippet : typ ctor + tab =>
genereert constructor
methode

8. Klassen : Members

4. Destructor

- Kuisen objecten op en worden automatisch uitgevoerd voor de garbage collector een object vrij geeft.
- Hebben geen access modifier – geen parameters en hebben dezelfde naam als de klasse met een tilde voorafgegaan.
- Worden zelden expliciet geschreven. Je weet ook niet wanneer ze worden uitgevoerd. Beter om IDisposable te gebruiken.

```
public class BankAccount {  
    ....  
    ~BankAccount() {  
        //implementatie  
    }  
}
```

8. Klassen : Members

5. Properties (de getter en setter)

◦ Java

```
public class BankAccount {  
    private String accountNumber;  
    public String getAccountNumber()  
    {  
        return accountNumber;  
    }  
    public void setAccountNumber(String value)  
    {  
        accountNumber = value;  
    }  
}
```

```
BankAccount account = new BankAccount("...");  
String nbr = account.getAccountNumber();  
account.setAccountNumber("...");
```

• C#

```
public class BankAccount {  
    private string accountNumber;  
    public string AccountNumber  
    {  
        get { return accountNumber; }  
        set { accountNumber = value; }  
    }  
}
```

Compiler weet wanneer getter of setter gebruikt wordt (bij =, links(set) of rechts(get))

```
BankAccount account = new BankAccount("...");  
string nbr = account.AccountNumber;  
account.AccountNumber="...";
```

8. Klassen : Members

5. Properties

```
public return-type name{ get{return ...;} set{ ... = value;}}
```

- Gecontroleerde toegang tot de attributen => getter/setter
- Naming conventie : Property heeft dezelfde naam als attribuut, maar begint met hoofdletter
- Kan get en set bevatten, of enkel get of enkel set

```
private decimal balance;  
public decimal Balance {  
    get  
    {  
        return this.balance;  
    }  
    set  
    {  
        this.balance = value;  
    }  
}
```

value = is de naam van de variabele die je bij een setter zou meegeven

Of beter, daar gebruiker niet rechtstreeks de balance mag wijzigen

```
private decimal balance;  
public decimal Balance {  
    get  
    {  
        return this.balance;  
    }  
}
```

Pag. 77

8. Klassen : Members

5. Properties

- Als property public dan ook getter en setter, tenzij je dit expliciet vermeldt

```
public decimal Balance {  
    get  
    {  
        return this.balance;  
    }  
    private set  
    {  
        this.balance = value;  
    }  
}
```

- Tip : om vertrekkend van een attribuut een property te genereren. Selecteer attribuut > rechtsklik > Refactor > Encapsulate Field

8. Klassen : Members

5. Properties:

- **AUTOMATIC PROPERTY** = verkorte schrijfwijze
 - In dit geval maak je **geen** attribuut aan
 - Je schrijft de property als volgt

```
public decimal Balance { get; set; }
```

- Als de compiler een empty get/set property implementatie tegenkomt zal hij automatisch een private attribuut genereren en de getter en setter implementeren.
- Indien buitenwereld setter niet mag oproepen, maak setter private

```
public decimal Balance { get; private set; }
```

- Tip : om snel automatic properties te creëren in code venster : typ prop en druk 2 * tab toets in (= code snippet) en pas datatype en naam aan

8. Klassen : Members

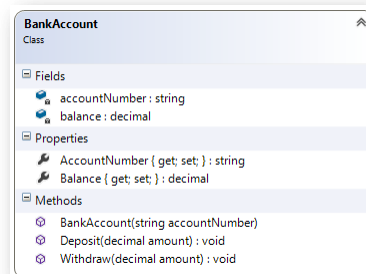
6. Regions

- Dienen om code te groeperen
- Een region kan je open- en dichtklappen
- Aanmaken : selecteer een stukje code selecteren > Rechtsklik > Surround with > Visual C# > #region of typ de code in.
- Voorzie in een klasse minstens 4 regions : Fields, Constructors, Methods, Properties

```
7 namespace Banking.Domain
8 {
9     public class BankAccount
10     {
11         #region Fields
12         private string bankAccountNumber;
13         private decimal balance = Decimal.Zero;
14         #endregion
15
16         Constructors
17
18         Methods
19
20         Properties
21     }
22 }
```


8. Klassen

- ▶ Aanmaken members van een klasse
 - De fields, methods, ... van de klasse kan je
 - rechtstreeks coderen
 - of je kan ontwerp vervolledigen in Class Diagram.



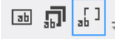
8. Klassen

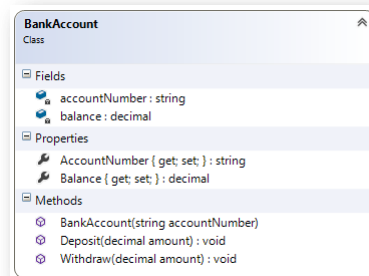
- ▶ Aanmaken members van een klasse
 - Rechtsklik op klasse BankAccount in class diagram > Class Details. (of menu View > Other Windows > Class Detail)
 - Geef onderstaande in (volgende slides bevatten uitleg)
 - Via design tool kan je geen automatic props aanmaken

Class Details			
	Name	Type	Modifier
Methods			
+	BankAccount		public
	{ accountNumber	string	None
	} <add parameter>		
+	Deposit	void	public
	{ amount	decimal	None
	} <add parameter>		
+	Withdraw	void	public
	{ amount	decimal	None
	} <add parameter>		
	<add method>		
Properties			
	AccountNumber	string	public
	Balance	decimal	public
	<add property>		
Fields			
	accountNumber	string	private
	balance	decimal	private

8. Klassen

► Aanmaken members van een klasse

- Je krijgt onderstaand klassendiagram
 - Klik bovenaan in toolbar  ;
 - Laatste icoontje : toont signatuur
- attribuut of property omzetten naar een associatie => Selecteer attribuut/property > rechtsklik > Show as (Collection) Association



8. Klassen

► Aanmaken members van een klasse

- Bekijk dan BankAccount.cs
 - Methodes bevatten throwen NotImplementedException
 - Maak setter van AccountNumber en Balance private
 - Voeg regions toe
- Hierna volgt theorie rond gebruik van klassen.
- Daarna TDD : de manier om de klasse te implementeren

```
namespace Banking
{
    public class BankAccount
    {
        #region Fields
        private string accountNumber;
        private decimal balance;
        #endregion

        Constructors

        #region Properties
        public string AccountNumber
        {
            get
            {
                throw new NotImplementedException();
            }
            private set
            {
            }
        }

        public decimal Balance
        {
```

8. Klassen

► Gebruik van klassen

- Declaratie van een variabele van het type BankAccount

```
BankAccount myAccount;
```

- Creatie van een instantie van BankAccount

```
myAccount = new BankAccount("123...");
```

- Creatie en instantiatie mag je in 1 statement doen

```
BankAccount myAccount = new BankAccount("123...");
```

- indien ook een default constructor zou bestaan

```
BankAccount myAccount = new BankAccount();
```

8. Klassen

► Gebruik van klassen

- **Object Initializers** : Combinatie van object instantiatie en initialisatie in 1 stap

```
public class BankAccount{  
    private string accountNumber;  
    public Account(string accountNumber) {...}  
    public decimal Balance { get; set; } }
```

- Verkorte schrijfwijze voor :

```
BankAccount myAccount = new BankAccount("123-123");  
myAccount.Balance = 100;
```

- Is :

```
BankAccount myAccount = new BankAccount("123-123") { Balance = 100 };
```

Sequentie van member initializers, tussen { } en gescheiden door een komma. Compiler zal dit vervangen door bovenstaande versie.

8. Klassen

► Gebruik van klassen

◦ Uitvoeren van een methode

- void

```
myAccount.Deposit(100.0);
```

- Met return waarde

```
string balance = myAccount.GetBalance().ToString();
```

◦ Uitvoeren properties : De plaats in de code bepaalt of de getter, dan wel de setter wordt uitgevoerd.

- Uitvoeren getter (opvragen inhoud)

```
string balance = myAccount.Balance.ToString();
```

- Uitvoeren setter (instellen inhoud)

```
myAccount.Balance = 100;
```

- Beide (uitvoeren getter en dan setter)

```
myAccount.Balance += 100;
```

8. Klassen

► Class View/Object Browser

- In menu kies View > Class View/Object Browser
- Class View : Geeft de klasse structuur van je project weer, de namespaces en de klassen in de vorm van een boomstructuur. Bekijken van types in huidig project
- Object Browser : ook inspecteren van referenced assemblies

***“Extraordinary products
are merely side effects of
good habits.”***



TESTEN



9. Unit testen

- ▶ Test Driven Development
- ▶ Aanmaken van test bibliotheek
- ▶ Aanmaken van unit test
 - Stappenplan
 - De 3 AAA's
 - Klasse Assert
 - Testen op exceptions
- ▶ Aanmaken van unit test voor constructor BankAccount
- ▶ Aanvullingen
- ▶ Tips
- ▶ Test List



Unit Testing makes your
developer lives easier

EASIER to

-find bugs

-maintain

-understand

-develop

9. Unit testen

► TDD – Motto : **Rood**, **Groen**, **Refactor**

- Doe het **Falen**
 - Geen code zonder falende test
- Doe het **Werken**
 - Zo eenvoudig mogelijk
- Maak het **Beter**
 - Refactor

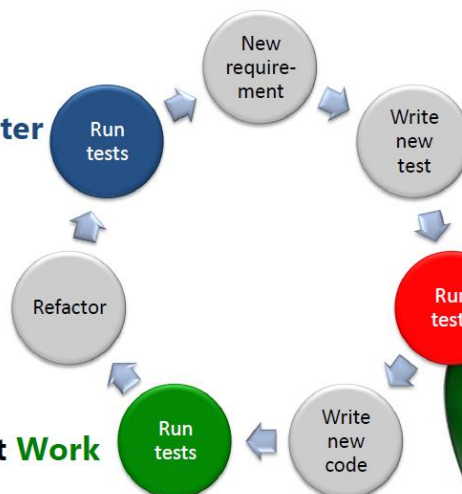


HoGent

Pag. 93

TDD cycle

Make it **Better**



Make it **Fail**

Make it **Work**



Confidence boost

- Isn't the green bar a feel good factor?



Confidence boost

Doordat er alleen functionaliteit geschreven wordt die ervoor zorgen dat de testen slagen ben je er zeker van dat er nooit code geschreven wordt die niet in de rest van de applicatie gebruikt wordt.

Je bent er ook zeker van dat alle functionaliteit getest werd.

Hierdoor kan je eenvoudig controleren of alles nog werkt nadat je wijzigingen hebt aangebracht aan je code



9. Unit testen

► De test bibliotheek

- Banking.Tests
- Ontwerp van de testklasse volgens TDD
 - Je schrijft testen VOOR je een klasse implementeert
 - Je kan echter vanuit deze testklasse de methodes van de klasse niet oproepen als deze methodes niet bestaan
 - Je maakt je klasse aan met een minimum aan code
 - Maak de klasse waarvoor je testen gaat schrijven aan
 - Voeg de methodes toe, maar zonder functionele code. Werp een NotImplementedException

9. Unit testen

► Aanmaken van unit testen

- Stappenplan
 1. Maak een ontwerp van de klasse => class diagram
 - Methodes throwen NotImplementedException
 2. Maak een testklasse
 1. In Banking.Tests, voeg folder Models toe (Rechtsklik Banking.Tests > Add > New Folder)
 2. Rechtsklik op Models folder > Add > New Item > Test > Kies een basic unit test en noem deze BankingAccountTest
 3. Schrijf de testen in BankAccountTest.cs
 4. Run de testen. Testen falen
 5. Pas de code in de klasse aan
 6. Run de testen opnieuw. Testen slagen
 7. Refactor indien nodig

9 Unit testen

good
✓

How to write Unit Tests?

The 3A Pattern

- Arrange
- Act
- Assert

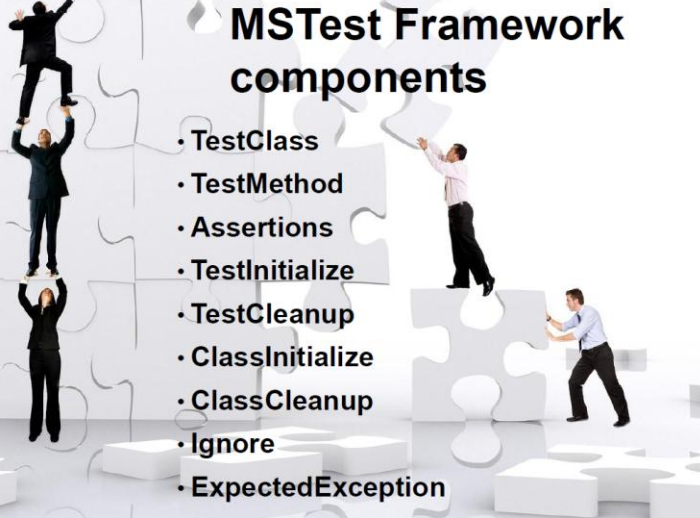


9. Unit testen

- ▶ Aanmaken van unit testen
 - **AAA** : de normale flow in een test methode
 - **A**rrange : Initialisatie : Maak een object van de te testen klasse, initialiseer variabelen,...
 - **A**ct : Roep de te testen methode op
 - **A**ssert : Controleer of de methode correct is uitgevoerd
 - Welke testen aanmaken : wees creatief!!!
 - Baseer je op use case, maar denk verder. Bedenk alternatieven.
 - Test zeker één normaal geval, maar vooral alle mogelijk foute gevallen en grensgevallen (bvb i.g.v. parameters : alle mogelijke inputwaarden voor parameter)
 - Geef betekenisvolle namen aan de test methodes
 - Zie cursus Analyse en Ontwerpen I

9. Unit testen

► Aanmaken van unit testen



MSTest Framework components

- TestClass
- TestMethod
- Assertions
- TestInitialize
- TestCleanup
- ClassInitialize
- ClassCleanup
- Ignore
- ExpectedException

HoGent

Pag. 101

9. Unit testen

► Aanmaken van unit testen

- De klasse Assert
 - Assert.AreEqual(expected, actual)
 - Test of expected gelijk is aan actual.
 - Alle primitieve datatypes
 - Voor vergelijken van objecten dien je de **Equals** methode te implementeren
 - Kan ook met extra parameter voor tonen boodschap (geldt ook voor volgende methodes), maar is overbodig als de namen van de testmethodes goed gekozen zijn!
 - Assert.IsTrue(bool conditie), Assert.IsFalse(bool conditie)
 - Test of conditie gelijk is aan true/false

9. Unit testen

- ▶ Aanmaken van unit testen
 - De klasse Assert
 - Assert.Is(Not)Null(actual)
 - Test of actual (niet) gelijk is aan null.
 - Assert.IsInstanceOfType(actual, typeof(classname))
 - Test of actual instantie is van de betreffende klasse
 - Testen op Exceptions
 - Als je foutieve parameterwaarden meegeeft aan een test methode moet deze methode een exception throwen.
 - Je voorziet de test methode van een extra attribuut [ExpectedException(typeof(ArgumentException))]
 - Als je test methode geen exception werpt krijg je een Unit Failure

9. Unit testen

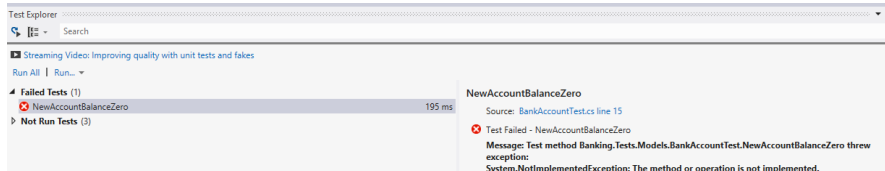
- ▶ Aanmaken van unit test voor constructor BankAccount
 - Open BankAccountTest.cs
 - Voeg statement using Banking.Models toe
 - Maak test voor de constructor : balance=0 voor nieuwe BA


```
[TestMethod]
public void NewAccountBalanceZero()
{
    // Arrange
    string accountNumber = "123-4567890-02";
    // Act
    BankAccount account = new BankAccount(accountNumber);
    //Assert
    Assert.AreEqual(0, account.Balance);
}
```

- ▶
 - Run test. Rechtsklik > Run tests of via Menu > Test. Test faalt.

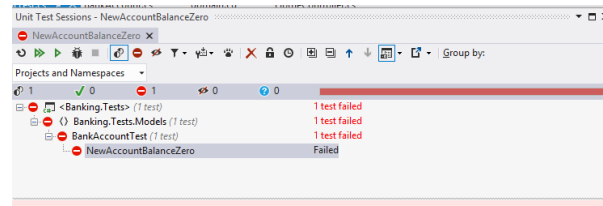
9. Unit testen

- ▶ Aanmaken van unit test voor constructor BankAccount
 - Run test. Rechtsklik > Run tests of via Menu > Test. Test faalt.



- Ook mogelijk via Resharper : rechtsklik  > Run (biedt meer mogelijkheden)

HoGent




Pag. 105

9. Unit testen

- ▶ Aanmaken van unit test voor constructor BankAccount
 - Een 2de test : Rekeningnummer moet overeenkomstig rekeningnummer zijn

```
[TestMethod()]
public void NewAccountMustSetAccountNumber()
{
    string accountNumber = "123-4567890-02";
    BankAccount account = new BankAccount(accountNumber);
    Assert.AreEqual(accountNumber, account.AccountNumber);
}
```

- Run test. Rechtsklik > Run tests of via Menu > Test. Test faalt.
- Ook mogelijk via Resharper : rechtsklik  > Run (biedt meer mogelijkheden)

HoGent

Pag. 106

9. Unit testen

► Aanmaken van unit test voor constructor BankAccount

- Ga naar BankAccount.cs.
 - Implementeer de constructor.

```
public BankAccount(string accountNumber)
{
    AccountNumber = accountNumber;
    Balance = 0;
}
```

- Run de testen opnieuw. Ze falen nog. Nog de properties implementeren

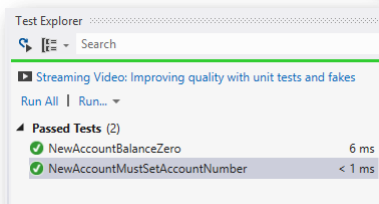
9. Unit testen

► Aanmaken van unit test voor constructor BankAccount

- Implementeer de properties
 - Balance : automatic property
 - AccountNumber : attr + prop : daar we in setter validatie zullen toevoegen

- Run de test opnieuw. Ze slagen

```
public decimal Balance {get; private set;}
private string accountNumber;
public string AccountNumber
{
    get
    {
        return accountNumber;
    }
    private set
    {
        accountNumber = value;
    }
}
```



9. Unit testen

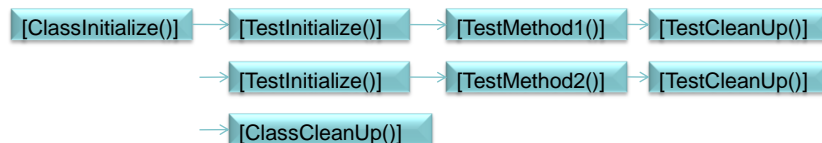
► Aanvullingen :

- Testfixture : object of resource die je nodig hebt in meerdere test methodes
 - Declareer private variabele voor testfixture in testklasse
 - Initialiseer deze in methode **[TestInitialize()]**
 - VS voert deze methode uit VOOR de uitvoering van iedere testmethode
 - Kuis de variabele op in de methode **[TestCleanup()]**
 - VS voert deze methode uit na iedere test methode-uitvoering

9. Unit testen

► Aanvullingen

- [ClassInitialize()] en [ClassCleanup()]
 - De methode [ClassInitialize()] wordt 1 maal uitgevoerd, voor alle test methodes en voor de [TestInitialize()] methode
 - Gebruik : initialisatie van testfixtures die de test methodes niet wijzigen en veel tijd vragen om te initialiseren.
 - Een methode [ClassCleanup()] wordt 1 maal uitgevoerd, na alle test methodes en na de [TestCleanup()]



9. Unit testen

► Aanvullingen

- Unit test die gebruik maakt van Testfixture

```
private BankAccount account;  
private string accountNumber;  
  
[TestInitialize]  
public void MyTestInitialize()  
{  
    accountNumber = "123-4567890-02";  
    account = new BankAccount(accountNumber);  
}  
  
[TestMethod]  
public void NewAccountBalanceZero()  
{  
    //Assert  
    Assert.AreEqual(0, account.Balance);  
}
```

9. Unit testen

► Maak nog deze unit testen aan voor de BankAccount constructor

Naam test	Rekeningnummer	Gevolg
NewAccountEmptyStringFails	string.Empty	ArgumentException
NewAccountNullFails	Null	ArgumentNullException
NewAccountTooLongFails	"123-4567890-0333"	ArgumentException
NewAccountWrongFormatFails	"063-1547563@60 "	ArgumentException
NewAccountDivisionBy97Fails	"123-4567890-03"	ArgumentException

- Of voeg uit de BankingStarter folder de testklasse BankAccountTestDeel2 toe
 - In Models folder in Banking.Tests > Add > Existing item en browse naar BankAccountTestDeel2.cs

9. Unit testen

► Implementeer

- zie help voor Regex, Match

```
public string AccountNumber
{
    get { return accountNumber; }
    set
    {
        if (value == null)
            throw new ArgumentNullException("Bankaccount number may not be null", (Exception)null);
        Regex regex = new Regex(@"\d{3}-\d{7}-\d{2}");
        Match match = regex.Match(value);
        if (!match.Success)
            throw new ArgumentException("Bankaccount number format is not correct");
        if (int.Parse(match.Groups[1].ToString() + match.Groups[2].ToString()) % 97 !=
            int.Parse(match.Groups[3].ToString()))
            throw new ArgumentException("97 test of the bankaccount number failed");
        this.accountNumber = value;
    }
}
```

HoGent

Pag. 113

9. Unit testen

► De testen voor Deposit/Withdraw(zie BankAccountTestDeel2.cs)

- WithdrawMustChangeBalance
 - Arrange: nieuwe bankrekening met geldig nummer
 - Act: Deposit 200, Withdraw 100
 - Assert: Balance = 100
- DepositMustChangeBalance
 - Arrange: nieuwe bankrekening met geldig nummer
 - Act: Deposit 100
 - Assert: Balance = 100
- WithdrawNegativeBalanceAllowedOnBankAccounts
 - Arrange: nieuwe bankrekening met geldig nummer
 - Act: Withdraw(100)
 - Assert: Balance = -100
- WithdrawNegativeAmountNotAllowed(). Idem voor Deposit

HoGent

Pag. 114

9. Unit testen

► Implementeer Withdraw/Deposit (klasse BankAccount)

```
#region Methods
public void Withdraw(decimal amount)
{
    if (amount < 0)
        throw new ArgumentException("Bedrag dat gestort wordt moet groter zijn dan 0");
    Balance -= amount;
}

public void Deposit(decimal amount)
{
    if (amount < 0)
        throw new ArgumentException("Bedrag dat afgehaald wordt moet groter zijn dan 0");
    Balance += amount;
}
#endregion
```

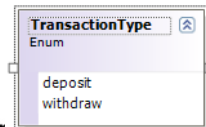
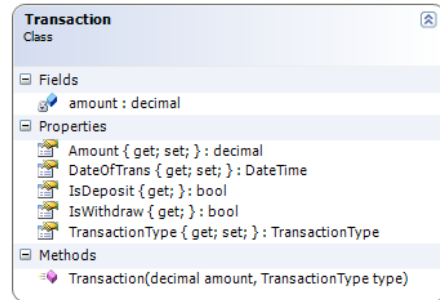
9. Unit testen

► Tips

- 1 methode test 1 ding. Beperk aantal Asserts. Ideaal 1 Assert/test methode
- Nadelen van veel Asserts in 1 methode
 - Als 1 Assert binnen test methode faalt, voert VS de rest methode niet uit
 - De methode wordt moeilijk te lezen
 - De kans dat je in de methode een bug schrijft wordt groter
 - De kans dat je de methode moet debuggen wordt groter
- De werking van 1 test methode mag niet afhangen van de werking van een andere testmethode
- Testmethodes moeten in willekeurige volgorde kunnen uitvoeren.
- Schrijf geen testen voor bestaande libraries die je gebruikt of voor gegenereerde code (get/set)

10. Associaties

- Klasse Transaction
 - Bijhouden van verrichtingen

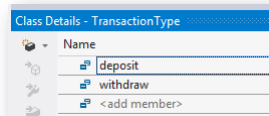


HoGer...

Pag. 117

10. Associaties

- Maak de enum class aan in het klassen diagram
 - Toolbox > sleep enum naar designer. Geef als name TransactionType in
 - Ga naar Class Details



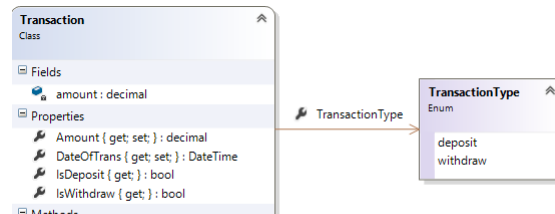
- Versleep Transaction.cs van root Banking naar Models folder
- Pas namespace aan in Banking.Models
- Verwijder unused using statements

HoGent

Pag. 118

10. Associaties

- ▶ Maak de klasse Transaction aan
 - De klasse is reeds aangemaakt en is onderdeel van de Banking Starter applicatie (zie chamilo)
 - Rechtsklik op de Models Folder > Add > Existing item en browse naar Transaction.cs in BankingStarter folder
 - Ga naar het class diagram en sleep transaction.cs op het class diagram
 - Rechtsklik op TransactionType in Class Diagram > Show as association



HoGent

Pag. 119

10. Associaties

- ▶ Implementatie

```
public class Transaction
{
    #region Attributes and Properties
    private decimal amount;
    public decimal Amount
    {
        get { return amount; }
        private set
        {
            if (value < 0)
                throw new ArgumentException("amount moet groter zijn dan 0");
            amount = value;
        }
    }

    public DateTime DateOfTrans {get; private set;}
    public TransactionType TransactionType {get; private set;}
    public bool IsWithdraw
    {
        get
        {
            return TransactionType == TransactionType.withdraw;
        }
    }
}
```

HoGent

Pag. 120

10. Associaties

► Implementatie (vervolg)

```
public bool IsDeposit
{
    get
    {
        return TransactionType == TransactionType.deposit;
    }
}

#endregion
#region Constructors
public Transaction(decimal amount, TransactionType type)
{
    Amount = amount;
    TransactionType = type;
    DateOfTrans = DateTime.Now;
}
#endregion
}
```

10. Associaties

► Klasse BankAccount

- In de klasse BankAccount voorzien we een generische lijst van transacties. Meer info ivm IList<T> en List<T> : zie Help
 - Voeg toe in Class Details of rechtstreeks in code

```
using System.Collections.Generic;
private IList<Transaction> transactions;
```

- Op class diagram kan je veld veranderen in associatie pijl > rechtsklik op veld > Show as (Collection) Association



- Voeg extra properties toe

```
public IEnumerable<Transaction> Transactions {...}
public int NumberOfTransactions {...}
```

10. Associaties

► Aanpassingen aan klasse BankAccount

```
public class BankAccount
{
    private string accountNumber;
    private IList<Transaction> transactions;

    public BankAccount(string accountNumber) { ...
        transactions = new List<Transaction>();
    }

    public void Deposit(decimal amount) { ...
        transactions.Add(new Transaction(amount, TransactionType.deposit));
    }

    public virtual void Withdraw(decimal amount) { ...
        transactions.Add(new Transaction(amount, TransactionType.withdraw));
    }
    ...
    public int NumberOfTransactions
    { get { return transactions.Count; } }

    public IEnumerable<Transaction> Transactions
    { get { return transactions; } }
```

10. Associaties

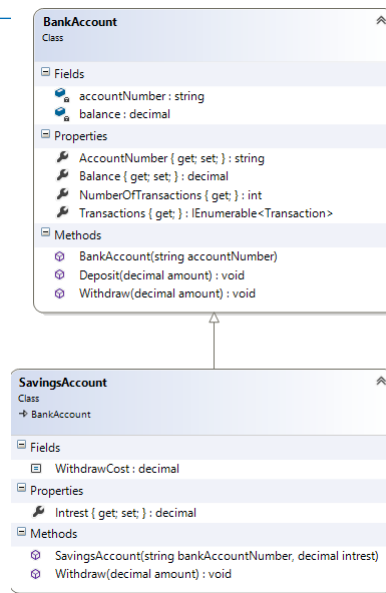
- Maak de bijhorende unit testen
 - Voeg nu ook TransactionTest.cs uit BankingStarter toe aan Models folder in Banking.Tests
 - Run de testen in *TransactionTest.cs*

11. Overerving

- ▶ Overerving is een mechanisme waarbij software opnieuw wordt gebruikt: nieuwe klassen worden gecreëerd vertrekkende van bestaande klassen
 - De superklasse bevat de gemeenschappelijke attributen, operaties en associaties
 - De subklasse erft alles van de superklasse : attributen, operaties, associaties
 - In een subklasse wordt het gedrag van de superklasse uitgebreid en/of gespecialiseerd
 - Het gedrag van de subklasse kan verder gespecialiseerd worden door methoden van de superklasse te overriden (herdefiniëren) in de subklasse.
 - De subklasse heeft een 'is een' relatie met de superklasse

11. Overerving

- Voeg SavingAccount.cs uit BankingStarter toe aan de Models folder in project Banking
- Drop op class diagram
- Voeg ook SavingAccountTest.cs toe aan Models folder Banking.Tests



11. Overerving

► Definitie superklasse

```
public class BankAccount
```

- Opmerking : Indien van een klasse niet mag worden afgeleid

```
public sealed class BankAccount
```

11. Overerving

► Definitie subklasse

```
public class SavingAccount : BankAccount
```

- Bij een subklasse hoort steeds één superklasse. .Net laat geen multiple inheritance toe. Een klasse kan wel meerdere interfaces implementeren
- Enkel de private members (attributen/methoden) van de superklasse zijn niet toegankelijk vanuit de subklasse.

11. Overerving

► Constructors

- Constructoren van de superklasse worden niet overgeërfd door de subklassen.
- Als de subklasse geen constructor bevat, maakt de compiler zelf een default constructor aan, die automatisch de constructor van de superklasse oproept.
 - Als de superklasse dan geen default constructor bevat krijg je een compiler fout
- Keyword “base” : aanroepen methode/constructor uit superklasse

11. Overerving

◦ Voorbeeld

```
public class SavingsAccount : BankAccount
{
    public const decimal WithdrawCost = 0.25M;

    public decimal Intrest {get; private set;}

    public SavingsAccount(string bankAccountNumber, decimal intrest) :
        base(bankAccountNumber)
    {
        Intrest = intrest;
    }
    ...
}
```

Erft van de superklasse BankAccount

base : aanroepen methode uit superklasse voor init members van superklasse

Je kan ook constructor van eigen class oproepen : this(bankaccountNumber, intrest)

11. Overerving

► Methodes : nieuwe methode toevoegen

```
public class SavingAccount : BankAccount
{
    public const decimal WithdrawCost = 0.25M;

    public decimal Intrest {get; private set;}

    public SavingsAccount(string bankAccountNumber, decimal intrest) : base(bankAccountNumber)
    {
        Intrest = intrest;
    }

    public void AddInterest()
    {
        Deposit(Balance * Intrest);
    }
}
```

Diagram illustrating the addition of new methods to a class hierarchy:

- Extra attribuut**: Points to the `WithdrawCost` attribute.
- Extra property**: Points to the `Intrest` property.
- Extra methode**: Points to the `AddInterest` method.

11. Overerving

► Methodes (vervolg)

- Methode die niet overschreven mag worden in subklasse
 - Dit is de standaard

```
public void Deposit(decimal amount)
{
    Balance += amount;
}
```

11. Overerving

► Methodes (vervolg)

- Methode overriding
 - Een methode uit de superklasse die je ook in subklasse uitwerkt
 - Als je de methode uitvoert op een object van de subklasse, voert .NET de versie van de subklasse uit

```
public class BankAccount {  
    public virtual void Withdraw(decimal amount) {  
        Balance += amount;  
    }  
}  
  
public class SavingAccount : BankAccount {  
    public override void Withdraw(decimal amount) {  
        base.Withdraw(amount);  
        base.Withdraw(WithdrawCost); //afhalen kost de klant geld  
    }  
}
```

Pas dit aan in klasse BankAccount

Duidt aan dat je de methode een andere implementatie kan geven in de subklasse

De nieuwe implementatie

11. Overerving

► Methode ToString() en Equals()


- Elke klasse is afgeleid van System.Object
- Deze bevat overridable methode ToString(), die geeft als standaardgedrag de naam van de klasse weer.
- En overridable Equals(), standaardgedrag : 2 variabelen zijn gelijk als ze wijzen naar hetzelfde object)
- Je kan een eigen versie aanmaken

```
public override string ToString()  
{  
    return String.Format("{0} - {1}", AccountNumber, Balance);  
}  
  
public override bool Equals(object obj)  
{  
    if (obj == null || !(obj is BankAccount))  
        return false;  
    return (this.AccountNumber == ((BankAccount)obj).AccountNumber);  
}
```

11. Overerving

- ▶ Instantie aanmaken van de subklasse

```
SavingAccount s;  
s = new SavingAccount("12-455665-12", 0.10M);  
s.WithDraw(20M);  
s.Deposit(10M);
```



Methode uit de subklasse

- Zie testklasse.
- Run de testen

12. Polymorfisme

- ▶ Polymorfisme kan optreden als men 2 of meer klassen laat overerven van één klasse. Zo kan men objecten van de subtypen opslaan in een collection die bestaat uit objecten van de superklasse.
- ▶ Er kan ook op een polymorfe manier een object methode aangeroepen worden. Hier wordt dan aan de hand van het type overervende klasse gekozen welke methode er moet worden uitgevoerd. Voorwaarde : de methode moet gedefinieerd zijn in de superklasse.
- ▶ Het type van het object bepalen :

```
BankAccount s = new SavingAccount("12-455665-12", 0.10M);  
if (s is SavingAccount) {...}
```

12. Polymorfisme

- Voorbeeld

```
BankAccount[] accounts = new BankAccount[3];
accounts[0] = new BankAccount("12-455665-12");
accounts[1] = new SavingAccount("12-455665-13", 0.05M);
accounts[2] = new SavingAccount("12-455665-14", 0.03M);
foreach (BankAccount a in accounts)
{
    a.WithDraw(10M);
}
```

13. Abstracte klasse

- ▶ Een abstracte klasse is een klasse waarvan geen instanties kunnen worden aangemaakt. U moet afleiden van deze klasse om ze te kunnen gebruiken.
- ▶ Bevat keyword “abstract”.
- ▶ Niet alle methodes moeten geïmplementeerd zijn. Bevat ook abstracte methodes (methode zonder body, met keyword abstract).

```
public abstract class BankAccount {
    public virtual void WithDraw(decimal amount) {.....}
    public abstract string PrintAccount();
}
```

14. Interface

- ▶ Is een klasse met methodes, properties en events maar zonder implementatie ervan
- ▶ De implementatie schrijft u in klassen die de interface implementeren. Een klasse kan meerdere interfaces implementeren

```
public interface IAccount {  
    void Withdraw(decimal amount);  
}  
  
public class SavingAccount : IAccount {  
    public void Withdraw(decimal amount) { .....}  
}
```

15. Statische members

- ▶ Informatie eigen aan de klasse, maar niet aan een bepaalde instantie van die klasse.
- ▶ Gebruik keyword static

```
double result;  
result = Math.Cos(45);
```

```
class SavingsAccount {  
    public static double currInterestRate = 0.04;  
    public static void SetInterestRate(double newRate )  
    { this.currInterestRate = newRate; }  
    public static double GetInterestRate()  
    { return this.currInterestRate; }  
    ....}
```

```
double ir = SavingsAccount.GetInterestRate();
```

16. Generic class

```
public class MagischeHoed<T>
{
    private IList<T> dingen = new List<T>();
    public void Add(T ding)
    {
        dingen.Add(ding);
    }
}
```

```
Konijn roger = new Konijn("Roger");
MagischeHoed<Konijn> hoed = new MagischeHoed<Konijn>();
hoed.Add(roger);
```

16. Generic class

- ▶ Je kan constraint toevoegen aan T
 - Je kan eisen dat T een class of struct is
 - Je kan eisen dat T een public default constructor heeft
 - Nodig als je ergens in een methode de default constructor aanroept, anders krijg je compilatiefout

```
public class MagischeHoed<T> where T : class, new()
```

- Je kan eisen dat T erft van een interface of van base class

```
public class MagischeHoed<TDier> where TDier : IDier
```

17. Naming Conventions

Identifier	Case	Example
Class	Pascal	AppDomain
Enum type	Pascal	ErrorLevel
Enum values	Pascal	FatalError
Event	Pascal	ValueChanged
Exception class	Pascal	WebException Note Always ends with the suffix Exception .
Read-only Static field	Pascal	RedValue
Interface	Pascal	IDisposable Note Always begins with the prefix I .
Method	Pascal	ToString
Namespace	Pascal	System.Drawing
Parameter	Camel	typeName
Property	Pascal	BackColor
Protected instance field	Camel	redValue Note Rarely used. A property is preferable to using a protected instance field.
Public instance field	Pascal	RedValue Note Rarely used. A property is preferable to using a public instance field.

18. Documenteren C# code via XML

- ▶ Start met `///` (triple slash) gevolgd door 1 van onderstaande XML elementen
 - Rechtstreeks in code toevoegen
 - Of in Class Diagram > Details View > Summary kolom

Predefined XML Documentation Element	Meaning in Life
<code><c></code>	Indicates that the following text should be displayed in a specific "code font"
<code><code></code>	Indicates multiple lines should be marked as code
<code><example></code>	Mocks up a code example for the item you are describing
<code><exception></code>	Documents which exceptions a given class may throw
<code><list></code>	Inserts a list or table into the documentation file
<code><param></code>	Describes a given parameter
<code><paramref></code>	Associates a given XML tag with a specific parameter
<code><permission></code>	Documents the security constraints for a given member
<code><remarks></code>	Builds a description for a given member
<code><returns></code>	Documents the return value of the member
<code><see></code>	Cross-references related items in the document
<code><seealso></code>	Builds an "also see" section within a description
<code><summary></code>	Documents the "executive summary" for a given member
<code><value></code>	Documents a given property

18. Documenteren C# code via XML

- Voorbeeld

```
/// <summary>
/// Constructor
/// </summary>
/// <param name="bankAccountNumber">Number bank account</param>
public BankAccount(string bankAccountNumber)
```

- In VS : documentatie wordt getoond



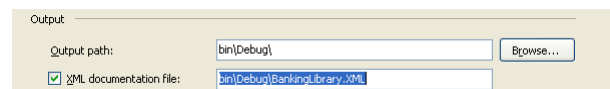
The screenshot shows a code editor with a C# constructor for BankAccount. A tooltip is displayed over the 'new BankAccount()' call, showing the XML documentation for the constructor. The tooltip lists the parameter 'bankAccountNumber' with the description 'Number bank account'.

HoGent

18. Documenteren C# code via XML

- ▶ Generatie xml file

- Via command prompt VS
 - Csc /doc:DocBankAccount.xml c:/.../*.cs
- Instellen in VS, wordt bij compilatie automatisch gegenereerd
 - Selecteer project : bvb BankingLibrary. Ga naar de Build tab van zijn Properties. Vul pad en filenaam in



- ▶ Generatie andere formaten (html,..) vertrekkende van xml file:

- NDoc : <http://sourceforge.net/projects/ndoc>

HoGent

19. Documentatie en Tutorials

- ▶ Tutorial :
 - <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