



Programming 1 (C#)

Week 5

Programma periode 1.1 (Programming 1)

- 01 (wk 36) Introduction C# / Visual Studio 2022 (Community), basic problem solving
- 02 (wk 37) branching, methods
- 03 (wk 38) loops, basic version control setup
- 04 (wk 39) classes, enums, arrays
- 05 (wk 40) public/private, fields/properties, values & references
- 06 (wk 41) inheritance, version control
- 07 (wk 42) Repetition / practice exam
- 08 (wk-43) *no classes*
- 09 (wk-44) exam (*practical, computer*)
- 10 (wk-45) -

Access modifiers

- Access modifiers (or access specifiers) zijn keywords in objectgeoriënteerde talen die de toegankelijkheid van klassen, methoden en andere members instellen.
- Tot nu toe hebben we (bijna) alles 'public' verklaard, wat betekent dat een constructor, methode of field overal toegankelijk is.
 - Voor fields betekent toegankelijk ook manipuleerbaar
- Stel je het volgende voor:

```
Employee teacher = new("Gerwin", "Van Dijken", "INF/IT", 6698.86);  
  
teacher.salary /= 2;  
  
Console.WriteLine(teacher.salary);
```
- We kunnen dergelijke manipulatie voorkomen met access modifiers

```
class Employee {  
    public string firstName, lastName;  
    public string department;  
    public double salary;  
  
    public Employee(string firstName, string lastName,  
                    string department, double salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.department = department;  
        this.salary = salary;  
    }  
  
    public void IncreaseSalary(float increase) {  
        salary += increase;  
    }  
}
```


Access modifiers

Door een kleine wijziging aan te brengen, staan we de manipulatie van de velden niet langer toe.

```
Employee teacher = new("Gerwin", "Van Dijken", "INF/IT", 6698.86);
```

```
teacher.salary /= 2;
```

```
Console.WriteLine
```

 readonly struct System.Double

Represents a double-precision floating-point number.

CS0122: 'Employee.salary' is inaccessible due to its protection level

Opmerking: We kunnen de code nog steeds misleiden om te doen wat we willen. 😊

```
Employee teacher = new Employee("Gerwin", "Van Dijken", "INF / IT", 6698.86);
```

```
teacher.IncreaseSalary(-teacher.GetSalary() / 2);
```

```
Console.WriteLine(teacher.GetSalary());
```

```
class Employee {  
    private string firstName, lastName;  
    private string department;  
    private double salary;  
  
    public Employee(string firstName, string lastName,  
                    string department, double salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.department = department;  
        this.salary = salary;  
    }  
  
    public void IncreaseSalary(float increase) {  
        salary += increase;  
    }  
  
    public double GetSalary() {  
        return salary;  
    }  
}
```

Access modifiers – rationale

- Het doel van een programmeur is om de complexiteit van individuele stukjes code zoveel mogelijk te beperken.
- Het public verklaren van methoden/fields betekent dat potentieel elk stuk code dat een verwijzing naar een object heeft, dergelijke methoden kan aanroepen of dergelijke velden kan manipuleren.
 - Hoe meer je het aantal locaties in code kunt beperken dat dergelijke bewerkingen kan uitvoeren, hoe minder de complexiteit van je code is.
- Voor de code aan de rechterkant kan de programmeur garanderen dat er geen andere manieren zijn om het salaris te manipuleren dan via de IncreaseSalary methode.

```
class Employee {  
    private string firstName, lastName;  
    private string department;  
    private double salary;  
  
    public Employee(string firstName, string lastName,  
                    string department, double salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.department = department;  
        this.salary = salary;  
    }  
  
    public void IncreaseSalary(float increase) {  
        salary += increase;  
    }  
  
    public double GetSalary() {  
        return salary;  
    }  
}
```

this

- In de vorige dia's gebruikten we het trefwoord 'this'
- In klassemethoden, kun je het keyword 'this' gebruiken
- 'this' is een verwijzing naar het object zelf
 - Je kunt 'this' gebruiken om expliciet te verwijzen naar objectleden (velden of methoden)

```
class Employee {  
    private string firstName, lastName;  
    private string department;  
    private double salary;  
  
    public Employee(string firstName, string lastName,  
                    string department, double salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.department = department;  
        this.salary = salary;  
    }  
  
    public void IncreaseSalary(float increase) {  
        salary += increase;  
    }  
  
    public double GetSalary() {  
        return salary;  
    }  
}
```


Type inference

- In een vorige dia gebruikten we `new()` zonder een typeverklaring
- Het type van het geconstrueerde object wordt afgeleid uit de typeverklaring.

```
Employee teacher = new("Gerwin", "Van Dijken", "INF/IT", 6698.86);  
teacher.salary /= 2;  
Console.WriteLine(teacher.salary);
```

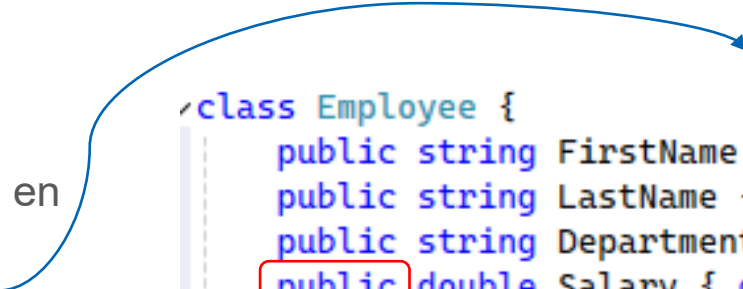
Access modifiers - overview

- Elk field en elke methode in een klasse heeft een specifieke ‘accessibility’ (can be set with so-called ‘access modifiers’)
 - public → field/method is available in its own class, in derived classes and outside the class;
 - internal → field/method is only available in its own assembly: class library (*.dll) or executable (*.exe)
 - protected → field/method is only available in its own class and in derived classes;
 - private → field/method is only available in its own class;
- In deze periode maken we vooral gebruik van public en private

Properties

- Tot nu toe gebruiken we 'fields' voor het opslaan en openen van gegevens binnen een klas.
- We zijn meestal beter af bij het gebruik van properties.
- Properties zijn fields met 'access facility'
 - The property heeft een set (assign) en een get (read) method/accessor
 - Properties zonder set accessor zijn read-only
 - Properties 'should be lightweight'

```
Employee teacher = new("Gerwin", "Van Dijken", "INF/IT", 6698.86);  
  
teacher.Salary /= 2;  
  
Console.WriteLine(teacher.Salary);
```

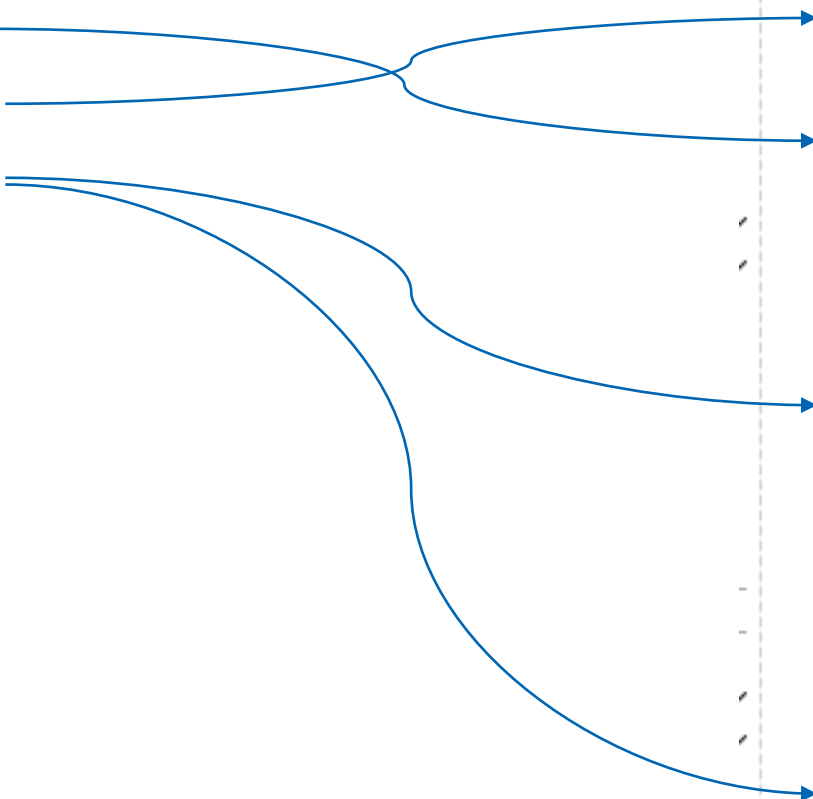


```
class Employee {  
    public string FirstName { get; }  
    public string LastName { get; }  
    public string Department { get; }  
    public double Salary { get; private set; }  
  
    public Employee(string firstName, string lastName,  
                    string department, double salary) {  
        FirstName = firstName;  
        LastName = lastName;  
        Department = department;  
        Salary = salary;  
    }  
  
    public void IncreaseSalary(double increase) {  
        Salary += increase;  
    }  
}
```

Types of properties

- Automatic properties
- Read-only properties
- Computed properties

```
class Book {  
    public string ISBN { get; }  
    public string Title { get; }  
    public string Author { get; }  
  
    public double Price { get; }  
  
    public double VAT {  
        get {  
            // const double VATPercentage = 0.09;  
  
            // Books will be taxed with 21% soon!  
            const double VATPercentage = 0.21;  
  
            return Price * VATPercentage;  
        }  
    }  
  
    public double PriceIncludingVAT {  
        get {  
            return Price + VAT;  
        }  
    }  
}
```



Properties met backing fields

- Soms willen we aangepaste code schrijven voor een get/set-bewerking.
- In dat geval hebben we een ‘backing field’ nodig.
 - Om de speciale aard ervan aan te geven, verklaren we het met een `_`-voorvoegsel.

```
class Book {  
    public string ISBN { get; }  
    public string Title { get; }  
    public string Author { get; }  
  
    private double _price;  
    public double Price {  
        get {  
            return _price;  
        }  
        set {  
            if (value <= 0) {  
                return; // Silent ignore  
            }  
            _price = value;  
        }  
    }  
}
```

Reference types vs value types

- De volgende typen worden standaard doorgegeven op waarde (pass by value):
 - Integer types (int, long, etc.)
 - Floating point types (float, double)
 - Boolean
 - Enum
- Alle andere typen worden doorgegeven door middel van verwijzing (pass by reference)

Parameters doorgeven aan methoden

- Er zijn 3 manieren om parameters door te geven aan methoden:
 1. pass by value
 2. pass by reference
 3. pass by reference out

Parameter 'pass by value'

```
void Start()
{
    int number1 = 8, number2 = 4;
    Console.WriteLine($"[before swap] number 1: {number1}, number 2: {number2}");
    SwapValues(number1, number2);
    Console.WriteLine($"[after swap] number 1: {number1}, number 2: {number2}");
}
```



```
void SwapValues(int value1, int value2)
{
    int temp = value1;
    value1 = value2;
    value2 = temp;
}
```

pass by value:

- *copy of number1 and number2 is used!*
(copied values are swapped)
- *only input*

```
file:///C:/Users/Gerwin van D...
[before swap] number 1: 8, number 2: 4
[after swap] number 1: 8, number 2: 4
```

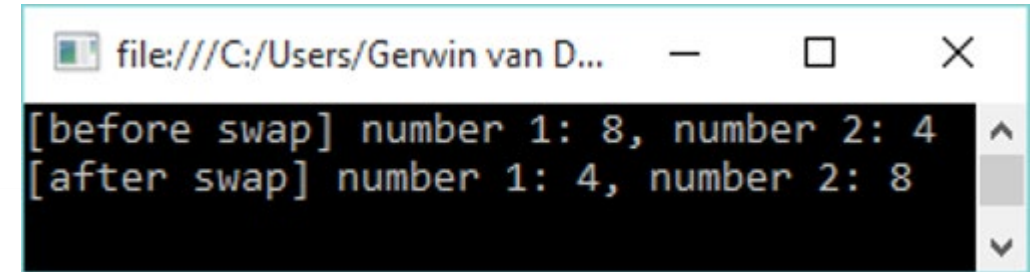
Parameter 'pass by reference'

```
void Start()
{
    int number1 = 8, number2 = 4;
    Console.WriteLine($"[before swap] number 1: {number1}, number 2: {number2}");
    SwapValues(ref number1, ref number2);
    Console.WriteLine($"[after swap] number 1: {number1}, number 2: {number2}");
}

void SwapValues(ref int value1, ref int value2)
{
    int temp = value1;
    value1 = value2;
    value2 = temp;
}
```

↓↑ ↓↑

pass by reference:
- references to number1 and number2 are used
 (number1 and number2 are swapped)
- both input and output



```
file:///C:/Users/Gerwin van D...
[before swap] number 1: 8, number 2: 4
[after swap] number 1: 4, number 2: 8
```

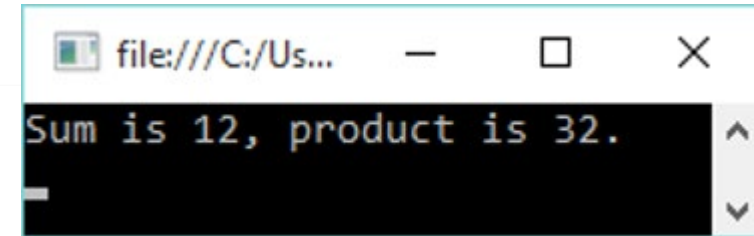

Parameter 'pass by reference out'

```
void Start()
{
    int number1 = 8, number2 = 4;
    int sum, product;
    CalcSumProduct(number1, number2, out sum, out product);
    Console.WriteLine($"Sum is {sum}, product is {product}.");
}

void CalcSumProduct(int num1, int num2, out int sum, out int product)
{
    sum = num1 + num2;
    product = num1 * num2;
}
```

pass by reference out:

- value is returned
- only output



file:///C:/Us... Sum is 12, product is 32.

Exercise – Scrabble (Wordfeud)

- Read (repeatedly) a word until user enters 'stop'. Determine for each word (unequal to 'stop') the Scrabble-score; calculate this score via a separate method 'CalculateScore' that receives a string-parameter and returns an integer value. At the end display the totalscore.

Dutch

Letter Count Points			Letter Count Points		
A	7	1	N	11	1
B	2	4	O	6	1
C	2	5	P	2	4
D	5	2	Q	1	10
E	18	1	R	5	2
F	2	4	S	5	2
G	3	3	T	5	2
H	2	4	U	3	2
I	4	2	V	2	4
J	2	4	W	2	5
K	3	3	X	1	8
L	3	3	Y	1	8
M	3	3	Z	2	5

Exercise – Scrabble (Wordfeud)

main program

```
totalScore = 0
read word
while word ≠ "stop"
    score = CalculateScore(word)
    totalScore = totalScore + score
    read word
display totalScore
```

Exercise – Scrabble (Wordfeud)

CalculateScore(word)

```
totalScore = 0
alfabet = "abcdefghijklmnopqrstuvwxyz"
letterValues = [1,4,5,2,1,4,3,4,2,4,3,3,3,1,1,4,10,2,2,2,2,4,5,8,8,5]

for i=0 to word.Length-1
    character = word[i]
    pos = alfabet.IndexOf(character)
    score = letterValues[pos]
    totalScore = totalScore + score

return totalScore
```

Homework

- (practical class) Programming 1
 - week 5 assignments → Moodle

