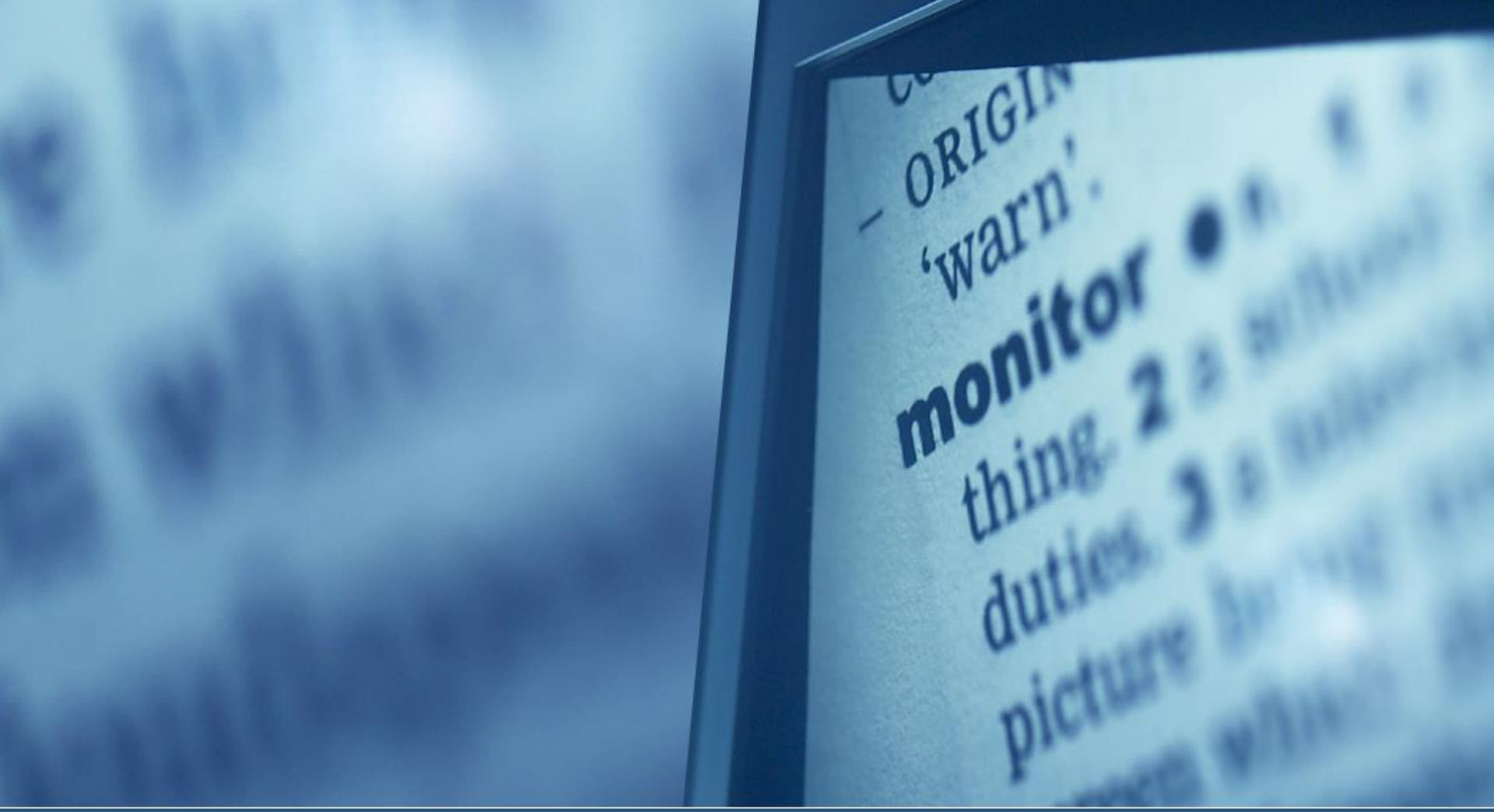


# Programmeren 3

# Programma periode 1.3

| | |
|---|---|
| 01 (wk-05) | classes / constructors / this / static |
| 02 (wk-06) | inheritance / override methods / abstract classes |
| 03 (wk-07) | access modifiers / properties |
| 04 (wk-08) | vakantie |
| 05 (wk-09) | database access / database layer |
| 06 (wk-10) | User Interface / UI + service layer |
| 07 (wk-11) | customizing UI |
| 08 (wk-12) | oefententamen |

---

| | |
|---|---|
| 09 (wk-13) | *tentamens* |
| 10 (wk-14) | *hertentamens* |

# access modifiers

# Scope: access modifiers

- Elke member field en methode in een class heeft een bepaalde 'toegankelijkheid'
- Deze toegankelijkheid is in te stellen met zogenaamde 'access modifiers'
  - public → field/methode is beschikbaar in eigen class, afgeleide classes, en buiten de class;
  - private → field/methode is alleen beschikbaar in eigen class;
  - protected → field/methode is alleen beschikbaar in eigen class en in afgeleide classes;

# Public fields/methods

- Public fields/methods zijn overal te gebruiken

```
class Person
{
    // member fields
    public string firstName, lastName;
    public DateTime dateOfBirth;

    // constructor
    public Person(string firstName, string lastName,
        DateTime dateOfBirth)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
    }

    public override string ToString()
    {
        return $"{firstName} {lastName}";
    }
}
```

# Public fields/methods

- Public fields/methods zijn

```
class Person
{
    // member fields
    public string firstName, lastName;
    public DateTime dateOfBirth;

    // constructor
    public Person(string firstName, string lastName,
        DateTime dateOfBirth)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
    }

    public override string ToString()
    {
        return $"{firstName} {lastName}";
    }
}
```

```
void Start()
{
    Person alex = new Person("Alex", "Smit");
    alex.firstName = "John";
    alex.lastName = "Smits";
}
```

*Public velden firstName en lastName zijn toegankelijk in o.a. de Start methode.*

# Private fields/methods

- Private fields/methods zijn alleen in de class zelf te gebruiken

```
class Person
{
    // member fields
    public string firstName, lastName;
    private DateTime dateOfBirth;

    // constructor
    public Person(string firstName, string lastName,
        DateTime dateOfBirth)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
    }

    public override string ToString()
    {
        return $"{firstName} {lastName} ({dateOfBirth:dd/MM/yyyy})";
    }
}
```

# Private fields/methods

- Private fields/methods zijn alleen in de class zelf te gebruiken

```
class Person
{
    // member fields
    public string firstName, lastName;
    private DateTime dateOfBirth;

    // constructor
    public Person(string firstName, string lastName,
        DateTime dateOfBirth)
    {
        this.firstName = firstName;

    }

    pub
    {

    }
}
```

```
void Start()
{
    Person alex = new Person("Alex", "Smit", new DateTime(2001, 4, 25));
    alex.firstName = "Al";
    alex.lastName = "Smits";
    alex.dateOfBirth = new DateTime(2001, 4, 26);
}
```

*Private fields are not accessible in (e.g.) the Start method.*

struct System.DateTime
Represents an instant in time, typically expressed as a date and time of day.

CS0122: 'Person.dateOfBirth' is inaccessible due to its protection level

# Private fields/methods

- Private fields/methods zijn alleen in de class zelf te gebruiken

```
class Employee : Person
{
    public string department;
    public float salary;

    public Employee(string firstName, string lastName,
                    string department, DateTime dateOfBirth)
     : base(firstName, lastName, dateOfBirth)
    {
        this.department = department;
        this.dateOfBirth = dateOfBirth;
    }

    public override string ToString()
    {
        return $"{base.ToString()} ({department})";
    }
}
```

*Een afgeleide class heeft geen toegang tot private members in de base class.*

# Protected fields/methods

- Protected fields/methods zijn in de class zelf en in afgeleide classes te gebruiken

```
class Person
{
    // member fields
    public string firstName, lastName;
    protected DateTime dateOfBirth;

    // constructor
    public Person(string firstName, string lastName,
        DateTime dateOfBirth)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
    }

    public override string ToString()
    {
        return $"{firstName} {lastName} ({dateOfBirth:dd/MM/yyyy})";
    }
}
```

# Protected fields/methods

- Protected fields/methods zijn in de class zelf en in afgeleide es te gebruiken

```
class Employee : Person
{
    public string department;
    public float salary;

    public Employee(string firstName, string lastName,
                    string department, DateTime dateOfBirth)
     : base(firstName, lastName, dateOfBirth)
    {
        this.department = department;
        this.dateOfBirth = dateOfBirth;
    }

    public override string ToString()
    {
        return $"{base.ToString()} ({department})";
    }
}
```

*Een afgeleide class heeft toegang tot **protected** members in de base class.*

# properties

# Properties

- Properties zijn fields met ‘toegangsregeling’
- Een property bestaat uit een set (write) en een get (read) methode / accessor
- Properties zonder set accessor zijn read-only
- Properties zonder get accessor zijn write-only
- Properties ‘should be lightweight’ *(geen langdurige operatie)*

# Properties - voorbeeld 1

```
class Book
{
    private string title;

    public string Title
    {
        get { return title; }
        set { title = value; }
    }
}
```

*Bij het **schrijven** (vullen) van een property (bv book1.Title = ...) wordt de **set**-code van de property uitgevoerd.*

*Bij het **lezen** van een property (bv text = book1.Title) wordt de **get**-code van de property uitgevoerd.*

```
void Start()
{
    Book book1 = new Book();

    // writing the Title property => set accessor is used
    book1.Title = "The Hobbit";

    // reading the Title property => get accessor is used
    Console.WriteLine($"Title of the book is {book1.Title}");
}
```

# Properties – example 1

```
3   class Book
4   {
5       private string title;
6
7       public string Title
8       {
9           get { return title; }
10          set { title = value; }
11      }
12  }
```

*Extra voordeel van een property: je kunt nu ook een breakpoint plaatsen, als je bv wilt weten wanneer een field gewijzigd wordt.*

```
void Start()
{
    Book book1 = new Book();

    // writing the Title property => set accessor is used
    book1.Title = "The Hobbit";

    // reading the Title property => get accessor is used
    Console.WriteLine($"Title of the book is {book1.Title}");
}
```

# Properties - voorbeeld 2

```
class Book
{
    private string title;
    private int count;

    public string Title
    {
        get { return title; }
        set { title = value; }
    }

    public int Count
    {
        get { return count; }
        set { count = value; }
    }
}
```

```
class Book
{
    private string title;
    private int count;

    public string Title
    {
        get { return title; }
        set { title = value; }
    }

    public int Count
    {
        get { return count; }
        set
        {
            if (value >= 0)
            {
                count = value;
            }
        }
    }
}
```

# Auto-implemented properties

- Auto-implemented properties: verkorte notatie, geen expliciete member fields (geen ‘backing field’)

```
class Book
{
    // automatic properties
    public string Title { get; set; }
    public decimal Price { get; set; }
    public int Count { get; set; }

    // constructor
    public Book(string title, decimal price, int count)
    {
        this.Title = title;
        this.Price = price;
        this.Count = count;
    }
}
```

*Latere interne wijzigingen veranderen de interface niet!*

*Dus als bv property Prijs extra code krijgt in de set, dan zal dat voor de buitenwereld geen gevolgen hebben.*

# Readonly properties

```
class Boek
{
  // backing field
  private float prijs;

  // readonly properties
  public string Titel { get; private set; }
  public float Prijs { get { return prijs; } }

  // read/write properties
  public int AantalExemplaren { get; set; }

  // constructor
  public Boek(string titel, float prijs, int aantal)
  {
    this.Titel = titel;
    this.prijs = prijs;
    this.AantalExemplaren = aantal;
  }
}
```

Property Titel is readonly (voor de buitenwereld) omdat de set alleen binnen de class te gebruiken is.

In bv de constructor kan de Titel wel ingesteld worden (omdat dit binnen de class is).

# ‘Calculated’ properties

```
class Book
{
    // automatic property
    public string Title { get; set; }

    // backing field voor property Count
    private int count;

    // property
    public int Count
    {
        get { return count; }
        set {
            if (value >= 0)
                count = value;
        }
    }

    // read-only property
    public decimal Price { get; private set; }

    // calculated property
    public decimal TotalValue {
        get {
            return Price * Count;
        }
    }
}
```

*calculated property*:
Hier worden *andere* properties gebruikt (om de return waarde te bepalen).

# Huiswerk voor volgende week

- Bestudeer de aangegeven paragrafen uit het ‘Yellow Book’ (zie Moodle)
- Week 3 opdrachten (zie Moodle)