



Programmeren 3

Lesmateriaal

Moodle-course:

- 2021 Inf1.3 Databases

Boek:

- *geen fysiek boek (op Moodle staan verwijzingen)*

Opdrachten:

- wekelijks (6x), individueel, verplicht
- inleveren op Moodle (CodeGrade), deadline volgende week
- alle AutoTests en handmatige controle moeten 10/10 zijn

Lessen en toetsing

Lessen:

- Programmeren 3 theorie: alle klassen samen
- Programmeren 3 praktijk: elke klas apart

Toetsing:

- praktijk-tentamen (programmeer opdrachten) - 1918IN133A
- wekelijkse opdrachten - 1918IN133B
- 3 EC's → praktijk tentamen ≥ 55 **EN** alle opdrachten ✓

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
<hr/>	
09 (wk-13)	<i>tentamens</i>
10 (wk-14)	<i>hertentamens</i>

Programmeer paradigma's

- Er zijn verschillende concepten van programmeren

- Imperatief programmeren (o.a. C, Pascal)

uitvoeren van commando's, toestand

```
printf("Hello World!\n");
```

- **Object-georiënteerd programmeren** (o.a. C#, Java)

werken met objecten

```
class Fiets : Voertuig { ... }
```

- Functioneel programmeren (o.a. Haskell, ML)

definieren / uitvoeren van functies

```
> list1 = [1, 2, 3, 4, 5]  
> filter odd list1
```

- Logisch programmeren (o.a. Prolog)

```
ouder_van(julia, augustus)  
vrouw(julia)  
moeder_van(X,Y) :- ouder_van(X, Y), vrouw(X)
```

Object Oriëntatie (OO)

- Een OO applicatie bestaat uit objecten die gezamenlijk de verantwoordelijkheid dragen voor de uitvoering van het programma
- Een object representeert vaak iets uit de echte wereld, bv een persoon, een machine, een auto, ...
- Een object is verantwoordelijk voor zijn eigen onderdelen/werking

classes en objecten

Classes

- Een class is een blauwdruk/sjabloon waarvan objecten gemaakt kunnen worden
- Classes worden gedeclareerd door de keyword "class", gevolgd door de class name en een verzameling van "class members" omsloten door { accolades }

```
class Person  
{  
    // ...  
}
```

- Afspraak: naam v/e class begint met een Hoofdletter

Objecten aanmaken

- Een object is een instantie van een class
- Je kunt meerdere objecten van een class aanmaken
(zoveel als je wilt, beter: zoveel als je nodig hebt)

- Een object wordt aangemaakt via keyword "new"

```
// create instances (objects) of class Person
Person person1 = new Person();
Person person2 = new Person();
Person person3 = new Person();
// ...
```



- Afspraak: naam v/e object begint met een kleine letter

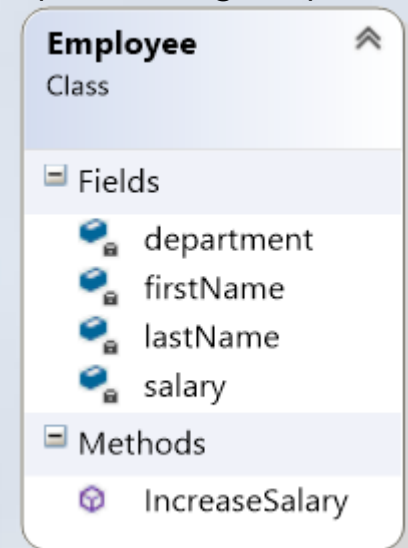
Class members

■ Fields (data) en methods (code)

```
class Employee
{
    // fields
    string firstName, lastName;
    string department;
    float salary;

    // method
    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

(class diagram)



Een class bevat
'fields' en 'methods'
die bij elkaar horen.

Elke 'Employee' heeft een
voornaam, achternaam, afdeling
en salaris, en het salaris kunnen
we verhogen.

Objecten aanmaken en initialiseren

- We kunnen objecten aanmaken en de velden van elk object vullen

De velden van objecten kunnen op verschillende manieren gevuld worden...

```
// one way to create and fill an object
Employee employee1 = new Employee();
employee1.firstName = "Kevin";
employee1.lastName = "Armstrong";
employee1.department = "Human Resources";
employee1.salary = 2500;

// another way to create and fill an object
Employee employee2 = new Employee()
{
    firstName = "Mary",
    lastName = "Clark",
    department = "Software Development",
    salary = 2850
};

// increase salary of employee1
employee1.IncreaseSalary(100);
```

employee1

Kevin
Armstrong
Human R
2600

employee2

Mary
Clark
Software D
2850

2 objecten worden hier aangemaakt: 1 object dat Kevin representeert, en 1 object dat Mary representeert

Alleen het salaris van Kevin wordt verhoogd (van 2500 naar 2600)

Program object

- Elke keer dat we de volgende code gebruiken ...

```
class Program
{
    static void Main(string[] args)
    {
        Program myProgram = new Program();
        myProgram.Start();
    }

    void Start()
    {
        //
    }
}
```

- ... maken v/e object aan van class Program: `new Program()`
- methode Start is een member van class Program, dat we via object "myProgram" aanroepen: `myProgram.Start()`

constructors

Classes - constructors

(1/4)

- Constructor is een methode die (automatisch) wordt aangeroepen bij het aanmaken van een object
- Het wordt gebruikt om elk nieuw object te initialiseren
- Naam van deze methode = class naam
- Een class kan meerdere constructors bevatten (met verschillende parameters)

Classes - constructors

(2/4)

```
class Employee
{
    // fields
    string firstName, lastName;
    string department;
    float salary;

    // constructor
    public Employee()
    {
        firstName = "";
        lastName = "";
    }

    // method
    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

Deze class Employee heeft één constructor

Classes - constructors

(3/4)

```
class Employee
{
    string firstName, lastName;
    string department;
    float salary;

    // constructor 1
    public Employee()
    {
        firstName = "";
        lastName = "";
    }

    // constructor 2
    public Employee(string firstName, string lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

Deze class Employee heeft twee constructors

Dit betekent dat je een Employee object op 2 verschillende manieren kunt aanmaken (zie volgende slide)

Classes - constructors

(4/4)

■ Instanties aanmaken

Welke constructor wordt hier gebruikt?

```
// create instances (objects) of class Employee
Employee emp1 = new Employee();
Employee emp2 = new Employee("Piet", "Paulusma");
```

```
class Employee
{
    // fields
    public string firstName, lastName;
    public string department;
    public float salary;

    public Employee()
    {
        // default salary is 0
        salary = 0;
    }

    public Employee(string fn, string ln)
    {
        firstName = fn;
        lastName = ln;
        salary = 0;
    }

    // method
    public void IncreaseSalary(float increase)
    {
        salary = salary + increase;
    }
}
```

Default constructor

- Een constructor zonder parameters wordt de “default constructor” genoemd. Als je geen constructor zelf maakt, dan is deze default constructor beschikbaar.

```
class Person
{
    public string firstName, lastName;
    DateTime dateOfBirth;
}
```



```
void Start()
{
    Person somePerson = new Person();
    // ...
}
```

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

    public Person(string firstName, string lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```



```
void Start()
{
    Person somePerson = new Person();
    // ...
}
```

Als je een constructor met parameters maakt, dan is de default constructor niet meer beschikbaar!!


this

What's this?

- In methoden van een object kun je keyword 'this' gebruiken
- 'this' is een verwijzing naar het object zelf
- Je kunt 'this' gebruiken om expliciet te verwijzen naar object members (velden of methoden)

```
class Player
{
    public string name;

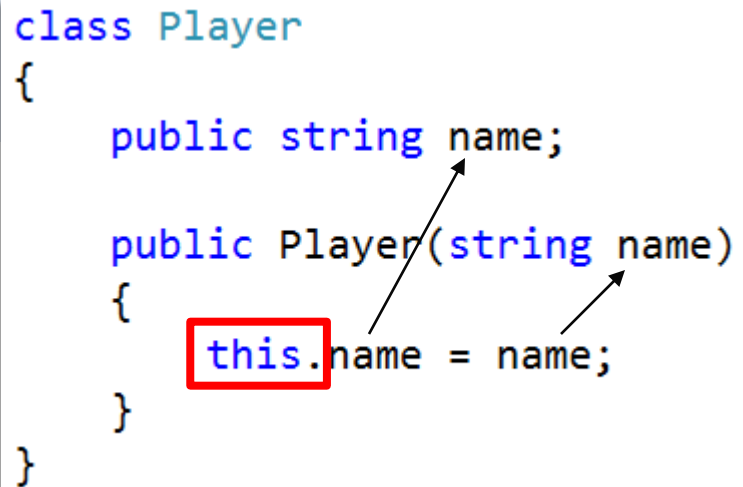
    public Player(string name)
    {
        name = name;
    }
}
```

 (parameter) string name

Assignment made to same variable; did you mean to assign something else?

```
class Player
{
    public string name;

    public Player(string name)
    {
        this.name = name;
    }
}
```



:this gebruiken in constructors

- Als je meerdere constructors hebt, dan kan de ene constructor een andere constructor aanroepen
- Dit kan gebruikt worden om duplicate code in de constructors te voorkomen

:this gebruiken in constructors

```
class Employee
{
    // fields
    public string firstName, lastName;
    public string department;
    public float salary;

    public Employee(string firstName, string lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        salary = 0;
    }

    public Employee(string firstName, string lastName, string department)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.department = department;
        salary = 0;
    }
}
```

*Duplicate code in de
(2) constructors...*

```
Employee employee1 = new Employee("Kevin", "Armstrong");
Employee employee2 = new Employee("Mary", "Clark", "Software Development");
```

:this gebruiken in constructors

```
class Employee
{
    // fields
    public string firstName, lastName;
    public string department;
    public float salary;

    public Employee(string firstName, string lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        salary = 0;
    }

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

We kunnen een constructor aanroepen vanuit een andere constructor via " : this (...)"

Dit roept de 1^{ste} constructor (met 2 parameters) aan

```
Employee employee1 = new Employee("Kevin", "Armstrong");
Employee employee2 = new Employee("Mary", "Clark", "Software Development");
```

static

Classes - instantie vs static

- Instantie class members behoren bij een specifieke instantie / object. Elk object heeft zijn eigen gegevens (status van een object)
(voorbeeld: voornaam, salaris, ...)
- Static class members *behoren bij de class*
(je kunt geen 'this' gebruiken in een static method)
- System.Math

```
int maxValue = Math.Max(number1, number2);
int memorySize = (int)Math.Pow(2, 16);
```
- OO: by default, gebruik geen static!

Classes - static members

■ Voorbeeld

```
class TemperatureConvertor
{
    public static double CelsiusToFahrenheit(double celsius)
    {
        // convert Celsius to Fahrenheit
        double fahrenheit = (celsius * 9 / 5) + 32;

        return fahrenheit;
    }
}
```

```
double celsius = 24.6;
double fahrenheit = TemperatureConvertor.CelsiusToFahrenheit(celsius);
double celsius2 = TemperatureConvertor.FahrenheitToCelsius(fahrenheit);

if (celsius != celsius2)
{
    // ...
}
```

Om een static methode aan te roepen,
wordt de class naam gebruikt
(je hebt geen object nodig)

Huiswerk voor volgende week

- Bestudeer de aangegeven paragrafen uit het 'Yellow Book' (zie Moodle)
- Week 1 opdrachten (zie Moodle)