



Programmeren 3

Vorige week...

Een class is een sjabloon waar we objecten van kunnen maken

```
class Player
{
    public string name;
    public uint score;
    public uint level;

    public Player(string name)
    {
        this.name = name;
        score = 0;
        level = 1;
    }

    public void AddScore(uint add)
    {
        score = score + add;
        level = 1 + (score / 1000);
    }
}
```

constructor,
methode die wordt
aangeropen voor
elk nieuw object

```
Player readyPlayer1 = new Player("Wade Watts");
```

Met 'new' maken
we een object
(instantie) aan

Elk object heeft zijn
eigen toestand
(bv een eigen naam)

```
readyPlayer1.AddScore(150);
```

We kunnen functionaliteiten
(methoden) van een object
aanroepen, zoals 'AddScore'.

Class – overerving (inheritance)

- Hergebruik van bestaande code
- Nieuwe classes definiëren 'bovenop' bestaande classes, om deze uit te breiden en/of gedrag aan te passen
- Afgeleide (derived) class is afgeleide van base class. Afgeleide erft alle members van de base class.

Voorbeeld van overerving

- Werknemer erft van Persoon (*Werknemer "is a" Persoon*)
- Directeur erft van Werknemer (*Directeur "is a" Werknemer*)

```
class Persoon {  
    public string voornaam, achterNaam;  
    DateTime geboorteDatum;  
  
    // ...  
}  
  
class Werknemer : Persoon {  
    public string afdeling;  
    public float salaris;  
    public int personeelsNummer;  
  
    public void VerhoogSalaris(int verhoging) {  
        salaris += verhoging;  
    }  
}  
  
class Directeur : Werknemer {  
    // ...  
}
```

Voorbeeld van overerving

- Werknemer erft van Persoon (*Werknemer "is a" Persoon*)
- Directeur erft van Werknemer (*Directeur "is a" Werknemer*)

```
static void Main(string[] args)
{
    // Directeur erft alles van Werknemer
    // (en Werknemer erft alles van Persoon)
    Directeur directeur = new Directeur();
    directeur.VerhoogSalaris(1000);
}

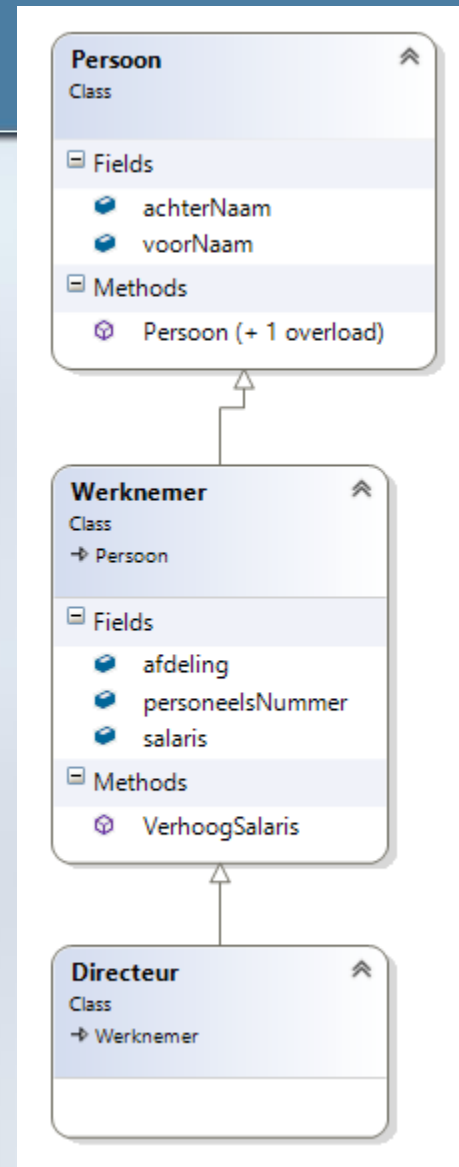
class Werknemer : Persoon {
    public string afdeling;
    public float salaris;
    public int personeelsNummer;

    public void VerhoogSalaris(int verhoging) {
        salaris += verhoging;
    }
}

class Directeur : Werknemer {
    // ...
}
```

Overerving

- Werknemer is afgeleide class van base class Persoon
- Directeur is afgeleide class van base class Werknemer

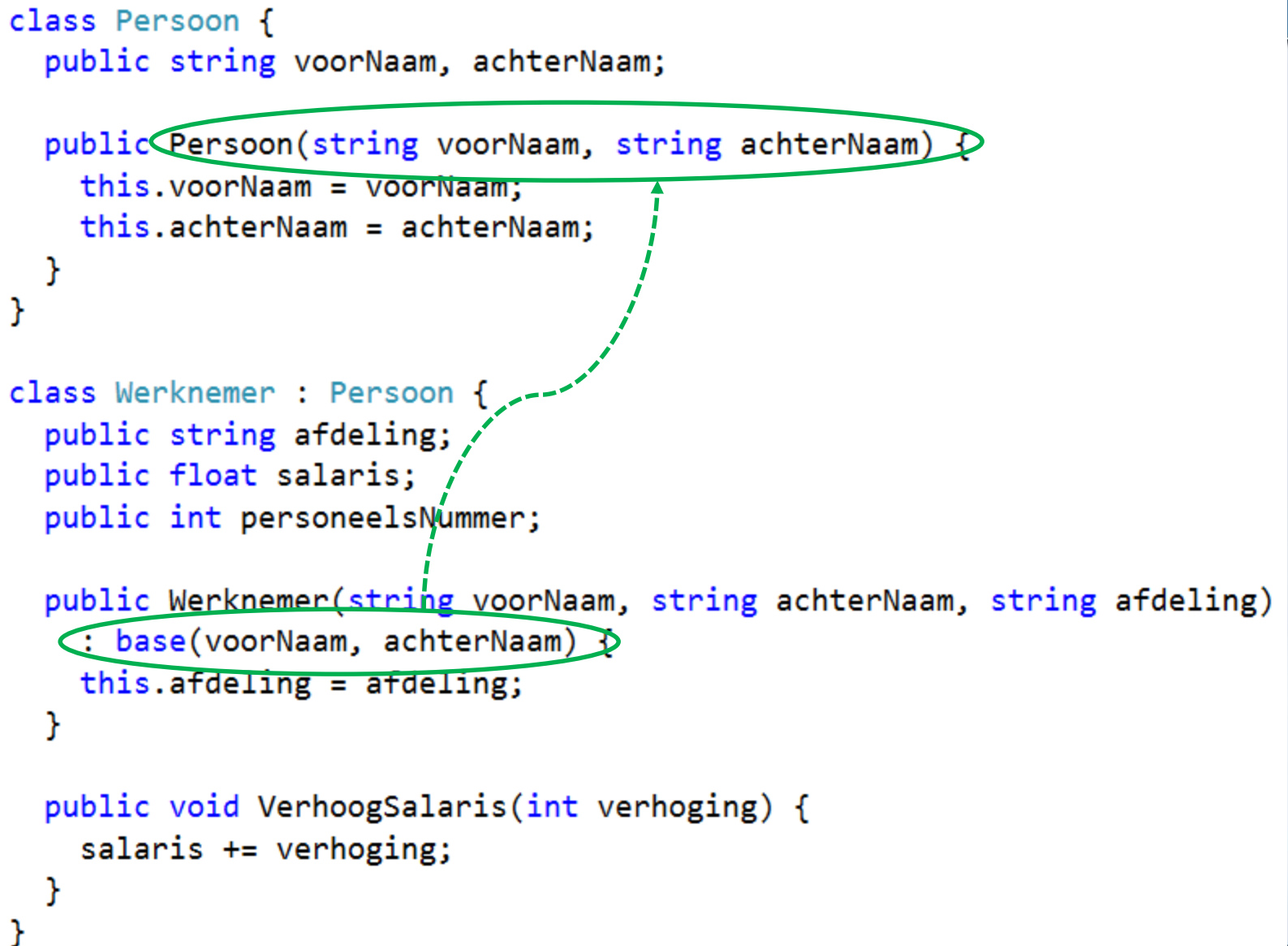


Constructors

- Je kunt een constructor van een base class expliciet aanroepen vanuit een constructor van de afgeleide class, om gegevens door te geven

Constructors

```
class Persoon {  
    public string voorNaam, achterNaam;  
  
    public Persoon(string voorNaam, string achterNaam) {  
        this.voorNaam = voorNaam;  
        this.achterNaam = achterNaam;  
    }  
}  
  
class Werknemer : Persoon {  
    public string afdeling;  
    public float salaris;  
    public int personeelsNummer;  
  
    public Werknemer(string voorNaam, string achterNaam, string afdeling)  
        : base(voorNaam, achterNaam) {  
        this.afdeling = afdeling;  
    }  
  
    public void VerhoogSalaris(int verhoging) {  
        salaris += verhoging;  
    }  
}
```



Constructors

```
class Persoon {  
    public string voorNaam, achterNaam;  
  
    public Persoon(string voorNaam, string achterNaam) {  
        this.voorNaam = voorNaam;  
        this.achterNaam = achterNaam;  
    }  
}
```

```
void Start()  
{  
    Werknemer piet = new Werknemer("Piet", "Paulusma", "Weer");  
    // ...  
}
```

```
public int personeelsnummer;
```

```
public Werknemer(string voorNaam, string achterNaam, string afdeling)  
    : base(voorNaam, achterNaam) {  
    this.afdeling = afdeling;  
}
```

```
public void VerhoogSalaris(int verhoging) {  
    salaris += verhoging;  
}  
}
```

Voornaam en achternaam worden verwerkt door de constructor van class Persoon, afdeling wordt verwerkt door de constructor van class Werknemer.

Methoden toevoegen

- We kunnen extra functionaliteit/gedrag aan een afgeleide class toevoegen (*de afgeleide class kan alles wat 'zijn base class' kan, aangevuld met extra functionaliteiten/methoden*)

Methoden toevoegen

- We kunnen extra class toevoegen (*class' kan, aangevuld met functionaliteiten/*

Zowel de directeur als de werknemer heeft een 'VerhoogSalaris' methode, maar alleen de directeur kan een werknemer ontslaan.

```
class Werknemer : Persoon {
    public string afdeling;
    public float salaris;

    public void VerhoogSalaris(int verhoging)
    {
        salaris += verhoging;
    }
}

class Directeur : Werknemer {

    public void OntslatWerknemer(int personeelsnummer)
    {
        // ...
    }
}

static void Main(string[] args)
{
    Directeur directeur = new Directeur();
    Werknemer werknemer = new Werknemer();
    directeur.OntslatWerknemer(12563);
    werknemer.OntslatWerknemer(13544);
}
```

Method override

- Een afgeleide class kan een methode van de base class 'overschrijven'. Dat betekent dat de afgeleide class een bestaande functionaliteit anders implementeert.

Method override

- Een afgeleide class 'overschrijven'. Dit betekent dat de afgeleide class bestaande functies

```
class Werknemer : Persoon {  
    public string afdeling;  
    public float salaris;  
  
    public virtual void VerhoogSalaris(int verhoging)  
    {  
        salaris += verhoging;  
    }  
}  
  
class Directeur : Werknemer {  
  
    public override void VerhoogSalaris(int verhoging)  
    {  
        salaris += (1.5f * verhoging);  
    }  
}  
  
static void Main(string[] args)  
{  
    Directeur directeur = new Directeur();  
    directeur.VerhoogSalaris(1000);  
}
```

Een directeur krijgt bij een salarisverhoging niet 1000 euro erbij, maar 1500 euro... (ander gedrag)

Polymorfisme

- Polymorfisme ('veelvormigheid'): verschillende objecten (classes) op een zelfde manier verwerken.
- Deze (verschillende) objecten moeten wel dezelfde base class hebben. De base class bepaalt de mogelijke bewerkingen op de objecten.

Polymorfisme

```
class GameObject {  
    public virtual void Draw() {  
        // ...  
    }  
}  
  
class Mario : GameObject {  
    public override void Draw() {  
        // teken hier Mario...  
    }  
}  
  
class Luigi : GameObject {  
    public override void Draw() {  
        // teken hier Luigi...  
    }  
}
```

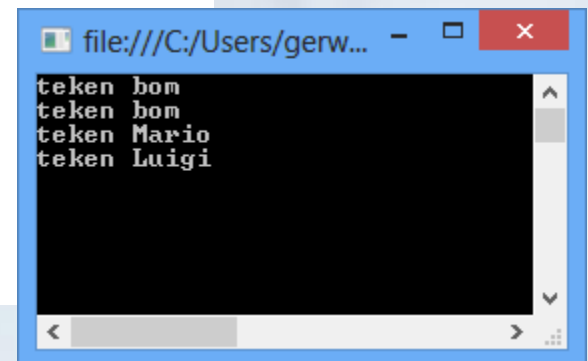


Polymorfisme voorbeeld

```
static void Main(string[] args)
{
    // maak lijst met verschillende game objecten
    List<GameObject> gameObjecten = new List<GameObject>();

    // voeg nieuwe game objecten toe
    gameObjecten.Add(new Bomb());
    gameObjecten.Add(new Bomb());
    gameObjecten.Add(new Mario());
    gameObjecten.Add(new Luigi());

    // teken alle game objecten, ieder object tekent zichzelf
    foreach (GameObject gameObject in gameObjecten)
    {
        gameObject.Draw();
    }
    Console.ReadKey();
}
```



A screenshot of a Windows console window. The title bar shows the file path "file:///C:/Users/gerw...". The console output displays four lines: "teken bom", "teken bom", "teken Mario", and "teken Luigi". The text is white on a black background.

```
file:///C:/Users/gerw...
teken bom
teken bom
teken Mario
teken Luigi
```


Abstracte classes

- Als het niet zinnig is dat van een base class een instantie/object kan worden gemaakt, dan maken we deze class abstract.
- Alle abstracte methoden in een (abstracte) base moeten geïmplementeerd worden in de afgeleide classes, anders zijn deze zelf ook abstract.

Abstract classes - voorbeeld

```
abstract class GameObject {  
    public abstract void Draw();  
}
```

```
class Bomb : GameObject {  
    public override void Draw() {  
        Console.WriteLine("teken bom");  
    }  
}
```

```
class Mario : GameObject {  
    public override void Draw() {  
        Console.WriteLine("teken Mario");  
    }  
}
```

```
class Luigi : GameObject {  
    // .  
}
```

'ConsoleApplication2.Luigi' does not implement inherited abstract member 'ConsoleApplication2.GameObject.Draw()'

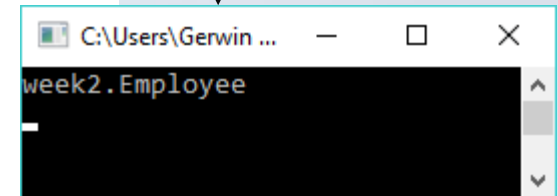
ToString() methode

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

    public Employee(string firstName, string lastName, string department)
        : base(firstName, lastName)
    {
        this.department = department;
        employeeNumber = 123;
    }

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

```
void Start()
{
    Employee piet = new Employee("Piet", "Paulusma", "Weer");
    Console.WriteLine(piet.ToString());
}
```



Zonder een ToString() methode overwrite, is de string-representatie van een object:
<Namespace>.<Classname>

ToString() methode

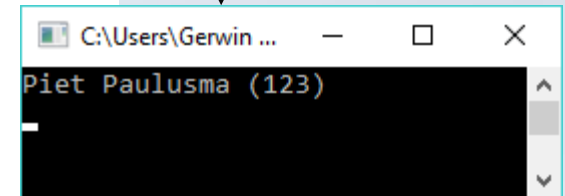
```
class Employee : Person
{
    public string department;
    public float salary;
    public int employeeNumber;

    public Employee(string firstName, string lastName, string department)
        : base(firstName, lastName)
    {
        this.department = department;
        employeeNumber = 123;
    }

    public virtual void IncreaseSalary(int increase)
    {
        salary += increase;
    }

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

```
void Start()
{
    Employee piet = new Employee("Piet", "Paulusma", "Weer");
    Console.WriteLine(piet.ToString());
}
```



A screenshot of a Windows console window. The title bar shows the path 'C:\Users\Gerwin ...'. The console output displays the string 'Piet Paulusma (123)' on a single line. The window has standard Windows controls (minimize, maximize, close) in the top right corner.

Met een ToString() methode overwriten, kun je je eigen string-representatie van een object bepalen.

: this(...)

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

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

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

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

```
void Start()
{
    Person president1 = new Person("Georg", "Bush");

    DateTime dateOfBirth = new DateTime(1911, 2, 6);
    Person president2 = new Person("Ronald", "Reagan", dateOfBirth);
}
```

: this(...)

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

    public Person(string firstName, string lastName)
    : this(firstName, lastName, DateTime.MinValue)
    {
    }

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

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

Default constructor

- Een constructor zonder parameters wordt de 'default constructor' genoemd. Als je geen enkele constructor definieert, dan is deze default constructor automatisch 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();
    // ...
}
```

Echter, als je een constructor met parameters definieert, dan is de default constructor niet meer beschikbaar!!

Huiswerk voor volgende week

- Bestudeer de aangegeven paragrafen uit het 'Yellow Book' (zie Moodle)
- Week 2 opdrachten (zie Moodle)