

4. Geavanceerde methodetechnieken

H7.Methoden



Overzicht



Methoden overloaden



Optionele parameters



Named parameters



Methoden overloaden

Method overloading

```
static void Main(string[] args)
{
    Console.WriteLine(ComputeArea(5, 6));
}
```

```
static int ComputeArea(int h, int w)
{
    int sq = h*w;
    return sq;
}
```

- Wat als je nu de oppervlakte van een cirkel ipv rechthoek wenst te berekenen?

Overloading Methods

- Methode overloaden=
 - Meerdere methoden schrijven met dezelfde naam
 - Compiler kan aan de hand van meegegeven parameters zien welke versie moet geladen worden.

- **Methoden zijn correct overloaded als:**
 - **Methodenaam & returntype dezelfde zijn, maar de parameter lijst is verschillend**

Method overloading

```
static void Main(string[] args)
{
    Console.WriteLine($"Rechthoek: {ComputeArea(5, 6)}");
    Console.WriteLine($"Circle: {ComputeArea(7)}");
}
```

```
static int ComputeArea(int h, int w)
{
    int sq = h*w;
    return sq;
}
```

```
static int ComputeArea(int radius)
{
    int sq =(int) (Math.PI*radius*radius);
    return sq;
}
```

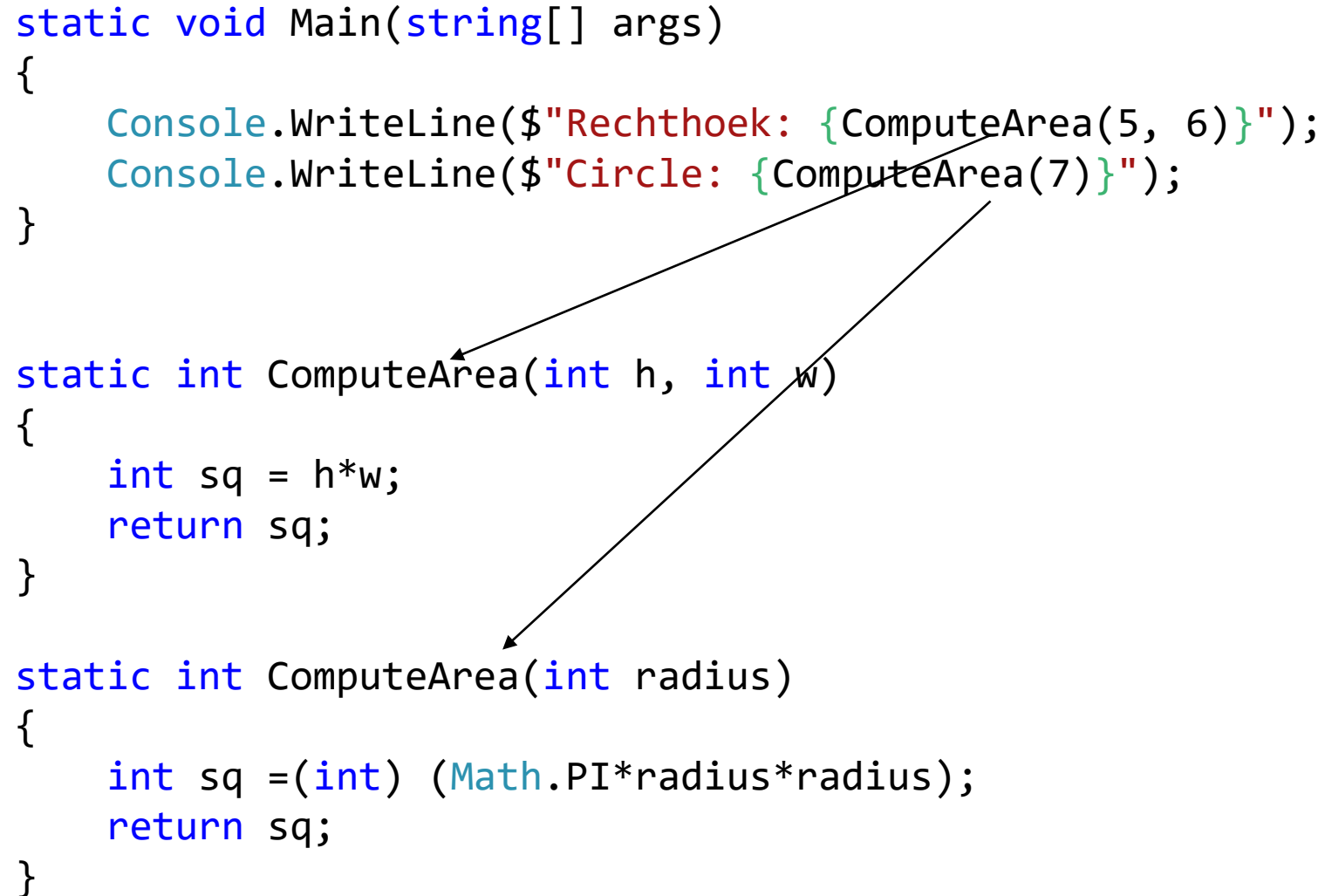
Method overloading

```
Rechthoek: 30  
Circle: 153
```

```
static void Main(string[] args)
{
    Console.WriteLine($"Rechthoek: {ComputeArea(5, 6)}");
    Console.WriteLine($"Circle: {ComputeArea(7)}");
}

static int ComputeArea(int h, int w)
{
    int sq = h*w;
    return sq;
}

static int ComputeArea(int radius)
{
    int sq =(int) (Math.PI*radius*radius);
    return sq;
}
```



Understanding Overload Resolution



“Overload resolution”

Proces waarbij C# regels hanteert om te bepalen welke overloaded versie uitgevoerd moet worden



Toepasbare methoden

Set van methoden die, gegeven de parameters, kan uitgevoerd worden



“Betterness rules”

Effectieve regels die C# toepast om te bepalen welke versie moet uitgevoerd worden.

Zelfde sort regels als die wanneer impliciete casting kan toegepast worden.

Betterness rule

```
Circle 1: 153  
Circle 2: 176  
Circle 3: 167
```

```
static void Main(string[] args)
{
    Console.WriteLine($"Circle 1: {ComputeArea(7)}");
    Console.WriteLine($"Circle 2: {ComputeArea(7.5)}");
    Console.WriteLine($"Circle 3: {ComputeArea(7.3f)}");
}

static int ComputeArea(int radius)
{
    int sq =(int) (Math.PI*radius*radius);
    return sq;
}

static int ComputeArea(double radius)
{
    int sq = (int)(Math.PI * radius * radius);
    return sq;
}
```

Betterness rule

```
Circle 1: 153
Circle 2: 176
Circle 3: 167
```

```
static void Main(string[] args)
{
    Console.WriteLine($"Circle 1: {ComputeArea(7)}");
    Console.WriteLine($"Circle 2: {ComputeArea(7.5)}");
    Console.WriteLine($"Circle 3: {ComputeArea(7.3f)}");
}

static int ComputeArea(int radius)
{
    int sq = (int) (Math.PI*radius*radius);
    return sq;
}

static int ComputeArea(double radius)
{
    int sq = (int)(Math.PI * radius * radius);
    return sq;
}
```

Understanding Overload Resolution (cont'd.)

Data type	Conversions are better in this order
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

Table 8-1

Betterness rules for data type conversion

Ap Voorbeeld: als je een `int` aan een methode meegeeft en er is geen methode met `int` als parametertype dan zal eerst de `long`-versie gekozen worden (indien aanwezig) anders `float`, dan `double`, etc.

Maar...

```
static void Main(string[] args)
{
    Toonverhouding(5, 3.4);
    Toonverhouding(6.2, 3);
}
```

```
static void Toonverhouding(int a, double b)
{
    Console.WriteLine($"{a}/{b}");
}
```

```
static void Toonverhouding(double a, int b)
{
    Console.WriteLine($"{a}/{b}");
}
```

But...

```
static void Main(string[] args)
{
    Toonverhouding(5, 3.4);
    Toonverhouding(6.2, 3);
}

static void Toonverhouding(int a, double b)
{
    Console.WriteLine($"{a}/{b}");
}

static void Toonverhouding(double a, int b)
{
    Console.WriteLine($"{a}/{b}");
}
```



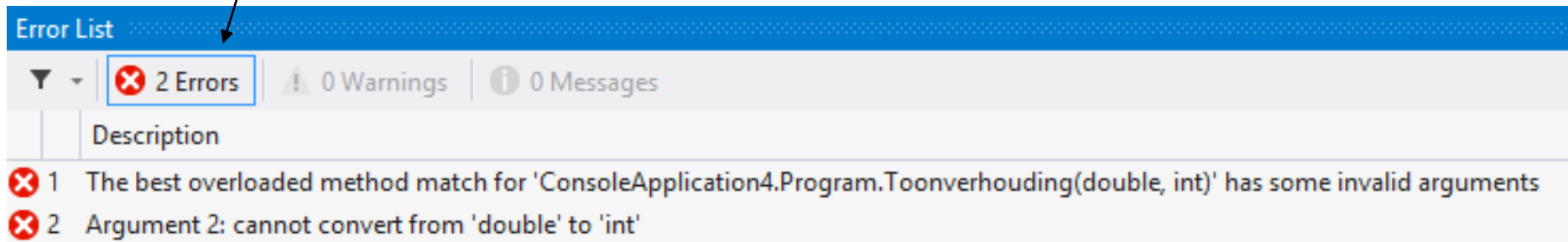
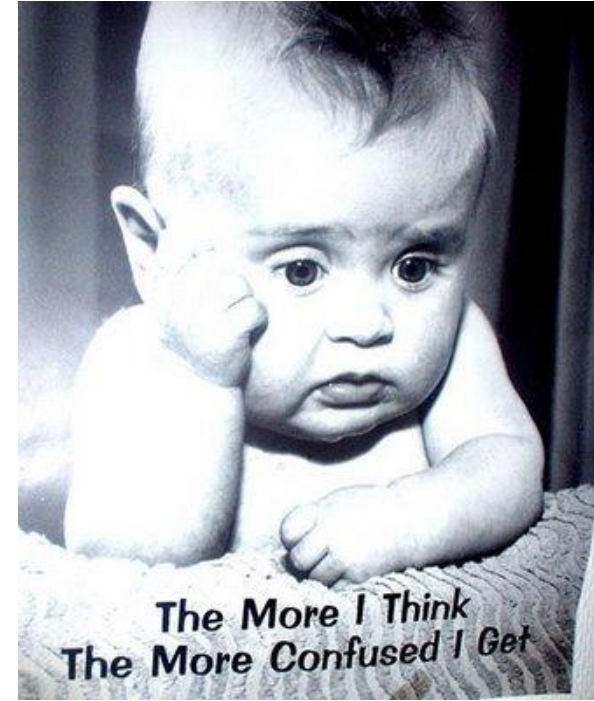
- De eerste parameter die past...hier is geen probleem.

Ambiguous overload

```
static void Main(string[] args)
{
    Toonverhouding(5.6, 3.4);
}

static void Toonverhouding(int a, double b)
{
    Console.WriteLine("{0}/{1}", a, b);
}

static void Toonverhouding(double a, int b)
{
    Console.WriteLine("{0}/{1}", a, b);
}
```

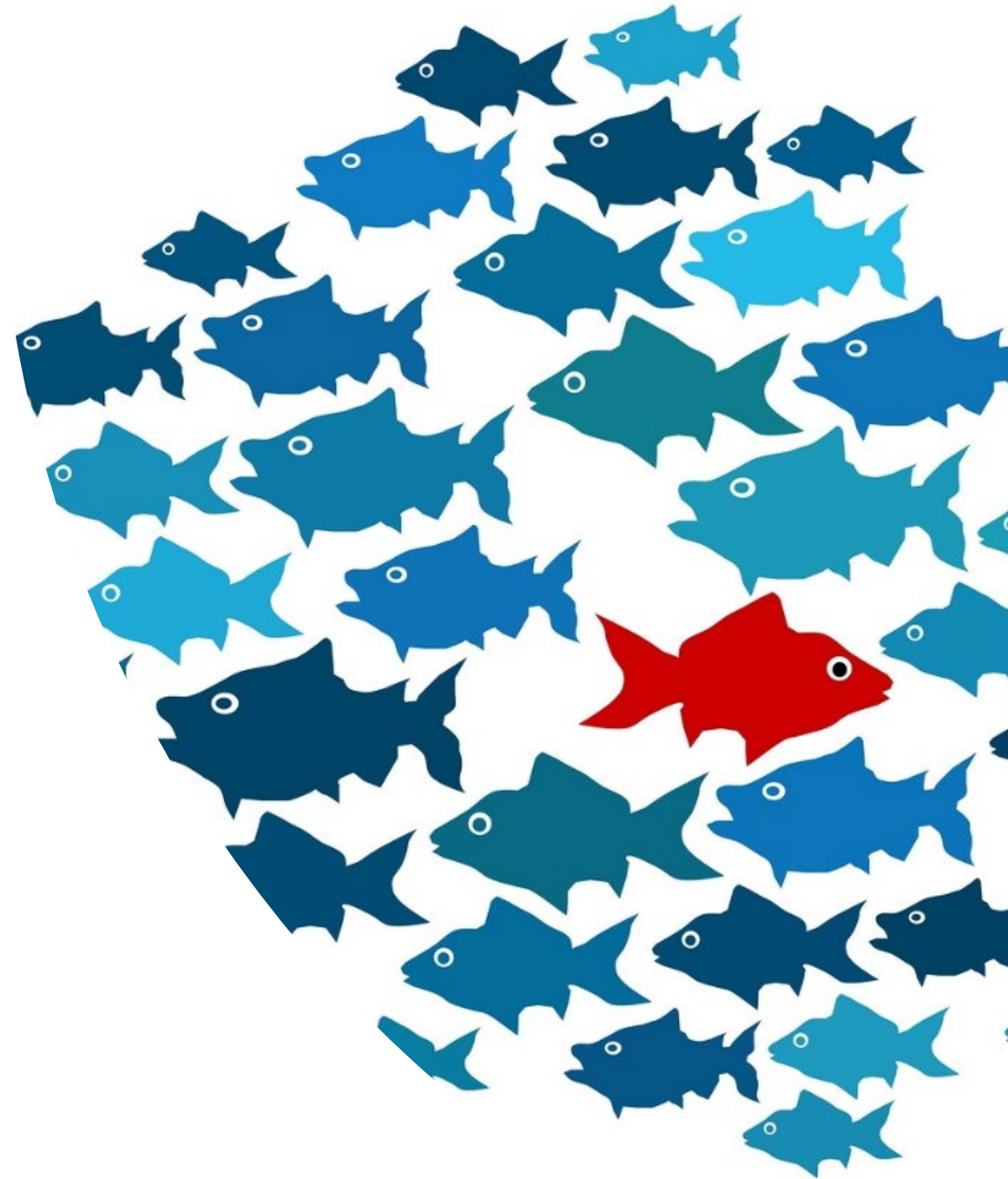


Ambiguous method

Compiler kan niet bepalen welke overloaded versie moet geladen worden

Verschillende return types?

- Methoden met identieke namen MAAR verschillende return types...
 - **Worden niet overloaded**



Optional/Default

Optional Parameters

- **Optional parameter=**
 - Parameter waar een standaard waarde reeds wordt toegekend indien deze geen waarde bij aanroep krijgt
- Maak een parameter optioneel door een standard waarde in de methode signature mee te geven.

- **Optionele parameters in een parameter lijst moeten ACHTER de niet optionele parameters staan.**

Voorbeeld

```
static void ExampleMethod(int required, string optionalstr = "default string", int age = 10)
{
    //...
}
```

- Geldige manieren om aan te roepen:

```
ExampleMethod(15,"tim", 25); //klassieke aanroep
ExampleMethod(20,"dams"); //age zal 10 zijn
ExampleMethod(35)// optionalstr zal "default string" en age zal 10 zijn
```

- Niet geldig:

```
ExampleMethode(3, 4) //daar de tweede param een string moet zijn
ExampleMethode(3, ,4)
```

Geldig of niet?

```
public static void DisplaySize(int length, int width, int height = 1)
```

Method declaration	Explanation
<pre>public static void M1(int a, int b, int c, int d = 10)</pre>	
<pre>public static void M2(int a, int b = 3, int c)</pre>	
<pre>public static void M3(int a = 3, int b = 4, int c = 5)</pre>	
<pre>public static void M4(int a, int b, int c)</pre>	
<pre>public static void M5(int a = 4, int b, int c = 8)</pre>	

Table 8-2 Examples of valid and invalid optional parameter method declarations

Geldig of niet?

Method declaration	Explanation
<code>public static void M1(int a, int b, int c, int d = 10)</code>	Valid. The first three parameters are mandatory and the last one is optional.
<code>public static void M2(int a, int b = 3, int c)</code>	Invalid. Because b has a default value, c must also have one.
<code>public static void M3(int a = 3, int b = 4, int c = 5)</code>	Valid. All parameters are optional.
<code>public static void M4(int a, int b, int c)</code>	Valid. All parameters are mandatory.
<code>public static void M5(int a = 4, int b, int c = 8)</code>	Invalid. Because a has a default value, both b and c must have default values.

Table 8-2 Examples of valid and invalid optional parameter method declarations

Geldig of niet?

```
static void Method1(int a, char b, int c=22, double d= 33.2)
```

Call to Method1()	Explanation
Method1(1, 'A', 3, 4.4);	
Method1(1, 'K', 9);	
Method1(5, 'D');	
Method1(1);	
Method1();	
Method1(3, 18.5);	
Method1(4, 'R', 55.5);	

Table 8-3 Examples of legal and illegal calls to Method1()

Geldig of niet?

```
static void Method1(int a, char b, int c=22, double d= 33.2)
```

Call to Method1()	Explanation
Method1(1, 'A', 3, 4.4);	Valid. The four arguments are assigned to the four parameters.
Method1(1, 'K', 9);	Valid. The three arguments are assigned to a, b, and c in the method, and the default value of 33.2 is used for d.
Method1(5, 'D');	Valid. The two arguments are assigned to a and b in the method, and the default values of 22 and 33.2 are used for c and d, respectively.
Method1(1);	Invalid. Method1() requires at least two arguments.
Method1();	Invalid. Method1() requires at least two arguments.
Method1(3, 18.5);	Invalid. The first argument, 3, can be assigned to a, but the second argument must be type char.
Method1(4, 'R', 55.5);	Invalid. The first argument, 4, can be assigned to a, and the second argument, 'R', can be assigned to b, but the third argument must be type int. You cannot “skip” parameter c, use its default value, and assign 55.5 to parameter d.

Table 8-3 Examples of legal and illegal calls to Method1()

Named parameters

Named parameters

- Sinds C# 4.0: laat optionele parameters er uit en roep de overige parameters met hun naam aan
- **Named mogen in eender welke volgorde geplaatst worden**
 - **MAAR ze moeten NA de unnamed parameters komen (tenzij C# 7.2)**
- Gebruik de naam van de parameter, gevolgd door ':' en de waarde


```
static void PrintOrderDetails(string sellerName, int orderNum, string productName)
{
    //do stuff
}
```

- Aanroep voorbeelden:

```
PrintOrderDetails(orderNum: 31, productName: "Red Mug", sellerName: "Gift Shop");
```

of ook:

```
PrintOrderDetails(productName: "Red Mug", sellerName: "Gift Shop", orderNum: 31);
```

Combinatie named en unnamed parameters

```
static void PrintOrderDetails(string sellerName, int orderNum, string productName)
{
    //do stuff
}
```

```
1 PrintOrderDetails("Gift Shop", 31, productName: "Red Mug");
2 PrintOrderDetails(sellerName: "Gift Shop", 31, productName: "Red Mug"); // C# 7.
3 PrintOrderDetails("Gift Shop", orderNum: 31, "Red Mug");
```

Enkele **NIET GELDIGE** voorbeelden:

```
1 PrintOrderDetails(productName: "Red Mug", 31, "Gift Shop");
2 PrintOrderDetails(31, sellerName: "Gift Shop", "Red Mug");
3 PrintOrderDetails(31, "Red Mug", sellerName: "Gift Shop");
```