C# Essentials: Methodes en parameters

Lector: Tom Quareme

Soorten methoden / procedures

- **Void procedures:**
  - Geven geen waarde terug.
  - Vb. Console.WriteLine(), Close(),...

- **Functie procedures:**
  - Geven 1 waarde terug van een bepaald datatype.
  - Vb. Round(), int.Parse(), ToString(),...

- **Event procedures** (zie hoofdstuk 8 nadien en reeds gebruikte events)**:**
  - Click(), KeyDown(), MouseEnter(),...

Waarom methoden / procedures zelf maken?

- Binnen object-georiënteerd programmeren heet een procedure een ***methode***.

- **Waarom methodes maken?**
  - Om herhaling van statements (code) te voorkomen.
  - De code die in methods staat is herbruikbaar, vb/ libraries.
  - Programma's zijn gemakkelijker te lezen en korter.
  - Programma's zijn gemakkelijker te maken (in groep).
  - Als je methodes public maakt in plaats van private:
    - kan je deze ook in andere files oproepen
    - of in andere projecten gebruiken.

- **Tip:** kijk wanneer je normaal gezien code zou gaan copy-pasten. Probeer dit dan te veralgemenen naar een herbruikbare method.

Console vs. WPF

- **Console:**
  - Voorzie altijd het keyword ***static*** wanneer je een methode definieert:

```
class Program
{
    private static void PrintMaximum(int getal1, int getal2)
    {
        int result = (getal1 > getal2) ? getal1 : getal2;
        Console.WriteLine(result);
    }
    static void Main(string[] args)
    {
        PrintMaximum(10, 9); // method gebruiken
        Console.ReadLine();
    }
}
```

- **WPF:**
  - Hier het keyword static niet gebruiken binnen de MainWindow class!

Console vs. WPF

- **WPF:**
  - Hier het keyword static niet gebruiken binnen de MainWindow class!
  - **Alles in volgende slides voor console is ook toepasbaar op non-static methods!!!**

```
public partial class MainWindow : Window
{
    // ...
    private void PrintMaximum(int getal1, int getal2)
    {
        int result = (getal1 > getal2) ? getal1 : getal2;
        Console.WriteLine(result);
    }

    private void BtnBereken_Click(object sender, RoutedEventArgs e)
    {
        int resultaat = PrintMaximum(10, 9); // method gebruiken binnen event
        TxtResultaat.Text = resultaat.ToString();
    }
}
```

Void methods

- Geven geen waarde terug. Als je toch vroeger uit de method wil gaan: gebruik return.

```
class Program
{
    private static void PrintMaximumBehalveNul(int getal1, int getal2)
    {
        int result = (getal1 > getal2) ? getal1 : getal2;
        if (result < 0)
            return;
        Console.WriteLine(result);
    }

    static void Main(string[] args)
    {
        PrintMaximumBehalveNul(10, 9);
        Console.ReadLine();
    }
}
```

Functie methods

- Geven 1 waarde terug van een bepaald datatype.

```
class Program
{
    private static int KiesMaximum(int getal1, int getal2)
    {
        int result = (getal1 > getal2) ? getal1 : getal2;
        return result;
    }

    private static int SomMethode(int x, int y)
    {
        return x + y;
    }

    static void Main(string[] args)
    {
        int som = SomMethode(2, 3);
        int maximum = KiesMaximum(4, 9);
        Console.WriteLine($"Som: {som}");
        Console.WriteLine($"Maximum: {maximum}");
        Console.ReadLine();
    }
}
```

Method overloading

- **Probleem:**
  - Je hebt een methode die hetzelfde doet, maar de parameters hebben ander datatype.

```
public static int PlusMethodInt(int x, int y)
{
    return x + y;
}

public static double PlusMethodDouble(double x, double y)
{
    return x + y;
}
```

Method overloading

- **Oplossing:**
  - In plaats van 2 methodes met verschillende naam te definiëren, kan je beter overloaden. Overloaden: methode dezelfde naam geven, maar verschillende parameters.

```
public static int PlusMethod(int x, int y)
{
    return x + y;
}

public static double PlusMethod(double x, double y)
{
   return x + y;
}
```

  - Afhankelijk van het datatype dat je doorgeeft bij het oproepen van de methode, wordt de juiste methode aangeroepen.

```
int myNum1 = PlusMethod(8, 5);
double myNum2 = PlusMethod(4.3, 6.26);
Console.WriteLine("Int: " + myNum1);
Console.WriteLine("Double: " + myNum2);
```

Pass by value

- Tot nu toe gaven we parameters door via “pass by value”:
  - De waarde van een variabele wordt zo meegegeven aan de methode.

```
public static void TestPassByValue(int waarde)
{
    Console.WriteLine($"Waarde in method voor wijziging: {waarde}"); // 1
    waarde += 10;
    Console.WriteLine($"Waarde in method na wijziging: {waarde}"); // 11
}
```

  - Er wordt een kopie van de variabele gemaakt voor binnen de methode te gebruiken. Dus de waarde verandert erbuiten niet!

```
int getal = 1;
TestPassByValue(getal);
Console.WriteLine($"Waarde na oproep: {getal}"); // nog steeds 1
```

Pass by reference: ref

- Parameters doorgeven via “pass by reference”:
  - We geven het geheugenadres van de variabele door in plaats van de waarde.

```
public static void TestPassByReference(ref int waarde)
{
    Console.WriteLine($"Waarde in method voor wijziging: {waarde}"); // 1
    waarde += 10;
    Console.WriteLine($"Waarde in method na wijziging: {waarde}"); // 11
}
```

  - Dus de waarde verandert erbuiten als de methode deze aanpast!

```
int getal = 1;
TestPassByReference(ref getal);
Console.WriteLine($"Waarde na oproep: {getal}"); // 11
```

Pass by reference: out

- Parameters doorgeven via “pass by reference”:
  - We geven het geheugenadres van de variabele door in plaats van de waarde.

```
public static void TestPassByOut(out int waarde)
{
    // Je moet de waarde initialiseren binnen de method
    // voordat je de waarde kan gebruiken buiten de method!!!
    waarde = 10;
}
```

  - Dus de waarde verandert erbuiten als de methode deze aanpast!

```
int getalletje; // hoeft niet geïnitialiseerd te zijn in tegenstelling tot ref!!!
TestPassByOut(out getalletje);
Console.WriteLine($"Waarde na out oproep: {getalletje}");
```

  - Wordt vooral gebruikt om (meerdere) parameters te returnen zogezegd.

```
bool isGelukt = int.TryParse("9", out int getal);
```

Pass by reference: out

- Wordt vooral gebruikt om (meerdere) waardes te returnen zogezegd.

```
bool isGelukt = int.TryParse("9", out int getal);
```

- Of:

```
public static void TestReturn2Waarden(out float waarde1, out int waarde2)
{
    waarde1 = 10.0f;
    waarde2 = 20;
}


float waarde1;
int waarde2;
TestReturn2Waarden(out waarde1, out waarde2);
Console.WriteLine($"Resultaat 2 waarden: {waarde1:n2} {waarde2}");
```

Verschil tussen ref en out

- **ref:**
  - De waarde van de variabele die je doorgeeft aan de methode is buiten de methode al geïnitialiseerd!!!

```
int getalleke = 1;
TestPassByReference(ref getalleke);
```

  - De methode kan de waarde van de variabele lezen en aanpassen.

- **out:**
  - De waarde van de variabele die je doorgeeft aan de methode is buiten de methode eventueel nog niet geïnitialiseerd.

```
int getalleke;
TestPassByOut(out getalleke);
```

  - De methode moet zelf een waarde toekennen.

```
public static void TestPassByOut(out int getal)
{
     getal = 10; // BELANGRIJK: INITIALISEREN BINNEN DE METHOD!!!
     getal += 10;
}
```

Private vs. public

- Je hebt al gezien dat we soms private of public gebruiken vooraan in de method definitie.

- **private:**
  - De methode kan enkel gebruikt worden door code in dezelfde class of struct.

- **public:**
  - De methode kan gebruikt worden door code in het hele gecompileerde programma.