



C# OOP Programmatie geavanceerd

Indexers—Operators—Attributes

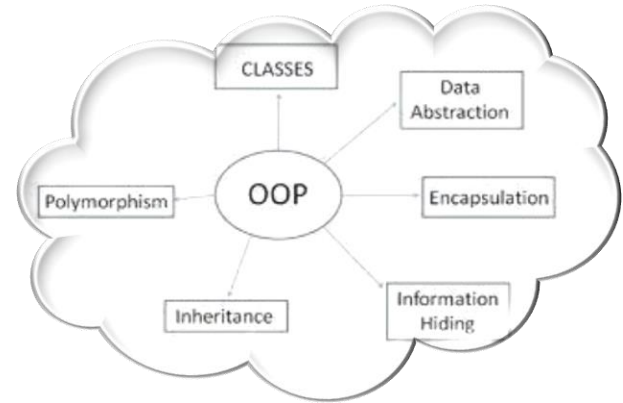


Inhoud

1. Indexers

2. Operatoren

3. Attributen



```
public int this [int index]
{
    ...
}
```



Indexers

Indexers

- **Indexers** verlenen toegang tot class data via index:
- Definieer de `[]` operator voor een gegeven type
 - Zoals bij array elements, bv:

```
IndexedType t = new IndexedType(50);  
int i = t[5];  
t[0] = 42;
```

- Een indexer kan één of meerdere parameters aanvaarden:

```
personInfo["Jos De Klos", 89]
```

- Voorbeeld van hoe men een indexer kan definiëren:

```
public int this [int index] { ... }
```

Indexers – Voorbeeld

```
class KlantenBestand
{
    private Dictionary<long, string> _klanten;

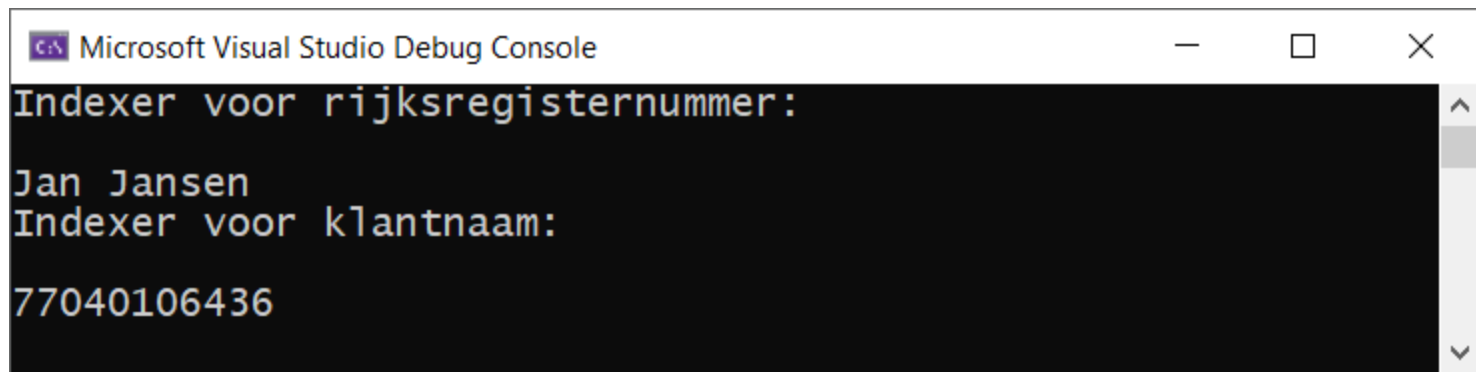
    public KlantenBestand()
    {
        _klanten = new Dictionary<long, string>();
    }
    public void VoegKlantToe(string klantNaam, long RijksRegNr)
    {
        _klanten.Add( RijksRegNr,klantNaam);
    }
    public void VerwijderKlant(long rijksRegNr)
    {
        _klanten.Remove(rijksRegNr);
    }
}
```

Indexers – Voorbeeld(2)

```
public long this[string klantNaam]
{
    get
    {
        if (!_klanten.ContainsValue(klantNaam))
            return -1;
        else
            foreach (KeyValuePair<long, string> element in _klanten)
            {
                if (element.Value == klantNaam) return element.Key;
            }
        return -1;
    }
}
public string this[long rijksRegNr]
{
    get
    {
        return _klanten[rijksRegNr];
    }
    set
    {
        _klanten[rijksRegNr] = value;
    }
}
}
```

Indexers – Voorbeeld(3)

```
class Program
{
    static void Main()
    {
        KlantenBestand klantenBestand = new KlantenBestand();
        klantenBestand.VoegKlantToe("Jan Jansen", 77040106436);
        Console.WriteLine("Indexer voor rijksregisternummer: ");
        Console.WriteLine("\n" + klantenBestand[77040106436]);
        Console.WriteLine("Indexer voor klantnaam: ");
        Console.WriteLine("\n" + klantenBestand["Jan Jansen"]);
    }
}
```



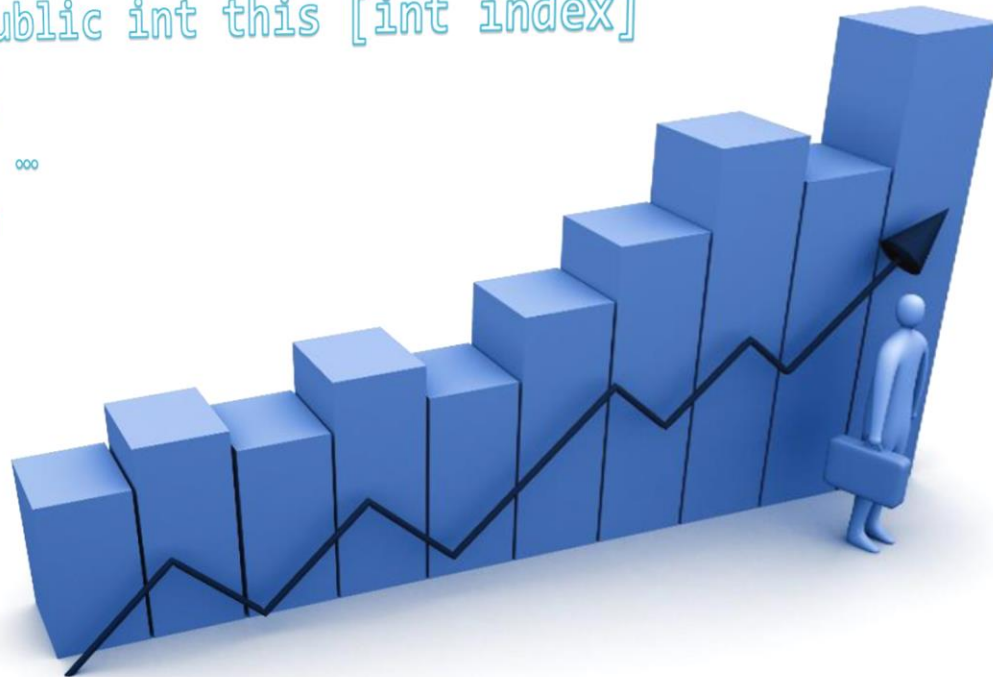
The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio Debug Console". The console output is as follows:

```
Indexer voor rijksregisternummer:
Jan Jansen
Indexer voor klantnaam:
77040106436
```

Indexers

Demo

```
public int this [int index]  
{  
    ...  
}
```



```
var value = list[5];
```


Oefening Indexer voor klasse Bank

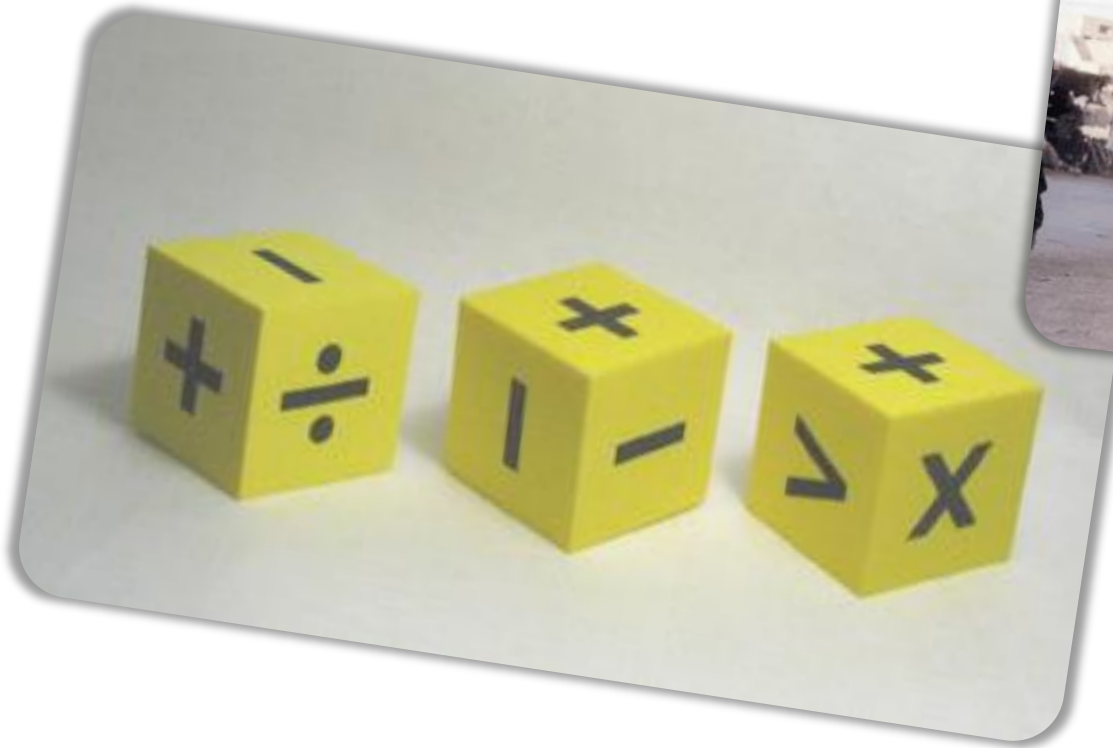
Maak een klasse Bank:

De klasse Bank heeft

- een private field Dictionary<string,Bankrekening>**
- een constructor Bank() die lijst Dictionary<string, Bankrekening> initialiseert**
- een Methode void VoegBankRekeningToe(Bankrekening rek) die een Bankrekening aan de Dictionary toevoegt**
- een indexer die een Bankrekening uit de Dictionary teruggeeft aan de hand van het RekNr**

Maak een klasse Bankrekening:

**Een Bankrekening heeft public properties RekNr (string)
een Saldo (double) met lees-en schrijftoegang**



Overloading
operatoren

Overloading Operatoren

- In C# kunnen bepaalde operatoren worden **overloaded (gedefinieerd)** door de programmeur
 - De prioriteit van de operatoren kan niet worden aangepast
 - Niet alle operatoren kunnen worden overloaded
- Overloading van een operator in C#
 - Lijkt op een static methode met 2 operands, bv:

```
public static Matrix operator +(Matrix m1, Matrix m2)
{
    return new m1.TelOp(m2);
}
```

Overloading Operatoren (2)

- Overloading is toegelaten voor:

- Unaire operatoren:

+, -, !, ~, ++, --, true and false

- Binaire operatoren:

+, -, *, /, %, &, |, ^, <<, >>, ==, !=, >, <, >= and <=

- Operators voor type conversion
 - Impliciete type conversion
 - Expliciete type conversion (**type**)

Overloading Operatoren – Voorbeeld

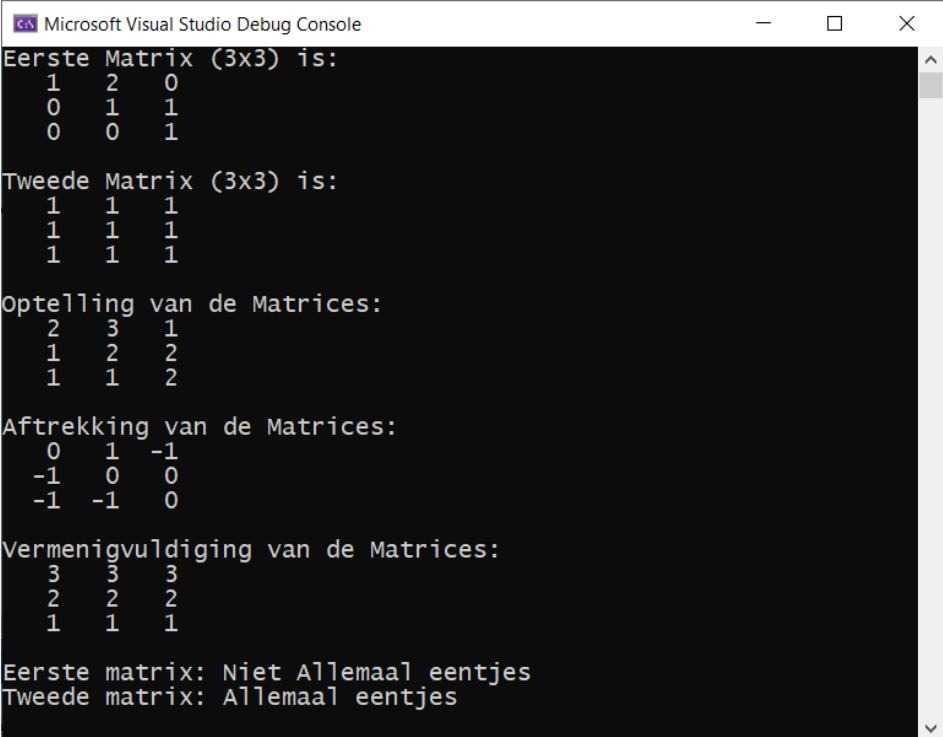
```
public class Matrix
{
    // Field
    private readonly int[, ] matrix = null;

    // Constructor
    public Matrix(uint rijen, uint kolommen)
    {
        this.matrix = new int[rijen, kolommen];
        this.Rijen = rijen;
        this.Kolommen = kolommen;
    }

    // Properties
    public uint Rijen { get; set; }

    public uint Kolommen { get; set; }

    // Indexer
    public int this[uint row, uint col]
    {
        get
        {
            return this.matrix[row, col];
        }
        set
        {
            this.matrix[row, col] = value;
        }
    }
}
```



Microsoft Visual Studio Debug Console

Eerste Matrix (3x3) is:

1	2	0
0	1	1
0	0	1

Tweede Matrix (3x3) is:

1	1	1
1	1	1
1	1	1

Optelling van de Matrices:

2	3	1
1	2	2
1	1	2

Aftrekking van de Matrices:

0	1	-1
-1	0	0
-1	-1	0

Vermenigvuldiging van de Matrices:

3	3	3
2	2	2
1	1	1

Eerste matrix: Niet Allemaal eentjes
Tweede matrix: Allemaal eentjes

Overloading Operatoren – Voorbeeld (2)

```
public override string ToString()
{
    StringBuilder result = new StringBuilder();

    for (int row = 0; row < this.Rijen; row++)
    {
        for (int col = 0; col < this.Kolommen; col++)
            result.AppendFormat("{0,4}", this.matrix[row, col]);
        result.AppendLine();
    }

    return result.ToString();
}

// Optelling (m1 + m2)
public static Matrix operator +(Matrix matrix1, Matrix matrix2)
{
    return TelOp(matrix1, matrix2);
}

// Aftrekking (m1 - m2)
public static Matrix operator -(Matrix matrix1, Matrix matrix2)
{
    return TrekAf(matrix1, matrix2);
}

private static Matrix TelOp(Matrix matrix1, Matrix matrix2)
{
    Matrix result = new Matrix(matrix1.Rijen, matrix1.Kolommen);

    for (uint row = 0; row < result.Rijen; row++)
        for (uint col = 0; col < result.Kolommen; col++)
            result[row, col] = matrix1[row, col] + matrix2[row, col];

    return result;
}
```

Overloading Operatoren – Voorbeeld (3)

```
private static Matrix TrekAf(Matrix matrix1, Matrix matrix2)
{
    Matrix result = new Matrix(matrix1.Rijen, matrix1.Kolommen);

    for (uint row = 0; row < result.Rijen; row++)
        for (uint col = 0; col < result.Kolommen; col++)
            result[row, col] = matrix1[row, col] - matrix2[row, col];

    return result;
}

// vermenigvuldiging (m1 * m2)
public static Matrix operator *(Matrix matrix1, Matrix matrix2)
{
    Matrix result = new Matrix(matrix1.Rijen, matrix2.Kolommen);

    for (uint row = 0; row < result.Rijen; row++)
        for (uint col = 0; col < result.Kolommen; col++)
            for (uint k = 0; k < matrix1.Kolommen; k++) // or i < matrix2.Rows
                result[row, col] += (dynamic)matrix1[row, k] * matrix2[k, col];

    return result;
}
```

Overloading Operatoren – Voorbeeld (4)

```
public static bool operator true(Matrix matrix) // matrix is true als alle elementen == 1
{
    return HeeftAllemaalEentjes(matrix);
}
public static bool operator false(Matrix matrix) //matrix is false als niet alle elementen ==1
{
    return HeeftAllemaalEentjes(matrix);
}
private static bool HeeftAllemaalEentjes(Matrix matrix)
{
    foreach (int element in matrix.matrix)
        if (element !=1)
            return false;

    return true;
}
```



The screenshot shows the Microsoft Visual Studio Debug Console with the following output:

```
Microsoft Visual Studio Debug Console
Eerste Matrix (3x3) is:
1 2 0
0 1 1
0 0 1

Tweede Matrix (3x3) is:
1 1 1
1 1 1
1 1 1

Optelling van de Matrices:
2 3 1
1 2 2
1 1 2

Aftrekking van de Matrices:
0 1 -1
-1 0 0
-1 -1 0

Vermenigvuldiging van de Matrices:
3 3 3
2 2 2
1 1 1

Eerste matrix: Niet Allemaal eentjes
Tweede matrix: Allemaal eentjes
```


Overloading Operatoren – Voorbeeld (5)

```
class Program
{
    static void Main()
    {
        var matrix1 = new Matrix(3, 3);
        matrix1[0, 0] = 1;
        matrix1[0, 1] = 2;
        matrix1[0, 2] = 0;
        matrix1[1, 0] = 0;
        matrix1[1, 1] = 1;
        matrix1[1, 2] = 1;
        matrix1[2, 2] = 2;
        matrix1[2, 2] = 0;
        matrix1[2, 2] = 1;
        var matrix2 = new Matrix(3, 3);
        for (uint row = 0; row < matrix2.Rijen; row++)
            for (uint col = 0; col < matrix2.Kolommen; col++)
                matrix2[row, col] = 1;

        Console.WriteLine("Eerste Matrix ({0}x{1}) is:", matrix1.Rijen, matrix1.Kolommen);
        Console.WriteLine(matrix1);

        Console.WriteLine("Tweede Matrix ({0}x{1}) is:", matrix2.Rijen, matrix2.Kolommen);
        Console.WriteLine(matrix2);

        Console.WriteLine("Optelling van de Matrices:");
        Console.WriteLine(matrix1 + matrix2);

        Console.WriteLine("Aftrekking van de Matrices:");
        Console.WriteLine(matrix1 - matrix2);

        Console.WriteLine("Vermenigvuldiging van de Matrices:");
        Console.WriteLine(matrix1 * matrix2);

        Console.WriteLine("Eerste matrix: {0}", matrix1 ? "Allemaal eentjes" : "Niet Allemaal eentjes");
        Console.WriteLine("Tweede matrix: {0}", matrix2 ? "Allemaal eentjes" : "Niet Allemaal eentjes");
    }
}
```

Overloading Operatoren

Demo



Oefening operatoren

Implementeer de operatoren $+$ en $-$ (optellen en aftrekken voor een klasse Bankrekening, die het Saldo van 2 bankrekeningen optelt of aftrekt een nieuwe bankrekening teruggeeft Saldo = som/verschil van 2 bankrekeningen en)