C#

Veerle Ongenae

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Gegevenstypes

- Waardetypes
- Referentietypes
- Declaratie en toekenning
- Tabellen
- Conversies

Types: Waarde-types

- Waarde-types
  - Voorgedefinieerd
    - int, long, float, double, bool, char, …
  - Zelfgedefinieerd
    - enum (zie verder)
    - struct

Types: Referentie-types

- Referentie-types
  - Voorgedefinieerd
    - object
    - string (!immutable!)
  - Zelfgedefinieerd
    - Klassen
    - Interfaces
    - …
  - Opmerking: null is de lege verwijzing

Opmerkingen

- Voorgedefinieerde types zijn meestal een verkorte vorm van een struct
  - Bv. int is de verkorte vorm van de struct System.Int32
  - Bv. string is de verkorte vorm van de struct System.String
- Elke type (ook waarde-types) is afgeleid van het type object

```
int geheleWaarde = 257;
string stringWaarde = geheleWaarde.ToString();
stringWaarde = 3.ToString();
```

Declaratie en Toekenning

- Declaratie variabele

  type naamVariabele;

- Toekenning

  naamVariabele = waarde;

- Verkort

  type naamVariabele = waarde;

- Declaratie constante

  const type naamConstante = waarde;

Voorbeeld

- TestObjectType.cs

Gegevenstypes

- Waardetypes
- Referentietypes
- Declaratie en toekenning
- Tabellen
- Conversies

Gewone Tabellen

- Eendimensionaal

```
int[] gehelen = new int[3];
gehelen[0] = 15;
gehelen[1] = -231;
gehelen[2] = 3;

int[] evenCijfers = new int[5]{0,2,4,6,8};

int[] onevenCijfers = {1,3,5,7,9};
```

Gewone Tabellen

- Meerdimensionaal

```
int[][] getallen = new int[3][];
getallen[0] = new int[1]{15};
getallen[1] = new int[4]{-231,48,97,-2};
getallen[2] = new int[2]{3,56};
```

Gewone Tabellen

- Meerdimensionaal

```
char[][][] letters = new char[2][][];
letters[0] = new char[3][];
letters[0][0] = new char[3]{'d','i','t'};
letters[0][1] = new char[2]{'i','s'};
letters[0][2] = new char[5]{'g','r','o','e','n'};
letters[1] = new char[1][];
letters[1][0] = new char[3];
letters[1][0][0] = 'a';
letters[1][0][1] = 'x';
letters[1][0][2] = 'p';
```

Rechthoekige Tabellen

- Meerdimensionaal

```
double [,] coord2D = {{1.3,-5.2}, {1.0,4.5}, {0.284,-1.003}};

double [,,] coord3D = new double[4,3,2];
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 3; j++)
        for (int k = 0; k < 2; k++)
            coord3D[i,j,k] = 1;
```

- NIET !!!

  coord3D[i][j][k]

Voorbeeld

- TestTabellen.cs

Gegevenstypes

- Waardetypes
- Referentietypes
- Declaratie en toekenning
- Tabellen
- Conversies

Conversies

- Tussen waardetypes
  - Impliciete conversie (veilig)

```
int geheleWaarde = 257;
long langeWaarde = geheleWaarde;
```

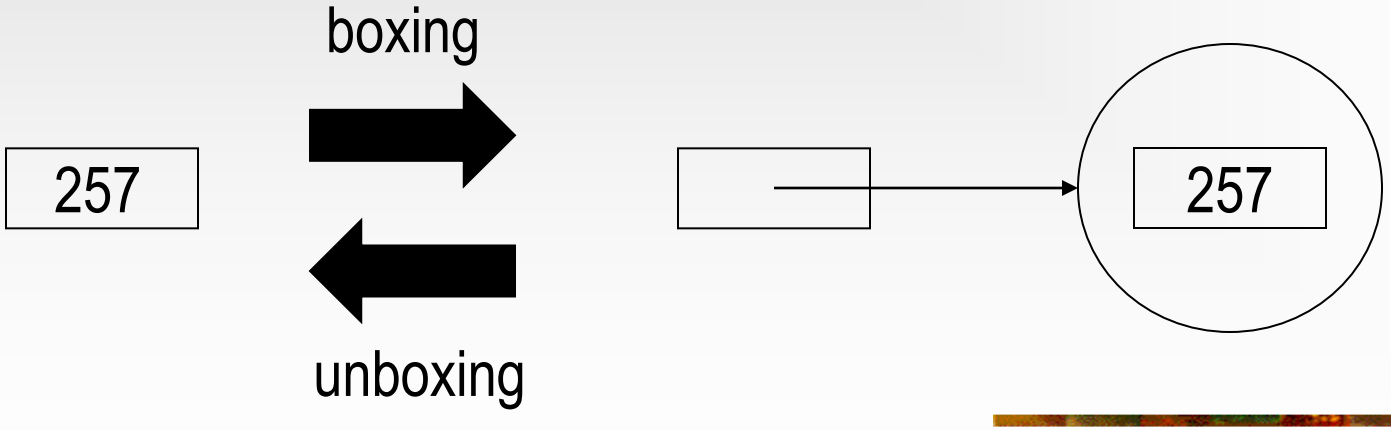
  - Expliciete conversie

```
long langeWaarde = 2200000000;
int geheleWaarde = (int) langeWaarde;
```

  - Voorbeeld: TestConversie
    - Uitschrijven geheleWaarde en langeWaarde

Conversies

- Tussen een waardetype en een referentietype
  - Boxing: waardetype → referentietype
    - Kopie inhoud waardevariabele
  - Unboxing: referentietype → waardetype



Conversies

- Tussen een waardetype en een referentietype
  - Voorbeeld

```
int geheleWaarde = 257;
// boxing
object objWaarde = geheleWaarde;
// unboxing
int geheel = (int) objWaarde;
```

Conversies

- Tussen een waardetype en een referentietype
  - Toepassing

```
// een methode declareren
public object Methode(object o) {
   ...
}
// methode toepassen
int geheleWaarde = 257;
int waarde = (int) Methode(geheleWaarde);
```

Conversies

- Tussen een waardetype en een referentietype
  - Uitvoer?

```
int geheleWaarde = 257;
object objWaarde = geheleWaarde;
geheleWaarde = 342;

System.Console.WriteLine(geheleWaarde);
System.Console.WriteLine(objWaarde);
```

Voorbeeld

- TestBoxing.cs

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Uitdrukkingen

- Zoals in C++ of Java
  - +, -, *, /, %
  - <, >, <=, >=, ==, !=
  - &&, ||
  - …
- Vergelijken strings

```
string woord1 = “Test”;
string woord2 = string.Copy(woord1);
bool idem = (woord1 == woord2); // true
idem = ((object)woord1 == (object)woord2); // false
```

Voorbeeld

- TestStrings.cs

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Programmastructuren

- Selectie

```
if (voorwaarde) {
    …
} else {
    …
}
if (voorwaarde) {
    …
} else if (voorwaarde2){
    …
} else {
    …
}
```

Programmastructuren

- Selectie

```
switch (uitdrukking) {
    case waarde1:
        … // opdrachten
        break;
    case waarde2:
        … break;
    default:
        … break;
}
```

  - Opmerkingen
    - uitdrukking: geheel, karakter, string, enum-type
    - waarde1, waarde2, … : constanten, converteerbaar naar type van uitdrukking
    - default: optioneel

Programmastructuren

- Herhaling

```
for (initialisatie; voorwaarde; iteratie) {
    … // opdrachten
}
foreach (type element in verzamel) {
    … // opdrachten
}
```

- Opmerking
  - verzamel implementeert System.Collections.IEnumerable-interface
  - element mag niet gewijzigd worden in de foreach-lus

Programmastructuren

- Herhaling

```
while (voorwaarde) {
   … // opdrachten
}
```

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Uitzonderingen

- Gooien

  throw uitzondering;

  - Het type van uitzondering is (afgeleid van) System.Exception

Uitzonderingen

- Opvangen

```
try {
    …
} catch (Exception e1) {
    …
} catch (Exception e1) {
    …
… // eventuele andere catch-blokken
} finally {
    …
}
```

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Opsomming (enum)

- Nieuw type
  - Bestaat uit verwante symbolische constanten
- Syntax

  `enum NaamType {SymCst1, SymCst2, …, SymCstn}`
- `NaamType`: naam van het nieuwe type
- `SymCsti`: namen van de constanten waaruit dit type bestaat (correspondeert met een onderliggend geheel getal)

Opsomming (`enum`)

- Voorbeeld

```
public enum Dag
    {Maandag, Dinsdag, Woensdag, Donderdag,
    Vrijdag, Zaterdag, Zondag}
```

- Voorbeeld gebruik

```
Dag[] week = {Dag.Maandag, Dag.Dinsdag};
foreach (Dag dag in week)
    System.Console.WriteLine(dag);
```

- Onderliggend type: int

```
foreach (Dag dag in week)
    System.Console.WriteLine((int) dag);
```

Opsomming (`enum`)

- Voorbeeld gebruik

```
Dag vandaag = Dag.Donderdag;
switch (vandaag) {
        case Dag.Maandag:
                System.Console.WriteLine("dag 1"); break;
        case Dag.Dinsdag:
                System.Console.WriteLine("dag 2"); break;
        …
        case Dag.Zondag:
                System.Console.WriteLine("dag 7"); break;
        default:
                System.Console.WriteLine("geen idee"); break;
}
```

Voorbeeld

- TestEnum.cs
- TestEnumVorm.cs

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Klassen

- Syntax

```
[toegang] class NaamKlasse {
    ...
}
```

- Toegang
  - public
  - protected (klasse + afgeleide klassen)
  - internal (binnen zelfde applicatie)
  - protected internal (protected of internal)
  - private (standaard)

Klassen

- Mogelijke leden
  - Constanten
  - Velden
  - Constructors
  - Methodes
  - Eigenschappen
  - Indexers
  - Genestelde klassen
  - Gebeurtenissen (later)
  - Operatoren (wordt niet behandeld)
  - Destructors (wordt niet behandeld)

Algemeen

- Conventies
  - Klasse: hoofdletter
  - Methode: hoofdletter
  - Namespace: hoofdletter
  - Velden: hoofdletter
  - Eigenschappen: hoofdletter
  - Lokale variabele: kleine letter
  - Naam bestaande uit verschillende woorden: tweede en volgende woord begint met hoofdletter

Velden en constanten

- Veld

  [toegang] [static] [readonly] type NaamVeld;

  - static:
    - Statisch
    - Leden van de klasse ipv het object
  - readonly:
    - Geïnitialiseerd bij declaratie of in de constructor
    - Kan niet meer gewijzigd worden (let op: verschil waardetype en referentietype)
- Constante

  [toegang] const type NaamConstante;

Klassen

- Mogelijke leden
  - Constanten
  - Velden
  - Constructors
  - Methodes
  - Eigenschappen
  - Indexers
  - Genestelde klassen

Constructors

- Syntax

```
[toegang] NaamKlasse(argumentenlijst) {
    …
}
```

- Opmerking

```
[toegang] NaamKlasse(argumentenlijst):this(arglijst) {
    …
}
```

Klassen

- Mogelijke leden
  - Constanten
  - Velden
  - Constructors
  - Methodes
  - Eigenschappen
  - Indexers
  - Genestelde klassen

Methodes

- Syntax

```
[toegang] [static] type NaamMethode(lijstParameters) {
      … // opdrachten methode
      return varType;
}
[toegang] [static] void NaamMethode(lijstParameters) {
      … // opdrachten methode
}
```

- Geen uitzondering aangeven in signatuur methode

Methodes

- Argumenten
  - Pass by value
    - Waardetypes: kopie inhoud variabele
    - Referentietypes: kopie referentie → inhoud object kan wel gewijzigd worden
- Inhoud waardetype wijzigen in een methode
  - Referentieparameter
  - Uitvoerparameter

Methodes: Referentieparameter

- Referentieparameter

```
void NaamMethode(ref int param) {
    … // opdrachten methode
}
// gebruik
int geheel = 2;
NaamMethode(ref geheel);
```

  - Nut: parameter van waardetype veranderen in methode

Methodes: Uitvoerparameter

- Uitvoerparameter
  - Idem referentieparameter, maar hoeft niet geïnitialiseerd

```
void NaamMethode(out int param) {
    … // opdrachten methode
}
// gebruik
int geheel;
NaamMethode(out geheel);
```

Methodes: Tabel Van Parameters

- Tabel van parameters
  - Onbepaald aantal argumenten van hetzelfde type
  - Eén parameter-tabel per methode
  - Meest rechtse argument
  - Vorm

```
type NaamMethode(params type[] naamParam) {
    ... // opdrachten methode
}
```

Methodes: Tabel Van Parameters

```
double Som(params double[] elementen) {
    double som = 0.0;
    for (int i=0; i < elementen.Length; i++)
        som += elementen[i];
    return som;
}
// gebruik
double resultaat;
resultaat = Som();
resultaat = Som(1.0);
resultaat = Som(2.3,2.6);
resultaat = Som(new double[] {1,3.0,4.2});
```

Methodes: Hoofdmethode

- Hoofdmethode: Main
- Verschillende vormen:

```
public static void Main() {
    …
}
public static void Main(string[] args) {
    …
}
```

Voorbeeld

- TestRefParam.cs
- TestParamTabel.cs

Klassen

- Mogelijke leden
  - Constanten
  - Velden
  - Constructors
  - Methodes
  - Eigenschappen
  - Indexers
  - Genestelde klassen

Eigenschappen

- Uitbreiding van velden
- Kenmerken van een object
- get/set-methodes om waarde aan te passen
  - Enkel get-methode → alleen lezen
- Zelfde syntax als velden in gebruik

Eigenschappen

- Syntax

```
public type NaamEigenschap {
    get {
        …
        return varType;
    }
    set {
        … // impliciete variabele value
    }
}
```

Eigenschappen

```
public class Persoon {
    private int leeftijd;

    public Persoon(int leeftijd) {
      this.leeftijd = leeftijd;
    }

    public int Leeftijd {
      get {
                return this.leeftijd;
      }
```

Eigenschappen

```
        set {
                if (value >= 0 && value <= 150)
                        this.leeftijd = value;
        }
    } // einde eigenschap Leeftijd
public bool Volwassen {
        get {
                return (this.leeftijd>=18);
        }
    }
}
```

Eigenschappen

```
public class TestPersoon {
    public static void Main() {
        Persoon[] personen = new Persoon[3];
        personen[0] = new Persoon(13);
        personen[1] = new Persoon(31);
        personen[2] = new Persoon(14);
        System.Console.WriteLine("eerst");
        for (int i = 0; i < 3; i++)
            System.Console.WriteLine(personen[i].Leeftijd);
        personen[0].Leeftijd = 18;
        System.Console.WriteLine("vervolgens");
        for (int i = 0; i < 3; i++)
            System.Console.WriteLine(personen[i].Leeftijd);
```

Eigenschappen

```
            System.Console.WriteLine("volwassen");
            for (int i = 0; i < 3; i++)
                if (personen[i].Volwassen)
                    System.Console.WriteLine(personen[i].Leeftijd);

        }
    }
```

Voorbeeld

- Persoon.cs
- TestPersoon.cs

Klassen

- Mogelijke leden
  - Constanten
  - Velden
  - Constructors
  - Methodes
  - Eigenschappen
  - Indexers
  - Genestelde klassen

Indexers

- Uitbreiding van tabellen
- Klasse kan gebruikt worden zoals een tabel of hashtabel
- Zelfde syntax als tabellen in gebruik
- I.p.v. index ook sleutel mogelijk (cfr. hashtabel)

Indexers: syntax

```
public type this[int index] {
    get {
        … return varType;
    }
    set {
        … // impliciete variabele value
    }
}
public type this[string sleutel] {
    get {
        … return varType;
    }
    set {
        … // impliciete variabele value
    }}
```

Indexers: voorbeeld

```
using System.Collections.Specialized;
public class TestIndexers {

    private static void VulOp(NameValueCollection geg,
    string naam, string voornaam, string onderwerp) {
        geg["naam"] = naam;
        geg["voornaam"] = voornaam;
        geg["onderwerp"] = onderwerp;
    }
```

Indexers: voorbeeld

```
private static void Schrijf(NameValueCollection geg)
{
    for (int i = 0; i < geg.Count; i++)
        System.Console.WriteLine(geg[i]);
    System.Console.WriteLine();
    System.Console.WriteLine(geg["naam"]);
    System.Console.WriteLine(geg["voornaam"]);
    System.Console.WriteLine(geg["onderwerp"]);
    System.Console.WriteLine();
}
```

Indexers: voorbeeld

```
public static void Main() {
    NameValueCollection geg = new
NameValueCollection();

    VulOp(geg,"De Ridder","Jan","Muziek");
    Schrijf(geg);

    VulOp(geg,"Peeters","Piet","Sport");
    Schrijf(geg);
  }
}
```

Uitvoer voorbeeld

- TestIndexers.cs

Bizar voorbeeld

```
public class MijnIndex {
    public object this[int index] {
        get {
            return index;
        }
        set {
            System.Console.WriteLine(value);
        }
    }
```

Bizar voorbeeld

```
public object this[string sleutel] {
    get {
        return sleutel;
    }
    set {
        System.Console.WriteLine(value);
    }
}
```

Bizar voorbeeld

```
public static void Main() {
        MijnIndex mi = new MijnIndex();
        object obj1, obj2 = new object();
        System.Console.WriteLine("Element 2: " + mi[2]);
        mi[3] = 15;
        System.Console.WriteLine("Element tekst: " +
mi["tekst"]);
        mi["help"] = "Help";
        obj1 = mi[1];
        System.Console.WriteLine(obj1);
        System.Console.WriteLine(obj2);
        System.Console.WriteLine("laatste");
        mi[5] = obj2;
    }
}
```

Uitvoer voorbeeld

- MijnIndex.cs

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Namespace

- Hiërarchische structuur
- Samenhangende stukken code (klassen, interfaces, enums, structs, …)
- Syntax

```
namespace Be {
    namespace Hogent {
        namespace Iii {
            public class Test {…}
            …
        }
    }
}
```

Namespace

- Andere syntax

```
namespace Be.Hogent.Iii {
    public class Test {…}
    …
}
```

- Gebruik

```
using Be.Hogent.Iii;
```

- Opmerking
  - Geen corresponderende directorystructuur
  - using enkel voor namespace, niet voor individuele klassen

Namespace: alias

- Een alias (verkorte naam) voor een lange klassenaam
- Voorbeelden

```
using NameValueCollection =
    System.Collections.Specialized.NameValueCollection;
using Test = Be.Hogent.Iii.Test;
```

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Overerving

- Afgeleide klassen
  - Algemene syntax
  - Overschrijven van methodes
- Abstracte klassen
- Interfaces

Afgeleide klassen

```
class A {
    public void F() { Console.WriteLine("A.F"); }
}

class B: A {
    public void G() { Console.WriteLine("B.G"); }
}

class Test {
    static void Main() {
        B b = new B();
        b.F();
        b.G();
    }
}
```

Afgeleide klassen

```
class A {
    public virtual void F() {
        Console.WriteLine("A.F");
    }
}

class B: A {
    public override void F() {
        base.F();
        Console.WriteLine(“B.F"); }
}
```

Afgeleide klassen

```
class Test {
    static void Main() {
        B b = new B();
        b.F();
        A a = b;
        a.F();
    }
}
```

Uitvoer voorbeeld

- TestAfgeleideKlasse1.cs
- TestAfgeleideKlasse2.cs
- Afgeleide klassen (enkelvoudige overerving)
  - `class B:A`
  - Overschrijven methodes
    - `virtual` in basisklasse/bovenklasse
    - `override` in afgeleide klasse
  - Constructor?

Afgeleide klassen: constructor

```
public class BasisKlasse {
    public BasisKlasse() {
        System.Console.WriteLine("Constructor
    BasisKlasse");
    }
    public virtual void MethodeBK() {
        System.Console.WriteLine("BasisKlasse.Method
    eBK");
    }
}
```

Afgeleide klassen: constructor

```
public class MijnKlasse:BasisKlasse {

    public MijnKlasse() : base() {
        System.Console.WriteLine("Constructor MijnKlasse");
    }
    public override void MethodeBK() {
        base.MethodeBK();
        System.Console.WriteLine("MijnKlasse.MethodeBK");
    }

    public void MethodeMK() {
        System.Console.WriteLine("MijnKlasse.MethodeMK");
    }
}
```

Afgeleide klassen: constructor

```
public class Test {
    public static void Main() {
        MijnKlasse mijnObject = new MijnKlasse();
        mijnObject.MethodeBK();
        mijnObject.MethodeMK();
        BasisKlasse basisObject = mijnObject;
        basisObject.MethodeBK();
        basisObject = new BasisKlasse();
        basisObject.MethodeBK();
    }
}
```

Uitvoer voorbeeld

- TestOvererving
- Afgeleide klassen
  - class B:A
  - Overschrijven methodes
    - virtual in basisklasse/bovenklasse
    - override in afgeleide klasse
  - Constructor
    - MijnConstructor(lijst):base(parLijst)
  - Methodes basisklasse oproepen
    - base.methode()

Abstracte klassen

```
abstract class A {
   public abstract void F();
}

class B: A {
   public override void F() {
     Console.WriteLine("B.F");
   }
}
```

Interfaces

- Contract
- Manier om naar een object te kijken
- Leden
  - Methoden
  - Eigenschappen
  - Indexers
  - Gebeurtenissen
- Conventie in .NET: start met de letter I

Interfaces

- Voorbeeld

```
interface IVoorbeeld {
        void Methode(object o);
        string Eigenschap {get; set;}
        string this [int index] {get; set;}
}
// implementeren
interface IMijnInterface : IVoorbeeld, IVoorbeeld2 {
        … // toevoegen methodes, eigenschappen, …
}
class MijnKlasse : IVoorbeeld, IVoorbeeld2 {
        … // overschrijven methodes, eigenschappen, …
 }
```

Afgeleide klasse en interface

- Klasse
  - Afgeleid van een basisklasse
  - Implementeert één of meerdere interfaces
- Syntax

```
class MijnKlasse: BasisKlasse,
   IMijnInterface, INogEenInterface, …
```

  - Eerste basisklasse, dan interfaces
- TestInterfaces.cs

Overzicht C#

- Gegevenstypes
- Uitdrukkingen
- Programmastructuren
- Uitzonderingen
- Opsommingen
- Klassen
- Namespace
- Overerving
- Gebruik

Algemeen

- Automatisch geheugenbeheer (garbage collection)
- .Net: klasse-bibliotheken
  - Bibliotheken die deel uitmaken van .NET SDK (beschikbaar voor meerdere programmeertalen: C#, VBScript, Jscript, …)
  - Voorbeelden
    - System
    - System.Collections
    - System.IO
  - Info: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

Compileren, uittesten

- Compiler (.Net SDK)

  csc MijnBestand.cs
- Resultaat

  MijnBestand.exe
- Meerdere bestanden

  csc /out:Programma.exe Bestand1.cs Bestand2.cs Bestand3.cs
- Een cs-bestand
  - Kan meerdere klassen bevatten
  - Hoeft niet de naam van de klasse te zijn

Compileren, uittesten

- Bibliotheek

  csc /target:library /out:Bib.dll Bestand1.cs Bestand2.cs Bestand3.cs

  - Kan klassen uit meerdere namespaces bevatten
- Gebruik bibliotheek

  csc /reference:Bib.dll Programma.cs

  - Expliciet gebruikte bibliotheken opgeven bij het compileren (cfr. CLASSPATH in Java)

Informatie over C#

- C# Language Specification (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csspec/html/CSharpSpecStart.asp)
- C# Tutorial (http://www.softsteel.co.uk/tutorials/cSharp/cIndex.html
- A Comparative Overview of C# (http://genamics.com/developer/csharp_comparative.htm)