
Introduction à C#

Clémentine Nebut & Abdelhak-Djamel Seriai
Université de Montpellier

Plan

- Les bases ...
- La documentation
- Les métadonnées
- Délégation et événements

Bases

Doc.

Métadonnées

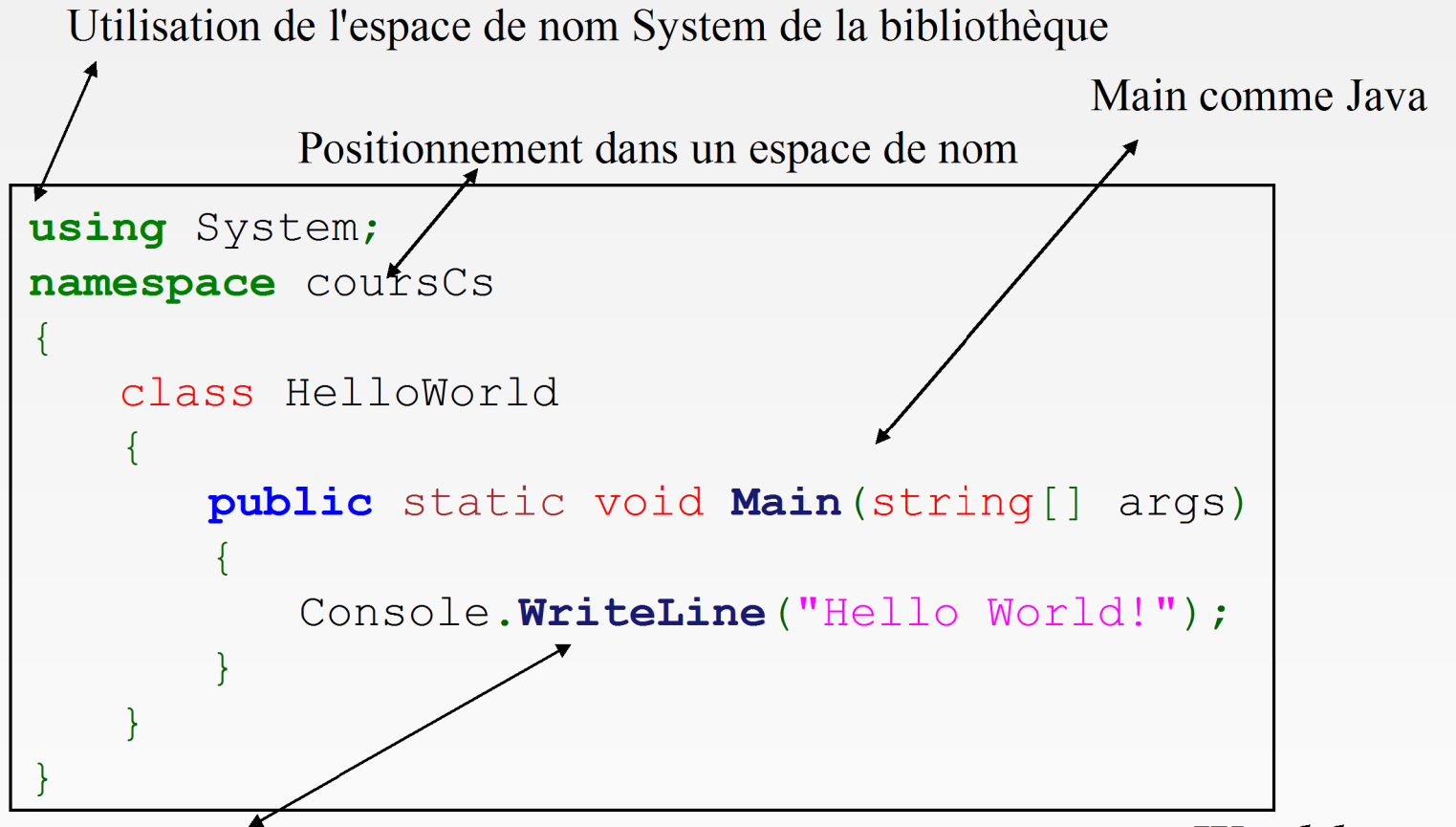
Délég. & evt

Conclu

Les bases ...

- HelloWorld
- Types
- Classes
- Instructions de base
- Les exceptions
- Conversion et typage

Hello world !



Appel à la classe System.Console (E/S standard)

World.cs

Le nom du fichier peut être différent de celui de la classe

Possibilité de faire des classes partielles (une classe dans plusieurs fichiers)

Bases

Doc.

Métadonnées

Délég. & evt

Conclu

Types de base

- sbyte, byte
- short, ushort
- int, uint
- long, ulong
- char
- float
- double
- bool
- decimal

Énumération

```
enum Couleur {Bleu, Blanc, Rouge};

public void PrintBlanc() {
    Couleur c=Couleur.Blanc;
    Console.WriteLine(c.ToString());
}
```

- Type sous-jacent : int
 - Bleu=0 ; Blanc =1 ; Rouge =2
 - conversion vers les entiers : `int i = (int) c`
 - arithmétique (+, -, ...)

- méthode ToString() de System.Object (en java : toString() ...)
- convention de nommage : noms de méthodes commencent par majuscule

Les structures

```
struct Cours{  
    public int VolumeCM;  
    public string Module;  
    public string Enseignant;  
  
    public Cours(int volume, string module, string enseignant){  
        VolumeCM=volume; Module=module; Enseignant=enseignant;  
    }  
}
```

- Même principe qu'une classe
- Est transmis par valeur

•C# est sensible à la casse

Les tableaux

```
int[] tab1 = new int[3]; // tableau de rang 1 et de taille 3
tab1[0]=12;

int[,] tab2 ; /* déclaration d'un tableau de rang 2 */
tab2 = new int[2,3] {{1,2,3},{4,5,6}}; // alloc. et init.

for (int l=0;l<tab2.GetLength(0);l++) {
    for (int c=0;c<tab2.GetLength(1);c++) {
        Console.Write(tab2[l,c]+" ");
    }
    Console.WriteLine();
}
```

- Commentaires // ou /* */

- Console.Write(a+"et"+b+"et"+c) <=>

- Console.Write("{0} et {1} et {2}", 'a', 'b', 'c');

- Les tableaux héritent de System.Array
- Indices à partir de 0

Les classes

- Membres = {propriétés, méthodes, champs, indexeurs, etc}
- Attribut <> champs

```
using System;
namespace coursCs
{
    public class Carre
    {
        public double Longueur;

        public Carre(double longueur) {
            Longueur=longueur;
        }

        public double Perimetre() {
            return Longueur*4;
        }

        public Carre Scale(double pourcentage) {
            Carre c=new Carre(Longueur*pourcentage);
            return c;
        }
    }
}
```

Champs

constructeur

méthodes

création d'instance

Bases

Doc.

Métadonnées

Délég. & evt

Conclu

Les classes partielles

```
partial class ClasseDeTest{  
    // une variable  
    private string maVariable;  
  
    public ClasseDeTest()  
    {  
        // mon constructeur  
    }  
}
```

```
partial class ClasseDeTest{  
    //une methode  
    public void maMethode()  
    {  
        MessageBox.Show(maVariable);  
    }  
}
```

Visibilité des classes

- **public**

- classe visible par n'importe quel programme d'un autre namespace

- **protected (*)**

- classe visible seulement par toutes les autres classes héritant de la classe conteneur de cette classe.

(*) : pour les classes internes

internal

classe visible seulement par toutes les autres classes du même assembly.

Private (*)

classe visible seulement par toutes les autres classes du même namespace

Par défaut

si classe interne : private

sinon : public

Implémentation et héritage

- Pas de différence syntaxique entre l'héritage et l'implémentation ($\neq$ Java)
- Héritage simple
- Implémentation multiple
- Classe non extensible : sealed (= final java)
- Référence à la superclasse : base (équivalent du super de Java)

Classes abstraites

- Contient au moins une méthode abstraite
- Peuvent contenir des attributs et des méthodes concrètes
- Ne peut pas être instanciée
- Méthode abstraite
 - méthode virtuelle, sans corps : mot clef `abstract`
 - `public` ou `protected`
 - définie lors d'un héritage, avec le mot clef `override`

Les classes abstraites

```
public abstract class IForme2{
    public string Nom;
    public abstract double Perimetre();
    public abstract double Aire();
    public void Print(){
        Console.WriteLine("Mon nom est {0}, mon aire est {1}", Nom, Aire());
    }
}

public class Rectangle : IForme2{
    public double Longueur, Largeur;
    public Rectangle(string nom, double longueur, double largeur){
        Longueur=longueur;
        Largeur=largeur;
        Nom=nom;
    }
    public override double Perimetre(){
        return 2.0*(Longueur+Largeur);
    }
    public override double Aire(){
        return Longueur*Largeur;
    }
}
```

Bases

Doc.

Métadonnées

Délég. & evt

Conclu

Les interfaces

```
public interface IForme
{
    double Perimetre();
    double Aire();
}
```

```
public class CarreImplemente : IForme
{
    double Longueur;
    public CarreImplemente(double longueur)
    {
        Longueur=longueur;
    }
    public double Perimetre() {
        return Longueur*4;
    }
    public double Aire() {
        return Longueur*Longueur;
    }
}
```

- Convention : les noms d'interface commencent par I
- Que des signatures de méthode

Appel aux éléments de la superclasse

- Constructeur
 - `public void Etudiant(int age):base(age)`
- Appel de méthode, accès aux membres et propriétés
 - `base.meth()`
 - `base.prop`

Visibilité des membres et méthodes

- **par défaut (aucun mot clef)**
 - private
- **public**
 - visibles par toutes les classes de tous les modules.
- **private**
 - visibles que dans la classe.

protected

visibles par toutes les classes incluses dans le module, et par les classes dérivées de cette classe.

Internal

visibles par toutes les classes incluses dans le même assembly.

Les membres

- Peuvent être :
 - constants (const)
 - en lecture seule pour les clients (readonly)
 - associés avec des accesseurs (-> propriété)
 - partagés par toutes les instances (static)

Les propriétés

- Même syntaxe de définition qu'un attribut
- Fonctionnement par invocations de 2 méthodes d'accès internes : get et set

```
public class Accesseurs{  
    public double Longueur; //champs  
    public double Aire {    // propriété  
        get {return Longueur*Longueur;}  
        set {Longueur=Math.Sqrt(value);}  
    }  
    public void Test(){  
        Aire=9; // appel de set  
        Console.WriteLine(Longueur); // écrit 3  
        Longueur=2;  
        Console.WriteLine(Aire); // écrit 4  
    }  
}
```

Propriétés et constructeurs

```
public class Jouet{  
    private double _prix; //champs  
    public double prix {    // propriété  
        get {return _prix;}  
        set {_prix=value;}  
    }  
}  
public class Test{  
    public void Test(){  
        Jouet buzz=new Jouet(prix=12);  
    }  
}
```

Le passage de paramètres

- Passage :
 - par valeur (pour les types de base)
 - par référence (pour les objets, ou mot clef ref)

```
public void Echanger(ref int a, ref int b)
{
    int temp=a;
    a=b;
    b=temp;
}

public void Echanger2(int a, int b)
{
    int temp=a;
    a=b;
    b=temp;
}
```

```
public void Test() {
    int a=1;
    int b=2;

    Echanger2(a,b);
    Console.WriteLine("a={0} et b={1}",a,b); //a=1 et b=2

    Echanger(ref a, ref b);
    Console.WriteLine("a={0} et b={1}",a,b); //a=2 et b=1
}
```

Paramètres de sortie

```
public void Foo(out int x) {  
    x=1900;  
}  
  
public void Test2() {  
    int a;  
    Foo(out a);  
    Console.WriteLine(a); // affiche 1900  
}
```

Les opérateurs

```
public class Coordonnees2D{
    public double X;
    public double Y;

    public Coordonnees2D(double x, double y) {
        X=x; Y=y;
    }

    public static Coordonnees2D operator + (Coordonnees2D c1,
                                             Coordonnees2D c2) {
        return new Coordonnees2D (c1.X+c2.X, c1.Y+c2.Y);
    }
}
```

```
Coordonnees2D c1=new Coordonnees2D(1,2);
Coordonnees2D c2= new Coordonnees2D(3,4);
c1=c1+c2;
Console.WriteLine("C1 = ({0}, {1})", c1.X, c1.Y); // c1=(4,6);
```

Les indexeurs

```
public double this[int index] {  
    get {  
        if (index==0) return X;  
        else if (index==1) return Y;  
        else return 9999.9999;  
    }  
    set {  
        if (index==0) X=value;  
        else if (index==1) Y=value;  
    }  
}
```

```
Coordonnees2D c1=new Coordonnees2D(1,2);  
Console.WriteLine("C1 = ({0}, {1})", c1[0], c1[1]); // c1=(1,2);
```

- Ne peuvent pas être static (membres de classe)

Les conditionnelles

- `if (cond) {...} else {...}`
- `switch (expr) {`
 - `case valeur1 : ... ; break;`
 - `case valeur2 : ... ; break;`
 - `...`
 - `default : ...;``}`
- Rq : `break` obligatoire

Les boucles

- `while(cond) {...}`
- `do {...} while (cond);`
- `for ( init ; tantQueCond ; incrément) {...}`
- `break;`
- `continue;`

Foreach

- Enumération de collections et de tableaux
- `foreach (type identificateur in expression)`
`instructions`
 - `type` : type de l'identificateur
 - `identificateur` : la variable d'itération
 - `expression` : collection d'objets ou tableau. Le type des éléments de la collection doit pouvoir être converti en le type de l'identificateur. Implémente l'interface `IEnumerable` ou déclare une méthode `GetEnumerator`.
 - `instructions` : les instructions à exécuter

Foreach : exemple

```
public int[] QuatreEntiers=new int[4]{10,8,6,4};  
public Carre[] TroisCarres=new Carre[3]{new Carre(1),  
                                         new Carre(2),new Carre(3)};  
  
public void Test(){  
    foreach (int c in QuatreEntiers){  
        Console.WriteLine(c);  
    }  
  
    foreach (Carre c in TroisCarres){  
        Console.WriteLine(c.Longueur);  
        c.Longueur=c.Longueur+1;  
    }  
  
    foreach (Carre c in TroisCarres){  
        Console.WriteLine(c.Longueur);  
    }  
}
```

Yield return

- pour retourner un itérable
- [http://msdn2.microsoft.com/en-us/library/65zzykke\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/65zzykke(VS.80).aspx)

```
public System.Collections.IEnumerator GetEnumerator()  
{  
    for (int i = 0; i < max; i++)  
    {  
        yield return i;  
    }  
}
```

```
public System.Collections.IEnumerator GetEnumerator()  
{  
    yield return "With an iterator, ";  
    yield return "more than one ";  
    yield return "value can be returned";  
    yield return ".";  
}
```

Yield

- Utilisé dans un bloc itérateur pour fournir une valeur à l'objet énumérateur ou signaler la fin de l'itération.
 - yield return <expression>;
 - yield break;

```
// yield-example.cs
using System;
using System.Collections;
public class List
{
    public static IEnumerable Power(int number, int
exponent)
    {
        int counter = 0;
        int result = 1;
        while (counter++ < exponent)
        {
            result = result * number;
            yield return result;
        }
    }
    static void Main()
    {
        // Display powers of 2 up to the exponent 8:
        foreach (int i in Power(2, 8))
        {
            Console.Write("{0} ", i);
        }
    }
}
```

Les exceptions

```
class MonException : Exception {}

class ExceptionTest {
    public static void Main (String [] args){
        int i = 0;
        try{
            while (true){afficheA10 (i++);}
            Console.WriteLine ("jamais exécuté ici");
        } catch (MonException e){
            /* e.StackTrace : pile des appels de méthodes
             * e.Message : le message associé à l'exception
             * e.InnerException : en cas d'exception imbriquée */ }
            //si on n'attrape pas MonException, une erreur est levée à l'exec
        finally{Console.WriteLine ("Instr. du finally tjs exécutées");}
        Console.WriteLine ("bloc exécuté que si l'erreur est attrapée");
    }
    private static void afficheA10 (int n){
        if (n > 10) throw new MonException();
        Console.WriteLine (n);
    }
}
```

On ne déclare pas les exceptions
pouvant être levées (pas de throws)

Conversion de type et typage

- Conversion de types comme en java :

Type2 y= ...;

Type1 x=(Type1) y;

- Conversion avec instruction as :

Type1 x=y as Type1;

– affecte *null* si conversion impossible

- Test de l'appartenance à un type : instruction *is*
if (x is Type1) ...

Boxing / unboxing

- Boxing : prendre un type primitif et le faire tenir dans un objet (int dans Integer en Java)
- En C# le boxing et unboxing est automatique

int i;

Object o_i =i;

Console.WriteLine((int) o_i);

La documentation

- Même principe que la javadoc :
 - introduction de commentaires spéciaux dans le code
 - ex : `/// <summary>`
 - outil de génération de doc
 - `csc /doc:maDoc.xml maClasse.cs`
- La doc est générée en XML et pas en HTML
- Microsoft fournit une feuille de style par défaut

Les tags de documentation

- `<summary>` petite description `</summary>`
- `<remark>` grosse description `</remark>`
- `<param name="xxx">`descrip. param`</param>`
- `<returns>`desc. de ce qui est retourné`</returns>`
- `<exceptions cref="xxx">`desc. `</exception>`
- `<example>`ex`</example>`
- `<c>` ou `<code>` code C# `</c>` ou `</code>`
- `<see cref="url"></see>`
- `<seealso cref="url"></seealso>`

NDOC

- Projet NDOC, ndoc.sourceforge.net
- Permet de générer aux formats :
 - javadoc
 - LaTeX
 - HTML
 - ...

Avec SharpDevelop et VS

- Aide à la documentation du code
- Dans les options du projet, mettre l'option génération de doc à vrai
 - à la compil, la doc XML sera générée
- Puis Projet->générer la documentation
 - ouvre NDOC

Les métadonnées (attributs)

- Similaire aux annotations Java
- Informations ajoutées en tête d'un élément de code
- Syntaxe : [attribut1, attribut2, ...]
- Exemples :
 - [WebMethod] indique que la méthode est un service web
 - [StaThread] indique qu'une méthode s'exécute dans le même espace mémoire en cas de thread

Définition d'attributs personnalisés

```
using System.Reflection;
public class Author: Attribute
{
    using System;
    public readonly string name;
    public Author(string name)
    {
        this.name = name;
    }
    public override String ToString()
    {
        return String.Format("Author: {0}", name);
    }
}
```

```
[Author("Damien Watkins")]
public class CPoint: Point
{
    public static void Main()
    {
        MemberInfo info = typeof(CPoint);
        object[] attributes =
            info.GetCustomAttributes();
        Console.WriteLine("Custom Attributes are:");
        for (int i = 0; i < attributes.Length; i++)
        {
            System.Console.WriteLine("Attribute "
                + i + ": is " + attributes[i].ToString());
        }
    }
}
```

Délégation

```
public class C{ // on déclare le type de délégué où l'on veut
    public delegate void Le_delegate (int a, string b); // déclaration
}

public class delegation{ //classe utilisant le délégué
    // association entre le type de délégué et le délégué effectif
    public static C.Le_delegate Affichage=new C.Le_delegate(D2.Meth2);
    public static void Main (String [] args){
        Affichage(7,"toto");Console.ReadLine();
    }
}

public class D1{
    public static void Meth1(int x, string y){
        Console.WriteLine("l'entier : {0} puis la chaine : {1}",x,y);}
}

public class D2{
    public static void Meth2(int x, string y){
        Console.WriteLine("la chaine : {0} puis l'entier : {1}",y,x);}
}
```

- Permet de passer en paramètre l'adresse d'une méthode

Les événements

- Mécanisme abonnement/notification
- Le producteur d'événements
 - déclare et produit des événements
- Le consommateur
 - s'abonne à des événements
- Quand un producteur produit un événement
 - tous les abonnés sont notifiés

Et bien d'autres choses ...

- Classiques
 - Introspection
 - Threads
 - Nunit
- Moins classiques
 - Linq
 - Pointeurs et code unsafe
- ...

- Bcp de points communs
- Quelques apports de C#
 - get/set
 - boxing automatique (OK en Java 1.6)
 - les attributs (en Java : annotations)
 - les pointeurs (/unsafe)
 - les délégués
 - ...

Quelques références

- Cours C# de www.labo-dotnet.com/Cours
- Cours C# :
<http://rmdiscala.developpez.com/cours/livres/LivreBases.html#csharp>

Quelques précisions sur le code C#

- Le package `java.lang` est implicitement inclut dans un programme Java alors que C# n'importe aucune classe de manière implicite.
- La méthode `main` en C# doit être écrite avec un « majuscule » : `Main`
- La méthode `Main` en C# peut retourner soit un entier ou un `void`.
- `Main` peut avoir soit un tableau de chaîne de caractères ou aucun paramètre.
- Les commentaires comme en Java
- C# fait la différence entre majuscules et minuscules