

C#, développer en .NET avec Visual Studio



C#, développer en .NET avec Visual Studio

SOMMAIRE JOUR 1

MODULE 1 Introduction à Microsoft .NET

- Leçon 1 Présentation Générale et tour d'horizon
- Leçon 2 Historique et aperçu des types d'application
- Leçon 3 Structure d'une application .NET

MODULE 2 Prise en main de Visual Studio

- Leçon 1 Les solutions dans Visual studio
- Leçon 2 Les projets dans Visual studio
- Leçon 3 Les propriétés
- Leçon 4 Génération et débogage

MODULE 3 Les bases du langage

- Leçon 1 premiers pas
- Leçon 2 Conventions de nommage
- Leçon 3 Les types et les variables
- Leçon 4 conversion de type

3

SOMMAIRE JOUR 2

Module 4 Les bases du langage Part 2

- Leçon 1 Les opérateurs
- Leçon 2 Les boucles
- Leçon 3 Les conditions
- Leçon 4 Passage d'arguments aux méthodes
- Leçon 5 Pointeurs, directives de compilation et attributs

Module 5 Les collections

- Leçon 1 Les tableaux
- Leçon 2 Les enumérateurs
- Leçon 3 Les collections génériques
- Leçon 4 Utilisation des Flux

Module 6 Gestion d'erreurs et Les évènements

- Leçon 1 Les exceptions
- Leçon 2 Les délégués
- Leçon 3 Les évènements

4

C#, développer en .NET avec Visual Studio

SOMMAIRE JOUR 3

Module 7 La programmation orienté objet

Leçon 1 La Programmation procédurale
Leçon 2 La programmation Orienté objet (POO)
Leçon 3 Les classes, représentation du monde objet
Leçon 4 La notion d'héritage
Leçon 5 Les modificateurs d'accès
Leçon 6 Interfaces et classes abstraites

Module 8 La programmation OO avec c#

Leçon 1 Composition d'une classe
Leçon 2 Les méthodes
Leçon 3 Utilisation
Leçon 4 L'héritage
Leçon 5 Les classes abstraites
Leçon 6 l'Encapsulation en csharp
Leçon 7 Types de classes

Module 9 Autres éléments du langage

Leçon 1 Les méthodes d'extension
Leçon 2 Les paramètres nommés et optionnels
Leçon 3 Le typage dynamique

5

SOMMAIRE JOUR 4

Module 10 Accès aux données

Leçon 1 ADO.NET
Leçon 2 Lecture des données
Leçon 3 Mise à jour des données
Leçon 4 Introduction à LINQ

Module 11 Introduction au Windows Form

Leçon 1 introduction
Leçon 2 Fenêtres et évènements
Leçon 3 Les contrôles

MODULE 12 Introduction à WPF

Leçon 1 Le rendu et la disposition en WPF
Leçon 2 Les contrôles en WPF
Leçon 3 Les ressources

6

SOMMAIRE JOUR 5

MODULE 12 Introduction à WPF

Leçon 1 Le rendu et la disposition en WPF
Leçon 2 Les contrôles en WPF
Leçon 3 Les ressources

MODULE 13 .NET Core

Leçon 1 Introduction
Leçon 2 Démarrage
Leçon 3 ConfigureServices

MODULE 14 Injection de dépendances

Leçon 1 Introduction
Leçon 2 Durées de vie du service

MODULE 15 Les middleware

Leçon 1 Introduction
Leçon 2 Mise en place

7

PRÉSENTATIONS ET TOUR DE TABLE

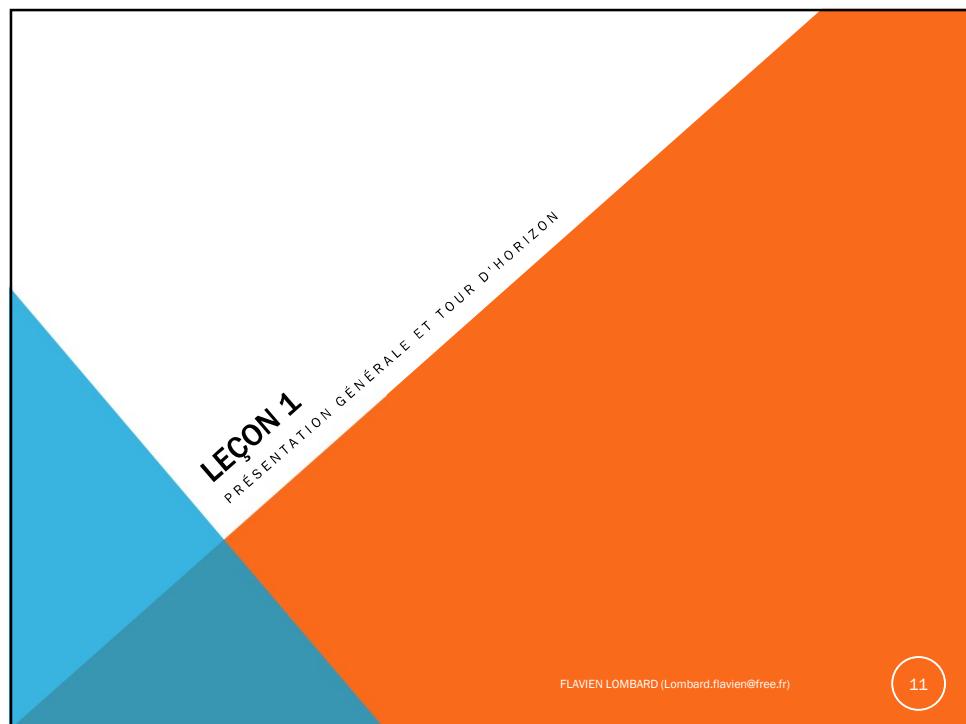


8

C#, développer en .NET avec Visual Studio



The template consists of a white rectangular area with horizontal ruling lines, intended for handwritten notes. This central area is surrounded by a thin black border. At the top left of this border, the word "NOTES" is printed in a small, black, sans-serif font. The entire template is set against a background divided into three colored quadrants: orange at the bottom left, blue at the bottom right, and white at the top and middle sections. In the bottom right corner of the blue quadrant, there is a small circular icon containing the number "10".



The page has a white background with a blue border. In the center, there is a large rectangular area with horizontal ruling lines for notes. The word "NOTES" is printed in bold capital letters at the top left of this area. At the bottom right of the slide, there is a small circular icon containing the number "12".

C#, développer en .NET avec Visual Studio

L'ÉCOSYSTÈME .NET

- Les périphériques clients : ordinateurs, Smartphones, XBox360, etc. ...
- Les services : AZURE,
- Les outils : .NET Framework, Visual Studio .NET
- La communauté .NET

13

LA PLATEFORME .NET

- Microsoft Windows, Linux, MAC (Mono et .NET Core)
- Protocoles de communication Interprocessus basés sur le Framework .NET remplaçant COM et OLE ;
- **Framework .NET remplace MFC, GDI...**
- Infrastructure de la machine virtuel
- IDE Visual Studio
- MSBuild : un outil de gestion de projet multi-compilateurs
- Portabilité;
- des composants facilitant le développement de services (BING) et d'applications locales (Winform WPF) ou web (ASP.NET)

14

C#, développer en .NET avec Visual Studio

LA PLATEFORME .NET

Le Framework .NET

Le Framework .NET est librement distribué

La version cliente ou la version développement doivent être installées après la mise en place du système.

A partir de 2003 Server, Windows intègre le Framework .NET dès l'installation.

Aujourd'hui, .NET 4.X est intégré aux systèmes Windows. La version Core doit être installée séparément.

15

LA PLATEFORME .NET

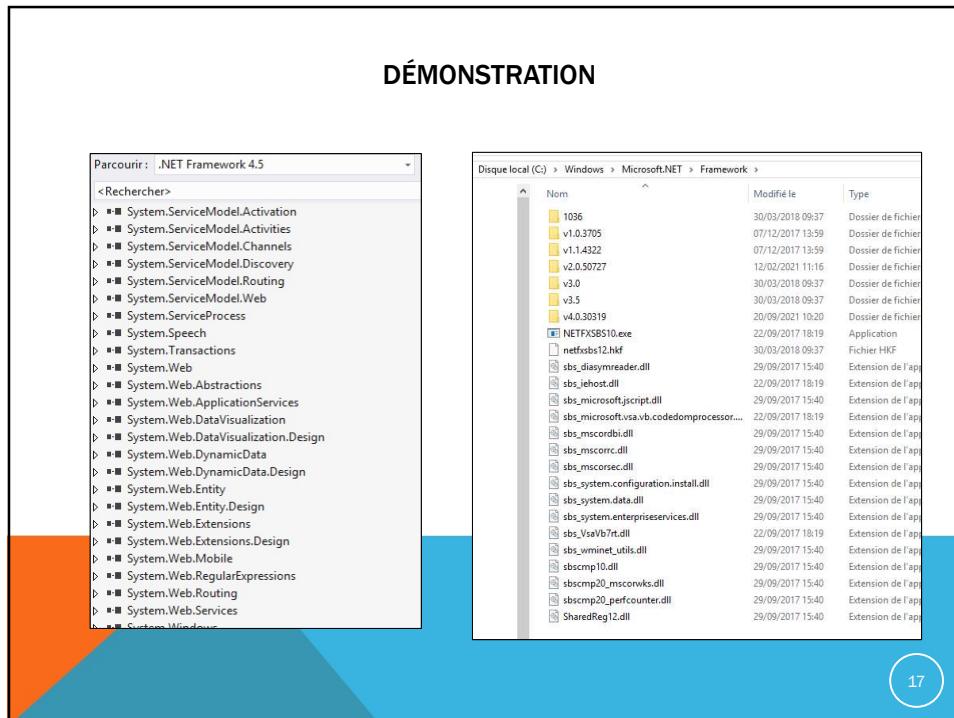
Le Framework .NET

- Ensembles de bibliothèques de code réutilisable.
- Organisation physique en « Assembly »
- Organisation fonctionnel en « Namespaces »

REM : Les assemblies se trouvent dans le répertoire :
C:\WINDOWS\Microsoft.NET

16

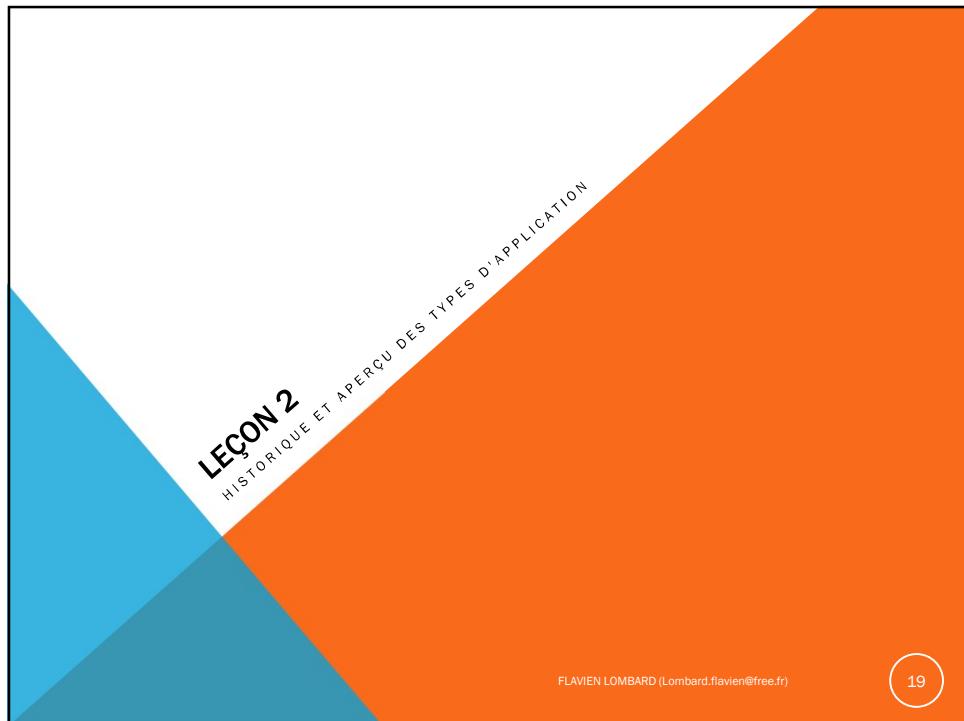
C#, développer en .NET avec Visual Studio



17



18



C#, développer en .NET avec Visual Studio

LA PLATEFORME .NET

Historique du Framework

Version	CLR	Sortie	Livré avec Visual Studio	Préinstallé avec Windows	Comprend
1.0	1.0	13 février 2002	.NET 2002	NC Client NC Serveur	
1.1	1.1	24 avril 2003	.NET 2003	NC Client 2003	NC
2.0		7 novembre 2005	2005	NC Client 2003 R2	
3.0		6 novembre 2006	NC	Vista NC	2.0
3.5	2.0	19 novembre 2007	2008	NC Client NC Serveur	3.0 SP1 (2.0 SP1)
3.5.1		4 février 2008	NC	NC	
3.5 SP1		11 août 2008	2008 SP1	NC Client NC Serveur	
3.5.1 SP1		22 juillet 2009	NC	7	2008 R2
4.0		12 avril 2010	2010	NC Client NC Serveur	3.0 SP2 (2.0 SP2)
4.5		15 août 2012	2012	8	2012
4.5.1		17 octobre 2013	2013	8.1	2012 R2
4.5.2		5 mai 2014	NC	NC Client NC Serveur	
4.6		20 juillet 2015	2015	10 v1507	NC Client NC Serveur
4.6.1	4.0	17 novembre 2015	2015 U1	10 v1511	NC Client NC Serveur
4.6.2		2 août 2016	NC	10 v1607	2016
4.7		5 avril 2017	2017 v15.3	10 v1703	NC Client NC Serveur
4.7.1		17 octobre 2017	2017 v15.5	10 v1709	v1709
4.7.2		30 avril 2018	NC	10 v1803	v1803
4.8		18 avril 2019	NC	10 v1903	2019

(mise à jour sur place)

En parallèle, Xamarin a créé le projet Mono iso fonctionnel aux versions de .NET

MICROSOFT a créé .NET Standard dont l'implémentation est .NET Core

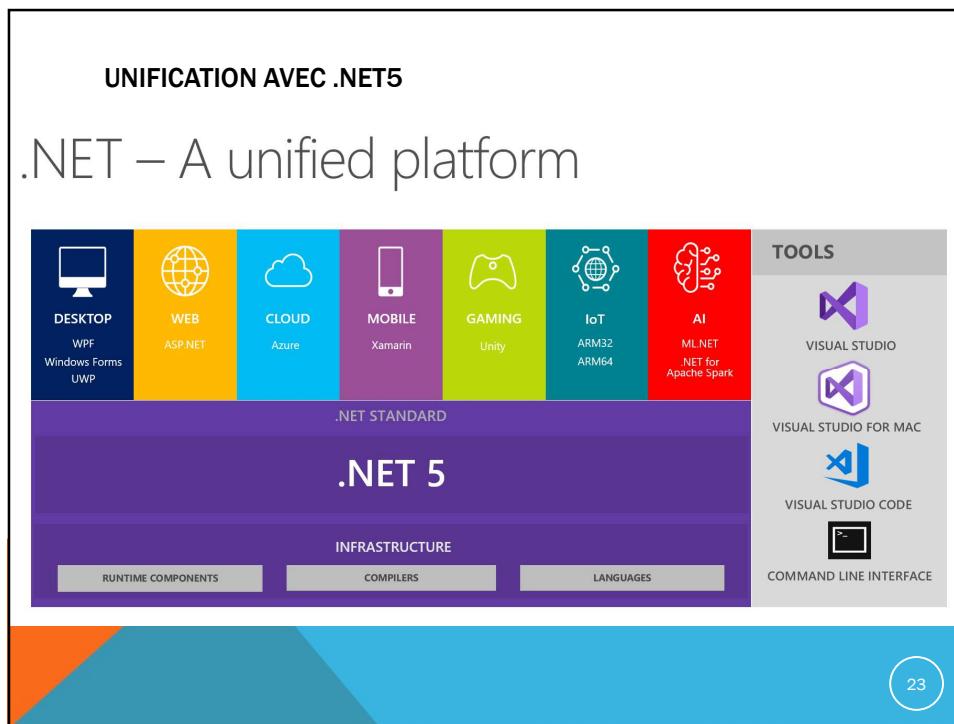
21

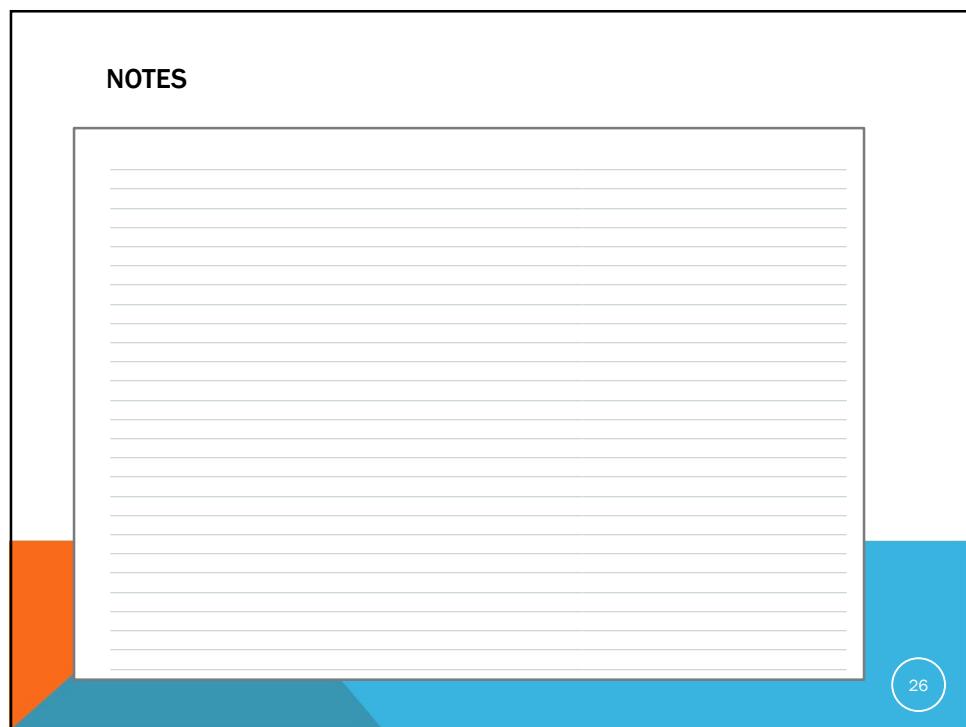
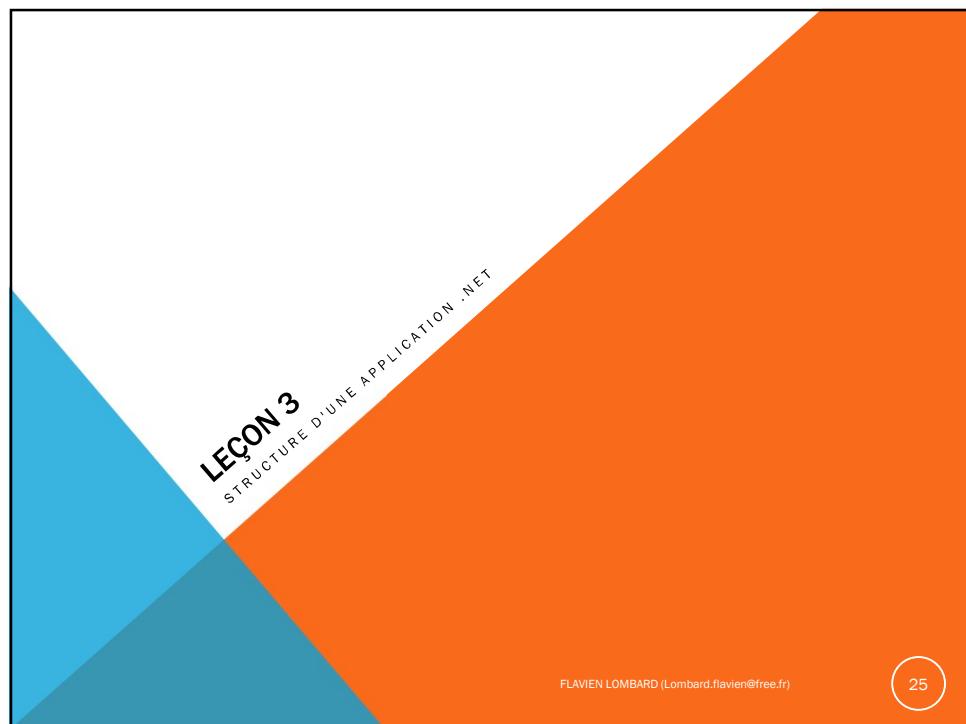
ETAT DES LIEUX

ASP.NET 4.x Desktop	ASP.NET Core 1.x
Web Form MVC WebAPI	Windows Form WPF
C#/VB Compiler	
.Net Framework	.Net Core
FCL/BCL (<i>System, Web, System.Data, System.Xml</i>) Net Framework Class Library	CoreFX (<i>System.Collections, System.IO, System.Xml</i>) Net Core Class Library, NuGetPackages
Common Language Runtime(CLR) <i>JIT Compiler/GC/CAS</i>	Common Language Runtime(CoreCLR) <i>RyuJIT Compiler(for 64-bit systems)/GC/SIMD</i>
Operating System	Operating System

22

C#, développer en .NET avec Visual Studio





LES DIFFÉRENTS TYPES DE LANGAGES

- C++, Pascal, C sont des langages natifs
- JavaScript, VBA.. sont des langages interprétés
- Les langages .NET et Java sont managés

27

COMPARAISON DES LANGAGES

- Les langages sont compatibles avec la CLI (*Common language Infrastructure*) ,s'ils respectent :
 - la CTS (*Common type system*)
 - la CLS (*Common Language Spécification*)
- Environ 30 langages actifs compatibles .NET.

28

COMPARAISON DES LANGAGES

VB .NET

Le langage historique de Microsoft

- Très verbeux
- insensible à la casse
- Prise en main facile pour les utilisateurs de VB6 et ou VBA
- Moins « productif » que c#
- Certaines classes ont été faites spécifiquement pour VB (même si elles peuvent être utilisées en c#) *Microsoft.VisualBasic*
- Moins sécurisé que C#

29

COMPARAISON DES LANGAGES

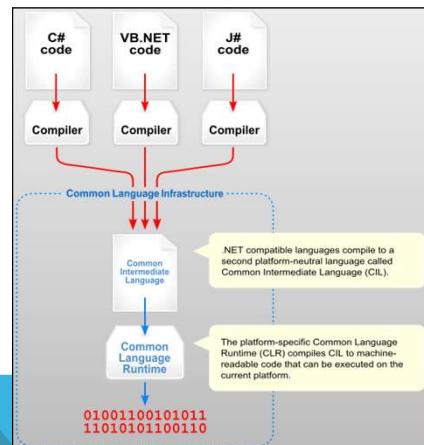
C#

Le langage phare de Microsoft

- Même syntaxe que Java ou C++
- Langage le plus populaire de la plateforme .NET
- Langage jeune donc non soumis aux problèmes de rétrocompatibilité lors de sa création.
- Productivité maximale car peu verbeux
- Plus difficile à apprendre.
- Très sécurisé

30

FINALEMENT, C# OU VB.NET ?



31

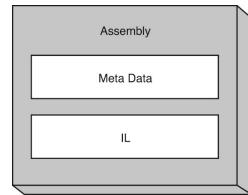
FINALEMENT, C# OU VB.NET ?

- 2 langages respectant le **CLS** (Common Langage Spécification)
- Compilation du code en **CIL** (Common Intermediate Langage)
- Utilisation de la CLR pour la compilation à la volé.
- VB.NET et C# utilisent .NET et ne sont donc pas limité en terme de fonctionnalités.

32

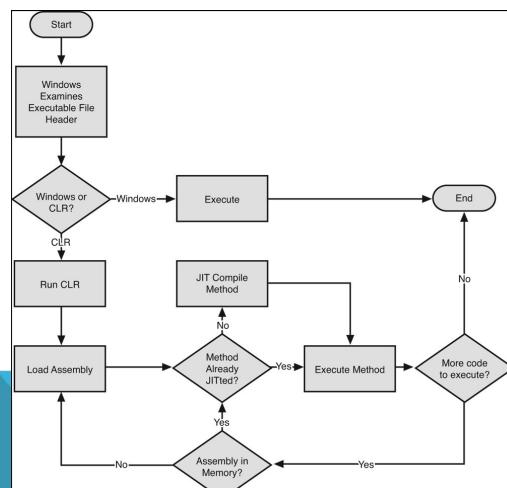
LE RÔLE DE LA CLR

- Le rôle de la CLR**
- sécurité
 - Interopérabilité
 - Gestion de la mémoire (Ramasse miettes)
 - BCL (Base Class Library)
 - Exécution du code
 - Gestion des métadonnées (Réflexion)



33

PRINCIPE D'EXÉCUTION DE LA CLR



34

QUESTIONS ? REMARQUES ?



35

36

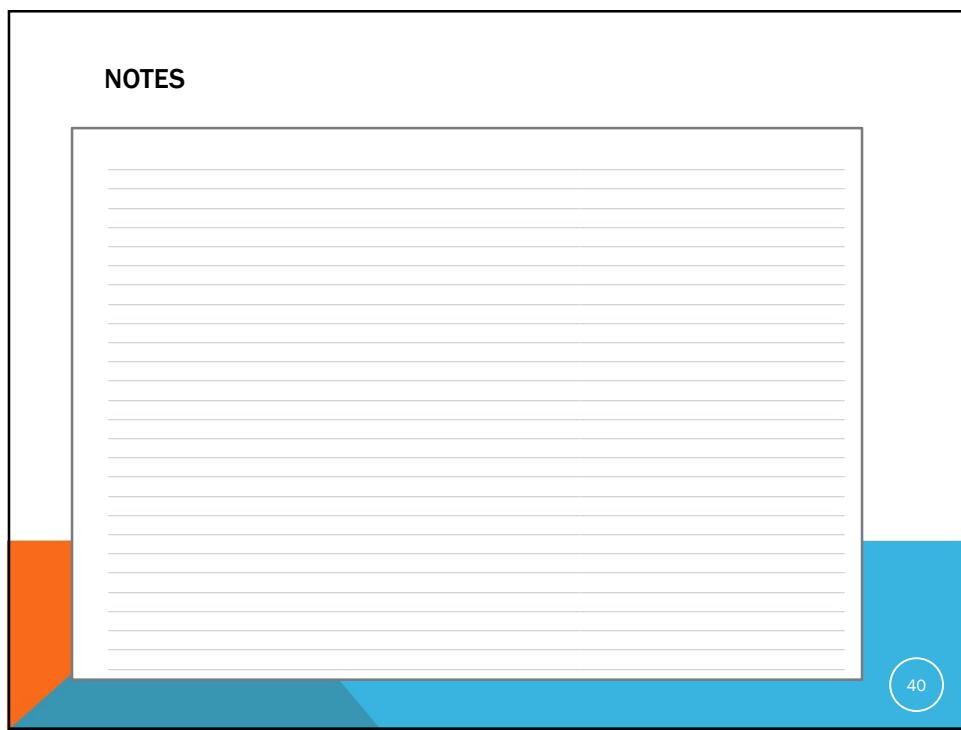
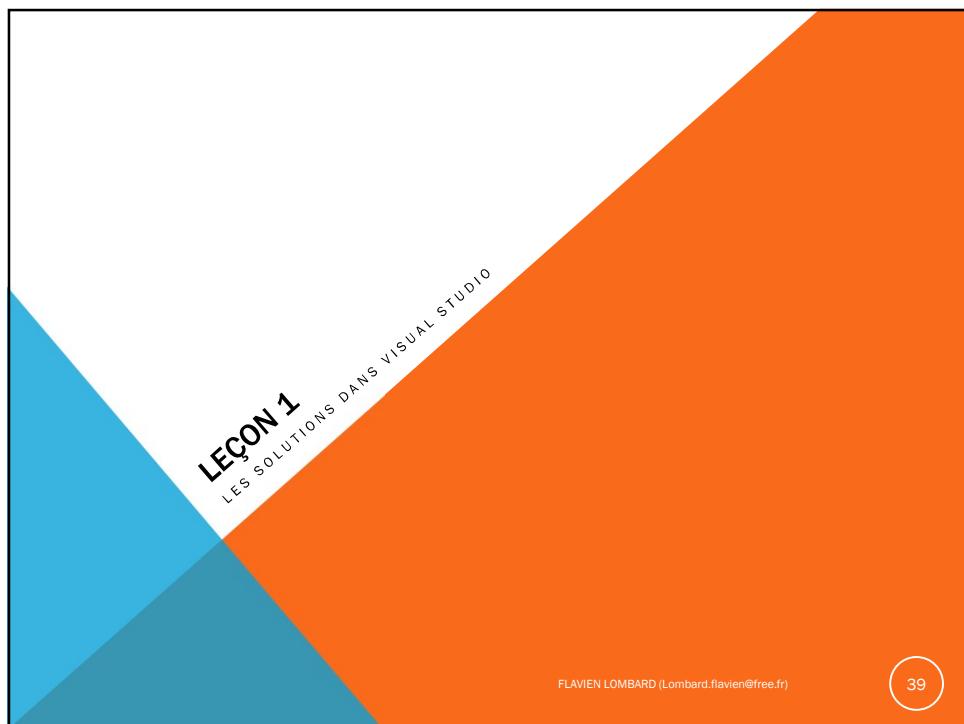


NOTES

Handwriting practice lines

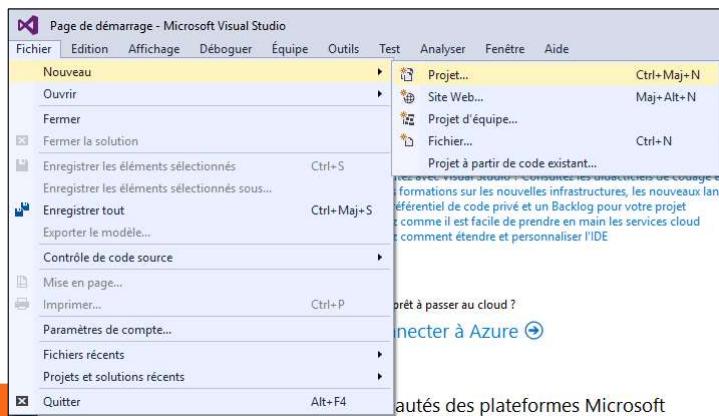
38

C#, développer en .NET avec Visual Studio



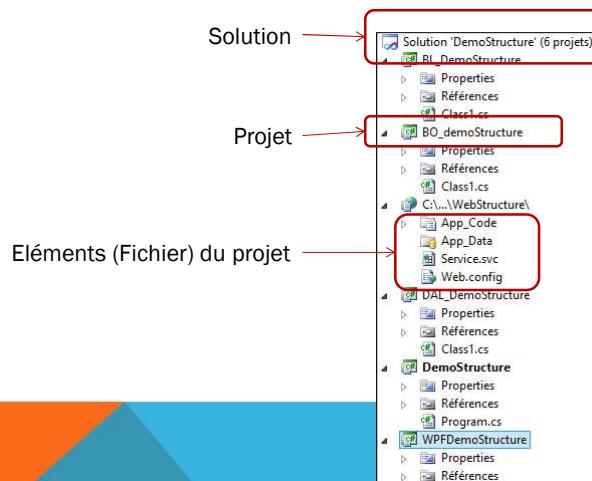
C#, développer en .NET avec Visual Studio

CRÉATION ET OUVERTURE D'UNE SOLUTION



41

STRUCTURE D'UNE SOLUTION



Eléments (Fichier) du projet

42

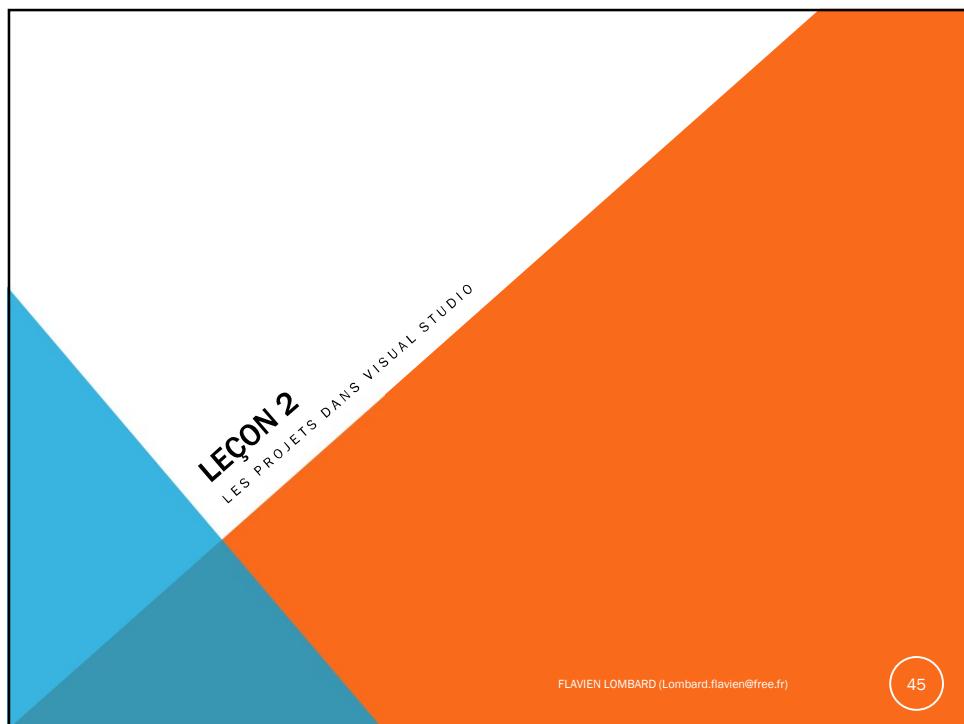
DÉMONSTRATION



43

44

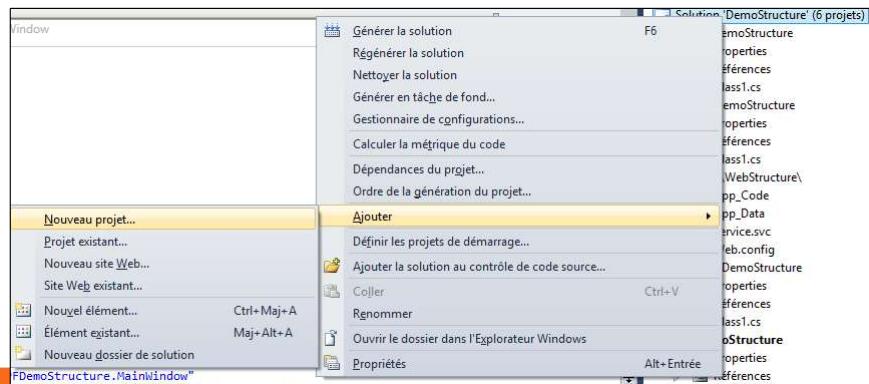
C#, développer en .NET avec Visual Studio



This image shows a template for taking notes. It features a large central area with horizontal ruling lines for writing. On the left side, there is a vertical column with a red header labeled "NOTES". The bottom of the page has a decorative border composed of red and blue triangles. In the bottom right corner, there is a small circular icon containing the number "46".

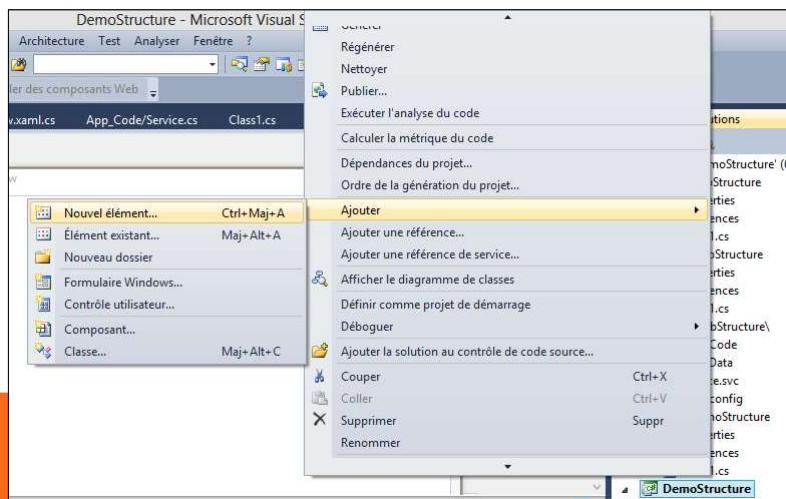
C#, développer en .NET avec Visual Studio

AJOUT D'UN PROJET À LA SOLUTION



47

AJOUT D'UN ÉLÉMENT AU PROJET



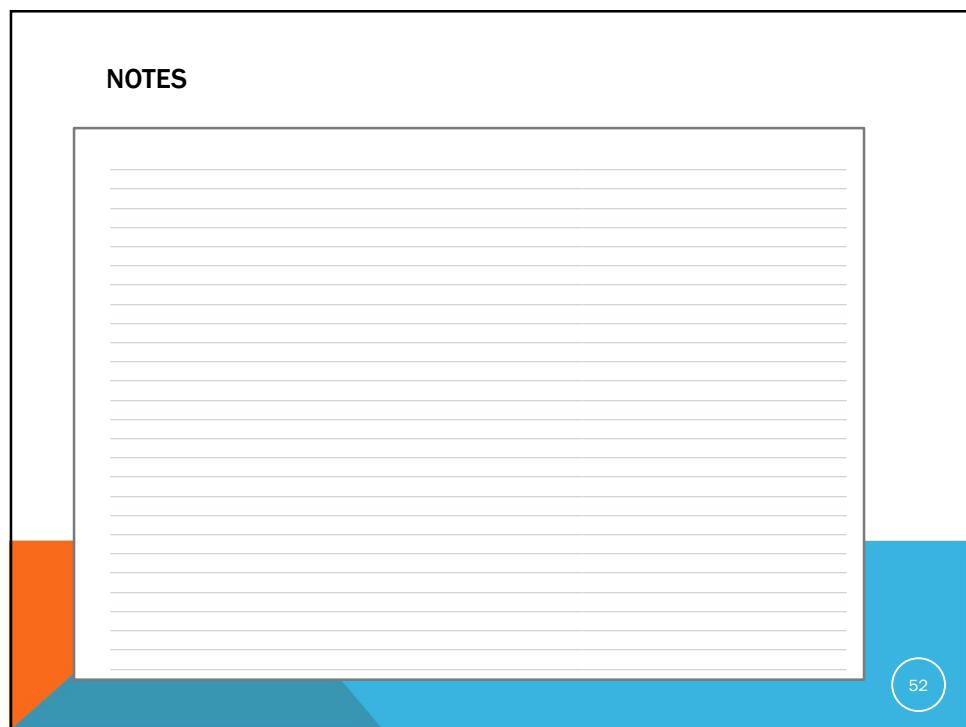
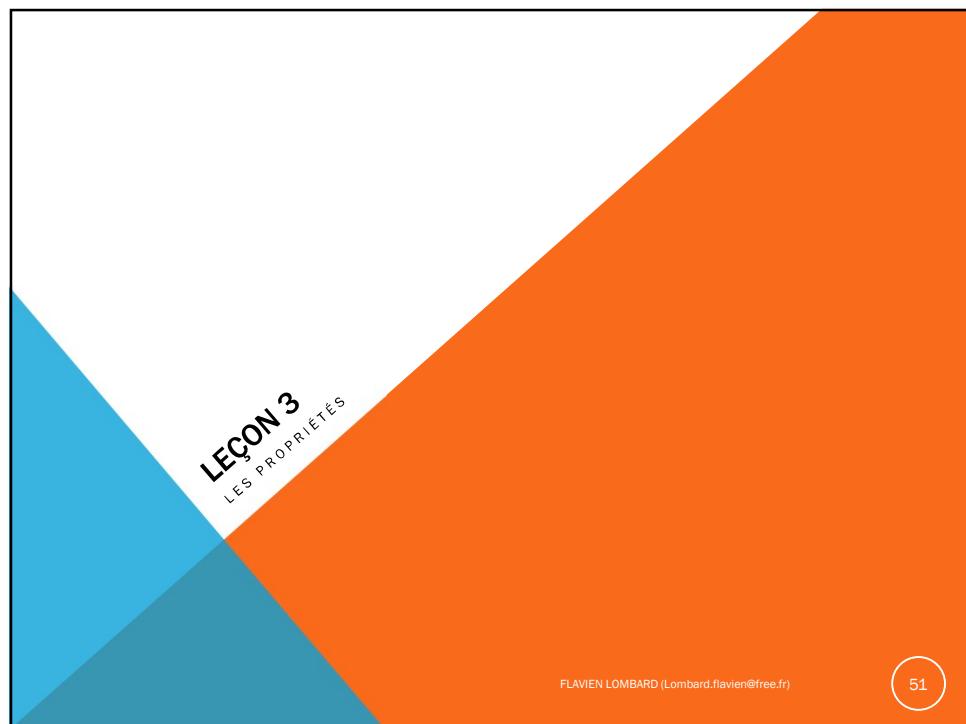
48

DÉMONSTRATION



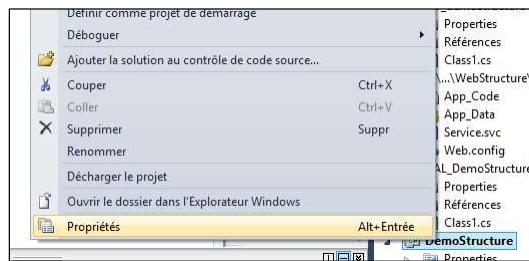
49

50



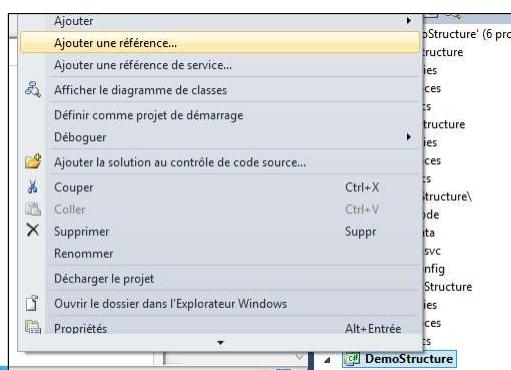
C#, développer en .NET avec Visual Studio

PROPRIÉTÉ D'UN PROJET



53

AJOUT DE RÉFÉRENCES



54

DÉMONSTRATION



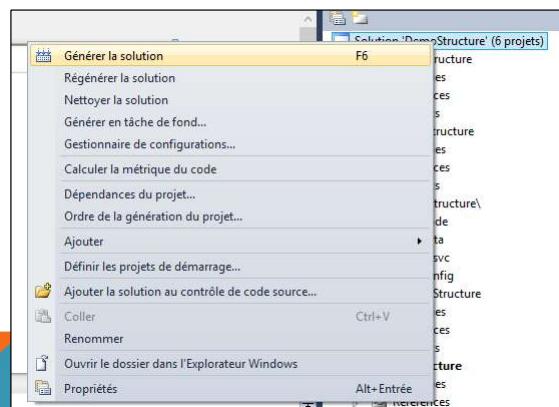
55

56



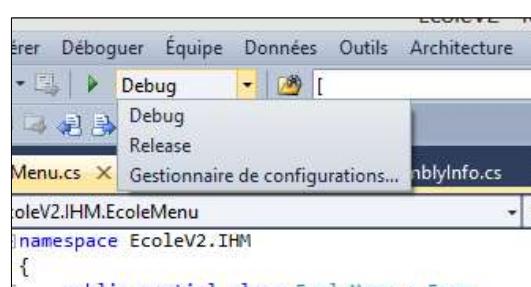
The image displays a template for taking notes. It features a large central area with horizontal ruling lines for writing. This central area is framed by a white border. The entire template is set against a background with a blue base and orange side panels. In the top left corner, the word "NOTES" is printed in capital letters. In the bottom right corner, there is a small circular icon containing the number "58".

RÉGÉNÉRATION ET EXÉCUTION



59

MODES D'EXÉCUTION

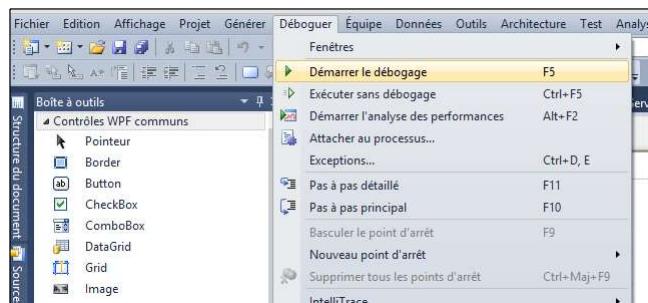


Mode débugge : Non optimisé, mais débogage facilité
Mode Release : Optimisé, mais pas de débogage

60

C#, développer en .NET avec Visual Studio

LE DÉBOGAGE



61

EXERCICES

Le premier exercice consiste à créer une première application console demandant à l'utilisateur son nom grâce à la console.

L'application devra alors répondre "Bonjour Mr [nom]"



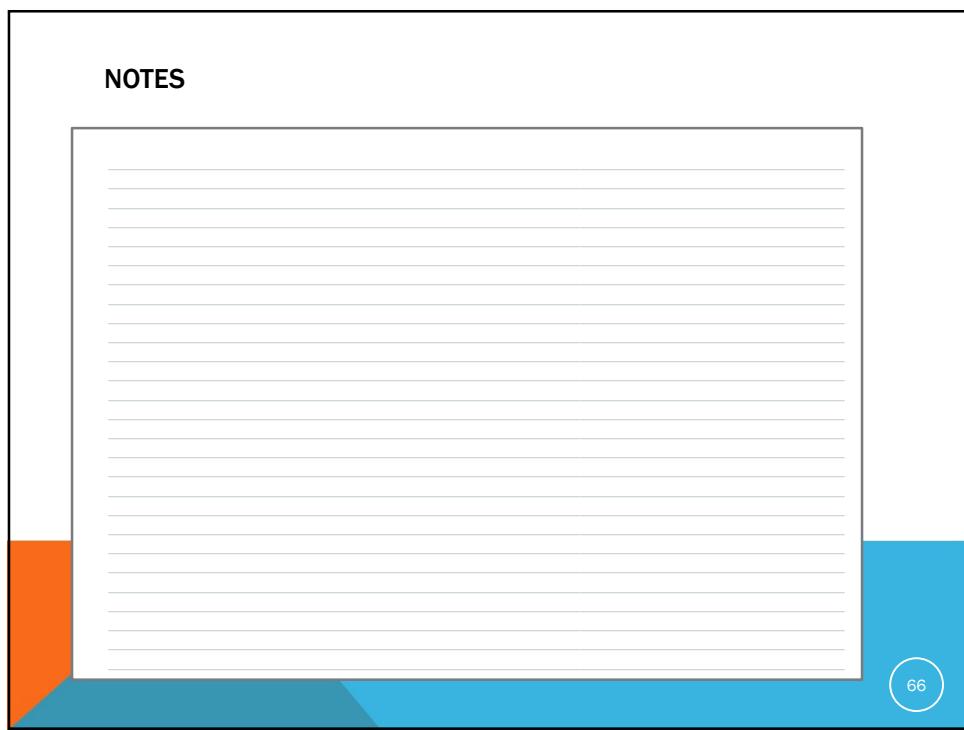
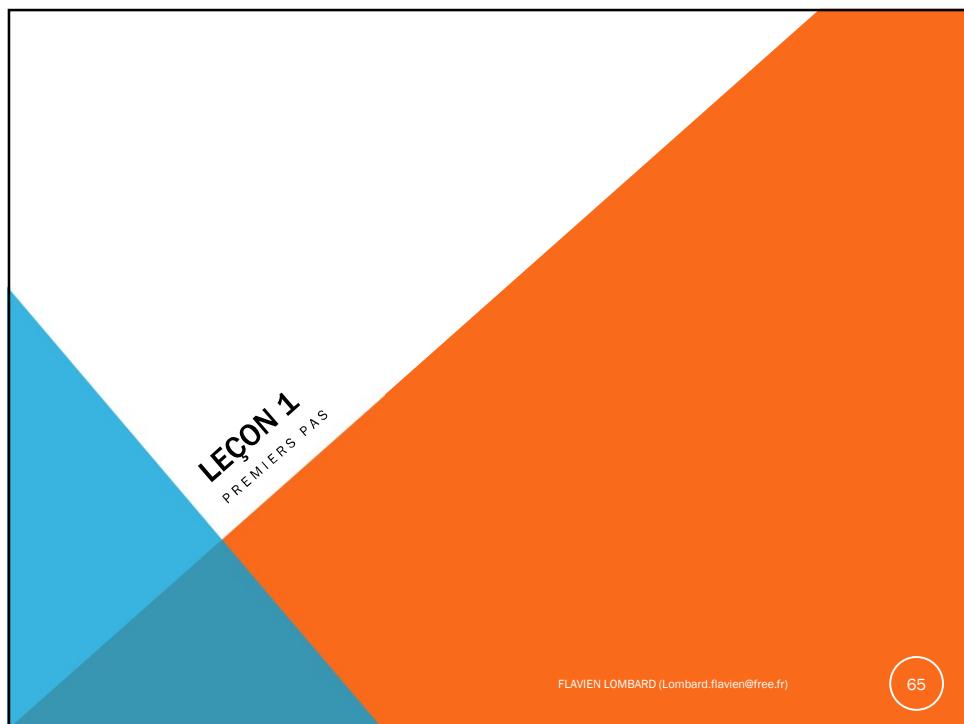
62



NOTES

Handwriting practice lines for notes.

64



PREMIERS PAS

- Chaque type, variable, classe sont des objets.
- Toutes les méthodes appartiennent obligatoirement à une classe.
- Le point d'entrée d'un programme porte le nom Main

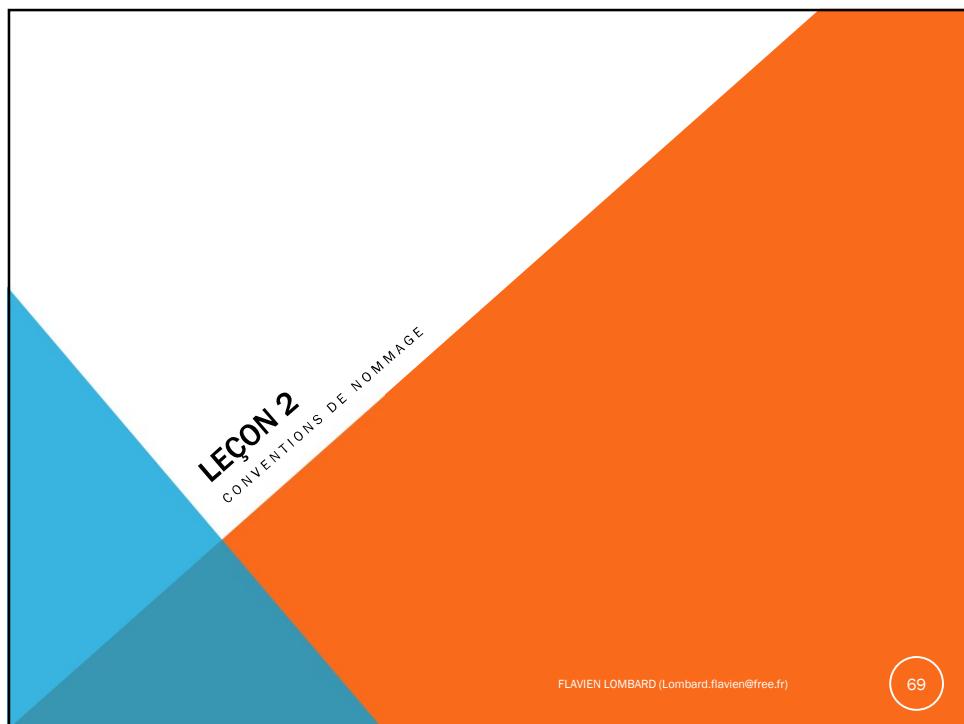
67

PREMIERS PAS

```
using System;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");
        }
    }
}
```

68



The image shows a template for taking notes. It consists of a large white rectangular area with horizontal ruling lines, set against a background of orange and blue triangles. The word "NOTES" is printed at the top left of the white area. At the bottom right, there is a small white circle containing the number "70".

LES CONVENTIONS DE NOMMAGE

- C# est sensible à la casse, VB est insensible à la casse.
- Les noms de méthode et de classe et des namespaces commencent par une majuscule
`Program, Module1, Main`
- Les paramètres et les variables locales sont en minuscules.
- Les noms composés de plusieurs mots sont en majuscules
`DemarrerVoiture(int quantiteCarburant)`

71

LES CONVENTIONS DE NOMMAGE

Conventions moins formelles

Les variables privées peuvent commencer par un Under score,
`private int _voiture;`

Les collections d'objet sont aux pluriels.

```
public Voiture maVoiture  
public Voiture[] mesVoitures
```

Les variables statiques peuvent commencer par un s suivi d'un Under score,

```
int s_ageLegal;
```

72

LES ESPACES DE NOM

- Organisation des classes du Framework et de votre projet
- Possibilité de créer deux classes de même nom, mais de contexte différent

Convention de nommage :

<NomSociete>.<NomApplication>.<NomModule>.<BriqueApplicative>.<...>

Exemple :

Contoso.SuperCRM.GestionClient.BusinessLayer

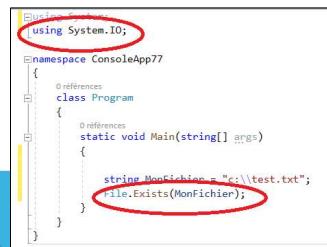
73

LES ESPACES DE NOM

Soit en utilisant la syntaxe de nom entièrement qualifié.

```
System.IO.File.Exists(MonFichier)
```

Utilisation dans le code en utilisant les mot clés **using**.



```
using System.IO;
namespace ConsoleApp77
{
    class Program
    {
        static void Main(string[] args)
        {
            string MonFichier = "c:\\test.txt";
            file.Exists(MonFichier);
        }
    }
}
```

74

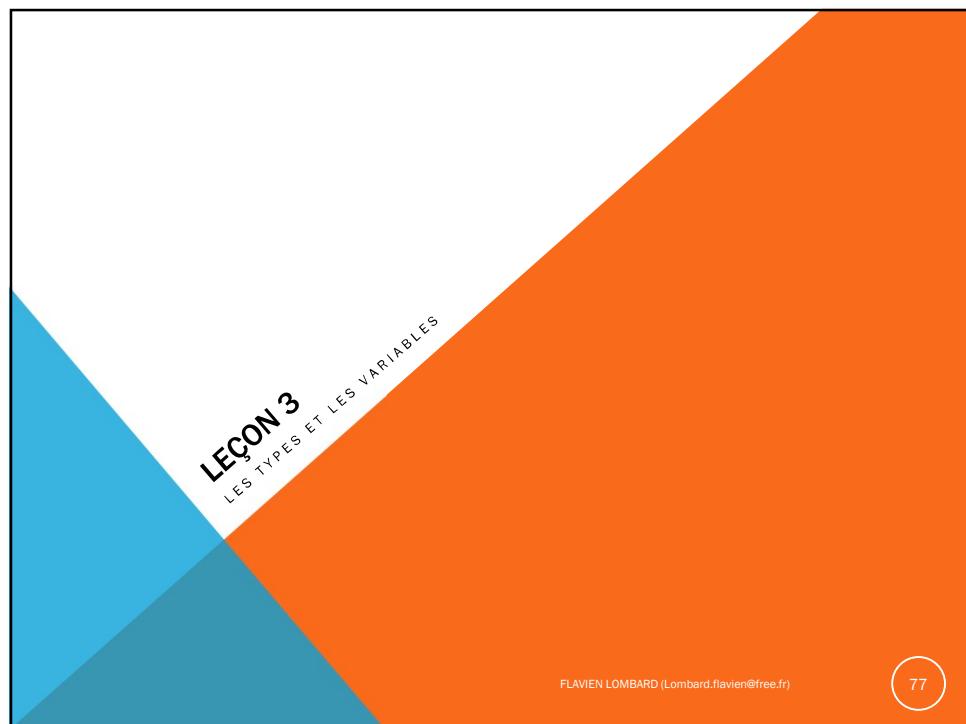
LES ESPACES DE NOM

Le Mot clé `global` permet de remonter à la racine des espaces de nom.

```
var regkey = global::Microsoft.Win32.Registry.LocalMachine;
```

75

76



NOTES

A large rectangular area labeled "NOTES" is provided for writing. It features horizontal ruling lines on a white background. The entire notes section is framed by a thick black border. The corners of the frame are colored: the top-left is orange, the top-right is blue, the bottom-left is orange, and the bottom-right is blue. A small circular icon containing the number "78" is located at the bottom right corner of the notes area.

LES IDENTIFICATEURS

Ne peuvent pas commencer par un chiffre.
Ne peuvent pas contenir d'espace.
Peuvent contenir des chiffres.
Peuvent commencer ou contenir un Under score.
Peuvent contenir des accents.

[String prénom;](#)

Rem : En c#, il est possible d'utiliser un mot réservé comme identificateur.
▪ Ex : int @int = 5;

79

TYPES VALEUR ET TYPES RÉFÉRENCE

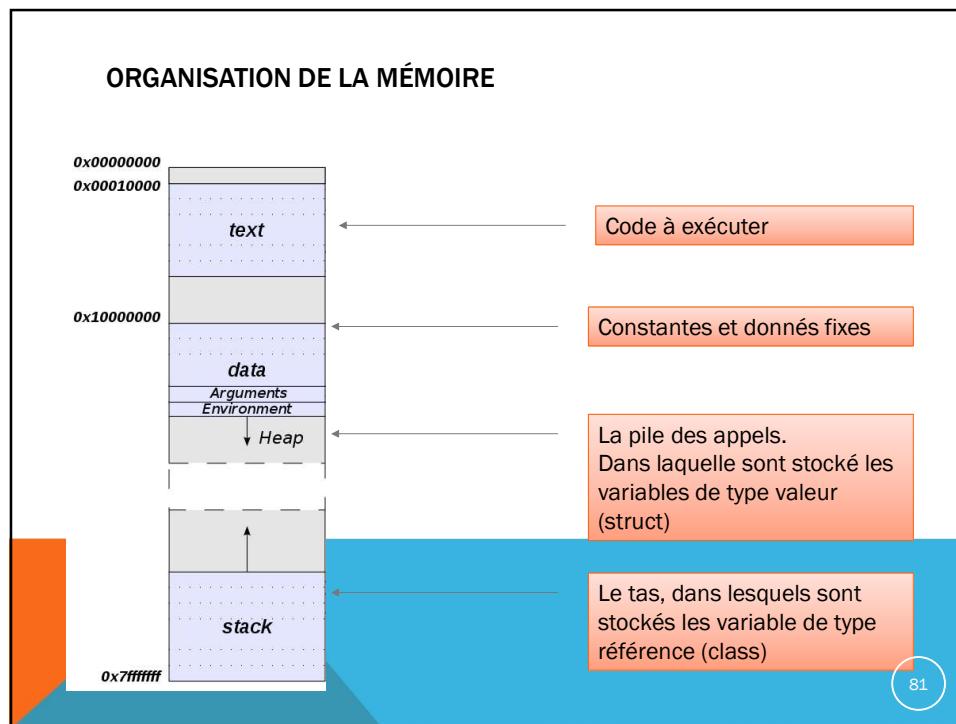
Les types valeurs sont créés directement sur la pile.

[System.Int32](#)
[System.DateTime](#)
[System.Drawing.Point](#)

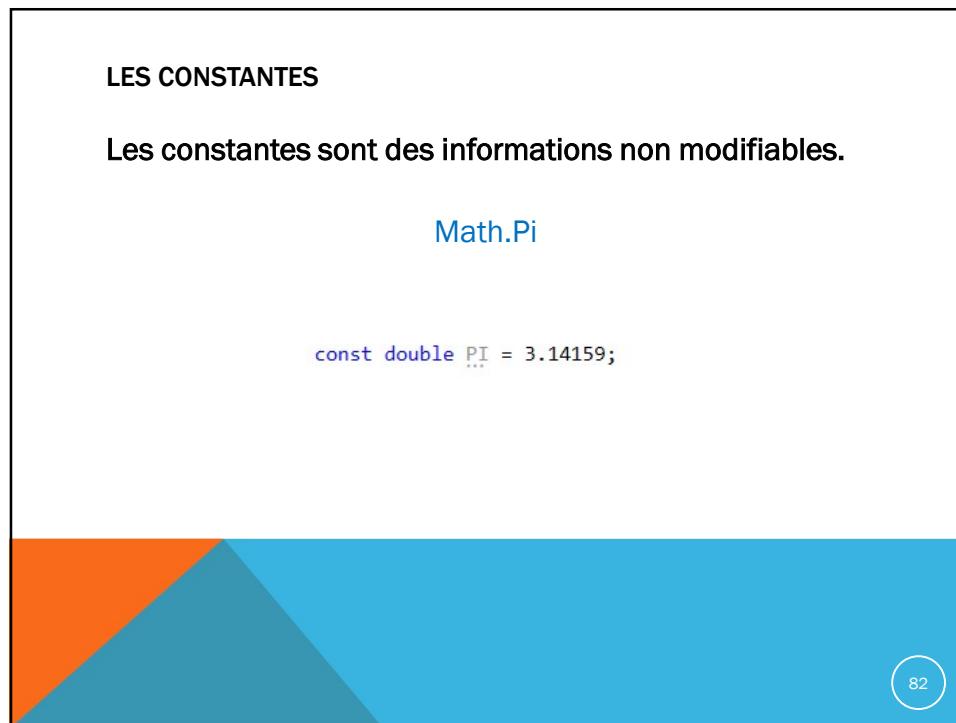
Les types références n'ont qu'une référence dans la pile.

[System.IO.File](#)
[System.Configuration.ApplicationSettingsBase](#)

80



81



82

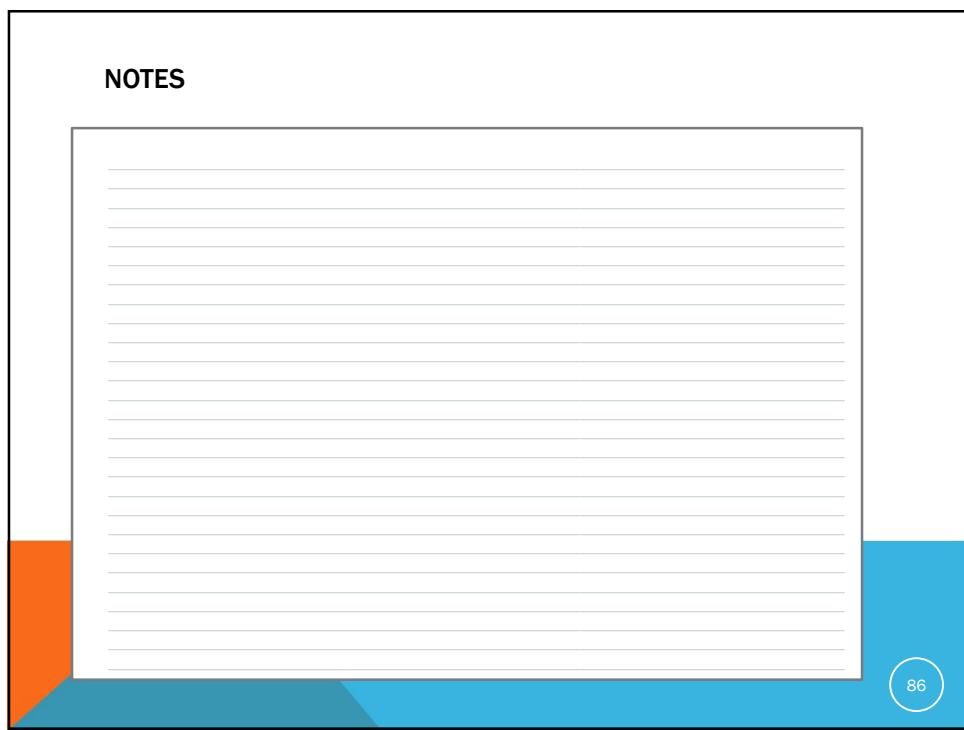
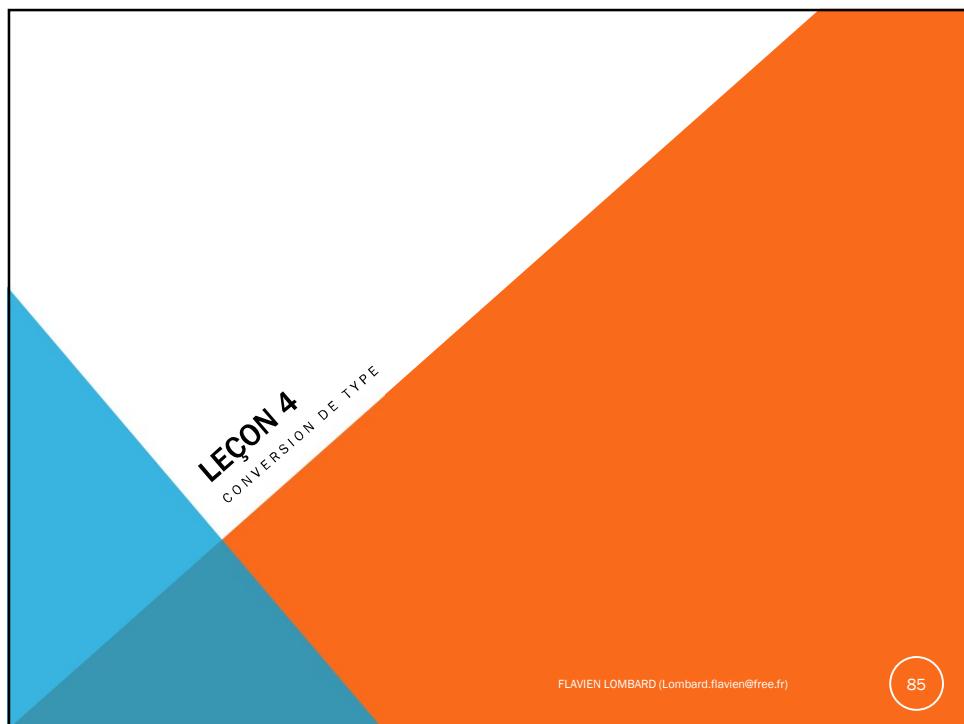
LES CONSTANTES

Les constantes sont des données en mémoire et doivent respecter leurs types

```
const short pi = 3.14159265358979323846;
```

83

84



LE TRANSTYPAGE

Il existe 2 types de transtypage en C#:

Implicite:

```
int a = 5;  
long b = a; // transtypage implicite OK car long est plus grand que int
```

Explicite:

```
double a = 5.5;  
int b = (int)a; // transtypage explicite car perte d'information
```

87

TRANSTYPAGE

Transtypage automatique en VB...

```
Dim l As Long = 0  
  
Dim a As Integer = 5  
Dim b As Integer = 10  
  
l = a + b
```

```
Dim mondouble As Double = 10.345678  
  
Dim monentier As Integer = mondouble
```

88

CONVERSION DE CHAINE DE CARACTÈRE

Conversion d'un objet en chaîne de caractère

```
double d = 13.34345;  
string s = d.ToString();
```

Conversion d'une chaîne en objet

```
string str = "34.56";  
double d = Convert.ToDouble(str);
```

89

CONVERSION DE CHAINE DE CARACTÈRE

Formatage

```
DateTime madate = new DateTime(2022, 1, 11);  
string str = madate.ToString("dd/MM/yyyy");
```

90

INFÉRENCE DE TYPE

Mot clé `var` en c#

Le compilateur déduit le type de la variable grâce à son affectation.

```
var mavariable = 13.89;  
string str = mavariable.ToString();
```

Utilisation de type anonyme

```
var mavoiture = new {modele="Saxo", Année=1990,marque="Citroen"};  
string str = mavoiture.ToString();
```

91

LES STRUCTURES

Equivalent aux classes mais elles sont stockées dans la pile (type valeur)

Pas d'héritage possible

Impossible d'affecter null à un objet de type structure

Un constructeur doit forcément affecter toutes les variables de la structure

```
struct Simple  
{  
    public int Position;  
    public bool Exists;  
    public double LastValue;  
}
```

92

LES ÉNUMÉRÉS

Liste de constantes typées

Facilité d'utilisation

```
3 références
enum JourDeLaSemaine
{
    Lundi,
    Mardi,
    Mercredi,
    Jeudi,
    Vendredi,
    Samedi,
    Dimanche
}
```

```
JourDeLaSemaine jour = JourDeLaSemaine.Lundi;

if (jour == JourDeLaSemaine.Lundi)
{
    Console.WriteLine("Nous sommes un Lundi");
}
```

93

LA CONSOLE

L'objet `Console` est très utile pour le débogage ou pour les traces.

3 propriétés :

- In: Lecture sur le flux d'entrée
- Out : Ecriture sur le flux de sortie
- Error: Ecriture sur le flux d'erreurs

94

QUESTIONS ? REMARQUES ?



95

EXERCICES

En utilisant la console, demander à l'utilisateur de créer un article composé d'un nom, d'un prix unitaire et d'une quantité.

Afficher le nom de l'article et le prix total (prix * qte)



96



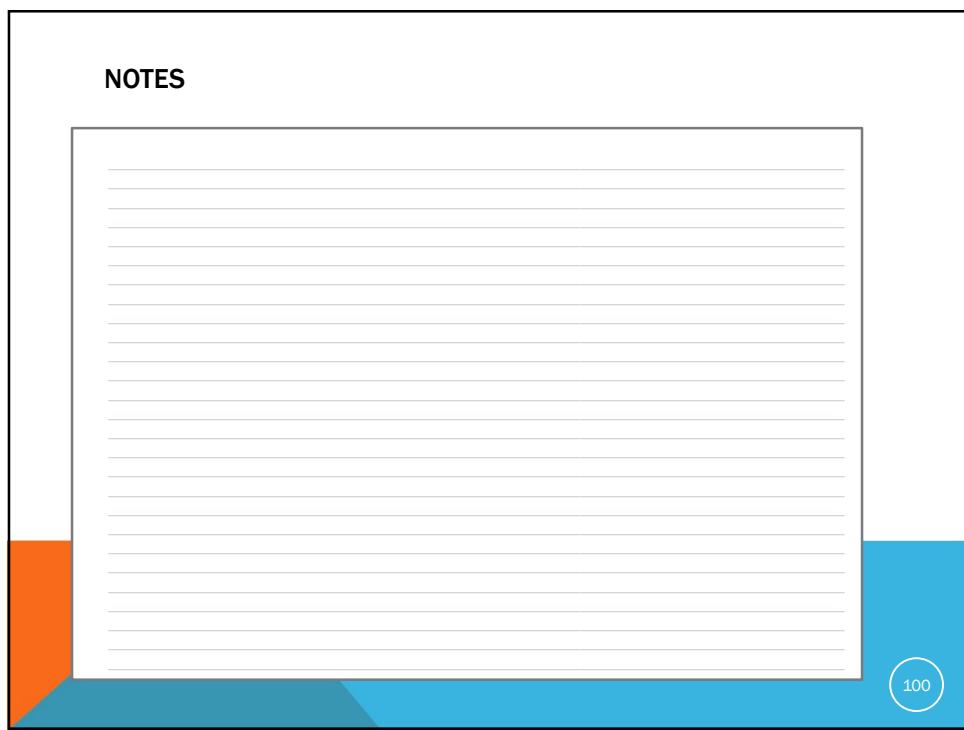
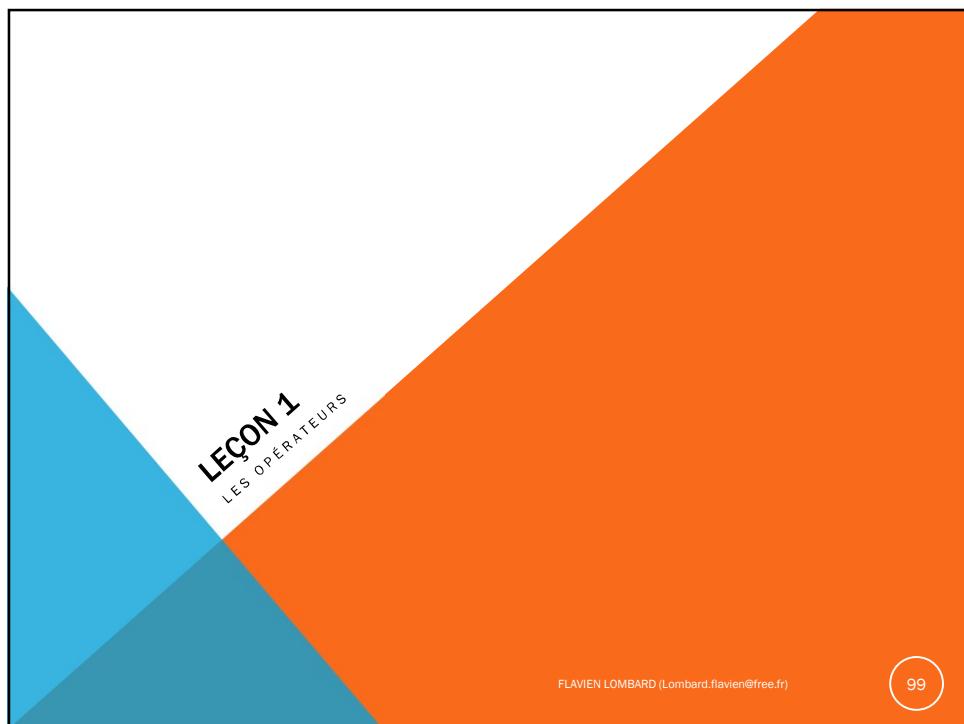
NOTES

A white rectangular area representing a lined notebook page, with horizontal ruling lines. It is positioned in the center of a larger frame.

An orange square block located at the bottom-left corner of the main frame.

A blue square block located at the bottom-right corner of the main frame.

98



OPÉRATEURS

Opérateurs arithmétiques

Type du résultat définit de manière à ne perdre le moins d'information possible

int / double -> double

Opérateurs de comparaison

Opérateurs : <, >, <=, >=, ==, !=

Opérateurs binaires

Travaillent sur la représentation binaire des valeurs

Opérateurs : &, |, ~, >>, <<, ^

101

OPERATEURS

Opérateurs booléens

Opérateurs : &&, ||, !

Opérateur ternaire

```
s2 = (s == "") ? "chaîne vide" : s + " la suite de s2";
```

Pré/pos incrémentation/décrémentation

i++, ++i, i--, --i

102

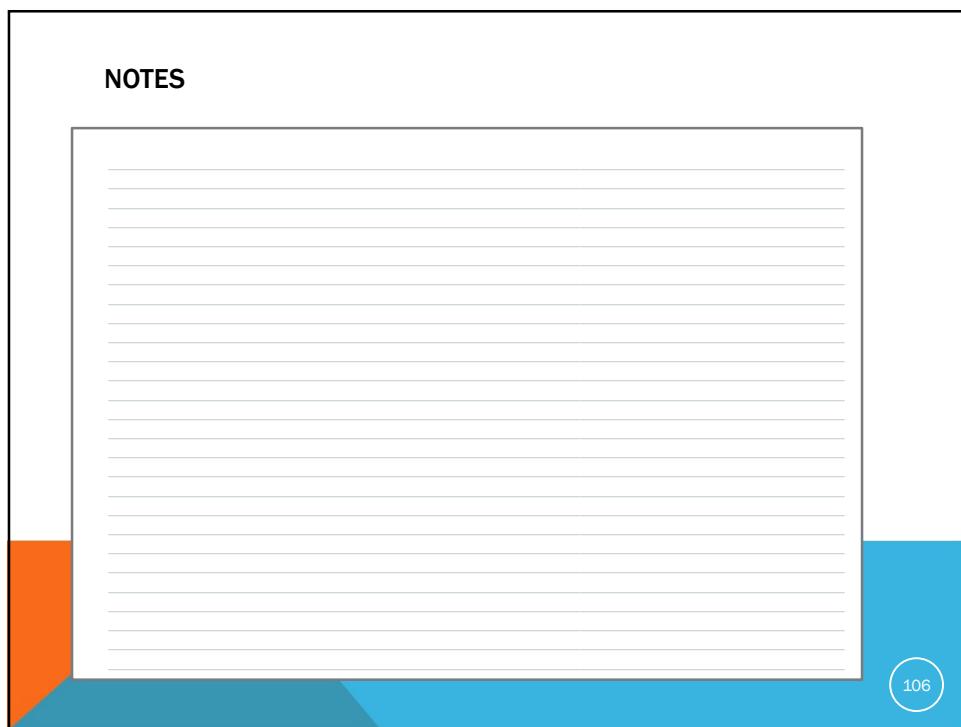
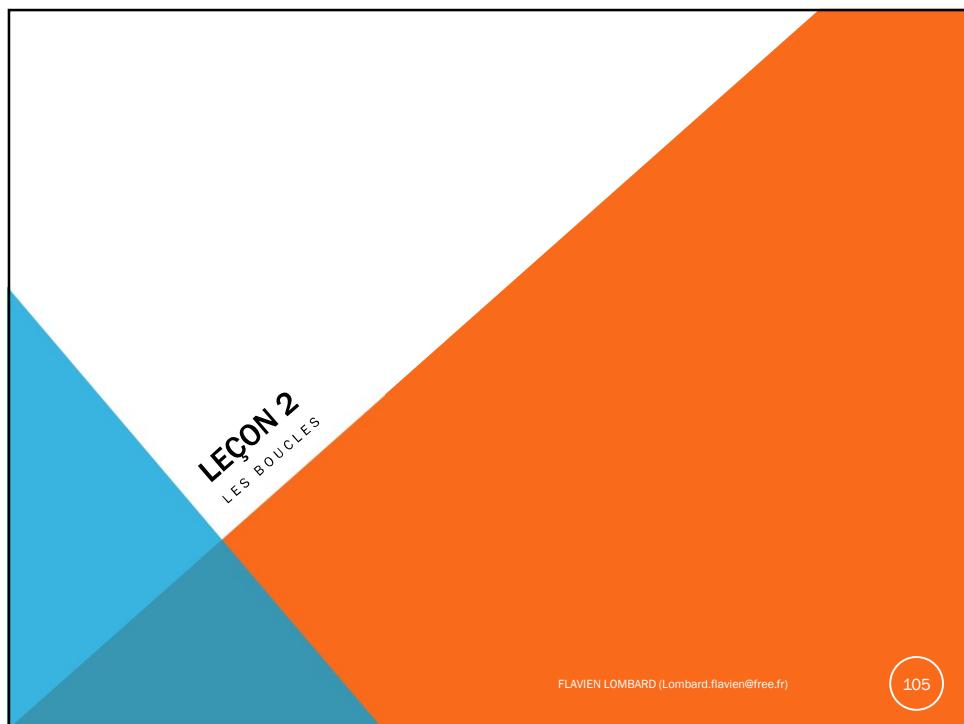
LES OPERATEURS

Les opérateurs sont aussi des méthodes d'objet.

```
int sum = 5+5;           // -----> 10
string strSum = "5"+ "5"; // -----> "55"
```

103

104



STRUCTURES DE BOUCLE

tant que ... faire ...

```
int i = 10;
while( i-- > 0 )
    Console.WriteLine( i );
```

faire ... tant que ...

```
int i = 10;
do
{
    Console.WriteLine( i );
} while (i-- > 0);
```

La boucle for

```
for (int i=0;i<10;i++)
{
    Console.WriteLine( i );
}
```

La boucle foreach

```
int[] tblei = new int[50];
foreach(int i in tblei)
    Console.WriteLine( i );
```

107

STRUCTURE DE BOUCLE

Sortie anticipée d'une boucle grâce aux mots clés
break

Passage direct à la prochaine itération grâce au mot clé
continue

```
for (int i=0; i<10; i++)
{
    if (i % 2 == 0) continue;
    if (i == 7) break;
    Console.WriteLine(i);
}
//1,3,5
```

Résultat : 1,3,5

108

STRUCTURE DE BOUCLE

Les labels

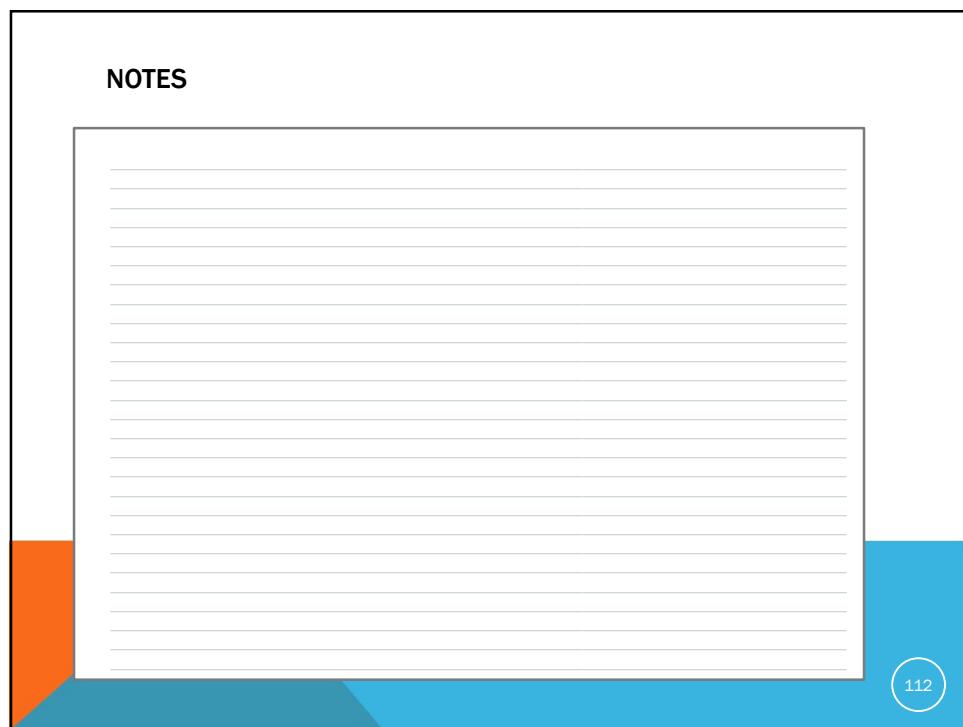
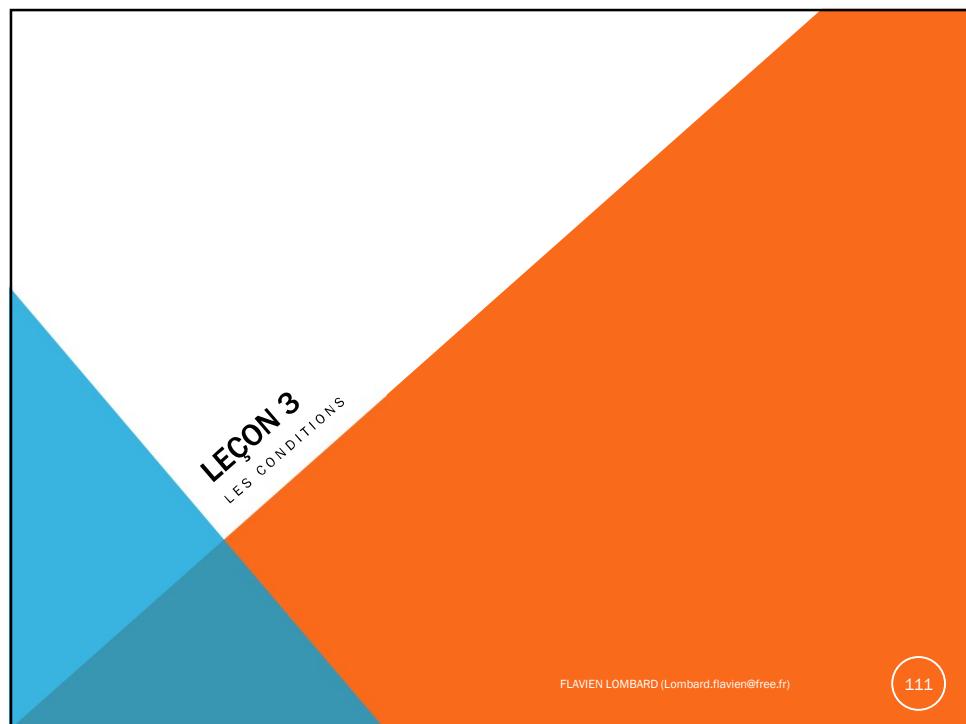
```
monlabel:  
    'Some Code'  
    GoTo monlabel
```

```
for (int x = 0; x < 100; x++)  
{  
    for (int y = 0; y < 100; y++)  
    {  
        if (x == 5 && y == 5) goto finBoucle;  
    }  
}  
finBoucle:;
```

Utilisation déconseillée car cela nuit à la lecture du code.

109

110



LES CONDITIONS

If... then ... else

```
int i=10;
if (i == 10)
{
    Console.WriteLine( i );
```

Notation ternaire

```
if (i == 10)
{
    Console.WriteLine( i );
}else
{
    Console.WriteLine( "i est différent de 10" );
```

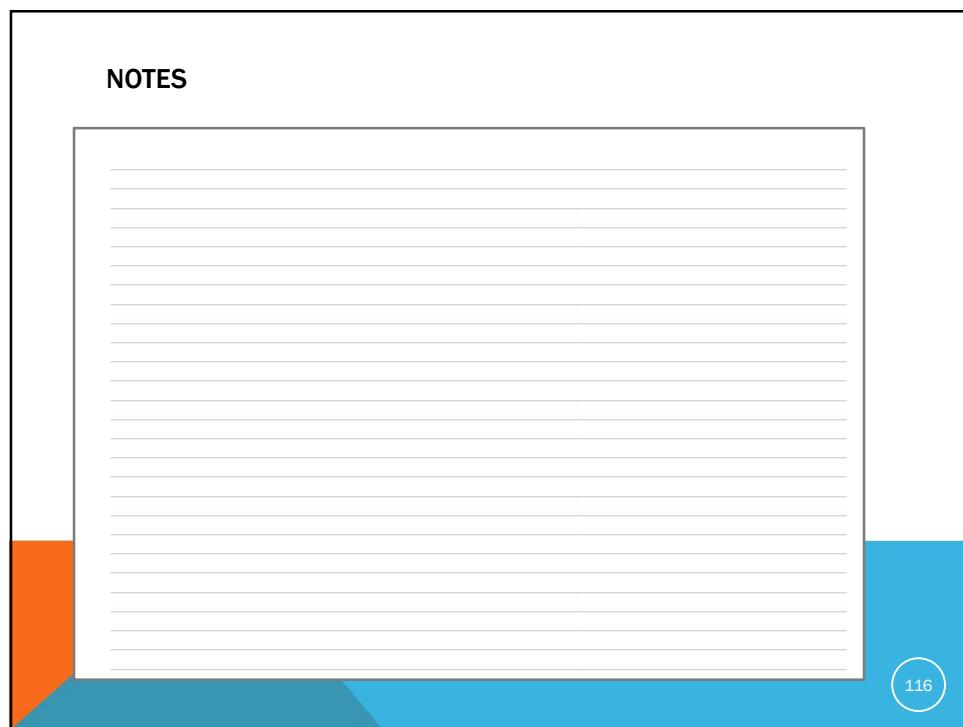
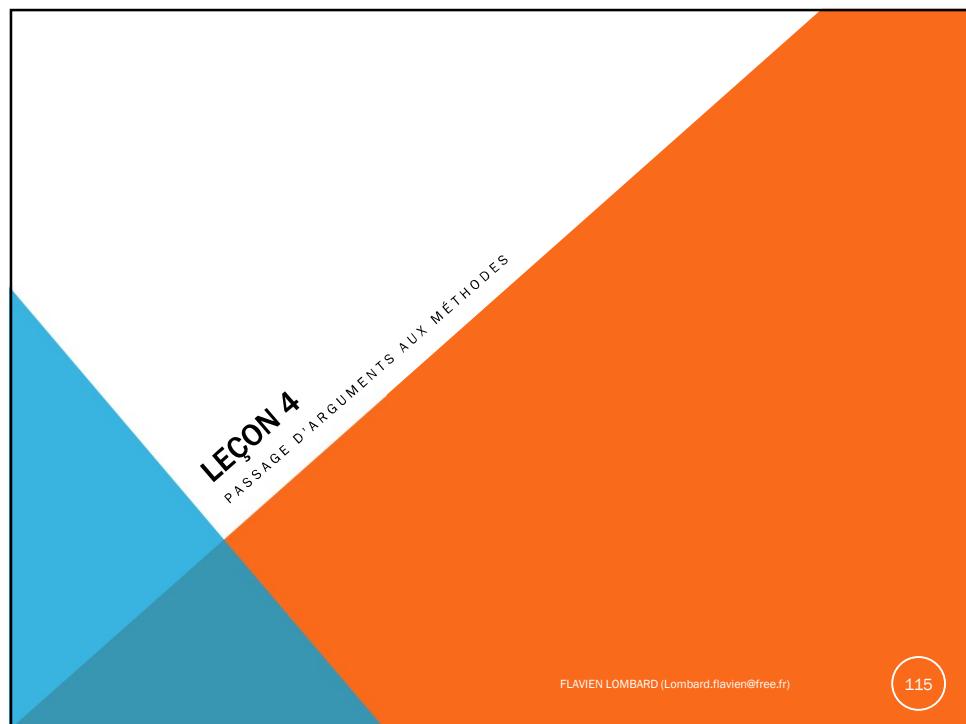
```
Console.WriteLine( (i == 10)?i.ToString():"i est différent de 10" );
```

113

CHOIX MULTIPLE : SWITCH/SELECT

```
switch( combienOnEst ) {
    case 1:
        Console.Out.WriteLine("FaireUnDemineur");
        break;
    case 2:
        Console.Out.WriteLine("JouerAuxDames");
        break;
    case 3:
        Console.Out.WriteLine("EssayerDeTrouverUnQuatrieme");
        goto case 4; // Fall through explicite nécessaire
    case 4: // Fall through implicite autorisé
    case 5:
        Console.Out.WriteLine("FaireUnPoker");
        break;
    case 22:
        Console.Out.WriteLine("FaireUnFoot");
        break;
    default:
        Console.Out.WriteLine("BoireDeLaBiere");
        break;
}
```

114



PASSAGE D'ARGUMENTS PAR VALEUR OU RÉFÉRENCE

Par défaut c'est la valeur de l'objet qui est recopié en argument de méthode

```
static void MethodeDeTest(int firstarg,Maclass secondarg)
{
    secondarg.Libelle = "tata";
    firstarg = 45;
}

static void Main(string[] args)
{
    Maclass cl = new Maclass();
    cl.Libelle = "toto";
    int param = 6;

    MethodeDeTest(param,cl);

    Console.Out.WriteLine(param);
    Console.Out.WriteLine(cl.Libelle);

    Console.ReadLine();
}
```

param = 6
cl.Libelle = toto

param vaut 6 car la valeur
est copié dans le
paramètre. La valeur
d'origine ne change pas.

117

PASSAGE D'ARGUMENTS

Mot clé ref pour un passage par référence

```
static void MethodeDeTest(ref int firstarg,ref Maclass secondarg)
{
    secondarg.Libelle = "tata";
    firstarg = 45;
}

static void Main(string[] args)
{
    Maclass cl = new Maclass();
    cl.Libelle = "toto";
    int param = 6;

    MethodeDeTest(ref param,ref cl);

    Console.Out.WriteLine(param);
    Console.Out.WriteLine(cl.Libelle);

    Console.ReadLine();
}
```

param = 45
cl.Libelle = toto

param vaut 45 car la
référence est copié dans
le paramètre. La valeur
d'origine change.

118

PASSAGE D'ARGUMENTS

Le mot clé out spécifique à c# assignation obligatoire

```
static void MethodeDeTest(out int firstarg, ref Maclass secondarg)
{
    Le paramètre de sortie 'firstarg' doit être assigné avant que le contrôle quitte la méthode actuelle
    |
    secondarg.Libelle = "tata";
}

static void Main(string[] args)
{
    Maclass cl = new Maclass();
    cl.Libelle = "toto";
    int param = 6;

    MethodeDeTest(out param, ref cl);

    Console.Out.WriteLine(param);
    Console.Out.WriteLine(cl.Libelle);

    Console.ReadLine();
}
```

119

LES ARGUMENTS VARIABLE

Arguments de type tableaux d'objets

```
static void MethodeDeTest(params object[] mesparametres)
{
    Console.Out.WriteLine(mesparametres[0]);
    Console.Out.WriteLine(mesparametres[1]);
    Console.Out.WriteLine(mesparametres[2]);
}

static void Main(string[] args)
{
    MethodeDeTest("1 textes", 34, new System.IO.FileInfo("c:\\test.txt"));
}
```

120

EXERCICES

Grace aux boucles, il est désormais possible de créer des algorithmes plus complexes et d'améliorer notre mini gestion de stock.

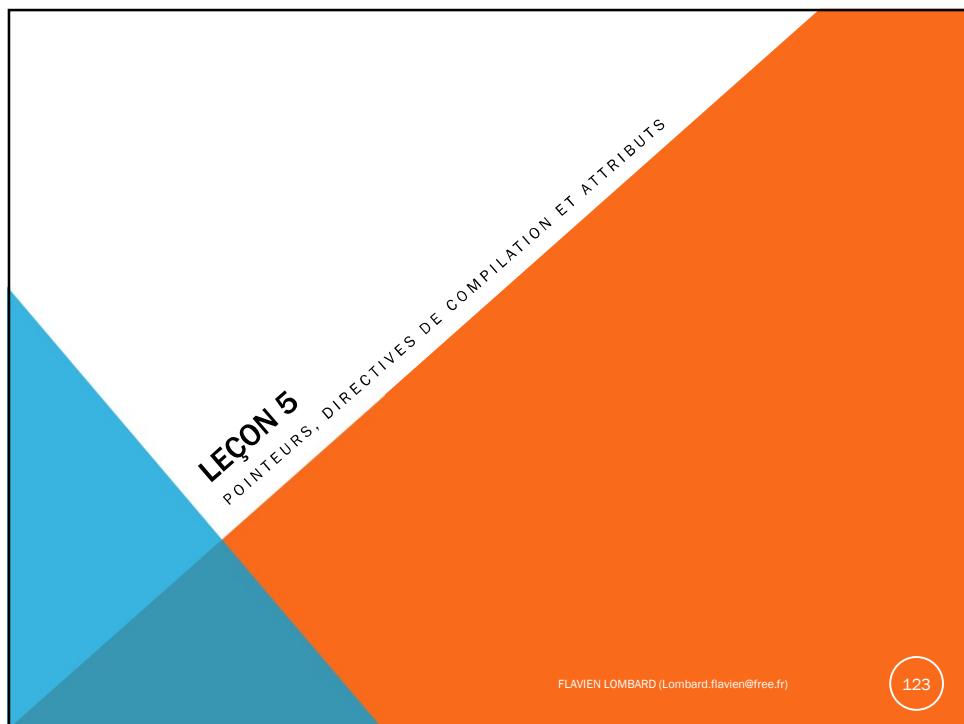
A partir de l'exercice précédent, rajouter la possibilité de demander à l'utilisateur s'il souhaite continuer avec un autre article ou pas.

Exercice 2 : concevoir un jeu demandant à l'utilisateur de saisir une valeur entre 0 et 100, l'ordinateur compare la valeur à un chiffre tiré au hasard et informe l'utilisateur si le nombre est plus grand ou plus petit. La partie continue tant que l'utilisateur n'a pas trouvé le chiffre, ou si plus de 10 essais ont été réalisés



121

122



The image shows a template for taking notes. It features a large central area for writing, divided into horizontal lines. This central area is surrounded by a thin black border. The entire template is set against a background with a blue base and orange side panels. In the bottom right corner of the template area, there is a small circular icon containing the number "124". Above the note-taking area, the word "NOTES" is printed in a bold black font.

LES POINTEURS

Uniquement en c#

Mot clé unsafe

Option /unsafe à spécifier lors de la compilation

Permet l'accès direct en mémoire

GC inactif, gestion de la mémoire .NET non vérifié...

```
class Program
{
    unsafe static void IncPtr(int* pInt)
    {
        (*pInt)++;
    }
    unsafe static void Main()
    {
        int i = 10;
        IncPtr(&i);
        Console.WriteLine("Nv val de i : " + i);
    }
}
```

```
static unsafe void Main(string[] args)
{
    Console.WriteLine("Taille du tableau : ");
    uint taille = uint.Parse(Console.ReadLine());
    long* tab = stackalloc long[(int)taille];
    for (int i = 0; i < taille; i++)
        tab[i] = i * i;
    for (int i = 0; i < taille; i++)
        Console.WriteLine(tab[i]);
}
```

125

DIRECTIVES DE COMPILEMENT

Permet de spécifier un comportement particulier du compilateur

```
#define TEST

using System;
using System.IO;
using ConsoleApplication1;

class Program
{
    static void Main(string[] args)
    {
#define DEBUG
        Console.WriteLine("Message de debug");
#define TEST
        Console.WriteLine( "Message de test" );
#define else
        Console.WriteLine( "Message normal" );
#define endif
    }
}
```

126

LES ATTRIBUTS

Les attributs sont fortement liés à la réflexion
Les attributs enrichissent la structure du code par des métadonnées.

```
[DllImport("Kernel32.dll")]
public static extern void Beep(int f, int duree);

[Serializable]
public string MonText;
```

127

VERSIONS DES COMPOSANTS

Gestion des versions.NET très poussé.
Chargement de dll dynamique
Possibilité d'utiliser 2 versions d'une même classe grâce aux alias

```
extern alias Bib1;
extern alias Bib2;

using System;
using System.IO;
using ConsoleApplication1;
using System.Runtime.InteropServices;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("{0} {1}", Bib1.Bib.Classe.Version,
                         Bib2.Bib.Classe.Version);
    }
}
```

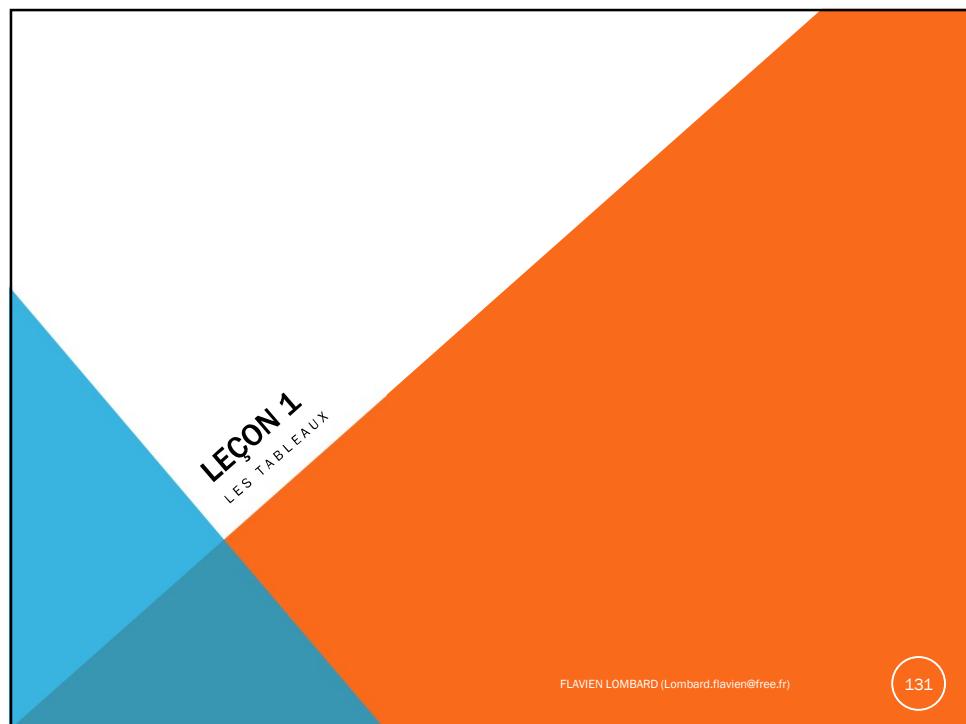
128



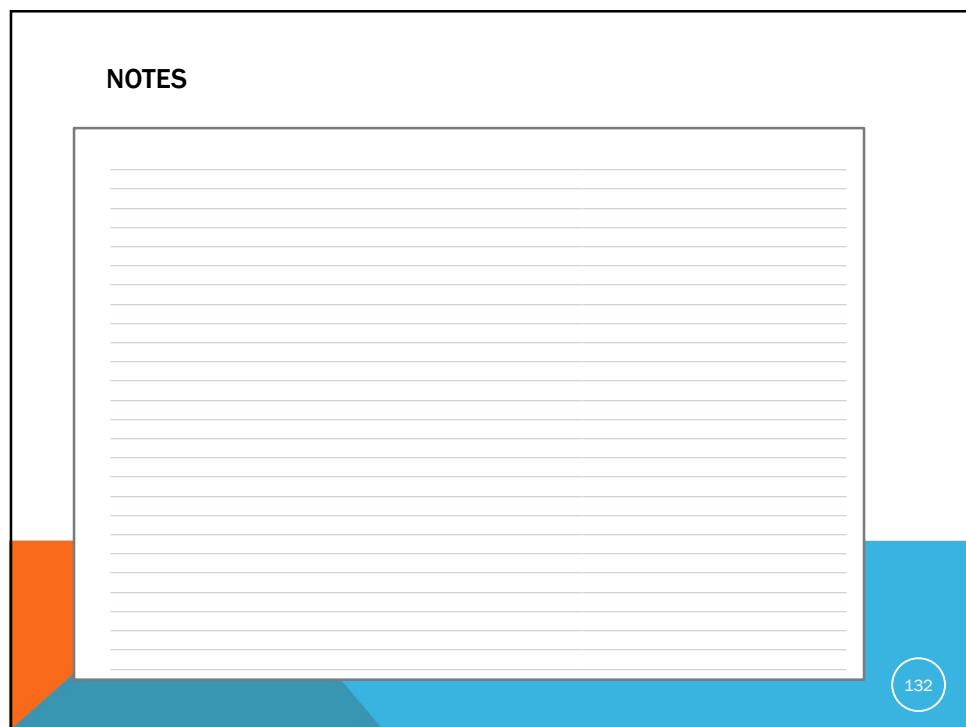
NOTES

Handwriting practice lines for notes.

130



FLAVIEN LOMBARD (Lombard.flavien@free.fr)



LES TABLEAUX SIMPLES

Les tableaux sont des objets comme les autres
Utilisation de la classe System.Array

```
int[] tabInt;  
System.Array t = new int[13];
```

Initialisation direct possible

```
int[] t = {1,2,3,4,5};
```

Affectation

```
int i = t[3];  
t[3] = 45;
```

133

TABLEAUX MULTIDIMENSIONNELS

Un tableau simple est une liste
Un tableau à 2 dimensions est une grille

```
int[,] tabInt;  
int[,] t = new int[13,4];  
  
int i = t[3,2];  
t[3,8] = 45;
```

134

TABLEAUX IMBRIQUÉS (JAGGED ARRAYS)

Tableaux contenant des tableaux

```
int [][] t = new int[2][]; // Deux dimensions  
t[0] = new int [2]; // 1ere ligne de 2 cases  
t[1] = new int [4]; // 2nde ligne de 4 cases  
t[1][3] = 30;
```

135

EXERCICES

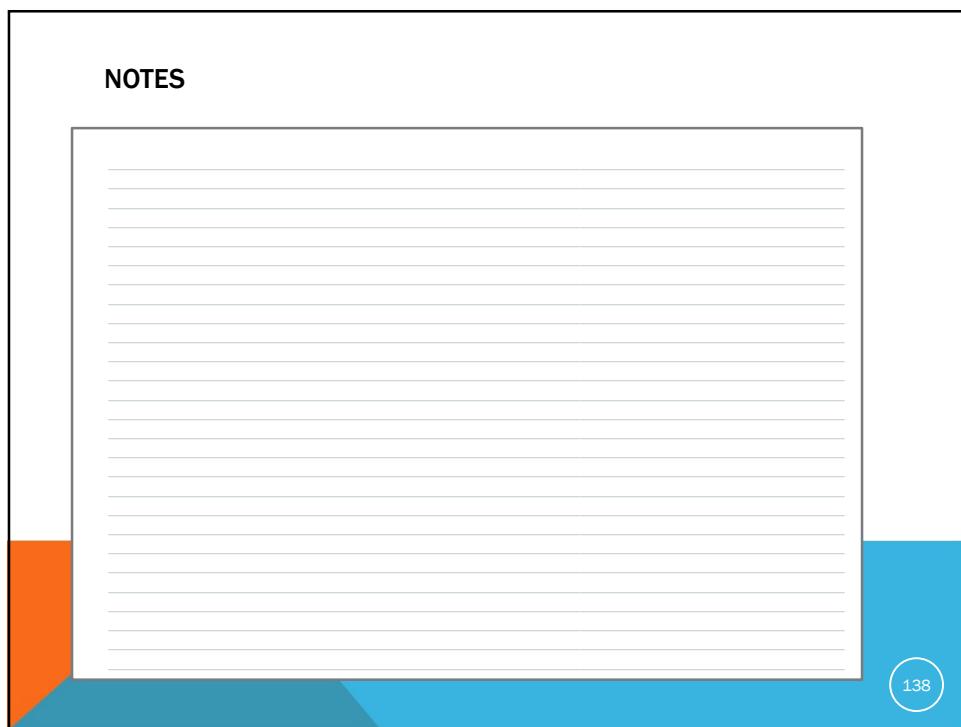
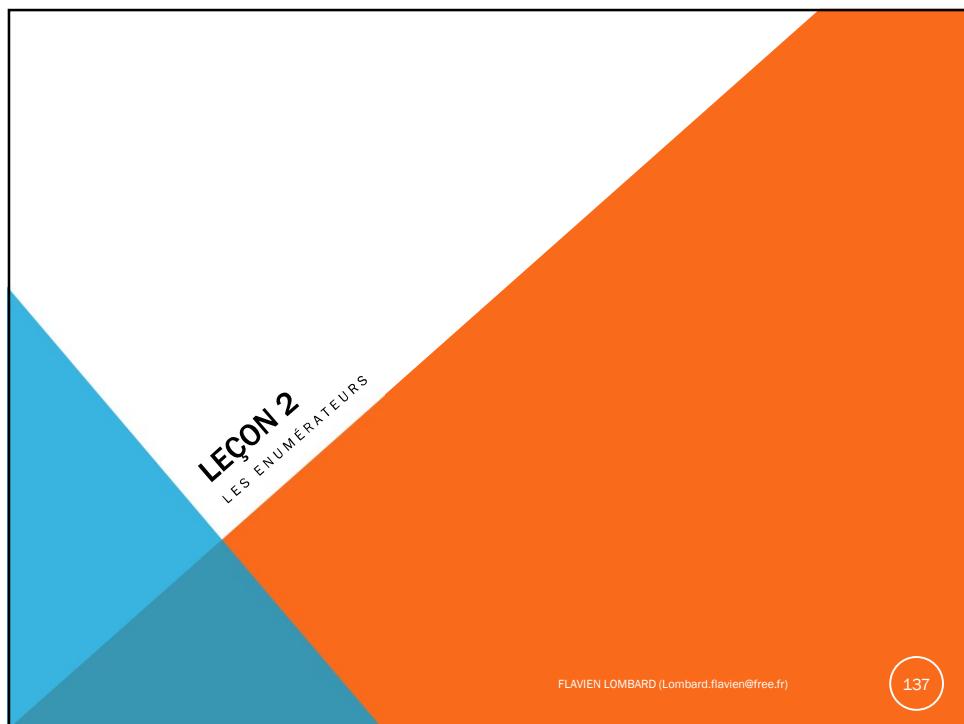
A partir de l'exercice précédent avec les articles, rajouter la possibilité de stocker des articles dans un tableau, tant que l'utilisateur le souhaite. Un tableau étant fixe, il est nécessaire de définir un tableau suffisamment grand (1000) est de gérer un index pour savoir où insérer l'article dans le tableau.

Lorsque l'utilisateur ne souhaite plus ajouter d'articles dans le tableau, on sort de la boucle et on parcourt les articles du tableau pour afficher le libelle et le prix total de chaque article.

Exercice 2 : Le jeu du pendu
on définit un tableau de mots, on tire au hasard un mot.
L'utilisateur doit proposer une lettre, si la lettre appartient au mot, on ajoute la lettre à un tableau de caractères permettant de savoir si le mot a été trouvé ou pas. Si la lettre n'appartient pas au mot, un compteur s'incrémentera jusqu'à un maximum de 10 (dessin du pendu)



136



ENUMÉRATEURS

La base de toute collections.
Utilisation de foreach pour parcourir un Enumerator

Implémentation de `GetEnumerator()`
`System.Collections.IEnumerator`

```
// Lit l'élément courant
object Current {get; }

// Première méthode invoquée par foreach,
// Rend vrai si l'élément courant est valide
bool MoveNext();

// Le premier élément de la collection devient le prochain élément
void Reset();
```

139

TABLEAUX DYNAMIQUES

ArrayList

- Un tableau n'est pas assez souple car sa taille est fixe.
- La classe `ArrayList` permet de stocker des objets sans spécifier de taille
- Taille dynamique
- Hérite de `IEnumerable` (peut donc être utilisé avec foreach)

Inconvénients :

- On peut stocker tous types de données
- Problème de Boxing/Unboxing

```
ArrayList array = new ArrayList();
array.Add(5);
array.Add("Toto");
array.Add(new Personne());

int a ... = (int)array[0];
```

140

TABLEAUX DYNAMIQUES

Le tri

Fonction « Sort »

fonctionne avec des classes héritant de **IComparable**

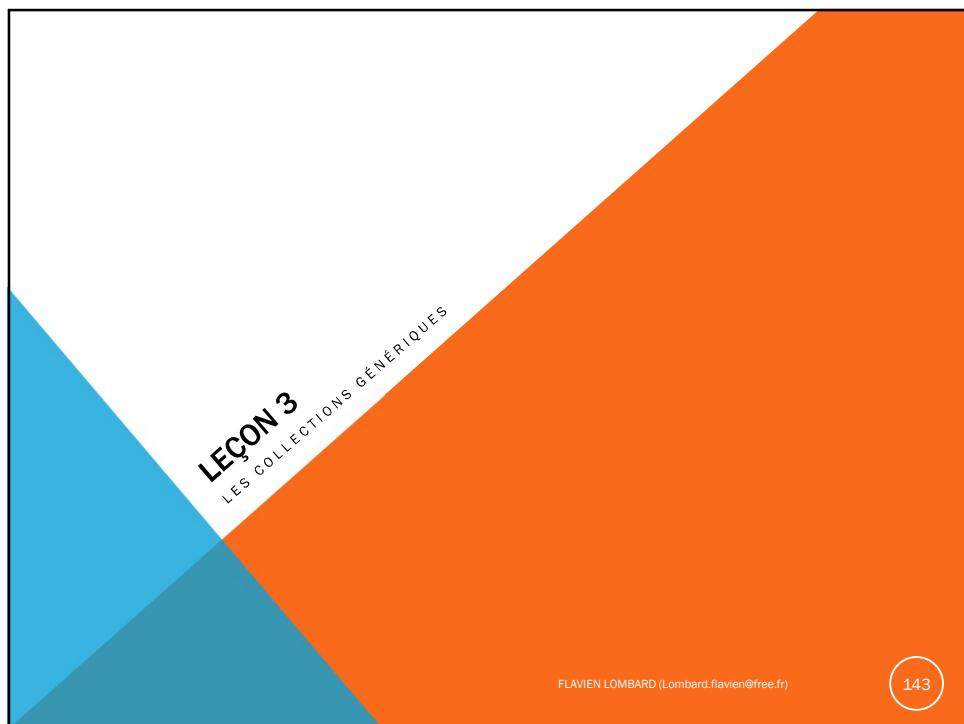
Ou avec un paramètre de type **IComparer**

```
ArrayList array = new ArrayList();
array.Add(new Personne(){Nom = "Dupond"});
array.Add(new Personne(){Nom = "Durand"});
array.Add(new Personne(){Nom = "Martin" });

array.Sort(new PersonneComparer());
```

```
class PersonneComparer : IComparer
{
    public int Compare(object x, object y)
    {
        Personne p1 = (Personne)x;
        Personne p2 = (Personne)y;
        return p1.Nom.CompareTo(p2.Nom);
    }
}
```

141



C'EST QUOI UN GÉNÉRIQUE ?

Une classe générique est une classe permettant des traitements sur des types inconnus à la conception.

Ce sont des Meta-class (classes qui génèrent des classes)

Lors de la compilation, le compilateur transforme ces classes en une classe avec le bon type avant de compiler.

Avantage du langage fortement Typé.

évite le boxing/unboxing

Utilisation de <T>

```
List<Personne> array = new List<Personne>();  
array.Add(new Personne(){Nom = "Dupond"});  
array.Add(new Personne(){Nom = "Durand"});  
array.Add(new Personne() { Nom = "Martin" });  
  
array.Add("Martin");
```

Erreur, car le type
n'est pas celui définit
(ici Personne)

145

LISTE GÉNÉRIQUE

List<T>

Classe générique équivalent à [ArrayList](#)

```
List<Personne> lst = new List<Personne>();  
  
lst.Add(new Personne("Jean"));  
lst.Add(new Personne("Jacques"));  
lst.Add(new Personne("Marcel"));  
lst.Add(new Personne("Jeanne"));  
lst.Add(new Personne("Fred"));  
  
foreach (Personne p in lst)  
    Console.Out.WriteLine(p);
```

146

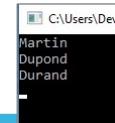
LISTES TRIÉES

SortedList

Liste trié avec table de hachage.
Pas de doublons dans la clé possible
Performance accrue pour le tri

Utilisable avec une clé, et pas un indice

```
SortedList<int, Personne> ListTrie = new SortedList<int, Personne>();  
  
ListTrie.Add(4,new Personne(){Nom = "Dupond"});  
ListTrie.Add(5,new Personne(){Nom = "Durand"});  
ListTrie.Add(3,new Personne() { Nom = "Martin" });  
  
foreach (var p in ListTrie)  
    Console.WriteLine(p.Value.Nom);
```



147

DICTIONNAIRE

Dictionary< TKey, TValue >

Permet un stockage de données par clé. La récupération d'une valeur a partir de la clé est quasi instantané, quelque soit le nombre d'éléments dans le dictionnaire.

Fournit un conteneur clé, valeur
La clé est unique, les doublons ne sont pas autorisés

Utilisation de `KeyValuePair` dans le foreach

```
Dictionary<int, Personne> Dico = new Dictionary<int, Personne>();  
  
Dico.Add(4,new Personne(){Nom = "Dupond"});  
Dico.Add(5,new Personne(){Nom = "Durand"});  
Dico.Add(3,new Personne() { Nom = "Martin" });  
  
foreach (var p in Dico)  
    Console.WriteLine(p.Value.Nom);
```



148

LES PILES ET LES FILES

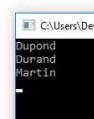
Stack<T>

Queue<T>

Définisse les piles FIFO (First In First Out) et FILO (First In Last Out)

```
Queue< Personne> Dico = new Queue< Personne>();
Dico.Enqueue( new Personne(){Nom = "Dupond"});
Dico.Enqueue( new Personne(){Nom = "Durand"});
Dico.Enqueue( new Personne() { Nom = "Martin" });

while (true)
{
    Personne p;
    if (!Dico.TryDequeue(out p)) break;
    Console.WriteLine(p.Nom);
}
```



```
Stack< Personne> Dico = new Stack< Personne>();
Dico.Push( new Personne(){Nom = "Dupond"});
Dico.Push( new Personne(){Nom = "Durand"});
Dico.Push( new Personne() { Nom = "Martin" });

while (true)
{
    Personne p;
    if (!Dico.TryPop(out p)) break;
    Console.WriteLine(p.Nom);
}
```



149

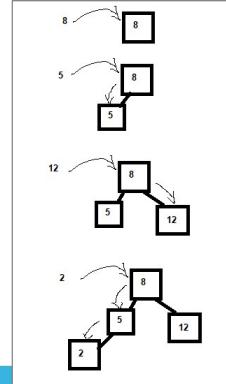
EXERCICES

A partir de l'exercice précédent avec les articles, transformer le tableau en List<T>

Exercice 2 : Création d'un algorithme d'arbre binaire
Un arbre binaire et une structure permettant de stocker une valeur et 2 sous structures de même type.

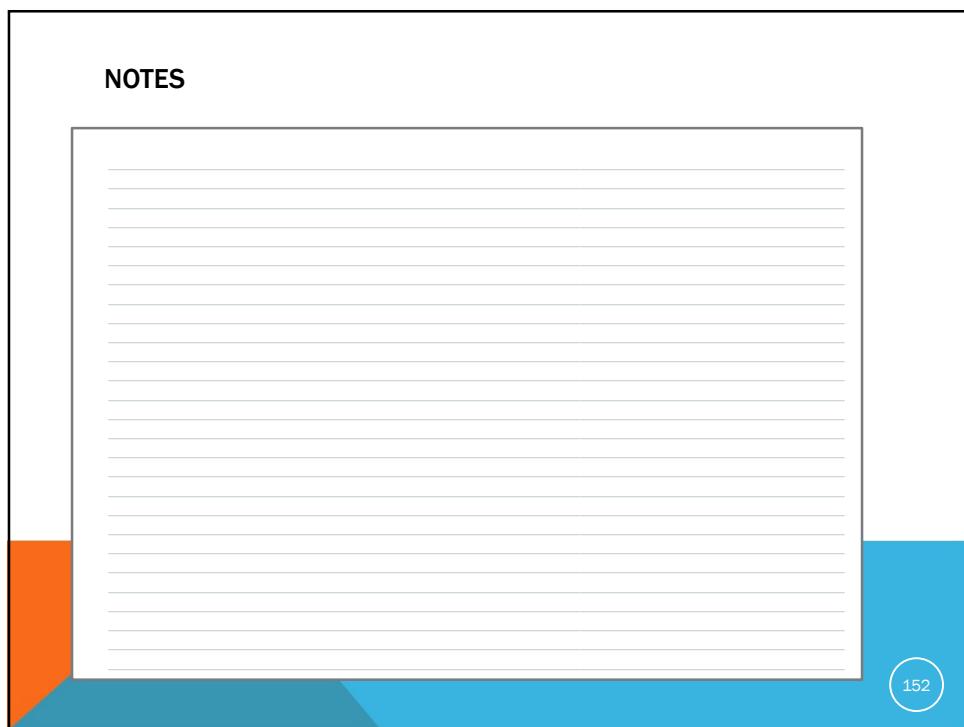
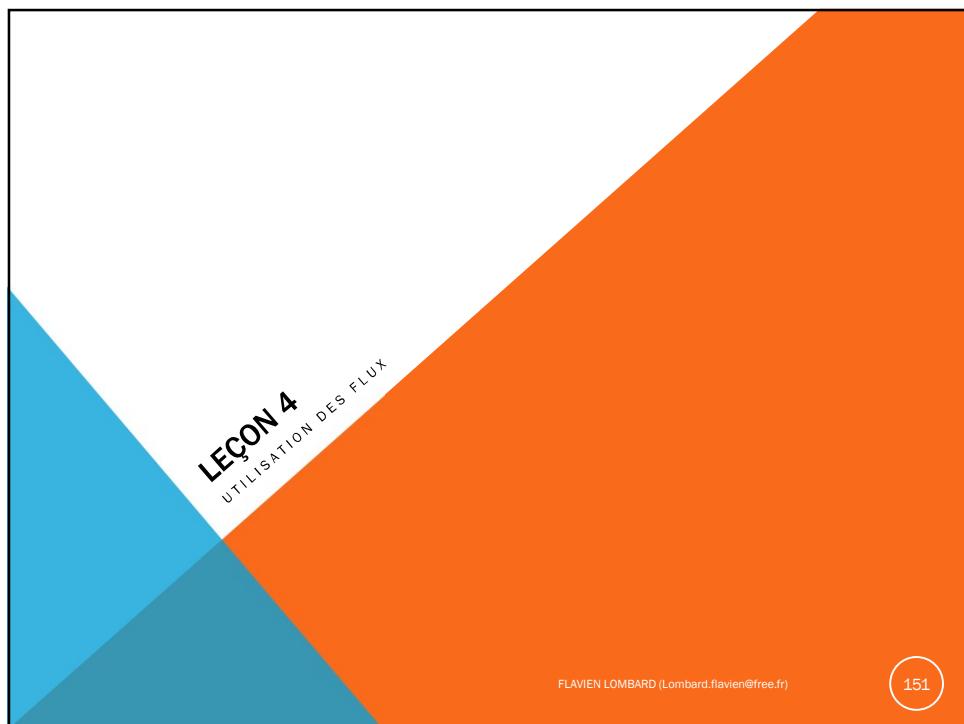
Lorsqu'on rajoute une valeur au travers une fonction, celle-ci compare la valeur de l'objet avec la valeur en paramètre, si c'est plus grand la valeur se répercute à droite, si c'est plus petit, elle se répercute à gauche.

Cela va constituer un arbre dont les éléments seront pré-triés. Par récursivité, il est possible de récupérer les valeurs dans l'ordre.



150





APPROCHE CLASSIQUE PAR LES LISTES

```

static void Main(string[] args)
{
    List<string> liste = new List<string>() { "bateau", "maison", "velo", "voiture" };

    liste = Majuscule(liste);
    liste = First2(liste);

    foreach (var s in liste)
        Console.WriteLine(s);
}

private static List<string> Majuscule(List<string> liste)
{
    List<string> lst = new List<string>();
    foreach (var s in liste)
    {
        Console.WriteLine("Majuscule");
        lst.Add(s.ToUpper());
    }
    return lst;
}

private static List<string> First2(List<string> liste)
{
    List<string> lst = new List<string>();
    foreach (var s in liste)
    {
        Console.WriteLine("First2");
        lst.Add(s.Substring(0,2));
    }
    return lst;
}

```

Dans cette approche, 3 listes sont créées et 3 boucles sont faites, les 1 derrière les autres

153

APPROCHE PAR FLUX

```

static void Main(string[] args)
{
    IEnumerable<string> liste = new List<string>() { "bateau", "maison", "velo", "voiture" };

    liste = Majuscule(liste);
    liste = First2(liste);

    foreach (var s in liste)
        Console.WriteLine(s);
}

private static IEnumerable<string> Majuscule(IEnumerable<string> liste)
{
    foreach (var s in liste)
    {
        Console.WriteLine("Majuscule");
        yield return s.ToUpper();
    }
}

private static IEnumerable<string> First2(IEnumerable<string> liste)
{
    foreach (var s in liste)
    {
        Console.WriteLine("First2");
        yield return s.Substring(0,2);
    }
}

```

Dans cette approche, 1 seul liste créée et 1 seul boucle est faite avec les 3 traitements

Le compilateur va imbriquer l'ensemble des foreach pour créer un flux de données

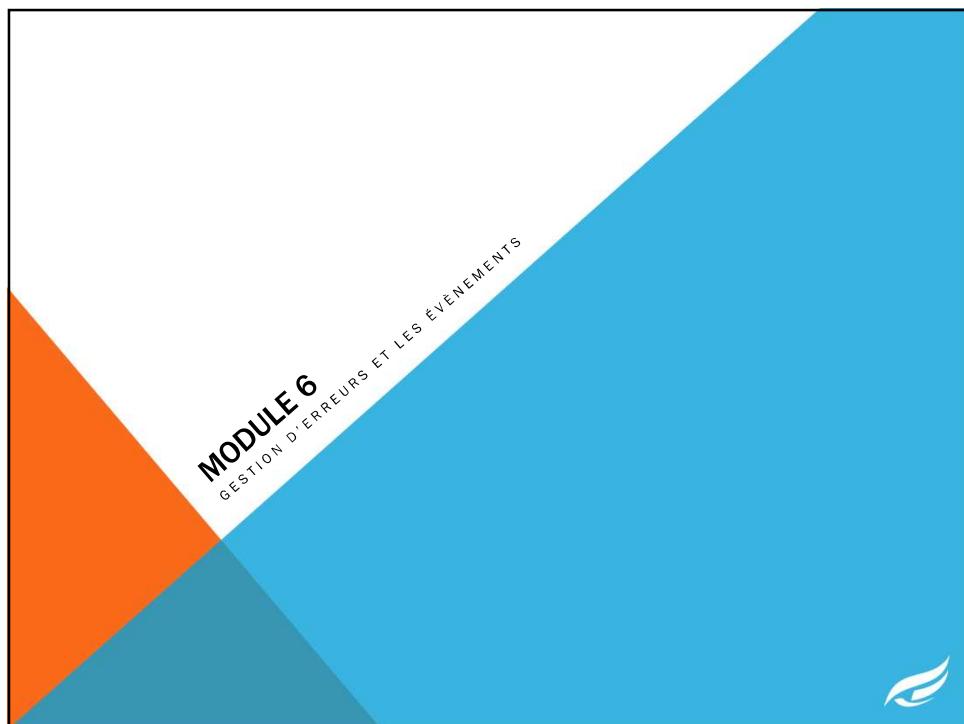
154

DÉMONSTRATION



155

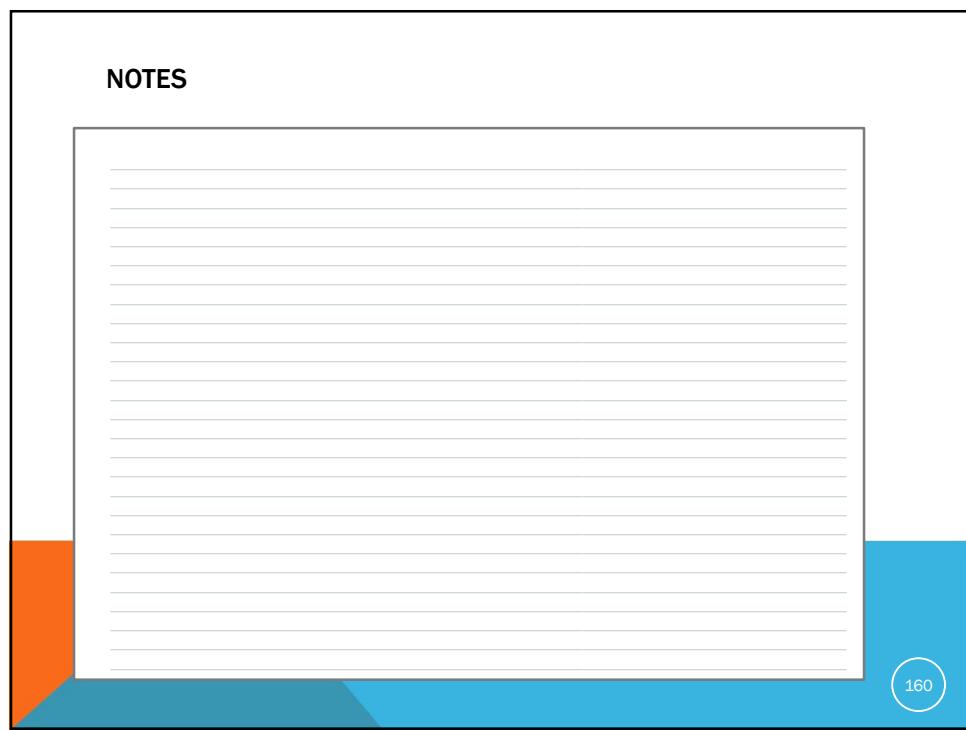
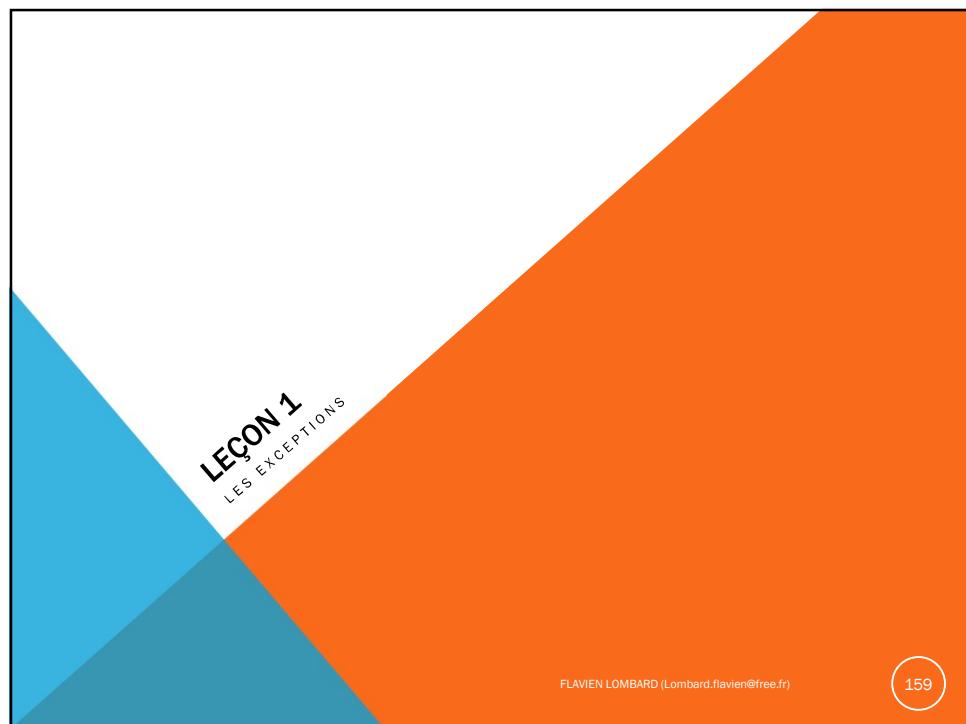
156



NOTES

A blank sheet of lined paper with horizontal ruling lines, intended for notes. It is positioned in the center of a white rectangular area with a thin black border, which is itself centered within a larger blue and orange frame.

158



TRAITEMENT D'ERREURS LES EXCEPTIONS

Evite les multiples tests.

Système d'exceptions hiérarchisé

Utilisation des exceptions au plus bas niveau de .NET

Utilisation de try...catch...finally

Possibilité de créer ses propres classes d'exception en dérivant System.exception

161

LES EXCEPTIONS

En C, l'approche pour la gestion des erreurs consiste à renvoyer un code en retour d'une fonction. Cette approche limite les valeurs possibles en sorti et elle est source d'erreur du fait de la mauvaise interprétation du code d'erreur.

une approche plus moderne, consiste à répercuter l'erreur dans la pile des appels et permettre aux appelants de traiter cette erreur.

C'est ce qu'on appelle une exception

```
static void Main(string[] args)
{
    int a = 4;
    int b = 0;

    Traitements(a, b);
}

public static int Traitements(int a, int b)
{
    //Calcul incluant une division
    int c = a / b; ✘
    return c;
}
```

Exception non gérée
System.DivideByZeroException : 'Attempted to divide by zero.'

Afficher les détails | Copier les détails | Démarrer la session Live Share
Paramètres d'exception

162

LES EXCEPTIONS

Une exception peut être attrapée (catch) par un bloc de code try[]:

Si une erreur est rencontré, l'exécution s'arrête et le code du bloc catch est immédiatement exécuté.

```
public static int Traitement(int a, int b)
{
    try
    {
        //Calcul incluant une division
        int c = a / b;
        return c;
    }catch (System.Exception)
    {
        return 0;
    }
}
```

163

LES EXCEPTIONS

Il est possible de relevé un autre type d'exception pour apporter des précisions sur l'erreur

```
public static int Traitement(int a, int b)
{
    try
    {
        //Calcul incluant une division
        int c = a / b;
        return c;
    }catch (System.Exception)
    {
        throw new ArgumentException("le parametre b ne peut pas valoir 0");
    }
}
```

164

LES EXCEPTIONS

Il est possible de préciser quelles exceptions peuvent être attrapées

```
public static int Traitement(int a, int b)
{
    try
    {
        //Calcul incluant une division
        int c = a / b;
        return c;
    }
    catch (System.DivideByZeroException)
    {
        throw new ArgumentException("le paramètre b ne peut pas valoir 0");
    }
    catch (System.Exception)
    {
        throw new ApplicationException("Erreur de calcul");
    }
}
```

165

LES EXCEPTIONS

Le bloc finally permet de libérer les ressources même si une erreur à eu lieu

```
public static int Traitement(int a, int b)
{
    try
    {
        //ouverture d'un fichier
        int c = a / b;
        //traitement sur le fichier
        return c;
    }
    catch (System.Exception)
    {
        throw new ApplicationException("Erreur de calcul");
    }
    finally
    {
        //Fermeture du fichier
    }
}
```

Ce code est toujours exécuté

166

LES EXCEPTIONS

Les exceptions peuvent être chainées pour faciliter le débogage

```

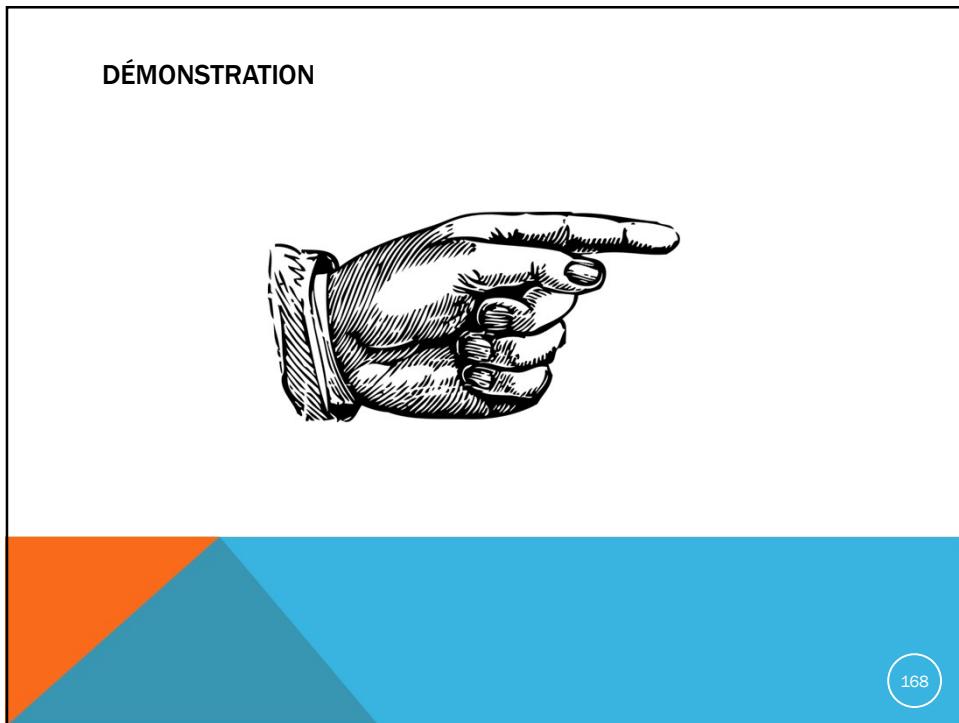
try
{
    Traitement(a, b);
}
catch (System.Exception ex)
{
    Console.WriteLine("ERREUR");
}

public static int Traitement(int a, int b)
{
    try
    {
        return ssTraitement(a, b);
    }
    catch (System.Exception ex)
    {
        throw new ApplicationException("erreur dans le traitement", ex);
    }
}

public static int ssTraitement(int a, int b)
{
    try
    {
        return a / b;
    }
    catch (System.DivideByZeroException ex)
    {
        throw new ArgumentException("b ne peut pas valoir 0", ex);
    }
}

```

167



EXERCICES

A partir de l'exercice précédent sur les articles

Gérer les erreurs de saisie des utilisateurs concernant le prix de l'article. (L'utilisateur est en mesure de saisir du texte à la place du prix)

Gérer cela avec des exceptions



169

USING ET IDISPOSABLE

Les classes **IDisposable** sont des classes qui nécessite une libération explicite des ressources

Le mot clé **using** est accompagné d'un bloc de code

using fait appelle a Dispose()

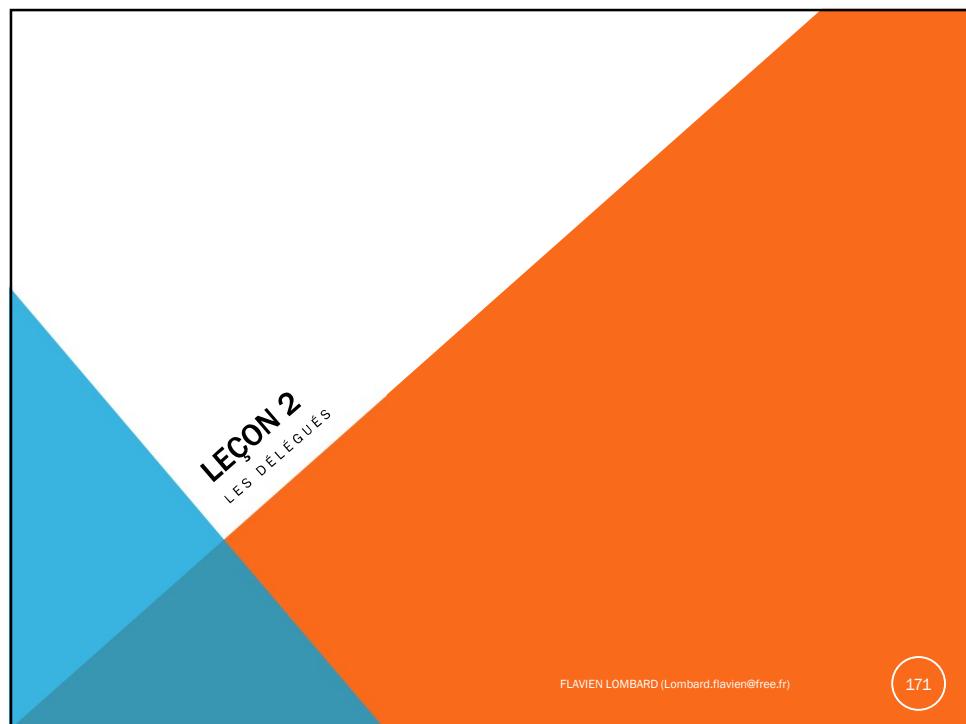
Dispose permet d'effectuer des opérations de nettoyage et de libération (socket, connexion DB)

```
using (StreamReader sr = new StreamReader("c:/boot.ini"))
{
    using (StreamWriter sw = new StreamWriter("c:/copie.txt"))
    {
        sw.WriteLine(sr.ReadToEnd());
    }
}
```

appel de sw.Dispose()
appel de sr.Dispose()

Le using inclut un mécanisme de
try...catch...finally, garantissant l'appel au dispose
dans tous les cas

170



DÉLÉGUÉS

Délégués = Pointeur de méthode

Un délégué est un type permettant de déclarer une variable qui pointe sur une méthode.

Mot clé `delegate`

Création comme une instance

Respect de la signature

```
delegate double Operation(int a, int b);

class Program
{
    static void Main(string[] args)
    {
        Operation op = Addition;
        double resultat = op(5, 6);
    }

    public static double Addition(int a, int b)
    {
        return a + b;
    }

    public static double Soustraction(int a, int b)
    {
        return a - b;
    }
}
```

73

DÉLÉGUÉS

Délégués = Gestionnaire de pointeurs

Possibilité de chainer les délégués.

Possibilité d'effectuer des opération sur les délégués.

Les délégués sont utilisés dans les événements.
Dans l'utilisation des méthodes de callback, avec LINQ...

```
delegate double Operation(int a, int b);

class Program
{
    static void Main(string[] args)
    {
        CalculEtAffiche(5, 6, Addition);
        CalculEtAffiche(5, 6, Soustraction);
    }

    public static void CalculEtAffiche(int a, int b, Operation fct)
    {
        double resultat = fct(a, b);
        Console.WriteLine("Le résultat est " + resultat);
    }

    public static double Addition(int a, int b)
    {
        return a + b;
    }

    public static double Soustraction(int a, int b)
    {
        return a - b;
    }
}
```

DÉLÉGUÉS

Les expressions lambda

```
class Program
{
    static void Main(string[] args)
    {
        CalculEtAffiche(5, 6, (a, b) => a + b);
        CalculEtAffiche(5, 6, (a, b) => a + b);
    }

    public static void CalculEtAffiche(int a, int b, Func<int, int, double> fct)
    {
        double resultat = fct(a, b);
        Console.WriteLine("Le résultat est " + resultat);
    }
}
```

Une expression lambda est une représentation d'une fonction par le sigle =>

Tout ce qui se trouve devant => correspond aux paramètres de la méthode et tout ce qui se trouve derrière représente le corps de la méthode

175

DÉMONSTRATION



176

EXERCICES

A partir de l'exercice précédent sur les arbres binaires

Plutôt que de trier des entiers (ce n'est pas très utile), il serait préférable de trier des objets,

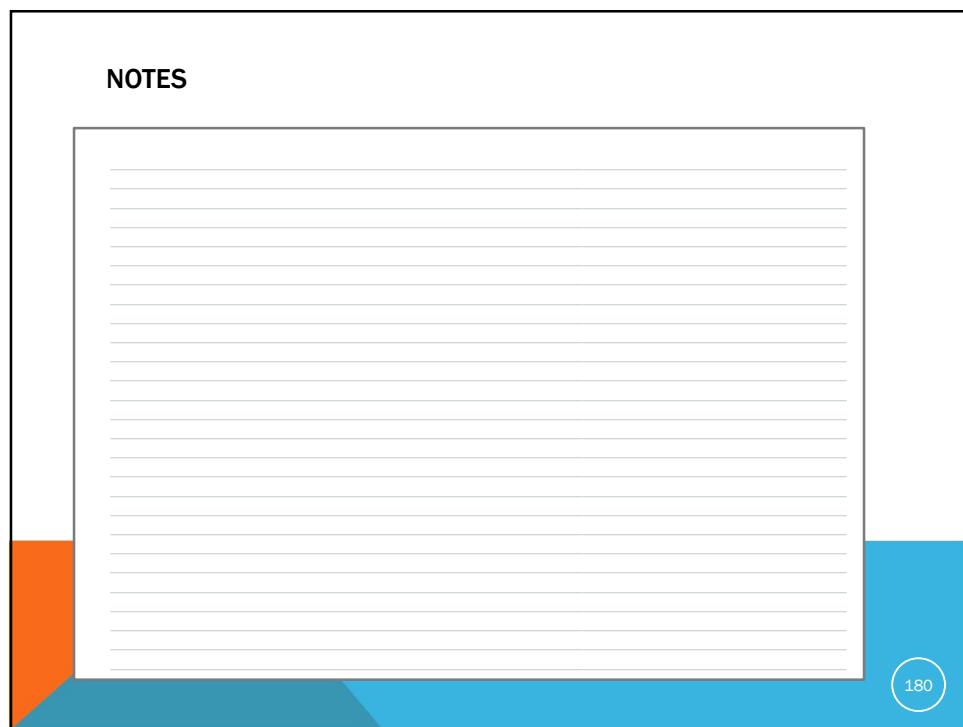
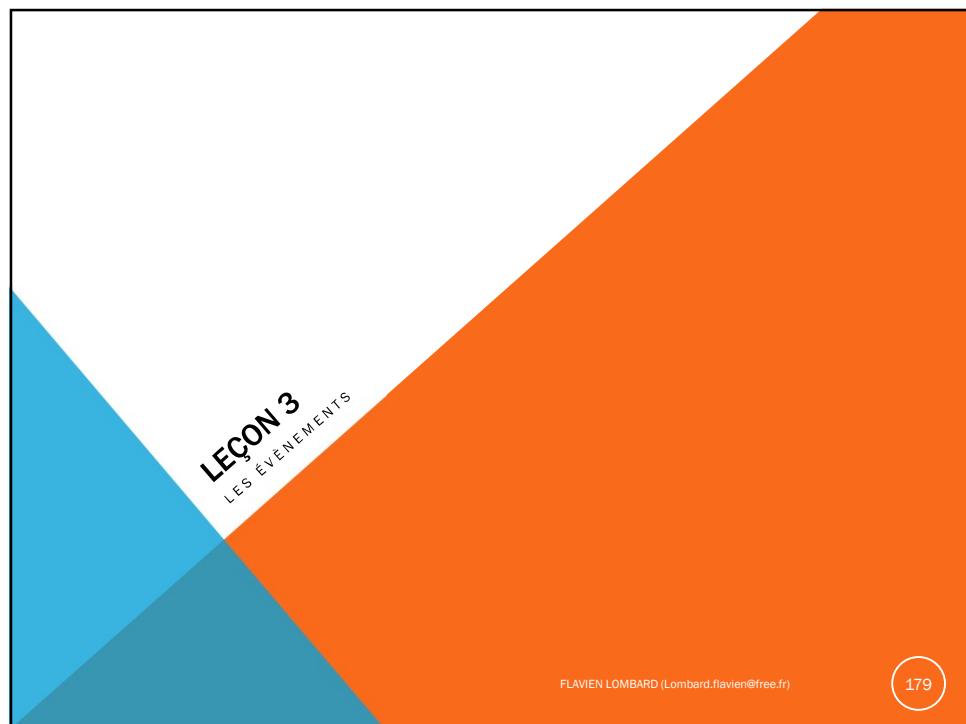
Il faut donc remplacer la valeur de type int, en valeur de type Object et utiliser une lambda pour gérer la comparaison permettant d'être à gauche ou à droite.



177



178



EVÈNEMENTS

Un évènement est un délégué avec des droits d'accessibilités particuliers

Mot clé **event (Event)**

Évènement = Délégué

Affectation impossible

Abonnement à un évènement grâce à **+= (AddHandler)**

Désabonnement grâce à **-= (RemoveHandler)**

Les évènements ont quasiment toujours la même signature :

```
public delegate void EventHandler(object? sender, EventArgs e);
```

Celui qui lève l'évènement

Paramètres de l'évènement

181

EVÈNEMENTS

Utilisation de **EventHandler** et **EventArgs**

Appelle d'un évènement

```
if (e != null) OnTextChanged(this, e);
```

```
class TicTac
{
    public event EventHandler OnTic;

    Timer tmr;

    public TicTac()
    {
        tmr = new Timer(o =>
        {
            if (OnTic != null)
                OnTic(this, new EventArgs());
        }, null, 0, 1000);
    }
}
```

REM : Les interfaces peuvent inclure des évènements

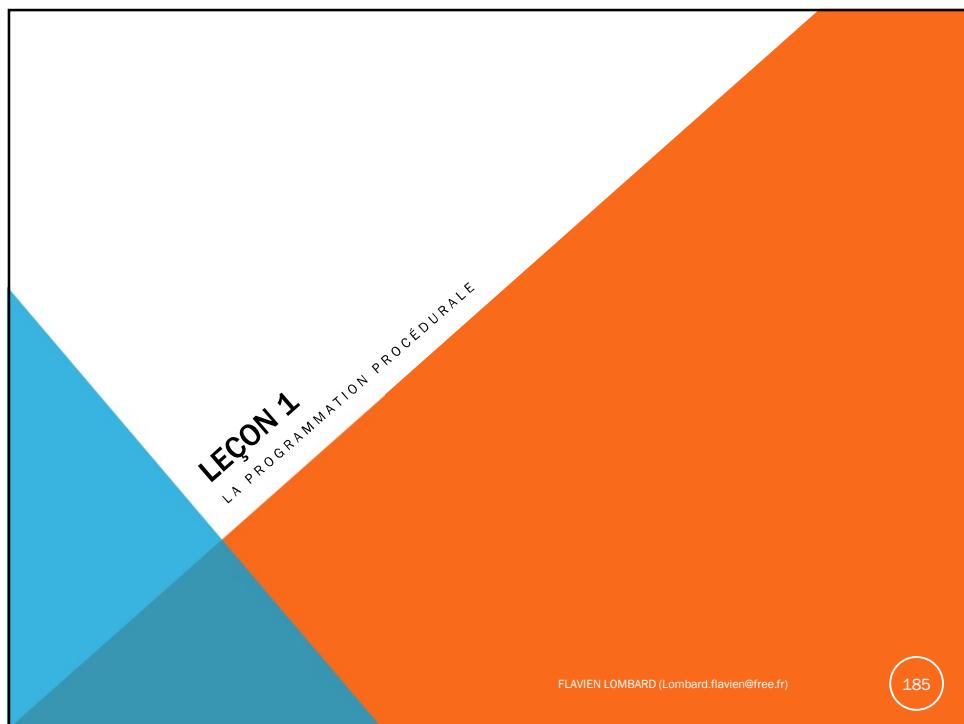
182



NOTES

Handwriting practice lines for notes.

184



The image shows a notes page. At the top left, it says "NOTES". Below that is a large rectangular area with horizontal lines for writing. The page has a blue footer bar at the bottom. In the bottom right corner, there is a small circular badge with the number "186".

L'APPROCHE PROCÉDURALE

Utilisation de fonctions et de procédures

- **Lisibilité**
 - plus compréhensible
 - effet « cloisonnement » (on ne développe que ce que doit faire la fonction que l'on développe)
- **Réutilisabilité**
 - Les fonctions peuvent être utilisés dans différents traitements, et/ou différents projets.
- **Modularité**
 - Les fonctions appelant d'autres fonctions peuvent être remaniés

187

LA PROGRAMMATION STRUCTURÉE

Inconvénients

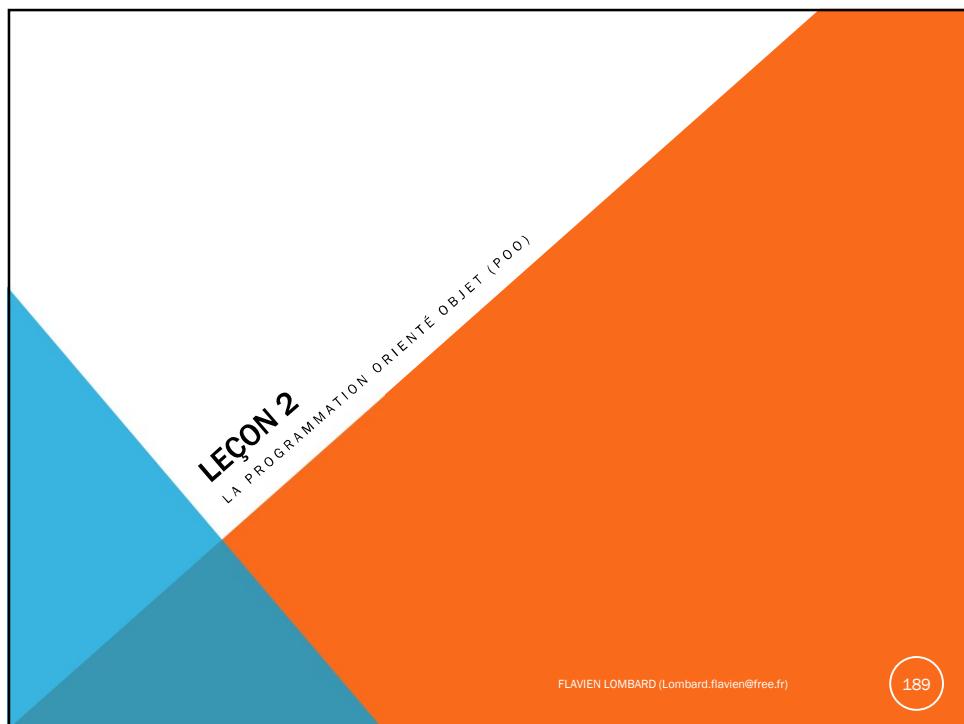
Il faut être très rigoureux et méthodique afin d'éviter le code spaghetti.

Impose une connaissance quasi-globale de l'application pour éviter de redévelopper une fonction existante.

Les fonctions ne sont pas suffisamment représentatives du monde que l'on cherche à reproduire.



188



The image shows a template for taking notes. It features a large central area with horizontal ruling lines for writing. This central area is surrounded by a white border. In the top-left corner of this border, the word "NOTES" is printed in bold capital letters. The entire template is set against a background with a blue base and orange side panels, matching the colors of the lesson cover above.

189

190

LA PROGRAMMATION OBJET

L'Objet

Regroupement de données et d'actions propre à un concept.

Quelques exemples :

Voiture, chat, maison, collection, tableaux, Socket, message SOAP...

En programmation objet, tout est objet!

Par exemple un entier (integer) est un objet.

191

LA PROGRAMMATION OBJET

L'effet boîte noire

Permet de gérer la façon dont sera utilisé les fonctions de l'objet

Lisibilité

Les objets représentent un concept, donc tous les éléments de l'objet sont en lien avec le concept

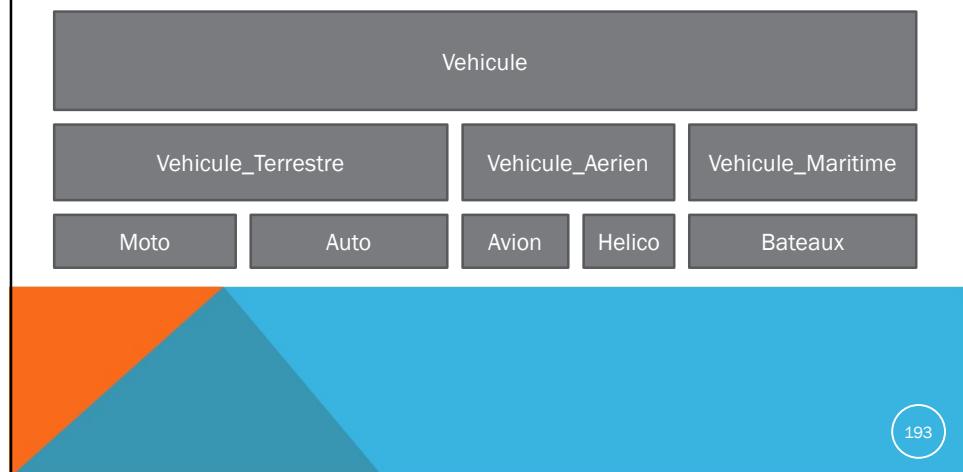
Réutilisation du code

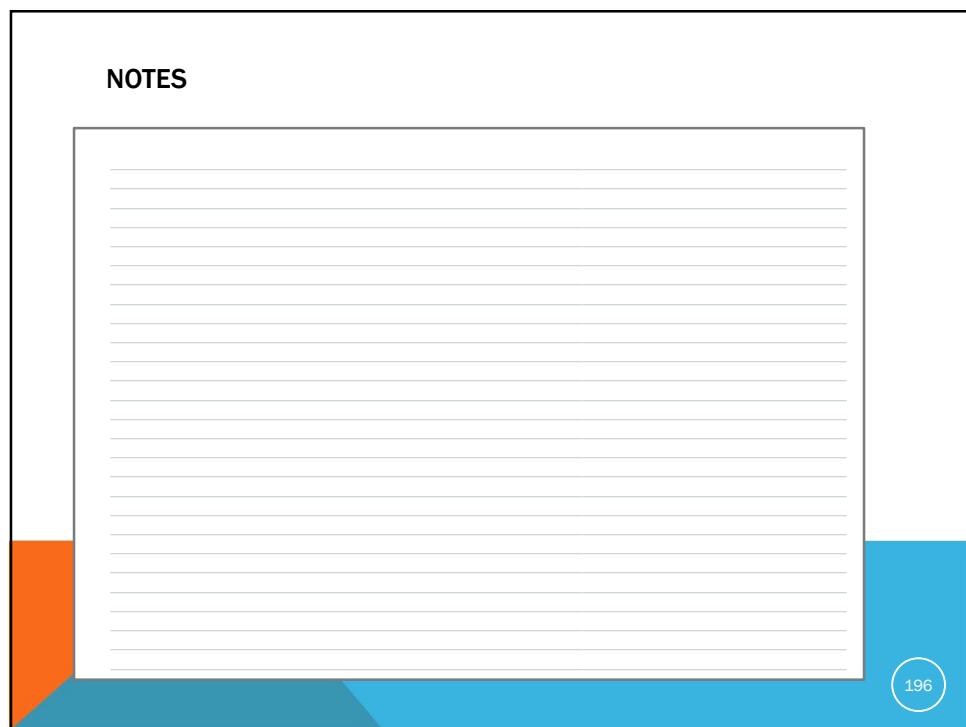
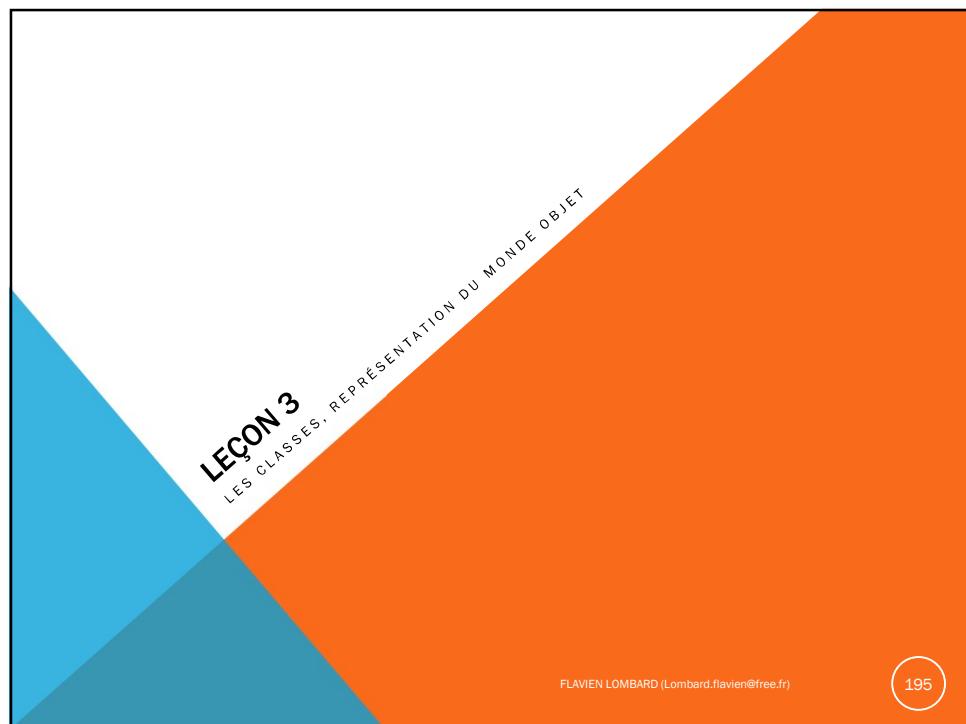
Les objets sont réutilisables dans plusieurs traitements et/ou projets

Représentation beaucoup plus claire du concept à développer

192

EXEMPLE D'ARCHITECTURE OBJET





LA CLASSE

Représentation abstraite de la structure d'un objet

Un objet est l'*instanciation* d'une classe.

Un objet est donc la concrétisation d'une classe.

La classe contient la définition des données (les attributs de classes) mais aussi des actions (fonctions/méthodes) utiles à l'objet.

Exemple

Commune
+code: var6
+code_commune: var4
+libelle_commune: var30
+code_departement: var5
+ajouter()
+modifier()
+supprimer()
+formulaire()

197

LE CONSTRUCTEUR

Fonction appelée obligatoirement au moment de l'instanciation d'un objet.

Syntaxe différente suivant les langages.

Permet d'initialiser correctement l'objet.

Allocation des ressources

Exemples :

Affectation de valeurs par défaut.

Lancement d'une action particulière à l'initialisation

...

198

LE DESTRUCTEUR

Fonction appelée obligatoirement au moment de la libération d'un objet.

Syntaxe différente suivant les langages.

Permet de libérer les ressources utilisées par l'objet.

Exemples :

Fermeture d'un fichier ouvert .

Synchronisation de Thread

...

199

LE DESTRUCTEUR

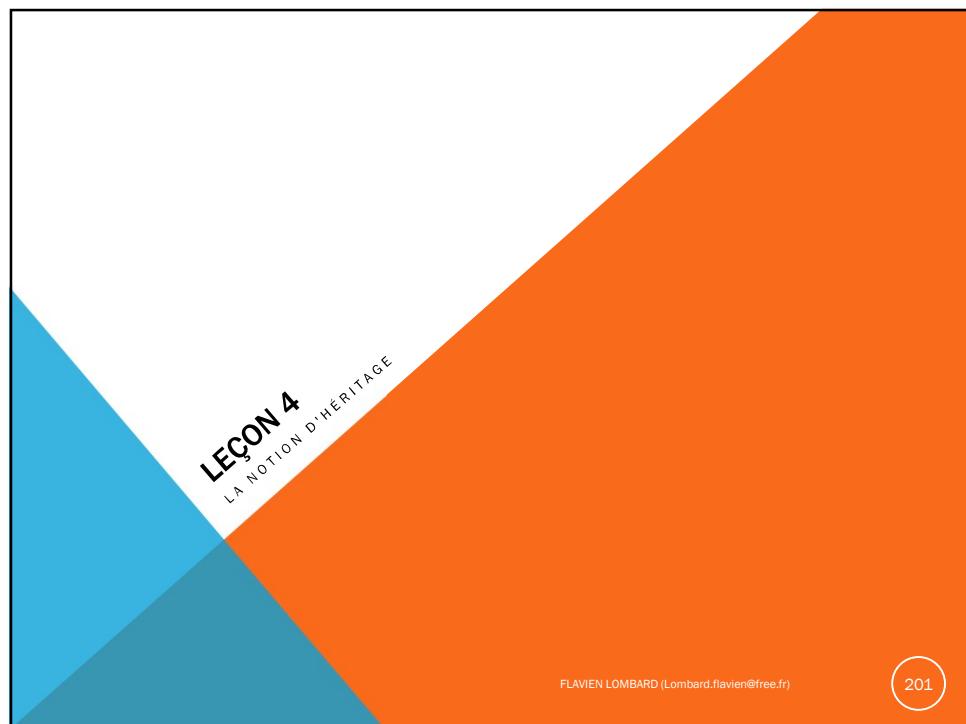
Durée de vie des objets

Les objets ont une durée de vie dépendant de leur emplacement dans le programme. Cela dépend de la porté de l'objet instancié.

En C++ par exemple, les objets doivent être libérés par le développeur.

En .NET, un objet est libéré de la mémoire grâce au Garbage Collector.

200



The image shows a notes page. At the top left, the word "NOTES" is printed in bold black capital letters. Below it is a large rectangular area with horizontal ruling lines for writing. The page has decorative borders at the bottom and sides: an orange border on the left, a blue border at the bottom, and a blue border on the right. In the bottom right corner, there is a small circular icon containing the number "202".

HÉRITAGE ET POLYMORPHISME

Héritage = spécialisation

L'héritage permet de définir des attributs et des méthodes spécifiques à un sous type d'objet.

Exemples:

L'attribut **NombreDeRoue** est spécifique aux classes **Vehicule_Terrestre**.

Par contre, **VitesseDeDeplacement** est commun à tous les véhicules.

Vehicule_Terrestre hérite de **Véhicule**, pour avoir les attributs **NombreDeRoue** et **VitesseDeDeplacement**

203

HÉRITAGE ET POLYMORPHISME

Dans cette exemple, **Vehicule_Terrestre** est la classe dérivé de **Véhicule**.

Vehicule est la classe de base (super classe) de **Vehicule_Terrestre**

204

HÉRITAGE ET POLYMORPHISME

Polymorphisme = changement de classe

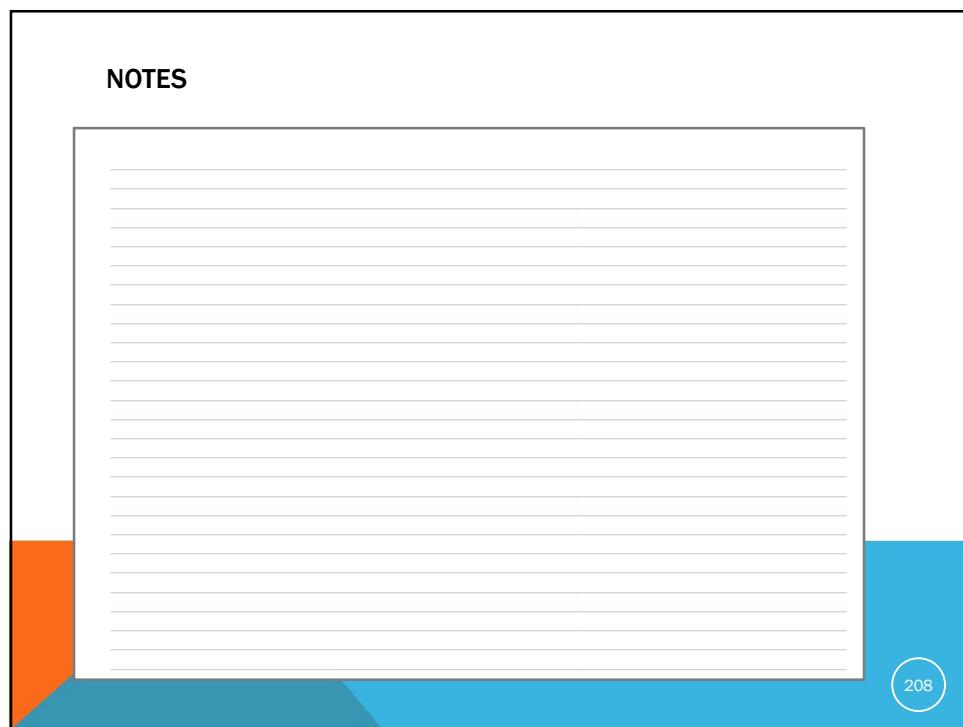
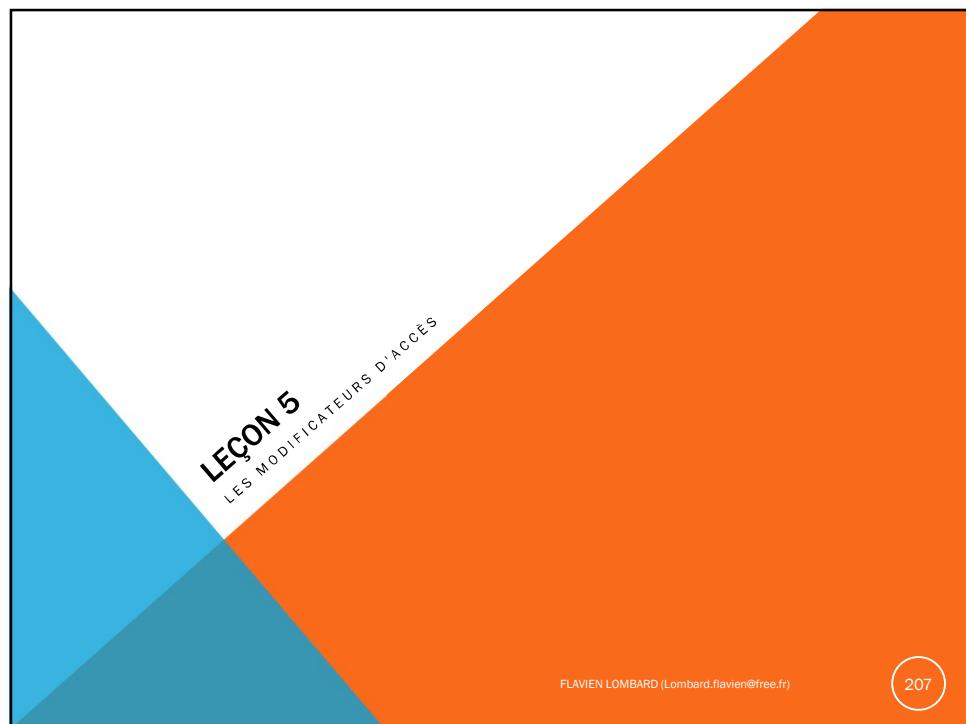
Un objet instancié pourra être utilisé comme étant de sa classe, ou de sa classe de base.

Exemple

Un tableau de Vehicule pourra contenir des Vehicule_Terrestre, ou des Vehicule_Maritime

205

206



LES MODIFICATEURS D'ACCÈS

Renforce le cloisonnement en sécurisant l'accès aux données.

Autorise ou masque l'accès à certains éléments de l'objet.

Les modificateurs d'accès est le mécanisme permettant l'encapsulation

209

LES MODIFICATEURS D'ACCÈS

public

L'élément est visible de tous les objets externe à la classe.

210

LES MODIFICATEURS D'ACCÈS

private

L'élément n'est accessible qu'à l'intérieur d'une même classe.

211

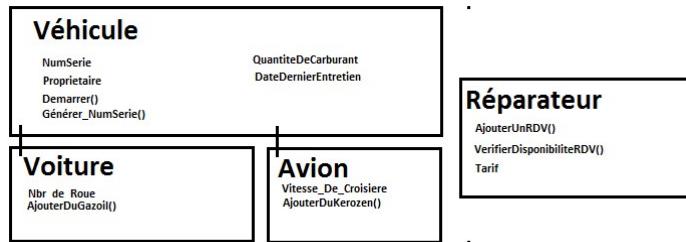
LES MODIFICATEURS D'ACCÈS

protected

L'élément est visible par les objets dérivés de la classe

212

EXERCICE



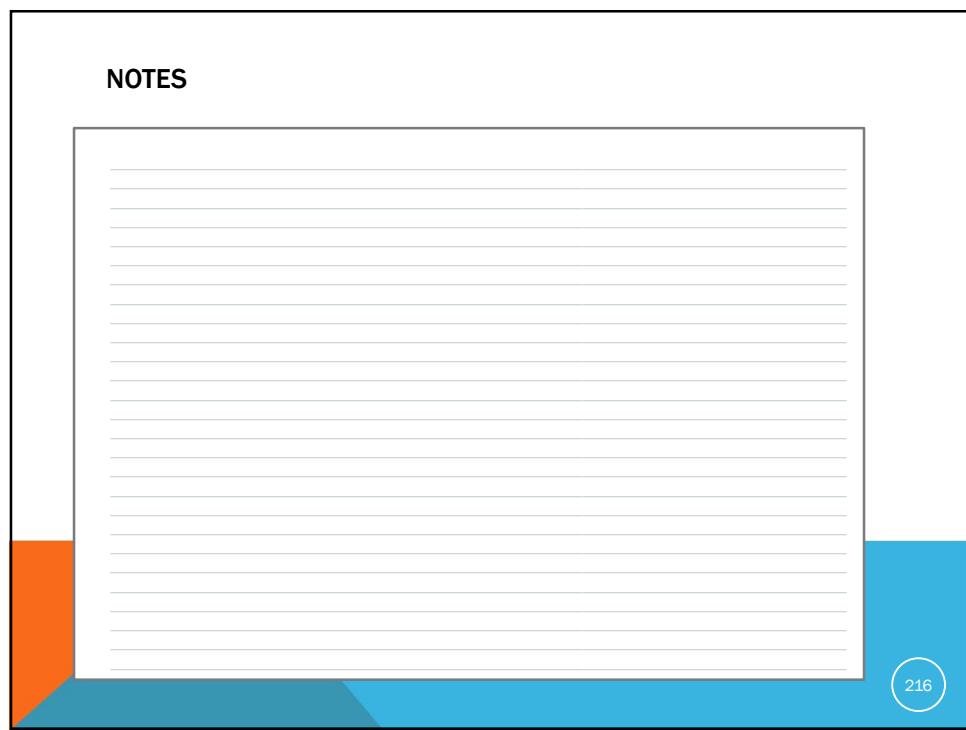
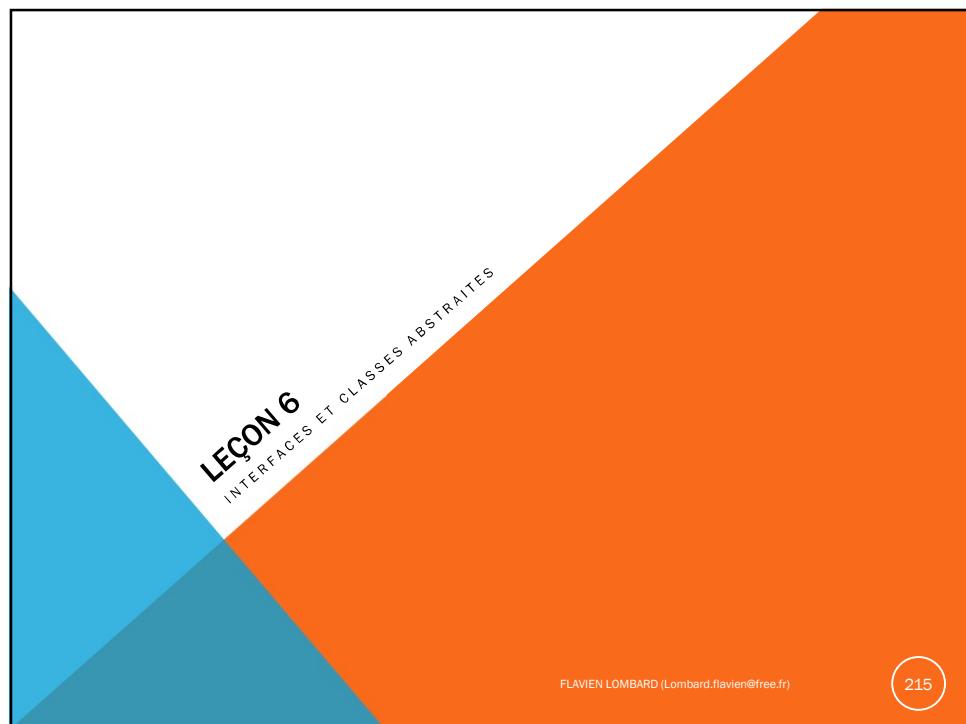
213

LES MODIFICATEURS D'ACCÈS

Exercices

- Seule la classe **Véhicule** peut générer un numéro de série lors de la construction de l'objet.
- Tout le monde peut **démarrer** un véhicule.
- Les réparateurs ont besoin de savoir le nombre de roues d'une voiture
- Les réparateurs n'ont pas besoin de connaître la vitesse de croisière d'un avion
- Les fonctions **AjouterDuKerozen()** et **AjouterDuGazoil()**, ont besoin de connaître la **QuantitéDeCarburant**, mais les réparateurs ne doivent pas le savoir.
- Par contre, les réparateurs peuvent ajouter du carburant
- Tout le monde peut prendre un RDV auprès d'un réparateur.
- Par contre seul la fonction **AjouterUnRDV()** peut vérifier les disponibilités
- Tous le monde peut vérifier les tarifs d'un réparateur
- Les réparateurs peuvent modifier la date d'entretien d'un véhicule.

214



INTERFACES

Interface = Contrat

Les classes qui héritent d'une interface ont obligatoirement les méthodes définies dans cette interface.

Exemple :

On peut définir une interface **IGestionDesRoues**, avec la méthode **CalculerPressionDesPneus()**

Cette interface sera appliquée sur **Moto**, **Voiture**, **Avion** mais pas **Helico** ni **Bateau**.

217

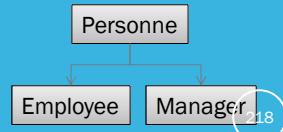
CLASSE ABSTRAITE

Une classe abstraite est une classe qui ne peut pas être instanciée.

Une classe abstraite est une classe qui peut contenir des méthodes non implémentées.

Obligation aux classes dérivant d'une classe abstraite à l'implémentation.

Ex : La classe personne est une classe abstraite. Car une personne est forcément un employé ou un manager



BONNES PRATIQUES

	Définition
S	<u>Responsabilité unique (Single Responsibility Principle)</u> une classe, une fonction ou une méthode doit avoir une et une seule responsabilité
O	<u>Ouvert/fermé (Open/closed principle)</u> une classe doit être ouverte à l'extension, mais fermée à la modification
L	<u>Substitution de Liskov (Liskov substitution Principle)</u> une instance de type T doit pouvoir être remplacée par une instance de type G, tel que G sous-type de T, sans que cela ne modifie la cohérence du programme
I	<u>Ségrégation des interfaces (Interface segregation principle) (en)</u> préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale
D	<u>Inversion des dépendances (Dependency Inversion Principle)</u> il faut dépendre des abstractions, pas des implémentations

Autres principes

DRY : Don't repeat yourself

KISS : Keep it simple and stupid

YAGNI : You aren't gonna need it

219

EXERCICES

Ecole :

Concevoir un projet objet permettant de représenter les différents acteurs gérés par une école.

Nous voulons ajouter dans une école des élèves, des profs et des administratifs.

Les élèves ont un nom, un prénom, une moyenne et peuvent s'afficher. Les profs ont un nom, un prénom, une matière et peuvent s'afficher. Les Administratifs ont un nom, un prénom, une fonction et peuvent s'afficher.

Une école a un nom. Il est possible d'ajouter dans une école et d'afficher tous les membres de l'école.

Décrire l'architecture objet du projet : graphe d'héritage, composition et interface de chaque classe.

Rajouter ensuite à notion de Salarie, (un élève n'est pas un salarié)



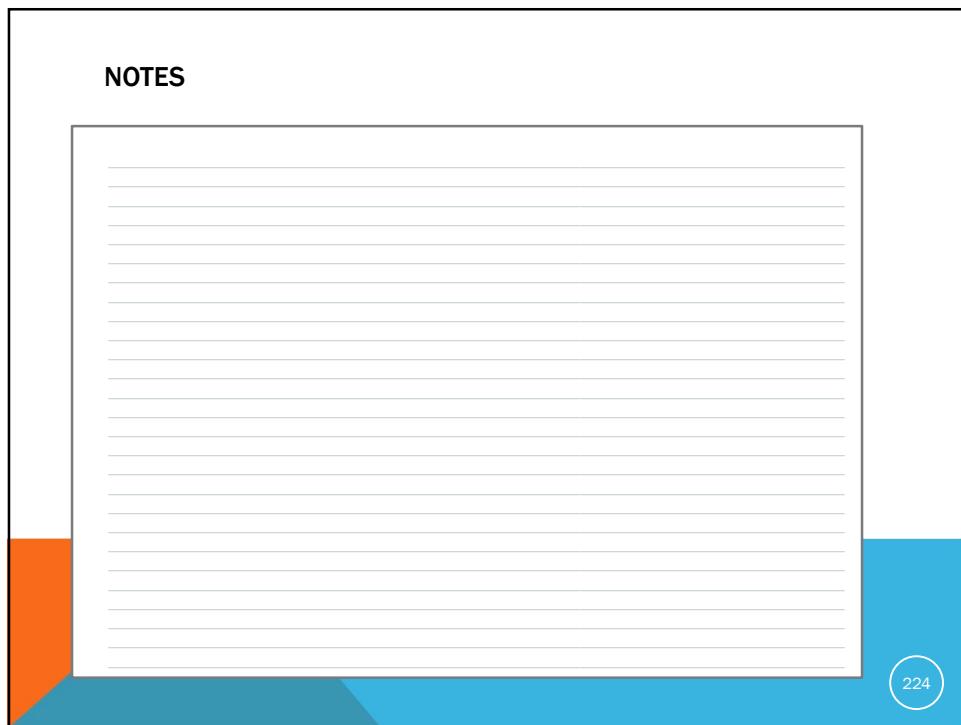
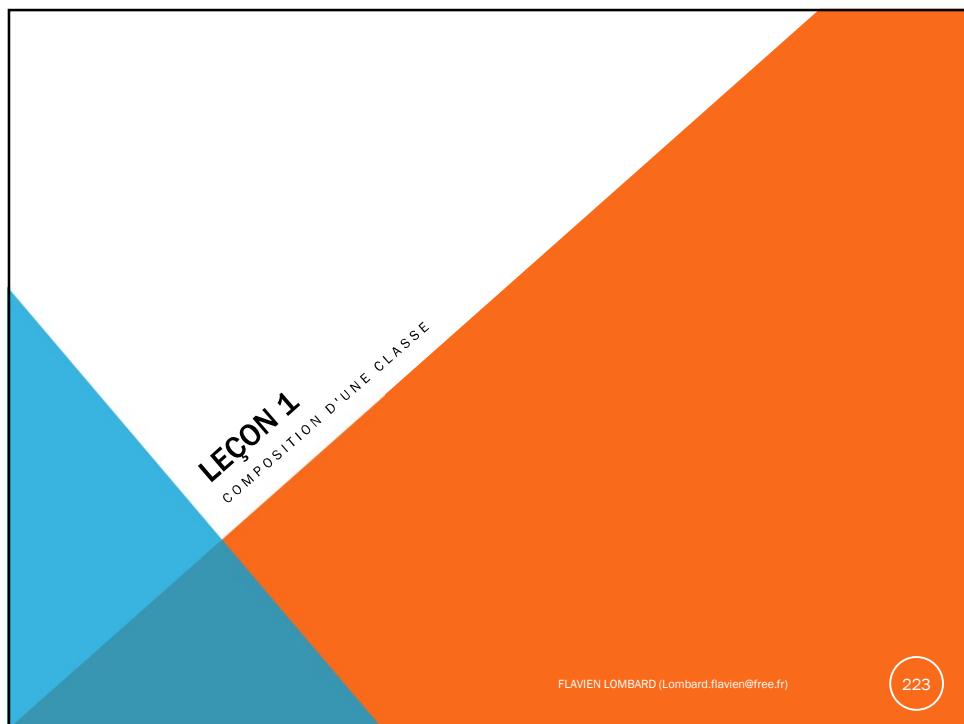
220



NOTES

A large rectangular area representing a notepad with horizontal ruling lines. The notepad is positioned in the center of a white rectangular frame. The frame has a thin black border. The corners of the notepad and the frame are rounded. The notepad is set against a background that features a blue base and orange side panels, matching the design of the module cover above.

222



COMPOSITION D'UNE CLASSE EN CSHARP

Propriétés
Champs (sur instances)
Méthodes (sur instance)
Constructeur
Champs de classe (Champs statiques)
Méthodes de classe (Méthodes statiques)
Indexeurs
Destructeur

225

DÉFINITION D'UNE CLASSE

Utilisation des champs/Méthodes statiques directement sur la classe.

Utilisation des champs/Méthodes sur un objet instancié.

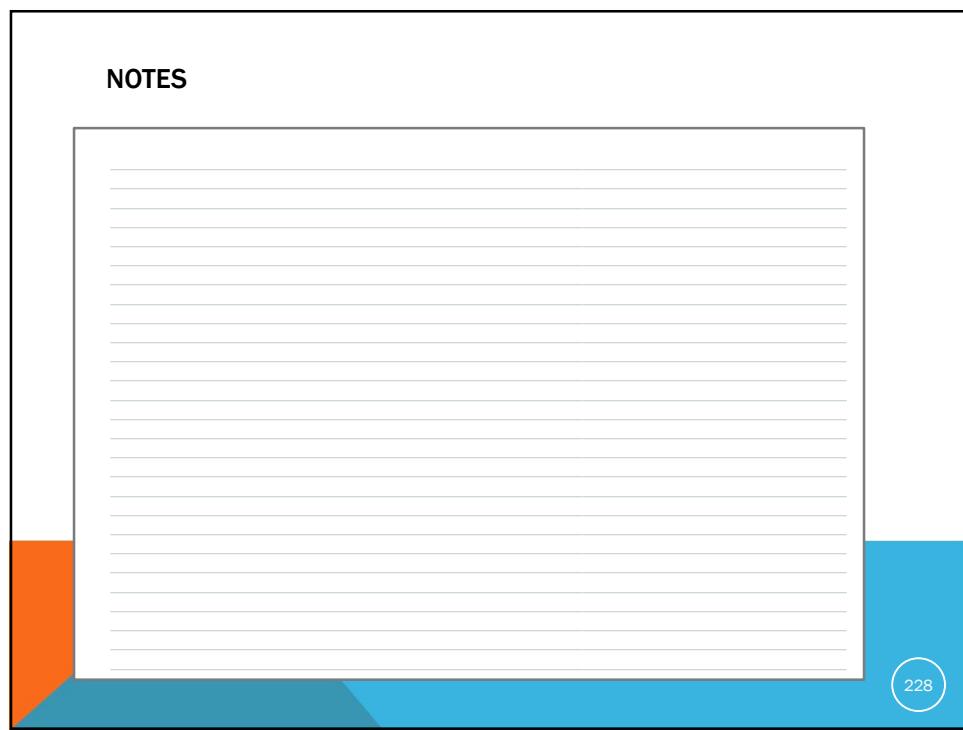
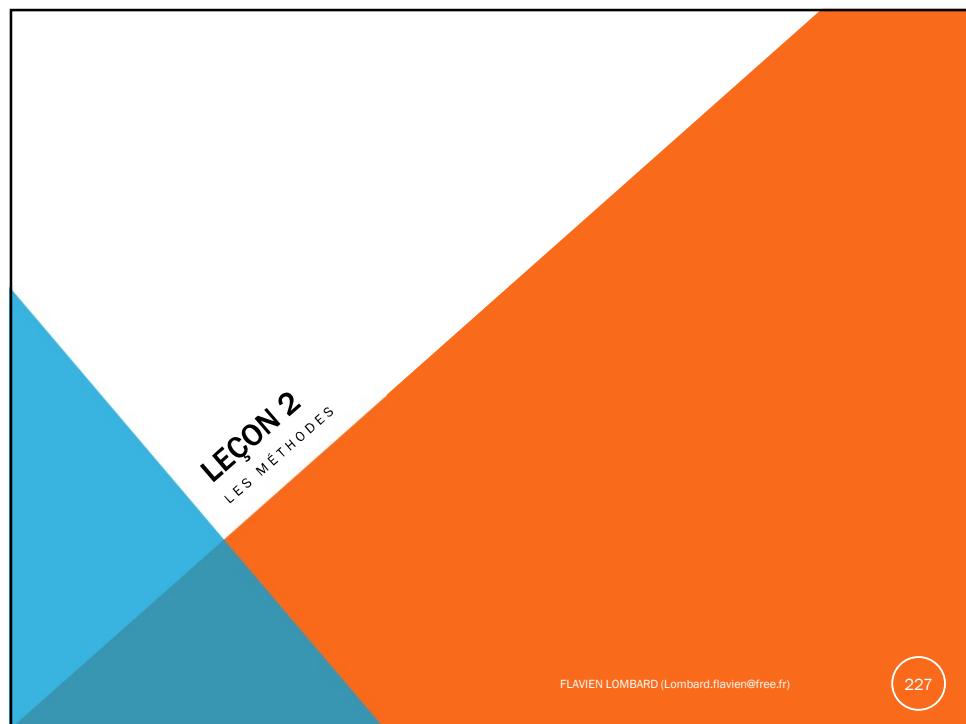
```
public class Personne
{
    public DateTime DateDeNaissance;

    public static double Getage(Personne p)
    {
        return DateTime.Now.Year - p.DateDeNaissance.Year;
    }
}
```

```
Personne p = new Personne();
p.DateDeNaissance = new DateTime(1980, 04, 04);

double age = Personne.Getage(p);
```

226



LES MÉTHODES D'INSTANCE

Les méthodes permettent les traitements dépendant des données de l'objet

```
public string GetInitials()
{
    return this.Nom[0].ToString() + this.Prenom[0].ToString();
}
```

229

LES MÉTHODES DE CLASSE

Les Méthodes de classe, mot clé **static**
Indépendant des données de l'instance

Ici la TVA est commune à tous les articles.
Alors que le prix est spécifique à un article

```
class Article
{
    public string nom;
    public double prixUnitaire;

    public static double GetTVA()
    {
        return 0.20;
    }
}
```

230

SURCHARGE DE MÉTHODES

Possibilité de mettre 2 méthodes avec le même nom mais avec des paramètres différents

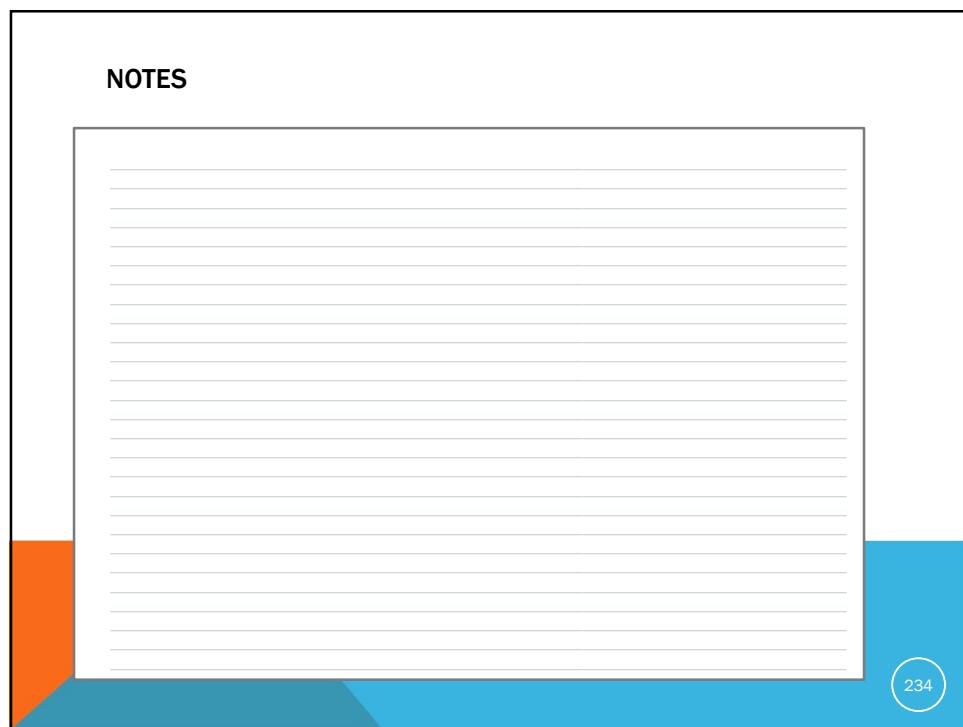
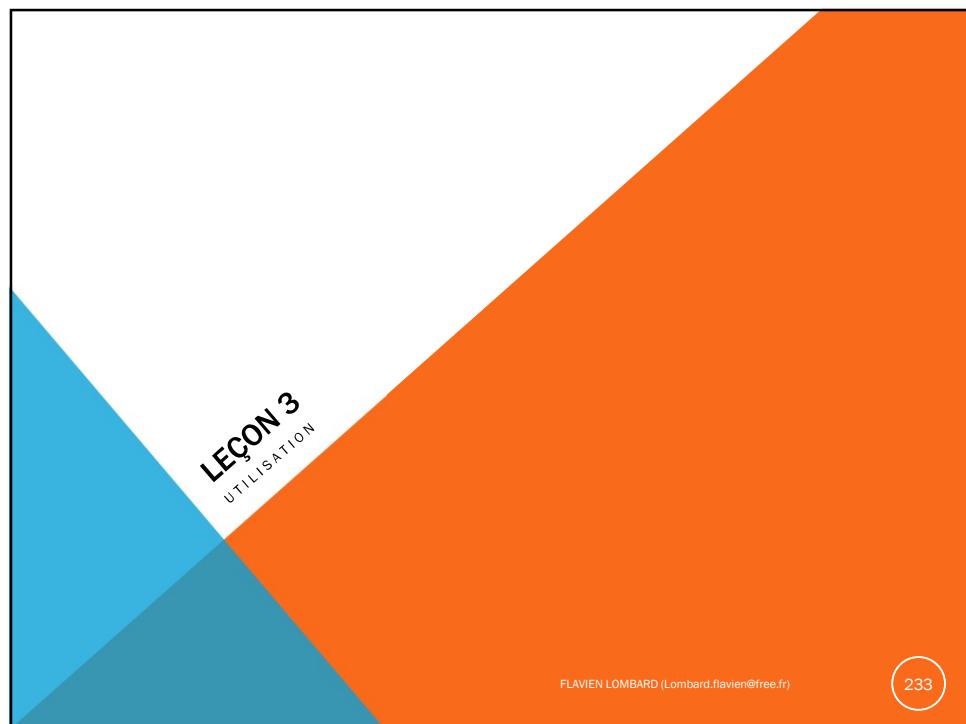
```
public Article CreationArticle()
{
    return new Article();
}

public Article CreationArticle(string nom)
{
    Article art = new Article();
    art.Nom = nom;

    return art;
}
```

231

232



UTILISATION D'UNE CLASSE

Le **Garbage collector** s'occupe de la libération de la mémoire

```
static void Main(string[] args)
{
    Personne p = new Personne();
    p.DateDeNaissance = new DateTime(1980, 04, 04);

    double age = Personne.Getage(p);

    p = null;
}
```

Il est possible d'aider le Garbage collector en déréférençant les objets au plus tôt.

La bonne pratique en csharp est de créer un objet juste avant son utilisation, et de le libérer dès que l'on s'en sert plus

235

LES CONSTRUCTEURS

Initialisation lors de l'instanciation.

Permet d'initialiser et de sécuriser la création d'objets

```
public class Personne
{
    public Personne()
    {
        Nom = "Toto";
        Prenom = "Jean";
    }

    public Personne(string nom)
    {
        Nom = nom;
        Prenom = "Jean";
    }
}
```

Un constructeur peut être statique

236

EXPRESSION D'INITIALISATION

Nouveauté C#3

Seul les membres publiques sont accessibles

```
Personne p1 = new Personne();
p1.DateDeNaissance = new DateTime(1980, 04, 04);
p1.Nom = "Toto";
p1.Prenom = "Jean";

Personne p2 = new Personne() { Nom = "Toto", Prenom = "Jean" };
```

237

TYPES ANONYMES

Nouveauté C#3

Créé pour LINQ, très utiles pour manipuler des données d'un certain type temporairement.

```
var p3 = new { LastName = "Toto", FirstName = "Jean", DateOfBirth = "05/05/1990" };
```

238

DESTRUCTEUR

Méthode particulière appelée lors de la libération d'un objet par le **Garbage Collector**

```
~Personne()
{
    Console.WriteLine(Nom + " vous dit bye bye");
}
```

239

LES ÉGALITÉS

Qu'est ce qui doit être égal ?

- L'objet ? -> Utilisation de **==**
- La valeur de l'objet? -> Utilisation de **Object.Equals**

La méthode **Object.Equals**, utilise la méthode **GetHashCode()**, pour calculer une valeur de comparaison

240

VISIBILITÉ DES CLASSES

Public

- La classe est visible par toutes les autres classes

internal

- La classe est visible uniquement par les classes de son assembly
(valeur par défaut)

241

VISIBILITÉ DES MEMBRES

Visibilité Publique

- Membres accessible par tous les objets
- Mot clé `public`

Visibilité Privé

- Membre accessible uniquement par sa classe
- Mot clé `private`

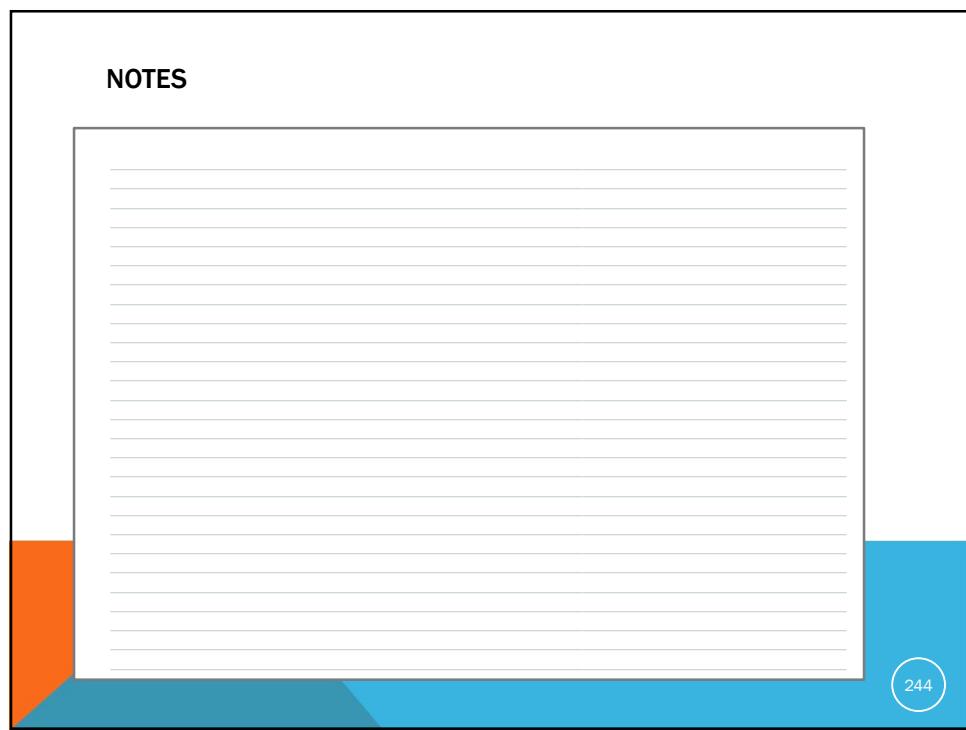
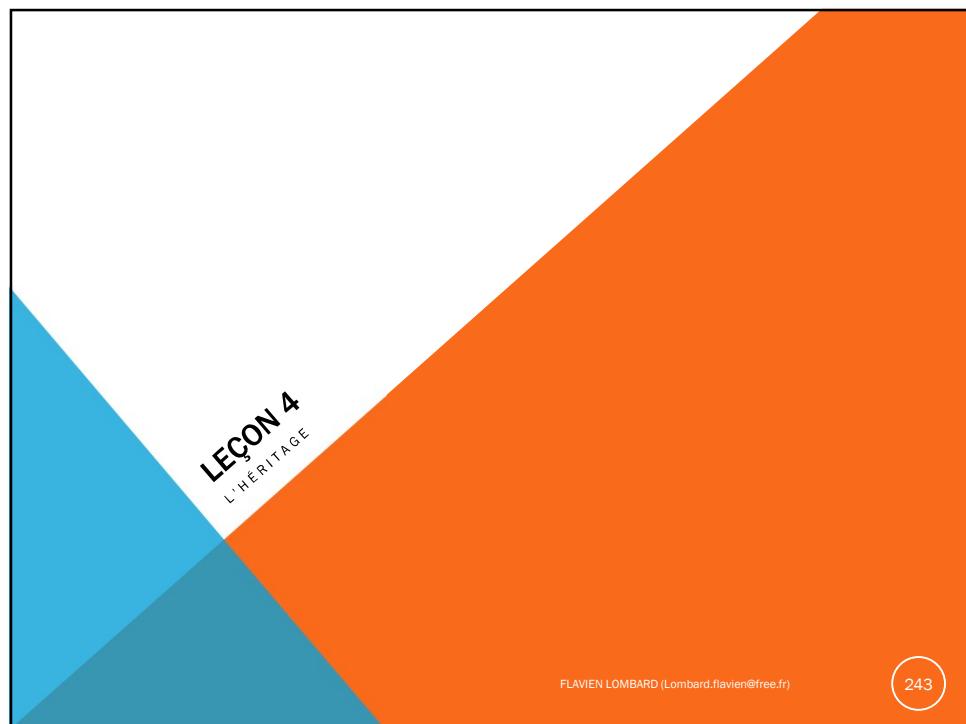
Visibilité Protégé

- Membre accessible uniquement par sa classe et ses classes dérivées
- Mot clé `protected`

Visibilité Interne

- Membre accessible dans l'assembly
- Mot clé `internal`

242



HÉRITAGE

Une classe ne peut hériter que d'une seule autre classe.
La classe dérivée hérite des membres (publiques et protégés) de la classe de base.

```
class Employee:Personne
{
    double Salaire;
}
```

245

HÉRITAGE

Une classe hérité est de type de la classe de base.
Une classe de base n'est pas considéré comme étant d'un type hérité

```
Personne John = new Employee("John",2300);
Employee personne = new Personne();
Personne.Personne() (+ 1 surcharge(s))

Erreur:
Impossible de convertir implicitement le type 'ConsoleApplication2.Personne' en 'ConsoleApplication2.Employee'.
}
```

246

HÉRITAGE DU CONSTRUCTEUR

Initialisation de la classe parent lors de l'instanciation d'une classe dérivée.

```
Employee(string nom, double salaire)
    : base(nom)
{
    Salaire = salaire;
}
```

247

SURCHARGE ET POLYMPHISME

Méthode de même signature dans la classe dérivée.

Utilisation du mot clé **new** optionnel mais conseillé

248

SURCHARGE ET POLYMPHISME

Utilisation de méthodes virtuels

```
public class Personne
{
    virtual void Afficher()
    {
        //Affiche les infos de la personne
    }
}
```

```
class Employee : Personne
{
    public override void Afficher()
    {
        Console.Out.WriteLine("infos de l'employee");
    }
}
```

249

TEST DU TYPE

Le type de l'objet peut être testé avec le mot clé **is**

```
object[] datas = new object[100];
//...
for(int i=0;i<100;i++)
{
    if (datas[i] is Article)
    {
        //Do stuff
    }
}
```

Il est également possible de connaître le type par réflexion

```
for(int i=0;i<100;i++)
{
    if (datas[i].GetType() == typeof(Article))
    {
```

250

INTERDIRE L'HÉRITAGE

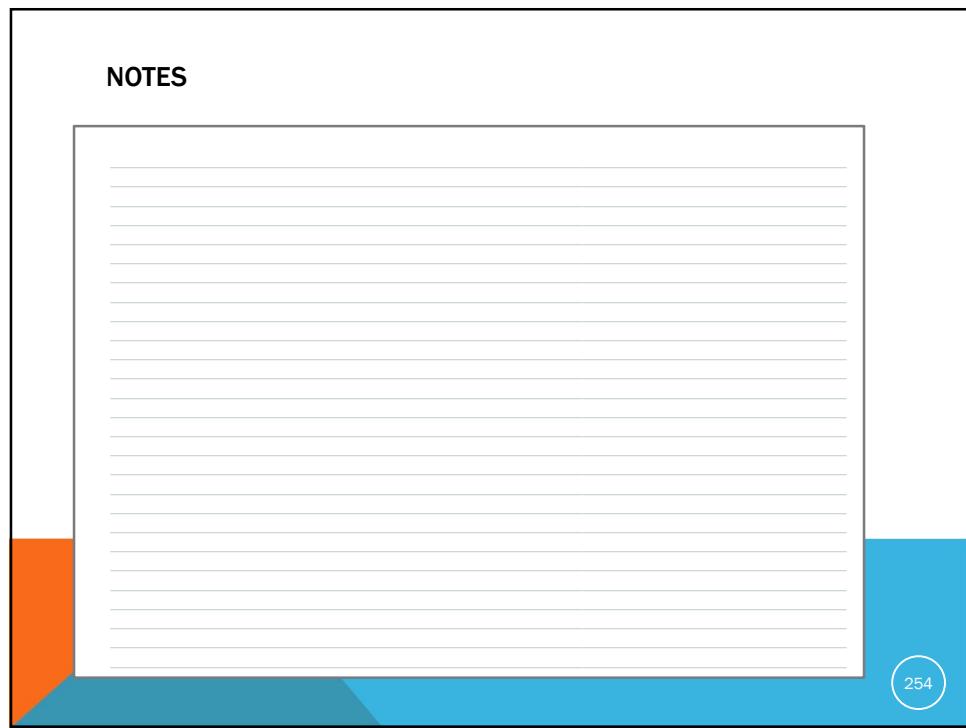
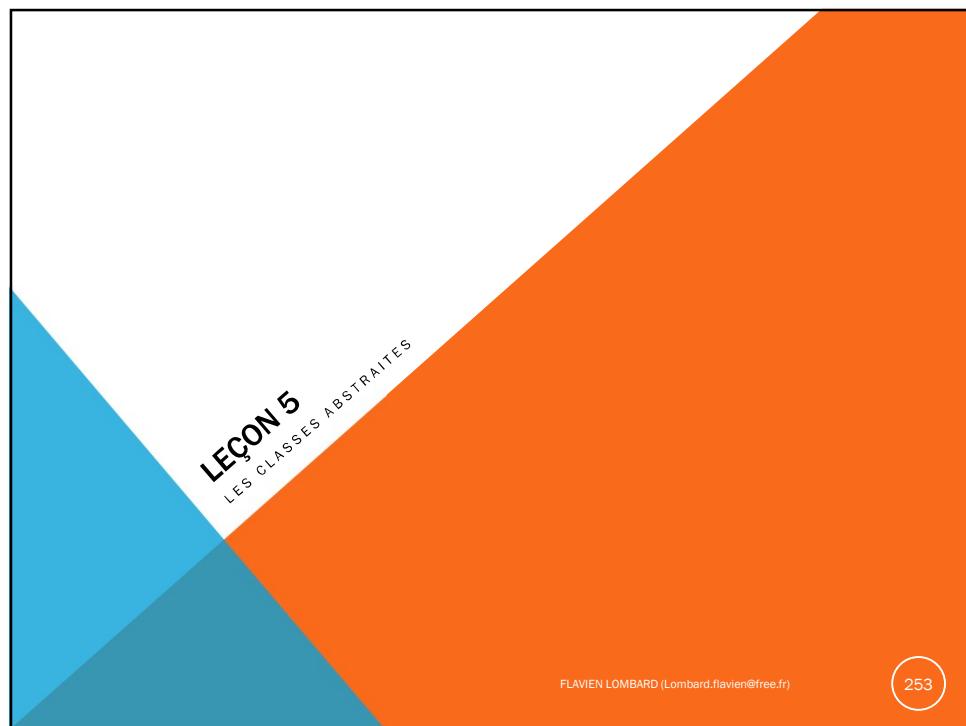
Utilisation du mot clé **sealed**

```
public sealed class Personne  
{
```

```
class Employee:Personne  
{  
    'ConsoleApplication2.Employee': impossible de dériver du type sealed 'ConsoleApplication2.Personne'
```

251

252



LES CLASSES ABSTRAITES

Classes dont certains membres ne sont pas implémentés.

Une classe abstraite ne peut pas être directement instanciée.

Obligation de définir les membres abstraits lors de la dérivation.

Mots clé `abstract` et `override`

```
public abstract class Personne  
{  
    public abstract double GetSalaire();
```

```
class Employe : Personne  
{  
  
    public override double GetSalaire()  
    {  
        return 1200;  
    }  
}
```

255

INTERFACES

Contrats pour les classes

Ne peut pas être instancié

Pas de champs

Pas de corps de méthode

Une Interface définit des membres public uniquement

Mettre un I par convention

256

INTERFACES

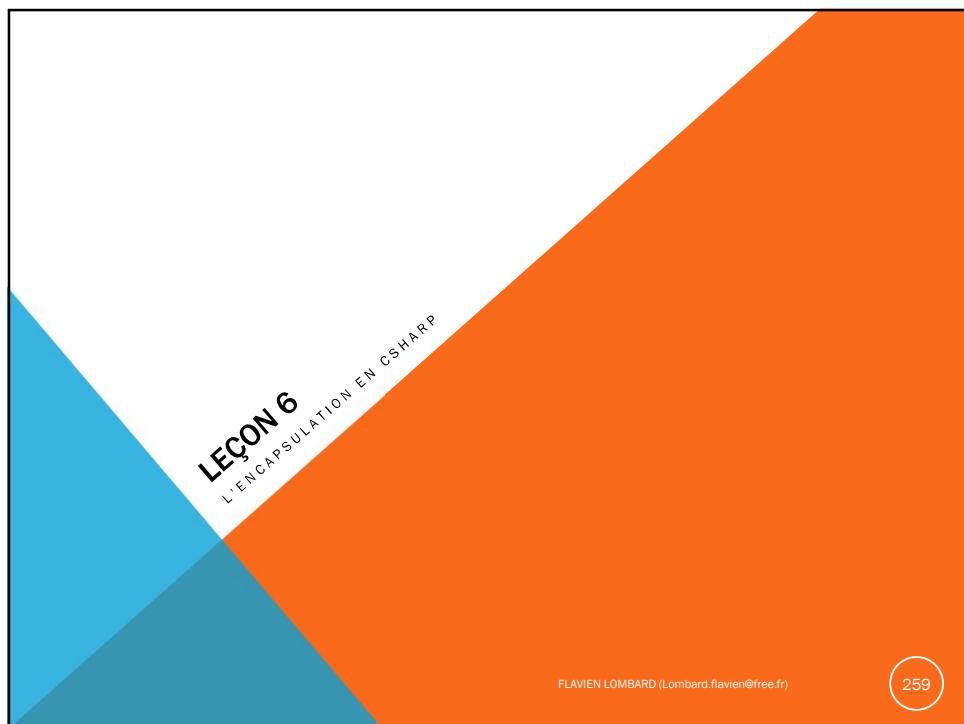
Utilisation par héritage comme pour une classe abstraite

```
interface ISalarie
{
    double GetSalaire();
}
```

```
class Employee:Personne,ISalarie
{
    public double GetSalaire()
    {
        return 1200;
    }
}
```

257

258



NOTION D'ENCAPSULATION

L'encapsulation permet de protéger la donnée en évitant qu'une variable ne prenne des valeur non souhaitées

```
public class Eleve : Personne
{
    private int note; ←
    public int GetNote()
    {
        return note;
    }

    public void SetNote(int nte)
    {
        if (nte >= 0 && nte <= 20) note = nte;
        else throw new ArgumentException("La note doit être compris en 0 et 20");
    }
}
```

La note sera toujours entre 0 et 20

261

PROPRIÉTÉS

Exécute une action lors de l'affectation ou la récupération d'une variable.

Une propriété améliore la lecture et l'usage de la donnée

```
private int note;
public int Note
{
    get {
        return note;
    }
    set {
        if (value >= 0 && value <= 20) note = value;
        else throw new ArgumentException("La note doit être compris en 0 et 20");
    }
}
```

Une propriété permet d'encapsuler un champ. Et donc de protéger les valeurs possibles prisent par ce champ

262

VISIBILITÉ DES PROPRIÉTÉS

Nouveauté C# 2.0

Les accesseurs **get** et **set** peuvent également avoir des modificateurs d'accès.

263

PROPRIÉTÉS AUTOMATIQUES

Nouveauté de C# 3.0

```
public int Id { get; private set; }
public string Nom { get; set; }
public string Prenom { get; set; }
```

264

INDEXEURS

Permet à un objet de se "comporter" comme un tableau

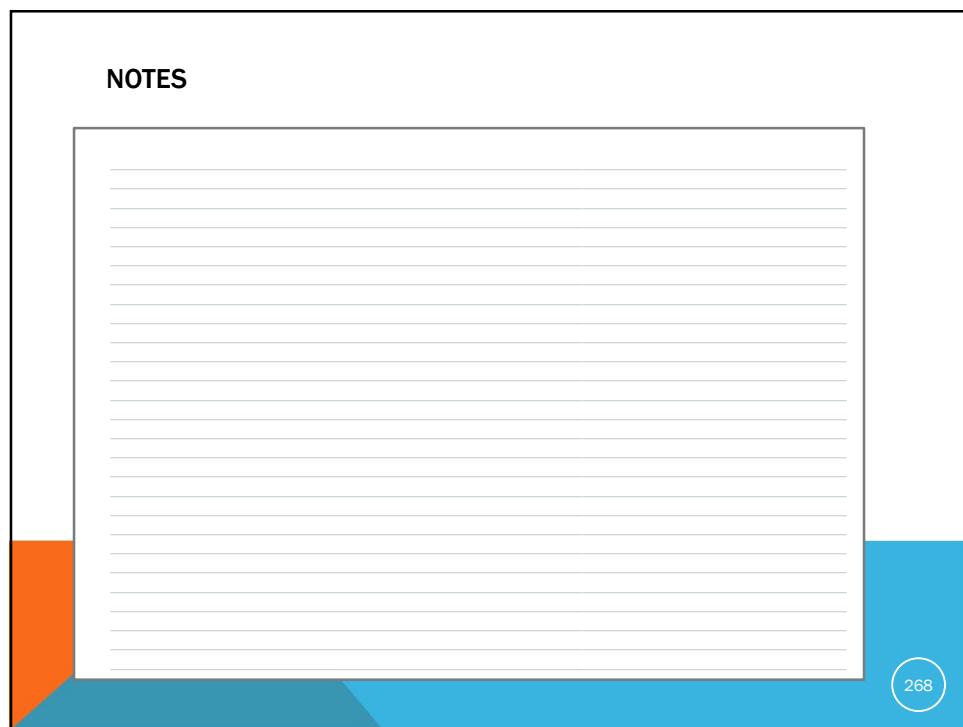
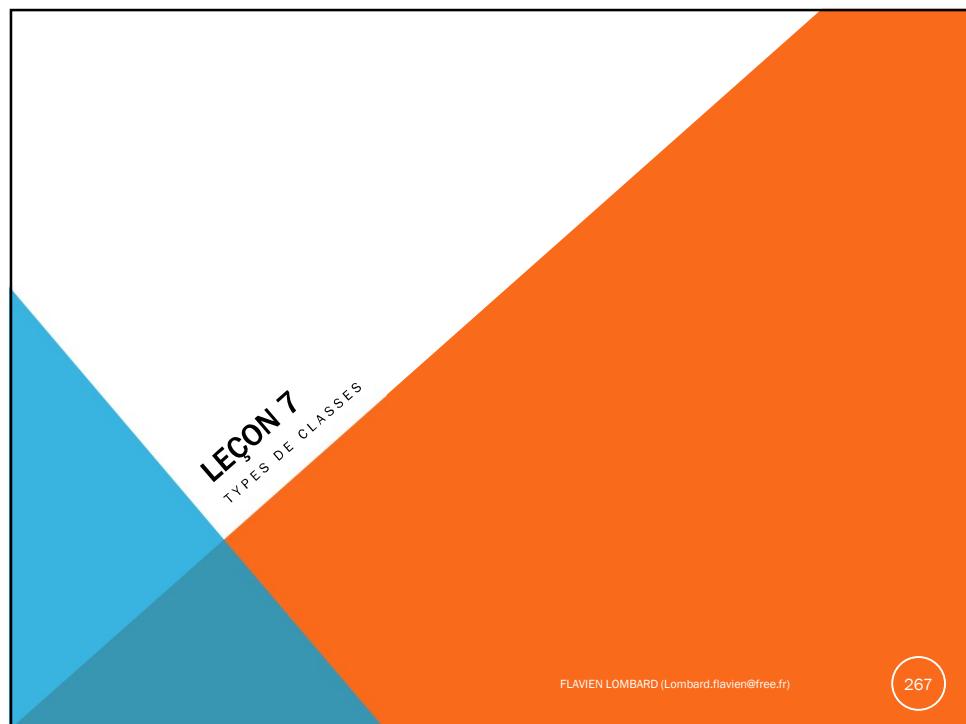
```
public class ListPersonnes
{
    List<Personne> lst = new List<Personne>();

    public Personne this[int idx]
    {
        get
        {
            return lst[idx];
        }
    }
}
```

```
ListPersonnes lst = new ListPersonnes();
Personne p = lst[4];
```

265

266



CLASSES PARTIELLES

Nouveauté C# 2.0

Très utilisé par VS pour la génération de code

Facilite le travaille en équipe

Mot clé partial (Partial)

269

LES CLASSES STATIQUES

Nouveauté de C# 2.0

Instanciation impossible.

Tous les membres sont également statiques

Mots clés static

```
public static class MgmtPersonnes
{
    static void AddPersonne(Personne p)
    {
        //Ajout d'une personne
    }
}
```

270

EXERCICES

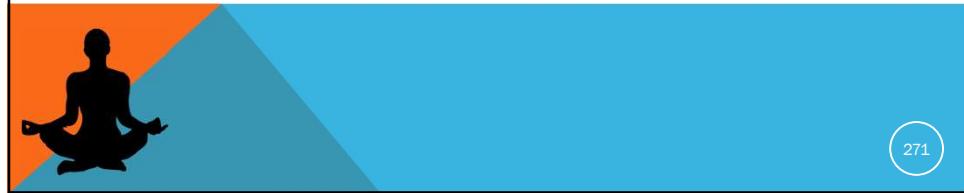
Ecole :

Création d'un projet console, et d'un projet bibliothèque de classe.
Le projet console faisant référence au projet de bibliothèque de classe.

Création d'un dossier de "BusinessObject" et d'un dossier "BusinessLayer" dans le projet Bibliothèque de classe.

Implémentation des classes de l'école vu précédemment en csharp dans le dossier "BusinessObject"

Implémentation de la classe "EcoleService" contenant la méthode "Inscrire" et "Embaucher" permettant d'inscrire un élève et d'embaucher un prof ou un administratif.
Ces méthodes se contentent d'ajouter les membres dans une liste contenue dans EcoleService



271



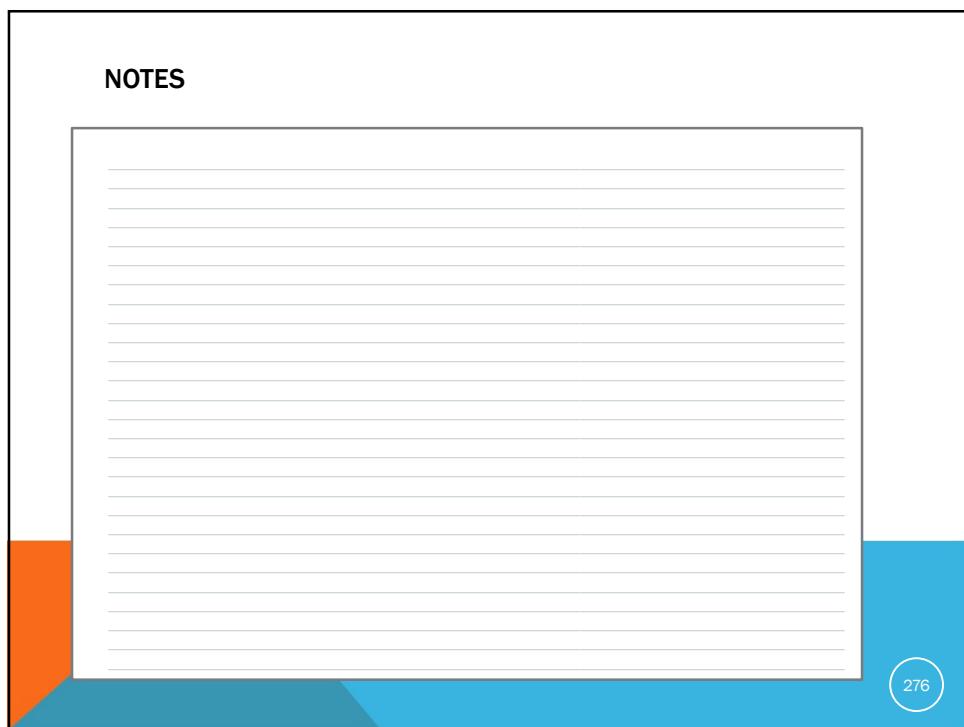
272



NOTES

A rectangular area containing horizontal lines for writing notes. It is positioned in the center of a larger blue and orange background, which is part of the module cover design.

274



MÉTHODES D'EXTENSION

Utiliser pour étendre plusieurs classes dérivant d'une interface

Utiliser pour étendre une classe dont on a pas les sources.

Méthode statique

Mot clé « this »

Le corps de la méthode d'extension peut être paramétrable avec des expressions lambda

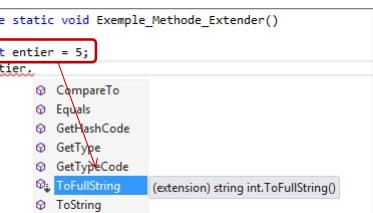
277

MÉTHODES D'EXTENSION

Ajout de la fonction 'ToFullString()' au type 'int'

```
public static class IntExtension
{
    public static string ToFullString(this int nombre)
    {
        return nombre.ToString() + " est un entier";
    }
}
```

```
private static void Exemple_Methode_Extender()
{
    int entier = 5;
    entier..
```



278

MÉTHODES D'EXTENSION

```
public static class IDictionaryExtention
{
    public static TValue GetValue<TKey, TValue>(
        this IDictionary<TKey, TValue> aList, TKey key, TValue defaultvalue)
    {
        TValue valeurretour;

        if (aList.ContainsKey(key))
            valeurretour = aList[key];
        else
            valeurretour = defaultvalue;

        return valeurretour;
    }
}
```

```
Dictionary<int, string> dico = new Dictionary<int, string>();
dico.Add(1, "Formation .NET");
dico.Add(2, "Formation WinForm");
dico.Add(3, "Formation WebForm");
dico.Add(4, "Formation MVC");

Console.WriteLine(dico.GetValue(34, "Formation inconnue !"));
```

Formation inconnue !

279

MÉTHODES D'EXTENSION

Cas particulier:

Méthode étendue masqué par une méthode de la classe de même signature.

1 classe ayant 2 interfaces possédant chacun 1 extension de même signature, provoque une erreur.

280

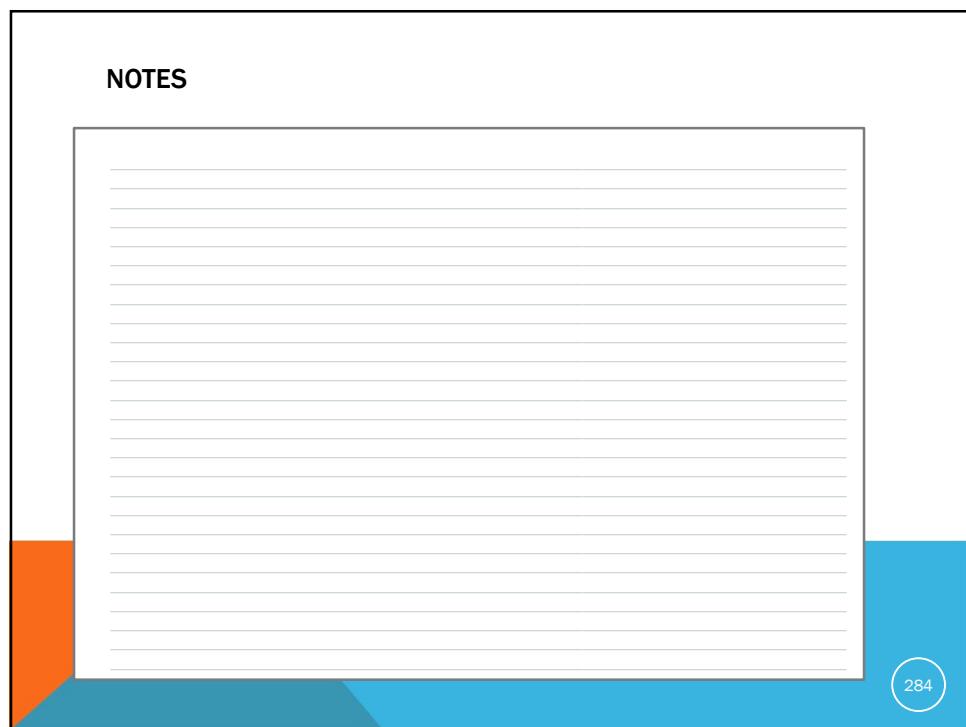
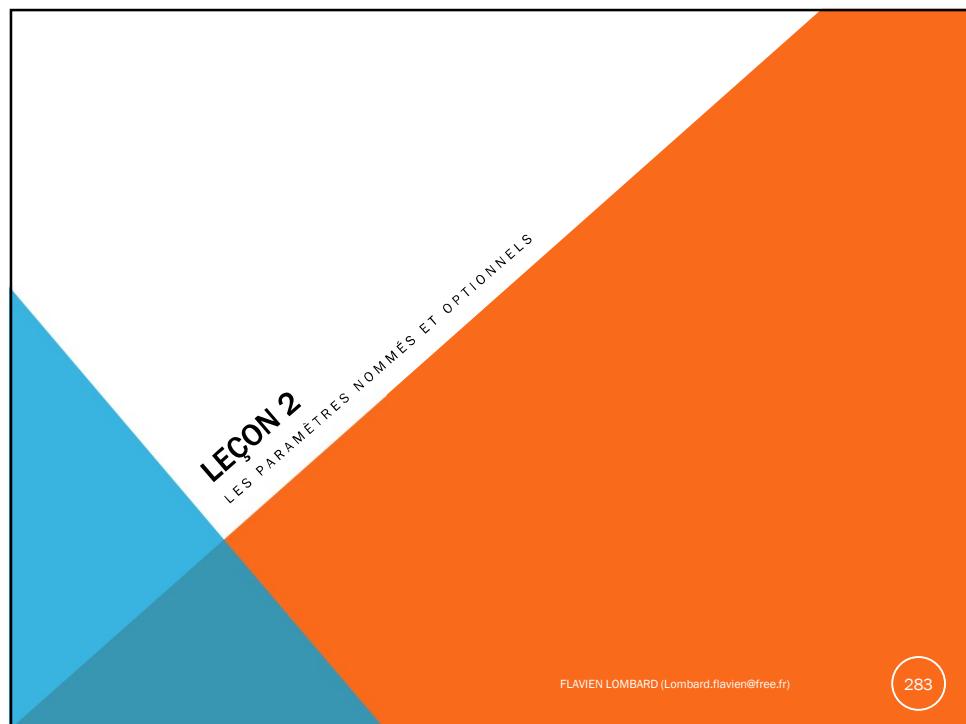
MÉTHODES D'EXTENSION

Quelques méthodes d'extension incluses au framework.

- All
- Any
- Concat
- Count
- Select
- Where
- Distinct
- First
- FirstOrDefault
- GroupBy
- Intersect
- Join
- OrderBy, OrderByDescending, ThenBy, ThenByDescending
- ToList

281

282



PARAMÈTRES NOMMÉS ET OPTIONNELS

Souplesse d'écriture du code.
Méthodes, constructeurs et indexeurs

```
public static void Calculer(int x, int y = 10, int z = 20)
{
    Console.Out.Write(x + " + " + y + " + " + z + " = ");
    Console.Out.WriteLine((x + y + z).ToString());
}
```

285

PARAMÈTRES NOMMÉS ET OPTIONNELS

Si un paramètre est optionnel, tous les autres paramètres doivent l'être également

```
public static void Calculer(int x, int y = 10, int z)
{
    Console.Out.Write(x + " + " + y + " + " + z + " = ");
    Console.Out.WriteLine((x + y + z).ToString());
}
```

Impossible

286

PARAMÈTRES NOMMÉS ET OPTIONNELS

L'utilisation des 'options' se fait en omettant le paramètre lors de l'appel.

```
Calculateur.Calculer(2);
Calculateur.Calculer(2, 40);
Calculateur.Calculer(2, 40, 100);
```

```
Calculateur.Calculer(2, x 100);
```

Impossible

287

PARAMÈTRES NOMMÉS ET OPTIONNELS

Appel de fonction avec des paramètres dans le désordre.

Omission des paramètres optionnels.

```
Calculateur.Calculer(x:2, z:100);
Calculateur.Calculer(z:100,x:2,y:40);
```

Ok

288

PARAMÈTRES NOMMÉS ET OPTIONNELS

La résolution des conflits :

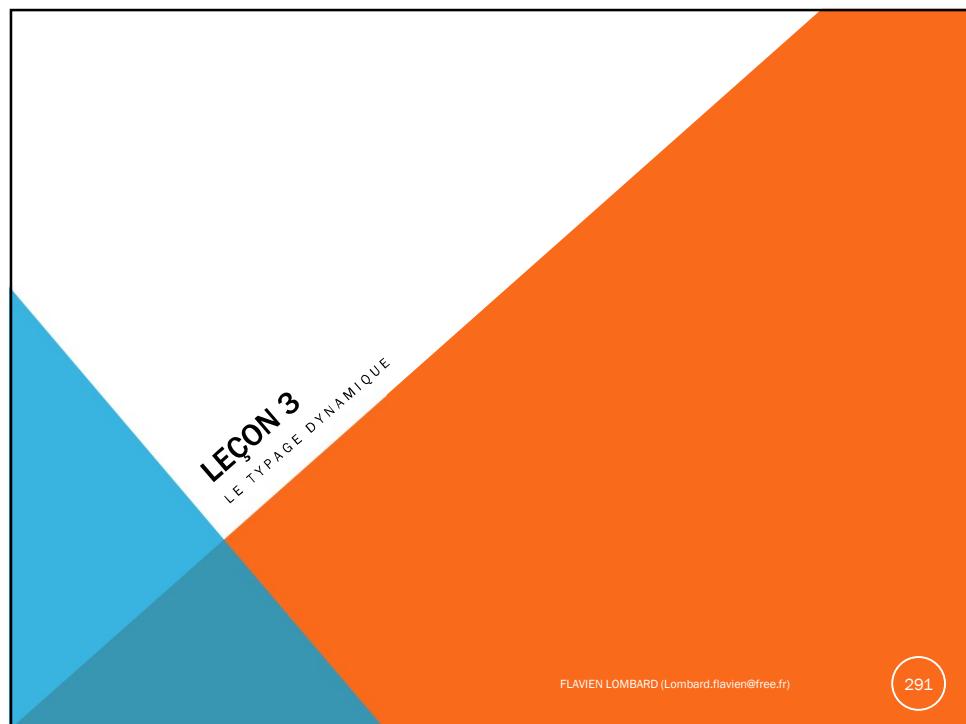
```
public static void LaunchMe(string s, int i = 1) { }
public static void LaunchMe(object obj) { }
public static void LaunchMe(int i, string s = "default") { }
public static void LaunchMe(int i) { }

public static void EnConflit()
{
    LaunchMe(14);
}
```

La méthode choisie est la méthode qui correspond le mieux à la signature de la fonction

289

290

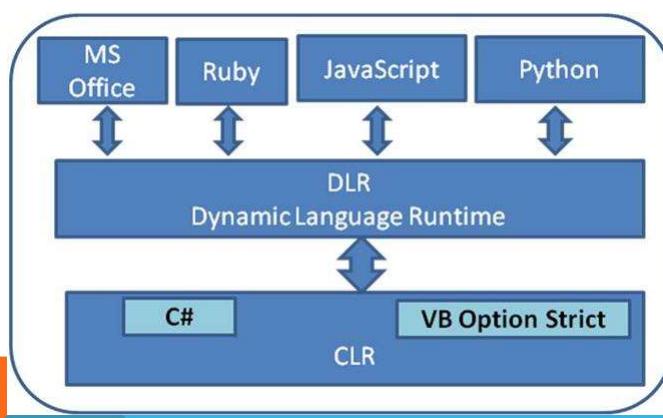


TYPAGE DYNAMIQUE ET LA DLR

Modification de la CLR pour y inclure la couche DLR
Permet l'ouverture de .NET à des Langages non fortement typés.(IronPython ou IronRuby)
Avancé majeur du Framework pour l'ouverture vers le Web

293

TYPAGE DYNAMIQUE ET LA DLR



294

C#, développer en .NET avec Visual Studio

TYPAGE DYNAMIQUE ET LA DLR

Mot clé `dynamic`

Type évalué à l'exécution

```
dynamic var1 = 5;
var1 = var1 + "Hello";
Console.Out.WriteLine("test = " + var1);
var1 = 9.5f;
var1 = var1 + 7;
Console.Out.WriteLine("total : " + var1);
```

test = 5Hello

total : 16,5

295

TYPAGE DYNAMIQUE ET LA DLR

Utilisation en tant que paramètre

```
public class Personne
{
    public string Nom { get; set; }
    public string Prenom { get; set; }
    public int Age { get; set; }
}

public class Animal
{
    public string Nom { get; set; }
    public string Age { get; set; }
}

public static class AgeClass
{
    public static void ShowAge(dynamic obj)
    {
        Console.Out.WriteLine(obj.Age);
    }
}
```

```
Personne p = new Personne() { Nom = "DOE", Prenom = "John", Age = 32 };
AgeClass.ShowAge(p);

Animal a = new Animal() { Nom = "Medor", Age = "5 Mois" };
AgeClass.ShowAge(a);

var anonymObject = new { Type = "Photon", Age = 0.0000045f };
AgeClass.ShowAge(anonymObject);

Console.ReadLine();
```

32
5 Mois
4.5E-06

296

TYPAGE DYNAMIQUE ET LA DLR

Utilisation déconseillée
Aucune vérification du compilateur
Engendre de fort risque de bug non prévu

297

TYPAGE DYNAMIQUE ET LA DLR

Utilisation d'expressions lambda indirect

```
dynamic var1 = 5;  
  
var1 = var1 + "Hello";  
  
Console.Out.WriteLine("test = " + var1);  
  
Func<int, int> lambda = (x) => x + 2;  
var1 = lambda;  
  
test(var1);
```

```
private static void test(Func<int,int> expr)  
{  
    Console.Out.WriteLine(expr(45));  
}
```

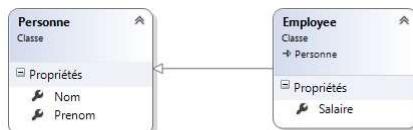
test = 5Hello
47
-

Méthode d'extension impossible avec dynamic

298

COVARIANCE/CONTRAVARIANCE

Covariance et ContraVariance existe depuis c#2 sur les délégués.



299

COVARIANCE/CONTRAVARIANCE

```
namespace System.Collections.Generic
{
    public interface IEnumerable<out T> : IEnumerable
    {
        Ienumerator<T> GetEnumerator();
    }
}

namespace System.Collections.Generic
{
    public interface Ienumerator<out T> : IDisposable, Ienumerator
    {
        T Current { get; }
    }
}
```

L'interface est covariante du type T
Ce qui signifie que tous les objets de type `IEnumerable<A>`
pourront être considérée comme un objet de type `IEnumerable`.
A la condition que le type A puisse être converti en type B

300

COVARIANCE/CONTRAVARIANCE

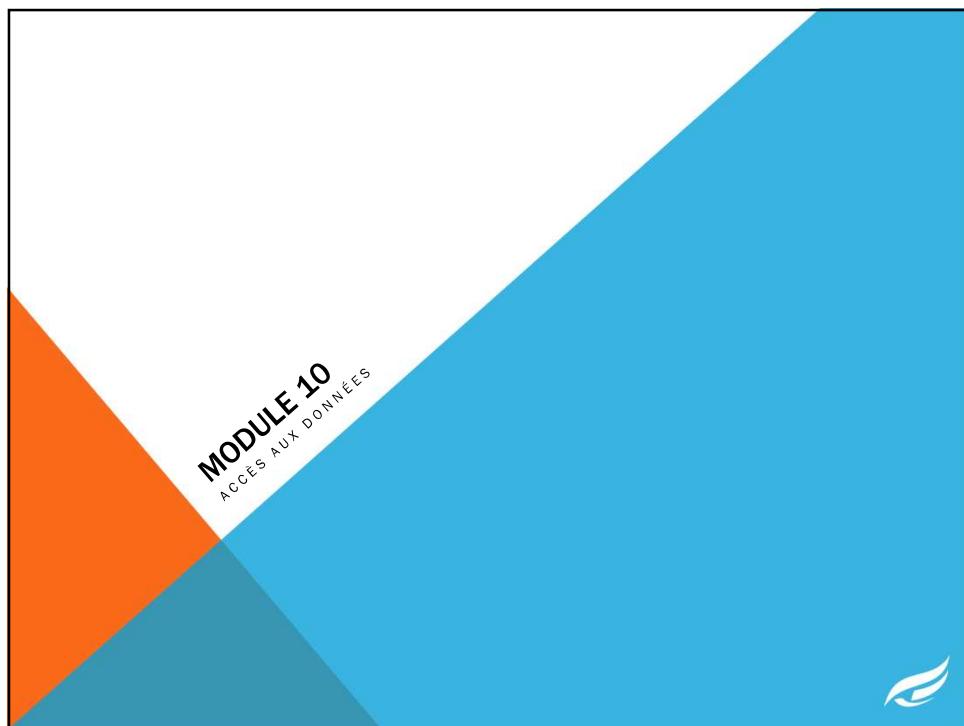
```
public interface IEqualityComparer<in T>
{
    bool Equals(T x, T y);
    int GetHashCode(T obj);
}
```

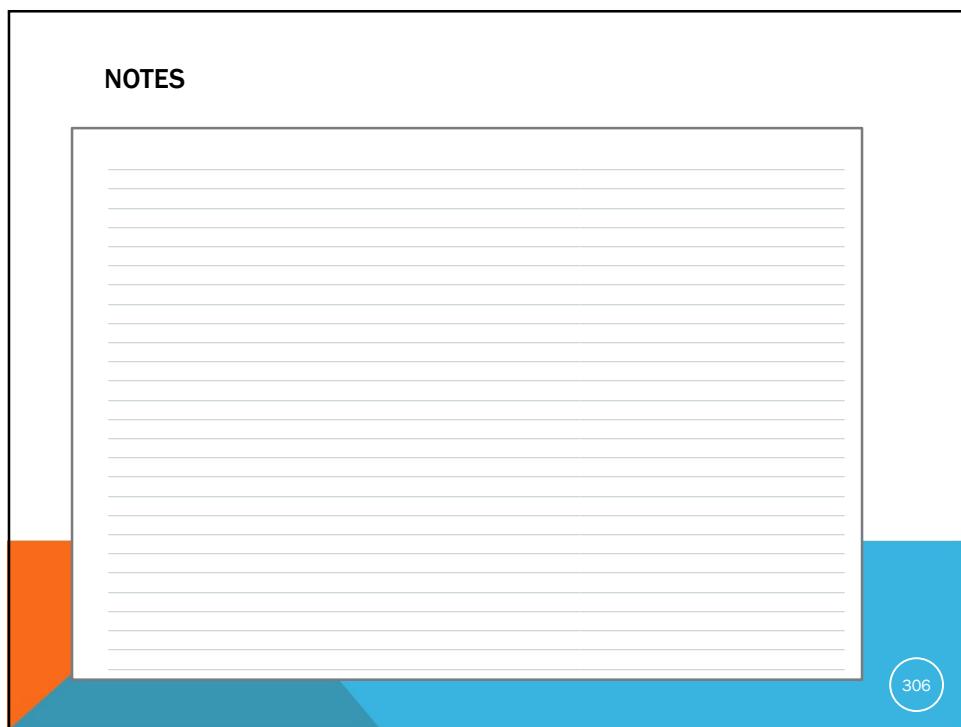
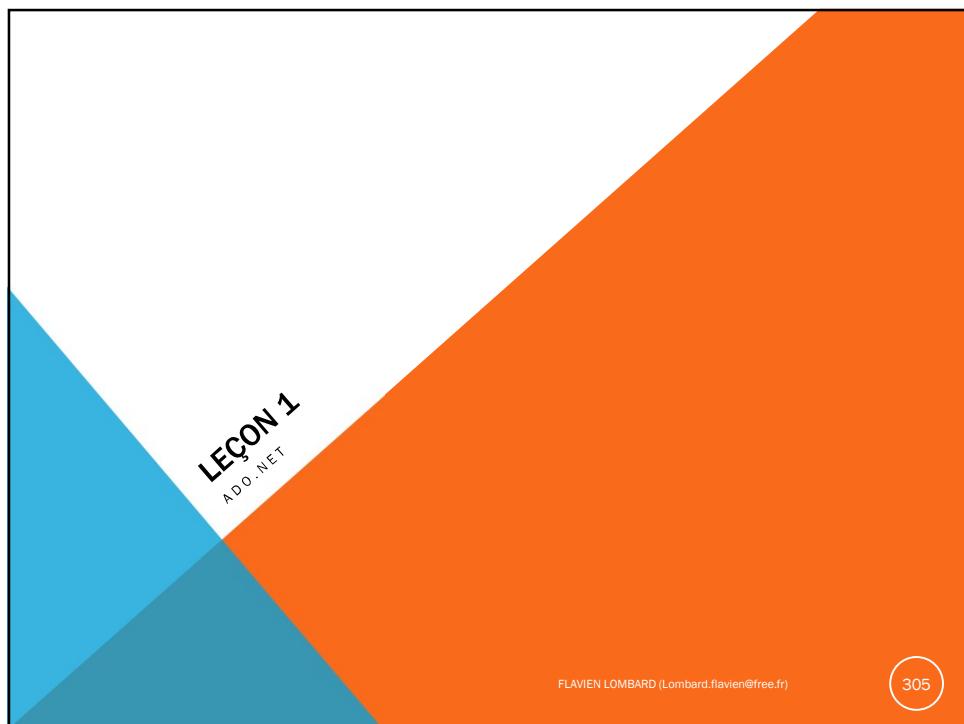
Le mot clé « in » signifie que le Type T est utilisé uniquement en paramètre dans cette Interface

C'est un contravariance du type T.

301







INTRODUCTION

L'accès au données avec .NET se fait grâce à ADO.NET

ADO.NET = System.Data

Ensemble de classes indépendantes + ensemble de classes surchargées dépendantes de la Base de données

307

PRINCIPALES CLASSES

Connection : **Connexion à la base de données**

Command : **Création de la requête SQL**

CommandBuilder : **Constructeur de command**

DataReader : **Lecteur sous forme de flux des données retournées**

DataAdapter : **Mise en forme des données retournées**

Version OLEDb, version SQLServer

ADO.NET est extensible

308

LES CLASSES « CLIENT »

Indépendantes de la BD

- DataSet :** Représente mémoire des données de la base
- DataTable :** représente les tables d'un DataSet
- DataColumn :** représente les colonnes d'un DataSet
- DataRow :** représente une ligne d'une table
- DataRelation :** représente une relation existant entre les deux tables d'un DataSet

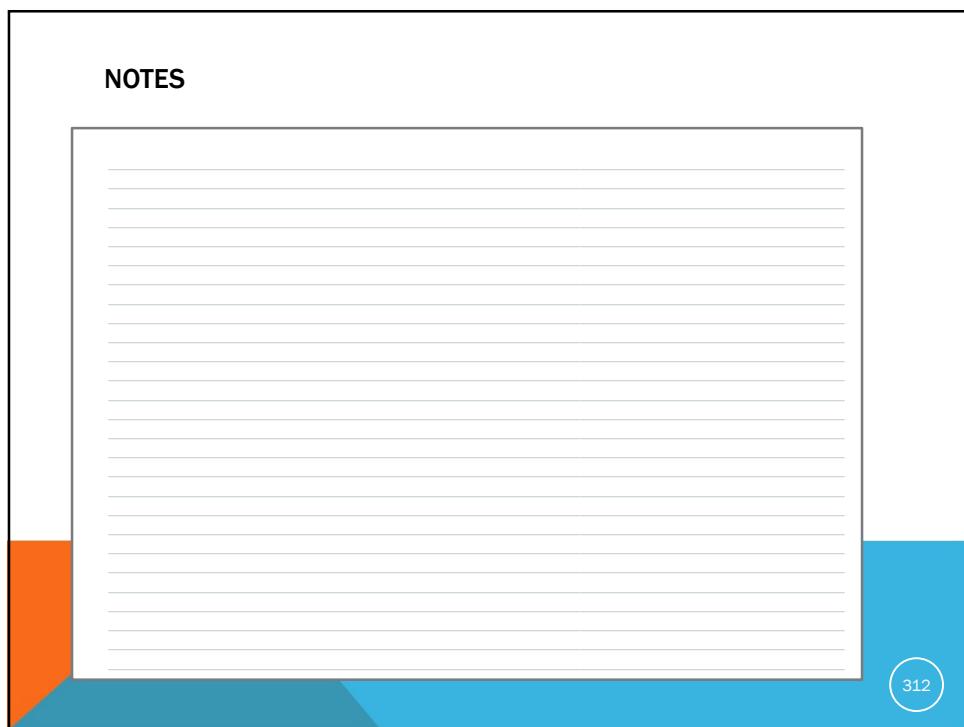
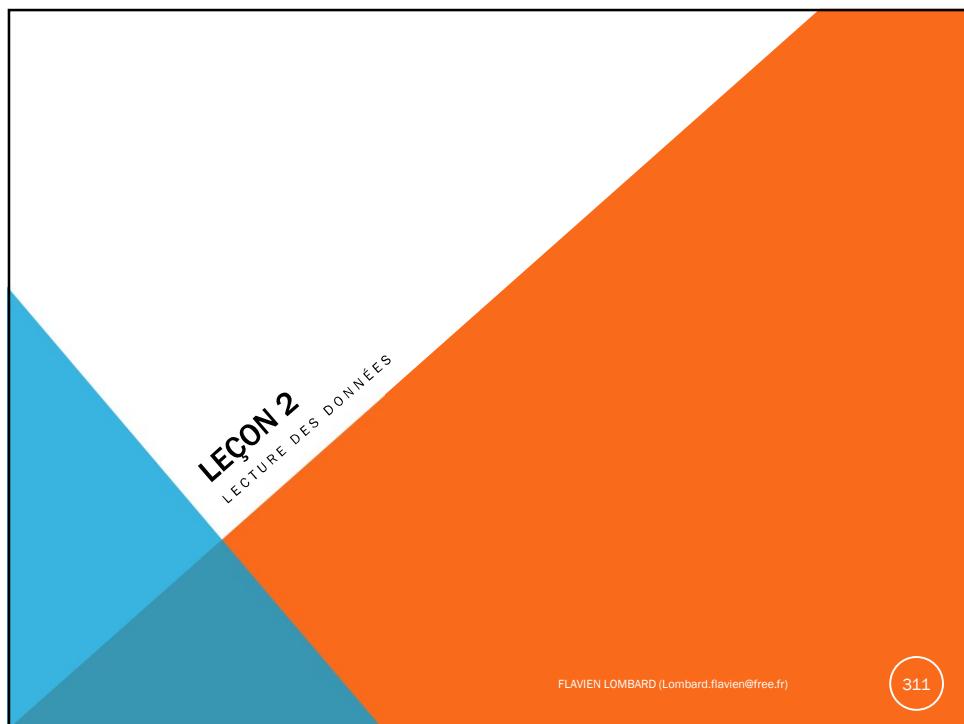
309

ACCÉDER AUX BASES DE DONNÉES

Paramétriser une base de donnée.

Utiliser une chaîne de connexion valide

310



LECTURE PAR FLUX

Accès en lecture seul, lecture en avant seulement

Connection
Command
DataReader

313

EXEMPLE DE LECTURE PAR FLUX

```
public static void GetElements()
{
    OleDbConnection cnx = new OleDbConnection(
        @"Provider=Microsoft.Jet.OLEDB.4.0;" +
        @"Data source=c:\temp\wind.mdb");
    OleDbCommand cmd = cnx.CreateCommand();
    cmd.CommandText = "select * from Customers where Country='France' and [PostalCode] like '75%'";
    cnx.Open();
    OleDbDataReader dr = cmd.ExecuteReader();
    cmd.Dispose();
    while (dr.Read())
    {
        Console.WriteLine("Enregistrement de la société \"{}\"", dr["CompanyName"]);
        for (int i = 0; i < dr.FieldCount; i++)
            Console.WriteLine("{} = {}", dr.GetName(i), dr[i]);
        Console.WriteLine();
    }
    dr.Close();
    cnx.Close();
    cnx.Dispose();
}
```

314

CHARGEMENT EN MÉMOIRE

Utilisation des DataSets
Accès en lecture/écriture

Moins Performant mais plus souple que DataReader (*schéma complet récupéré*)

Indépendant de la BD

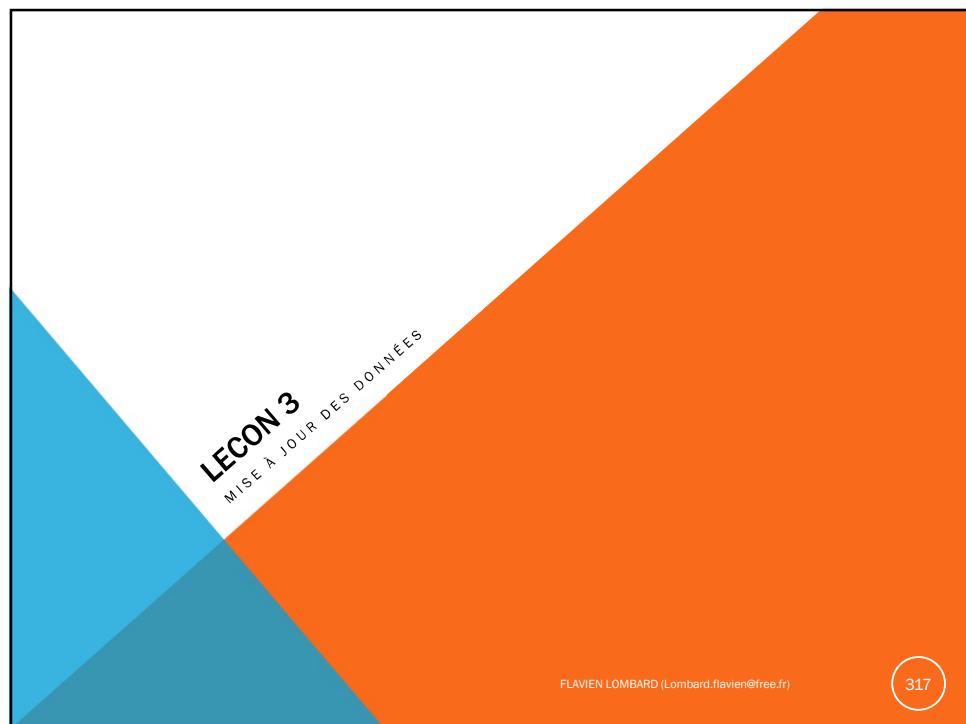
Utilisation de DataAdaptater pour Remplir un DataSet

315

CHARGEMENT EN MÉMOIRE

```
public static void GetElements()
{
    OleDbConnection cnx = new OleDbConnection(
        @"Provider=Microsoft.Jet.OLEDB.4.0;" +
        @"Data source=c:\temp\wind.mdb");
    OleDbCommand cmd = cnx.CreateCommand();
    cmd.CommandText = "select * from Clients where Pays='France' " +
        "and [Code postal] like '75%'";
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    DataSet ds = new DataSet();
    cnx.Open();
    da.Fill(ds, "Clients");
    da.Dispose();
    cmd.Dispose();
    cnx.Dispose();
    for (int n = ds.Tables["Clients"].Rows.Count; n > 0; n--)
    {
        DataRow record = ds.Tables["Clients"].Rows[n - 1];
        Console.WriteLine("Enregistrement de la société \"{}\"", record["Société"]);
        for (int i = 0; i < ds.Tables["Clients"].Columns.Count; i++)
            Console.WriteLine("{} = {}", ds.Tables["Clients"].Columns[i].ColumnName,
                record[i]);
        Console.WriteLine();
    }
    ds.Dispose();
}
```

316



The image displays a template for taking notes. It features a large central area with horizontal ruling lines for writing. This central area is flanked by two vertical columns: an orange column on the left and a blue column on the right. Both the orange and blue areas have triangular cutouts at the bottom corners. In the bottom right corner of the blue area, there is a small circular icon containing the number "318". Above the note-taking area, the word "NOTES" is printed in capital letters.

C#, développer en .NET avec Visual Studio

MISE A JOUR DES DONNÉES

Utilisation de CommandBuilder

Utilisation du schéma du DataSet pour générer la commande

Fonction Update pour mettre à jour

319

```
public static void Update()
{
    OleDbConnection cnx = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data source=c:\temp\nwind.mdb");
    cnx.Open();
    OleDbCommand cmd = cnx.CreateCommand();
    cmd.CommandText = "select * from Customers where Country='France' and [PostalCode] like '75%'";
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    OleDbCommandBuilder cb = new OleDbCommandBuilder(da);
    cb.QuotePrefix = "[";
    cb.QuoteSuffix = "]";
    DataSet ds = new DataSet();
    da.Fill(ds, "Customers");
    for (int n = 0; n < ds.Tables["Customers"].Rows.Count; n++)
    {
        DataRow record = ds.Tables["Customers"].Rows[n];
        if ((string)(record["CustomerID"]) == "PARIS")
        {
            Console.WriteLine("Avant :{0}", record["ContactName"]);
            record["ContactName"] = "Gérard Mensoif";
            da.Update(ds, "Customers");
            Console.WriteLine("Après :{0}", record["ContactName"]);
            Console.WriteLine("Je viens d'exécuter la commande : {0}",
                cb.GetUpdateCommand().CommandText);
            break;
        }
    }
    cb.Dispose();
    ds.Dispose();
    da.Dispose();
    cnx.Close();
    cnx.Dispose();
}
```

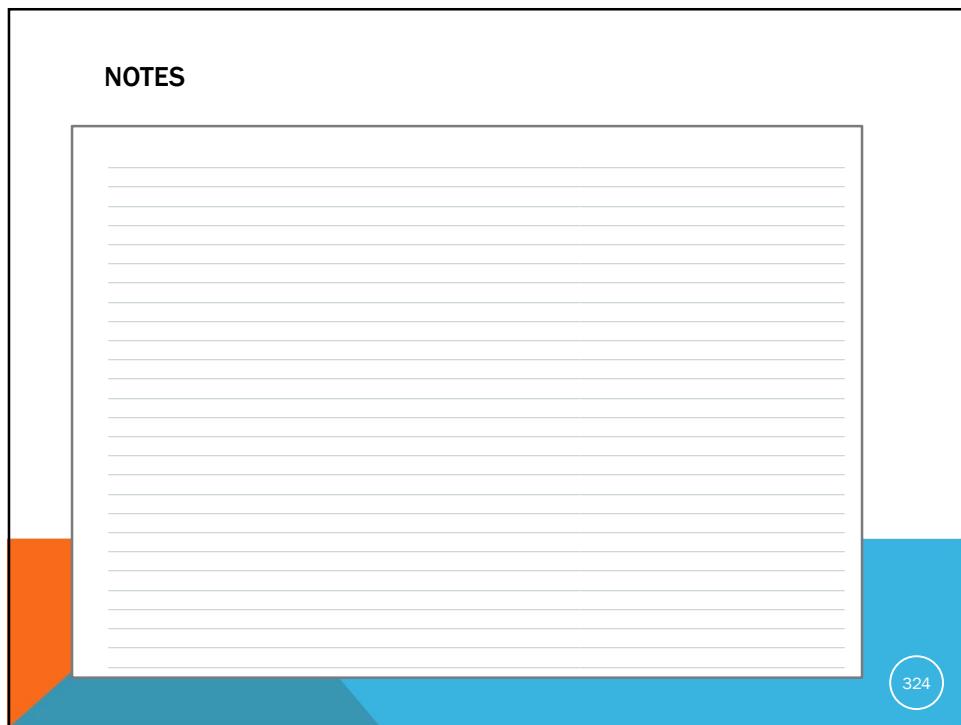
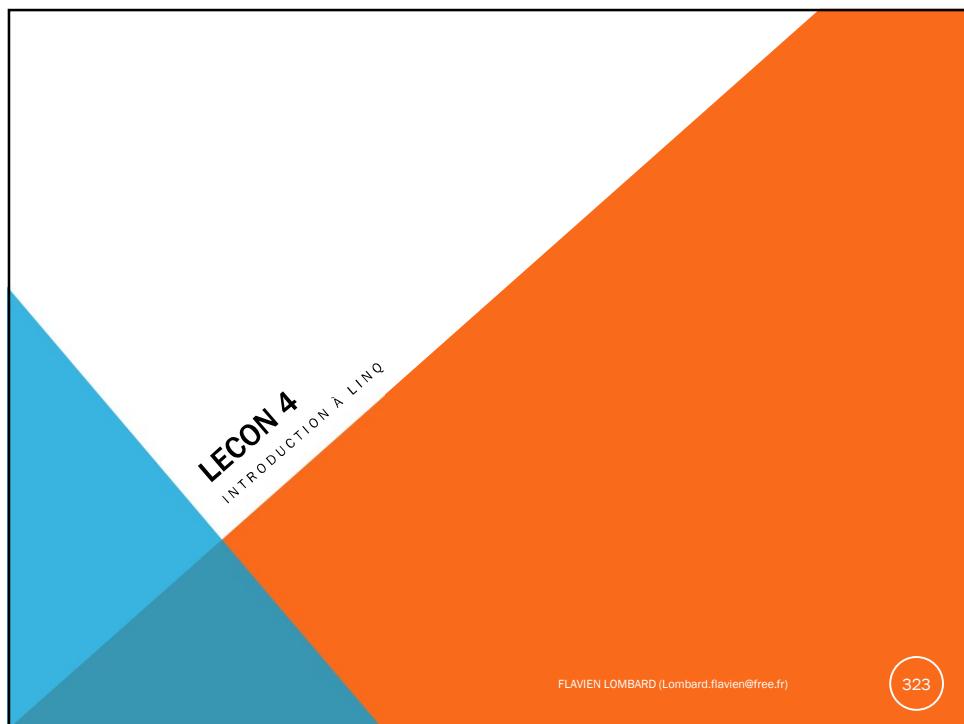
320

C#, développer en .NET avec Visual Studio

```
Dim cnx = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data source=c:\temp\Nwind.mdb")
cnx.Open()
Dim cmd = cnx.CreateCommand()
cmd.CommandText = "select * from Customers where Country='France' and (PostalCode) like '75%'"
Dim da = New OleDbDataAdapter(cmd)
Dim cb = New OleDbCommandBuilder(da)
cb.QuotePrefix = "("
cb.QuoteSuffix = ")"
Dim ds = New DataSet()
da.Fill(ds, "Customers")
For n As Integer = 0 To ds.Tables("Customers").Rows.Count
    Dim record = ds.Tables("Customers").Rows(n)
    If (GetType(record("CustomerID"), String) = "PARIS") Then
        Console.WriteLine("Avant :{0}", record("ContactName"))
        record("ContactName") = "Gérard Mensoif"
        da.Update(ds, "Customers")
        Console.WriteLine("Après :{0}", record("ContactName"))
        Console.WriteLine("Je viens d'exécuter la commande : {0}",
            cb.GetUpdateCommand().CommandText)
    End If
Next
cb.Dispose()
ds.Dispose()
da.Dispose()
cnx.Close()
cnx.Dispose()
```

321

322



LINQ TO ADO

Principe

- LINQ est un moyen d'accéder de manière uniforme aux données
- LINQ permet d'accéder aux données via ADO déconnecté

LINQ to DataSet

- LINQ to DataSet est bien sur indépendant du fournisseur
- LINQ permet d'accéder aux données via ADO connecté

LINQ to SQL

- LINQ to SQL ne fonctionne qu'avec SQL Server

325

LINQ TO DATASET

System.Data.DataSetExtensions

Ensemble de méthodes d'extension permettant d'utiliser les DataSet avec LINQ

326

LINQ TO SQL

permet de générer du SQL à partir de l'arbre d'expression
Permet donc également de manipuler directement la base sans SQL.
Fortement typé
DataTable, DataSet et Datareader sont masqués

327

LINQ TO SQL

DataContext

Classe « masquant » toutes la mécanique de LINQ to SQL

- Gestion des connections
- Gestion de lecture des données
- Gestion d'écritures des données
- Gestion des DataSet
- Contient les entités (représentation objet d'une table de la base)

328

EXERCICES

Ecole :

Création d'un dossier "DataAccessLayer" dans le projet de bibliothèque de classe.

Création d'une base de donnée manuellement pour stocker les élèves

Création d'une classe permettant de faire un CRUD (Create, Retrieve, Update, Delete) sur les élèves en base de donnée.



DÉMONSTRATION DE ENTITYFRAMEWORK CODEFIRST

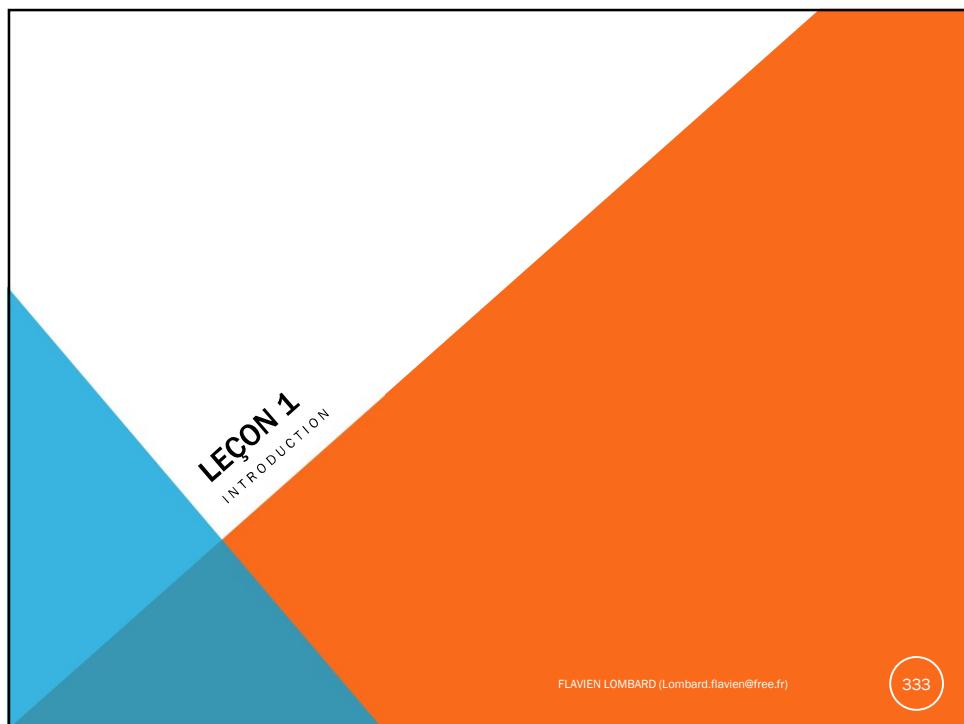




NOTES

Handwriting practice lines

332



The notes page has a header "NOTES" and a large rectangular area for writing with horizontal lines. The page is framed by a blue border, with orange and blue corner accents. A small circular icon with the number "334" is at the bottom right.

NOTES

334

DÉFINITIONS

Plateforme de développement .NET pour les applications fenêtrées.

Existe depuis .NET 1.0

Anclinement dialogue MFC

Windows Form s'articule autour des formulaires (Forms)

Le formulaire n'est pas l'application !!

System.Windows.Form

335

MESSAGEBOX

Affichage de la boîte de dialogue standard Windows



336

MESSAGEBOX

Utilisation de la méthode Show

Surcharge de la méthode pour affiner

Boutons, icônes, texte...

```
MessageBox.Show("Hello world !", "Hello");

MessageBox.Show("Hello world !", "Hello",
    MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Asterisk,
    MessageBoxDefaultButton.Button1);
```

337

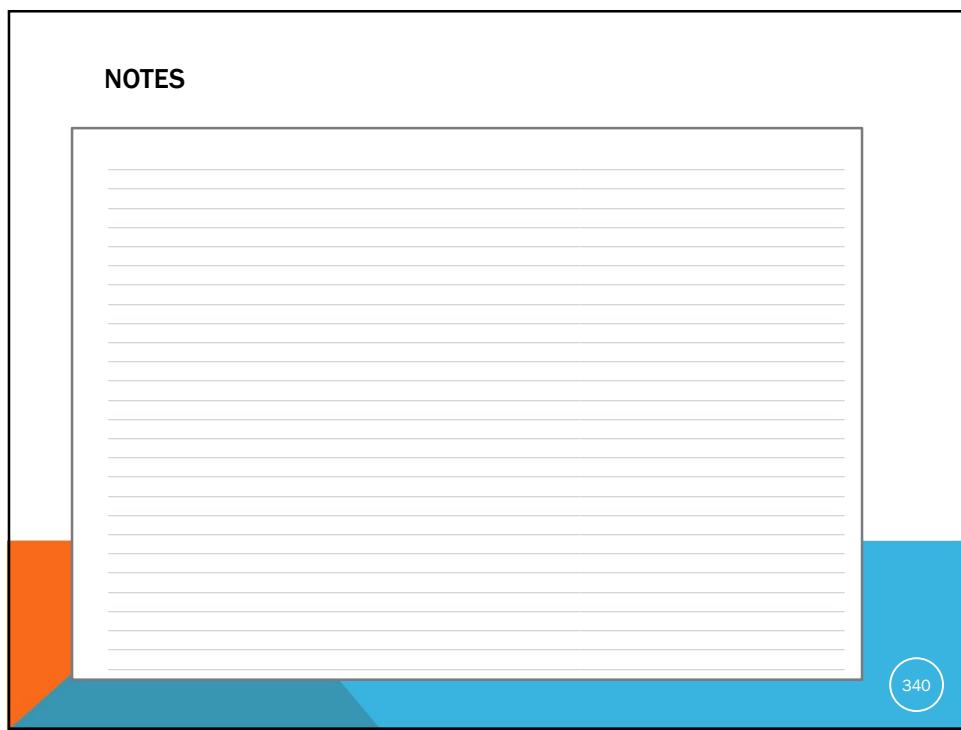
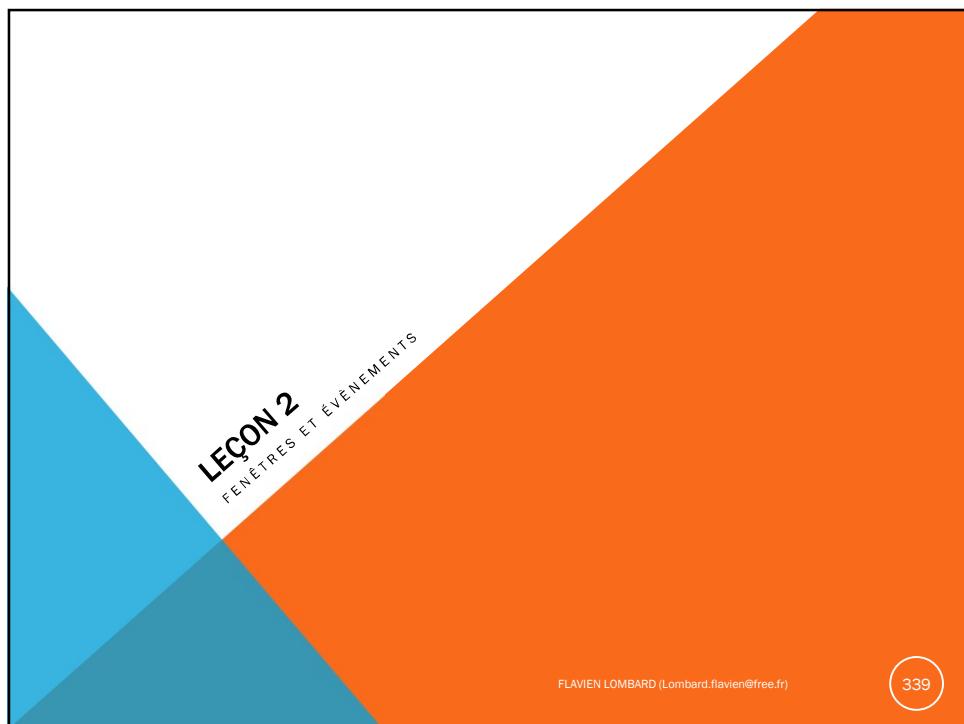
PROPRIÉTÉS D'AFFICHAGE DE BASE

Le Formulaire est la base d'un Windows Form

System.Windows.Form est la classe gérant les Formulaires.

Show	Affiche le formulaire
ShowDialog et DialogResult	Affichage modal
Hide	Masque le formulaire
Close	Ferme le formulaire

338



FENÊTRES ET ÉVÉNEMENTS

C'est l'utilisateur qui exécute le programme
L'utilisateur envoie des événements au programme.
Les événements et délégués sont au cœur des formulaires.
Un délégué lié à un événement est un Handler

341

FENÊTRES ET ÉVÉNEMENTS

Les événements se gèrent comme des délégués
Sender est l'objet qui a signalé l'événement
e contient les infos supplémentaires liées à l'événement

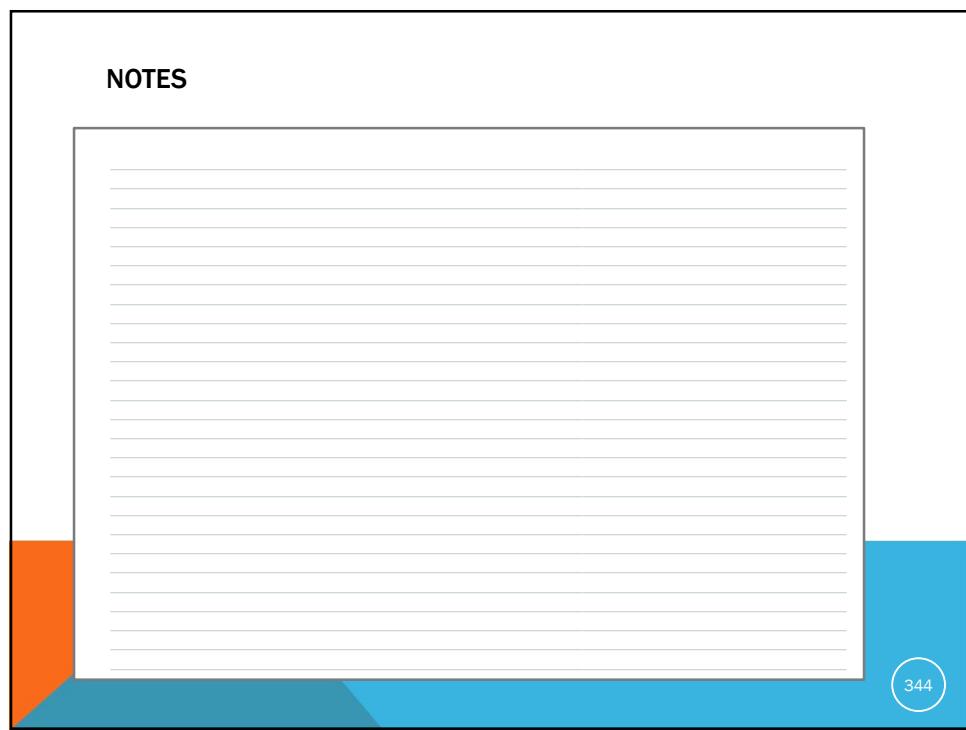
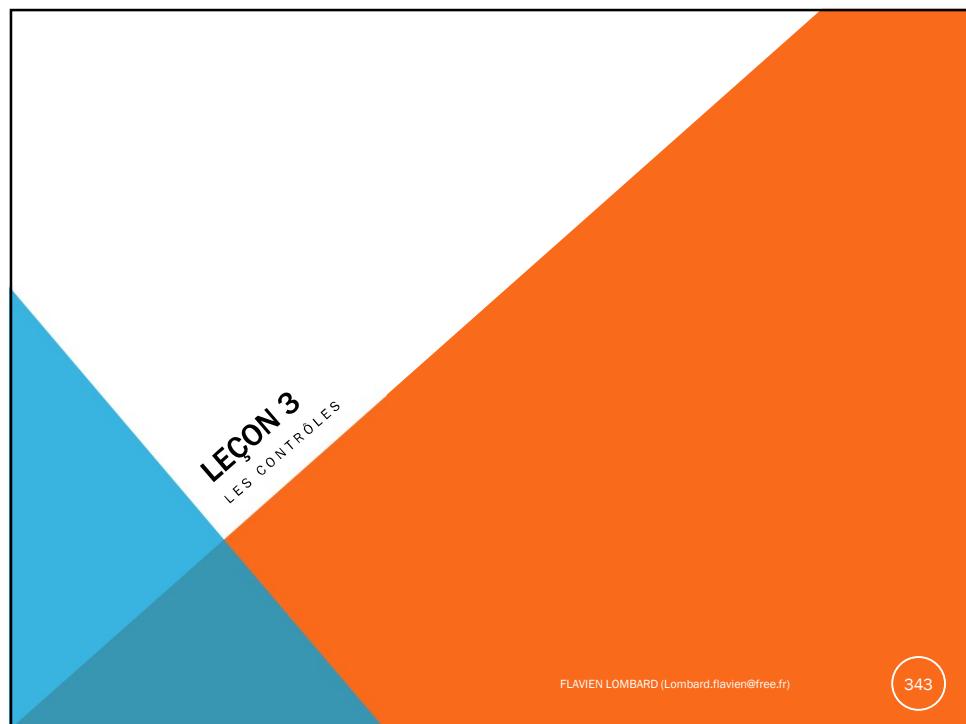
```
public Form1()
{
    InitializeComponent();
    this.Load += new System.EventHandler(this.Form1_Load);
}

private void Form1_Load(object sender, EventArgs e)
{

}

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
End Sub
```

342



LES CONTRÔLES

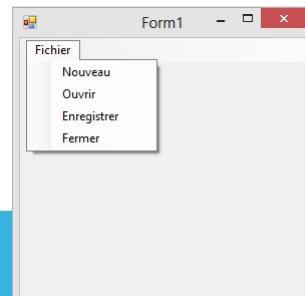
Hérite de `System.Windows.Form.Control`
Comportements graphique
Interactions avec le système et l'utilisateur grâce aux évènements
Les contrôles appartiennent à un formulaire
`Designer.cs`

345

MENUS

Organisation des commandes accessibles à l'utilisateur.

- `MenuStrip`
- `ToolStripMenuItem`



346

MANIPULER DES CONTRÔLES

Ajout, Suppression grâce au Designer de VS

Ajout, suppression par le code

347

MANIPULER DES CONTRÔLES

Interaction des contrôles entre eux

Les contrôles sont des classes.

Pour interagir, il suffit de modifier les propriétés de la classe

348

MANIPULER DES CONTRÔLES

Le Focus

Le focus est le control qui prend les évènements utilisateur à un instant t.

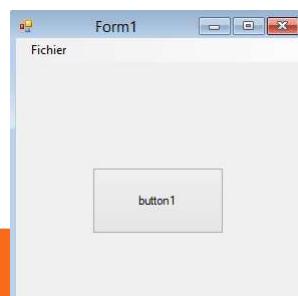
```
public bool Control.Focus();
public virtual bool Control.Focused { get; }
```

349

CONTRÔLE BUTTON

Effectue une action sur un click de souris.

- String Button.Text : texte affiché sur le bouton
- EventHandler Button.Click : Evènement sur le clique du bouton
- Utilisation du caractère & pour les touches de raccourcis
monBouton.Text = "&Fermer"



350

CONTRÔLE TEXTBOX

Permet de récupérer des informations texte de l'utilisateur

- String TextBox.text : Affecte ou récupère le contenu du textbox
- bool TextBox.Multiline : Définit si la textbox contient plusieurs lignes
- KeyPressEventHandler TextBox.KeyPress : Evénement signalé lorsqu'une touche est saisie
- EventHandler TextBox.Click : Evénement au click du contrôle



351

CONTRÔLE LABEL

Permet l'affichage d'information textuel.

- Un label ne peut pas recevoir le focus
- string Label.Text : Texte du label

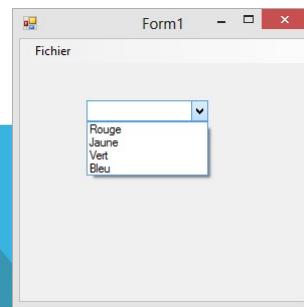


352

CONTRÔLE COMBOBOX

Permet de sélectionner une valeur dans une liste. Permet également de saisir une valeur qui n'est pas dans la liste

- **Collection Combobox.Items** : Objets contenus dans le contrôle
- **Object Combobox.SelectedItem** : Objet sélectionné
- **String Combobox.selectedText** : Texte contenu dans le control

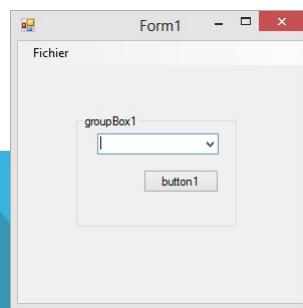


353

CONTRÔLES CONTENEURS

Contrôles contenant d'autres contrôles

- Panel
- FlowLayoutPanel
- TableLayoutPanel
- GroupBox



354

DIALOGUES PRÉDÉFINIS

Appelez les dialogues Système

- OpenFileDialog : Ouverture d'un fichier
- SaveFileDialog : Enregistrement d'un fichier
- ColorDialog : Choix d'une couleur
- PageSetupDialog : Setup d'impression
- FontDialog : Choix d'une police de caractère

355

DÉMONSTRATION



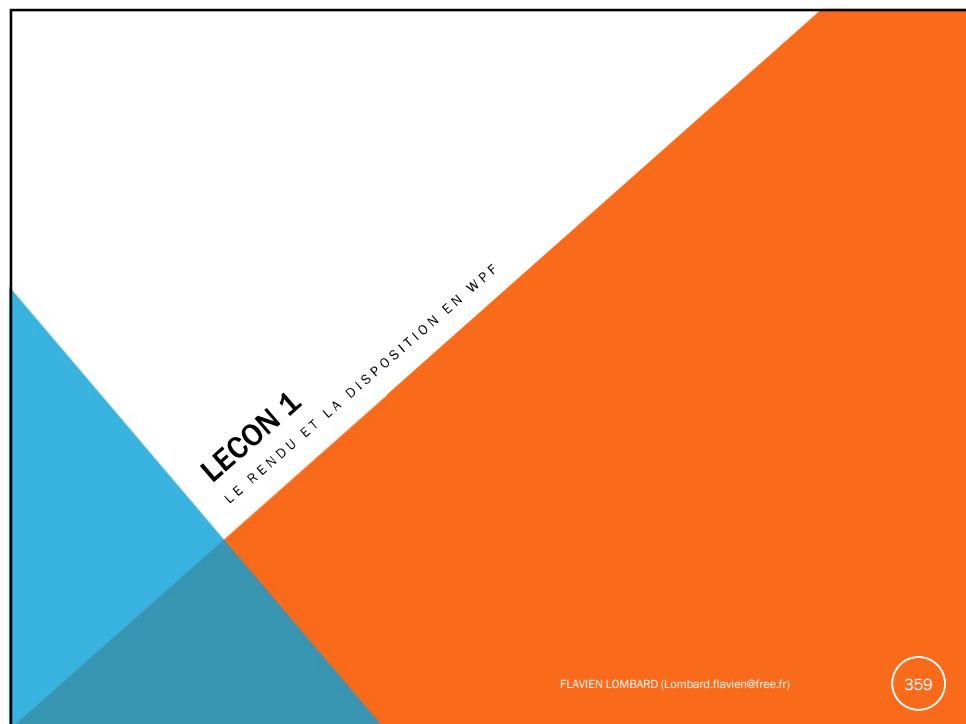
356



NOTES

A blank sheet of lined paper with horizontal ruling lines, intended for notes. It is positioned in the center of a larger frame.

358



The notes page has a white background with a central rectangular area for writing, featuring horizontal ruling lines. The word "NOTES" is printed in bold capital letters at the top left. The page is framed by a thick black border. At the bottom, there is a decorative footer consisting of a blue horizontal bar with orange triangular corners.

360

INTRODUCTION À WPF

WPF est une évolution de WinForm

Utilisation du XAML

Graphiquement plus riche

Plusieurs types d'applications:

- Boîtes de dialogue
- Navigateur
- Hébergé (XBAP)
- Silverlight

361

MODÈLE DE MISE EN PAGE WPF

1 Passe de mesure

Hello World!

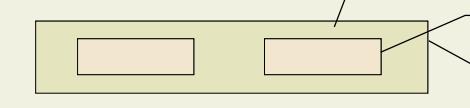
• cadre de délimitation rectangulaire supposé

• Recupéré par un appel à `GetLayout` sur le `FrameworkElement`

Evaluation de chaque membre de la collection des enfants pour déterminer la "DesiredSize"

2 Passe d'arrangement

Element **Window** ou **Page**



Objets fils

Classes de mise en page

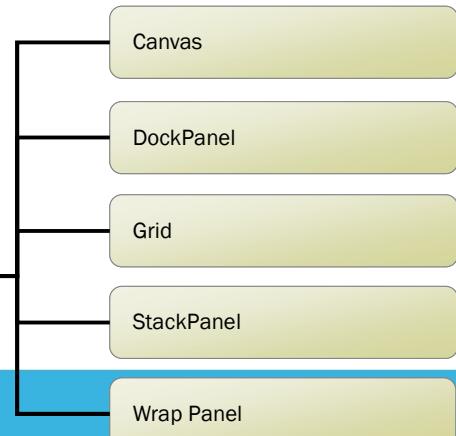
Determination de la taille finale de chaque élément enfant et placement dans l'élément de mise en page.

362

LES CLASSES DE MISE EN PAGE

Panel

Background
Children
ZIndex

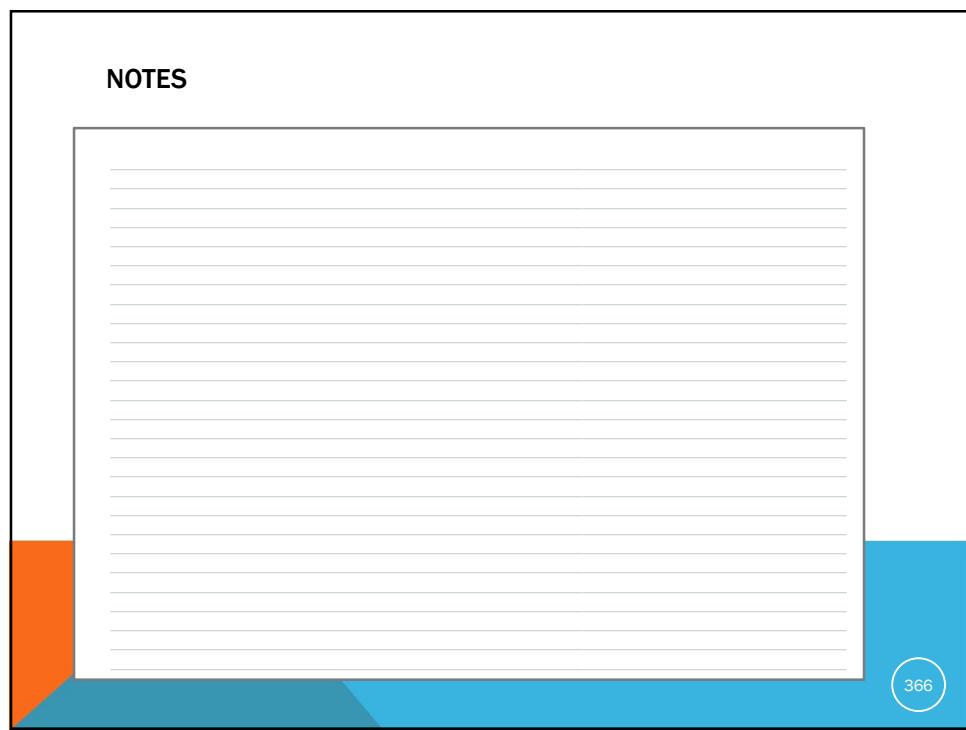
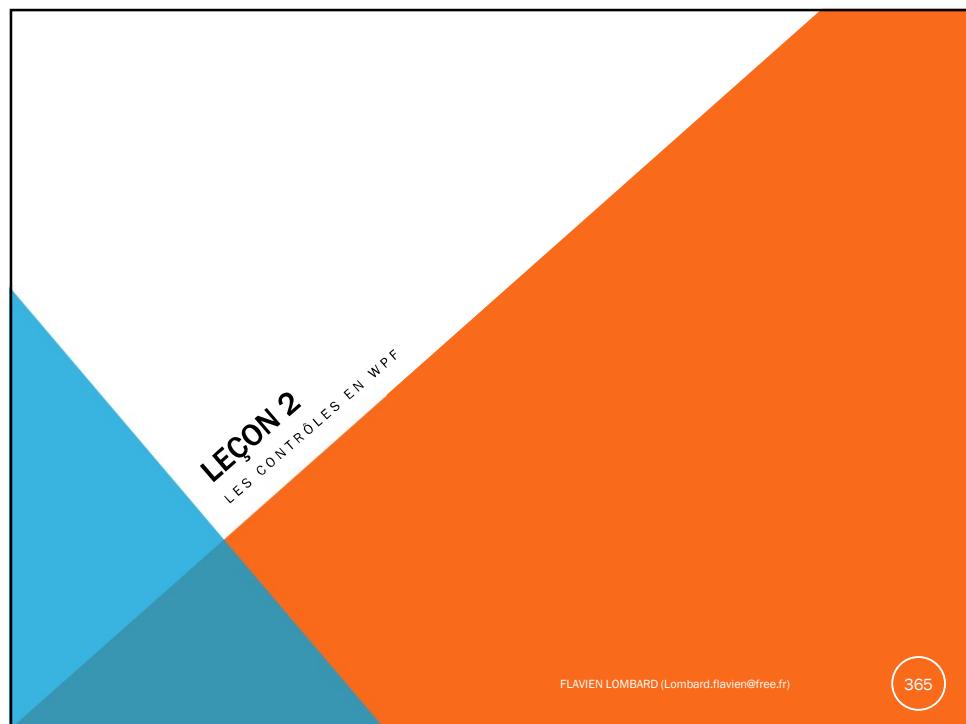


363

DÉMONSTRATION

- Classe Canvas
- Classe StackPanel
- Classe WrapPanel
- Classe DockPanel
- Classe Grid

364



LES CONTRÔLES DE CONTENEUR SIMPLE

- Contient un seul élément
- Contient une propriété "Content"

Exemples

C'est le contenu texte d'un **Button**

Controles communs:

- Button
- CheckBox
- GroupItem
- Label
- RadioButton
- RepeatButton
- ToggleButton
- ToolTip

367

LES CONTRÔLES CONTENEUR À EN-TÊTE

- contrôles conteneur spécialisés
- Contient une propriété "Content"
- Contient une propriété "Header"

TabItem header

Les contrôles conteneur à en-tête :

- Expander
- GroupBox
- TabItem

Exemple

GroupBox

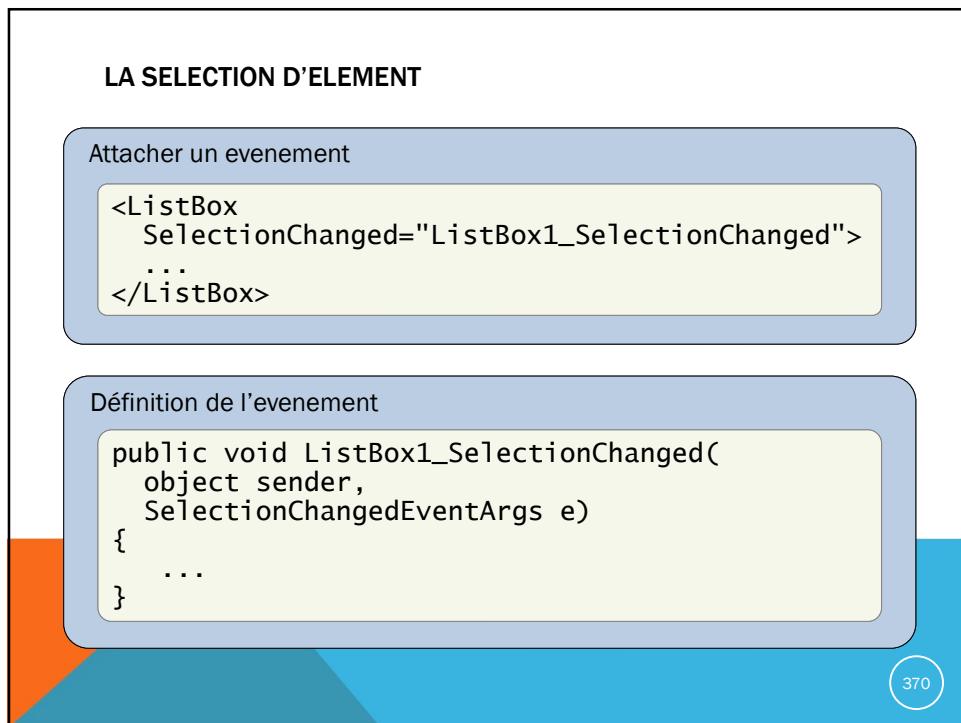
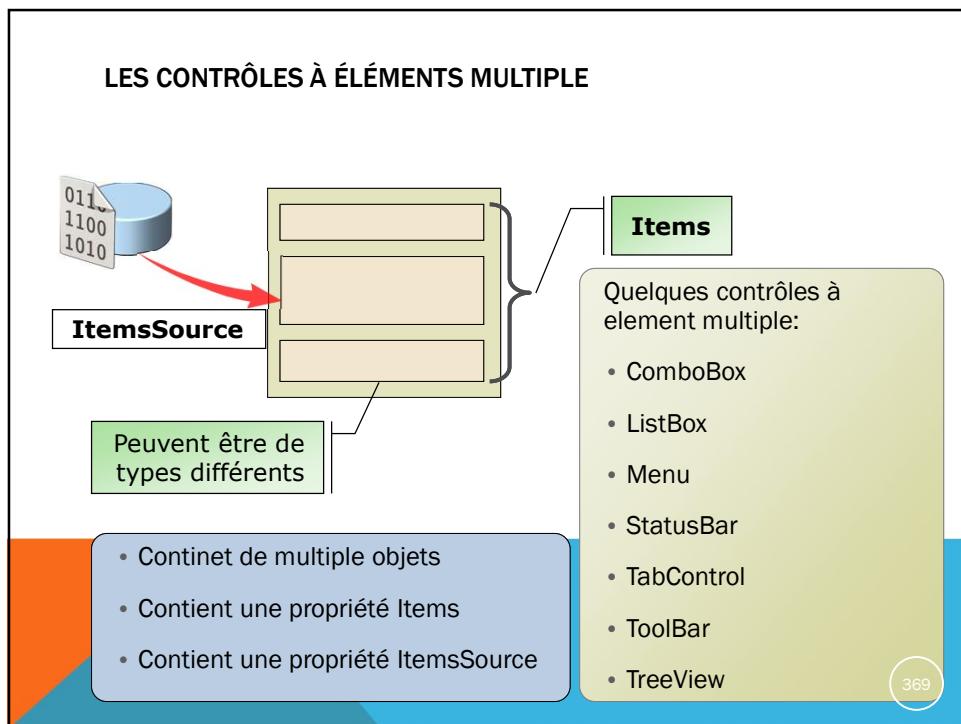
Page 1 Page 2

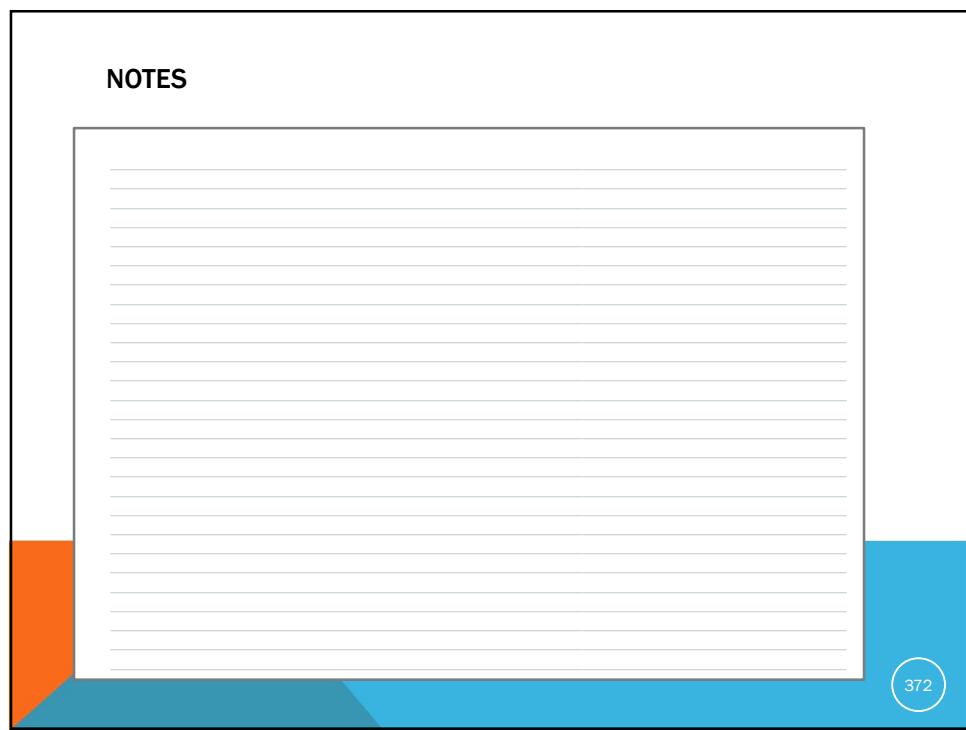
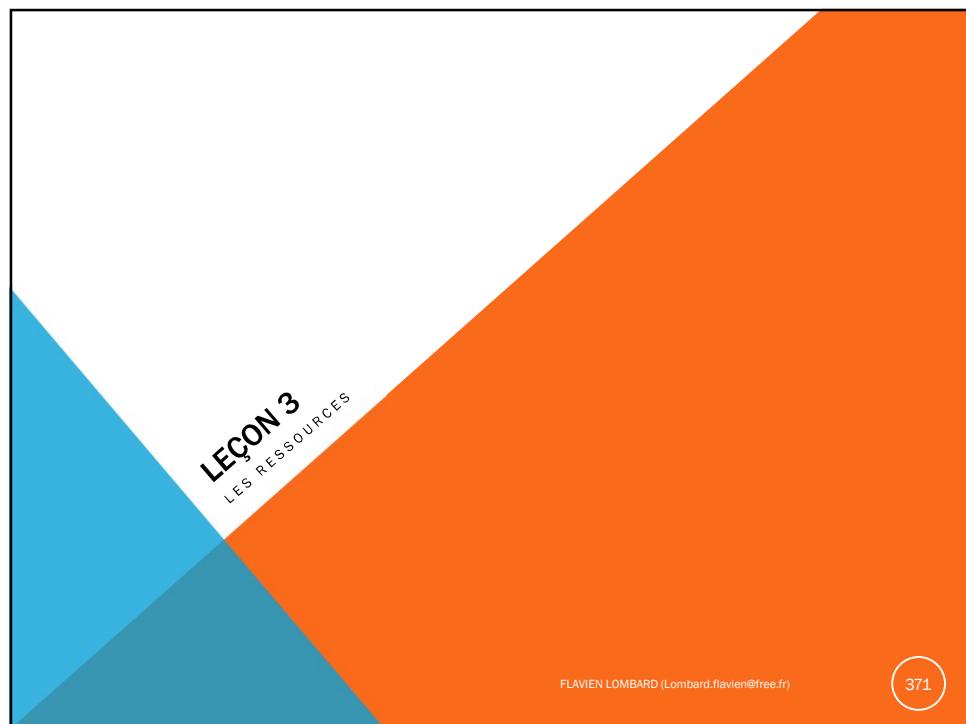
GroupBox Header

My Expander
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

Expander

368

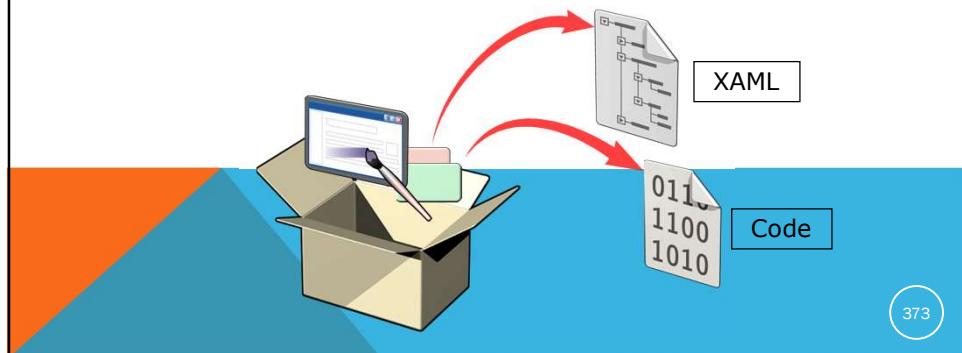




LES RESSOURCES

Les ressources permettent la réutilisation simple d'objets et de valeurs.

Par exemple: brushes, styles, et modèle de contrôle



373

DÉFINIR UNE RESSOURCE

Les ressources peuvent être définies à différents niveaux :

- Au niveau de l'application
- Au niveau de la fenêtre ou de la page
- Au niveau de l'élément

XAML

```
<Window.Resources>
    <SolidColorBrush x:Key="blueBrush" Color="Blue"/>
    <SolidColorBrush x:Key="whiteBrush" Color="White"/>
    <sys:Double x:Key="myValue">100</sys:Double>
</Window.Resources>
```

374

RÉFÉRENCEMENT DES RESSOURCES EN XAML

Pour référencer une ressource statique

```
PropertyName="{StaticResource ResourceKey}"  
  
<Button Background="{StaticResource blueBrush}"  
        Foreground="{StaticResource whiteBrush}">  
    Text  
</Button>
```

Pour référencer une ressource dynamique

```
PropertyName="{DynamicResource ResourceKey}"  
  
<Button Background="{DynamicResource blueBrush}"  
        Foreground="{DynamicResource whiteBrush}">  
    Text  
</Button>
```

375

RÉFÉRENCEMENT DES RESSOURCES PAR LE CODE

Méthode **FindResource**

```
SolidColorBrush brush = (SolidColorBrush)  
this.FindResource("whiteBrush");
```

Méthode **SetResourceReference**

```
this.myButton.SetResourceReference(  
    Button.BackgroundProperty, "whiteBrush");
```

Ou
TryFindResource

Propriété **Resources**

```
SolidColorBrush brush = (SolidColorBrush)  
this.Resources["whiteBrush"];
```

376

DÉMONSTRATION



377

378



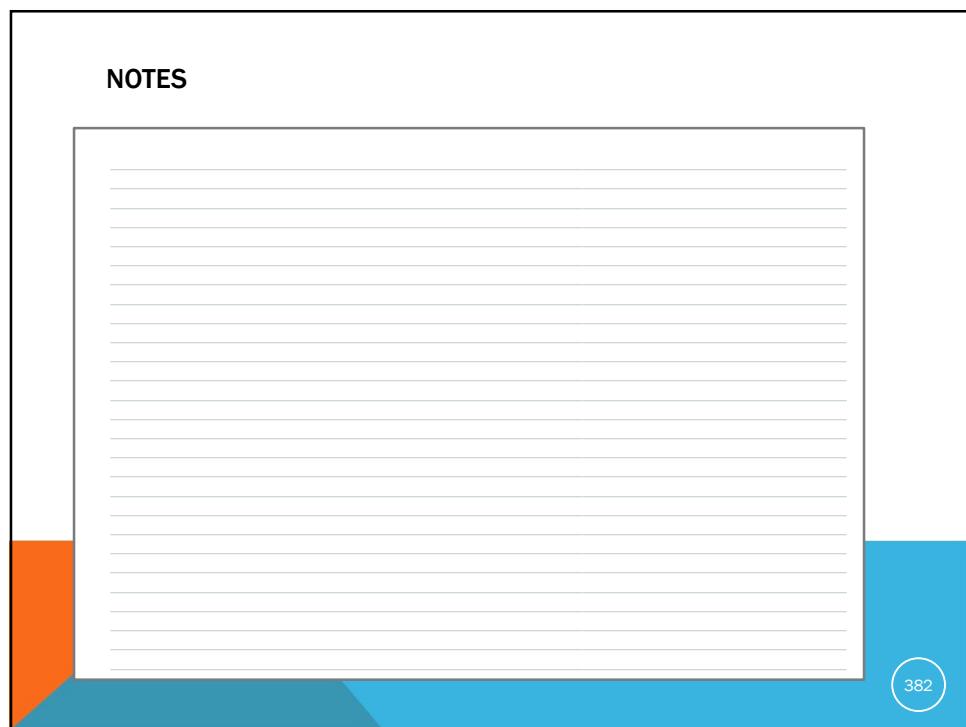
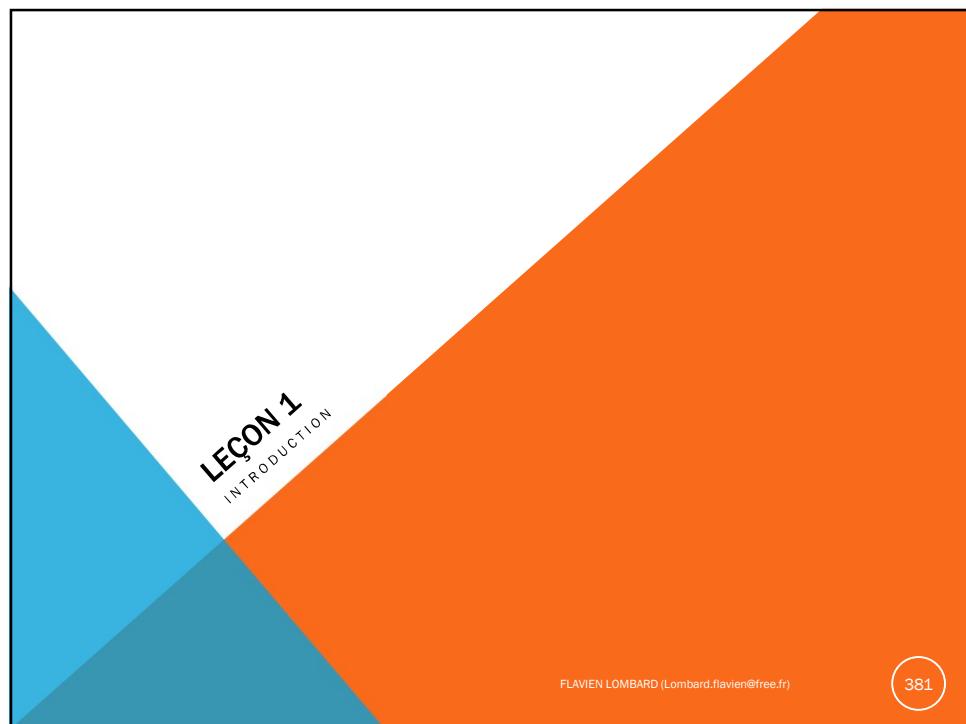
NOTES

A blank sheet of lined paper with horizontal ruling lines, intended for notes. It is positioned in the center of a larger rectangular frame.

A solid orange rectangular block located at the bottom-left corner of the main frame.

A solid blue rectangular block located at the bottom-right corner of the main frame.

380



C#, développer en .NET avec Visual Studio

INTRODUCTION

Framework multiplateforme à hautes performances et open source

- Applications et Services web, applications [IoT](#) et backends mobiles.
- Windows, MacOs et Linux.
- Environnement cloud ou local
- Exécution sur .NET Core ou .NET Framework.

383

AVANTAGES

- Un scénario unifié pour créer une interface utilisateur web et des API web.
- Conçu pour la testabilité.
- Razor Pages permet de coder des scénarios orientés page de façon plus simple et plus productive.
- Capacité à développer et à exécuter sur Windows, MacOs et Linux.
- Open source et centré sur la communauté.
- L'intégration de Framework modernes, côté client et des workflows de développement.
- Un environnement prêt pour le cloud et basé sur des fichiers de configuration système.
- Une Injection de dépendances intégrée.
- Un pipeline de requête HTTP léger, haute performance et modulaire.
- La capacité à héberger sur IIS, Nginx, Apache, Docker, ou d'un auto-hébergement dans votre propre processus.
- La gestion de version des applications côte à côte lorsque la cible est [.NET Core](#).
- des outils qui simplifient le développement web moderne.

384

TOUR D'HORIZON

WebSocket / SignalR
WebAssembly (Wasm)

385

QUELQUES NOUVEAUTÉS DE ASP.NET CORE

SignalR

SignalR a été réécrit pour ASP.NET Core 2.1. ASP.NET Core SignalR inclut un certain nombre d'améliorations :

- Un modèle simplifié de montée en puissance parallèle.
- Un nouveau client JavaScript sans dépendance de jQuery.
- Un nouveau protocole binaire compact basé sur MessagePack.
- Prise en charge des protocoles personnalisés.
- Un nouveau modèle réponse de streaming.
- Prise en charge des clients basés sur des WebSocket nus.

Bibliothèques de classes Razor

Le temps de démarrage de l'application est nettement plus rapide.
Les mises à jour rapides des pages et vues Razor au moment de l'exécution sont toujours disponibles dans le cadre d'un flux de travail de développement itératif.

HTTPS

Https est actif par défaut

RGPD

ASP.NET Core fournit des API et des modèles qui aident à satisfaire à certaines des exigences du RGPD

386

NOUVEAUTÉS DE ASP/NET CORE

Tests d'intégration

- Un nouveau package est introduit qui simplifie la création et l'exécution de tests. Le package [Microsoft.AspNetCore.Mvc.Testing](#)

[ApiController], ActionResult<T>

NET Core ajoute de nouvelles conventions de programmation qui facilitent la génération d'API web propres et descriptives

IHttpClientFactory

ASP.NET Core 2.1 inclut un nouveau service IHttpClientFactory qui facilite la configuration et l'utilisation d'instances de HttpClient dans les applications

Configuration du transport Kestrel

Dans ASP.NET Core 2.1, le transport par défaut de Kestrel n'est plus basé sur Libuv, mais sur des sockets managés

Générateur d'hôte générique

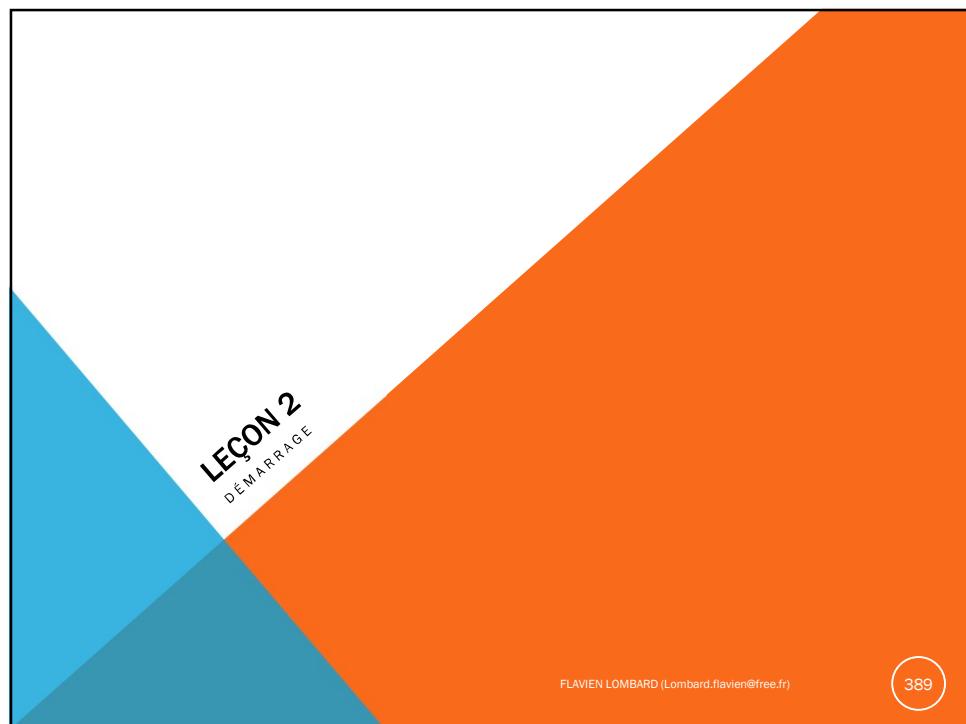
Le générateur d'hôte générique (HostBuilder) a été introduit. Ce générateur peut être utilisé pour les applications qui ne traitent pas les requêtes HTTP (messagerie, les tâches en arrière-plan, etc.).

Modèles SPA mis à jour

Les modèles d'applications monopages pour Angular, React et React avec Redux sont mis à jour pour utiliser les systèmes de génération et les structures de projet standard pour chaque framework.

387

388



CLASSE STARTUP

Utilisation de la classe **Startup** pour la configuration et l'exécution de l'application

- **ConfigureServices** permettant de configurer les services de l'application.
- **Configure** pour créer le pipeline de traitement de demande de l'application.

```
public class Startup
{
    // Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        ...
    }

    // Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app)
    {
        ...
    }
}
```

391

ENREGISTREMENT DE LA CLASSE STARTUP

La classe **Startup** doit être enregistré dès le démarrage de l'application

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

392

UTILISATION COURANTE DE STARTUP

Une utilisation courante de l'injection de dépendances dans la classe Startup consiste à injecter :

- IHostingEnvironment pour configurer les services par environnement.
- IConfiguration pour lire la configuration.
- ILoggerFactory pour créer un enregistreur d'événements dans Startup.ConfigureServices.

```
public class Startup {
    private readonly IHostingEnvironment _env;
    private readonly IConfiguration _config;
    private readonly ILoggerFactory _loggerFactory;

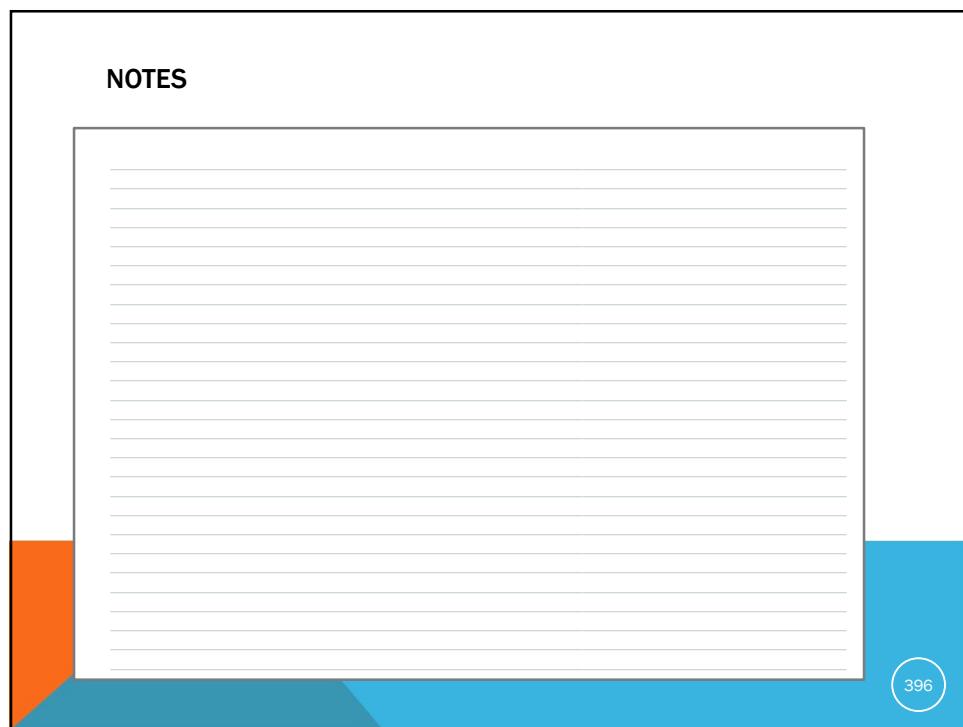
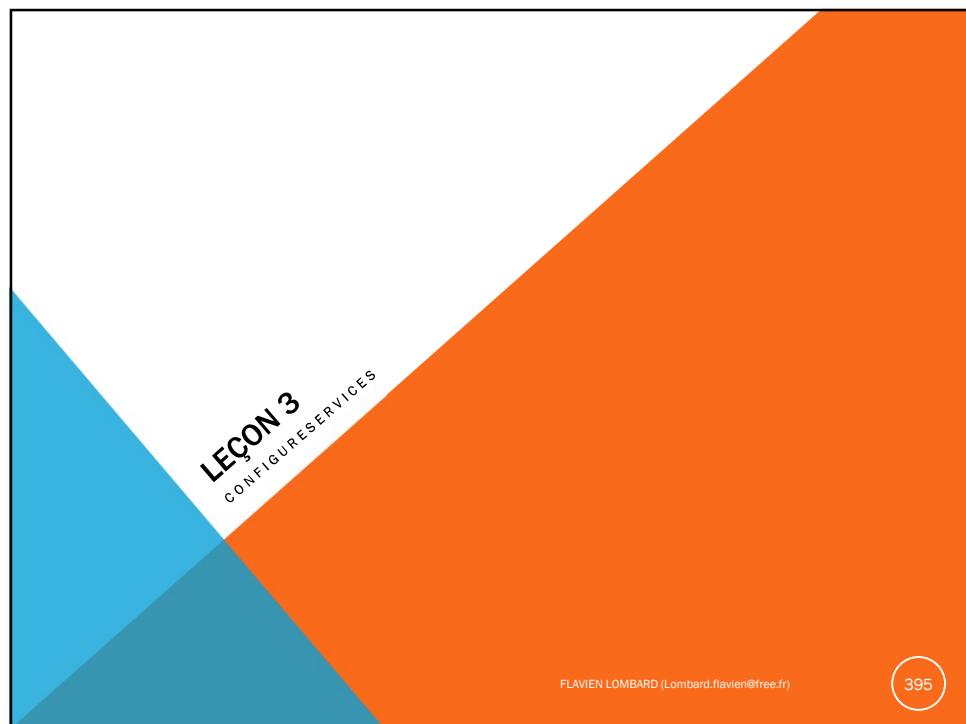
    public Startup(IHostingEnvironment env, IConfiguration config, ILoggerFactory loggerFactory) {
        _env = env;
        _config = config;
        _loggerFactory = loggerFactory;
    }

    public void ConfigureServices(IServiceCollection services) {
        var logger = _loggerFactory.CreateLogger<Startup>();

        if (_env.IsDevelopment())
            logger.LogInformation("Development environment");
        else
            logger.LogInformation($"Environment: {_env.EnvironmentName}");
    }
}
```

393

394



MÉTHODE CONFIGURESERVICES

La méthode ConfigureServices est :

- Facultatif
- Appelée par l'hôte web avant la méthode Configure pour configurer les services de l'application.
- L'emplacement où les options de configuration sont définies.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity< ApplicationUser, IdentityRole >()
        .AddEntityFrameworkStores< ApplicationDbContext >()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient< IEmailSender, AuthMessageSender >();
    services.AddTransient< ISmsSender, AuthMessageSender >();
}
```

Par default, on appelle toutes les méthodes d'ajout de services Add[service] puis leurs configurations Configure[service]

397

MÉTHODE CONFIGURE

La méthode Configure est utilisée pour spécifier la façon dont l'application répond aux requêtes HTTP

Composants intergiciels
(Middleware)

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Error");
    }

    app.UseStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller}/{action=Index}/{id?}");
    });
}
```

398

MÉTHODES PRATIQUES

Les méthodes pratiques `ConfigureServices` et `Configure` peuvent être utilisées au lieu de spécifier une classe `Startup`

```
public class Program
{
    public static IHostingEnvironment HostingEnvironment { get; set; }
    public static IConfiguration Configuration { get; set; }

    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                HostingEnvironment = hostingContext.HostingEnvironment;
                Configuration = config.Build();
            })
            .ConfigureServices(services =>
            {
                services.AddMvc();
            })
            .Configure(app =>
            {
                var loggerFactory = app.ApplicationServices
                    .GetRequiredService<ILoggerFactory>();
                var logger = loggerFactory.CreateLogger<Program>();
            });
}
```

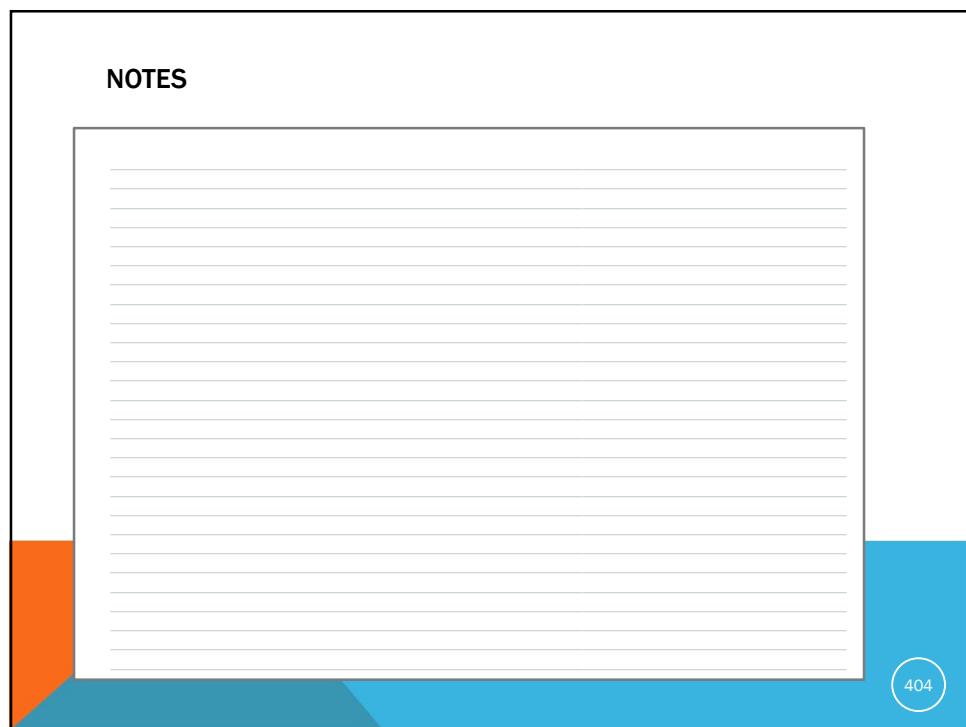
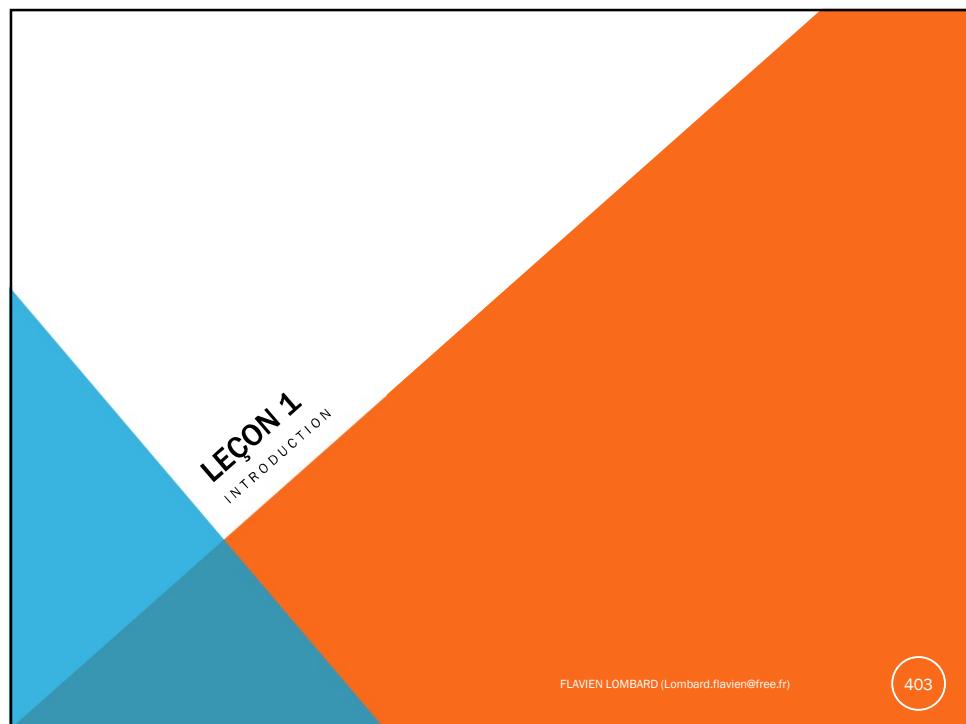
399



NOTES

A rectangular area with horizontal ruling lines, intended for handwritten notes. It is positioned in the center of a larger blue and orange background, which is part of the slide's design.

402



INTRODUCTION

ASP.NET Core prend en charge le modèle de conception de logiciel avec injection de dépendances (DI).
Une technique permettant d'obtenir une inversion de contrôle (IoC) entre les classes et leurs dépendances.

- Une *dépendance* est un objet qui nécessite un autre objet

```
public class Calculator
{
    public Produit Produit { get; set; }

    double calculate()
    {
        return Produit.Prix * Produit.Qte;
    }
}
```

La dépendance est injectée par le constructeur

```
public class Calculator
{
    public IProduit Produit { get; set; }

    public Calculator(IProduit prod)
    {
        Produit = prod;
    }

    double calculate()
    {
        return Produit.Prix * Produit.Qte;
    }
}
```

405

IOC (INVERSION OF CONTROL)

ASP.NET Core fournit un conteneur de service intégré, `IServiceProvider`. Les services sont inscrits dans la méthode `Startup.ConfigureServices` de l'application.

```
public class Startup
{
    public Startup(IConfiguration configuration)...
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IProduit, Produit>();
    }
}
```

La dépendance, peut avoir elle-même des dépendances.

ASP.NET core va se charger
d'injecter les dépendances dans le
constructeur

```
public class Produit : IProduit
{
    private readonly ILogger<Produit> _logger;

    public Produit(ILogger<Produit> logger)
    {
        _logger = logger;
    }

    public string Libelle { get; set; }
    public int Qte { get; set; }
    public double Prix { get; set; }
}
```

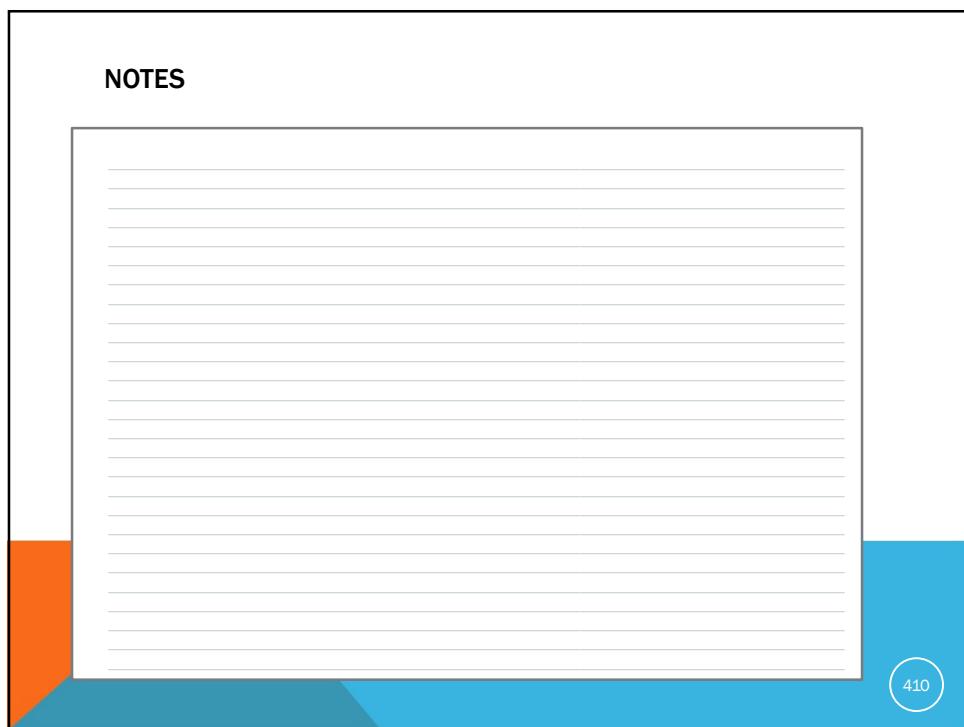
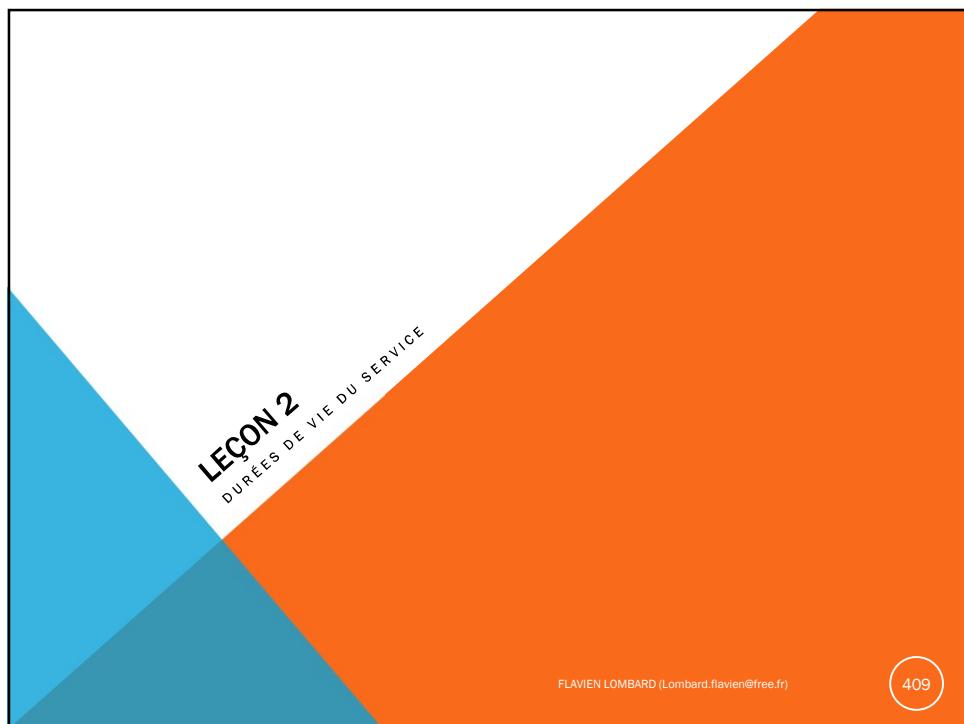
406

SERVICES FOURNIS PAR LE FRAMEWORK

Type de service	Durée de vie
Microsoft.AspNetCore.Hosting.Builder.IApplicationBuilderFactory	Transient
Microsoft.AspNetCore.Hosting.IApplicationLifetime	Singleton
Microsoft.AspNetCore.Hosting.IHostingEnvironment	Singleton
Microsoft.AspNetCore.Hosting.IStartup	Singleton
Microsoft.AspNetCore.Hosting.IStartupFilter	Transient
Microsoft.AspNetCore.Hosting.Server.IServer	Singleton
Microsoft.AspNetCore.Http.IHttpContextFactory	Transient
Microsoft.Extensions.Logging.ILogger<T>	Singleton
Microsoft.Extensions.Logging.ILoggerFactory	Singleton
Microsoft.Extensions.ObjectPool.ObjectPoolProvider	Singleton
Microsoft.Extensions.Options.IConfigureOptions<T>	Transient
Microsoft.Extensions.Options.IOptions<T>	Singleton
System.Diagnostics.DiagnosticSource	Singleton
System.Diagnostics.DiagnosticListener	Singleton

407

408



LES DIFFÉRENTS TYPES DE SERVICE

- **Transient**

Des services à durée de vie temporaire (Transient) sont créés chaque fois qu'ils sont demandés. Cette durée de vie convient parfaitement aux services légers et sans état.

- **Scoped**

Les services à durée de vie délimitée (Scoped) sont créés une seule fois par requête.

- **Singleton**

Les services avec une durée de vie singleton sont créés la première fois qu'ils sont demandés.
Chaque requête ultérieure utilise la même instance.

411

INSCRIPTIONS

L'inscription des services se fait dans la classe **ConfigureServices**

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>{...});

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    services.AddTransient<IProduit, Produit>();
    services.AddScoped<IProduit, Produit>(); ou
    services.AddSingleton<IProduit, Produit>(); ou
}
```

412

APPELER DES SERVICES À PARTIR DE MAIN

Créez un `IServiceScope` avec `IServiceScopeFactory.CreateScope` pour résoudre un service `Scoped` dans le scope de l'application. Cette approche est pratique pour accéder à un service `Scoped` au démarrage pour exécuter des tâches d'initialisation

```
public static void Main(string[] args)
{
    var host = CreateWebHostBuilder(args).Build();

    using (var serviceScope = host.Services.CreateScope())
    {
        var services = serviceScope.ServiceProvider;

        try
        {
            var serviceContext = services.GetRequiredService<MyScopedService>();
            // Use the context here
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred.");
        }
    }

    host.Run();
}
```

413

CONCEPTION DE SERVICES

Les bonnes pratiques :

- Concevoir des services afin d'utiliser l'injection de dépendances pour obtenir leurs dépendances.
- Éviter les appels de méthode statiques avec état (une pratique appelée `static cling`).
- Éviter une instanciation directe de classes dépendantes au sein de services. L'instanciation directe associe le code à une implémentation particulière.

Respecter les principes SOLID

- S – Single Responsibility Principle
- O – Open/Closed Principle
- L – Liskov Substitution Principle
- I – Interface Segregation Principle
- D – Dependency Inversion Principle

414

SUPPRESSION DES SERVICES

Le conteneur appelle `Dispose` pour les types `IDisposable` qu'il crée. Si une instance est ajoutée au conteneur par le code utilisateur, elle n'est pas supprimée automatiquement.

```
// Services that implement IDisposable:  
public class Service1 : IDisposable {}  
public class Service2 : IDisposable {}  
public class Service3 : IDisposable {}  
  
public interface ISomeService {}  
public class SomeServiceImplementation : ISomeService, IDisposable {}  
  
public void ConfigureServices(IServiceCollection services)  
{  
    // The container creates the following instances and disposes them automatically:  
    services.AddScoped<Service1>();  
    services.AddSingleton<Service2>();  
    services.AddSingleton<ISomeService>(sp => new SomeServiceImplementation());  
  
    // The container doesn't create the following instances, so it doesn't dispose of  
    // the instances automatically:  
    services.AddSingleton<Service3>(new Service3());  
    services.AddSingleton(new Service3());  
}
```

415

DÉMONSTRATION



416