(Leçon 01)
Introduction à la technologie .NET
Configurer l'environnement

LOTFI DERBALI, PH.D
DERBALI.IT@GMAIL.COM
HTTPS://GITHUB.COM/DERBALI-IT

Contenu

- .NET Core: Installation et configuration
- Révision des concepts clés en programmation C# et sur l'utilisation de Visual Studio

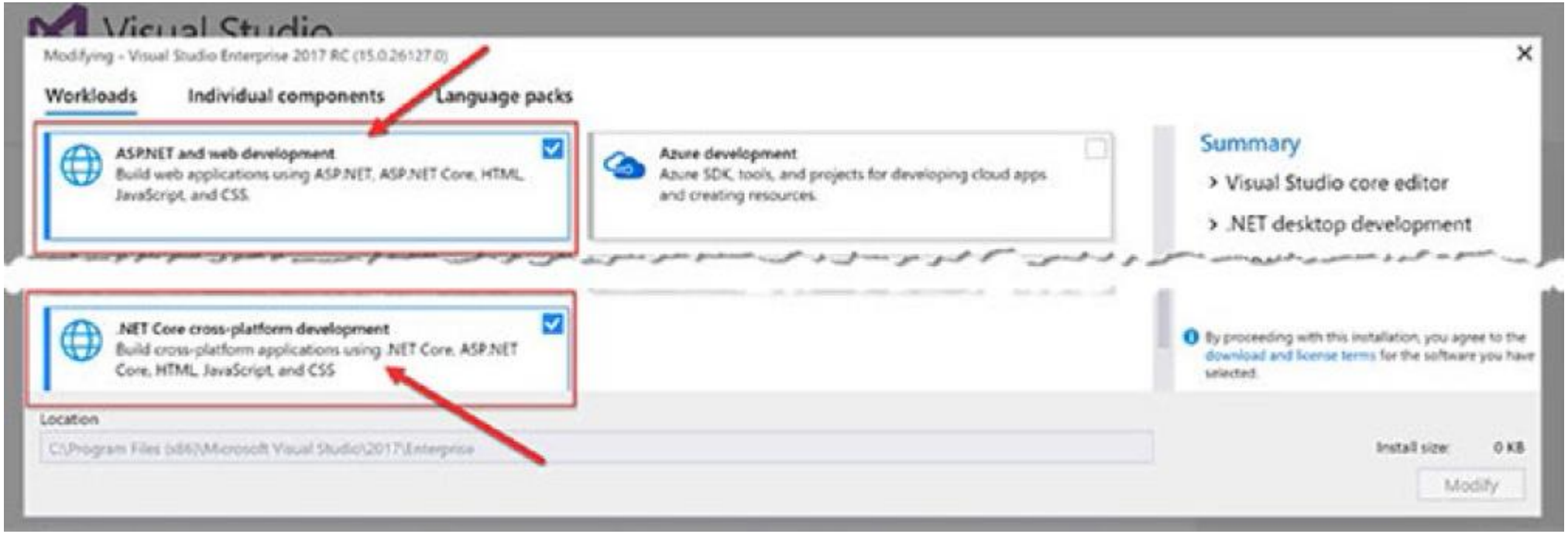
Qu'est-ce que le .NET Core ?

- Le Framework .NET est intrinsèquement lié à Windows depuis ses débuts en 2002
- .NET Core est une transformation complète du Framework .Net sous la forme de composants modulaires, utilisables à la demande, cross plateformes et … open-source !

- Les principales caractéristiques de .NET Core sont :
  - Multiplateforme
    - En utilisant .Net Core vous vous assurez que votre code est (ou sera) portable aisément sur d'autres plateformes.
  - Open source : disponible sur GitHub
  - Déploiement flexible
  - Modulaire :
    - .NET Core est disponible sous forme de petits packages axés sur les fonctionnalités.
    - utilisation des NuGet

Installer Visual Studio Community 2017

- https://www.visualstudio.com/downloads/
  - Il est important de choisir:
    - ASP.NET Web Development
    - .NET Core Cross Platform development



  - Environnements: Windows, Linux, Mac.

Concepts C#

- Les données de C#
  - les nombres entiers
  - les nombres réels
  - les nombres décimaux
  - les caractères et chaînes de caractères
  - les booléens
  - les objets

Conversion entre types simples et types objets

Utilisation des Wrappers

int	Int32
long	Int64
decimal	Decimal
bool	Boolean
char	Char
byte	Byte
float	Float
double	Double
enum	Enum

Les conversions entre nombres et chaînes de caractères

nombre -> chaîne	"" + *nombre*
chaine -> int	*int.Parse(chaine) ou Int32.Parse*
chaîne -> long	*long.Parse(chaine) pu Int64.Parse*
chaîne -> double	*double.Parse(chaîne) ou Double.Parse(chaîne)*
chaîne -> float	*float.Parse(chaîne) ou Float.Parse(chaîne)*

- Déclaration des constantes

const type nom=valeur;

ex : const *float PI=3.141592F*;

- Gestion des exceptions
  - De nombreuses fonctions C# sont susceptibles de générer des exceptions, c'est à dire des erreurs.
  - la clause *try/catch*

```
try{
 appel de la fonction susceptible de générer l'exception
} catch (Exception e){
 traiter l'exception e
}
instruction suivante
```

Les tableaux de données

- Un tableau C# est un objet permettant de rassembler sous un même identificateur des données de même type.

Type[] Tableau[]=new Type[n]

```
int[] entiers=new int[] {0,10,20,30};
```

- Un tableau à deux dimensions pourra être déclaré comme suit :

Type[,] Tableau=new Type[n,m];

```
double[,] réels=new double[,] { {0.5, 1.7}, {8.4, -6}};
```

- Ecriture sur écran
  - Il existe différentes instructions d'écriture à l'écran:

```
Console.Out.WriteLine(expression)
Console.WriteLine(expression)
Console.Error.WriteLine (expression)
```

- Lecture de données tapées au clavier
  - Le flux de données provenant du clavier est désigné par l'objet *Console.In* de type *StreamReader*

```
String ligne=Console.In.readLine();
```

- Structure de choix simple
  - La structure if … else
  - La structure de cas switch … case
- Structure de répétition
  - La structure for
  - La structure foreach
  - La structure while
  - La structure do … while
- *Instructions de gestion de boucle*
  - Break
    - fait sortir de la boucle for, while, do ... while.
  - Continue
    - fait passer à l'itération suivante des boucles for, while, do ... while

- Les énumérations
  - Une énumération est un type de données dont le domaine de valeurs est un ensemble de constantes.

```
enum mention {Passable,AssezBien,Bien,TrèsBien,Excellent};
// une variable qui prend ses valeurs dans l'énumération mentions
mention maMention=mention.Passable:
```

- Classes
  - Un objet est une entité qui contient des données qui définissent son état (on les appelle des propriétés) et des fonctions (on les appelle des méthodes). Un objet est créé selon un modèle qu'on appelle une classe

Les différents accès

- Privé
  - Un champ privé (private) n'est accessible que par les seules méthodes internes de la classe
- Public
  - Un champ public (public) est accessible par toute fonction définie ou non au sein de la classe
- Protégé
  - Un champ protégé (protected) n'est accessible que par les seules méthodes internes de la classe ou d'un objet dérivé (voir ultérieurement le concept d'héritage).

- Les accesseurs et les modifieurs
  - Une propriété permet de lire (**get**) ou de fixer (**set**) la valeur d'un attribut.

```
class Person
{
    public int Id { get; set; }
    public string Name { get; set; }

    public Person(int id, string name)
    {
        Id = id;
        Name = name;
    }

    public override string ToString()
    {
        return "Person["+Id + "   " + Name + "]" ;
    }
}
```

Héritage

- La classe enseignant dérive de la classe personne
- ***Pas d'héritage multiple en C#***
- Surcharge d'une méthode ou d'une propriété
  - le mot réservé *base* fait référence au constructeur de la classe de base

```
class Teacher : Person
{
    public float Salary { get; set; }

    public Teacher(int id, string name, float salary) : base(id, name)
    {
        Salary = salary;
    }
    public override string ToString()
    {
        return "Teacher[" + base.ToString() + "    " + this.Salary + "]";
    }
}
```

Les structures

- La structure C# est directement issue de la structure du langage C et est très proche de la classe.
- Différences avec les classes
  - La notion d'héritage n'existe pas avec les structures.
  - Passage par valeur dans les structures

```
struct NewColor {
    public string golor { get; set; }
    public string hexcode { get; set; }

    public NewColor(string golor, string hexcode)
    {
        this.golor = golor;
        this.hexcode = hexcode;
    }
}
```

Les interfaces

- Une interface est un ensemble de prototypes de méthodes ou de propriétés qui forme un contrat.
- Une classe qui décide d'implémenter une interface s'engage à fournir une implémentation de toutes les méthodes définies dans l'interface.

```
interface IStats
{
    int NumberOfElements();
    double Mean();
}
```

```
class Group : IStats
{
    public List<Person> Students { get; set; }

    public Group()
    {
        this.Students = new List<Person>();
    }

    public int NumberOfElements()
    {
        return this.Students.Count;
    }

    public double Mean()
    {
        throw new NotImplementedException();
    }

}
```