



MODULE C#

- 1) Introduction**
- 2) Classes et objets**
- 3) Accessibilité**
- 4) Types**
- 5) Attributs, propriétés et constructeur**
- 6) Fonctions et événements**
- 7) Héritage (abstract / virtual et interface)**
- 8) Opérateurs**
- 9) Conditions**
- 10) Exceptions**
- 11) Gestion de fichier**
- 12) Base de données**

MODULE C#

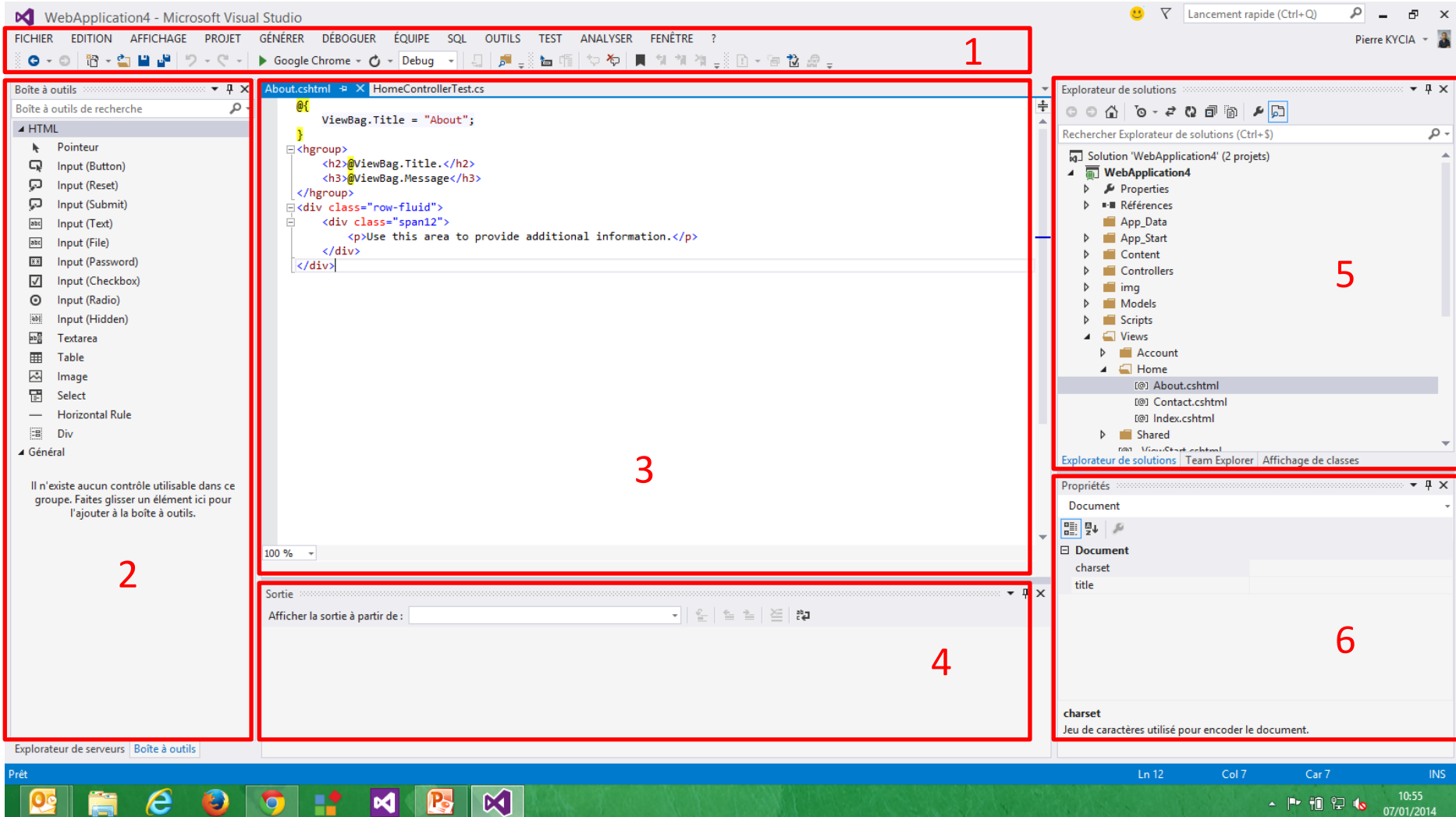
INTRODUCTION



- ✓ **C# = Java + C++**

- ✓ **Pour développer il faut un IDE**
 - Visual Studio 2013 Express
 - Framework .NET 4.5

- ✓ **Langages du framework .NET**
 - VB
 - C++
 - C# → le plus utilisé
 - F#

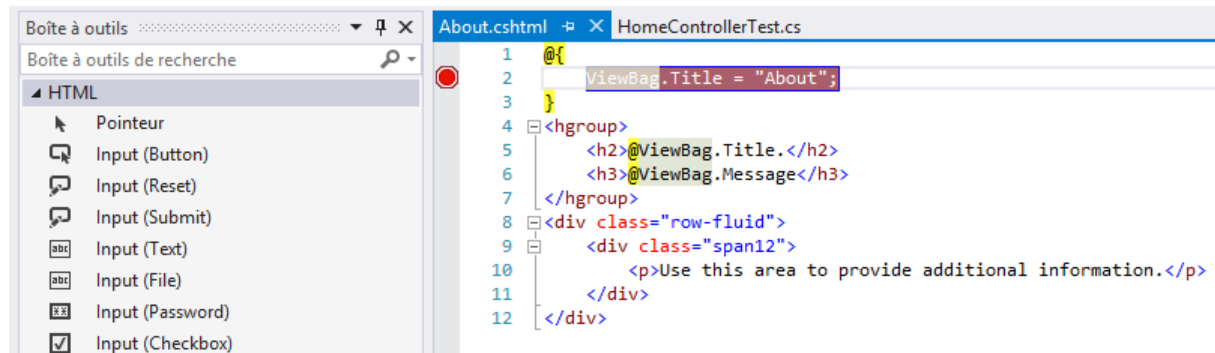


The screenshot shows the Microsoft Visual Studio IDE for a project named 'WebApplication4'. The interface is annotated with red numbers 1 through 6:

- 1**: Points to the top menu bar and toolbar.
- 2**: Points to the 'Boîte à outils' (Toolbox) on the left, which contains HTML and General controls.
- 3**: Points to the central code editor showing the 'About.cshtml' file.
- 4**: Points to the 'Sortie' (Output) window at the bottom.
- 5**: Points to the 'Explorateur de solutions' (Solution Explorer) on the right, showing the project structure.
- 6**: Points to the 'Propriétés' (Properties) window on the right, showing document properties like 'charset'.

- ✓ **Resharper → la meilleure extension**
- ✓ **VS Commands → meilleure visualisation de code**
- ✓ **StyleCop → convention de code, qualité de code**
- ✓ **Ghost Doc → génère les commentaires**
- ✓ **Image Optimizer → optimise vos images**
- ✓ **Web Essentials → utile pour les designers web (CSS...)**

✓ Créer un point d'arrêt



✓ Lancer le debugger : Debug → Start Debug (F5)

✓ Step Over (F10) → exécute le code de chaque ligne

✓ Step Into (F11) → rentre dans le code

MODULE C#

CLASSE ET OBJET



- ✓ **Pour ajouter une classe dans VS**
 - Clic droit dans l'explorateur de solution → Add → Class

```
1  [ ] using System;  
2  [ ] using System.Collections.Generic;  
3  [ ] using System.Linq;  
4  [ ] using System.Web;  
5  
6  [ ] namespace WebApplication4.Models  
7  [ ] {  
8  [ ]     public class MaClasse  
9  [ ]     {  
10 [ ]  
11 [ ]     }  
12 [ ] }
```

- ✓ **using ???**

- ✓ **using → permet d'emboîter à l'infinie des espaces de noms**
 - `using System.Collections;
xxx;`
 - `using monAlias = System.Collections;
monAlias.xxx;`

- ✓ Un objet est une instance de classe sauf si statique

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace WebApplication4.Models
7  {
8      public class MaClasse2
9      {
10         // Création de l'objet maClasse de type MaClasse
11         MaClasse maClasse = new MaClasse();
12     }
13 }
```

- ✓ **X attributs**
- ✓ **X propriétés**
- ✓ **X constructeurs**
- ✓ **X méthodes**
 - D'instances
 - Statiques
- ✓ **X événements**

```

7  {
8  -  /// <summary>
9  -  /// Ma Classe
10 -  /// </summary>
11 -  public class MaClasse
12 {
13 +  ATTRIBUTS
22
23 +  PROPERTIES
33
34 +  CONSTRUCTOR
46
47 +  METHODS
49
50 +  STATICS
52
53 +  EVENTS
55 }
56 }

```

MODULE C#

ACCESSIBILITE



- ✓ **5 niveaux d'accessibilité**
 - **public** → accès non limité
 - **protected** → restreint à la classe ou types dérivés
 - **internal** → restreint à l'assembly en cours
 - **protected internal** → restreint à l'assembly ou à la classe conteneur ou types dérivés
 - **private** → restreint au type conteneur

Membres de	Accessibilité des membres par défaut	Accessibilité déclarée autorisée du membre
enum	public	Aucun
class	private	public protected internal private protected internal
interface	public	Aucun
struct	private	public internal private

```

7  {
8  -  /// <summary>
9  -  /// Ma Classe
10 -  /// </summary>
11 -  public class MaClasse
12 -  {
13 -  -  #region PROPERTIES (5)
14 -  |
15 -  |      public string Nom { get; set; }
16 -  |      private string Prenom { get; set; }
17 -  |      protected int Id { get; set; }
18 -  |      internal int Code { get; set; }
19 -  |      protected internal string Telepone { get; set; }
20 -  |
21 -  |      #endregion
22 -  |
23 +  |      CONSTRUCTOR (1)
35 -  |
36 +  |      METHODS
38 -  |
39 +  |      STATICS
41 -  |
42 +  |      EVENTS
44 -  |      }
45 -  }

```


- ✓ **Champs / attributs en privé**
- ✓ **Propriétés en publique « sauf si inutile »**
- ✓ **Méthodes en publique « sauf si inutile »**

```

7  {
8  /// <summary>
9  /// Ma Classe
10 /// </summary>
11 public class MaClasse
12 {
13     #region ATTRIBUTS
14
15     private string _nom;
16     private string _prenom;
17     private int _id;
18     private int _code;
19     private string _telephone;
20
21     #endregion
22
23     #region PROPERTIES
24
25     public string Nom
26     {
27         get { return _nom; }
28         set { _nom = value; }
29     }
30
31
32     #endregion

```

MODULE C#

LES TYPES



Type C#	Type .NET Framework
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
Object	System.Object
short	System.Int16
ushort	System.UInt16
string	System.String

Type	Plage	Taille
sbyte	-128 à 127	Entier 8 bits signé
byte	0 à 255	Entier 8 bits non signé
char	U+0000 à U+ffff	Caractère Unicode 16 bits
short	-32 768 à 32 767	Entier 16 bits signé
ushort	0 à 65 535	Entier 16 bits non signé
int	-2 147 483 648 à 2 147 483 647	Entier 32 bits signé
uint	0 à 4 294 967 295	Entier 32 bits non signé
long	-9,223,372,036,854,775,808 à 9,223,372,036,854,775,807	Entier 64 bits signé
ulong	0 à 18,446,744,073,709,551,615	Entier 64 bits non signé



TYPES

Flottants

Type	Plage approximative	Précision
float	$\pm 1,5e-45$ à $\pm 3,4e38$	7 chiffres
double	$\pm 5,0e-324$ à $\pm 1,7e308$	15-16 chiffres

- ✓ **List**
- ✓ **Array**
- ✓ **ArrayList → à éviter**
- ✓ **HashTable**
- ✓ **Dictionary → très performant**
- ✓ **...**

MODULE C#

Attributs, propriétés et
constructeur



- ✓ **Définis les champs de la classe**
- ✓ **Exemple avec la classe voiture**

```
private string _marque;  
private double _poid;  
private string _couleur;  
private int _puissance;  
private bool _estAssure;
```
- ✓ **Rappel : les attributs sont « toujours » privés**

- ✓ **Permet d'obtenir et définir la valeur des attributs**

- ✓ **Exemple avec la classe voiture**

```
private string _marque;  
public string Marque  
{  
    get { return _marque; }  
    set { _marque = value; }  
}
```

- ✓ **Rappel : les propriétés sont « toujours » publiques**

- ✓ Permet d'instancier une classe avec des paramètres en arguments si besoin

- ✓ Exemple avec la classe voiture

`public Voiture() { } → Constructeur par défaut`

```
public Voiture(string marque, float poids)
{
    this.Marque = marque;
    this.Poids = poids;
}
```

`Voiture maVoiture = new Voiture("BMW", 2000);`

- ✓ Rappel : les constructeurs sont « toujours » publics

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5
6 namespace WebApplication4.Models
7 {
8     public class Voiture
9     {
10         #region ATTRIBUTES
11         private string _marque;
12         private float _poids;
13         #endregion
14
15         #region PROPERTIES
16         public string Marque
17         {
18             get { return _marque; }
19             set { _marque = value; }
20         }
21
22         public float Poids
23         {
24             get { return _poids; }
25             set { _poids = value; }
26         }
27         #endregion
28
29         #region CONSTRUCTOR
30         public Voiture(string marque, float poids)
31         {
32             this.Marque = marque;
33             this.Poids = poids;
34         }
35         #endregion
36     }
37 }

```

MODULE C#

Fonctions et événements



- ✓ **2 types**
 - Fonction → retourne une valeur
 - Procédure → ne retourne rien

```
public string AfficherInformationVoiture()  
{  
    return "Marque : " + this.Marque + " Poids : " +  
    this.Poids.ToString();  
}
```

```
public void AfficherInformationVoitureDansConsole()  
{  
    Console.WriteLine("Marque : " + this.Marque + " Poids : " +  
    this.Poids.ToString());  
}
```

- ✓ **Créer une instance de classe (si nous sommes dans une autre classe) puis les invoquer**

```
// Instance de classe
```

```
Voiture maVoiture = new Voiture();
```

```
// Appel de la fonction AfficherInformationVoiture
```

```
maVoiture.AfficherInformationVoiture();
```

```
// Appel de la procédure AfficherInformationVoitureDansConsole
```

```
maVoiture.AfficherInformationVoitureDansConsole();
```

- ✓ **Si nous sommes déjà dans la classe Voiture, comment faire ?**
- ✓ **Qu'obtenons-nous ?**

```
// Instance de classe
```

```
Voiture maVoiture = new Voiture();
```

```
// Appel de la fonction AfficherInformationVoiture
```

```
string tmp = maVoiture.AfficherInformationVoiture();
```

```
// Appel de la procédure AfficherInformationVoitureDansConsole
```

```
maVoiture.AfficherInformationVoitureDansConsole();
```


✓ Imaginons cette classe **Personne**

```
public class Personne
{
    public static void AfficherPersonneConsole()
    {
        return;
    }
}
```

✓ Comment appeler la méthode **AfficherPersonneConsole** ?

- Depuis cette même classe ?
- Depuis la classe **Voiture** ?

✓ Intérêt des éléments statiques ?

- ✓ **Permet de notifier une autre classe lorsqu'une action a été faite**
- ✓ **La classe qui déclenche l'événement = EDITEUR**
- ✓ **La classe qui reçoit l'événement = ABONNEE**
- ✓ **Exemple : le click sur un bouton sur un projet Windows Forms**
 - Créez le projet (Windows → Application Windows Forms)
 - Ajoutez un bouton depuis la boîte à outils
 - Dans propriétés, cliquez sur l'icône éclair
 - Trouvez click puis nommez votre événement et entrée



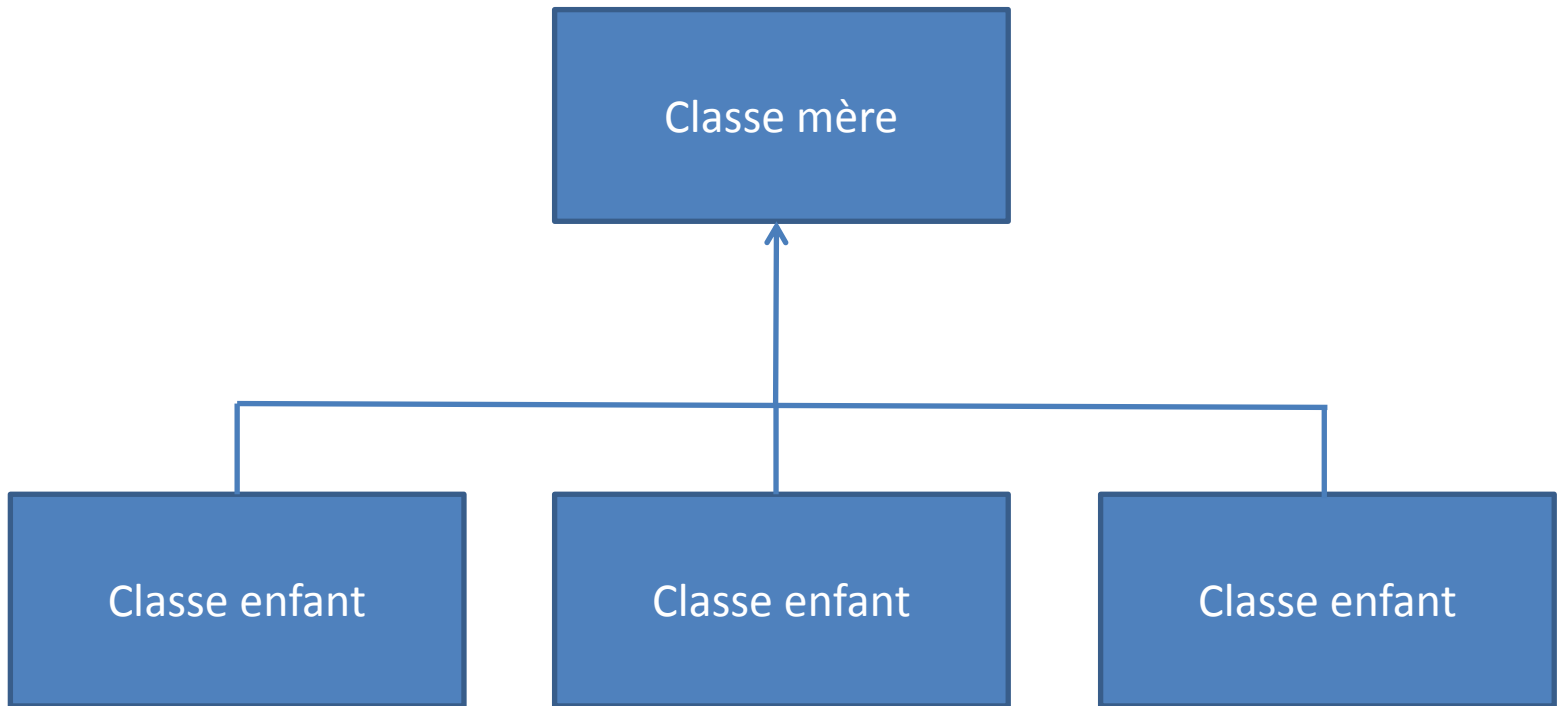
- ✓ Click
- ✓ MouseClick
- ✓ MouseCapturedClick
- ✓ MouseDown
- ✓ MouseUp
- ✓ KeyDown
- ✓ ...

MODULE C#

Héritage et compagnie



- ✓ Permet de réutiliser, modifier ou étendre le comportement défini dans d'autres classes



- ✓ Imaginons une classe **Personne** de ce genre

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

public Personne (string nom, string prenom, int age)
{
    this.Nom = nom;
    this.Prenom = prenom;
    this.Age = age;
}

public void AfficherInformation()
{
    Console.WriteLine("NOM : " + this.Nom + " – PRENOM : " + this.Prenom + " – AGE : " + this.Age);
}
```

- ✓ Imaginons une classe **Adulte** de ce genre

```
public class Adulte
{
    public string Nom { get; set; }
    public string Prenom { get; set; }
    public int Age { get; set; }
    public double Poids { get; set; }
}

public Adulte (string nom, string prenom, int age, double poids)
{
    this.Nom = nom;
    this.Prenom = prenom;
    this.Age = age;
    this.Poids = poids;
}

public void AfficherInformation()
{
    Console.WriteLine("NOM : " + this.Nom + " – PRENOM : " + this.Prenom + " – AGE
: " + this.Age + " – POIDS : " + this.Poids) ;
}
```

- ✓ Imaginons une dernière classe **Enfant** de ce genre

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

public Enfant (string nom, string prenom, int age, string sport)
{
    this.Nom = nom;
    this.Prenom = prenom;
    this.Age = age;
    this.Sport = sport;
}

public void AfficherInformation()
{
    Console.WriteLine("NOM : " + this.Nom + " – PRENOM : " + this.Prenom + " – AGE
: " + this.Age + " – SPORT : " + this.Sport);
}
```


- ✓ Que remarquez-vous au niveau du code ?
- ✓ Que pouvons-nous faire ?

- ✓ Regrouper tous les éléments commun entre ces 3 classes
- ✓ Commençons par les propriétés

Personne

Nom

Prénom

Age

Adulte

Nom

Prénom

Age

Poids

Enfant

Nom

Prénom

Age

Sport

- ✓ Que pouvons-nous regrouper ?

✓ Continuons avec les constructeurs

Personne

`public Personne (string nom, string prenom, int age)`

Adulte

`public Adulte (string nom, string prenom, int age, double poids)`

Enfant

`public Enfant (string nom, string prenom, int age, string sport)`

✓ Que pouvons-nous regrouper ?

- ✓ **Simplifions la méthode pour la rendre générique (afficher toutes les informations communes aux 3 classes)**
- ✓ **1 volontaire ?**

- ✓ Et en C# l'héritage, comment on fait ?

public class MaClasseEnfant : MaClasseMere

- ✓ 1 volontaire pour la classe Personne ?
- ✓ 1 volontaire pour la classe Adulte ?
- ✓ 1 volontaire pour la classe Enfant ?

- ✓ C'est une classe dont l'implémentation n'est pas complète et qui n'est pas instanciable
- ✓ Son rôle est de factoriser le code
- ✓ Définie avec le mot clé « abstract »

Exemple

```
public abstract class Personne
```

De ce fait, écrire

```
Personne p = new Personne(); // générer une erreur à la compilation
```

Par contre

```
Personne p = new Adulte("IBRAHIMOVIC", "ZLATAN", 30, 90); // OK
```

- ✓ **Comment accéder à la propriété « Poids » de la classe Adulte ?**

```
Personne p = new Adulte("IBRAHIMOVIC", "ZLATAN", 30, 90);  
p.Nom; // OK  
p.Prenom; // OK  
p.Age; // OK  
p.Poids; // Non proposé !!!
```

- ✓ **Il faut effectuer une conversion de notre variable**

```
Personne p = new Adulte("IBRAHIMOVIC", "ZLATAN", 30, 90);  
Double poids = ((Adulte) p).Poids;
```

- ✓ **2 mots clés à retenir**
 - Virtual
 - abstract

- ✓ **Leur différence ?**
 - Virtual → nous ne sommes pas obligés d'implémenter la méthode dans la classe enfant
 - Abstract → nous serons obligés d'implémenter la méthode dans la classe enfant

Exemple

```
public abstract class Personne
{
    public abstract void AfficherInformation();
}

public class Adulte : Personne
{
    public override void AfficherInformation()
    {
        Console.WriteLine("Hello");
    }
}
```

Exemple

```
public abstract class Personne
{
    public virtual void AfficherInformation()
    {
        Console.WriteLine("Hello");
    }
}

public class Adulte : Personne
{
}
```

- ✓ Une interface contient uniquement les signatures des méthodes, propriétés ou événements. L'implémentation de ces derniers se fait dans la classe qui implémente l'interface
- ✓ Une interface seule ne sert à rien
- ✓ Une interface est toujours publique (accessibilité)
- ✓ Son but étant de rendre accessible des services sans connaître l'implémentation du code (on ne voit que les signatures des méthodes)

Exemple

interface IMetier

```
{  
    string Metier { get; set; }  
    void AfficherMetier();  
}
```

public class Acteur : IMetier

```
{  
    private string _metier;  
  
    public string Metier  
    {  
        get { return _metier; }  
        set { _metier = value; }  
    }  
  
    public Acteur (string metier)  
    {  
        this.Metier = metier;  
    }  
  
    public void AfficherMetier()  
    {  
        Console.WriteLine("Je suis {0}", this.Metier);  
    }  
}
```

- ✓ Il est possible d'utiliser plusieurs interfaces sur une même classe
- ✓ 1 volontaire pour écrire un exemple ?

- ✓ Une interface est toujours publique et permet de rendre accessible sans contrainte d'implémentation des services
- ✓ Une classe abstraite n'est pas instanciable et factorise le code
- ✓ Une interface peut hériter de plusieurs classes
- ✓ Une classe abstraite n'hérite qu'une seule classe

MODULE C#

LES OPERATEURS



Exemple

```
int i = 1 + 1; // addition
```

```
int j = 1 - 1; // soustraction
```

```
int k = 1 * 2; // multiplication
```

```
int l = 10 / 2; // division
```

```
int m = 5 % 2; // modulo
```

Affectation simplifiée

```
int tmp = 5;
```

```
tmp = tmp + 5;
```

```
tmp += 5;
```

```
tmp -= 5;
```

```
tmp *= 5;
```

```
tmp /= 5;
```

```
tmp %= 5;
```


Opérateur ET (&&)

```
bool a = true && true;  
bool b = true && false;  
bool c = false && true;  
bool d = false && false;
```

Opérateur OU (||)

```
bool a = true || true;  
bool b = true || false;  
bool c = false || true;  
bool d = false || false;
```

Opérateur OU exclusif (^)

```
bool a = true ^ true;  
bool b = true ^ false;  
bool c = false ^ true;  
bool d = false ^ false;
```

ATTENTION VALABLE UNIQUEMENT SUR LES TYPES PRIMITIFS (int, string, float, bool...)

```
int a = 10;  
int b = 5;  
bool reponse = (b == a);
```

Supérieur (>)

Inférieur (<)

Supérieur ou égal (>=)

Inférieur ou égal (<=)

Différent (!=)

Négation (!)

```
bool a = true;  
bool b = false;  
Console.WriteLine(!a); // false  
Console.WriteLine(!b); // true
```

Opérateur ET (&)

$1101 \rightarrow 13$

$1001 \rightarrow 9$

$1101 \& 1001 = 1001$

Opérateur OU (|)

$1101 \rightarrow 13$

$1001 \rightarrow 9$

$1101 | 1001 = 1101$

Opérateur OU Exclusif (^)

$1101 \rightarrow 13$

$1001 \rightarrow 9$

$1101 \wedge 1001 = 0100$

MODULE C#

CONDITIONS





LES CONDITIONS

IF - IF ELSE – IF ELSE IF ELSE

```
if (expression booléenne)
{
    // Code
}
```

```
if (expression booléenne)
{
    // Code
}
else
{
    // Code
}
```

```
if (expression booléenne)
{
    // Code
}
else if (expression booléenne)
{
    // Code
}
else
{
    // Code
}
```

```
string val = "LOSC";  
  
switch (val)  
{  
    case "LOSC":  
        Console.WriteLine("BIEN");  
        break;  
  
    case "OM":  
        Console.WriteLine("PAS MAL");  
        break;  
  
    case "RCL":  
        Console.WriteLine("LIGUE 2");  
        break;  
  
    default:  
        Console.WriteLine("POURQUOI PAS");  
        break;  
}
```

Bonne alternative au IF ELSE IF et plus performant

```
bool ok = true;  
string val = ok ? "VRAI" : "FAUX";
```

Plus rapide à écrire mais plus difficile a lire

TANT QUE

```
while (expression booléenne)
{
    // Code
}
```

POUR

```
for (int i = 0; i < 10; i++)
{
    // Code à exécuter
}
```

FAIRE TANT QUE

```
do
{
    // Code à exécuter
} while (expression booléenne)
```

POUR CHAQUE

```
List<string> maListe = new List<string>();  
maListe.Add("Element 1");  
maListe.Add("Element 2");  
maListe.Add("Element 3");  
maListe.Add("Element 4");
```

```
foreach (string val in maListe)  
{  
    // Code  
}
```

MODULE C#

EXCEPTIONS



- ✓ **BUT : GERER LES IMPREVUS ET INTERCEPTER L'ERREUR POUR EVITER UN PLANTAGE**
- ✓ **DOIT ETRE COMPRISE ENTRE UN BLOC TRY / CATCH / FINALLY**

Exemple

```
try
{
    // Code
}
catch(Exception ex)
{
    Console.WriteLine("Erreur : {0} " + ex.Message);
}
finally
{
    Console.WriteLine("On affiche ce message dans tous les cas");
}
```

- ✓ Il est possible de créer sa propre classe d'exception (indispensable pour des besoins spécifiques)
- ✓ Il faut créer une classe qui hérite de la classe Exception

Exemple

```
public class MonException : Exception
```

```
{  
    private DateTime _dateErreur;  
  
    public MonException()  
    {  
        _dateErreur = DateTime.Now;  
    }  
  
    public void AfficherErreur()  
    {  
        MessageBox.Show(base.Message, string.Format("Erreur le {0}",  
            _dateErreur.ToString()), MessageBoxButtons.OK, MessageBoxIcon.Error);  
    }  
}
```

Utilisation

```
Try  
{  
  
} catch (MonException ex)  
{  
    ex.AfficherErreur();  
}
```

- ✓ **Exception** → classe de base pour toutes les exceptions
- ✓ **NullReferenceException** → objet nul référencé
- ✓ **ArgumentException** → classe de base pour les arguments
- ✓ **FileNotFoundException** → classe de base pour les exceptions liées aux fichiers
- ✓ **SqlException** → classe quand une erreur SQL Server retourne une erreur
- ✓ ...

MODULE C#

GESTION DES FICHIERS



Lire le fichier et mettre son contenu dans un string :

```
string contenu = System.IO.File.ReadAllText(@"cheminFichier");
```

Lire le fichier ligne par ligne

```
string[] mesLignes = System.IO.File.ReadAllLines(@"cheminFichier");
```

Ecrire le contenu d'un string dans un fichier

```
string monTexte = "Amis du jour, bonjour";
```

```
System.IO.File.WriteAllText(@"CheminFichier", monTexte);
```

Ecrire le contenu d'un tableau de string dans un fichier

```
String[] monTab = { "Amis du jour, bonjour", "Texte 2", "Texte 3" };
```

```
System.IO.File.WriteAllLines(@"CheminFichier", monTab);
```


Créer un fichier

```
using (FileStream fs = File.Create(CheminDuFichier))
```

```
{
```

```
    Byte[] info = new UTF8Encoding(true).GetBytes("Contenu");
```

```
    fs.Write(info, 0, info.Length);
```

```
}
```

Supprimer un fichier

```
File.Delete(CheminDuFichier);
```

MEME LOGIQUE POUR LES DOSSIERS

MODULE C#

BASE DE DONNEES



ADO.NET → Permet la communication avec les bases de données (SQL Server, Oracle...)

Les objets disponibles

Connection

Command

DataReader

DataSet

DataAdapter

Description

Ouvre une connexion vers une source de données

Exécute une commande sur une source de données

Lit un flux de données en lecture seule à partir d'une source de données en mode connecté

Lit un flux de données en mode déconnecté

Remplit un DataSet et répercute les mises à jour dans la source de données

Pour SQL Server → `System.Data.SqlClient.SqlConnection`

Pour Oracle → `System.Data.OracleClient.OracleConnection`

Pour MySql → `System.Data.MySqlClient.MySqlConnection`

L'ouverture de la connexion se fait grâce à « Open » et la fermeture par « Close »

Exemple

```
string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial  
Catalog=Northwind";
```

```
try  
{
```

```
    SqlConnection oConnection = new SqlConnection(strConnexion);  
    oConnection.Open();  
    Console.WriteLine("Etat de la connexion : " + oConnection.State);  
    oConnection.Close();
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);
```

```
}
```

Pour créer une commande il faut utiliser « SqlCommand »

Pour SQL Server → `System.Data.SqlClient.SqlCommand`

...

SqlCommand possède 3 méthodes « Execute » principales :

- `ExecuteReader` → récupère des données et retourne un objet `DataReader`
- `ExecuteScalar` → récupère une valeur unitaire (nombre de ligne...)
- `ExecuteNonQuery` → exécute une commande ne retournant pas de lignes

Exemple

```
string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial  
Catalog=Northwind";  
string strRequete = "INSERT INTO Region VALUES (5,'Sud')";  
try  
{  
    SqlConnection oConnection = new SqlConnection(strConnexion);  
    SqlCommand oCommand = new SqlCommand(strRequete,oConnection);  
    oConnection.Open();  
    oCommand.ExecuteNonQuery();  
    oConnection.Close();  
}  
catch (Exception e)  
{  
    Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);  
}
```

Rappel : « DataReader » est le résultat de la méthode « ExecuteReader »

Cet objet stocke en mémoire une seule ligne à la fois (performances ++)

Recommandé à utiliser si :

- **Pas besoin de réaliser un cache des données**
- **Jeu d'enregistrement très important**
- **Accès à des données rapidement**

Une ligne entière est chargée en mémoire à chaque appel de la méthode « Read »

Pour accéder aux champs de la ligne il faut utiliser :

- **GetInt32** → si colonne de type entier
- **GetString** → si colonne de type chaine de caractère
- **GetDate** → si colonne de type dateTime
- ...

La méthode « **Close** » permet de fermer un DataReader

Pour augmenter les performances, il est parfois nécessaire de soumettre X requêtes à la fois, pas de soucis, DataReader possède la méthode « **NextResult** » qui permet de passer d'un jeu d'enregistrement à un autre

NextResult = NextRecordSet

EXEMPLE

```
string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" + "Initial Catalog=Northwind";
string strRequete = "SELECT CategoryID, CategoryName FROM Categories;" + "SELECT EmployeeID,
LastName FROM Employees";
try
{
    SqlConnection oConnection = new SqlConnection(strConnexion);
    SqlCommand oCommand = new SqlCommand(strRequete,oConnection);
    oConnection.Open();
    SqlDataReader oReader = oCommand.ExecuteReader();
    do
    {
        Console.WriteLine("\t{0}\t{1}", oReader.GetName(0), oReader.GetName(1));
        while (oReader.Read())
        {
            Console.WriteLine("\t{0}\t{1}", oReader.GetInt32(0), oReader.GetString(1));
        }
    }
    while (oReader.NextResult());
    oReader.Close();
    oConnection.Close();
}
catch (Exception e)
{
    Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);
}
```

Comment appeler une procédure stockée ?

Utiliser l'objet « SqlCommand »

Exemple

```
SqlCommand maCommande = new SqlCommand("NomPS", "Connection ");  
maCommande.CommandType = CommandType.StoredProcedure;
```

Pourquoi utiliser les procédures stockées ?

EXEMPLE

```
string strConnexion = "Data Source=localhost; Integrated Security=SSPI;" +  
"Initial Catalog=Northwind";  
string strProcedureStockee = "GetCustomerOrders";  
try  
{  
    SqlConnection oConnection = new SqlConnection(strConnexion);  
    SqlCommand oCommand = new SqlCommand(strProcedureStockee,  
oConnection);  
    oCommand.CommandType = CommandType.StoredProcedure;  
    oCommand.Parameters.AddWithValue("@ParamPS",  
valeurDuParametre);  
    oConnection.Open();  
    oCommand.ExecuteNonQuery();  
    oConnection.Close();  
}  
catch (Exception e)  
{  
    Console.WriteLine("L'erreur suivante a été rencontrée :" + e.Message);  
}
```

Utiliser ConnectionStrings du fichier de configuration pour stocker la connection à la base de données

Exemple

```
<connectionStrings>
  <add name="conString"
    connectionString="Data Source=.\SQLEXPRESS;
                      database=Northwind;Integrated Security=true"/>
</connectionStrings>
```

Mettre chaque appel d'une PS entre using

Exemple

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    // Votre code
}
```

Utiliser le plus souvent possible des PS au lieu de requêtes en dur

MODULE C#

AUTRES



Pour les fichiers de configuration :

<http://nico-pyright.developpez.com/tutoriel/vc2005/configurationsectioncsharp/>

MODULE C#

TP



BESOIN LOGICIEL

Visual Studio 2012 / 2013 Express for Windows

SQL Server 2012 Express

SQL Server 2012 Management Studio Express

TYPE DE PROJET

Application Windows Forms

LANGAGE DE PROGRAMMATION

C#

BDD

Créer une BDD avec 2 tables en relation (1 – N)

Chaque table doit avoir comme champs un ID auto-increment, une description texte et une date

Alimenter les 2 tables avec 20 lignes

Une application WinForms avec 2 onglets

- **ONGLET FICHIER** : Premier onglet permet de charger un fichier local (extension txt UNIQUEMENT) à son poste et afficher le contenu du fichier dans la page
- Possibilité de modifier le contenu du fichier et d'enregistrer les modifications avec message de confirmation ou d'erreur
- **ONGLET BDD** : Deuxième onglet permet de se connecter à une BDD SQL Server en rentrant les informations manuellement (voir page de login SQL SERVER 2012 Management Studio Express pour exemple)
- Create Read Update Delete sur les 2 tables
- Lecture sur le chargement de la page, suppression sur sélection d'une ou X lignes, mise à jour d'une ligne sélectionné sur clic d'un bouton, création d'une nouvelle entrée sur clic d'un bouton

MERCI DE VOTRE ATTENTION



AVEZ-VOUS DES QUESTIONS ?