Concepts avancés

Programmation .Net avec C#

Utilisation des attributs

- On peut définir des métadatas en utilisant « [ ] »
- Valable pour les classes, les méthodes, les propriétés …
- Utiles pour :
  - Informations de version.
  - Descriptions dans les assemblies.
  - Descriptions de services web.
  - Sérialisation des objets.
  - …

Utilisation des attributs

Méthode obsolète

```
[Obsolete("Utiliser la métode 'Methode2'")]
public void Methode1()
{
    // Traitement
}
```

Propriété sérialisable

```
[DataMember]
public int Valeur { get; set; }
```

Gestion des exceptions

- Inconvénients d'une gestion d'erreurs non structurée :
  - Le code est difficile à lire, à débugger et à maintenir.
  - Il est facile de laisser échapper des erreurs.
- Avantages d'une gestion structurée des exceptions :
  - Supportée par de nombreux langages.
  - Permet de créer des blocs protégés de code.
  - Permet de filtrer les exceptions un peu comme avec un Select Case.
  - Permet de faire des traitements imbriqués.
  - Le code est plus facile à lire, à déboguer et à maintenir.
- Utilisation des exceptions :
  - Le bloc Try, Catch, Finally.

Gestion des exceptions

```
try
{
    // Le code à essayer
}
catch
{
    // Définition des exceptions et les actions à prendre
    // Il peut y avoir plusieurs traitements
}
finally
{
    // Bloc optionnel
    // S'exécute dans tous les cas
}
```

Le bloc Try, Catch, Finally

Gestion des exceptions

```
int i1 = 22, i2 = 0;
double result;

try
{
    result = i1 / i2;
    Console.WriteLine(result.ToString());
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
finally
{
    Console.WriteLine("Finally");
}
```

Exemple

Gestion des exceptions

- La classe System.Exception :
  - Message :
    - Pourquoi l'exception est survenue.
  - Source :
    - Nom de l'application ou de l'objet qui a généré l'exception.
  - StackTrace :
    - Historique de l'exception.
  - ToString :
    - Nom , message, nom intérieure et historique de l'exception

Gestion des exceptions

```
try
{
    // Opérations sur des variables
}
catch (DivideByZeroException)
{
    Console.WriteLine("Vous avez essayer de diviser par 0");
}
catch (OverflowException)
{
    Console.WriteLine("Vous avez rencontré un Overflow");
}
catch (Exception ex)
{
    Console.WriteLine("L'exception : " + ex.Message + " est survenue");
}
finally
{
    Console.WriteLine("Finally");
}
```

Filtrer les exceptions

Utilisation des génériques

- Equivalent au « Template » du C++.
  - Concept de paramètres et de type.
  - A privilégier aux « ArrayList ».
  - Namespace : « System.Collections.Generic ».
- Précise à la déclaration, un ou plusieurs types, supportés par la classe.
- Avantages :
  - Pas de casting implicite en Object.
  - Meilleure gestion du volume.
  - Plus rapide surtout lorsque les éléments sont des types valeur.
  - Peut s'appliquer aux classes, interfaces, méthodes, délégués …

Utilisation des génériques

```
List<string> list = new List<string>()
{
    "Element1",
    "Element2",
    "Element3",
    "Element4",
    "Element5",
    "Element6",
    "Element7",
    "Element8"
};
```

```
Dictionary<int, string> dic = new Dictionary<int, string>();

dic.Add(5, "Cinq");
dic.Add(10, "Dix");
dic.Add(9, "Neuf");
dic.Add(1, "Un");

KeyValuePair<int, string> key1 = dic.ElementAt(1);

string valeur = string.Empty;
dic.TryGetValue(9, out valeur);
```

```
ObservableCollection<int> listeEntier = new ObservableCollection<int>();

listeEntier.Add(5);
listeEntier.Add(14);
listeEntier.Add(3);
listeEntier.Insert(1, 230);
listeEntier.RemoveAt(0);
```

Les delegates

- Un mécanisme qui permet d'appeler une méthode dont le prototype est connu mais dont l'implémentation est donnée à l'exécution.
- Similaire aux pointeurs de fonctions en C++.
- Basé sur la classe « System.Delegate ».
- En général, on utilise un délégué quand on veut passer une méthode en paramètres d'une autre méthode.
- Son intérêt :
  - Souplesse.
  - Réorganisation / Refactorisation de code.

Les delegates

- Étapes d'utilisation :
  1. Créer une définition de délégué (mot clé delegate).
  2. Créer des méthodes qui ont les mêmes types de paramètres et de valeurs retournées.
  3. Créer un délégué depuis la définition de 1 et les méthodes de 2.
  4. Utiliser le délégué.

Les delegates

```
public class ClassDelegate
{
    public delegate int Calcul(int x, int y);

    public int Addition(int x, int y)
    {
        return x + y;
    }

    public int Multiplication(int x, int y)
    {
        return x * y;
    }
}

public class ClassTest
{
    public void UseDelegate()
    {
        ClassDelegate c = new ClassDelegate();

        ClassDelegate.Calcul calcul = new ClassDelegate.Calcul(c.Addition);
        Console.WriteLine(calcul(5, 4));

        calcul = new ClassDelegate.Calcul(c.Multiplication);
        Console.WriteLine(calcul(5, 4));
    }
}
```

Exemple de delegate

Les delegates d’événements

- Les événements sont un mécanisme du C# permettant à une classe d’être notifiée d’un changement.
- La base des événements est le délégué. On pourra stocker dans un événement un ou plusieurs délégués qui pointent vers des méthodes respectant la signature de l’événement.
- Un événement est défini grâce au mot-clé : event.

Les delegates d’événements

```
public class Meteo
{
    public delegate void DelegateDeChangementDeTemperature(int nouvelleTemperature);
    public event DelegateDeChangementDeTemperature ChangementDeTemperature;

    4 references
    public int Temperature { get; set; }

    0 references
    public void FaitFroid()
    {
        Temperature--;

        if (ChangementDeTemperature != null)
        {
            ChangementDeTemperature(Temperature);
        }

    }

    0 references
    public void FaitChaud()
    {
        Temperature++;

        ChangementDeTemperature?.Invoke(Temperature);

    }
}
```

```
public void Test()
{
    Meteo meteo = new Meteo { Temperature = 20 };

    meteo.ChangementDeTemperature += Meteo_ChangementDeTemperature;
    FaitChaud();
    meteo.ChangementDeTemperature -= Meteo_ChangementDeTemperature;
}

2 references
private void Meteo_ChangementDeTemperature(int nouvelleTemperature)
{
    Console.WriteLine("La nouvelle température est : " + nouvelleTemperature);
}
```

Méthodes d’extensions

- Ajout de fonctionnalités à un type sans le dériver ou le compiler.
- Utilisation du mot clé « this » en paramètre de fonction.
- Doit être définie dans une classe statique du projet.

Méthodes d’extensions

```
public static class Class1
{
    public static bool IsMultiple2(this System.Int32 i)
    {
        return (i % 2) == 0;
    }
}
```

```
if (15.IsMultiple2())
{
    Console.WriteLine("15 is a multiple of 2");
}
else
{
    Console.WriteLine("15 isn't a multiple of 2");
}
```

Exemple

Types anonymes

- Définition d'une variable dont le type n'est connu qu'à l'exécution.
- Respecte le principe du typage fort C#.
  - La définition est déférée, mais une fois le type déterminé par le compilateur, il ne peut plus changer.
- Possibilité de déclaration de types complexes.
- Mot clé « var ».

Types anonymes

```
var t = new[] { 1, 2, 3 };  // t sera considéré comme un Tableau d'Int
var d = 2.7;                // d sera considéré comme un double

// Déclaration de types complexes anonymes.
var v = new { Amount = 100, Message = "Salam" };
Console.WriteLine(v.Amount + " " + v.Message);

// Une fois le type détérminé, on ne peut plus changer
var s = "Salam";

s = 100;
```

```
struct System.Int32
Represents a 32-bit signed integer.

Error:
   Cannot implicitly convert type 'int' to 'string'
```

Exemples

Expressions Lambda

- Une expression Lambda est une fonction ne possédant pas de nom permettant de calculer et retourner une valeur unique.
- Mot clé : « => » qui signifie « Goes to »

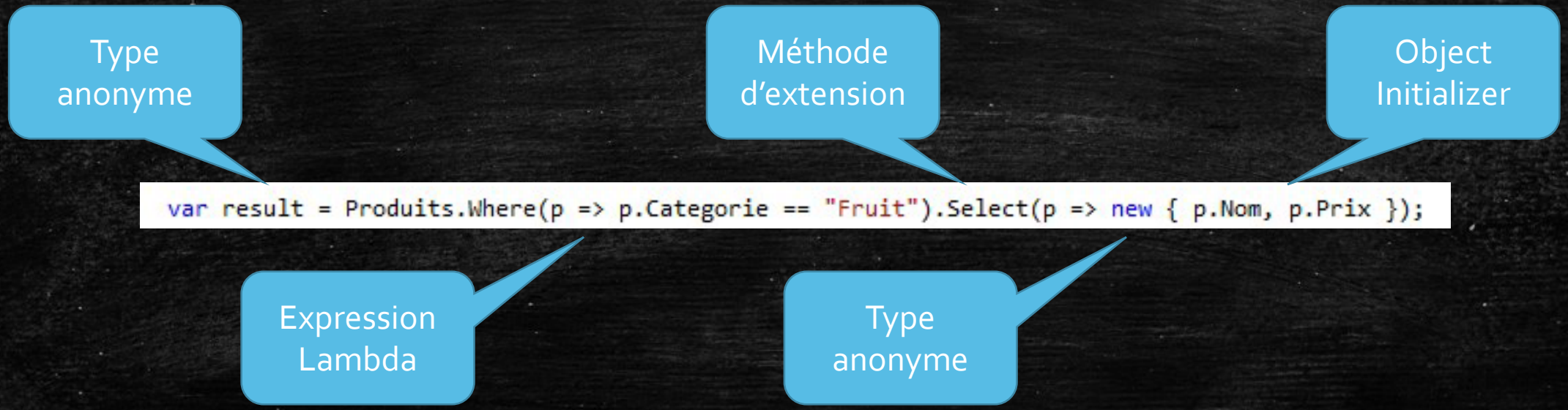
```
delegate int d(int x);

public void TestMethod()
{
    d val1 = k => k + 1;
    int r1 = val1(5);   // retourne 5 + 1 = 6

    d val2 = k => k * 2;
    int r2 = val2(5);   // retourne 5 * 2 = 10

    d val3 = k => (k.IsMultiple2()) ? 0 : 1;
    int r3 = val3(5);   // retourne 1 car 5 n'est pas multiple de 2
}
```

Linq

- Mécanisme de « requêtage » sur les objets.
  - Philosophie ensembliste « SQL ».
- Offre la possibilité de faire des requêtes sur des sources différentes :
  - Linq to Objects.
  - Linq to XML.
  - Linq to SQL
  - …

Linq

- S'appuie sur des briques du langage :
  - Initialiseur.
  - Méthodes d'extension.
  - Types anonymes.
  - Lambda expressions.



Linq to Objects

- LINQ to Objects fait référence à l'utilisation directe de requêtes LINQ avec n'importe quelle collection.
- LINQ peut interroger toutes les collections énumérables telles que :
  - List<T>
  - Array
  - ObservableCollection<T>
  - Dictionary<TKey, TValue>
  - …
- Auparavant, il faut écrire des boucles foreach complexes pour spécifier comment récupérer des données d'une collection.
- Avec LINQ, il suffit d’écrire du code déclaratif qui décrit ce que vous souhaitez récupérer.

Linq to Objects

- Avantage des requêtes LINQ par rapport aux boucles foreach traditionnelles :
  - Elles sont plus concises et lisibles, surtout lors du filtrage de plusieurs conditions.
  - Elles fournissent des fonctions puissantes de filtrage, de classement et de regroupement avec un minimum de code d'application.
  - Elles peuvent être appliquées à d'autres sources de données avec peu ou pas de changement.
- 2 façons de faire

Linq to Objects

1ère façon

```
var listProd1 = Produits.Where(p => p.Categorie == "Fruit").OrderBy(p => p.Prix).ToList();
```

2ème façon

```
var listProd2 = from p in Produits where p.Categorie == "Fruit" orderby p.Prix select p;
```

Linq to SQL

- LINQ to SQL est une implémentation de O/RM (object relational mapping) incluse dans le .NET Framework.
- Permet la modélisation d'une base de données relationnelle avec des classes .NET.
- Il est possible de récupérer des données d'une base en utilisant LINQ, mais également mettre à jour, insérer et supprimer des données dans celle-ci.
- Il prend totalement en charge les transactions, les vues et les procédures stockées

Linq to SQL

```
NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
```

Exemple de recherche avec Linq to SQL

Linq to XML

- LINQ to XML est une interface de programmation XML en mémoire compatible avec LINQ.
- Il permet de travailler avec du code XML dans les langages de programmation .NET Framework.
- Le principal avantage de LINQ to XML est son intégration avec LINQ.
  - Il permet d'écrire des requêtes sur le document XML en mémoire afin de récupérer des collections d'éléments et d'attributs.
  - Il est comparable en terme de fonctionnalités (mais pas en termes de syntaxe) à XQuery et Xpath.
- LINQ to XML s'apparente au modèle DOM :
  - Place le document XML en mémoire pour permettre son interrogation et sa modification avant de l'enregistrer ou le sérialiser.
  - Toutefois il procure un nouveau modèle objet qui est plus léger et plus facile à manipuler.

Linq to XML

```xml
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
```

Un fichier XML

Linq to XML

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
```

Chargement d'un fichier XML

Linq to XML

```
IEnumerable<string> partNos =
    from item in purchaseOrder.Descendants("Item")
    select (string) item.Attribute("PartNumber");
```

Exemple d’obtention de la valeur d'attribut de numéro de référence pour chaque élément de la commande

Linq to XML

```
IEnumerable<XElement> partNos =
    from item in purchaseOrder.Descendants("Item")
    where (int) item.Element("Quantity") *
        (decimal) item.Element("USPrice") > 100
    orderby (string)item.Element("PartNumber")
    select item;
```

Exemple de génération d'une liste des éléments avec une valeur supérieure à $ 100, triés par numéro de référence avec Linq to XML

Types nullables

- Peuvent représenter toutes les valeurs d'un type sous-jacent, et une valeur « null » supplémentaire.
- Pour rendre un type nullable : « ? ».
- Mots clés : « HasValue » et « Value ».

Types nullables

```
int? x = 10;

if (x.HasValue)
{
    Console.WriteLine(x.Value);
}

int? a = 10, b = null;

a = a * 10;      // a = 100 (Multiplication par 10)
a = a + b;       // a = null (Ajout de b qui est nulle)

DateTime date1 = DateTime.Now;
DateTime? date2;

if (date1 != DateTime.MinValue)
{
    date2 = date1;
}
else
{
    date2 = null;
}
```

Exemples

Implémentation de la méthode Dispose

- Les objets peuvent implémenter le pattern « Dispose ».
  - Par l'implémentation de l'interface « IDisposable » qui contient une seule méthode « void Dispose() ».
  - Le « Dispose » permet aux objets de libérer des ressources sans attendre le passage du « Garbage Collector ».
- Mot clé « using » :
  - Appel automatique de la méthode Dispose.

Implémentation de la méthode Dispose

```
public class TestClass : IDisposable
{
    // Propriétés et méthodes et code de la classe

    // Implémentation de la méthode Dispose
    public void Dispose()
    {
        // Si la connection est ouverte :
        SqlCon.Close();
    }
}
```

```
TestClass test = new TestClass();

// Traitement et utilisation de la classe

// Appel de la méthode Dispose
test.Dispose();
```

Exemple de la méthode « Dispose ».

Implémentation de la méthode Dispose

```
public void ModifierEtudiant(Etudiant e)
{
    using (SqlCommand cmd = new SqlCommand())
    {
        this.SqlCon.Open();
        cmd.Connection = this.SqlCon;
        cmd.CommandType = System.Data.CommandType.Text;
        cmd.CommandText = @"Update Etudiant Set Nom=@Nom,
                            Prenom=@Prenom, Ville=@Ville Where Id=@Id";
        cmd.Parameters.Add(new SqlParameter("Id", e.Id));
        cmd.Parameters.Add(new SqlParameter("Nom", e.Nom));
        cmd.Parameters.Add(new SqlParameter("Prenom", e.Prenom));
        cmd.Parameters.Add(new SqlParameter("Ville", e.Ville));

        cmd.ExecuteNonQuery();

        this.SqlCon.Close();
    }
}
```

Exemple du mot clé « using »