Formation C# Intermédiaire
Formateur : Mohamed DERKAOUI (mderkaoui@dawan.fr)

Notes : https://mensuel.framapad.org/p/cs-interm

Support de cours :
C# : http://demo.dawan.biz/mohamed/cs.pdf
Entity Framework : http://demo.dawan.biz/mohamed/ef.pdf
C# Intermédiaire : http://demo.dawan.biz/mohamed/CS_interm.pdf
Design Patterns : http://demo.dawan.biz/mohamed/DesignPatterns.pdf

---

Exemples du cours : http://demo.dawan.biz/mohamed/TrainingInterm2.zip

---

Fixer les projets d'ici le 23 novembre

A revoir _____
- EF : mapping
- Linq
- Tests unitaires

Facteurs de visibilité :

- public :
* une classe publique est accessible partout
* une variable publique est accessible partout

- protected :
* une classe ne peut être protected
* un membre (champ/méthode) protégé est accessible dans la classe et les sous-classes du même package

- private :
* une classe ne peut être privée
* un membre privé n'est accessible que dans la classe où il est déclaré

- default (ne rien mettre) :
* une classe sera accessible dans le même namespace ou les sous-namespaces
* un membre est accessible uniquement dans la classe où il est déclaré

Sérialisation :

Pouvoir stocker l'état d'un objet (en mémoire) dans un support persistant
Il existe plusieurs types de sérialisation :

- **binaire** : BinaryFormatter (la plus rapide)
  la classe doit être décorée avec l'attribut [Serializable]

- **xml** : XmlSerializer

- **Json** : DataContractJsonSerializer

• la classe doit être décorée avec l'attribut [DataContract]

Généricité :

Pouvoir créer des classes/méthodes/interfaces indépendantes du type d'objets à utiliser

---

Réflexion :

Pouvoir introspecter un objet en accédant à son type
Intérêt : faire des opérations sur l'objet ou accéder à son type dynamiquement

---

Révision POO :

- **objet** : élément identifiable (abstrait ou concret)
- **classe** : définition d'un type d'objets
- **classe abstraite** : modèle pour les sous-classes
- **interface** : contrat que les classes doivent respecter

Comparaison des objets :
- **Equals(object obj)** de la classe Object
- **CompareTo(T obj)** de l'interface IComparable<T>
- **Compare(T obj1, T obj2)** de l'interface IComparer<T>

Arguments variables : on ajoute params au paramètre et on l'utilise comme un tableau
Un seul argument variable par méthode et il doit être positionné à la fin de la liste des paramètres de la méthode :

```
//Sum(tab1);
//Sum(new int[]{2,4,5});
//Sum(3,6);
public static int Sum(params int[] tab)
{
int res = 0;
```

```
foreach (int x in tab)
res += x;

return res;
}
```

Expression Lambda :

fonction anonyme (implémentation d'une interface fonctionnelle)
Intérêt : simplifier la syntaxe

Toute expression lambda peut être convertie en type délégué.

Un **delegate** est un type référence qui peut être utilisé pour encapsuler une méthode anonyme (comme un pointeur fonction)
En .NET : System.Action et System.Func sont des définitions génériques pour des délégués.

Héritage :

Objectif : Etendre une classe existante afin de la réutiliser
- Une classe fille hérite d'une seule classe mère
- Une classe fille hérite des membres (champs/méthodes) de la classe mère mais ne peut accéder qu'aux champs public ou protected.
- Une classe fille n'hérite pas des constructeurs de la classe mère mais peut y faire appel en utilisant le mot-clé "base"
- Une classe fille peut redéfinir (override) une méthode héritée à condition qu'elle soit marquée "virtual" dans la classe mère

---

Polymorphisme :

Un objet peut prendre plusieurs formes
On peut le typer :
- avec une classe
- avec une interface

Intérêts :
- manipuler un objet sans se soucier de son type
- mélanger des objets de différentes natures

Envoi d'un email en .NET

Namespaces : System.Net, System.Net.Mail
Classes : MailMessage, SmtpClient
- On doit créer un objet de type MailMessage puis utiliser un serveur smtp pour pouvoir envoyer l'email

---

Design Pattern (Patron de conception) :

Schéma de classes représentant une solution à un problème donné
Il existe 23 design patterns répertoriés par le **GangOfFour** classés en :
- patterns de **création :** comment instancier des objets ?
- patterns de **structure :** comment assembler des objets ?
- patterns de **comportement :** comment organiser la communication inter-objets ?

Références :
https://refactoring.guru/
https://www.dofactory.com/net/design-patterns
https://sourcemaking.com/design_patterns

Exemple :
Pattern de création - **Singleton**
Problème : garantir une instanciation unique de la classe

Procédé :
Mettre une variable statique
Rendre le constructeur privé pour interdire l'instanciation directe
Ajouter une méthode ou une propriété statique pour pouvoir contrôler les instances.

Design pattern Observer

Pattern de comportement
Problème : s'abonner à des événements et recevoir des notifications
Exemple : être notifié lors du changement de prix d'un produit
=> Schéma de la solution

la BCL (framework .NET) propose des interfaces pour l'implémentation du pattern Observer/
https://miro.medium.com/max/1235/1*BdKQ2aD5MfCV0ZkjWj-SXg.png

---

Value objects : objets valeurs

détenants du métier mais ne contiennent pas d'identifiants uniques
la comparaison d'objets valeur inclut l'ensemble des champs

Exercice :
Company possède un nom et une géolocalisation
la géolocalisation possède longitude et latitude
on peut comparer 2 sociétés :
- par nom
- par géolocalisation

Créer plusieurs sociétés
Trier par nom alpha
Trier par geolocalisation

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace project2Library.EntitiesVsValueObjects
{
//ValueObject : objet immutable (renseigné à la création), valeur ne
bouge pas
public class Geolocation : IComparable<Geolocation>
{
private double _longitude;
private double _latitude;

public Geolocation(double longitude, double latitude)
{
_latitude = latitude;
_longitude = longitude;
}

public override bool Equals(object obj)
{
return obj is Geolocation geolocation &&
_longitude == geolocation._longitude &&
_latitude == geolocation._latitude;
}
```

```
public override string ToString()
{
return $"{_longitude},{_latitude}";
}

public override int GetHashCode()
{
return base.GetHashCode();
}

public int CompareTo(Geolocation other)
{
//comparaison basée sur la distance
double earthRadius = 6371; //en km => miles = 3958.75
double dLat = ToRadians(other._latitude - _latitude);
double dLng = ToRadians(other._longitude - _longitude);

double sindLat = Math.Asin(dLat / 2);
double sindLng = Math.Asin(dLng / 2);

double a = Math.Pow(sindLat, 2) + Math.Pow(sindLng, 2)
* Math.Cos(ToRadians(_latitude)) *
```

- `Math.Cos(ToRadians(other._latitude));`

```
double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
double dist = earthRadius * c;
return Convert.ToInt32(dist);
}

private static double ToRadians(double angle)
{
return (Math.PI / 180) * angle;
}
}
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace project2Library.EntitiesVsValueObjects
```

```
{
public class Company : IComparable<Company>
{
public int? Id { get; set; }
public string Name { get; set; }
public Geolocation Location { get; set; }

public Company(int? id, string name, Geolocation location)
{
Id = id;
Name = name;
Location = location;
}

public int CompareTo(Company c)
{
return Name.CompareTo(c.Name);
}

public int CompareByLocation(Company c)
{
return Location.CompareTo(c.Location);
}

public override bool Equals(object obj)
{
return obj is Company company &&
Id == company.Id;
}

public override int GetHashCode()
{
return 2108858624 + Id.GetHashCode();
}
}
}
```

---

http://demo.dawan.biz/mohamed/Example2.zip

BDD (Behaviour Driven Development)

- **Evolution du TDD (Test Driven Development)**

User Story : En tant qu'utilisateur, je dois saisi login/pwd afin de me connecter à mon espace client

Du point de vue du code :

```
(c) UserService
bool checkLogin(username,pwd)
void checkLoginTest()
{ Assert.Equals(true,user.checkLogin("toto","t"))};
```

On demande au client de nous donner des situations de tests écrit en langage Gujerkin.
Un test (scénario, utilisateur,...) sous forme textuelle (**langage Gujerkin**) --------(un outil)--------> Test unitaire
Exemple :

GIVEN username="toto" AND pwd "t"
WHEN checklogin
THEN url="/adminSpace"

L'outil (ex. SpecFlow ou autre...) le transforme en code de test unitaire.
Ce scénario est à mettre dans un fichier .feature, et SpecFlow s'occupe de créer les classes de test.

NuGET _____
Gestionnaire de packages permettant d'administrer les DLLs ajoutées dans un projet
Il nous évite de télécharger manuellement les dlls et de les référencer nous même
Il maintient un fichier : packages.config pour référencer la liste des Dlls ajoutées

_____

```
//Cas1 _____
    //CalculatorDataService sd = new CalculatorDataService(new UserRepository());
    //sd.Compute(1)

    //CalculatorDataService sd2 = new CalculatorDataService(CsvRepository());
    //sd2.Compute(1)

    ////Cas2 _____
    //CalculatorDataService sd1 = new CalculatorDataService();
    //sd2.Compute(1, new UserRepository());
    //sd2.Compute(2,new CsvRepository());


    ////Cas3
    //CalculatorDataService sd = new CalculatorDataService();
    //sd.Repo = new UserRepository();
    //sd.Compute(....)

    //sd.Repo = new CsvRepository();
    //sd.Compute(...)
```

_____

Gestion des logs en .NET

Objectif : écrire des traces qui vont servir :

- pour avoir un historique des actions de l'utilisateur/appli
- pour avoir un historique des erreurs

<u>Deux possibilités</u> :

1- utiliser l'API native (biblio intégrée dans System.Diagnostics)

elle ne propose que l'écriture dans un fichier ou dans l'observateur d'évènements Windows

Namespace : System.Diagnostics

Classes : TextWriterTraceListener, EventLogTraceListener

Procédé :

- Ajout d'une configuration dans App.Config ou Web.Config :

```
<system.diagnostics>
<trace autoflush="true" indentsize="4">
<listeners>
<remove name="Default" />
<add name="myListener1"
type="System.Diagnostics.TextWriterTraceListener"
traceOutputOptions="DateTime"
initializeData="project1.log" />
</listeners>
</trace>
</system.diagnostics>
```

- Utilisation de la classe Trace ou Debug pour l'écriture des logs :

Trace.TraceInformation("ceci est mon premier log");

```
try
{
throw new Exception("Mon erreur");
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
Trace.TraceError(ex.StackTrace);
}
```

2- Utiliser un API externe : Log4Net, NLog,...

Apports : multiples supports de logs.

On pourra :

- écrire dans la console
- écrire dans un fichier (avec une rotation sur la date ou la taille ou les 2)
- envoyer le log par email
- écrire en bdd
- ...

Vocabulaire :

Logger : objet permettant d'écrire les logs

Appender : support de logs (ConsoleAppender, FileAppender, SmtpAppender)

Layout : Type de logs (text, xml)
Pattern : format du message (date nomClasse typeLog message saut de ligne)
Level : niveau du message (INFO, WARN, ERROR, ...)

Procédé :
- ajout de la bibliothèque Log4Net (utiliser NuGet pour l'utiliser)
- ajout d'une configuration dans le fichier App.Config :
  la section de configuration log4net avec la configuration des appenders
  la liste complète des appenders :
  https://logging.apache.org/log4net/release/config-examples.html

-Instanciation d'un logger
- Ecriture des traces :
  //Utilisation de Log4Net : il faut ajouter la dll + la config dans App.Config
  //Il faut charger la config Xml avant d'écrire des logs

```
XmlConfigurator.Configure();

ILog myRootLogger = LogManager.GetLogger(typeof(Program)); //root logger
myRootLogger.Info("acces 1");

ILog myErrorLogger = LogManager.GetLogger("myErrorLogger");
//myErrorLogger
myErrorLogger.Error("il y a eu un crash");
```

Design Patterns Factory :

Pattern de création permettant de créer une usine pour créer des objets
L'appelant a un point d'entrée unique (une méthode statique permettant de créer le bon type en fonction d'un paramètre)

Design Pattern Builder :

pattern de création pour construire progressivement un objet complexe
(voir implémentation)

Dupliquer un objet (cloner) :

On peut implémenter l'interface ICloneable
Object possède déjà une implémentation avec une méthode MemberWiseClone
qui duplique les champs de premier niveau (non objet)
On doit redéfinir la méthode Clone() dans les classes métier pour définir l'objet à retourner et gérer le pb de références

(voir exemple de code)

```
#region DESIGN PATTERNs -  PROTOTYPE
        Console.WriteLine("PROTOTYPE / CLONEABLE _____");

        //Copie d'un objet simple
        LineArticle art1 = new LineArticle() { Description = "Table", Price = 200 };
        LineArticle art2 = (LineArticle) art1.Clone();
        Console.WriteLine("art2 : " + art2);

        art1.Description = "chaise";
        Console.WriteLine("art1 : " + art1);
        Console.WriteLine("art2 : " + art2);

        //Copie d'un objet complexe
        CartLine lineC1 = new CartLine() { Article = art1, Qty = 3 };
        CartLine lineC2 = (CartLine)lineC1.Clone();
        Console.WriteLine("lineC1 : " + lineC1);
        Console.WriteLine("lineC2 : " + lineC2);
        Console.WriteLine("-----");

        lineC1.Article.Description = "Voiture";
        Console.WriteLine("lineC1 : " + lineC1);
        Console.WriteLine("lineC2 : " + lineC2);

        Console.WriteLine("-----");
        Cart myCart1 = new Cart();
        myCart1.Date = DateTime.Now;
        myCart1.Lines = new List<CartLine> { lineC1, lineC2 };

        Cart myCart2 = (Cart)myCart1.Clone();

        myCart1.Lines[0].Article.Description = "TOTO";
        Console.WriteLine("----");

        #endregion
```

http://demo.dawan.biz/mohamed/Prototype.zip

FastMapper : convertir rapidement un type vers un autre

- - soit un utilisant une convention de nommage
- - soit en faisant la configuration manuellement
    - `fastmapperutility.Map(dest => dest.propMapsTo,`
        - `src =>src.propToMap))`

•

<add name="MyDbContext"
connectionString="Server=JMOH-PC;Database=bdds1;Trusted_Connection=True;MultipleActiveResultSets=True;App=EntityFramework" providerName="System.Data.SqlClient" />