TP ADO.NET Avancé

Le but du TP est de créer une petite application qui va nous permettre de travaillé en mode déconnecté en chargeant en mémoire et dans des collections les enregistrements de notre BD.

On le fera évoluer de sorte que la source de données soit totalement indépendante de l'application.

Le principal problème avec notre façon de travailler actuelle est que l'on garde en permanence la connexion ouverte. En effet dans le cas d'applications client lourd, on a tendance à ouvrir une connexion sur la BD au démarrage de l'application et à la fermer en sortant de l'application.
Or dans l'utilisation de l'application, on s'en sert finalement peu. On consomme donc des ressources inutilement et cela peut avoir aussi un impact budgétaire si on paye le nombre de connexions simultanées nécessaires sur le SGBD (exemple Oracle). D'où l'intérêt de travailler en mode déconnecté en suivant le principe suivant :

On ouvre la connexion au plus tard et on la ferme au plus tôt

Préparation du travail :

Vous aurez un répertoire racine et le premier projet que vous allez créer se trouvera dans le dossier Exercice1. Il aura pour nom ReaderObject. Ce nom sera identique pour tous les exercices.

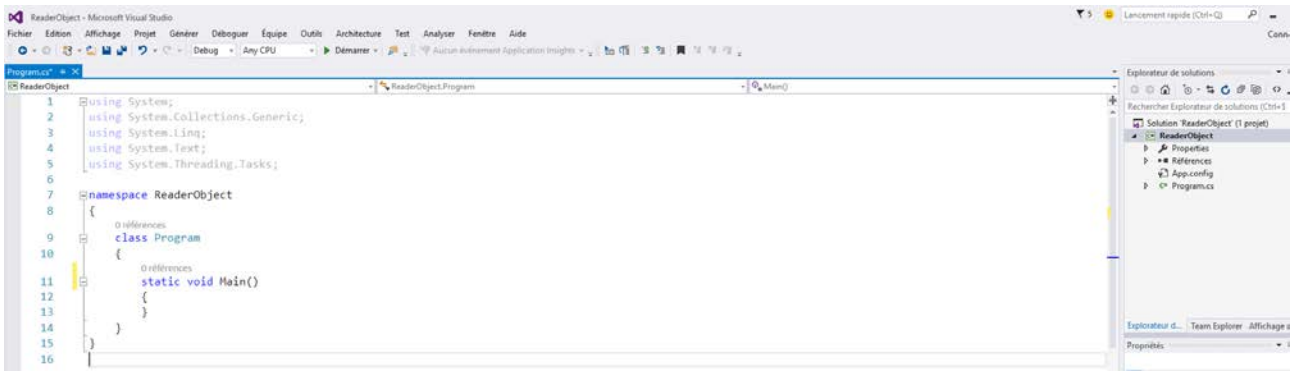
Ce TP étant progressif, à la fin de l'exercice 1, vous ferez une copie du dossier que vous appellerez Exercice2 et ainsi de suite. Le point de départ de l'exercice n sera donc le projet dans l'état de la fin de l'exercice n-1.

Exercice 1 : Mise en place de l'application et chargement d'une collection d'objets

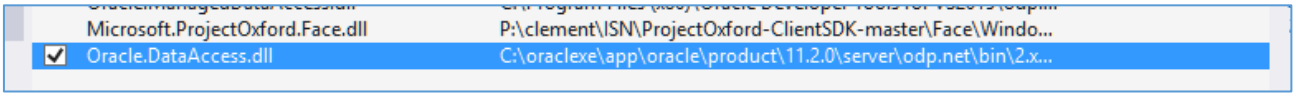
nous allons :

- ✓ Créer un projet console appelé ReaderObject
- ✓ Créer une connexion vers une BD oracle
- ✓ Lire les données
- ✓ Remplir une collection en mémoire d'objets calquant notre table
- ✓ Fermer la connexion
- ✓ Afficher ces données sur la console.

Création du projet
Créer un projet console appelé ReaderObject



Rajouter une référence sur l'API Oracle :



Référencer l'espace de noms Oracle dans le fichier program.cs

```
using Oracle.DataAccess.Client;
```

Rajouter une classe Employe :

```
class Employe
{

    2 références
    public Int16 Numemp { get; set; }
    2 références
    public String Nomemp { get; set; }
    2 références
    public String Prenomemp { get; set; }
    2 références
    public String Poste { get; set; }
    2 références
    public Single Salaire { get; set; }
    3 références
    public Single? Prime { get; set; }
    2 références
    public String CodeProjet { get; set; }
    3 références
    public Int16? Superieur { get; set; }


    0 références
    public override string ToString()
    {
        return Numemp + " - " + Nomemp + " - " + Prenomemp + " - " + Poste + " - " + Salaire + " - " + Prime + " - " + CodeProjet + " - " + Superieur;
    }
}
```

Vous remarquerez que l'on a redéfini la méthode ToString()

Remarquez la déclaration de la property Prime et de la property Superieur. Pourquoi le point d'interrogation.

Il vous reste maintenant à écrire le code de la méthode Main() qui

- ✓ Déclare une collection d'objets de la classe Employe,
- ✓ Récupère les employés de la table Employé dans un objet DataReader,

- ✓ Remplit la collection d'objets Employe à partir de ce DataReader,
- ✓ Ferme la connexion vers la base de données
- ✓ Parcourt et affiche les employés contenus dans la collection

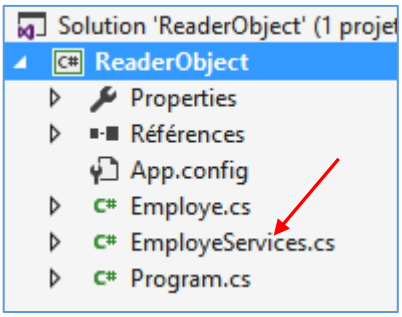
Résultat attendu :

```
1 - DUPONT - Pierre - A22 - 10675,28 - 1000 - PR1  - 4
2 - JOLIBOIS - Rolland - A12 - 13400,95 - 1500 - PR2  - 5
3 - BEAUMONT - Jean - A25 - 15780,41 - 2000 - PR1  - 6
4 - DUCHATEL - Mireille - B12 - 14677,24 - 3000 - PR2  - 7
5 - MARTIN - Robert - B21 - 16591,67 - 0 - PR3  - 7
6 - MAZAUD - Patricia - B14 - 16591,67 - 1500 - PR3  - 8
7 - GIMOND - Antoine - C17 - 26801,92 -  -  - 15
8 - SAULT - Jean - C23 - 27057,17 -  -  - 15
9 - GALLI - Jean Daniel - A25 - 12762,81 - 2000 - PR2  - 5
10 - JACONO - Marie - B14 - 14677,24 - 1500 - PR5  - 20
11 - ANTHONY - Henri - A15 - 12124,68 - 0 - PR2  - 10
12 - CANE - Michel - B15 - 14248,87 - 1400 - PR1  - 8
13 - GOMEZ - Joseph - A12 - 9572,12 - 2000 - PR3  - 12
14 - RIVIERE - Maurice - C18 - 29354,48 -  -  - 15
15 - RUSSOT - Eric - C23 - 38288,45 -  -  -
16 - BERNARDI - Patrick - A14 - 12507,57 - 2500 - PR2  - 18
17 - BEUGNIES - Maurice - B12 - 14248,87 - 2400 - PR1  - 14
18 - FARNY - Daniel - B15 - 14677,24 - 1700 - PR5  - 8
19 - ESTIVAL - Sophie - B17 - 15315,38 - 1000 - PR4  - 8
20 - LUNEAU - Henri - C17 - 28716,33 -  -  - 15
21 - BRESSON - Pierre - C23 - 17867,94 -  -  - 15
```

Exercice 2 : Création d'une classe de services

Cette façon de faire est correcte mais peut être largement améliorée

L'idée est de séparer les classes :
garder la classe Employe pour stocker les données rapatriées de la BD
Créer une nouvelle classe qui implémentera les méthodes relatives aux traitements des objets de la classe Employe. On appellera cette classe EmployeServices



Cette classe va comporter pour l'instant 2 méthodes :

- ✓ ***HydrateEmploye*** qui accepte en paramètre un DataReader et qui retourne un objet de la classe Employe.

- ✓ ***FindAllEmployes*** qui n'accepte pas de paramètre et qui retourne la collection d'employés correspondant aux employés ramenés depuis la BD.

```
public static Employe HydrateEmploye(OracleDataReader readerEmploye)
{
    Employe employe = new Employe();
    employe.Numemp = Convert.ToInt16(readerEmploye["NUMEMP"]);
    employe.Nomemp = readerEmploye["NOMEMP"] as String;
    employe.Prenomemp = readerEmploye["PRENOMEMP"] as String;
    employe.Poste = readerEmploye["POSTE"] as String;
    employe.Salaire = Convert.ToSingle(readerEmploye["SALAIRE"]);
    if (readerEmploye["Prime"] == DBNull.Value)
    {
        employe.Prime = null;
    }
    else
    {
        employe.Prime = Convert.ToSingle(readerEmploye["Prime"]);
    }

    employe.CodeProjet = readerEmploye["CODEPROJET"] as String;
    if (readerEmploye["SUPERIEUR"] == DBNull.Value)
    {
        employe.Superieur = null;
    }
    else
    {
        employe.Superieur = Convert.ToInt16(readerEmploye["SUPERIEUR"]);
    }
    return employe;
}
```

```
public List<Employe> FindAllEmployes()
{
    List<Employe> employes = new List<Employe>();
    string ch = "Data Source=(DESCRIPTION =(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))(CONNECT_DATA =(SERVER = DEDICATE
    using (OracleConnection CnOracle = new OracleConnection(ch))
    {
        try
        {
            using (OracleCommand cmdTousLeslEmployes = new OracleCommand("select * from employe", CnOracle))
            {
                CnOracle.Open();
                OracleDataReader readerEmploye = cmdTousLeslEmployes.ExecuteReader();
                while (readerEmploye.Read())
                {
                    Employe employe = HydrateEmploye(readerEmploye);
                    employes.Add(employe);


                }
                readerEmploye.Close();

            }
            CnOracle.Close();
        }
        catch (OracleException ex)
        {
            Console.WriteLine(ex.Message);
        }
        return employes;

    }
}
```

Nous allons maintenant, sur le même principe créer la méthode FindEmployeById qui accepte en paramètre la valeur du numemp de l'employe.

Nous réutiliserons la méthode HydrateEmploye.

```
public Employe FindEmployeById(Int16 id)
{
    Employe employe = null;
    string ch = "Data Source=(DESCRIPTION =(ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))(CONNECT_DATA =(SERVER = DEDIC
    using (OracleConnection CnOracle = new OracleConnection(ch))
    {
        try
        {
            using (OracleCommand cmdTousLeslEmployes = new OracleCommand("select * from employe where numemp= :numemp", CnOracle))
            {
                OracleParameter pId = new OracleParameter("numemp", OracleDbType.Int16, System.Data.ParameterDirection.Input);
                pId.Value = id;
                cmdTousLeslEmployes.Parameters.Add(pId);
                CnOracle.Open();
                OracleDataReader readerEmploye = cmdTousLeslEmployes.ExecuteReader();
                if (readerEmploye.Read())
                {
                    employe = HydrateEmploye(readerEmploye);
                }
                readerEmploye.Close();

            }
            CnOracle.Close();
        }
        catch (OracleException ex)
        {
            Console.WriteLine(ex.Message);
        }
        return employe;
    }
}
```

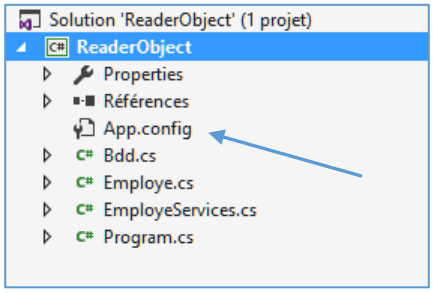
On remarque qu'une partie du code est identique dans les méthodes

Exercice 3 : Factorisation du code.

L'idée est de ne pas avoir à recopier plusieurs fois le même bout de code, ce qui facilitera la mise au point et la maintenance de l'application.

On va donc commencer par utiliser le fichier de configuration, externe à l'application. Il s'agit du fichier App.config que l'on voit dans l'explorateur de solutions.

Ce fichier est au format XML et sa grammaire propose un certain nombre de noms d'éléments prédéfinis.
Exemple : `<connectionStrings>`



On va commencer par mettre la chaine de connexion dans le fichier de configuration.
Assurez-vous que votre fichier de configuration App.config est bien dans votre solution.
S'il n'y est pas, il faut le créer.

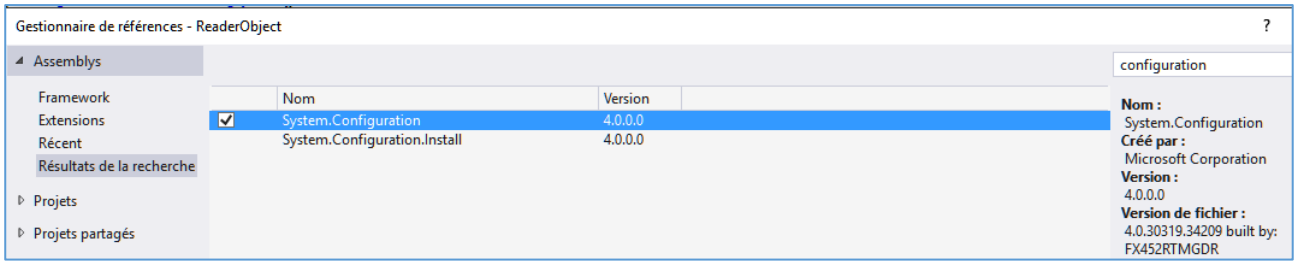
Vous utiliserez

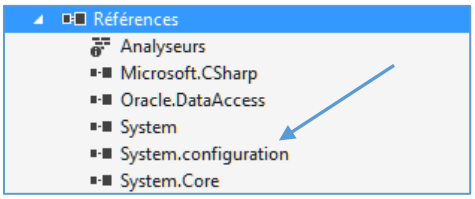
- ✓ l'élément prédéfini <connectionStrings> </connectionStrings> pour entrer la chaine de connexion
- ✓ l'élément <appSettings> pour rajouter des paramètres :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
    </startup>
  <connectionStrings>
    <add name="oracle" providerName="Oracle.DataAccess.Client"
        connectionString="Data Source=(DESCRIPTION =(ADDRESS = (PROTOCOL = TCP)(HOST = {0})(PORT = {1}))
        (CONNECT_DATA =(SERVER = DEDICATED)(SERVICE_NAME = {2})));User Id={3};Password={4};"
         />
  </connectionStrings>
  <appSettings>
    <add key="SERVEUR" value="localhost"/>
    <add key="PORT" value="1521"/>
    <add key="SID" value="XE"/>
    <add key="USERID" value="coursado"/>
    <add key="PWD" value="coursado"/>
  </appSettings>
</configuration>
```

Vous adapterez les valeurs des paramètres à votre configuration.

Vous poserez ensuite une référence sur l'assembly System.Configuration





On va Ensuite modifier le code de la connexion pour la méthode FindAllEmployes() :

```
public List<Employe> FindAllEmployes()
{
    List<Employe> employes = new List<Employe>();
    string ch = String.Format(ConfigurationManager.ConnectionStrings["oracle"].ToString(), ConfigurationManager.AppSettings["SERVEUR"],
                                                                                            ConfigurationManager.AppSettings["PORT"],
                                                                                            ConfigurationManager.AppSettings["SID"],
                                                                                            ConfigurationManager.AppSettings["USERID"],
                                                                                            ConfigurationManager.AppSettings["PWD"]
                                                                                            );
  using (OracleConnection CnOracle = new OracleConnection(ch))
    {
```

Vous pouvez tester maintenant.

Cette méthode est intéressante, la chaine de connexion et les paramètres sont extérieurs à l'application, toutefois, on se rend compte que l'on va dupliquer ce code dans chaque méthode accédant à la base de données.

Pour remédier à ceci, on va créer une classe Bdd qui va stocker une instance de l'objet OracleConnection.

```
class Bdd
{
    2 références
    public OracleConnection CnOracle { get; set; }

    1 référence
    public Bdd()
    {
        string ch = String.Format(ConfigurationManager.ConnectionStrings["oracle"].ToString(), ConfigurationManager.AppSettings["SERVEUR"]
                                                                                                ConfigurationManager.AppSettings["PORT"],
                                                                                                ConfigurationManager.AppSettings["SID"],
                                                                                                ConfigurationManager.AppSettings["USERID"],
                                                                                                ConfigurationManager.AppSettings["PWD"]
                                                                                                );
        CnOracle = new OracleConnection(ch);
    }
}
```

Il faut bien entendu modifier le code des classes qui ont des méthodes nécessitant une connexion à la BD

Et notamment pour nous la classe EmployeServices.

On crée un attribut privé de type OracleConnection et un constructeur dont l'objectif est de renseigner cette propriété privée

```
class EmployeServices
{
    private OracleConnection cnOracle;
    1 référence
    public EmployeServices()
    {
        Bdd bdd = new Bdd();
        cnOracle = bdd.CnOracle;
    }
```

Mettons en œuvre cette modification dans la méthodeFindEmployeById() :

```
public Employe FindEmployeById(Int16 id)
{
    Employe employe = null;
     using (cnOracle)
    {
        try
        {
            using (OracleCommand cmdTousLeslEmployes = new OracleCommand("select * from employe where numemp= :numemp", cnOracle))
            {
                OracleParameter pId = new OracleParameter("numemp", OracleDbType.Int16, System.Data.ParameterDirection.Input);
                pId.Value = id;
                cmdTousLeslEmployes.Parameters.Add(pId);
                cnOracle.Open();
```

C'est déjà mieux, l'application entière va utiliser une classe mettant à disposition une connexion pour effectuer les requêtes.

Testez ici votre application et mettez-la au point. Vous devriez toujours obtenir le même résultat, la méthode Main() n'a pas été modifiée :

```
1 - DUPONT - Pierre - A22 - 10675,28 - 1000 - PR1  - 4
2 - JOLIBOIS - Rolland - A12 - 13400,95 - 1500 - PR2  - 5
3 - BEAUMONT - Jean - A25 - 15780,41 - 2000 - PR1  - 6
4 - DUCHATEL - Mireille - B12 - 14677,24 - 3000 - PR2  - 7
5 - MARTIN - Robert - B21 - 16591,67 - 0 - PR3  - 7
6 - MAZAUD - Patricia - B14 - 16591,67 - 1500 - PR3  - 8
7 - GIMOND - Antoine - C17 - 26801,92 -  -  - 15
8 - SAULT - Jean - C23 - 27057,17 -  -  - 15
9 - GALLI - Jean Daniel - A25 - 12762,81 - 2000 - PR2  - 5
10 - JACONO - Marie - B14 - 14677,24 - 1500 - PR5  - 20
11 - ANTHONY - Henri - A15 - 12124,68 - 0 - PR2  - 10
12 - CANE - Michel - B15 - 14248,87 - 1400 - PR1  - 8
13 - GOMEZ - Joseph - A12 - 9572,12 - 2000 - PR3  - 12
14 - RIVIERE - Maurice - C18 - 29354,48 -  -  - 15
15 - RUSSOT - Eric - C23 - 38288,45 -  -  -
16 - BERNARDI - Patrick - A14 - 12507,57 - 2500 - PR2  - 18
17 - BEUGNIES - Maurice - B12 - 14248,87 - 2400 - PR1  - 14
18 - FARNY - Daniel - B15 - 14677,24 - 1700 - PR5  - 8
19 - ESTIVAL - Sophie - B17 - 15315,38 - 1000 - PR4  - 8
20 - LUNEAU - Henri - C17 - 28716,33 -  -  - 15
21 - BRESSON - Pierre - C23 - 17867,94 -  -  - 15
------------------------------
4 - DUCHATEL - Mireille - B12 - 14677,24 - 3000 - PR2  - 7
```

Il y a toutefois un inconvénient :
on peut, si l'on n'est pas vigilant utiliser plusieurs objets de cette classe. C'est-à-dire que l'on pourrait instancier plusieurs objets OracleConnection, ce qui est inutile car cela consomme des ressources inutilement.

L'idée est serait d'utiliser toujours le même objet.

C'est ce que l'on verra dans l'exercice suivant.

Pour rappel :
La méthode Main() est celle-ci :

```
// parcours des employes
List<Employe> employes;
EmployeServices serviceEmploye = new EmployeServices();
employes = serviceEmploye.FindAllEmployes();
foreach (Employe employe in employes)
{
    Console.WriteLine(employe);
}
Console.WriteLine("------------------------------");
Employe unEmploye = serviceEmploye.FindEmployeById(4);
Console.WriteLine(unEmploye);
```

Exercice 4 : Design Pattern Singleton

Un design pattern (patron de conception) est une méthode de programmation visant à mettre en œuvre un certain nombre de bonnes pratiques visant à améliorer la qualité et la maintenabilité du code.
Il existe un certain nombre de design pattern dont certains sont simples à mettre en place.

Le Design Pattern que l'on va mettre en œuvre dans ce cas-là est le *Singleton*. C'est en général celui que l'on étudie en premier car il est simple à comprendre et à mettre en œuvre.

Le principe du ***singleton*** est de restreindre l'instanciation d'un objet à une instance unique.
La différence avec une classe statique est la possibilité de maitriser la construction de l'objet et de pouvoir le cas échéant utiliser une autre instance que celle par défaut.

Ce pattern convient parfaitement à la classe de service CustomerServices. L'usage en sera simplifié et le constructeur permettra de récupérer la chaine de connexion.

Voici donc le code modifié de la classe EmployeServices

```
class EmployeServices
{
    private static EmployeServices instance;

    2 références
    public static EmployeServices Instance
    {
        get
        {
            return instance ?? (instance = new EmployeServices());
        }
    }
```

On a utilisé des membres statiques de sorte à ne pas avoir à instancier la classe, ce qui simplifie le code.

Questions :

Expliquer le code suivant :

```
return instance ?? (instance = new EmployeServices());
```

Comment aurait-on pu l'écrire différemment ?

Testez ici votre application et mettez-la au point. Vous devriez toujours obtenir le même résultat, la méthode Main() n'a pas été modifiée.

Exercice 5 : Effectuer un couplage faible avec la BD

L'idée est de ne plus avoir de dépendance entre le code applicatif et la base de données.
Le changement de base de données ne doit pas impacter le code.
La source de données devient transparente et indépendante du code qui ne gère que des collections de classes.

Mise en œuvre :
Dans le framework .net, toutes les classes xxxConnection héritent de la classe **abstraite** DbConnection.
Pour vérifier, on peut placer son curseur sur OracleConnection et faire F12 :

```
namespace Oracle.DataAccess.Client
{
    public sealed class OracleConnection : DbConnection, ICloneable
    {
        public OracleConnection();
```

Idem pour sqlserver : la classe SqlConnection

```
namespace System.Data.SqlClient
{
    public sealed class SqlConnection : DbConnection, ICloneable
    {
        public SqlConnection();
```

Ou Mysql avec la classe MySqlConnection :

```
namespace MySql.Data.MySqlClient
{
    public sealed class MySqlConnection : DbConnection, IDisposable, ICloneable
    {
        public MySqlConnection();
        public MySqlConnection(string connectionString);
```

Vérifiez qu'il en est de même pour les classes :

- ✓ OracleDataReader, SqlDataReader, MySqlDataReader qui héritent toutes de la classe **abstraite** DbDataReader:
- ✓ OracleCommand, SqlCommand, MySqlCommand qui héritent toutes de la classe **abstraite** DbCommand

```
namespace Oracle.DataAccess.Client
{
    public sealed class OracleDataReader : DbDataReader
    {
        ~OracleDataReader();

        public override object this[string columnName] { get; }
        public override object this[int i] { get; }
```

On va donc utiliser ces classes pour faire un code plus générique !

En fait, l'utilisation des classes DbXXXX va permettre de compiler l'application sans pour autant savoir quelle classe dérivée de la classe DbXXX devra être instanciée. En effet, le nom du provider de données sera stocké dans le fichier de configuration.

Et c''est donc au moment de **l'exécution** que l'application va connaitre le fournisseur de données (provider) et va donc instancier la bonne classe.

Dans notre cas, on pourra choisir : Oracle ou Mysql

Mise en œuvre

Dans un premier temps, on va donc changer tous les noms de classe OracleConnexion, OraceCommand et OracleDataReader par les noms de classe DbConnexion, DbCommand et DbDataReader

Il vous appartient de rajouter les espaces de noms qui conviennent

Classe Bdd :

```
public class  Bdd
{
    2 références
    public static DbConnection CnOracle {
        get
        {
            string ch = String.Format(ConfigurationManager.ConnectionStrings["oracle"].ToString(), ConfigurationManager.AppSettings["SERVEUR"],
                                                                                    ConfigurationManager.AppSettings["PORT"],
                                                                                    ConfigurationManager.AppSettings["SID"],
                                                                                    ConfigurationManager.AppSettings["USERID"],
                                                                                    ConfigurationManager.AppSettings["PWD"]

                                    );
            return new DbConnection();
        }
    }
}
```

On voit qu'une erreur apparait …. C'est normal, on a dit plus haut que la classe DbConnection est abstraite, donc par définition elle est non instanciable.

La solution ?

On va mettre en place un nouvel outil de la POO, à savoir un Design Pattern appelé ***Factory*** (la fabrique)

L'idée du Design Pattern Factory est de pouvoir instancier un objet dont la classe est une classe dérivée d'une classe abstraite.
La classe de l'objet à instancier n'est donc pas connue au moment de la compilation, ni par la méthode appelante.

On ne programmera pas ces design pattern, on va utiliser ceux fournis par les classes DbXXX qui sont des classes que l'on a vu abstraites et qui vont se charger de l'instanciation de la bonne classe dérivée.

Dans notre cas, la fabrique (le factory) sera la classe abstraite **DbProviderFactory** qui va se charger :

- ✓ de récupérer une instance de System.Data.Common.DbProviderFactory pour un nom de fournisseur de données spécifié grâce à la méthode GetFactory(NomDuFournisseurDeDonnées) de la classe abstraite DbProviderFactories

- ✓ d'instancier un objet xxxConnection ou xxxCommand ou xxxDataReader en fonction du fournisseur de donnés (provider) récupéré à l'étape précédente.

Pour Oracle, le nom du provider (fournisseur de données) est Oracle.DataAccess.Client

Comme on va travailler sur des objets génériques, on va rajouter dans la classe Bdd une méthode **GetCommande** qui retournera un objet DbCommand. En fait un objet OracleCommand ou MysqlCommand.

Ainsi, la classe Bdd va subir de profondes modifications
Elle se présentera ainsi :

```
public class  Bdd
{
    2 références
    public static DbConnection GetConnexion {
        get
        {
            string fournisseurDeDonnees = ConfigurationManager.AppSettings["PROVIDER"];
            DbProviderFactory factory = DbProviderFactories.GetFactory(fournisseurDeDonnees);
            DbConnection connection = factory.CreateConnection();
            string ch = String.Format(ConfigurationManager.ConnectionStrings[fournisseurDeDonnees].ToString(), ConfigurationManager.AppSettings["SERVEUR"],
                                                                                    ConfigurationManager.AppSettings["PORT"],
                                                                                    ConfigurationManager.AppSettings["SID"],
                                                                                    ConfigurationManager.AppSettings["USERID"],
                                                                                    ConfigurationManager.AppSettings["PWD"]

                                      );
            connection.ConnectionString = ch;
            return connection;
        }
    }
    2 références
    public static DbCommand GetCommande
    {
        get
        {
            string fournisseurDeDonnees = ConfigurationManager.AppSettings["PROVIDER"];
            DbProviderFactory factory = DbProviderFactories.GetFactory(fournisseurDeDonnees);
            DbCommand commande = factory.CreateCommand();
            return commande;
        }
    }
}
```

Notez que la méthode CnOracle a disparu, elle n'a plus de raison d'être !!!

Il faut maintenant modifier la classe EmployeServices pour effacer toute trace de Oracle …. En dehors du nom du provider.

```
public List<Employe> FindAllEmployes()
{
    using (var MaConnexion = Bdd.GetConnexion)
    {
        List<Employe> employes = new List<Employe>();
        try
        {
            using (DbCommand cmdTousLesEmployes = Bdd.GetCommande)
            {
                cmdTousLesEmployes.CommandText = "select * from EMPLOYE";
                cmdTousLesEmployes.Connection = MaConnexion;
                MaConnexion.Open();
                DbDataReader readerEmploye = cmdTousLesEmployes.ExecuteReader();
                while (readerEmploye.Read())
                {
                    Employe employe = HydrateEmploye(readerEmploye);
                    employes.Add(employe);


                }
                readerEmploye.Close();
            }

            MaConnexion.Close();

        }
        catch (DbException ex)
        {
            Console.WriteLine(ex.Message);
        }
        return employes;
    }
}
```

Vous remarquerez

Que l'on a affecté explicitement à l'objet DbCommand les propriétés CommandText et Connection, alors qu'avant on passait ces valeurs en paramètres du constructeur de l'objet Command.

On a renommé la variable objet CnOracle en un nom plus générique : MaConnexion.

L'objet DbDataReader n'a pas besoin d'être instancié explicitement. Il ne figure donc pas dans la classe Bdd.

Exercice:

Modifier en conséquence la méthode FindEmployeById et exécuter le code principal qui n'a pas à être modifié.

… attention, pensez à tout …. Plus de trace de Oracle dans le code de la classe EmployeServices

Modifier l'application de sorte que le nom du fournisseur de données n'apparaisse pas en clair dans la classe Bdd (lignes 17 et 19).

```
16          {
17              DbProviderFactory factory = DbProviderFactories.GetFactory("Oracle.DataAccess.Client");
18              DbConnection connection = factory.CreateConnection();
19              string ch = String.Format(ConfigurationManager.ConnectionStrings["oracle"].ToString(), Configur
20                                                                          ConfigurationManager.App
```

Rappel de la méthode Main() :

```
0 références
static void Main()
{


    // parcours des employes

    foreach (Employe employe in EmployeServices.Instance.FindAllEmployes())
    {
        Console.WriteLine(employe);
    }
    Console.WriteLine("------------------------------");
    Console.WriteLine(EmployeServices.Instance.FindEmployeById(4));


    Console.ReadKey();
}
```

Exercice 6 : utilisation de plusieurs fournisseurs de données

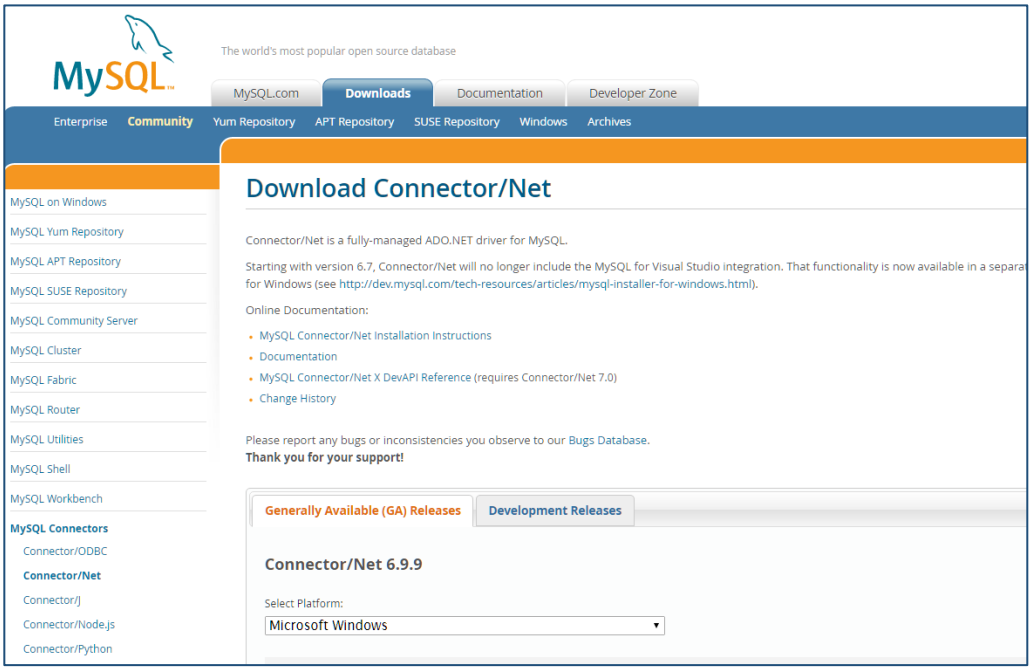
On va s'amuser encore un peu ….

L'idée maintenant est d'utiliser, comme dans le cours ADO, plusieurs fournisseurs de données (provider).

On va utiliser les mêmes tables dans les environnements Oracle (c'est fait) et MySql.

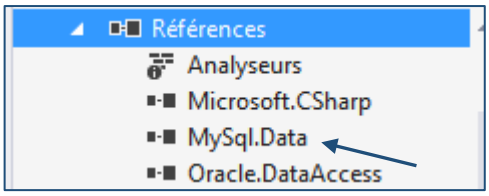
Il faut installer le fournisseur de données MySql.

URL de téléchargement : https://dev.mysql.com/downloads/connector/net/6.9.html



Téléchargez et installez.

Posez la référence :



Dans phpMyAdmin ou MysqlWorkbench (connecté en tant que root) , créez dans la base mysql un utilisateur adomysql/adomysql

Exécutez les scripts crebase_Mysql09V2.sql et CREOCC_mysqlV14.sql fournis par votre professeur.

Donnez les droits à l'utilisateur adomysql sur la base dbcours

La chaine de connexion pour se connecter à mysql est celle-ci :

```
String CnString = "server=localhost;User Id=adomysql;Password=adomysql; Persist Security Info=True;database=dbcours";
```

On constate qu'il y a 3 paramètres obligatoires plus le paramètre database qui indique la base de données par défaut.

Pour Oracle il y en avait 5.

On va donc modifier le fichier de configuration.

Il faut savoir que l'on ne peut pas rajouter les éléments XML que l'on veut dans le fichier app.confg.
Par contre on peut rajouter des sections, à condition de les déclarer. Ainsi on va créer **une section par provider**.

Attention, la déclaration des sections se fait toujours en début de fichier App.config :

Voici à quoi va ressembler dans un premier temps le fichier app.config :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="Oracle.DataAccess.Client"  type="System.Configuration.NameValueSectionHandler" />
  </configSections>
  <Oracle.DataAccess.Client>
    <add key="0" value="localhost"/>
    <add key="1" value="1521"/>
    <add key="2" value="XE"/>
    <add key="3" value="coursado"/>
    <add key="4" value="coursado"/>
  </Oracle.DataAccess.Client>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
  </startup>
  <connectionStrings>
    <add name="Oracle.DataAccess.Client" providerName="Oracle.DataAccess.Client"
        connectionString="Data Source=(DESCRIPTION =(ADDRESS = (PROTOCOL = TCP)(HOST = {0})(PORT = {1}))
        (CONNECT_DATA =(SERVER = DEDICATED)(SERVICE_NAME = {2})));User Id={3};Password={4};"
         />
  </connectionStrings>
  <appSettings>
    <add key="PROVIDER" value="Oracle.DataAccess.Client"/>
  </appSettings>
</configuration>
```

On a déclaré au tout début qu'on aurait une section Oracle.DataAccess.Client, du nom du fournisseur de données :

```
<configSections>
  <section name="Oracle.DataAccess.Client"  type="System.Configuration.NameValueSectionHandler" />
</configSections>
```

On a ensuite créé cette section et créé à l'intérieur les paramètres de la connexion, comme avant dans AppSettings :

```
<Oracle.DataAccess.Client>
  <add key="0" value="localhost"/>
  <add key="1" value="1521"/>
  <add key="2" value="XE"/>
  <add key="3" value="coursado"/>
  <add key="4" value="coursado"/>
</Oracle.DataAccess.Client>
```

On a mis dans la section AppSettings le nom du provider à utiliser :

```
<appSettings>
  <add key="PROVIDER" value="Oracle.DataAccess.Client"/>
</appSettings>
```

Enfin, on a changé le nom de la chaine de connexion pour Oracle, on lui a donné le nom du … fournisseur de données :

```
<connectionStrings>
  <add name="Oracle.DataAccess.Client" providerName="Oracle.DataAccess.Client"
      connectionString="Data Source=(DESCRIPTION =(ADDRESS = (PROTOCOL = TCP)(HOST = {0})(PORT = {1}))
      (CONNECT_DATA =(SERVER = DEDICATED)(SERVICE_NAME = {2})));User Id={3};Password={4};"
       />
</connectionStrings>
```

Il ne nous reste plus qu'à modifier notre code de la classe Bdd :

```
public static DbConnection GetConnexion {
    get
    {
        string fournisseurDeDonnees = ConfigurationManager.AppSettings["PROVIDER"];
        DbProviderFactory factory = DbProviderFactories.GetFactory(fournisseurDeDonnees);
        DbConnection connection = factory.CreateConnection();
        // construction de la chaine de connexion :
        string provider = ConfigurationManager.AppSettings["provider"];
        var section = ConfigurationManager.GetSection(provider) as NameValueCollection;
        int nb = section.Count;
        string[] parametres = new string[nb];
        section.CopyTo(parametres, 0);
        string ch = String.Format(ConfigurationManager.ConnectionStrings[provider].ConnectionString, parametres);

        // affectation de la chaine de connexion créée.
        connection.ConnectionString = ch;
        return connection;
    }
```

En premier on récupère le provider

```
string provider = ConfigurationManager.AppSettings["provider"];
```

Ensuite, on récupère la section correspondant au nom du provider que l'on stocke dans un objet section

```
var section = ConfigurationManager.GetSection(provider) as NameValueCollection;
```

Question : quel est la classe de cet objet ?

Puis on récupère le nombre d'éléments de cette section pour instancier un tableau ayant autant d'éléments que d'éléments de la section (nombre de paramètres de la chaine de connexion)

```
int nb = section.Count;
string[] parametres = new string[nb];
```

Enfin, on crée la chaine de connexion en fusionnant les paramètres dans la chaine :

```
section.CopyTo(parametres, 0);
string ch = String.Format(ConfigurationManager.ConnectionStrings[provider].ConnectionString, parametres);
```

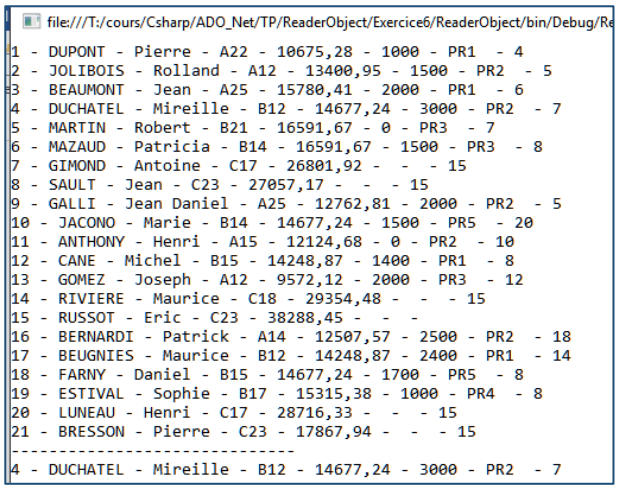
On peut tester, toujours avec le même main et la même classe EmployeServices :

```
static void Main()
{
    // parcours des employes

    foreach (Employe employe in EmployeServices.Instance.FindAllEmployes())
    {
        Console.WriteLine(employe);
    }
    Console.WriteLine("------------------------------");
    Console.WriteLine(EmployeServices.Instance.FindEmployeById(4));

    Console.ReadKey();
}
```

Résultat :



Maintenant on va faire la même chose pour Mysql :
On modifie le fichier App.config :

```
<configSections>
  <section name="Oracle.DataAccess.Client"  type="System.Configuration.NameValueSectionHandler" />
  <section name="MySql.Data.MySqlClient"  type="System.Configuration.NameValueSectionHandler" />
</configSections>
```

Puis on rajoute les paramètres :

```
<MySql.Data.MySqlClient>
  <add key="SERVER" value="localhost"/>
  <add key="USER" value="adomysql"/>
  <add key="PWD" value="adomysql"/>
  <add key="DB" value="dbcours"/>
</MySql.Data.MySqlClient>
```

Enfin, on change le nom du provider :

```
<appSettings>
  <add key="PROVIDER" value="MySql.Data.MySqlClient"/>
</appSettings>
```

On exécute …. Et ça marche ….

```
file:///T:/cours/Csharp/ADO_Net/TP/ReaderObject/Exercice0/ReaderObject/bin/Debug/ReaderObject.exe
1 - DUPONT - Pierre - A22 - 8000 - 1000 - PR1 - 4
2 - JOLIBOIS - Rolland - A12 - 10500 - 1500 - PR2 - 5
3 - BEAUMONT - Jean - A25 - 12000 - 2000 - PR1 - 6
4 - DUCHATEL - Mireille - B12 - 11500 - 3000 - PR2 - 7
5 - MARTIN - Robert - B21 - 13000 - 0 - PR3 - 7
6 - MAZAUD - Patricia - B14 - 13000 - 1500 - PR3 - 8
7 - GIMOND - Antoine - C17 - 21000 -  -  - 15
8 - SAULT - Jean - C23 - 21200 -  -  - 15
9 - GALLI - Jean Daniel - A25 - 10000 - 2000 - PR2 - 5
10 - JACONO - Marie - B14 - 11500 - 1500 - PR5 - 20
11 - ANTHONY - Henri - A15 - 9500 - 0 - PR2 - 10
12 - CANE - Michel - B15 - 10800 - 1400 - PR1 - 8
13 - GOMEZ - Joseph - A12 - 7500 - 2000 - PR3 - 12
14 - RIVIERE - Maurice - C18 - 23000 -  -  - 15
15 - RUSSOT - Eric - C23 - 30000 -  -  -
16 - BERNARDI - Patrick - A14 - 9800 - 2500 - PR2 - 18
17 - BEUGNIES - Maurice - B12 - 10800 - 2400 - PR1 - 14
18 - FARNY - Daniel - B15 - 11500 - 1700 - PR5 - 8
19 - ESTIVAL - Sophie - B17 - 12000 - 1000 - PR4 - 8
20 - LUNEAU - Henri - C17 - 22500 -  -  - 15
21 - BRESSON - Pierre - C23 - 14000 -  -  - 15
------------------------------
4 - DUCHATEL - Mireille - B12 - 11500 - 3000 - PR2 - 7
```

ZERO lignes de code modifiées !!!

Pour vous en convaincre, réessayez en rajoutant la ligne suivante en début de Main() :

```
Console.WriteLine(ConfigurationManager.AppSettings["PROVIDER"]);
```

Testez à nouveau :

```
Oracle.DataAccess.Client
1 - DUPONT - Pierre - A22 - 10675,28 - 1000 - PR1  - 4
2 - JOLIBOIS - Rolland - A12 - 13400,95 - 1500 - PR2  -
3 - BEAUMONT - Jean - A25 - 15780,41 - 2000 - PR1  - 6
4 - DUCHATEL - Mireille - B12 - 14677,24 - 3000 - PR2
```

Vous remarquerez, si ce n'est déjà fait un défaut dans le résultat…. A vous de trouver la solution …. Il y en a au moins une !!!

Epilogue…..
Si vous ne connaissez pas les noms des providers installés sur votre machine, vous pouvez exécuter le bout de code suivant :

```
var factories = DbProviderFactories.GetFactoryClasses();
foreach (DataRow factory in factories.Rows) {
    Console.WriteLine(factory[2] + " (" + factory[0] + ")");
}
```

Et vous obtiendrez le résultat suivant :

```
System.Data.Odbc (Odbc Data Provider)
System.Data.OleDb (OleDb Data Provider)
System.Data.OracleClient (OracleClient Data Provider)
System.Data.SqlClient (SqlClient Data Provider)
Oracle.DataAccess.Client (Oracle Data Provider for .NET)
System.Data.SqlServerCe.4.0 (Microsoft SQL Server Compact Data Provider 4.0)
MySql.Data.MySqlClient (MySQL Data Provider)
```

Mettez en œuvre les classes Cours et Projets Seminaire Inscrit.
Prévoyez des méthodes supplémentaires….

N'y a-t-il pas une amélioration de code à prévoir ?

Mise en pratique :

Reprendre le cas soins et le modifier pour mettre en pratique tous ces concepts.

Commentez vos programmes et générez la documentation.