Services Réseaux

COURS N°5 : TP5 ACCÈS AUX DONNÉES

Antony BRUGERE
a.brugere@cs3i.fr

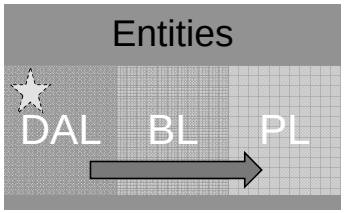
Frédéric CHASSAGNE
frederic.chassagne@capgemini.com

Plan du cours

- L 'accès aux données
  - Introduction
  - Accéder aux données
  - Manipuler les données
  - Sauvegarder les données
- La cryptographie : le hashage.
- Les tests unitaires

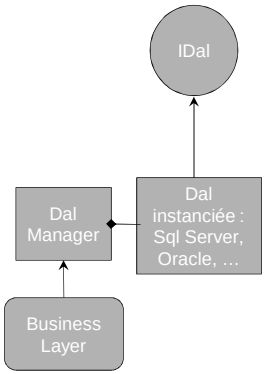
Architecture logicielle

- Couches du modèle 4 tiers
  - Presentation Layer
    - Interface Homme Machine
  - Business Layer
    - Noyau métier
  - Data Access Layer
    - Accès aux données
  - Entities Layer
    - Objets basiques



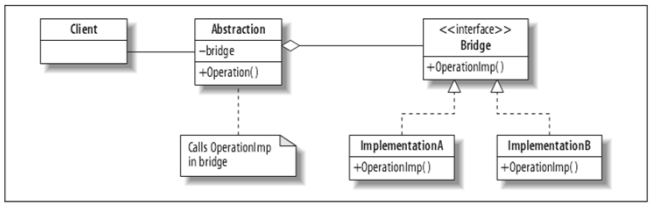
Introduction

- Data Access Layer
  - Flexibilité
  - Interopérabilité
- Interface
  - ∑ des besoins / persistance
- Dal Instanciée(s)
  - Implémentation Interfaces
- Couche supérieure
  - Utilisation du manager pour accéder aux données



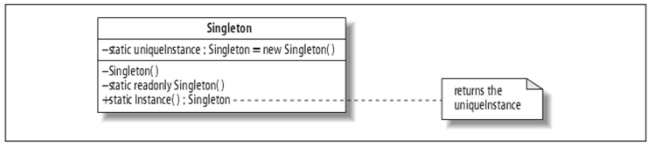
Introduction

- Dal = Implémentation d'un Bridge Pattern
- But = Support multi SGBD / Switch de Bdd aisé



Introduction

- Accès à la Dal depuis la couche supérieure
  - Voir seulement un objet = DalManager
  - Peut être mis à disposition par un singleton



Introduction

```
public class Singleton
{
    private static volatile Singleton
_instance;
    private static readonly object padlock =
new object();

    public static Singleton Instance
    {
      get {
        if (_instance == null)
        {
          lock (padlock)
          {
            if (_instance== null)
              {
                  _ instance = new
Singleton();
              }
            }
          }
          return _ instance;
```

```
private Singleton() {}

}
```

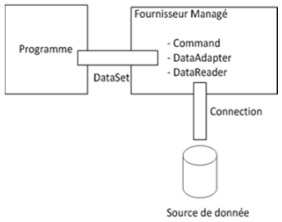
- Implémentation
  - Lock ThreadSafe
  - Attribut static
  - Constructeur private
- Utilisation
  - Singleton.Instance

Introduction

- Plusieurs moyens de faire de l'accès aux données en .net
  - Ado.net
  - Linq To Sql
  - Entity Framework
  - …
- Moyen de base vu ici = Ado.net

Introduction

- But Ado.net = Assurer la persistance des données
  - Dans une base de données
  - Dans un fichier Xml
- Principe



Introduction

- Les Objets Ado.net de base
  - Connection
  - Command
  - DataAdapter
  - DataReader
- Objets Abstraits, Existence d'une implémentation par SGBD
- Exemple d'implémentation de Connection
  - SqlConnection
  - OracleConnection
  - …

Introduction

- Connection
  - Ouvrir un lien physique vers la source de données
  - Nécessite une connectionString
- Command
  - Permet l'interrogation de la source de données
  - Nécessite un objet Connection
- DataAdapter
  - Permet la récupération/modification des enregistrement de la source e données
  - Nécessite un objet Command

Introduction

- DataReader
  - Alternative au DataAdapter pour la récupération d'enregistrement
  - Necessite un Objet Command
- DataSet
  - Objet contenant les enregistrements
  - Utile à la manipulation dans le programme
  - Remplit par l'objet DataAdapter
  - Voir comme 1 Collection de DataTables
    - 1 DataTable = Représentation Objet d'1Table de la Bdd

Introduction

- Ado.net permet l'accès à de multiples SGBD au travers d'1 même façade
- Pour ce cours, SGBD = Sql Server
- L'api à utiliser = System.Data.SqlClient

Accéder aux données

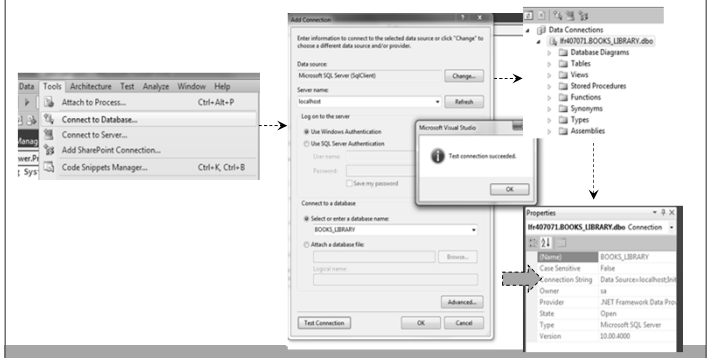
- Exemples donnés à partir d'une base Sql Server
- Objets manipulés
  - SqlConnection
  - SqlCommand
  - SqlDataAdapter
  - SqlDataReader
- But : Obtenir des données dans un objet Dataset / DataTable
  - Dataset = Collection de DataTables
  - DataTable = Représentation objet d'une table de la Bdd

Accéder aux données

- Créer un accès aux données en Ado.net
  - But : Accéder aux données de la persistance pour les mettre à disposition de la couche business
- Etape 1 : Connexion à la base de données
  - Nécessite une connectionString
  - Forme spécifique à chaque SGBD
  - Consultable sur le web
  - Peut être obtenu à partir de Visual Studio

Accéder aux données

- Obtenir la connectionString depuis Visual Studio



Accéder aux données

- 1 fois la connectionString obtenue, Instancier un objet SqlConnection avec la connectionString
- Attention :
  - Les objets Ado.net ne sont pas prévu pour persister
  - Le framework gère lui-même 1 pool de connexion
  - Les objets doivent être libéré dès que possible
- Code:

```
using (SqlConnection sqlConnection = new SqlConnection(_connectionString))
```

Accéder aux données

- Etape 2 – Récupérer les données
- A l'intérieur du bloc using
  - Instancier 1 objet SqlCommand avec en entrée
    - La requête à exécuter
    - L'objet SqlConnection
  - Instancier 1 objet SqlDataAdapter avec en entrée
    - L'objet SqlCommand créé juste avant
  - Instancer 1 DataTable pour récupérer les resultats et passer le en entrée de la méthode Fill() du DataAdapter

Accéder aux données

- Résumé de l'étape 2 avec 1 SqlDataAdapter

```
private DataTable SelectByDataAdapter(string request)
{
    DataTable results = new DataTable();

    using (SqlConnection sqlConnection = new SqlConnection(_connectionString))
    {
        SqlCommand sqlCommand = new SqlCommand(request, sqlConnection);
        SqlDataAdapter sqlDataAdapter = new SqlDataAdapter(sqlCommand);
        sqlDataAdapter.Fill(results);
    }

    return results;
}
```

- A la fin de cette étape, le résultat de la requête se trouve dans la DataTable instanciée

Accéder aux données

- SqlDataReader = Alternative au SqlDataAdapter
- Permet la lecture des enregistrements grâce à un pointeur
- Plus rapide que le SqlDataAdapter
- Préférable pour les requêtes volumineuses (vrai qq soit le SGBD)
- But : Obtenir une liste de String contenant des données de la base
- Etape 1 : Identique à l'exemple précédent

Accéder aux données

- A l'intérieur du bloc using
  - Instancier 1 objet SqlCommand en lui donnant en entrée
    - La requête à exécuter
    - L'objet SqlConnection
  - Appeler explicitement la méthode Open() de l'objet SqlConnection
  - Appeler la méthode ExecuteReader() de l'objet SqlCommand et lui affecter un objet SqlDataReader
  - Boucler sur la méthode Read() de l'Objet SqlDataReader
    - A l'intérieur, accéder aux informations par leur position dans la requête (0, 1, 2, …)
  - Appeler explicitement la méthode Close() de l'objet SqlConnection

Accéder aux données

- Résumé de l'étape 2 avec 1 SqlDataReader

```
private List<string> SelectByDataReader(string request)
{
    List<string> results = new List<string>();

    using (SqlConnection sqlConnection = new SqlConnection(_connectionString))
    {
        SqlCommand sqlCommand = new SqlCommand(request, sqlConnection);
        sqlConnection.Open();

        SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();

        while (sqlDataReader.Read())
        {
            results.Add(String.Format("{0} ({1})",
                                      sqlDataReader.GetString(1),
                                      sqlDataReader.GetGuid(0).ToString()));
        }
        sqlConnection.Close();
    }

    return results;
}
```

Accéder aux données

- Ado.net permet également l'exécution de procédures stockées
- But : Obtenir le résultat d'une procédure stockée existant dans la base
- Etape 1 : Identique à l'exemple précédent

Accéder aux données

- Etape 2
- A l'intérieur du bloc using
  - Instancier 1 objet SqlCommand sans paramètre d'entrée
  - Affecter sa propriété Connection à l'objet SqlConnection
  - Affecter sa propriété CommandType à « StoredProcedure »
  - Affecter sa propriété CommandText au nom de la procédure à exécuter
  - Exécuter la méthode ExecuteReader() comme dans l'exemple précédent

Accéder aux données

- Résumé de l'étape 2 avec 1 Procédure Stockée

```
private List<string> SelectByStoredProcedure(string entryParam)
{
    List<string> results = new List<string>();

    using (SqlConnection sqlConnection = new SqlConnection(_connectionString))
    {

        SqlCommand sqlCommand = new SqlCommand();
        sqlCommand.Connection = sqlConnection;

        sqlCommand.CommandText = "GetBooksByType";
        sqlCommand.CommandType = CommandType.StoredProcedure;
        sqlCommand.Parameters.AddWithValue("typeId", entryParam);

        sqlConnection.Open();

        SqlDataReader sqlDataReader = sqlCommand.ExecuteReader();

        while (sqlDataReader.Read())
        {
            results.Add(sqlDataReader.GetString(1));
        }

        sqlConnection.Close();
    }

    return results;
}
```

Accéder aux données

- L'accès aux données utilisent des objets dédiés
- Ces objets ne doivent pas être utilisé dans les autres couches
- Manipuler les données demande donc de transformer ces objets

Manipuler les données

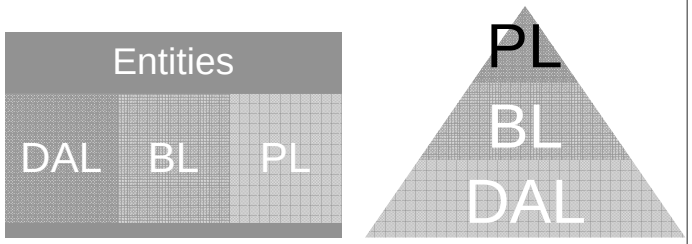
- Ne jamais faire d'accès aux données dans une couche présentation ou Business
- Créer une couche dédiée qui s'occupe de la persistance et met à disposition des Entités ou des interfaces d'objets
- Pas de mise à disposition directe d'objets DataTable car trop couplé à la forme de la base de données

Manipuler les données

- La couche Ado.net est donc cantonnée à la couche DAL
- La couche Business Layer relai les demandes (s'il n'y a pas de besoin métiers spécifique, la couche est très simple)
- La couche Présentation affiche les résultats

Manipuler les données

- But pour la manipulation des données
  - Renseigner les objets Entités dans le modèle en 4 couches
  - Renseigner les interface dans le modèle en 3 couches



Manipuler les données

- Au niveau de la présentation, ce sont les collections d'entités ou d'interfaces qui sont manipulées
- Lors de la sauvegarde des données, ce sont ces collections qui reviennent à la couche d'accès aux données
- Il faut donc faire le chemin inverse, alimenter les objets DataTable avec les collection d'Entités

Sauvegarder les données

- Couche « Data Access Layer »
  - Les collections à sauvegarder arrivent dans la couche par l'Objet DataAccessManager
- Fonctionnement d'un Update
  1. Récupérer la DataTable originale
  2. Parcourir la Collection & mettre à jour les lignes de la DataTable correspondante
  3. Utiliser l'objet SqlCommandBuilder pour effectuer la mise à jour

Sauvegarder les données

- Update via un SqlCommandBuilder
  - Créer un bloc using avec un SqlConnection puis à l'intérieur de ce bloc :
    1. Créer 1 SqlCommand en donnant la requête de Select et la DataTable à update
    2. Créer 1 SqlDataAdapter en donnant le SqlCommand
    3. Créer 1 SqlCommandBuilder en donnant le SqlDataAdapter
    4. Affecter la propriété UpdateCommand du SqlDataAdapter à la méthode GetUpdateCommand() du SqlCommandBuilder
    5. Idem pour les propriétés InsertCommand et DeleteCommand
    6. Affecter la Propriété MissingSchemaAction du SqlCommandBuilder à la valeur AddWithKey
    7. Appeler la méthode Update() du SqlDataAdapter

Sauvegarder les données

- Résumé du Update

```
private int UpdateByCommandBuilder(string request, DataTable authors)
{
    int result = 0;

    using (SqlConnection sqlConnection = new SqlConnection(_connectionString))
    {
        SqlCommand sqlCommand = new SqlCommand(request, sqlConnection);
        SqlDataAdapter sqlDataAdapter = new SqlDataAdapter(sqlCommand);

        SqlCommandBuilder sqlCommandBuilder = new SqlCommandBuilder(sqlDataAdapter);

        sqlDataAdapter.UpdateCommand = sqlCommandBuilder.GetUpdateCommand();
        sqlDataAdapter.InsertCommand = sqlCommandBuilder.GetInsertCommand();
        sqlDataAdapter.DeleteCommand = sqlCommandBuilder.GetDeleteCommand();

        sqlDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

        result = sqlDataAdapter.Update(authors);
    }

    return result;
}
```

Sauvegarder les données

- Possibilité d'ajouter un scope de Transaction à l'ensemble
- Intérêt
  - Possibilité de RollBack en cas d'exception
- Pour mettre en place, il suffit de
  - Créer la transaction

```
SqlTransaction myTrans = sqlConnexion.BeginTransaction();
```

  - Utiliser la signature du SqlCommand avec la transaction

```
SqlCommand sqlComm = new SqlCommand(requete, sqlConnexion,
                                    myTrans);
```

  - Utiliser dans le Catch de la méthode

```
myTrans.RollBack();
```

Cryptographie - Hachage

Définition : Opération qui consiste à appliquer une fonction mathématique permettant de créer l'empreinte ou signature numérique d'un message, transformant le message de taille variable en un code de taille fixe. C'est une fonction à sens unique : il est impossible de retrouver le message original.

Deux utilisations possible :

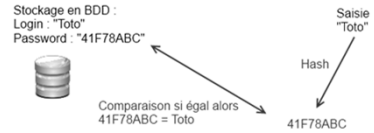
- Créer une clef unique pour un document, image ou autre afin de l'identifier rapidement.
- Rendre illisible certaines données pour l'oeil humain.

C'est cette dernière qui nous intéresse ici.

Cryptographie - Hachage

Le principe :

1) les mots de passe des utilisateurs sont stockés au format "hashé" dans la base de données.
2) Lorsque l'utilisateur s'identifie on "hash" le mot de passe saisie et on le compare à la chaîne stockée en base de données. Si les chaînes sont identiques = même mot de passe.



Cryptographie - Hachage

En .Net plusieurs fonctions de hashage disponible :
- MD5
- SHA1

etc.

1 classe abstraite regroupe les implémantations :
HashAlgorithm

Se trouve dans "System.Security.Cryptography"

Importance de saler 1 Hash pour le rendre unique

Cryptographie - Hachage

Exemple d'un hashage SHA1 :

```
public static class HashSH1
{
    /// <summary>
    /// Hash une chaîne de caractère en SHA1.
    /// </summary>
    /// <param name="data"></param>
    /// <returns></returns>
    public static string GetSHA1HashData(string data)
    {
        SHA1 sha1 = SHA1.Create();

        byte[] hashData = sha1.ComputeHash(Encoding.Default.GetBytes(data));

        StringBuilder returnValue = new StringBuilder();

        for (int i = 0; i < hashData.Length; i++)
        {
            returnValue.Append(hashData[i].ToString());
        }

        // return hexadecimal string
        return returnValue.ToString();
    }
}
```

Tests unitaires

- But
  - Permettre la validation de briques de code par l'exécution et l'analyse du résultat
  - Permet de tester et de maitriser chaque couche indépendamment des autres
- Caractéristiques
  - Automatique
  - Répétable
  - Disponible
- Microsoft Unit Testing Framework intégré à Visual Studio (projet de test)

Tests unitaires

- Modèle basique de programmation de test unitaire
  - [TestClass]/[TestMethod]
  - Une méthode de test par fonctionnalité à évaluer
- Analyse du résultat
  - Class statique Assert
  - Méthodes utiles
    - IsNotNull
    - AreEqual
    - IsTrue

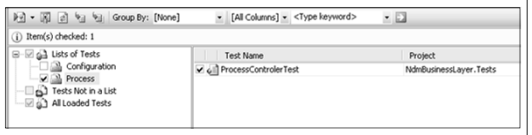
Tests unitaires

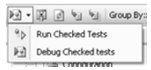
- Exemple :

```
[TestMethod]
public void TestConversion()
{
    ServiceClient clt = new ServiceClient();

     double res = clt.CurrencyConvert(10, SupportedCurrency.Euro,
    SupportedCurrency.LivreSterling);

    Assert.IsTrue(res < 10);
}
```

Tests unitaires

- Execution des tests unitaires dans VS 2010
  - Création d'un projet de tests
  - Lancement par la liste contenu dans le .vsmdi
  - 2 modes
    - Debug
    - Run





TP4

- Accès aux données
  - Gérer la connexion à une base de données SQL Server
  - Echanger des objets entre la bdd et l'application
  - Tester la couche d'accès aux données avec des tests unitaire