1 Présentation

Dans ce chapitre, nous étendrons le composant d'accès aux données que nous avons créées, afin d'implémenter des règles et traitements métier des applications que nous développerons. Le modèle d'entités qu'il contient est constitué de classes, où à chaque classe correspond une entité. Regardons la définition de l'une de ces classes, par exemple la classe *Secteur-Activite*, dans le fichier d'arrière-plan du modèle d'entités nommé *LearningCompany.Designer.cs* :

```
public partial class SecteurActivite : EntityObject
{
  // Implémentation de la classe
}
```

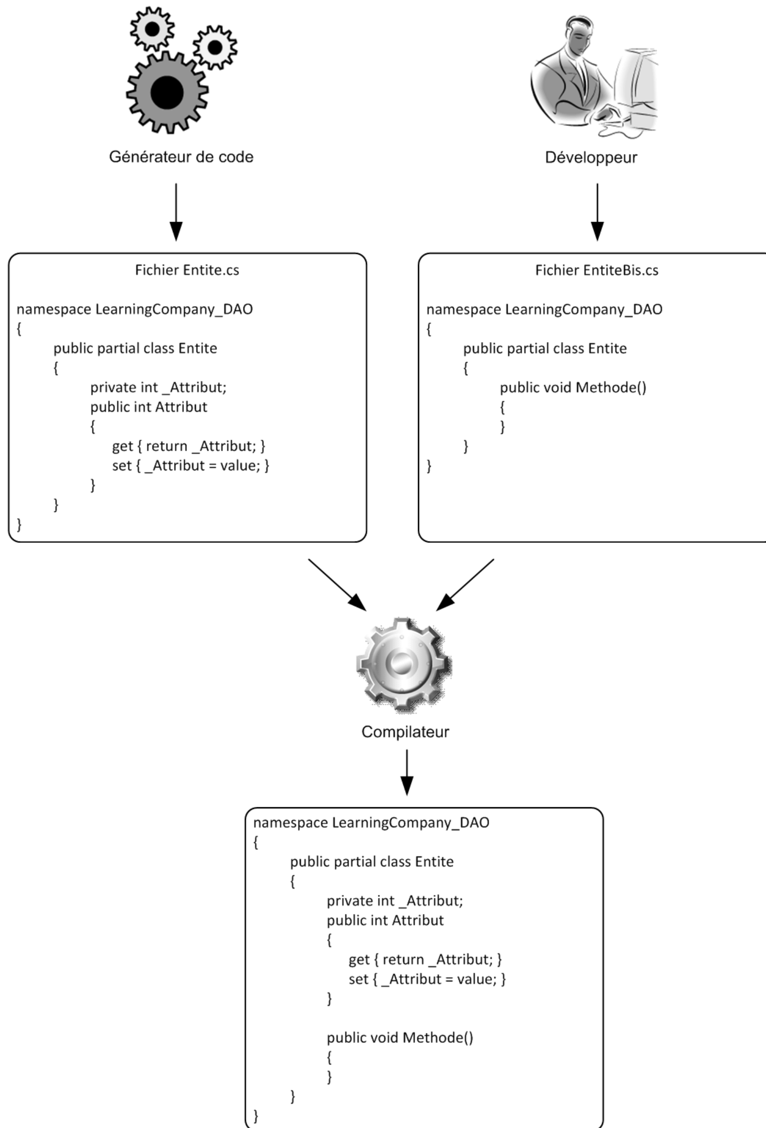
Nous pouvons observer que cette classe est une classe partielle. Toutes les entités de notre modèle sont définies ainsi. De ce fait, nous utiliserons le mécanisme des classes partielles pour étendre les entités et implémenter les contraintes, les règles et traitements métier.

2 Création des classes partielles

2.1 Rappels sur les classes partielles

Deux classes partielles sont deux classes d'un même projet, ayant le même nom, contenues dans le même espace de nom et définies avec le mot clé *partial*. Elles trouvent leur intérêt (entre autres) dans les générateurs de code, tels que le *Framework Entity* ou *Linq to SQL*. Une fois compilés, l'ensemble des membres définis dans les différents fichiers définissent une seule et même classe.

© Editions ENI - Toute reproduction interdite

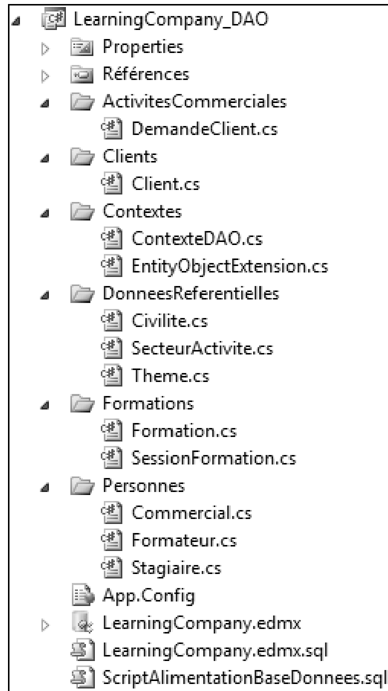


Compilation des classes partielles

Dans notre composant *LearningCompany_DAO*, nous ajoutons les classes partielles étendant les classes du modèle d'entités, en les regroupant suivant leur domaine fonctionnel. Ce regroupement permet de classer les classes partielles afin de pouvoir y accéder plus facilement tout en structurant notre projet.

© Editions ENI - Toute reproduction interdite

Nous créons alors les répertoires et classes suivants, comme le montre l'image ci-dessous :



Eléments du projet LearningCompany_DAO

Pour chacune des classes créées, nous leur appliquons les modifications suivantes :

- Nous modifions la définition de la classe en la déclarant publique afin de pouvoir l'utiliser en dehors du projet et partielle afin que ses membres complètent les membres de l'entité correspondante dans le modèle d'entités.
- Nous modifions l'espace de nom afin que la classe se situe dans le même espace de nom que la classe du modèle d'entités qu'elle complète, à savoir *LearningCompany_DAO*.

En appliquant ces modifications, nous devons avoir des classes dont l'implémentation correspond au prototype suivant :

```
namespace LearningCompany_DAO
{
    public partial class ClassePartielleEntite
    {
    }
}
```

© Editions ENI - Toute reproduction interdite

Une fois les classes partielles créées, nous devons définir et implémenter leurs membres. En définissant ces membres, nous détaillerons les règles fonctionnelles régissant les données et les traitements métier.

2.2 Contenu des classes partielles

2.2.1 Les fabriques

Les classes des entités générées contiennent une fabrique nommée Create*XXXX*, où *XXXX* correspond au nom de l'entité, permettant de créer une instance de l'entité. Cette fabrique accepte en paramètre l'ensemble des attributs scalaires que nous avons définis dans l'entité. Les propriétés de navigation ne sont pas prises en compte.

D'un point de vue implémentation avec le langage C#, une fabrique est une méthode statique, retournant une instance de la classe dans laquelle elle se situe. Voici un exemple de fabrique, que nous trouvons dans la classe *Client*, dans le fichier de code d'arrière-plan du modèle d'entités :

```
public static Client CreateClient(global::System.Int32
identifiant, global::System.String reference,
global::System.String raisonSociale, global::System.String
adresse, global::System.String codePostal, global::System.String
ville, global::System.String telephone, global::System.String
email, global::System.String urlSiteWeb, global::System.String
motDePasse, global::System.Int32 secteurActivite_Identifiant,
global::System.Int32 commercial_Identifiant)
{
    Client client = new Client();
    client.Identifiant = identifiant;
    client.Reference = reference;
    client.RaisonSociale = raisonSociale;
    client.Adresse = adresse;
    client.CodePostal = codePostal;
    client.Ville = ville;
    client.Telephone = telephone;
    client.Email = email;
    client.UrlSiteWeb = urlSiteWeb;
    client.MotDePasse = motDePasse;
    client.SecteurActivite_Identifiant = secteurActivite_Identifiant;
    client.Commercial_Identifiant = commercial_Identifiant;
    return client;
}
```


© Editions ENI - Toute reproduction interdite

Dans la classe partielle, si nous ajoutons un constructeur, alors il masquera le constructeur par défaut de la classe (constructeur non visible, public, et n'acceptant aucun paramètre) permettant de créer une instance de la classe. Si ce nouveau constructeur possède des paramètres, il aura alors une signature différente du constructeur par défaut, et une erreur de compilation sera levée lors de la compilation de la fabrique *CreateClient*, étant donné qu'elle l'utilise.

Afin de faciliter la création des objets métier que nous manipulerons dans nos applications, nous créons de la même manière nos propres fabriques dans les classes partielles que nous avons ajoutées.

Voici un exemple de fabrique que nous ajouterons dans la classe *Client* :

```
public static Client CreateClient(string aRaisonSociale, string
aReference, string aAdresse, string aCodePostal, string aVille,
string aTelephone, string aEmail, string aUrlSiteWeb,
SecteurActivite aSecteurActivite, Commercial aCommercial)
{
    Client oClient = new Client();

    oClient.RaisonSociale = aRaisonSociale;
    oClient.Reference = aReference;
    oClient.MotDePasse = "achanger";
    oClient.Adresse = aAdresse;
    oClient.CodePostal = aCodePostal;
    oClient.Ville = aVille;
    oClient.Telephone = aTelephone;
    oClient.Email = aEmail;
    oClient.SecteurActivite = aSecteurActivite;
    oClient.Commercial = aCommercial;

    return oClient;
}
```

Cette fabrique permet de créer un client à partir d'une raison sociale, d'une référence, d'informations de domiciliation, d'un numéro de téléphone, d'une adresse mail et d'une URL du site Web, d'un secteur d'activité et le commercial qui sera son interlocuteur.

2.2.2 Les méthodes de chargement de données

Nous ajoutons dans les entités deux méthodes de chargement de données. Une première méthode permettant d'obtenir une entité à partir de son identifiant (toute entité possède une propriété nommée *Identifiant* constituant la clé primaire de l'entité).

© Editions ENI - Toute reproduction interdite

Voici un exemple :

```
public static SecteurActivite GetInstance(int aIdentifiant)
{
    return (from oSecteurActivite in
ContexteDAO.ContexteDonnees.SecteursActivite
            where oSecteurActivite.Identifiant == aIdentifiant
            select oSecteurActivite).FirstOrDefault();
}
```

Au travers d'une requête *LINQ To Entities*, cette méthode permet d'interroger le contexte de données afin d'obtenir une instance de l'entité *SecteurActivite* dont l'identifiant est celui passé en paramètre. Si aucun secteur d'activité n'est trouvé, alors la valeur *Null* est renvoyée.

Puis nous ajoutons une seconde méthode permettant de charger l'ensemble des objets métier d'une entité. Voici un exemple :

```
public static List<SecteurActivite> GetListeInstances()
{
    return (from oSecteurActivite in
ContexteDAO.ContexteDonnees.SecteurActivite
            orderby oSecteurActivite.Libelle
            select oSecteurActivite).ToList();
}
```

Au travers d'une requête *LINQ To Entities*, cette méthode permet d'interroger le contexte de données afin d'obtenir l'ensemble des secteurs d'activités contenus dans la base de données, triés dans l'ordre alphabétique sur leur libellé. Si aucun secteur d'activité n'est trouvé, alors une liste vide est renvoyée.

2.2.3 Ajout d'accesseurs en lecture seule

Dans les classes partielles, nous ajouterons des accesseurs en lecture seule, qui permettront d'exécuter un test fonctionnel ou une logique métier et de retourner une valeur. Ces propriétés pourront être liées aux contrôles d'affichage (mise en œuvre du *DataBinding*) de données, ce qui facilitera grandement le développement des formulaires. Voici un exemple implémenté dans la classe *Formateur* :

```
public string NomPrenom
{
    get
```


© Editions ENI - Toute reproduction interdite