TP .Net

Début avec le Framework .NET

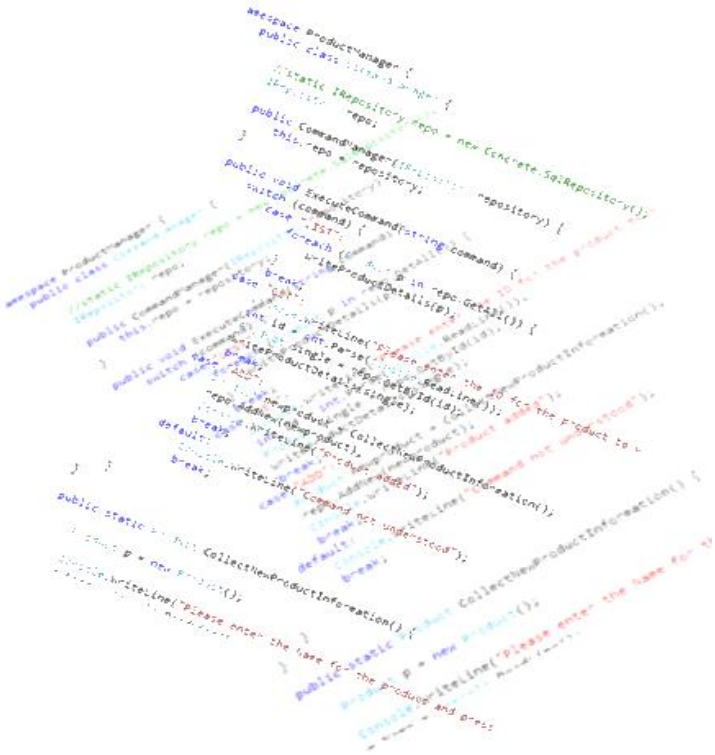
Lemettre Arnaud

Version 1.0

19 Pages

28/02/2009

EPITA-MTI2014-NET-TP



Multimédia
Technologies de l'Information

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Propriétés du document

Auteur	**Lemettre Arnaud**
Version	1.0
Nombre de pages	19
Références	EPITA-MTI2014-NET-TP

Historique du document

Date de vision	Version	Auteur	Changements
12/12/2010	0.1	Lemettre	création

Site de référence

description	url
Site MTI	
Blog MTI	

Sommaire

Introduction 4
Partie 1 5
Prérequis 5
Création du 1er projet 5
Factorielle 5
Partie 2 6
Création du 2ème projet 6
Création d'une classe 6
Lecture d'un fichier 6
Logique métier 7
Gestion de l'interface 7
Partie 3 9
Création du 3ème projet 9
Création des classes 9
Atelier 9
Gestion des exceptions 10
Implémentation des outils 11
La partie Métier 12
Les interfaces 12
Les différentes stratégies 12
Logique Métier 13
L'interface 18
Modalité de rendu 19

Introduction

Le but de ce TP sera la manipulation du framework .NET. Ce TP se décompose en plusieurs parties, chaque partie est indépendante.

Ce travail est à faire individuellement, tout code similaire sur deux personnes sera considéré comme un travail non rendu et non négociable.

Durant cette série de TP les notions suivantes seront abordées :

- Développement d'une 1ère application
- Manipulation de dossiers
- Gestion des exceptions
- Manipulation de design Pattern (Factory, Strategy, …)
- Développement de fonction générique

Ce TP est corrigé par **moulinette**, vous devez donc être rigoureux, ce TP ne demande pas plus de 4h de travail. Le sujet est très long car très détaillé pour votre 1er TP.

Bonne chance ;)

Informations : dans l'ensemble du sujet login_l correspond à votre login EPITA.

Partie 1

Prérequis

Vous devez créer une solution sous visual studio 2012. Cette solution devra se nommer :

login_lTP1

où login_l correspond à votre login EPITA.

Création du 1er projet

Ajouter un projet de type console. Ce projet devra s'exécuter pour le framework .NET 4.5.

Il portera le nom suivant : ConsoleFirstProgram

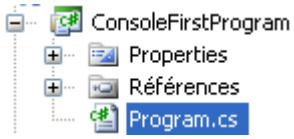


Figure 1 : Résultat de la création du projet

Ce projet comportera les fonctions suivantes.

Factorielle

La 1ère fonction à développer sera la factorielle. Pour rappel :

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Nom de la fonction	**Facto**	
Paramètre en entrée	int	Le nombre pour la factorielle
Paramètre en sortie	Int	Le résultat
Espace de nom	ConsoleFirstProgram	
Classe	Program	
Portée	Private static	

Signature:

```
private static int Facto(int nb);
```

Dans le cas où les paramètres sont bons, la fonction retournera le bon résultat, dans tous les autres cas la fonction devra retourner -1. La fonction ne devra pas déclencher d'exceptions. Pour tester votre fonction, vous pouvez appeler dans la fonction Main de votre programme.

Partie 2

Création du 2ème projet

Ajouter un projet de type console. Ce projet devra s'exécuter pour le framework .NET 4.5.

Il portera le nom suivant : ConsoleSecondProgram

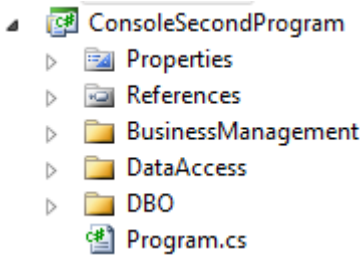


Figure 2 : Résultat de la création du projet

Il faudra rajouter les dossiers suivants :

- BusinessManagement
- DataAccess
- DBO

Le but de cette partie sera de manipuler les fichiers textes et de mettre en place une architecture en couche.

Ce projet comportera les fonctions suivantes.

Création d'une classe

Pour commencer cette partie, vous allez développer une classe nommée Person (public) ayant pour getter / setter :

Getter / setter	Type
Address	String
Firstname	String
Name	String

Cette classe doit être dans le namespace : ConsoleSecondProgram.DBO

Lecture d'un fichier

La lecture d'un fichier permet de faire un accès aux données. Vous devez donc créer une classe dans l'espace de nom ConsoleSecondProgram.DataAccess. Cette classe se nomme ReadFile et devra implémenter l'interface IDisposable.

Le constructeur de la classe prendra en paramètre un string qui représente le nom du fichier à lire. Le constructeur initialisera un StreamReader. Cette classe aura pour méthode :

Nom de la fonction	ReadData	
Paramètre en entrée		
Paramètre en sortie	List<DBO.Person>	La liste des personnes contenue dans le fichier
Espace de nom	ConsoleSecondProgram.DataAccess	
Classe	ReadFile	
Portée	Public	

Signature:

```
public List<DBO.Person> ReadData();
```

Cette méthode devra lire le fichier qui a été déclaré dans le constructeur de la classe. Les données dans le fichier sont stockées selon la forme suivante :

```
Nom;Prénom;Adresse
Nom2;Prénom2;Adresse2
```

La liste contiendra donc deux éléments dans ce cas de figure. Le nombre d'enregistrements peut être aléatoire ! Vous devez catcher les exceptions et en renvoyer une seule ayant pour message : « Erreur durant la lecture » et englobant l'exception générée.

Logique métier

Afin de coordonner l'accès aux données, il faut une classe pour englober les accès aux données. Cette classe va se nommer Person et devra avoir comme portée public static.

Cette classe sera dans le namespace : ConsoleSecondProgram.BusinessManagement et contiendra une méthode.

Nom de la fonction	ReadData	
Paramètre en entrée	String	Le nom du fichier
Paramètre en sortie	List<DBO.Person>	La liste des personnes contenue dans le fichier
Espace de nom	ConsoleSecondProgram. BusinessManagement	
Classe	Person	
Portée	Public static	

Signature :

```
public static List<DBO.Person> ReadData(string filename);
```

Gestion de l'interface

Pour faire fonctionner cette application, il vous faut compléter la méthode Main de la classe Program. Cette méthode devra initialiser une liste de Person (DBO) puis lire un fichier CSV et vous vérifierez que vous remontez les bons éléments.

Information

Pour information à la fin de cette partie ce projet doit avoir au minimum les méthodes et propriétés ci-dessus.

Partie 3

Création du 3ème projet

Le 3ème projet vous permettra de manipuler des designs patterns au sein d'une architecture n-tier. Vous devrez implémenter le design pattern *factory* ainsi que *strategy*. Ce projet permettra de faire le calcul de la moyenne d'ateliers basée sur les notes de soutenance et de suivis. La méthode de calcul peut changer d'un semestre à l'autre d'où l'implémentation de différentes stratégies. Pour obtenir une traçabilité, ces ateliers pourront être enregistrés sur le disque.

Précision : Pour l'ensemble des retours à la ligne (\n, \r\n, <retour>) pour éviter toute confusion, merci d'utiliser `Environment.NewLine`

Ajouter un projet de type console. Ce projet devra s'exécuter pour le framework .NET 4.5.

Il portera le nom suivant : ConsoleTP1

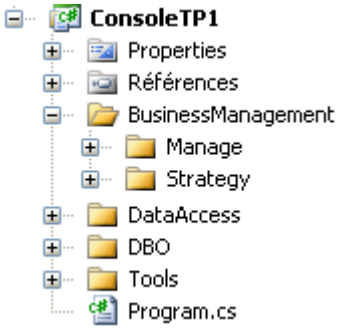


Figure 3 : Résultat de la création du projet

Il faudra également ajouter les dossiers suivants à ce projet :

- BusinessManagement
  - Manage
  - Strategy
- DBO
- Tools

Ce projet comportera les fonctions ci-après.

Création des classes

Atelier

Pour réaliser cette partie, vous devez implémenter deux classes. La première classe sera une classe d'objet « fonctionnelle » tandis que la seconde sera plus technique.

Concernant la 1ère classe fonctionnelle, elle représentera un atelier.

Getter / setter	Type
Name	String
Description	String
TrackMark	decimal[]
VivaMark	decimal[]

De plus, cette classe devra overrider la méthode ToString.

Nom de la fonction	ToString	
Paramètre en entrée		
Paramètre en sortie	String	La représentation textuelle de l'objet
Espace de nom	ConsoleTP1.DBO	
Classe	Atelier	
Portée	public override	

Signature :

```
public override string ToString()
```

Cette fonction devra retourner une string contenant chaque note contenue dans TrackMark séparée par un espace, puis mettre un « ; » pour afficher les notes contenues dans VivaMark.

Exemple pour un objet Atelier:

Nom : Microsoft

Description : L'atelier

TrackMark : [1, 2]

VivaMark : [5, 10]

Devra afficher :

```
1 2 ;5 10
```

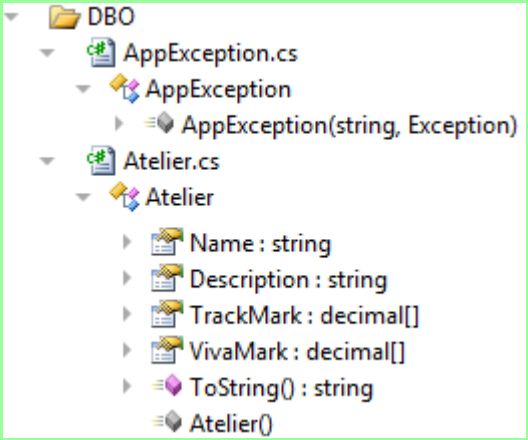
Pour le détail => 1<espace>2<espace>;5<espace>10<espace>

Gestion des exceptions

Pour la gestion des exceptions, il faut pouvoir également en déclencher de types personnalisés. Pour cela, il faut créer une classe qui se nomme AppException et qui hérite d'Exception. Le constructeur devra prendre en argument deux paramètres, un de type string et le deuxième paramètre de type Exception.

Nom de la classe	AppException
Herite	Exception
Constructeur	AppException(String message, Exception ex)
Namespace	ConsoleTP1.DBO

Information



Pour information : à la fin de cette partie, ce projet doit avoir au minimum les méthodes et propriétés ci-dessus.

Implémentation des outils

Dans certaines applications, des fonctionnalités transverses peuvent apparaître dans les besoins néanmoins ces fonctionnalités non pas une logique métier. On peut donc créer une classe Tools dans le dossier tools. Cette classe pourra contenir un certain nombre de fonctions. Pour ce TP, vous devrez implémenter une fonction générique permettant d'afficher dans une console tous les éléments d'un tableau. Celle-ci répondra aux critères suivant :

Nom de la fonction	**PrintArray**	
Paramètre en entrée	E[]	Tableau contenant les valeurs à afficher
Paramètre en sortie	void	
Espace de nom	ConsoleTP1.Tools	
Classe	Tools	
Portée	public static	

Signature :

```
public static void PrintArray<E>(E[] inputArray);
```

Le format d'affichage est le suivant :

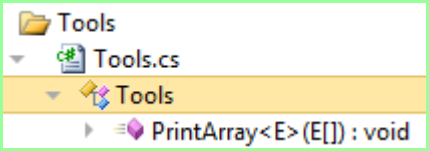
TrackMark : [1, 2]

Devra afficher :

```
1 2
```

Pour le détail => 1<espace>2<espace><retour>
<retour>

Information



Pour information : à la fin de cette partie ce projet doit avoir au minimum les méthodes et propriétés ci-dessus.

La partie Métier

Les interfaces

Nous allons maintenant nous concentrer sur la partie métier de l'application qui nous permettra de réaliser les algorithmes de calcul. Pour cela, nous implémenterons plusieurs designs patterns. Nous allons dans un premier temps déclarer nos interfaces. Une interface qui représentera le contrat de nos ateliers.

Nom de l'interface	**IAtelier**
Namespace	ConsoleTP1.BusinessManagement
Portée	Public
Signature des fonctions	decimal Calculate(DBO.Atelier atelier)
	string Presentation(DBO.Atelier atelier)

Pour le pattern strategy, nous avons besoin également d'une interface qui nous permettra de pouvoir changer la manière de calculer les notes selon les semestres.

Nom de l'interface	**ICalculateStrategy**
Namespace	ConsoleTP1.BusinessManagement
Portée	Public
Signature des fonctions	decimal CalculateAverage(DBO.Atelier atelier)

Les différentes stratégies

Pour réaliser le calcul de la moyenne de chaque atelier, nous allons appliquer des stratégies. Pour cela, créer une classe dans le namespace ConsoleTP1.BusinessManagement.Strategy qui se nommera CalculateFirstSemester et implémentera l'interface ICalculateStrategy. La méthode CalculateAverage pour cette stratégie devra renvoyer la moyenne de TrackMark et VivaMark d'un atelier selon le coefficient suivant 1 pour TrackMark et 2 pour VivaMark.

Nom de la classe	**CalculateFirstSemester**
Hérite	ICalculateStrategy
Constructeur	
Namespace	ConsoleTP1.BusinessManagement.Strategy
Portée	public class

Pour la fonction qui calcule la moyenne, elle doit répondre aux contraintes suivantes :

Nom de la fonction	**CalculateAverage**	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	decimal	La moyenne
Espace de nom	ConsoleTP1.BusinessManagement.Strategy	
Classe	CalculateFirstSemester	
Portée	public	

Signature:

```
public decimal CalculateAverage(DBO.Atelier atelier)
```

Pour la seconde stratégie, le principe est le même sauf pour la méthode de calcul. Les coefficients sont inversés, autrement dit coefficient suivant 2 pour TrackMark et 1 pour VivaMark.

Nom de la classe	**CalculateSecondSemester**
Hérite	ICalculateStrategy
Constructeur	
Namespace	ConsoleTP1.BusinessManagement.Strategy
Portée	public

Pour la fonction :

Nom de la fonction	**CalculateAverage**	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	decimal	La moyenne
Espace de nom	ConsoleTP1.BusinessManagement.Strategy	
Classe	CalculateSecondSemester	
Portée	public	

Signature:

```
public decimal CalculateAverage(DBO.Atelier atelier)
```

Logique Métier

Maintenant que nous avons mis en place nos stratégies pour le calcul, nous devons mettre en place la factory de nos ateliers qui contiendra une logique métier spécifique pour chaque atelier.

Nous allons donc créer un atelier (par défaut) duquel hériterons les autres ateliers qui voudront spécifier certaines méthodes. Pour cela créer une classe dans ConsoleTP1.BusinessManagement.Manage, celle-ci se nommera DefaultAtelier et devra implémenter IAtelier.

Nom de la classe	**DefaultAtelier**
Hérite	IAtelier
Constructeur	public DefaultAtelier()
Namespace	ConsoleTP1.BusinessManagement.Manage
Portée	public

Cette classe aura un attribut permettant de changer la stratégie de calcul.

Getter / setter	Type
CalculateStrategy	ICalculateStrategy

Et elle implémentera les fonctions suivantes :

Nom de la fonction	**Calculate**	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	decimal	La moyenne
Espace de nom	ConsoleTP1.BusinessManagement. Manage	
Classe	DefaultAtelier	
Portée	public	

Signature :

```
public decimal Calculate(DBO.Atelier atelier)
```

Cette fonction devra calculer la moyenne d'un atelier sans pondération (VivaMark + TrackMark) toutefois si le getter / setter CalculateStrategy est différent de null alors cette fonction devra renvoyer le résultat du calcul défini par la stratégie courante.

Nom de la fonction	**Presentation**	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	String	Le résultat à afficher
Espace de nom	ConsoleTP1.BusinessManagement. Manage	
Classe	DefaultAtelier	
Portée	public virtual	

Cette fonction devra respecter la sortie suivante :

```
Nom : Nom_Atelier\n\n\n Description : Description_Atelier
```

Où Nom_Atelier correspond au nom de l'atelier et Description_Atelier correspond à la description de l'atelier.

Il faut également créer une autre classe Adobe qui héritera de DefaultAtelier. Cette classe devra spécialiser la méthode Presentation.

Nom de la classe	**Adobe**
Hérite	DefaultAtelier
Constructeur	
Namespace	ConsoleTP1.BusinessManagement.Manage
Portée	public

Nom de la fonction	Presentation	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	String	Le résultat à afficher
Espace de nom	ConsoleTP1.BusinessManagement. Manage	
Classe	Adobe	
Portée	public override	

Signature :

```
public override string Presentation(DBO.Atelier atelier)
```

Cette fonction devra respecter la sortie suivante :

```
Nom_Atelier\n\n\n Description : Description_Atelier
Url : http://adobe.com
```

Cette classe sera exactement la même que celle pour l'atelier Microsoft.

Nom de la classe	Microsoft
Hérite	DefaultAtelier
Constructeur	
Namespace	ConsoleTP1.BusinessManagement.Manage
Portée	public

Nom de la fonction	Presentation	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	String	Le résultat à afficher
Espace de nom	ConsoleTP1.BusinessManagement. Manage	
Classe	Microsoft	
Portée	public override	

Signature :

```
public override string Presentation(DBO.Atelier atelier)
```

Cette fonction devra respecter la sortie suivante :

```
Nom_Atelier\n\n\n Description : Description_Atelier
Url : http://microsoft.com
```

Pour les autres ateliers, il vous faudra créer une classe OtherTechno.

Nom de la classe	OtherTechno
Hérite	DefaultAtelier
Constructeur	
Namespace	ConsoleTP1.BusinessManagement.Manage
Portée	public

Avec pour spécialisation la méthode Présentation :

Nom de la fonction	Presentation	
Paramètre en entrée	DBO.Atelier	Atelier contenant les notes pour faire la moyenne
Paramètre en sortie	String	Le résultat à afficher
Espace de nom	ConsoleTP1.BusinessManagement. Manage	
Classe	Microsoft	
Portée	public override	

Signature :

```
public override string Presentation(DBO.Atelier atelier)
```

Cette fonction devra respecter la sortie suivante :

```
Nom_Atelier\n\n\n Description : Description_Atelier
Url : http://mti.epita.fr
```

Maintenant que nous avons l'ensemble de nos ateliers, il nous faut construire la factory.

Nous allons créer une classe FactoryAtelier, dans ConsoleTP1.BusinessManagement, celle-ci nous renverra une instance de Manage.Atelier contenant la logique métier pour un atelier.

Nom de la classe	FactoryAtelier
Hérite	
Constructeur	
Namespace	ConsoleTP1.BusinessManagement
Portée	public

La récupération de la logique métier se fera par une méthode :

Nom de la fonction	GetAtelier	
Paramètre en entrée	String	Type de l'atelier
Paramètre en sortie	IAtelier	La logique métier pour un atelier
Espace de nom	ConsoleTP1.BusinessManagement	
Classe	FactoryAtelier	
Portée	public static	

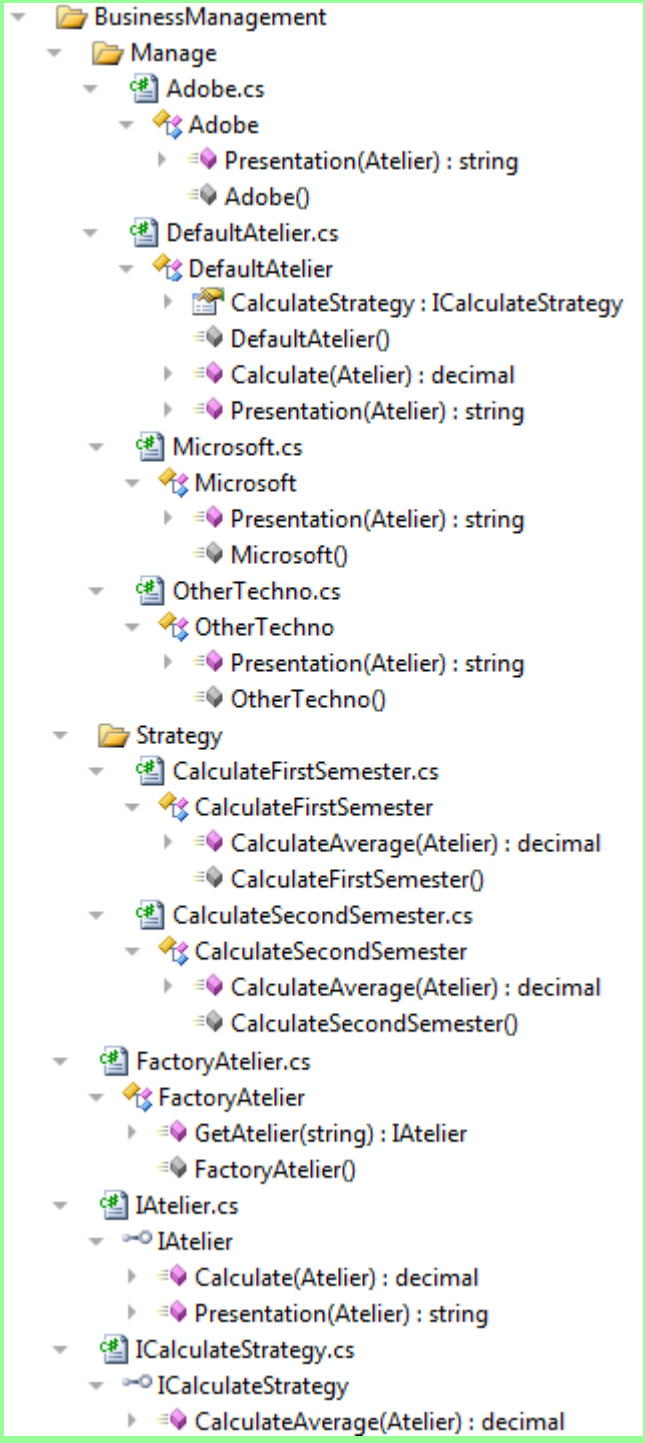
Signature :

```
public static IAtelier GetAtelier(string typeName)
```

On passera donc le type d'un atelier sous la forme d'une chaîne de caractères. En analysant cette dernière, la fonction devra nous retourner une instance de classe atelier. Attention, votre méthode ne doit pas être case sensitive. Liste exhaustive des cas :

Pour « microsoft » vous devez retourner une instance de la classe Microsoft, pour « adobe » vous devez retourner une instance de la classe Adobe, pour « opensource » et « adminsys » vous devez retourner une instance de la classe OtherTechno, pour tout le reste vous devez retourner une instance de la classe DefaultAtelier.

Information



Pour information : à la fin de cette partie ce projet doit avoir au minimum les méthodes et propriétés ci-dessus.

L'interface

Pour tester l'ensemble de votre développement, réaliser une fonction dans la classe Program qui initialisera une collection d'Atelier avec différentes valeurs.

Nom de la fonction	**InitList**	
Paramètre en entrée		
Paramètre en sortie	IList<DBO.Atelier>	La collection d'ateliers initialisés.
Espace de nom	ConsoleTP1	
Classe	Program	
Portée	private static	

Signature :

```
private static IList<DBO.Atelier> InitList()
```

Puis, dans la méthode Main vous allez pouvoir écrire l'algorithme de test. Une fois la liste initialisée, parcourrez chaque élément de la liste pour inscrire sur la console : le nom de l'atelier et l'affichage des éléments du tableau TrackMark puis ceux du tableau VivaMark. A la suite de quoi, vous pouvez récupérer la logique métier pour cet atelier en particulier et ainsi calculer puis afficher sa moyenne. Il ne vous reste plus qu'à déclencher la fonction de sauvegarde.

Modalité de rendu

Les fichiers seront à rendre dans une tarball ayant pour nom :

login_l.zip

Cette tarball devra comprendre à la racine:

- Un dossier contenant la solution Visual Studio qui devra compiler.
  Nom : login_lTP1
  Une fois décompressée nous devrions avoir :
  /Login_lTP1
  *.sln
  /ConsoleFirstProgram
  /ConsoleSecondProgram
  /ConsoleTP1

Le tout à envoyer sur l'adresse mti.rendu.dotnet@gmail.com avec les balises suivantes :

[MTI2014][NET][login_l][TP1]