



Cégep de Saint-Hyacinthe
Département d'informatique

Programmation orientée objet

420-2DP-HY

(3-3-3)

Framework et principe fondamental (MVVM)

(Version 1.1)

1 heure

Préparé par

Martin Lalancette

Comprendre les éléments suivants:

- Qu'est-ce que le *.Net framework* ou *.Net Core*
- Les *namespaces*
- Principe fondamental
- Qu'est-ce que MVVM?

Table des matières

Qu'est-ce que le <i>.NET framework</i> ou <i>.Net Core</i> ?	3
Les avantages	3
Les inconvénients	3
Les versions	4
Vérifier la version <i>.NET Core</i>	4
Comprendre les <i>namespace</i> (espace de nom)	5
Références	6
Explorateur d'objets (Menu Affichage)	7
Principe fondamental	7
Structurer un projet selon MVVM (version légère)	8
Bibliographie	11

Qu'est-ce que le *.NET framework* ou *.Net Core*?

« Un framework est un ensemble d'outils et de composants logiciels organisés conformément à un plan d'architecture et des modèles, l'ensemble formant ou promouvant un squelette de programme. Il est souvent fourni sous la forme d'une **bibliothèque logicielle**, et accompagné du plan de l'architecture cible du framework » (Wikipédia, 2012). Le terme *framework* est traduit de la façon suivante l'*Office de la langue française* : **cadre d'applications** ou **cadriciel**. Il contient des bibliothèques de classes qui permettent de faciliter la tâche du programmeur. Il s'occupe de gérer l'exécution d'une application, d'allouer la mémoire pour l'emmagasinement des données et des instructions du programme.

Les avantages

- Permet de coder plus rapidement et simplement.
- Facilite le déploiement des applications. La dimension des exécutables (.EXE) et de bibliothèques (.DLL) se trouve ainsi réduite, car les objets du *framework* n'y sont pas compilés directement. Ils seront recompilés sur l'ordinateur client lors du premier démarrage.
- En facilitant la maintenance grâce aux versions (Global Assembly Cache).
- Sert de guide en divisant les classes d'un domaine visé en modules.

Les inconvénients

- Nécessite plus de ressources du système parfois.
- Le code produit et compilé à l'aide de cette technologie peut être facilement décompilé. Risque de perte de secrets et risque de contournement des contrôles de licences (droits et brevets).
- Le framework doit être installé sur le poste client pour fonctionner

Pour connaître ou changer le framework avec lequel vous compilez, faire les étapes suivantes:

1. Dans le menu, cliquer sur PROJET
2. Choisir Propriétés de ...
3. Du côté gauche, choisir Application.

Exemple :

Les versions

Il est nécessaire de vérifier quels sont les outils déjà installés sur votre poste de développement afin de vous synchroniser avec les autres développeurs. À savoir (deux types de outils) :

- **Runtime** → Pour exécution seulement (plus léger).
- **SDK (Software Development Kit)** → Développement et exécution avec assistance (débugage).

Voici des liens à visiter :

1. [Download .NET Framework ou Core](#)
2. [Histoire de .NET Framework](#)
3. [Versions de C#](#)

Vérifier la version .NET Core

Avant de commencer à créer des applications Web, il faut vérifier la version .NET Core installée et la mettre à jour s'il y a lieu. Voici la procédure pour vérifier :

1. Ouvrir une fenêtre PowerShell,
2. Taper la commande **dotnet --info**.

```

Windows PowerShell
Copyright (c) Microsoft Corporation. Tous droits réservés.

Tester le nouveau système multiplateforme PowerShell https://aka.ms/powershell

PS C:\Users\lala> dotnet --info
.NET Core (réfutant tous les global.json) :
  Version: 3.1.401
  Commit: 5b6f5e5805

Environnement d'exécution :
  OS Name: Windows
  OS Version: 10.0.19041
  OS Platform: Windows
  RID: win10-x64
  Base Path: C:\Program Files\dotnet\sdk\3.1.401\

Host (useful for support):
  Version: 3.1.7
  Commit: fcfdef8d5b

.NET Core SDKs installed:
  2.1.805 [C:\Program Files\dotnet\sdk]
  3.1.401 [C:\Program Files\dotnet\sdk]

.NET Core runtimes installed:
  Microsoft.AspNetCore.App 2.1.17 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
  Microsoft.AspNetCore.App 2.1.21 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
  Microsoft.AspNetCore.App 2.1.22 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
  Microsoft.AspNetCore.App 3.1.7 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]
  Microsoft.NETCore.App 2.1.17 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  Microsoft.NETCore.App 2.1.21 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  Microsoft.NETCore.App 3.1.7 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  Microsoft.WindowsDesktop.App 3.1.7 [C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App]

To install additional .NET Core runtimes or SDKs:
  https://aka.ms/dotnet-download
PS C:\Users\lala>
  
```

Exercice 1. : Faire la procédure précédente.**Comprendre les *namespace* (espace de nom)**

Les espaces de nom sont un élément de langage qui *permet d'organiser le code et de créer des noms de classes uniques*. Par exemple, vous décidez de créer une classe appelée **Report**. Les chances que d'autres compagnies de développement (ex. : [Infragistics](#), [XCEED](#)) puissent concevoir des classes qui portent le même nom sont possibles. Dans ce cas, comment gérer l'ambiguïté de ces noms? La solution est d'organiser le code dans un espace de nom. **Dans le domaine de la programmation .NET, il existe une convention commune qui consiste à donner le nom de la compagnie comme espace de noms.**

Voici des exemples avec la classe Report :

```
namespace Infragistics
{
    class Rectangle { /*...*/ }
}

namespace XCEED
{
    class Rectangle { /*...*/ }
}

namespace STHInformatique
{
    class Rectangle { /*...*/ }
}
```

Nous avons donc ici trois compagnies distinctes qui ont créé chacune leur propre classe **Rectangle**. Pour créer/instancier un objet Rectangle de la compagnie STHInformatique il faudra écrire :

```
STHInformatique.Rectangle rectangle1 = new STHInformatique.Rectangle();
```

Dans le cas de la compagnie XCEED :

```
XCEED.Rectangle rectangle2 = new XCEED.Rectangle();
```

Le *framework .NET* de Microsoft utilise délibérément les espaces de noms afin d'organiser toutes ses classes. Par exemple, l'espace de nom **System** regroupe toutes les classes de base. L'espace de nom **System.Data** rassemble les classes touchant l'accès des données.

Évidemment, l'utilisation d'espaces de nom peut demander au programmeur de saisir de longues lignes de code. C# résout ce problème grâce à l'utilisation de l'instruction **using** au début du fichier. Exemples :

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

Dans le cas de nos compagnies, si je décide d'utiliser la classe Report de la compagnie STHInformatique, je pourrais simplifier le code de la façon suivante :

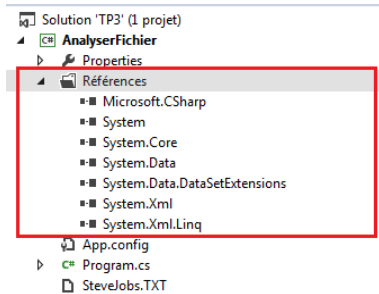
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using STHInformatique;

...

class Program
{
    static void Main(string[] args)
    {
        Rectangle rectangle = new Rectangle(); // Création du premier objet.
        ...
    }
}
```

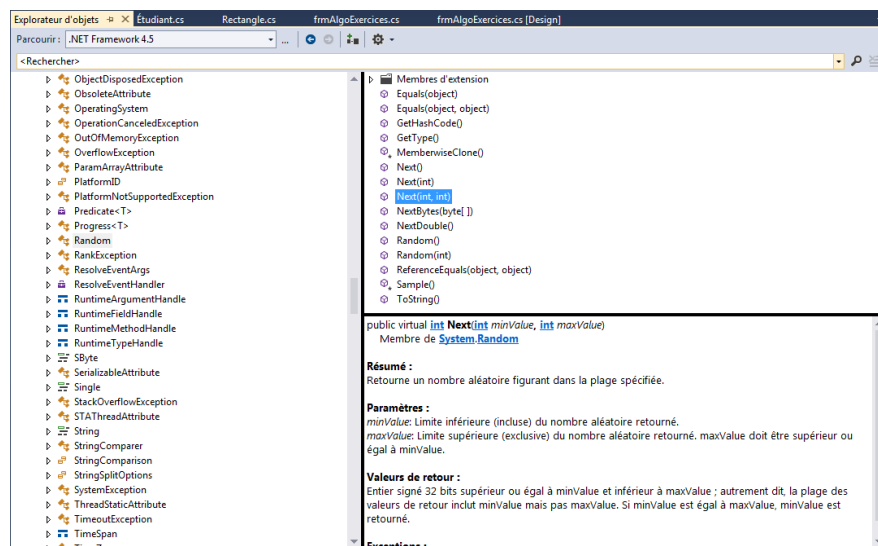
Références

Pour connaître les composantes (classes) externes qui sont rattachées à notre projet, il faut consulter les Références via l'explorateur de solutions.



Explorateur d'objets (Menu Affichage)

L'explorateur d'objets nous permet d'avoir accès aux espaces de noms, classes, méthodes, propriétés et, etc. des composantes incluses dans notre projet. Son accès se fait via le menu Affichage et Explorateur d'objets (CTRL+W et J). Suivre les explications du professeur pour son fonctionnement. Exemple d'écran :



Principe fondamental

Dorénavant, il faudra toujours appliquer le principe suivant dans la conception de nos applications : **séparer l'interface visuelle de la logique de programmation**. Ça implique quoi?

- Le minimum de code dans la classe Program (Console), les événements d'un formulaire (WPF) → **à faire le plus possible!!!**

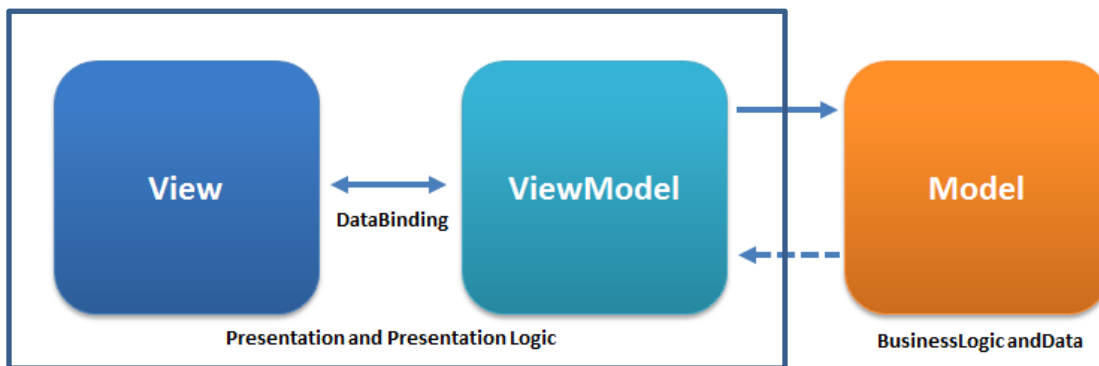
- Placer les algorithmes dans des classes spécialisées → **Encapsulation**.
Console → Méthodes (rôle d'affichage) appellent les membres de la classe pour obtenir les informations et s'en sert pour les afficher.
WPF → Les événements n'auront qu'à appeler les membres appropriés pour exécuter ces algorithmes.
- Éviter à tout prix (même interdit!) de gérer directement l'affichage d'informations dans les classes spécialisées.

Pourquoi?

- Pour faciliter l'évolution. En isolant ainsi les algorithmes dans des classes, il y est plus facile de faire évoluer l'application lors de changement de technologie.
 - Il y a belle lurette! → programmation en mode console
 - Hier → Programmation Windows Forms
 - Aujourd'hui → Programmation WPF (il y a en d'autres → MVC, ASP .Net)
 - Demain → Universal Windows Platform (UWP) → Windows 10 ???
- Pour partager les algorithmes entre les technologies (différentes plateformes). Exemple du jeu Tic Tac Toe : Les algorithmes de validation et d'intelligence artificielle (IA) **devraient être centralisés** dans une classe, peu importe la façon que c'est affiché. Cette classe (même code) pourrait se retrouver dans un projet de type :
 - Console (ligne de commande)
 - Windows Forms
 - WPF
 - MVC, Asp.net → Web

Structurer un projet selon MVVM (version légère)

Le patron MVVM (Model-View-ViewModel) consiste à de séparer la vue de la logique et de l'accès aux données en accentuant les principes de binding et d'événement.



(Wikipedia, 2022)

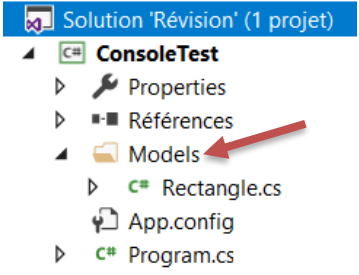
Ce principe sera vu en profondeur dans le cours de la technique. Concernant le cours actuel, vous serez initiés à deux des trois éléments de ce patron soit : **View** et **Model**. Description :

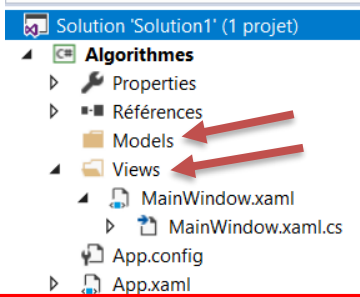
Élément	Description
Modèle (<i>Model</i>)	Élément qui contient les données ainsi que la logique en rapport avec les données: lecture et enregistrement. Le modèle représente l'univers dans lequel s'inscrit l'application. Par exemple pour une application de banque, le modèle représente des comptes, des clients, ainsi que les opérations telles que dépôt et retraits, et vérifie que les retraits ne dépassent pas la limite de crédit, etc.
Vue (<i>View</i>)	Une vue contient des éléments visuels ainsi que la logique nécessaire pour afficher les données provenant du modèle et/ou recevoir les entrées de données.
Vue-Modèle (<i>ViewModel</i>)	Cet élément sert à faire les liens (Data binding) entre les propriétés d'un modèle et la vue. Il peut être utilisé pour convertir, valider des données.

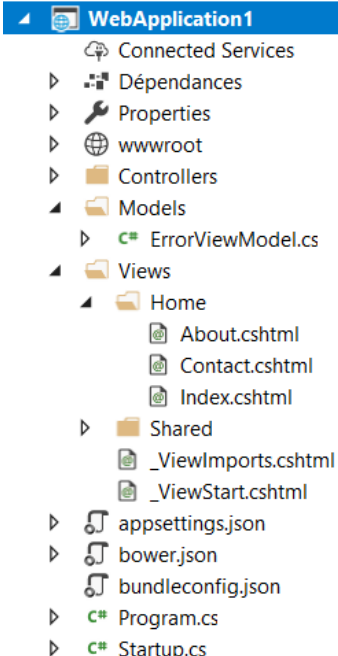
Dans nos projets (selon le type), nous ajouterons les dossiers :

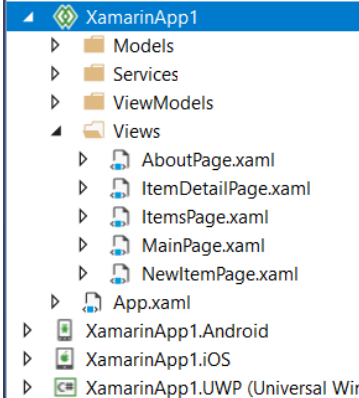
Dossier	Description
Models	Ce dossier contiendra les classes associées à la logique d'affaire de l'application. Ex. : Rectangles.cs, Clients.cs, etc. Souvent reliés aux bases de données.
Views (Windows Forms, WPF)	Ce dossier contiendra les formulaires définis pour l'application. Ex. : MainWindow.xaml, frmClient.xaml, etc. Il contiendra également les modèles de vues.

Voici des exemples de structures MVVM appliquées à différents types de projets Microsoft :

Projet Console :


Projet WPF (léger) :


Projet MVC :


Projet Xamarin :


Liens sur ce sujet :

- [The MVVM pattern](#)
- [Model-View-ViewModel \(wikipedia\)](#)

Bibliographie

Wikipédia. (2012, 10 23). *Framework*. Consulté le 11 25, 2012, sur Wikipédia:
<http://fr.wikipedia.org/wiki/Framework>

Wikipedia. (2022, 01 04). *Model–view–viewmodel*. Récupéré sur Wikipedia:
<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>