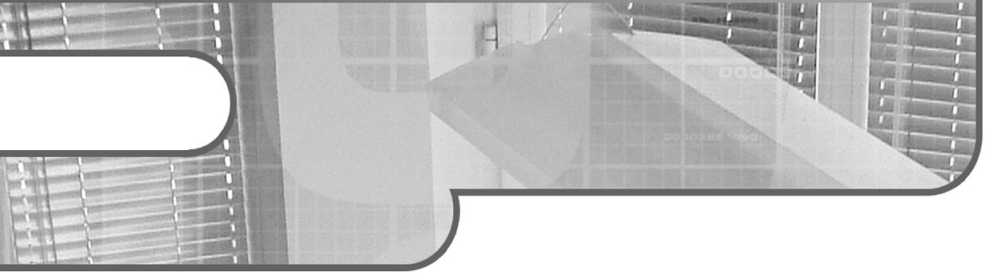


Chapitre 3
Les nouveautés d'ASP.NET Core

1. Introduction

ASP.NET existe depuis 2002 et bien des changements ont été effectués sur le framework depuis sa première version. Il faut rappeler une chose sur ASP.NET Core : la nouvelle plateforme web de Microsoft n'est en aucun cas une suite de la version 4.6 du framework que nous connaissions, mais bien un renouveau qui doit marquer une nouvelle ère de la technologie de Microsoft dans le Web moderne.

Certains diront que le framework n'a pas tant changé que cela (surtout la partie MVC), pourtant c'est bien "sous le capot" que les changements ont été les plus profonds, en commençant par le namespace `System.Web` qui n'existe plus. Ensuite, annoncé comme cross-platform, ASP.NET Core est plus modulable qu'il ne l'a été dans les années précédentes. Via ***NuGet*** pour les composants serveurs, puis via ***Grunt*** ou ***Gulp*** pour la partie cliente du site web, le nouveau framework bénéficie également d'un nouveau runtime, appelé ***CoreCLR***, permettant l'exécution d'une application web Microsoft sur Linux ou Mac.

2. Les nouveaux outils open source

ASP.NET Core arrive avec un tout nouveau panel d'outils open source permettant la gestion des nouveaux projets web. Fort heureusement pour le développement, l'ensemble de ces outils est réuni autour d'une seule et même interface en ligne de commande : dotnet.

2.1 L'environnement d'exécution dotnet

dotnet a été conçu afin de faire fonctionner des applications .NET de manière cross-platform sur Windows, Mac et Linux et ceci sans développer un runtime différent pour chaque plateforme. C'est à la fois un environnement d'exécution et un SDK embarquant ainsi tout ce qui est nécessaire pour le bon fonctionnement des applications web ASP.NET cross-platform.

Totalement orienté ***package-first***, Microsoft a poussé le concept de modularité très loin, permettant même à l'environnement d'exécution d'embarquer, de gérer et de créer de lui-même les packages dont il a besoin, et ceci de manière automatique via NuGet. dotnet est capable de cibler plusieurs frameworks (.NET Core ou le framework .NET Full), et ainsi générer les packages NuGet directement. De plus, dotnet embarque le nouveau moteur d'exécution CoreCLR conçu spécialement pour les problématiques de compatibilité sur les autres plateformes.

dotnet est intégré à Visual Studio 2015 pour offrir une expérience développeur enrichie, mais l'environnement d'exécution est également pilotable via les lignes de commande. Dans un projet ASP.NET Core, il est possible de rajouter des outils Microsoft et ainsi piloter son projet avec dotnet en ligne de commande. Ces outils se rajoutent en même temps que le paquet NuGet dans le fichier *.csproj* :

```
<ItemGroup>
    <DotNetCliToolReference Inclu-de="Microsoft.VisualStudio.Web.
CodeGeneration.Tools" Version="2.0.4" />
    <DotNetCliToolReference Include="BundlerMinifier.Core"
Version="2.0.238" />
  </ItemGroup>
```

© Editions ENI - All rights reserved

Les commandes déclarées permettent par exemple de lancer un outil de génération de code, anciennement appelé *scaffolding*. Ce dernier peut permettre de générer le contrôleur et les vues qui correspondent à un modèle de données bien précis. La commande est utilisable via `dotnet <command>` :

```
dotnet Microsoft.VisualStudio.Web.CodeGeneration.Tools
```

Au sein de l'application elle-même, il est possible d'utiliser les services de dotnet via certaines interfaces C# disponibles via injection de dépendances. On retrouve des services comme `IServiceEnvironment` permettant de modifier la configuration de l'environnement d'exécution pendant que l'application elle-même fonctionne.

L'utilitaire `dotnet` contient également une liste de commandes prédéfinies permettant d'effectuer certaines actions sur le projet ASP.NET Core :

– `dotnet new` : initialise un projet C# console simple.

– `dotnet restore` : restaure les dépendances du projet selon le *.csproj*.

– `dotnet build` : construit l'application .NET Core.

– `dotnet publish` : publie une application .NET Core.

– `dotnet run` : lance l'application depuis les sources.

– `dotnet test` : lance les tests utilisant le *test runner* défini dans le *.csproj*.

– `dotnet pack` : crée un paquet NuGet depuis le code source.

Cet outil supporte ainsi plusieurs processus de lancement de l'application ASP.NET Core. Le premier consiste simplement à restaurer les packages nécessaires à l'application, puis à lancer le projet depuis les sources.

```
dotnet new
dotnet restore
dotnet run
```

Cependant, l'utilitaire permet également de lancer l'application directement depuis une DLL après avoir lancé la génération du projet.

```
dotnet build
dotnet run bin/Debug/netcoreapp1.0/test-app.dll
```

Avec les deux processus ci-dessus, l'outil montre qu'il est capable d'exécuter plusieurs types d'applications :

– Des applications "portables", c'est-à-dire des applications qui dépendent de la version de .NET Core installée sur la machine. Cela veut dire que ce type d'application sera capable de fonctionner sur différentes installations de .NET Core, peu importe les systèmes d'exploitation utilisés. Les applications "portables" permettent également de centraliser les librairies de .NET : elles embarqueront donc uniquement les librairies externes.
– Des applications "autonomes", c'est-à-dire des applications contenant toutes les dépendances dont elles ont besoin pour fonctionner, incluant le runtime .NET Core faisant entièrement partie de l'application. Cela permet de faciliter le déploiement de l'application mais alourdit le package. De plus, il convient à l'application de spécifier la version du runtime à utiliser dans le *.csproj*.

2.2 L'utilitaire dotnet restore

Un projet web ASP.NET Core possède plusieurs dépendances à des packages externes provenant souvent de NuGet. Pour faire fonctionner une application web, il va tout d'abord falloir restaurer les packages nécessaires au bon fonctionnement du projet. C'est une des nouveautés d'ASP.NET Core : le projet embarque uniquement ce dont il a besoin, et donc il a fallu concevoir un outil permettant de restaurer ces dépendances, ceci de manière cross-platform et totalement transparente. dotnet restore permet de faire cette restauration.

Intégré à .NET Core et installé avec dotnet, dotnet restore permet de scruter le projet ASP.NET Core et de récupérer les packages dont il a besoin dans le cloud. Visual Studio 2015 utilise automatiquement cet utilitaire lorsque les dépendances du projet sont mises à jour. Il est cependant possible de lancer le processus à la main grâce à la commande suivante :

```
> dotnet restore
```

© Editions ENI - All rights reserved

Remarque

Il faut noter toutefois que, dans la console depuis laquelle la commande est lancée, il faut se placer à la racine du projet. En effet, `dotnet restore` *va inspecter le fichier .csproj pour identifier les dépendances à récupérer.*

Grâce à dotnet restore et à sa simplicité d'utilisation, il est très facile de restaurer les dépendances d'un projet ASP.NET Core, et ceci indépendamment de la plateforme.

2.3 Gérer ses paquets NuGet avec dotnet pack

L'outil `dotnet pack` est utilisé pour générer un package NuGet à partir d'un projet .NET. Il peut être utilisé de différentes manières pour gérer ses paquets NuGet. Dans un premier temps, nous pouvons utiliser la commande suivante, à la racine d'un projet .NET, afin de simplement générer un paquet NuGet :

```
dotnet pack mon_projet.csproj
```

Cette commande générera un package NuGet à partir du projet spécifié et l'enregistrera dans le répertoire `bin\Debug` ou `bin\Release`, selon le mode de compilation utilisé.

Afin d'inclure les informations de version et de métadonnées dans le package NuGet généré, nous pouvons utiliser les options `--version` et `--metadata` :

```
dotnet pack mon_projet.csproj --version 1.2.3
--metadata authors="John Doe"
```

Dans cet exemple, nous avons spécifié la version `1.2.3` du package et ajouté une métadonnée `authors` avec la valeur "`John Doe`".

Pour publier le package NuGet généré sur un dépôt NuGet, nous pouvons utiliser l'option `--publish` en spécifiant l'URL du dépôt :

```
dotnet pack mon_projet.csproj --publish https://mynugetrepo.com
```

Cette commande publiera le package NuGet généré sur le dépôt NuGet spécifié.

Pour plus d'informations, nous pouvons consulter la documentation officielle ou utiliser la commande `dotnet pack --help` pour afficher la liste complète des options disponibles :

- `--output` : répertoire de sortie pour les paquets générés.
- `--no-build` : génère un paquet NuGet sans lancer la génération du projet .NET.
- `--no-restore` : génère un paquet NuGet sans lancer la restauration des paquets NuGet du projet.
- `--include-symbols` : inclue les symboles de compilations à côté du paquet généré dans le dossier de sortie.
- `--include-source` : inclut les fichiers PDB et les fichiers sources. Les sources iront dans le dossier `src`.
- `--serviceable` : définit le niveau de maintenance du paquet.
- `--nologo` : n'affiche pas le logo lors du démarrage de la commande.
- `--interactive` : permet d'attendre ou non une interaction utilisateur si nécessaire.
- `--verbosity` : indique le niveau de verbosité des logs affichés lors du lancement de la commande.
- `--version-suffix` : définit la valeur de la propriété `$(VersionSuffix)` à utiliser lors de la génération du projet.
- `--configuration` : définit la configuration à utiliser lors de la génération. Les valeurs peuvent être `Debug` ou `Release`.
- `--use-current-runtime` : définit s'il faut utiliser le runtime actuel comme runtime cible.

La commande `dotnet pack` est très utile pour la gestion des paquets NuGet des projets .NET et permet au développeur de créer et publier des paquets facilement.

© Editions ENI - All rights reserved

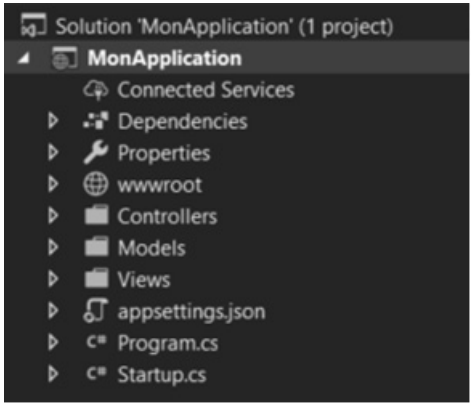
3. La structure d'une solution

Une solution ASP.NET Core est la base d'un projet web utilisant les technologies Microsoft. Elle permet rapidement de déployer un site et de structurer le code utilisé pour faire fonctionner l'application. Une solution peut comporter à la fois le code côté serveur et le code côté client, tout en incluant des mécanismes afin de bien séparer les deux parties. Ce chapitre va traiter des différents composants d'une solution ASP.NET Core et expliquer leurs rôles dans la configuration ou le déploiement de l'application web.

3.1 Les fichiers .csproj

Un projet ASP.NET Core met en œuvre une toute nouvelle philosophie de conception d'applications web issue de chez Microsoft s'inspirant beaucoup de l'open source.

Le nouveau template ressemble à ceci :



Nouveau template de projet ASP.NET Core

Partie 3 : Les CSS 3

Chapitre 3-1
Intégrer les styles CSS

1. Le rôle des CSS

Rappelons que les **CSS**, *Cascading Style Sheets*, permettent de mettre en forme le contenu des éléments HTML et s'occupent aussi de la mise en page des sites web. Avec les CSS, nous avons donc deux objectifs bien définis :

– La mise en forme qui concerne le formatage des textes, avec par exemple l'application d'une couleur, le changement de casse des caractères, la mise en évidence des paragraphes de texte avec la mise en forme de l'interligne ou le retrait de première ligne…

– La mise en page concerne l'agencement des blocs de contenu HTML de la page, avec une barre de navigation, un en-tête, un pied de page, une zone d'affichage des contenus…

  Les CSS permettent de modifier dynamiquement la mise en page selon le média de diffusion. C'est-à-dire que la mise en page va être différente suivant que vous consultez le site sur l'écran d'un ordinateur, d'une tablette ou d'un smartphone.

Rappelons que les CSS3 sont divisées en modules indépendants et qu'elles sont élaborés avec leur propre vitesse.

Il est maintenant bien acquis, et depuis longtemps déjà, qu'il faut séparer les CSS de la structure du HTML. Nous allons donc voir dans ce chapitre où placer les règles CSS qui constituent une feuille de style.

2. Les styles intégrés dans un élément HTML

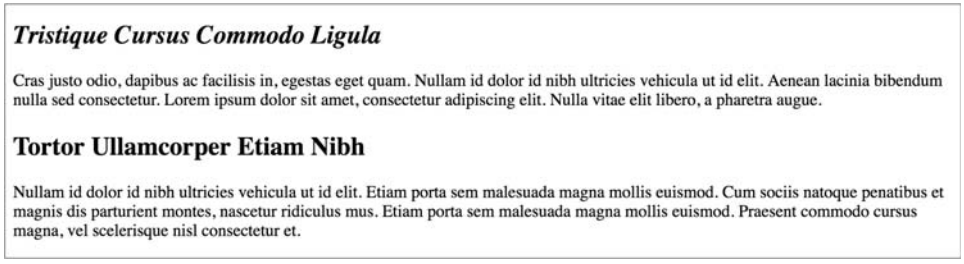
La première possibilité d'intégrer des styles CSS est de définir une règle directement dans l'élément HTML concerné. Dans ce cas, cette règle ne s'applique qu'à cet élément et à nul autre.

Pour cela, nous devons utiliser l'attribut `style` dans l'élément HTML souhaité et indiquer la propriété et la valeur voulues.

Dans cet exemple, nous appliquons de l'italique au premier titre `<h2>` :

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Ma page web</title>
</head>
<body>
   <h2 style="font-style: italic">Tristique Cursus Commodo Ligula</h2>
   <p>Cras justo odio, dapibus ac facilisis...</p>
   <h2>Tortor Ullamcorper Etiam Nibh</h2>
   <p>Nullam id dolor id nibh ultricies...</p>
</body>
</html>
```

Voici l'affichage obtenu :



Ainsi, vous voyez que la portée de cette règle CSS est des plus réduites, puisqu'elle ne s'applique qu'à l'élément HTML dans lequel elle est définie. Elle ne peut pas être utilisée ailleurs.

© Editions ENI - All rights reserved

3. Les styles définis dans la page

La deuxième possibilité est de définir les règles CSS dans une page HTML. Dans ce cas, les règles ne peuvent s'appliquer qu'aux éléments HTML contenus dans cette page HTML et nulle part ailleurs, dans aucun autre fichier HTML. Les règles CSS se définissent dans l'élément `<head>` dans un élément `<style>`.

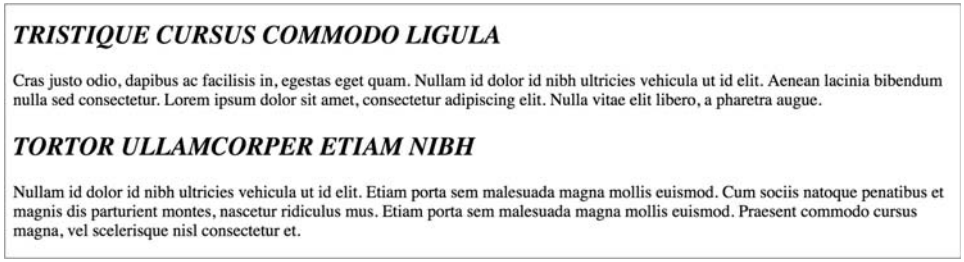
Voici un exemple :

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Ma page web</title>
   <style>
       .titre-article {
           font-style: italic;
           text-transform: uppercase;
       }
   </style>
</head>
<body>
   <h2 class="titre-article">Tristique Cursus Commodo Ligula</h2>
   <p>Cras justo odio, dapibus ac facilisis...</p>
   <h2 class="titre-article">Tortor Ullamcorper Etiam Nibh</h2>
   <p>Nullam id dolor id nibh ultricies...</p>
</body>
</html>
```

Dans l'élément `<style>`, nous n'avons qu'une seule règle CSS et elle est nommée `.titre-article`. Cette règle définit un texte en italique et passe en majuscules tout le texte sur lequel elle sera appliquée.

Dans la page HTML, dans les deux éléments `<h2>`, nous appliquons cette règle avec l'attribut `class`.

Voici l'affichage obtenu :



Vous concevez donc parfaitement que la portée des règles CSS est plus importante que précédemment, puisqu'elles peuvent s'appliquer à tous les éléments HTML présents dans la page web.

4. Les styles définis dans un fichier .css

La troisième possibilité est de déclarer les règles CSS dans un fichier séparé des pages HTML. Ce fichier aura alors comme extension **.css**. Ce fichier ne contiendra rien d'autre que les règles CSS voulues. Ensuite, nous devrons lier ce fichier CSS aux pages HTML voulues pour appliquer les règles CSS aux éléments HTML voulus.

Voici un exemple très simple d'un fichier CSS, nommé **styles.css**, qui ne contient que deux règles CSS :

```
.titre-article {
   margin-left: 20px;
   padding-left: 10px;
   border-left: 5px solid #000;
}

.nom {
   font-style: italic;
}
```

© Editions ENI - All rights reserved

Voici le fichier HTML :

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Ma page web</title>
   <link href="styles.css" rel="stylesheet">
</head>
<body>
   <h2 class="titre-article">Tristique Cursus Commodo Ligula</h2>
   <p>Cras justo odio ... id nibh <span class="nom">Ultricies
vehicula</span> ut id elit...</p>
   <h2 class="titre-article">Tortor Ullamcorper Etiam Nibh</h2>
   <p>Nullam id dolor ... <span class="nom">Etiam porta</span>
sem malesuada magna mollis euismod...</p>
</body>
</html>
```

La liaison entre le fichier HTML et le fichier CSS se fait dans l'élément `<head>` . C'est l'élément `<link>` qui est utilisé :

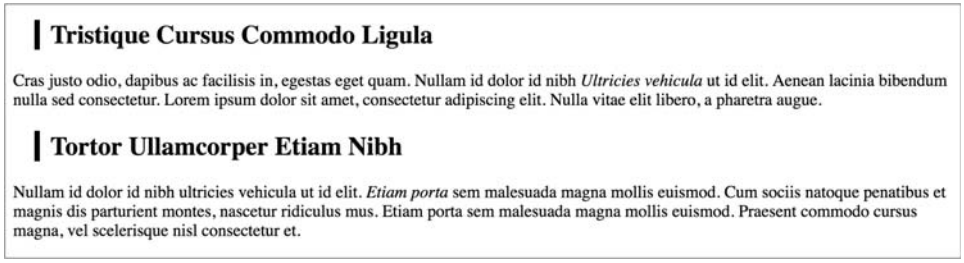
– L'attribut `href="styles.css"` indique le chemin d'accès au fichier CSS.

– L'attribut `rel="stylesheet"` précise le type de relation. Ici, il s'agit d'une relation à une feuille de style.

– L'attribut `type="text/css"` est facultatif car le type les feuilles de style est par défaut CSS.

L'application des styles se fait de nouveau avec l'attribut `class` dans les éléments HTML voulus.

Voici l'affichage obtenu :



Vous pouvez bien sûr lier plusieurs fichiers CSS à une même page HTML et un même fichier CSS peut être lié à plusieurs pages HTML. Vous comprenez bien que la portée des styles CSS est ici la plus importante.

5. Les styles importés

La dernière possibilité d'intégrer une feuille de style CSS est d'utiliser la règle `@import`. Attention, notez que cette règle a été implémentée dans les CSS 2. Il ne s'agit donc pas d'une règle HTML ni d'un élément HTML.

Cette règle permet d'importer un fichier .css dans un autre fichier .css. Voici sa syntaxe qui se place usuellement au début d'un fichier .css :

```
@import url("autres-styles.css")
```

La règle `@import` utilise la propriété `url()` pour indiquer le chemin d'accès au fichier .css qui doit être importé.

Vous pouvez aussi utiliser la règle `@import` dans un fichier .html, dans l'élément `<head>`. Voici la syntaxe à utiliser :

```
<style>
   @import url("autres-styles.css");
</style>
```

Mais pour des raisons de performance, cette méthode est moins utilisée.

© Editions ENI - All rights reserved

Chapitre 3-2
Définir les styles CSS

1. La structure d'une règle de style

1.1 La terminologie des CSS

Nous parlerons de CSS, de style, de règle, de déclaration, de propriété et de valeur. Il convient de bien définir cette terminologie afin d'utiliser les bons termes.

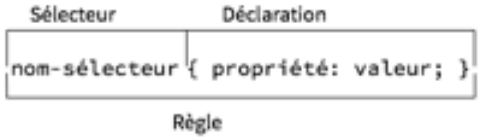
Les **CSS**, pour *Cascading Style Sheets*, sont une technologie développée par le W3C qui permet de mettre en forme et de mettre en page les pages des sites web structurés en HTML.

Un **style** est une mise en forme, ou une mise en page utilisant les CSS, qui est mémorisée et qui, ensuite, peut être appliquée à un ou plusieurs éléments HTML.

Une **règle** CSS permet la création d'un style. Cette règle est créée avec une syntaxe précise utilisant un **sélecteur** et une **déclaration**. Cette dernière est constituée de **propriétés** et de **valeurs**.

1.2 Définir une règle de style

Un style CSS est bâti avec une règle. Cette règle est constituée de plusieurs parties. Voici un petit schéma illustrant une règle CSS :



– La **règle** est constituée d'un sélecteur et d'une déclaration.

– Le **sélecteur** indique la portée du style, c'est-à-dire sur quel élément HTML peut s'appliquer le style créé. Il existe beaucoup de sélecteurs, nous les étudierons dans un prochain chapitre.

– La **déclaration** est indiquée entre accolades.

  C'est dans cette déclaration que sont indiquées la ou les **propriétés** CSS utilisées. Chaque propriété utilise une ou plusieurs **valeurs**. La propriété est séparée de la ou des valeurs par le caractère deux-points **:**. Chaque ligne dans la déclaration se termine par le caractère point-virgule **;**.

Pour plus de visibilité et de lisibilité, il est d'usage d'aller à la ligne après l'accolade ouvrante et avant l'accolade fermante. Ainsi, chaque couple propriété/valeur se trouve sur une seule ligne.

Voici une règle CSS n'utilisant aucun espace ni retour à la ligne. La visibilité n'est pas au rendez-vous :

```
nom-selecteur{propriete1:valeur1;propriete2:valeur2;propriete3: valeur3;}
```

Voici une règle CSS utilisant plusieurs lignes dans sa déclaration :

```
nom-selecteur {
   propriete1: valeur1 ;
   propriete2: valeur2 ;
   propriete3: valeur3 ;
}
```

© Editions ENI - All rights reserved

Chapitre 3
Programmation orientée objet

1. Principes de la programmation orientée objet

La programmation orientée objet (POO) est un paradigme très répandu en développement logiciel. Il vient compléter un panorama déjà riche du paradigme procédural ainsi que du paradigme fonctionnel.

La POO est une forme de conception de code visant à représenter les données et les actions comme faisant partie de classes, elles-mêmes devenant des objets lors de leur création en mémoire. Cette notion a été rapidement présentée dans le chapitre précédent, il est maintenant temps de comprendre son fonctionnement plus en détail.

1.1 Qu'est-ce qu'une classe ?

Une classe est un élément du système que forme votre application. Une classe contient deux types d'élément de code : des données ainsi que des méthodes, représentant des actions. Il faut voir la classe comme étant une boîte dans laquelle il est possible de ranger ces deux types d'éléments. Pour faire un parallèle avec la vie réelle, nous pouvons facilement comprendre que la définition d'une classe s'applique à un objet comme un ordinateur, par exemple. Ce dernier dispose de méthodes (allumer, éteindre...) ainsi que des propriétés (nombre d'écrans, quantité de RAM...).

Conceptuellement, une classe n'est qu'une définition. Une fois que vous avez statué sur ce qu'elle doit contenir ainsi que ses méthodes, il convient de la créer. Cette action s'appelle l'instanciation. À la suite de cette opération, nous obtenons une instance en mémoire d'un objet.

Pour tenter une comparaison, prenons l'exemple d'une usine de fabrication d'objets en bois. Afin de pouvoir créer un objet, il faut un plan (la classe). Grâce à ce dernier, la machine peut découper et assembler les divers éléments (les données et méthodes) afin de créer une nouvelle instance (instanciation).

En C#, la déclaration d'une classe se fait grâce au mot-clé `class`. Il y a quelques spécificités possibles, notamment la portée, que nous étudierons juste après, dans la section Que peut-on déclarer dans une classe ? - Les méthodes, ainsi que les concepts de `static`, `sealed` et celui de `partial`. La syntaxe complète de la déclaration d'une classe est la suivante :

```
PORTÉE [static] [sealed] [partial] class NOM_CLASSE
```

Le nom de la classe est libre mais répond à deux règles :

– Il ne peut contenir que des caractères alphanumériques et le signe underscore (« _ »).

– Il ne peut pas commencer par un chiffre.

En plus de ces règles, les développeurs C# respectent souvent une convention syntaxique : l'utilisation du PascalCase. Cela indique que le nom commence par une majuscule et chaque mot est symbolisé par une majuscule également, par exemple, `OrdinateurPortable`. Le langage et le compilateur n'interdisent pas d'écrire `ordinateurPortable`, `Ordinateurportable` ou encore `ordinateurportable`, mais ces différentes déclarations ne respectent pas la convention largement admise et appliquée. Finalement, bien que ce soit possible, il est recommandé d'éviter tout caractère accentué dans le nom d'une classe. Par exemple, il est préférable d'appeler sa classe `Pieton` plutôt que `Piéton`, cela afin que le code C# produit soit le plus proche possible de ce que nous aurions en anglais.

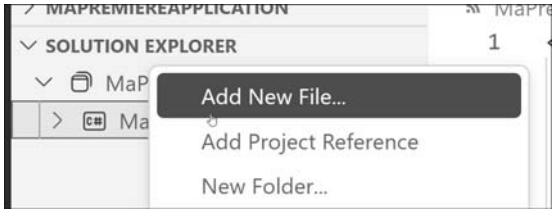
© Editions ENI - All rights reserved

Dans le programme de base créé en C# dans le précédent chapitre, une classe `Program` est créée par défaut. Nous pouvons constater qu'il n'y a ni notion de portée ni notion de `partial` ou `static`. Une fois qu'une classe est déclarée, elle définit un bloc, dans lequel nous pouvons implémenter les données et les méthodes dont notre programme a besoin pour fonctionner.

1.1.1 Les classes dans Visual Studio Code

Pour créer une classe dans Visual Studio Code, il est nécessaire de suivre les étapes suivantes :

- Dépliez la vue **Solution Explorer** du projet.
- Placez-vous sur le dossier où vous souhaitez créer la nouvelle classe (ou directement sur le nom du projet si vous souhaitez la créer à la racine).
- Faites un clic droit pour sélectionner l'élément de menu **Add New File**.
- Renseignez le nom de la classe dans la petite pop-up qui s'est ouverte en haut au centre de l'écran (toujours sans espaces ni caractères spéciaux).



Ajout d'une nouvelle classe avec Visual Studio Code

À la suite de ces manipulations, un nouveau fichier, portant le nom de la classe suivi de l'extension .cs, est disponible dans la hiérarchie à gauche. Par défaut, ce fichier sera ouvert et contient la classe qui a été déclarée dans l'espace de noms correspondant au dossier de destination.

1.1.2 L'héritage

Il existe un concept extrêmement important en POO : l'héritage. Globalement, si vous avez la possibilité de dire « X est un Y », l'équivalent pourrait être de dire « X hérite de Y ». X reprend toutes les propriétés et tous les comportements de Y, mais le spécifie. Donnons un exemple concret : « Un Mac est un ordinateur ». Donc, au niveau du développement orienté objet, un Mac reprend toutes les propriétés d'un ordinateur ainsi que ses comportements, mais les spécifie en y apportant ses propres éléments. On dit dans ces cas-là que `Mac` est une classe fille de la classe `Ordinateur`.

En C#, cette notion est centrale car tous les éléments que vous allez manipuler héritent naturellement de la classe `System.Object`, qui définit le comportement de base de n'importe quel objet. De surcroît, contrairement à d'autres langages (comme le C++), il n'est pas possible, en C#, d'hériter de plusieurs classes : une seule classe mère est possible. Si aucune classe mère n'est spécifiée, c'est par définition la classe `System.Object` qui constitue la classe mère (sans qu'une quelconque manipulation soit requise).

Remarque

En C#, il n'est pas possible d'hériter de plusieurs classes. Il faut donc choisir la classe dont on hérite. En l'absence de précision, le compilateur génère automatiquement, de façon transparente, un héritage de la classe `System.Object`, comme décrit ci-dessus. Si nous spécifions un héritage, cela ne veut pas dire que la classe hérite de `System.Object` ET de la classe héritée, mais uniquement de la classe héritée, qui remplace l'héritage généré par le compilateur. La classe héritée, elle-même, hérite soit d'une autre classe, soit directement de `System.Object`. En finalité, toutes les classes en C# héritent d'une façon ou d'une autre de `System.Object`.

Afin d'indiquer qu'une classe hérite d'une autre, il faut utiliser le deux-points, suivi de la classe dont on souhaite hériter :

```
class Ordinateur { }
class Mac : Ordinateur { }
```

Bien entendu, ce n'est pas parce qu'une classe hérite d'une autre qu'elle a forcément accès à tout ce qui a été défini au sein de la classe mère.

© Editions ENI - All rights reserved

1.1.3 L'encapsulation

Tout ce qui se trouve à l'intérieur d'une classe est désigné par un terme bien spécifique : l'encapsulation. Avec celle-ci vient également la notion de portée, qui indique comment les choses sont perçues d'un point de vue extérieur à la classe.

La portée permet de définir la visibilité d'un élément d'une classe ou de la classe elle-même. Il existe en tout sept portées en C# :

- `public` : définit que l'élément est totalement visible dans et en dehors de la classe.
- `private` : définit que l'élément n'est visible qu'au sein de la classe où il est déclaré alors qu'il est totalement invisible de l'extérieur.
- `internal` : définit que l'élément est visible uniquement au sein du projet où il est déclaré. Nous pouvons considérer l'élément comme étant public, mais simplement au sein du projet dans lequel il est déclaré. Un autre projet qui référence notre projet n'a pas connaissance d'un élément déclaré comme `internal`. Par défaut, en l'absence de portée explicite sur une classe, c'est la portée `internal` qui est sélectionnée par le compilateur.
- `protected` : définit que l'élément est visible uniquement au sein de la classe où il est déclaré ainsi que dans sa hiérarchie de classes filles. Cela rejoint le concept de l'héritage, que nous verrons plus loin dans ce chapitre.
- `protected internal` : définit un cumul entre `protected` et `internal`. Un élément déclaré avec cette portée est visible par la classe concernée ainsi que ses classes filles, tout comme par toutes les autres classes au sein du même projet. Cela signifie également que si une classe fille est déclarée en dehors du projet actuel, elle peut accéder à un élément `protected internal`, tout comme n'importe quelle classe du même projet.
- `private protected` : définit une intersection entre `protected` et `internal`. Un élément déclaré avec cette portée n'est visible que par la classe concernée ainsi que les classes filles qui sont définies au sein du même projet. Cela veut dire qu'une classe fille définie en dehors du projet actuel ne pourra pas accéder à cet élément.

– `file` : ajoutée en C# 11, cette portée définit une visibilité uniquement dans le cadre du fichier en cours. Cette portée est très particulière car elle n'a pas vocation à être directement utilisée par les développeurs. Elle existe surtout pour les outils de génération automatique de code. Néanmoins, dans de très rares cas, il peut être utile de déclarer un élément qui n'existe que dans le cadre d'un fichier pour un algorithme précis. Il est à noter également que, contrairement aux autres portées, cette portée n'est valide que pour la déclaration d'un type. Elle ne peut pas s'appliquer sur une méthode, un champ ou une propriété.

Avec toutes ces portées, il est possible de créer la classe qui correspond finement au besoin de votre application, pour éviter que certains éléments ne sortent du périmètre de la classe. En reprenant notre exemple, considérons que la classe `Ordinateur` dispose d'un booléen indiquant si la machine est allumée ou non. Afin d'éviter que quelqu'un ne puisse manipuler directement cette donnée, la manière de procéder est de la définir comme étant publiquement accessible en lecture, mais privée pour ce qui est de l'écriture. En conséquence, seule une méthode publique, définie dans cette classe, comme par exemple `Allumer` ou `Eteindre`, peut changer la valeur de cet indicateur. Nous nous préservons ainsi d'un changement d'état non maîtrisé (car nous pouvons considérer que l'opération d'extinction nécessite d'effectuer quelques opérations en amont avant de basculer le booléen).

1.2 Que peut-on déclarer dans une classe ?

Nous l'avons vu, il existe deux types d'éléments que nous pouvons déclarer dans une classe : des méthodes (actions) et des données. Voyons rapidement comment les déclarer.

1.2.1 Les méthodes

Une méthode traduit une action qu'il est possible d'invoquer sur la classe. Lors de la déclaration d'une méthode, il faut se poser les questions suivantes :

– S'agit-il d'une action qui doit pouvoir être réalisée depuis l'extérieur ou uniquement depuis l'intérieur de la classe ?

– Est-ce qu'une valeur de retour particulière est attendue ?

© Editions ENI - All rights reserved

– Certaines informations sont-elles nécessaires pour que cette méthode fonctionne ?

Vous avez déjà eu un aperçu d'un appel de méthode dans le premier chapitre, sur la classe `Console` : `WriteLine` et `ReadLine`. Ces deux méthodes illustrent bien les points cités précédemment :

– `WriteLine` doit pouvoir être appelée depuis l'extérieur. Nous n'attendons pas de valeur en retour à son appel, mais il est nécessaire de lui transmettre l'information que nous souhaitons écrire.
– `ReadLine` doit également pouvoir être appelée depuis l'extérieur. Nous avons besoin de récupérer l'information saisie par l'utilisateur uniquement, sans besoin de lui transmettre une quelconque information.

La syntaxe de déclaration d'une méthode dans une classe est la suivante :

```
PORTÉE [static] TYPE_RETOUR NOM_METHODE([PARAMÈTRES])
```

Le type de retour doit correspondre à un type C# connu. Par exemple, si nous souhaitons créer une méthode qui réalise l'addition de deux nombres et renvoie le résultat, le tout accessible publiquement, nous la déclarons comme suit :

```
public int Addition(int premier, int second) {}
```

Remarque

Dès lors que nous déclarons une méthode avec une valeur de retour sans écrire le contenu de la méthode, le compilateur émet immédiatement une erreur de compilation. Ceci est dû au fait que chaque méthode retournant un résultat doit obligatoirement comporter une instruction `return`.

Lorsqu'une méthode doit renvoyer une valeur, il faut utiliser le mot-clé `return` afin de définir la valeur que nous souhaitons renvoyer. L'instruction `return` peut être utilisée directement avec une valeur ou alors nous pouvons nous servir d'une variable du type de retour attendu. Dans le cas de l'exemple ci-dessus, ces deux façons d'écrire la méthode sont valides :

```
public int Addition(int premier, int second)
{
    return premier + second;
}
```

```
public int Addition(int premier, int second)
{
    int resultat = premier + second;
    return resultat;
}
```

Un élément important à garder en mémoire : à l'instar de ce que nous avons vu dans le chapitre précédent avec la déclaration de classes du même nom au sein du même espace de noms, il n'est pas possible de déclarer deux fois la même méthode à l'intérieur d'une même classe. Si les noms sont identiques et que les paramètres le sont également, alors le compilateur C# considère qu'il s'agit de la même méthode. La valeur de retour ne constitue pas un élément distinctif. Ainsi, la déclaration des deux méthodes suivantes dans la même classe est impossible et cela provoque une erreur de compilation :

```
public int Addition (int premier, int second)
{
    return premier + second;
}
public void Addition (int premier, int second)
{
}
```

Remarque

Comme nous pouvons le constater dans l'exemple ci-dessus, le mot-clé void précise que la méthode ne renvoie aucun résultat. La notion de type de retour étant obligatoire, il faut utiliser ce mot-clé pour indiquer les cas où il n'y en a pas.

Si la méthode ne prend pas de paramètres, la présence de parenthèses ouvrantes et fermantes accolées au nom de la méthode est malgré tout nécessaire pour signifier qu'il s'agit d'une méthode :

```
public void MaMethode()
{
}
```

Au sein d'une méthode qui déclare son propre bloc, il est possible de déclarer des variables et constantes qui sont considérées uniquement comme locales (c'est-à-dire visibles au sein de la méthode et de tous ses sous-blocs, mais invisibles dans les blocs parents, directs ou indirects).

© Editions ENI - All rights reserved