

- Si vous ouvrez des sockets de connexion, fermez toujours ceux-ci dans un bloque `Finally`, cela garantit la fermeture en cas d'exceptions;
- Déclarez les variables le plus près possible de leur utilisation. Une seule déclaration par ligne;

## Commentaires

De bons et utiles commentaires vont donner au code une meilleure maintenabilité.

- Ne pas commenter chaque ligne de code ni chaque variable déclarée;
- Écrire des commentaires uniquement là où ils sont nécessaires. Mais un code correctement lisible ne demandera que peu de commentaire;
- Si vous avez une partie de code complexe, documenter la correctement avec suffisamment d'information.
- Si vous initialisez une variable numérique à une autre valeur que 0, documentez la raison de cette nécessité

## Exceptions

Utilisez `Throw` sans spécifier l'exception d'origine, cela préserve le call stack, (pile d'appel)

# VB.Net

## Standard et bonne méthode

Par Dany Côté

7

## La casse des acronymes

Les acronymes courts (1 ou 2 caractères) doivent être en majuscules et les acronymes long (3 caractères et plus) doivent être considérés comme un mot normal.

Exemple les classes: `IORead` et `XmlReader`

**Attention :** Inclure des acronymes que s'ils sont connus et bien compris par tous.

Il coûte moins cher d'augmenter la puissance d'une machine tous les 5 ans plutôt que de devoir dépenser du temps et de l'argent pour identifier les problèmes, essayer de comprendre le code et ce tout au long du cycle de vie de l'application.

## Convention de nommage et standard

Il existe deux conventions d'écritures plus célèbres que les autres. Les méthodes Pascal et Camel.

**Écriture Pascal** – Le 1<sup>er</sup> caractère de chaque mot est en majuscule, les autres en minuscules.

Exemple : `BackColor`

**Écriture Camel** – Le 1<sup>er</sup> caractère de chaque mot sauf le 1<sup>er</sup> mot est en majuscule, les autres sont en minuscules.

Exemple : `backColor`

La meilleure méthode est de mélanger les deux écritures pour se faire une règle de nommage.

## Présentation du code

- Utiliser TAB pour l'indentation Ne pas utiliser SPACES;
- Définir la taille de tabulation à 2;
- Les commentaires doivent être au même niveau d'indentation que le code;
- Utiliser une ligne blanche pour séparer le groupe de code commenté;
- Il doit y avoir 1 ligne et 1 seule entre les méthodes d'une classe;

## Du « bon » code

Tout le monde peut écrire du code, mais la majorité des développeurs réalisent du code qui « fonctionne », mais pas du « bon » code. Produire cela, c'est un art que vous devez apprendre et pratiquer.

## La technique

Tout le monde peut avoir une définition différente de ce qu'est un bon code. Pour nous, un bon code doit respecter les critères suivants :

- Réutilisabilité
- Maintenabilité
- Efficacité

Certains programmeurs exploitent trop le dernier critère qu'est l'efficacité, au détriment de la réutilisabilité et de la maintenabilité. Si l'on considère le retour sur investissement à long terme, efficacité et performance sont moins importantes que la réutilisabilité et la maintenance.

1

## Règle de nommage

- Utiliser l'écriture Pascal pour les noms de Classes, de fonctions (procédures, méthodes, propriétés);
- Utiliser l'écriture Camel pour les Variables et paramètres de fonction;
- Utiliser le préfixe « I » avec l'écriture Pascal pour les Interfaces Ex. : IEntity;
- Préfixez les variables, propriétés ou fonctions retournant un boolean par « is »;

**Attention :** Ne pas utiliser la notation hongroise. Anciennement, les programmeurs aimaient avoir le type de donnée dans le préfixe et utilisaient m\_ comme préfixe des variables membre (variables privées). Maintenant, ce n'est plus recommandé dans le standard de programmation .NET. Il est toutefois recommandé de définir les variables privées relatives à une classe en utilisant le préfixe « \_ »

3

## Bonne programmation

- Le nom d'une fonction doit décrire ce que celle-ci fait;
- Une méthode doit faire qu'un seul travail, même si celui-ci est très court;
- Toujours tester toutes les valeurs possibles;
- Convertir les chaînes de caractère en minuscules avant de les comparer Sauf si la casse est importante;
- Utiliser `String.Empty` au lieu de ""  
Ex: `if (name = String.Empty) Then;`
- Utiliser Enum lorsque c'est nécessaire, ne pas utiliser de nombre ou de caractère pour différencier les éléments;
- Ne pas exposer les variables à l'extérieur, Exposez une propriété;
- Ne pas mettre de code compliqué dans un événement, Appelez une fonction qui fait le travail;
- Éviter de définir trop de paramètres pour une méthode, Si il y en a plus de 5, envisagez de créer une classe ou une structure et de passer celle-ci;
- Si vous avez une méthode qui retourne une collection, retournez celle-ci vide plutôt que `Nothing`;

6

Identificateur	Exemple
Classe	<code>Public Class <b>HelloWorld</b></code>
Type d'énumération	<code>Enum <b>NiveauErreur</b></code>
Valeurs d'énumération	<code>Enum NiveauErreur <b>FatalError</b> End Enum</code>
Évènement	<code>Private Event <b>ValueChanged</b></code>
Interface	<code><b>IDisposable</b></code>
Méthode	<code>Public Function <b>ToString</b>() as String</code>
Paramètre	<code>Sub S (<b>typeName</b> as String)</code>
Propriété	<code>Public Property <b>BackColor</b></code>
Variable	<code>Dim <b>pleinePage</b> As String = 0</code>
Variable globale	<code>Dim <b>_pleinePage</b> As String</code>

Exemple:

```
Dim _nombreMaximal As Integer = 0
Sub DireBonjour(ByVal nom As String)
    Dim pleinePage As String
    pleinePage = "Bonjour " & name
End Sub
```

4