REGLES DE BONNES PRATIQUES DE DEVELOPPEMENT

1. DOCUMENTATION ET COMMENTAIRES

Introduction

La documentation est essentielle à la compréhension des fonctionnalités du code. Elle peut être intégrée directement au code source, tout en restant aisément extractible dans un format de sortie tel que HTML ou PDF. Cette intégration favorise la cohérence entre documentation et code source, facilite l'accès à la documentation, permet la distribution d'un code source auto-documenté. Elle rend donc plus aisée la maintenance du projet.

Entêtes de méthode

Description

Toute définition de méthode doit être précédée du bloc de documentation contenant au minimum :
- Une description de la méthode
- Tous les arguments
- Toutes les valeurs de retour possibles

Exemple

```
/// <summary>
/// Méthode qui vérifie :
/// si les champs obligatoires sont renseignés,
/// si les formats sont valides,
/// et si les valeurs saisies sont cohérentes
/// </summary>
/// <returns>booléen ok</returns>

private bool verif() {…}
```

Commentaires des instructions du code

Description

Il existe deux types de commentaires :
1. les commentaires mono ligne qui inactivent tout ce qui apparaît à la suite, sur la même ligne : //
2. les commentaires multi-lignes qui inactivent tout ce qui se trouve entre les deux délimiteurs, que ce soit sur une seule ligne ou sur plusieurs /* */

Les commentaires mono-ligne permettant de commenter le reste, à savoir, toute information de documentation interne relative aux lignes de code. Ceci afin d'éviter des erreurs de compilation dues aux imbrications des commentaires multi-lignes.
Les commentaires multi-lignes sont réservés pour l'inactivation de portions de code.

2. NOMMAGE DES IDENTIFICATEURS

Cette convention concerne les éléments suivants du langage :

- les méthodes,
- les paramètres formels de méthodes,
- les constantes,
- les variables.

Pour l'ensemble de ces éléments, la clarté des identificateurs est conseillée. Le nom attribué aux différents éléments doit être aussi explicite que possible, c'est un gage de compréhension du code.

Nommage des méthodes

Description

L'identificateur d'une méthode est un verbe, ou groupe verbal.
Les noms de fonctions ne peuvent contenir que des caractères alphanumériques. Les tirets bas ("_") ne sont pas permis. Les nombres sont autorisés mais déconseillés.
Les noms de fonctions doivent toujours commencer avec une lettre en minuscule. Quand un nom de fonction est composé de plus d'un seul mot, la première lettre de chaque mot doit être mise en majuscule. C'est ce que l'on appelle communément la "notationCamel".

Exemples : filtrerChaineBD(), verifierInfosConnexion(), estEntier()

Intérêts

Maintenabilité : lisibilité.

Nommage des variables et paramètres

Description

L'identificateur d'une variable ou paramètre indique le rôle joué dans le code ; c'est en général un nom, ou groupe nominal. Il faut éviter de faire jouer à une variable plusieurs rôles.
Les noms de variables et paramètres ne peuvent contenir que des caractères alphanumériques. Les tirets bas sont autorisés uniquement pour les membres privés d'une classe. Les nombres sont autorisés mais déconseillés.

Comme les identificateurs de fonctions, les noms de variables et paramètres adoptent la notation Camel.

Exemples : nomEtud, login

Intérêts

Maintenabilité : lisibilité.

3. CODAGE

Fonctions/Méthodes

- *Modularité*

Description

Le codage doit être réalisé en recherchant le plus possible la modularité :
- chaque méthode doit réaliser un et un seul traitement,
- chaque méthode doit être construite de manière à posséder la plus forte cohésion et la plus grande indépendance possible par rapport à son environnement.

Intérêts

Maintenabilité et fiabilité : modularité.

- *Nombre de paramètres des méthodes*

Description

Les méthodes ne doivent pas comporter un trop grand nombre de paramètres. La limite de 5 à 6 paramètres est recommandée. Tout dépassement de cette limite doit être justifié.

Compléments

Cette règle s'applique, tout spécialement, dans le cadre de la programmation par objets qui permet justement de réduire le nombre de paramètres des fonctions.

Intérêts

Maintenabilité : lisibilité.

Instructions

- *Ecriture des instructions d'affectation*

Description

Il faut utiliser dès que possible les formes abrégées des instructions d'affectation.

Compléments

Les instructions d'affectation du type :

A = A <op> <exp> ;

peuvent être notées sous leur forme abrégée :

A <op>= <exp> ;

Exemples

Écrire : total *= 0.90; *au lieu de :* total = total * 0.90;

- *Parenthésage des expressions*

Description

Il est recommandé d'utiliser les parenthèses à chaque fois qu'une expression peut prêter à confusion.

Exemples

Il ne faut pas écrire ...

```
if (nbLignes == 0 && nbMots == 0)
```

...mais plutôt écrire

```
if ((nbLignes == 0) && (nbMots == 0))
```

Intérêts

L'ajout de parenthèses dans les expressions comportant plusieurs opérateurs permet d'éviter des confusions sur leur priorité.

Maintenabilité : lisibilité.

- *Limitation de l'utilisation des break et continue dans les itératives*

Description

Utilisation modérée
Les ruptures de séquence break et continue doivent être utilisées avec modération dans les itératives.

Compléments

L'abus de ce type d'instructions peut rendre le code difficile à comprendre. Elles pourront toutefois être utilisées ponctuellement. Dans ce cas, un commentaire devra le signaler.

Intérêts

Limiter les instructions break et continue améliore la structuration du code. Ces instructions (qui sont des "goto" déguisés), lorsqu'elles sont utilisées fréquemment, peuvent en effet dénoter une mauvaise analyse des conditions d'itérations dans certains cas.

- *Ecriture des switch*

Description

1. Tout le contenu à l'intérieur de l'instruction "switch" doit être indenté avec 4 espaces. Le contenu sous chaque "case" doit être indenté avec encore 4 espaces supplémentaires.
2. Les structures switch doivent obligatoirement comporter une clause default.
3. Le niveau d'imbrication des switch ne doit pas dépasser 2.
4. Chaque cas ou groupe de cas doit se terminer normalement par une instruction break. Les cas ne se terminant par un saut **break** doivent spécifier un commentaire rappelant que l'exécution se poursuit.
5. L'instruction break est obligatoire à la fin du cas par défaut. Cela est redondant mais protège d'une erreur en cas de rajout d'autres cas par la suite.

4. FORMULAIRES WINDOWS

Nommage des formulaires et des champs de formulaires

Description

Les noms des contrôles graphiques débuteront par un préfixe rappelant leur type.
Les préfixes retenus concernent les formulaires et les champs contenus dans les formulaires.

Type d'élément	Préfixe
Formulaire	frm
Zone de texte	txt
Etiquette (label)	lbl
Bouton d'option (bouton radio)	rb
Case à cocher	chk
Bouton de commande	btn
Zone de liste (comboBox, listBox)	cbo, lst
Grille de données (dataGridView)	dgv
Boîte de groupe (groupBox)	gb
Onglets (tabControl)	tbc
Pages d'onglets (tabPage) Le nom d'un contrôle contenu dans une page sera précédé du nom de la page suivi du caractère "_" Exemple : *tb1_txtIntitule* est une zone de texte contenue dans la tabPage tb1	tb