GUIDE DE CODAGE

Date : 03/12/2005

Version : v 1.0 beta

Responsable du document : Olivier TANKOANO

Historique des révisions

Date	Version	Description	Auteur
03/12/2005	V 1.0 beta	Création	Olivier TANKOANO

Table des matières :

1. Introduction

Ce document décrit les règles à suivre durant les phases de codage de l'application 2DB et la démarche à suivre pour configurer le logiciel Eclipse.

1.1. Objectif

Les règles décrites dans ce document permettront d'obtenir une homogénéité du code produit améliorant ainsi la lisibilité de celui-ci.

1.1.1. Portée

Ce document est destiné à tous les développeurs et testeurs de l'application 2DB.

1.1.2. Contenu du document

Ce document regroupe les règles de nommage des identifiants Java et des fichiers sources, ainsi que les règles de codage et d'organisation des fichiers.

2. Conventions de nommage

2.1. Conventions générales

L'ensemble des noms de classes, interfaces, variables …. devront être en Anglais.

2.1.1. Nommage des Classes et Interfaces

Les noms des Classes doivent commencer par une majuscule ainsi que chacun des mots composant le nom de la classe. Lors du développement les classes devront respecter la loi « Modèle – Vue - Control ». Par conséquent les classes de type vue débuteront par « V_ », celles de type control par « C_ » et les autres de type modèle par « M_ ».

Exemple : V_ActivityWindow, C_CheckData, M_Process

Le nom des Interfaces doivent débuter par un 'I' majuscule, et les différents mots composant ce nom doivent débuter par une majuscule.

Exemple : M_IProcess, C_ IHandleExecptions

2.1.2. Nommage des méthodes

De manière générale, les noms des méthodes des Classes et Interfaces doivent débuter par une minuscule, puis chacun de leurs mots (excepté le premier) doit débuter par une majuscule.

Exemple : aadActivity (),deleteActivity ()

Les « getters » et « setters » (méthodes d'accès en lecture et écriture des membres privés) doivent débuter par « get » et « set » suivi du nom de l'attribut dont chacun des mots doivent débuter par une majuscule.

Exemple : getName (), setName ()

De même, les méthodes de test d'une valeur retournant un booléen doivent commencer par « is ».

Exemple : isEmpty (),isSelected ()

Le nom de l'objet qui contient la méthode est implicite, il est inutile de le faire apparaître dans le nom de la méthode.

Exemple : activity1.getName () et non activity1.getActivityName ()

2.1.3. Nommage des attributs de Classes et interfaces

Attributs non constants (appartenant à une classe)

Le nom des attributs non constant doit débuter par la lettre « m », suivi de chacun de leurs mots débutant par une majuscule.

Exemple : mNbHours

Attributs constants

Le nom des attributs constants (attributs déclarés en « static final ») ne doit être composé que de majuscules et chacun des mots le constituant doit être séparé par un underscore « _ ».

Exemple : MAX_NB_ACTIVITIES

Nommage des variables locales à une méthode

Les règles de nommage des attributs s'appliquent pour les variables locales à une méthode mais sauf que celle-ci doivent débuter par la lettre « l » au lieu de « m ». Les attributs doivent avoir des noms complets et significatifs.

Exemple : lTemporaryActivity

Les variables utilisées seulement dans un bloc (boucle, parcours d'indice, …) peuvent avoir des noms courts.

Exemple : cpt, i, tmp

2.2. Conventions particulières

2.2.1. Nombre d'objets

Les variables qui représentent un nombre d'objets doivent débuter par le préfixe « nb ». Ils doivent débuter par un « m », « l » suivant leur origine.

Exemple :lpNbLines, mNbPoints

3. Conventions de codage en JAVA

3.1. « Package » et « Import »

3.1.1. Mot clé « package »

Il doit apparaître au début d'un fichier classe et suivi du nom du package auquel appartient la classe.

Exemple : package 2DB.application.boundary, package 2DB.application.control

3.2. Mot clé « import »

Les lignes d'import doivent suivre la ligne désignant le package.

L'utilisation de * pour importer toutes les classes d'un même package n'est à utiliser que lors d'importations massives de classes d'un même package (car son utilisation ralentit la compilation).

3.3. Contenu des classes et interfaces

3.3.1. Architecture des fichiers sources (.java) décrivant une classe ou une interface

Chacune des classes et interfaces doit contenir ces informations dans l'ordre indiqué :

1. Ligne de déclaration d'appartenance au package (cf. 3.1.1).
2. Ligne de déclaration des importations de classes (cf. 3.1.2).
3. Description de la classe / interface au format JavaDoc.
4. Ligne de déclaration de la classe / interface.
5. Attributs privés de la classe.
6. Attributs publics de la classe.
7. Constructeurs.
8. Méthodes de la classe.
9. Méthodes implantant l'interface héritée.
10. Getters et setters.

3.4. Attributs et variables

3.4.1. Déclaration des attributs

Les attributs de la classe ne doivent pas être déclarés en « public » Il est nécessaire de les déclarer en « private » et d'implanter des « getters » et « setters » sur chacun des attributs.

3.5. Utilisation des assert

Le mot clé « assert » s'applique à un booléen permet de s'assurer du comportement du code produit (pré/post-condition ou invariant sur une portion de code). Il devra donc impérativement être utilisé afin de faciliter la détection des erreurs le plus rapidement possible dans le développement. Il ne servira que lors du débogage du code car il sera désactivé dans la version finale du logiciel.

NB : Le mot clé « assert » ne remplace pas les exceptions car ces dernières seront utilisées dans la version finale du logiciel. Les exceptions font partie du comportement d'une classe. Elles serviront donc à détecter des erreurs d'utilisation de la classe tandis que les assertions détecteront des erreurs d'implémentation.

Exemple :

```
Class MyList
{
        …
        Public static final NB_MAX_ELEMENTS = 1000;
        …

        Public void add(Object obj)
        {
                If(this.size() == NB_MAX_ELEMENTS)
                {
                        Throw( new Exception("The list is full!"));
                }
                int oldSize = this.size();
                //code d'ajout de l'élément
                …
                …
                //fin code d'ajout de l'élément

                assert( list.size() == oldSize() + 1);
        }

        …
}
```

4. Indentation et commentaries

4.1. Indentation

L'indentation se fera à l'aide de 4 espaces et non d'une tabulation. Les accolades ouvrantes se placeront sur la ligne suivant leur en-tête.

Exemples :

```
if ( condition )
{
  instruction 1 ;
  instruction 2 ;
}
```

```
while ( condition )
{
  instruction 1 ;
  instruction 2 ;
}
```

```
void setNon ( String pNom )
{
  Nom = pNom ;
}
```

Comme nous pouvons le distinguer dans l'exemple, avant et après chaque opérateur un espace devra être mis, de même pour les parenthèses. De même qu'il faudra placer un espace avant chaque « ; ».

Entre chaque méthode une ligne doit être sautée afin de rendre le code plus lisible.

4.2. Commentaires

4.2.1. Convention

Afin de documenter le code, les commentaires JavaDoc en anglais seront utilisés.

4.2.2. Pour une classe

Chaque classe sera précédée du commentaire JavaDoc suivant :

```
/**
 * Description de la classe
**/
```

4.2.3. Pour une méthode

Chaque méthode sera précédée du commentaire *JavaDoc* suivant :

```
/**
 * @author Auteur de la méthode
 * descriptif de la méthode (fonctionnalité de la methode + mini
 * algorithme si elle est complexe)
 * @param <param_0> Description du premier paramètre nommé 'param_0'
 * @param .....
 * @param <param_n> Description du dernier paramètre nommé 'param_n'
 * @return Description de la valeur retournée (si retour il y a)
 * @exception Description de l'exception qui peut être levée (si exception il y a)
**/
```

4.2.4. Pour un attribut

Chaque attribut sera suivi du commentaire suivant si nécessaire :

```
nomvariable ;  // description de la variable
```

4.2.5. Génération de la documentation

Ligne de commande permettant de générer la documentation :

javadoc –d docs @packages

où packages est le nom du fichier texte contenant la liste de tous les packages de l'application.

4.2.6. Portion de code en commentaire

Toutes personnes mettant une partie du code en commentaire devra spécifier son nom, ainsi que les raisons qui l'ont poussées à mettre cette portion en commentaire.

5. Formatage du code

Afin d'être sûr que les règles énoncées précédemment dans ce document sont respectées, et après avoir configurer Eclipse, sélectionner le code à formater dans l'éditeur de code, cliquez sur le bouton droit, allez dans l'option « Source » puis choisissez « Format » (Ctrl + Shift + F).

Pour configurer les options de formatage d'Eclipse, allez dans Le menu « Windows », choisissez « Préférences ». Dans l'arborescence, développez le noeud « Java », puis aller dans le sous-noeud « Code Style ».

Dans l'option « Code Formatter », importer le fichier « Format2DB.xml ».

6. Exemple

```
/**
* This interface contains Geometric figures common functions
*/
public interface IFigure
{
  public int Surface () ;
}
```

```
/**
* This class represents figures of type Rectangle
 */
public class Rectangle implements IFigure
{
  int mLength ; // length of the rectangle
  int mWidth ;  // width of the rectangle

  /**
   * default constructor
   */
  Rectangle ()
  {
    mLength = 0 ;
    mWidth = 0 ;
  }

  /**
   * @author TANK Oli
   * Constructor
   * @param pLength : length of the rectangle
   * @param pWidth : width of the rectangle   */

  Rectangle (int pLength, int pWidth)
  {
    mLength = pLength ;
    mWidth = pWidth ;
  }

  /**
   * @author TANK Oli
   * returns the surface of the rectangle
   * @return Surface of the rectangle
   */
  public int Surface ()
  {
    int lSurface ;
    lSurface = mLength * mWidth ;
    return lSurface ;
  }
  /**
   * @return Returns the width of the rectangle
   */
  public int getWidth ()
  {
    return mWidth ;
  }

  /**
   * @param initialize llength witrh pLength.
   */
  public void setLength (int pLength)
  {
    mLength = pLength ;
  }
}
```

7. Guide HTML/XML

7.1. Indentation du code html

Comme pour le codage en java il sera recommandé de ne pas utiliser la tabulation mais deux espaces. Les balises fermantes devront être alignées avec les balises ouvrantes correspondantes pour permettre une meilleure lisibilité du code. Le contenu de chaque balise devra se situer à la ligne suivante de l'ouverture de la balise sans oublier l'indentation.

7.2. Exemple de code

```
<font>
  Texte qui sera afficher
</font>

<table>
  <tr>
    <td>
      Cellule 1
    </td>
    <td>
      Cellule 2
    </td>
  </tr>
</table>
```