Isi Engineering Process Publisher

Guide de programmation

Version 2.2

Superviseurs de projet : Claude Aubry
Bernard Cherbonneau
Responsable Nicolas Pujos

Historique des révisions

VERSION	DATE	DESCRIPTION DES EVOLUTIONS	AUTEURS
2.0	1/12/03	Création	Nicolas Pujos
2.1	24/01/04	Modifications commentaires	Nicolas Michel
2.2	28/01/04	Modifications nom de méthode	Nicolas Michel

Table des matières

Chapître 1 Introduction

Ce document décrit les règles à suivre durant les phases de codage de l'application IEPP.

1.1 Objectif

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

1.2 Portée

La portée de ce document s'étend à tous les développeurs et testeurs de l'équipe IEPP.

1.3 Références

- http://java.sun.com/j2se/javadoc : Site officiel de Sun : compléments d'informations sur les commentaires Java avec "*JavaDoc*".

1.4 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. Pour finir, il énumère quelques recommandations générales relatives à la programmation.

Chapître 2 Conventions de nommage

2.1 Conventions générales

2.1.1 Nommage des Classes et Interfaces

Le nom des Classes et Interfaces doit débuter par une Majuscule, chacun des mots devant aussi débuter par une Majuscule (ex: *"Processus"*, *"GestionExceptionsXML"*).

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 (ex: *"ajouterActivite()", "supprimerActiviteSelectionnee()").*

Les *getters* et *setters* (méthodes d'accès en lecture/écriture des membres privés) doivent respectivement débuter par "get" et "set" (ex: "*getNbActivites()*", "*setActivite()*").

De même, les méthodes de test d'une valeur qui retournent un booléen doivent commencer par "est" (ex: *"estVide()", "estSelectionne()"*).

Le nom de l'objet qui contient la méthode est implicite : il est inutile de le faire apparaître dans le nom de la méthode (ex: *"activite.getNom()"* et non *"activite.getNomActivite()"*).

2.1.3 Nommage des attributs de Classes et Interfaces

Attributs non constants

Le nom des attributs non constants doit débuter par une minuscule, puis chacun de leurs mots (excepté le premier) doit débuter par une majuscule. (ex: *"activites", "nbActivites").*

Attributs constants

Le nom des attributs constants (attributs déclarés en *"static final"*) ne doit être composé que de Majuscules et d'*underscores* : chaque mot écrit exclusivement en Majuscule sera séparé du suivant par un *underscore*
(ex:"PI","NOMBRE_MAXI_ACTIVITES", …)

2.1.4 Nommage des variables locales à une méthode

Les règles de nommage des variables locales sont les mêmes que celles qui s'appliquent aux attributs (*cf. ci-dessus)*
*(*ex: *"activite", "nbActivites")*

Les variables déclarées au début d'une méthode et utilisées tout au long de celle-ci doivent avoir des noms complets et significatifs (ex: *"largeurTotalePage").*

Les variables utilisées seulement dans un bloc (boucle, parcours d'indice, …) peuvent avoir des noms courts (ex: *"cpt", "i", "j")*

2.2 Conventions particulières

2.2.1 Nombre d'objets

Les variables qui représentent un nombre d'objets sont composées du préfixe "nb" suivi du nom de l'objet au pluriel (ex: *"nbLignes", "nbPoints"*)

Chapître 3 Conventions de codage en Java

3.1 Mots-clés "package" et "import"

3.1.1 Mot-clé "package"

Directement après les commentaires de description de la classe (*cf. §3.2.1*) doit se trouver le mot-clé *package*, suivit du nom du package auquel la classe appartient. Chaque fichier (⇔ classe) doit appartenir à un package (ex: package "IEPP", "iepp.application", …).

3.1.2 Mot-clé "import"

Les lignes d'*import* doivent directement suivre l'instruction *package*, et présentent de manière triée les classes importées en commençant par les plus fondamentales et en les regroupant de manière logique.
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).

3.2 Contenu des classes et interfaces

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

Les déclarations de classes et d'interfaces doivent être présentées de la manière suivante :

1. Ligne de déclaration d'appartenance au package (utilisation du mot-clé *"package"*)
2. Lignes de déclaration des importations de classes (utilisation du mot-clé *"import"*)
3. Documentation de la classe / interface au format JavaDoc
4. Ligne de déclaration de la classe / interface (utilisation des mots-clés *"class", "interface", "extends",* et *"implements"*)
5. Attributs privés de la classe
6. Attributs publics de la classe
7. Constructeur(s)
8. Méthodes de la classe
9. Méthodes implémentant l'interface héritée

3.3 Attributs et variables

3.3.1 Déclaration des attributs

Les attributs de classe ne doivent pas être déclarés "*public*", sauf cas exceptionnel (utilisation d'un attribut d'une classe comme d'un champ d'une structure). Il est nécessaire de les déclarer "*private*" et de créer des méthodes (*getters* et *setters*) permettant d'y accéder.

3.3.2 Initialisation des attributs et variables

Les attributs et variables doivent être initialisés au moment de leur déclaration. Si aucune valeur particulière ne se justifie, des valeurs telles que **""**, **0**, ou **null** seront utilisées pour l'initialisation.

Chapître 4 Indentation et commentaires

4.1 Indentation

L'indentation de base se fera à l'aide d'une tabulation. Par souci de lisibilité, les accolades ouvrantes et fermantes seront alignées sur une même verticale, et placées seules sur des lignes vides :

```
while (condition)
{
      instr1;
      instr2;
}
```

et non

```
while (condition) {
instr1;
instr2;
}
```

4.2 Commentaires

4.2.1 Convention

Afin de générer la documentation du code de manière immédiate et automatique, les commentaires JavaDoc 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 :

```
/**
* descriptif de la méthode (fonctionnalité de la methode + mini
* algorithme si elle est compliquée)
* @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 précédé du commentaire *JavaDoc* suivant :

```
/**
* Descriptif de l'attribut (facultatif : plage de valeur, valeur initiale)
*/
```

4.2.5 Génération de la documentation

Ligne de commande pour 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.

FIN DE DOCUMENT