École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
4e année
2014 - 2015

Développement d'un serveur de jeu pour l'initiation à la programmation

Convention de nommage

Encadrants
nameNicolas MONMARCHE
nicolas.monmarche@univ-tours.fr
Sébastien AUPETIT
aupetit@univ-tours.fr
Carl ESSWEIN
carl.esswein@univ-tours.fr

Université François-Rabelais, Tours

Etudiants
Ali AALAOUI
ali.aalaoui@etu.univ-tours.fr
Olivier BUREAU
olivier.bureau@etu.univ-tours.fr
Anthony DEMUYLDER
anthony.demuylder@etu.univ-tours.fr
Nicolas GOUGEON
nicolas.gougeon@etu.univ-tours.fr
Jonathan LEBONZEC
jonathan.lebonzec@etu.univ-tours.fr
Benjamin MAUBAN
benjamin.mauban@etu.univ-tours.fr
Juliette RICARD
juliette.ricard@etu.univ-tours.fr
Minghui ZHANG
minghui.zhang@etu.univ-tours.fr

DI5 2008 - 2009

Version du 25 février 2015

Table des matières

Table des figures

Liste des tableaux

Introduction

En informatique, une convention de nommage est un ensemble de règles destinées à choisir les identifiants logiciels (noms des éléments du programme) dans le code source et la documentation. [cf Wikipédia] En d'autres termes, une convention est une approche des bonnes pratiques de programmation qui permet à tout développeur d'une équipe de coder proprement.

Ainsi, nous pensons qu'il est nécessaire de mettre en place une convention de nommage afin que chaque membre de l'équipe code à l'identique. Cela favorise la lisibilité du code.
Cette convention de nommage facilitera aussi les futures modifications et maintenances sur le projet.

Pour ce projet nous utiliserons majoritairement le langage Java. Ainsi, cette convention de nommage est particulièrement adaptée à ce langage.

Nommage

Pour ce projet, nous utilisons exclusivement l'anglais.

2.1 Fichiers

En règle générale, le nom du fichier est le nom de la classe, si c'est un fichier de classe. Sinon, le nom du fichier décrit l'utilisation du fichier.
Chaque nouveau mot (même le premier) est en minuscule et commence par une majuscule Les abréviations sont à éviter car cela peut fausser la compréhension de l'utilisation du fichier.

Exemple :
- UsefulFonctions.java
- DoctorAgenda ;java

2.2 Classes

Les noms de classe sont constitués de mots, en l'occurence, de noms communs. Chaque mot commence par une majuscule, le reste étant en minuscule.
Chaque nouveau mot dans un nom de classe commence par une majuscule. Les abréviations sont à éviter.

Exemple :
- class Diary
- class DoctorAgenda

2.3 Constantes

Les noms des constantes sont constitués de mots, en l'occurrence, des caractéristiques. Chaque mot est tout en majuscule, et tous les mots sont séparés par un _.

Exemple :
- LENGTH_MAX
- TEMPERATURE

2.4 Variables

Les noms des variables sont les plus importants. En effet, ce sont eux qui vont le plus aider à la compréhension du code. Ainsi chaque variable doit posséder un nom cohérent avec son utilisation. Pour les indices de boucles, nous utilisons les lettres i, j, k, ...
Le premier mot est entièrement en minuscule (incluant la première lettre). Chaque nouveau mot, dans un nom de variable, est en minuscule et commence par une majuscule. Les abréviations sont à éviter au maximum.

Exemple :
- int numberOfPupils
- String namePerson

2.5 Méthodes

Une méthode effectue habituellement une action. Ainsi, le nom de la méthode doit être suffisamment explicite pour savoir ce qu'elle fait. Le nom de la classe est entièrement en minuscule.
Chaque nouveau mot dans un nom de méthode commence par une majuscule. Les abréviations sont à éviter vu que l'on veut traduire au maximum l'action.

Exemple :
- void fileParser()
- void classInAscendingOrder()

Le cas des *getters* et des *setters* est un peu particulier : le nommage est identique à celui d'une méthode classique (premier mot tout en minuscule, puis majuscule sur les premières lettres des mots suivants). Or un getter (resp. setter) utilise, dans son nom, le nom d'une variable. Ainsi cette variable passe sa première lettre en majuscule lorsqu'elle est utilisée par un getter ou un setter.

Exemple :
- void getFilmTitle()
- void setFilmTitle()

Commentaires

La documentation est un outil permettant à tous de comprendre le code.

3.1 Implémentation

Les commentaires d'implémentation s'utilisent pour décrire les actions du code auquel on fait référence. Ils sont utilisés notamment pour expliquer un bout de code ou décrire une spécificité.

3.1.1 Ligne

Si un commentaire est court, c'est-à-dire sur une seule ligne, on utilise //. Si ce commentaire est mis à la suite d'une instruction il est indenté.

Exemple :

- // To start a game
- int number ; // This comment is useless
  String name ; // This comment is useless too

3.1.2 Bloc

Si un commentaire est long, c'est-à-dire sur plusieurs ligne, on utilise /* ... */.

Exemple :

- /*
  To start a new game
  with a bot
  */

3.2 Documentation (Javadoc)

L'idée des commentaires de documentation est de pouvoir générer automatiquement une documentation grâce à Javadoc. Ces commentaires se placent en en-tête des fichiers, des classes, des fonctions et des variables.

La Javadoc est un outil développé par Sun Microsystems permettant de créer la documentation des API d'une application à partir des commentaires présents dans le code source.

Tableau des tags de documentation Javadoc :

Tag Javadoc	Description
@author	Référence le développeur de la classe ou de la méthode
@version	Référence la version d'une classe ou d'une méthode
@see	Précise une référence utile à la compréhension
@param	Référence un paramètre de méthode. Requis pour chaque paramètre.
@return	Référence le type de valeur de retour. A ne pas utiliser pour une méthode sans retour.
@deprecated	Marque la méthode comme dépréciée (à éviter d'utiliser)
@exception / @throws	Référence aux exceptions susceptibles d'être levées par la méthode

Tout commentaire de documentation sera de la forme suivante
/**
* Description
* @tag ...
*/.

3.2.1 Classe

Voici un exemple d'utilisation de Javadoc pour documenter une classe :

```
/**
* Student management class
* @author Peter Joe
* @version 2.6
*/

public class Student
{
...
}
```

3.2.2 Méthode

Voici un exemple d'utilisation de Javadoc pour documenter une méthode :

```
/**
* Valid movement
* @param rowOfThePiece Row of the piece to move
* @param destinationRow Destination row of the piece of chess
* @return true if the movement is valid
*/

public class Student
{
...
}
```

Déclarations

4.1 Classes et méthodes

Les déclarations des classes et des méthodes se font dans un ordre logique. Nous définissons ici cet ordre :

```
/**
* Documentation classe
*/

class Doctor
{
	/* Déclaration des attributs */
	/* Déclaration des constructeurs */
	/* Déclaration des méthodes */
	/* Déclaration des getters & setters */
}
```

4.2 Attributs

Les attributs sont classés par visibilité (les public sont rassemblés et les private aussi). Dans chaque visibilité, les attributs sont classés par ordre d'utilisation.

Style et lisibilité

5.1 Caractères par ligne

Le nombre maximum de caractères par ligne est de 90 caractères.

5.2 Instructions

Une seule instruction par ligne est tolérée.

Si la liste des paramètres d'une fonction dépasse 90 caractères, on met chacun des paramètres à la ligne alignés au premier.

Exemple :

```
public void classInAscendingOrder(int number,
                                  String name,
                                  int age)
```

5.3 Encodage

L'encodage utilisé est UTF-8 car c'est un encodage multi-plateforme.

5.4 Indentation

L'indentation du code se fait par une tabulation.

5.5 Parenthèses

Les parenthèses servent à éviter les ambiguïtés. On les utilisera donc aussi souvent que nécessaire. Il n'y a pas d'espace entre une parenthèse et ce qu'elle contient.

Exemple :

NON ACCEPTE

```
if (a == b && c == d)
```

ACCEPTE

```
if ((a == b) && (c == d))
```

5.6 Accolades

Les accolades se mettent à la ligne et non à la fin d'une ligne.

Exemple :

```
if (condition)
{
```

[action]
}

5.7 Espaces

Voici une liste des divers endroits où un espace est obligatoire afin de permettre une meilleure visibilité du code :

- après un mot clé (for, while, switch, if, ...)
- après une virgule ou un point-virgule
- avant et après un opérateur (+, =, <, ...)
- entre une variable et sont typage (cast)

Débogage

Nous utilisons le framework JUnit afin de faire les tests nécessaires évitant un débogage trop important. En effet, chaque classe dispose de fonctions de test afin de tester chacun des cas critiques. Cela évite d'avoir de mauvaises surprises tout au long du projet.

Convention de nommage

Département Informatique
4e année
2014 - 2015

Développement d'un serveur de jeu pour l'initiation à la programmation

Résumé : Convention de nommage pour le projet d'ingénierie logicielle : Développement d'un serveur de jeu pour l'initiation à la programmation

Mots clefs : convention, nommage

Abstract: Naming convention for the software engineering project : Development of a game server for the introduction to programming

Keywords: convention, naming

Encadrants
nameNicolas MONMARCHE
nicolas.monmarche@univ-tours.fr
Sébastien AUPETIT
aupetit@univ-tours.fr
Carl ESSWEIN
carl.esswein@univ-tours.fr

Université François-Rabelais, Tours

Etudiants
Ali AALAOUI
ali.aalaoui@etu.univ-tours.fr
Olivier BUREAU
olivier.bureau@etu.univ-tours.fr
Anthony DEMUYLDER
anthony.demuylder@etu.univ-tours.fr
Nicolas GOUGEON
nicolas.gougeon@etu.univ-tours.fr
Jonathan LEBONZEC
jonathan.lebonzec@etu.univ-tours.fr
Benjamin MAUBAN
benjamin.mauban@etu.univ-tours.fr
Juliette RICARD
juliette.ricard@etu.univ-tours.fr
Minghui ZHANG
minghui.zhang@etu.univ-tours.fr

DI5 2008 - 2009