2018

Convention de codage JAVA

2I013 - PROJET

ENZO WESQUY

SORBONNE UNIVERISTE - SCIENCES

Table des matières

1. Code

1.1. Généralités

La position de l'accolade ouvrante (pour la définition de classes, de méthodes, etc.) doit toujours être précédé d'un retour à la ligne.

1.2. Nommage des classes

Une lettre majuscule au début de chaque mot :

```
visibility class MyClassStarter
{
…
}
```

Si possible, préférer un nom **actif** plutôt qu'un nom passif :

```
visibility class MyClassStarting          // A éviter
visibility class MyClassStarter           // Ok
```

1.3. Nommage des méthodes

Une lettre majuscule au début de chaque mot sauf pour le premier :

```
visibility type myFunction(…)
{
…
}
```

Si possible, commencer par un verbe :

```
visibility type lineRead(…)         // A éviter
visibility type readLine(…)         // Ok
```

Remarques : accesseurs :

- Utiliser « get » comme premier mot pour les « getters » et « set » pour les « setters » :

```
type myClass.getProperty()
void myClass.setProperty(type)
```

- Si possible, utiliser « is » et le participe passé pour les fonctions qui retournent un booléen :

```
type myClass.isInitialized()
```

1.4. Nommage des membres d'une classe

Commencer par « m_ » et une lettre majuscule au début de chaque mot, sauf pour le premier mot :

```
visibility class MyClass
{
	private m_myPrivateVariable;
	public m_myPublicVariable;
}
```

1.5. Nommage des constantes

Les constantes devront être écrites de cette façon :

```
visibility static final type CONSTANT_NAME = value;
```

1.6. Style de syntaxe

```
if (condition)
{
	Statement1
}
else if ((condition1) && (condition2))
{
	Statement2
}
else
{
	Statement3
}
```

```
switch (operation)
{
	case Multiply:
		result *= amount;
		break;
	case Add:
		result += amount;
		break;
	default:
		break;
}
```

Remarques : pour une meilleure lecture du code par un tiers, toutes les conditions seront suivies d'accolades, même si elles ne contiennent qu'une ligne de code.

Remarques : l'utilisation de conditions ternaire est conseillée si elles n'altèrent pas la visibilité du code.

1.7. Bonnes pratiques

Initialisation de tous vos membres dans votre constructeur.

Aucun « *magic number* » ne soit apparaitre spontanément dans le code. Il est nécessaire de définir une constante sous une forme ou une autre (comme défini 1.4).

Dans un souci de cohérence de lecture des documents il est indispensable de remplacer les 4 espaces par une tabulation si l'éditeur de texte n'est pas configuré ainsi.

Ajout du mot clé de visibilité (public/private/protected) même s'il est défini par défaut.

2. Commentaires

2.1. Généralités

Les commentaires de même que les différents nommages utiliseront de préférence la langue anglaise.

Les fichiers devront comporter le cartouche suivant :

```
/** ****************************************************************

       Nom du fichier

       @author       Nom de l'auteur
       @version      Numéro de version

// Copyright (c) 2010 DASSAULT AVIATION, All Rights Reserved
// No part of this software and its documentation may be used, copied,
// modified, distributed and transmitted, in any form or by any means,
// without the prior written permission of DASSAULT AVIATION.

**************************************************************************/
```

2.2. Classes

Les fichiers de classes devront être structurés de la manière suivante :

- Définition de la classe
- Définition des constantes
- Définition des membres publics
- Définition des membres privés
- Constructeur
- Méthodes
- Accesseurs

```
/** ****************************************************************

       Nom du fichier

       @author       Nom de l'auteur
       @version      Numéro de version
```

```
// Copyright (c) 2010 DASSAULT AVIATION, All Rights Reserved
// No part of this software and its documentation may be used, copied,
// modified, distributed and transmitted, in any form or by any means,
// without the prior written permission of DASSAULT AVIATION.

**************************************************************************/

visibility class MyClass
{
  /* ************************************************
   * *************** Constants ********************
   * ************************************************/

   visibility static final type CONSTANT_NAME = value;

  /* ************************************************
   * ***************** Context ********************
   * ************************************************/

  /* Public */
  public m_myPublicVariable;

  /* Private */
  private m_myPrivateVariable;

  /* ************************************************
   * *************** Construcor *******************
   * ************************************************/

  visibility MyClass(…)
  {
   …
  }

  /* ************************************************
   * ***************** Methods ********************
   * ************************************************/

  visibility type myMethod (…)
  {
   …
  }

  /* ************************************************
   * ************* Getters/Setters ****************
   * ************************************************/

  visibility type getProperty()
  {
   …
  }

  visibility type setProperty(…)
  {
   …
  }
}
```

2.3. Méthodes

Lorsqu'il est nécessaire (i.e. pas pour les accesseurs), la définition des méthodes doit être précédé d'un cartouche expliquant son utilité, ses paramètres ainsi que sa valeur de retour.

```
/**
 * Do something.
 *
 * @param param1
 *              Description of param1.
 *
 * @param param2
 *              Description of param2.
 *
 * @return Some value.
 */
visibility type myMethod(type param1, type param2)
{
   return value;
}
```