JAVADOC

Introduction

Il y a deux bonnes raisons pour commenter correctement son code. Tout d'abord, cela permet de mieux comprendre ce que l'on a écrit. Mais surtout, cela facilitera le travail de ceux qui voudraient comprendre, utiliser, voire maintenir votre travail.

Javadoc, créé par Sun Microsystems (la société qui a créé le langage Java), permet, en inspectant le code Java des classes, de produire une documentation très complète de votre code.
L'outil génère des pages HTML contenant au minimum la liste des classes, la liste des méthodes et la liste des variables. Nous verrons ensuite qu'il est possible d'ajouter tout un tas d'informations, de commentaires, afin de générer une véritable documentation, exhaustive et facilement lisible.

Les tags Javadoc

Il est possible d'ajouter une grande quantité d'informations à un code, en utilisant les commentaires. Mais ils ne sont pas pris en compte par l'outil Javadoc. Mais Sun a bien fait les choses, il existe des commentaires spécialement faits pour la Javadoc, et mieux encore, des tags précis qui permettent de détailler des informations sur chaque classe, chaque méthode, etc.

Les commentaires

Pour rappel, il existe trois types de commentaires en Java. Les commentaires en lignes, les commentaires sur plusieurs lignes, et les commentaires Javadoc.

```
/**
 * Ceci est un commentaire Javadoc.
 * Il commence par un slash suivis de deux étoiles.
 * Chaque ligne doit ensuite commencer par une étoile.
 * Enfin, il fini par une étoile suivie d'un slash.
 */
protected void commentaires(){
        // Ceci est un commentaire sur une ligne
        [code];
        /* Ceci est un commentaire sur
        plusieurs lignes */
        [suite du code]
}
```

Donc toutes les informations que nous mettrons se trouverons dans ce commentaire Javadoc. Il doit se situer sur la ligne immédiatement avant le nom de la classe, de la méthode, ou de la variable.

Les tags Javadoc

Les tags Javadoc sont au nombre de neuf :

- @author
- @version
- @param
- @return
- @throws (synonyme: @exception)
- @see
- @since
- @serial
- @deprecated

@author et @version : l'auteur et le numéro de version de la classe

Le tag @author renseigne le nom de l'auteur de la classe. Le tag @version indique le numéro de version de la classe. Ce dernier tag est utilisé ensuite par le tag @since (voir plus bas). **Important :** ces tags ne peuvent être utilisés que pour une classe ou une interface. Jamais pour une méthode.

```
/**
 * MonMembre est la classe représentant un membre du Site.
 *
 * @author John Doe
 * @version 3.0
 */
public class MonMembre {

[...]

}
```

@param : les paramètres de la méthode

Le tag @param sert à renseigner le ou les paramètres de la méthode. Derrière le tag, il faut renseigner le nom du paramètre (son type sera inclus automatiquement dans la Javadoc).

```
/**
 * Met à jour le niveau du membre.
 *
 * @param level
 *            Le nouveau level du membre.
 */
protected void setLevel(MyLevel level) {
        this.level = level;
}
```

@return : l'objet retourné par la méthode

Le tag @return sert à renseigner l'objet retourné par la méthode.

```
/**
 * Retourne le level du zéro.
 *
 * @return Une instance de MyLevel, qui correspond à niveau du membre.
 */
public MyLevel getLevel() {
            return level;
}
```

@throws : les exceptions propagées

Le tag @throws indique la présence d'une exception qui sera propagée si elle se lève. Il faut bien indiquer le type de l'exception, et la raison de l'exception.

```java
/**
 * Retourne l'adresse du profil.
 *
 * @return L'URL du profil, générée à partir de l'id.
 *
 * @throws MalformedURLException  Si l'url est mal formée.
 */
public URL getURLProfil() throws MalformedURLException{
        URL url = new URL("http://www.site.com/membres-"+id+".html");
        return url;
}
```

@see : réfère à une autre méthode, classe, etc.

Le tag @see permet de faire une référence à une autre méthode, classe, etc. Concrètement, cela se symbolisera par un lien hypertexte dans la Javadoc. C'est donc un des tags les plus importants.

```java
/**
 * Le "level" du membre. Ce "level" peut être modifié.
 *
 * @see <a href="URL#value">label</a> (tags html reconnus par Javadoc)
 * @see MyLevel (package.class#member [label]   si autre package ou projet)
 * @see "Livre traitant du sujet = pas d'URL disponible" (String)
 */
private MyLevel level;
```

@since : depuis quelle version

Le tag @since permet de dater la présence d'une méthode, d'un paramètre. Derrière ce tag, il faut noter un numéro de version de la classe.

```java
/**
 * Met à jour le pseudo du membre.
 *
 * @since 3.0
 */
public void setPseudo(String pseudo) {
        this.pseudo = pseudo;
}
```

@serial:field-description | include | exclude

Le tag @serial peut contenir un optionnel champ de description (expliquer le sens du champ et la liste des valeurs acceptables)

Les arguments include et exclude renseignent quels package ou classe peut être inclus ou exclus de la page de sérialisation Javadoc. LE fonctionnement est le suivant:

- Une classe *public* ou *protected* qui implément *serializable* est incluse sauf si sa classe (ou son package) est marqué *@serial exclude*
- Une classe *private* qui implément *serializable* est exclue sauf si sa classe (ou son package) est marqué *@serial include*

@deprecated : indique qu'une méthode est dépréciée

Le tag @deprecated doit décrire la version depuis laquelle cette méthode / classe est dépréciée. Mais aussi ce qu'il faut utiliser à la place.

```
/**
 * Retourne la liste des amis.
 *
 * @deprecated Depuis Java 1.4, remplacé par getListeAmis()
 */
protected Vector getVectorAmis(){
}
```

Ces 9 tags sont des tags de type ***block***.

Il existe aussi des tags de type ***inline***:

{@link}:permet de créer un lien vers un autre élément de la documentation.

Le mode de fonctionnement de ce tag est similaire au tag @see : la différence est que le tag @see créé avec le doclet standard un lien dans la section "See also" alors que le tag {@link} créé un lien à n'importe quel endroit de la documentation.

```
{@link package.class#membre [label] }
```

{@value}:permet d'afficher la valeur d'un champ.

Lorsque le tag {@value} est utilisé sans argument avec un champ static, le tag est remplacé par la valeur du champ.

Lorsque le tag {@value} est utilisé avec comme argument une référence à un champ static, le tag est remplacé par la valeur du champ précisé. La référence utilisée avec ce tag suit la même forme que celle du tag @see

{@literal}:permet d'afficher un texte qui ne sera pas interprété comme de l'HTML.

Le contenu du texte est repris intégralement sans interprétation. Notamment les caractères < et > ne sont pas interprétés comme des tags HTML.

Pour afficher du code, il est préférable d'utiliser le tag {@code}

{@code}:permet d'afficher un texte dans des tags <code> ... </code> qui ne sera pas interprété comme de l'HTML.

Le contenu du texte est repris intégralement sans interprétation. Notamment les caractères < et > ne sont pas interprétés comme des tags HTML.

Le tag {@code texte} est équivalent à <code>{@literal texte}</code>

Exemple :

{@code 0<b>10}

{@docRoot}:représente le chemin relatif par rapport à la documentation générée.

Ce tag est pratique pour permettre l'inclusion de fichiers dans la documentation.

Exemple :

Historique

{@inheritDoc}:permet de demander explicitement la recopie de la documentation de l'entité de la classe mère la plus proche correspondante.

Il peut être utilisé :

- dans le block de description principal : dans ce cas tout le commentaire de l'entité de la classe mère est repris
- dans un tag @return, @param, @throws : dans ce cas tout le texte du tag correspondant de la classe mère est repris

Conventions d'écriture

- La première phrase Javadoc de la classe doit être une courte description de la classe.
- Ce conseil vaut aussi pour les méthodes, variables, etc.
- Pour la forme générale des constructeurs et methodes, n'utilisez pas les parenthèses « add » suivi de « method » et non « add() »
- Utilisez la troisième personne pour commenter une méthode.
- Les descriptions de méthodes commencent par un verbe.
- Détaillez le fonctionnement (algo) des méthodes si besoin (éventuellement le tag {@code}).
- Utiliser "ce" (this) plutôt que "le" (the).
- Les tags @param et @return doivent être systématiquement indiqués (sauf méthodes sans paramètres ou méthodes void).

Eclipse et Javadoc

Eclipse intègre très bien la Javadoc durant le développement. Si vous passez le pointeur de votre souris sur le nom d'une classe ou d'une méthode, une petite fenêtre d'information s'accroche sous le pointeur de votre souris, pour vous montrer la documentation correspondante.

Mieux, si vous voulez garder cette information disponible, vous pouvez l'afficher dans la vue "Doc" en bas de l'écran, et continuer à travailler sur votre code.

Générer la Javadoc

Quand vous êtes dans votre projet, il suffit de cliquer sur le menu **Projet**. Plus sélectionnez l'option "Générer la Javadoc".

Un menu avec des options apparait. Remplissez le champ "Commande Javadoc" si ce n'est pas déjà fait. Ensuite, vous pouvez sélectionner les différents packages que vous voulez documenter, ainsi que la visibilité de la documentation. Si vous choisissez "Private", toutes les méthodes et variables seront documentées et visibles dans la documentation générée. Enfin, cliquez sur "Terminer".

Vérifiez la console, pour voir si des erreurs sont apparues pendant la génération.

S'il n'y a pas d'erreurs, votre documentation est prête. Par défaut, elle se trouve dans le répertoire de votre projet, dans un dossier "doc".