Errata 2

La Javadoc

> *« Un langage de programmation est censé être une façon conventionnelle de donner des instructions à un ordinateur, et doit pouvoir être écrit et relu par des personnes différentes.*
> *Il n'est pas censé être obscur, bizarre et plein de pièges subtils.*
> *Ça, ce sont les attributs de la magie. »*
>
> *David Small*[1]

Il n'y a pas **la** documentation mais **les** documentations. Nous distinguons en effet la *documentation destinée à l'utilisateur et à l'utilisatrice* de la *documentation destinée au développeur et à la développeuse.*

La documentation destinée à l'utilisateur et à l'utilisatrice
: c'est le manuel de l'utilisateur. Ce sont des copies d'écrans et des explications du fonctionnement du programme. C'est l'ensemble des informations nécessaires à l'utilisation du programme.

La documentation destinée au développeur et à la développeuse
: c'est l'explication des méthodes que le développeur ou la développeuse peut être amenée à utiliser.

C'est ce deuxième type de documentation qui nous intéresse dans cette section.

Contenu

1. Cette citation date des années nonantes, dans un article critiquant le langage C (cfr. [Sma90a]) et faisant suite à une critique du langage Pascal (cfr. [Sma90b]) par un développeur Assembleur.

2.1 La documentation

La documentation destinée au développeur ou à la développeuse s'intéresse autant à la personne qui va utiliser le code — *je veux utiliser la méthode* `Math.abs` — qu'à la personne qui va maintenir le code — *comment avais-je écrit cette méthode de tri ?* —.

La personne qui va utiliser le code s'intéresse principalement à « ce que fait » la méthode. La personne qui doit maintenir le code s'intéresse bien sûr à ce que fait la méthode mais également à « comment elle le fait ». Elle a besoin de se rappeler rapidement comment elle avait fait pour *coder* la méthode pour adapter son code plus rapidement. Notons que le développeur ou la développeuse qui va la maintenir n'est peut être pas la même personne que celle qui l'a écrite la première fois. Et c'est très bien.

Penser à la documentation, c'est sans doute d'abord penser à documenter le code que l'on écrit mais c'est également penser à la documentation du code que l'on utilise. Le langage Java vient avec toute une API (cfr. section **??** page ??) que le développeur et la développeuse utilisent quotidiennement et que personne ne connait par cœur et complètement. Il est donc nécessaire d'avoir la possibilité de trouver l'information rapidement et dans un format commun quelle que soit la personne qui a rédigé la documentation.

La pratique montre également que l'écriture de la documentation n'est pas la tâche préférée lorsque l'on développe. Il est donc nécessaire qu'il soit facile d'écrire et de maintenir — parce qu'il faudra également maintenir la documentation comme le code — la documentation. La documentation sera donc écrite avec le code — *litterate programming* — afin de faciliter la maintenance et la synchronisation entre le code et sa documentation.

2.2 La documentation dans le code

Le principe de la « documentation dans le code » (*litterate pragramming*) est très simple :

- ▷ la documentation accompagne le code ;
- ▷ un outil extrait la documentation pour en faire un document facile à lire ;
- ▷ ce document — quelle que soit la personne qui écrit la documentation — suit la même structure, le même format, le même style… et est donc facile à lire.

Coder, c'est éviter de réinventer la roue en réécrivant du code existant car assez générique. Beaucoup d'actions à exécuter sont finalement assez fréquentes : rechercher un minimum, trier, mélanger, ajouter un enregistrement dans une base de données, utiliser une liste d'éléments triés et sans doublons, envoyer un mail, créer un document au format PDF, etc. Une des tâches du développeur et de la développeuse est donc de *savoir* quelle méthode, dans quelle classe utiliser. Comme personne ne *sait* tout, une bonne développeuse ou un bon développeur est une personne qui peut chercher et trouver rapidement. Avoir la documentation rassemblée au même endroit et écrite sous la même forme permet de gagner énormément de temps lorsque l'on cherche et lorsque que l'on lit.

La documentation commence toujours par une description de la classe complète (voir par exemple la figure 2.1) qui précise le rôle de la classe.

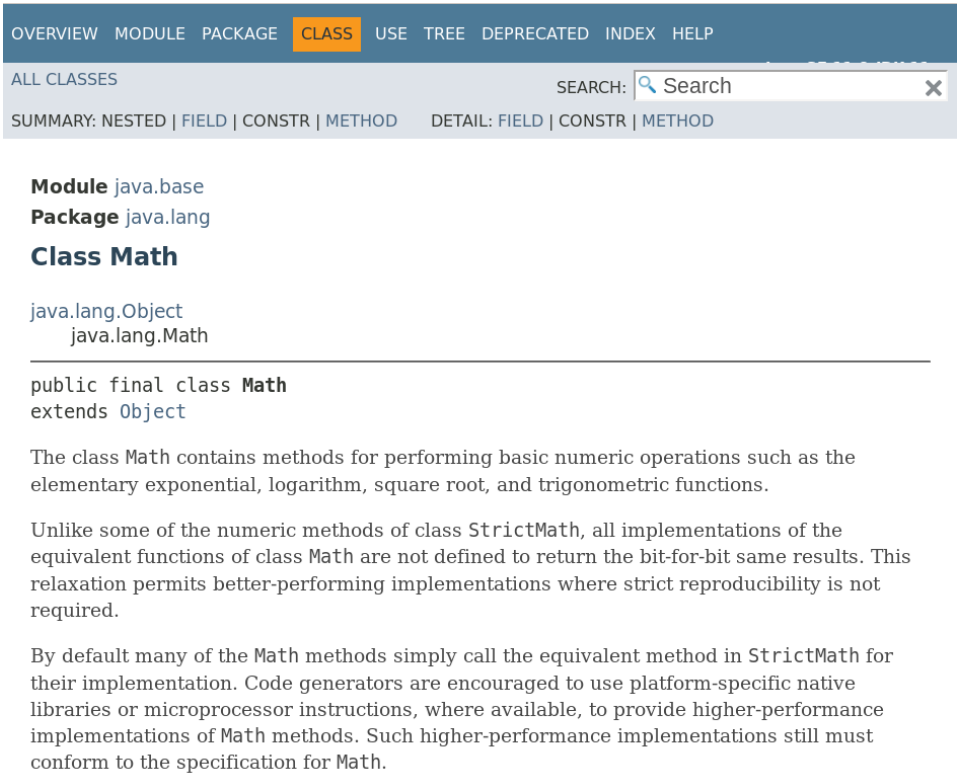

FIGURE 2.1 – Exemple : le haut de page de la documentation de la classe Math pour JDK11

Cette description de la classe est suivie de la liste des méthodes de la classe accompagnée de la description courte de chacune d'entre elles. Plus bas se trouve la description longue de la méthode (voir par exemple la figure 2.2 page suivante).

L'outil `javadoc` permet l'harmonisation de la documentation Java et la génération complète de la documentation.

2.3 Le commentaire `javadoc`

Comme nous l'avons vu à la section **??** page ??, il existe 3 manières d'écrire les commentaires en Java. Seuls les commentaires identifiés par /** */ sont destinés à l'outil javadoc.

Le commentaire *javadoc* se place au dessus de la déclaration de la classe pour la documentation de la classe et au dessus de la déclaration de chaque méthode pour la documentation des méthodes de la classe.

Nous reviendrons sur la **déclaration d'une classe** dans le cours de Développement II[2].

La **déclaration de la méthode** se compose :

- ▷ des *modifiers* comme public et static ;

2. Pour la personne curieuse, la notion de déclaration d'une classe se trouve dans [GJS+17] à la section 8.1

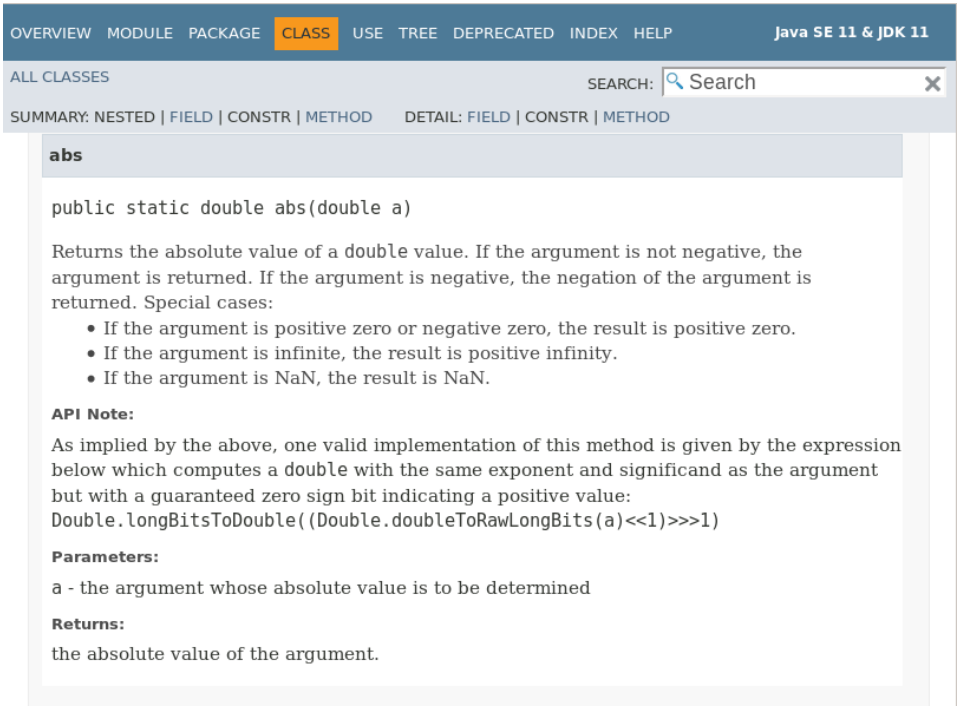

FIGURE 2.2 – Exemple : la description longue de la documentation de la méthode Math.abs pour JDK11

- ▷ du type de retour ;
- ▷ du nom de la méthode
- ▷ de la liste de ses paramètres ;
- ▷ d'une liste éventuelles d'exceptions ;
- ▷ du corps de la méthode.

Le nom de la méthode et la liste des paramètres forment la **signature** de la méthode (cfr. section 8.4.2 de [GJS+17]). *javadoc* utilise la signature de la méthode et son type de retour pour générer la documentation. Sur base des informations contenues dans le commentaire, du type de retour de la méthode et de sa signature, *javadoc* produit des pages html ayant la même forme que celles de la documentation officielle de l'API Java.

Un commentaire comme :

```java
/**
 * Description courte de la méthode terminée
 * par un point.
 *
 * Un peu plus en détail ce que fait la méthode. La description
 * longue peut contenir des balises html comme <strong>écrire
 * en gras</strong> par exemple.
 *
 * @param par1 rôle du paramètre (son type est déduit)
 * @return valeur de retour
 */
public static <Type> <nom>(<params>){
  // Statement
}
```

java

produit si l'on choisit `String` comme type de retour et un paramètre de type `int`, une documentation à l'allure de la figure 2.3,

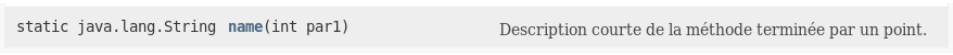

FIGURE 2.3 – Javadoc : description courte

à laquelle correspond la description longue de la figure 2.4.

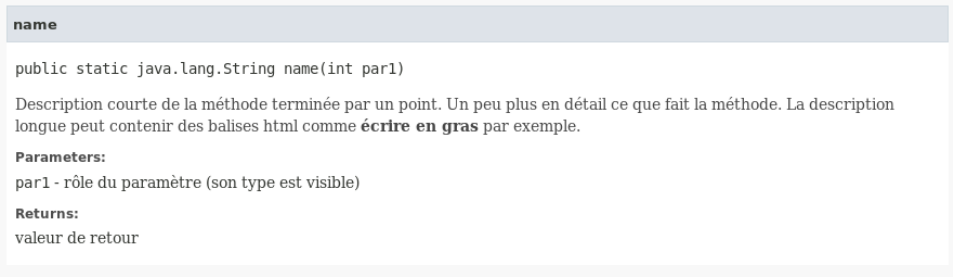

FIGURE 2.4 – Javadoc : description longue

2.3.1 Les tags

Afin de mettre correctement la documentation en forme, *javadoc* utilise des **tags**. Ce sont des mots commençant par un *arobase* — @ — qui sont reconnus par la commande `javadoc`. Certains sont requis, d'autres optionnel. Certains sont destinés à la documentation de la classe [3] d'autres aux méthodes.

Nous retiendrons ces *tags* placés idéalement dans cet ordre :

@author
: renseignant le ou les auteur · es de la classe. Un *tag* par auteur · e. Dans le cadre d'un développement commun, l'auteur peut être un groupe. Ce tag ne génère pas de *javadoc*, il n'est visible que dans la classe. (*classes*, *requis*) ;

@version
: comprend la date de modification et un numéro de version commençant à 1.1. (*classes*, *requis*) ;

3. Le terme « classe » est ici dans son sens large et comprend : les classes, les interfaces et les énumérations comme nous le verrons en Développement II

Par exemple *1.39, 1996-01-23*

@param
décrivant chaque paramètre de la méthode. Il y en a un par paramètre ; (*méthodes*, *requis*) ;

@return
décrivant ce que retourne la méthode ; (*méthodes*, *requis* excepté si la méthode ne retourne rien) ;

@throws
listant les exceptions éventuellement lancées par la méthode ; (*méthodes*) ;

@see
renseignant une autre classe ou une autre méthode relative à ce qui est documenté.

2.3.2 Le style

Voici quelques règles de style [4] pour la rédaction de la *javadoc*.

1. Seules les classes et les méthodes publiques sont commentées.
2. Les mots clés (*keywords*), les noms de classes, de méthodes sont écrit entre les balises `<code> </code>`
3. On préfèrera la 3^e personne à la 2^e personne.
4. La description d'une méthode commence par un verbe parce qu'elle implémente habituellement une opération, une action.

   Gets the label of this button.
   sera préféré à
   This method gets the label of this button.
5. Ajouter le commentaire au nom de la méthode sans répéter le nom de la méthode. Normalement le nom de la méthode est choisi pour représenter ce qu'elle fait. Il est donc inutile de répéter ces mots dans la documentation. La documentation doit idéalement apporter une plus-value au nom de la méthode.

   Éviter cette description qui ne fait que répéter les mots contenus dans le nom de la méthode.

```java
/**
 * Sets the tool tip text.
 *
 * @param text the text of the tool tip
 */
public void setToolTipText(String text)
```

Préférer cette description qui définit ce qu'est une infobulle (*tool tip*) dans un contexte plus large.

4. Ces règles sont principalement issues de `https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`.

```java
/**
 * Registers the text to display in a tool tip.
 *
 * The text displays when the cursor lingers over
 * the component.
 *
 * @param text the string to display. If the text
 * is null, the tool tip is turned off for
 * this component.
 */
public void setToolTipText(String text)
```

java

6. Le tag `@link` peut être utilisé... avec parcimonie car un lien attire l'attention. Il ne faut pas en abuser. Ce tag s'écrit dans le texte et fait référence vers l'API.
7. Lorsqu'une référence à une autre méthode est faite, on omet les parenthèses `add` ou on écrit le type des paramètres `add(int, Object)` sans nom pour les paramètres.
8. Lors de la description éventuelle d'une variable, omettre le sujet dans un soucis de brièveté.

   Counter of vowels
   sera préféré à
   This is a counter of vowels.
9. Éviter les abréviations qui pourraient ne pas être comprises par tous et toutes.

2.4 L'outil `javadoc`

Il est possible de générer la *javadoc* d'un simple clic à partir d'un IDE comme il est possible de le faire à partir d'une console par :

```
$
javadoc MyClass.java
```

terminal

Certaines options sont intéressantes :

-d <directory>
: demande de générer la documentation dans le répertoire `directory` plutôt que dans le répertoire courant. Crée le répertoire au besoin ;

-charset utf-8
: ajoute du code html permettant la prise en charge de l'utf-8 ce qui aura pour effet que les caractères accentués seront correctement affichés dans la documentation.

Pour générer toutes la documentation en une seule fois en utilisant ces options, on préfèrera donc une commande du style :

```
$
javadoc *.java -d doc -charset utf-8
```
terminal

Et si les classes ne sont pas toutes dans le répertoire courant, pourquoi ne pas demander à son shell de les chercher ?

```
$
javadoc $(find . -name '*.java') -d doc -charset utf-8
```
terminal

Références

[GJS+17] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, and Daniel Smith. *The Java Language Specification. Java SE 9 Edition*. Oracle America Inc., 2017.

[Sma90a] David Small. Hérésie. deuxième partie. `http://valvassori.free.fr/dave_small/heresie2.php3`, ±1990.

[Sma90b] David Small. Hérésie. première partie. `http://valvassori.free.fr/dave_small/heresie1.php3`, ±1990.

Table des matières