



- Java -

L'utilitaire **javadoc**  
pour une documentation efficace





# SOMMAIRE

✓ 1 - <a href="#">Introduction</a>	2
✓ 2 - <a href="#">Présentation de l'outil javadoc</a>	3
✓ 3 - <a href="#">Mise en œuvre de l'outil javadoc</a>	5
✓ 4 - <a href="#">Quelques conventions</a>	10
✓ 5 - <a href="#">Tableaux des principaux tags javadoc</a>	11
✓ 6 - <a href="#">Détails des principaux tags javadoc</a>	12
✓ 7 - <a href="#">Quelques conseils</a>	21
✓ 8 - <a href="#">Les fichiers générés</a>	22
✓ 9 - <a href="#">Le projet <i>DémonstrationJavadoc</i></a>	24
✓ 10 - <a href="#">Résumé</a>	26

# - 1 - Introduction

## Rappel

Dans la phase de codage du cycle de vie, lié au **développement d'applications informatiques**, il n'y a pas que des activités strictement réservées à la génération de code.

En effet, professionnellement parlant, il y a une activité que l'on ne doit pas négliger : c'est la **phase de documentation** du code généré pour l'ensemble de l'application.

Cette activité est traditionnellement et malheureusement **très (trop) souvent négligée**, ce qui peut poser des problèmes dans une équipe de développement, notamment lors de la reprise de ce code. Aujourd'hui, **dans un souci de qualité**, il est impératif de documenter son code **simultanément** à la phase d'écriture de ce code, et non plus après.

Cet aspect est particulièrement justifié de nos jours avec la **montée inexorable de la complexité** des applications informatiques.



Plusieurs raisons militent en faveur de la **génération de commentaires** dans votre code :

1. La **validation** du code généré : le fait de **rédigier la description** de vos classes et de vos méthodes vous oblige à **réfléchir** et éventuellement **reprendre** la définition de votre code source. Vous comprenez et **entérinez** ce que vous avez écrit. En quelque sorte, cet « auto-comité de relecture » que vous mettez en œuvre approuve le code dont vous êtes l'auteur.
2. Si vous travaillez en équipe, votre code peut être relu, modifié, maintenu par d'autres développeurs : il est alors indispensable de faciliter leur lecture en les informant sur les choix et principes que vous avez mis en œuvre. Il faut que votre code leur soit le plus facilement et rapidement accessible. Une documentation bien faite, claire et structurée favorisera alors **l'appropriation de votre code**.
3. Fournir à un client une application informatique avec une documentation technique rigoureuse, sous un format universel, facilite énormément la **communication**.
4. Documenter votre code devient, pour toutes ces raisons, une étape incontournable dans la création et la maintenance de vos futurs programmes **Java**.

Cette activité de documentation est d'autant plus intéressante et séduisante que le **JDK** propose en standard un outil permettant de faciliter la vie du programmeur : **javadoc**.

**javadoc** permet de générer la **documentation technique complète** d'un projet **Java**.

Le principe de base de cet outil est très simple : il analyse votre code source et en extrait des informations pour générer une documentation, riche, conviviale et au format **HTML** avec une mise en forme CSS riche.

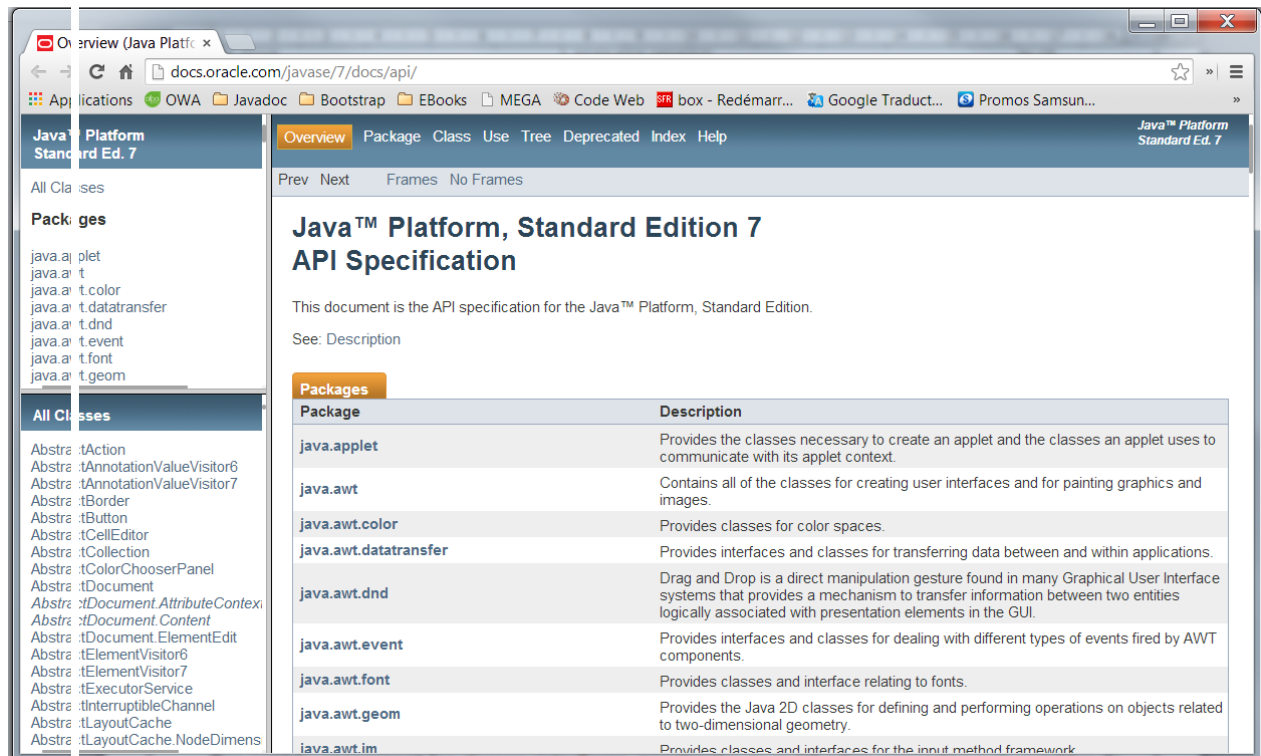


Figure 1 : La page d'accueil de la documentation Java SE 7.



## -2 - Présentation de l'outil javadoc

### Introduction



**javadoc** est un outil créé par *Sun* (devenue *Oracle*) qui vient avec l'installation du **JDK** et que l'on retrouve dans le répertoire des exécutables (`C:\Program Files\Java\jdk.x.x.x\bin`).

Il a été conçu pour permettre au développeur d'insérer des commentaires dans son code **Java**. L'insertion de ces commentaires particuliers va être exploitée par **javadoc** afin de produire et d'automatiser une documentation technique au format **HTML**.

**javadoc** produit donc une **documentation très détaillée** de votre code source **Java** avec notamment :

- une page d'index générale permettant une navigation facilitée au sein des classes.
- une description complète pour chaque classe, ses méthodes et ses variables...
- une liste complète des classes et interfaces, la hiérarchie des classes impliquées.
- des références interclasses croisées **sous forme de liens hypertexte**.

### Principes

Le principe de documenter son code en même temps qu'on le génère et ce, **dans le même fichier**, est très pratique et rend la documentation générée **inévitablement très liée au code qu'elle décrit**.

Les commentaires « collent » au maximum aux subtilités de la programmation qu'ils décrivent.

**javadoc** présente aussi l'avantage de **standardiser** la documentation **Java**. Le format, la présentation et l'exploitation des commentaires sont universels et tout développeur **Java** acquiert des réflexes pour consulter toute documentation **Javadoc**.

La documentation même de **Java** ( *Java SE* et *Java EE* ) a été produite avec l'outil **javadoc** et représente une excellente démonstration de ce que peut faire l'outil.

Le terme **javadoc** est souvent utilisé pour désigner l'outil (*javadoc.exe*) mais aussi pour évoquer la documentation **HTML** résultant de son action.

L'avantage principal de **javadoc** est évident : réunir dans un même contenant le code technique et la documentation associée à ce code. Ce contenant étant bien sûr le fichier source **Java**.

## - 3 - Mise en œuvre de l'outil *javadoc*

### Installation

Comme précisé ci-dessus, *javadoc* vient avec le JDK et, en conséquence, aucune installation n'est nécessaire.

### Introduction

*javadoc* explore donc l'ensemble du code source d'un projet *Java*, y recherche les éventuels commentaires spécifiques (que l'on va commenter ci-après) afin d'obtenir des *précisions supplémentaires* sur les entités composant votre code source.

Ces entités concernent les *classes*, *interfaces*, *constructeurs*, *méthodes*, *paramètres* et *variables* de retour.

A l'issue de l'exploitation de tous ces éléments, *javadoc* génère une documentation au format *HTML* de toutes ces entités en [créant des liens hypertextes pour naviguer entre elles](#).

Cette documentation générée peut donc être *singulièrement enrichie* grâce aux commentaires *javadoc* dédiés et déposés dans votre code, juste au-dessus de l'*entité Java* (classe, interface, méthode, constructeur, variable d'instance) dont vous voulez préciser le rôle, l'objectif ou le fonctionnement.

### Les commentaires *Javadoc*

Les commentaires *javadoc* obéissent à une *syntaxe rigoureuse*. Rappelons que le langage *Java* admet trois types de commentaires, dont la troisième forme est précisément dédiée à *javadoc*.

Commentaires	Description
<code>/* texte */</code>	Le compilateur ignore tout texte entre <code>/*</code> et <code>*/</code> . Ce type de commentaire est appelé « <b>bloc</b> ».
<code>// texte</code>	Le compilateur ignore tout texte à partir de <code>//</code> et jusqu'à la fin de la ligne. Ce type de commentaire est appelé « <b>fin de ligne</b> ».
<code>/** documentation */</code>	C'est un commentaire de type documentation. <i>javadoc</i> utilise et explore ce type de commentaires pour générer la documentation <b>HTML</b> globale.

En utilisant les deux premiers commentaires usuels ( *bloc* et *fin de ligne* ) dans le tableau précédent, **Java** rend possible l'insertion de commentaires concernant la description de toute entité **Java**.

**javadoc** ignore complètement ces deux types de commentaires et ne s'en préoccupe pas.

**Sun** a conçu et mis à disposition des développeurs des commentaires spécifiques, destinés à **javadoc**. Ces commentaires dits « *de documentation* » possèdent le format général suivant :

```
/**
 * Début du commentaire
 * Chaque ligne dans le bloc débute par *
 * Fin du commentaire
 */
```

- Entre ces deux repères, on peut insérer tout texte pertinent *en tant que commentaire*. Mais on a également la possibilité d'y insérer des *balises* ou *tags* particuliers propres à **javadoc**.
- Ces tags **javadoc** vont préciser et détailler chaque élément (classe, interface, méthode, etc) sur lequel ils s'appliquent et génèrent des rubriques particulières dans la documentation résultante que l'on va découvrir.

Vous produisez ainsi une **formulation informelle** (votre commentaire) et une **formulation formelle** (votre code).

Après avoir terminé de coder chaque entité de codage, il est recommandé de relire l'ensemble pour vérifier si votre formulation formelle (votre code) **n'est pas en contradiction** avec votre formulation informelle.

Dans le code, chaque commentaire **javadoc** doit être apposé **immédiatement au-dessus** de l'entité qu'il précise.

- Le format de ces commentaires commence donc par */\*\** et se termine par *\*/*.
- Il peut contenir un texte libre et des balises particulières que l'on appelle des « **tags javadoc** ».

Le commentaire est très souvent fourni sur plusieurs lignes. Chaque ligne entre le début du commentaire et la fin débute par un *\**. Dans ce commentaire **javadoc**, on peut y insérer du **HTML**, en veillant tout particulièrement à respecter une certaine sobriété.

Cette méthode, au départ contraignante, s'avère extrêmement utile. Si l'on s'impose cette façon de travailler, on peut parfois déceler des erreurs *avant même de commencer à tester le code*.



Le commentaire **javadoc** ressemble alors au format suivant :



```
/**
 * @
 * @
 */
```

Les caractères @ sont suivis par les **tags javadoc**.

Nous allons présenter un exemple de code où figurent les trois formes de commentaires : les deux usuels et celui spécifique à **javadoc**.

Cet exemple concerne la classe *Calculatrice* qui nous servira tout au long de notre étude pour mener toutes nos expérimentations sur **javadoc**.



```
/**
 * Ce commentaire est dédié à javadoc. Il débute par /** et se termine par */.
 * Il est placé juste au-dessus de la classe Calculatrice : il la décrit.
 * Chaque ligne débute par une étoile.
 * On peut y insérer des tags HTML qui produiront leur effet dans la documentation résultante.
 * Projet : <b>Calculatrice</b>
 * <p>Cette classe fournit une méthode de classe modélisant les quatre opérations arithmétiques<p>
 * Elle propose également une autre méthode de classe permettant le calcul de la factorielle d'un nombre
 * entre 0 et 20 !!!!!
 * @author Michel-HP
 */
public class Calculatrice { // La classe principale
/**
 * Cette méthode de classe permet d'effectuer les 4 opérations arithmétiques de base.
 * @param operateur : le type de calcul souhaité
 * @param var1 : le premier opérande
 * @param var2 : le second opérande
 * @return Le résultat du calcul
 * @throws com.michel.exceptions.CalculException
 */
public static double calculer( char operateur, double var1, double var2) throws CalculException {
....
}
...
}
```

Listing 1 : Exemple de commentaires **javadoc** sur la classe *Calculatrice*.

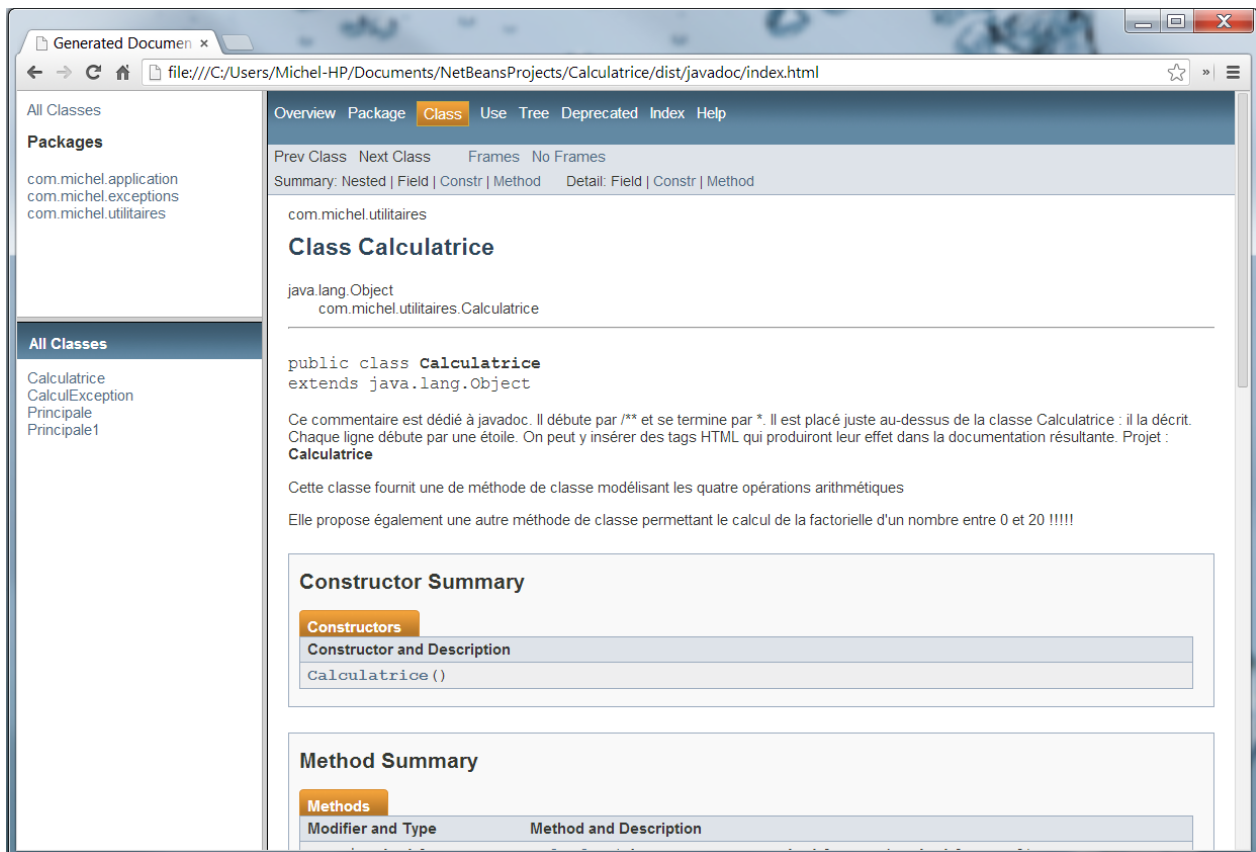


Figure 2 : Le commentaire *javadoc* décrivant la classe *Calculatrice* du listing 1.

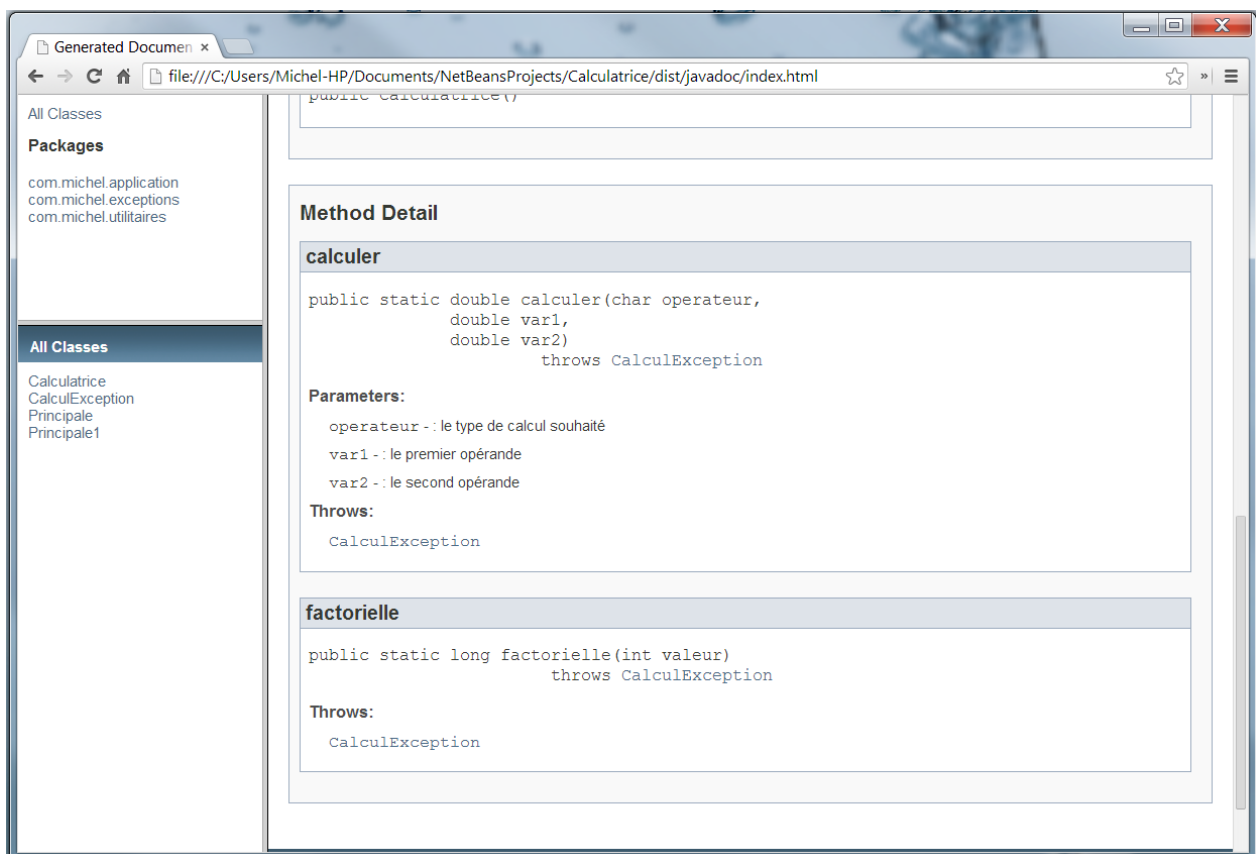


Figure 3 : Le commentaire *javadoc* décrivant la méthode *calculer* du listing 1.

## Plus de précisions

**javadoc** exploite donc les commentaires `/**` et `*/` qui lui **sont dédiés** dans les fichiers sources **java**. Il peut aussi exploiter d'autres fichiers comme des fichiers **HTML**, images, pour générer une documentation complète dont les pages résultantes sont liées par des liens hypertexte.

En explorant les fichiers sources **java**, **javadoc** lit et exploite les commentaires « *javadoc* » insérés juste au-dessus de chaque entité qu'ils décrivent par du texte.

## - 4 - Quelques conventions

Le commentaire **javadoc**, placé immédiatement au dessus de l'élément décrit (classe, interface, constructeur, méthode, ou propriété) commence par une description générale de cet élément.

Cette description courte a pour **vocation d'être un résumé**. Ce bloc se termine par le caractère '.' suivi d'un séparateur (espace ou tabulation ou retour chariot) ou à la rencontre du premier **tag javadoc**.

Le texte du commentaire est du **HTML**. Vous pouvez donc utiliser des balises **HTML** pour enrichir le formatage de votre documentation technique. Il est particulièrement commun d'utiliser le tag **<p>** pour générer des paragraphes et le tag **<code>** pour mettre en exergue des extraits représentatifs de votre code.

Au sein du bloc des commentaires **javadoc**, on peut y insérer des mots-clés spécifiques à **javadoc**. Ces tags particuliers débutent tous par le caractère **@** et doivent apparaître au début d'une ligne, soit derrière le caractère '\*'.

Si plusieurs de ces mots-clés doivent être fournis, alors ils le sont **en un seul bloc**, les uns derrière les autres.

Ces tags permettent d'enrichir finement la description d'un élément particulier et de générer selon un format particulier cette description.

Avant de parcourir ces mots-clés particuliers, citons quelques-uns des éléments usuels du code qui peuvent bénéficier d'un tel traitement .

Les paramètres d'une méthode : **@param**  
La valeur retournée par une méthode : **@return**  
La (les) exception(s) levées par une méthode : **@throws**  
La version actuelle d'une classe : **@version**  
L'auteur de la classe : **@author**  
...

### **Javadoc** distingue deux types de tag :

- Les *Block tags* : leur format est **@tag** et ils sont groupés ensemble en début de ligne comme dans l'exemple de la [méthode calculer du listing 1](#).
- Les *Inline tags* : ils peuvent apparaître n'importe où dans le bloc de commentaire. Ils obéissent au format suivant : **{@tag}**

## - 5 - Tableau des principaux tags *javadoc*

Nom du tag	Description
@author	Le nom du ou des auteurs de l'élément concerné.
{@ extrait de code}	Equivalent à <code>&lt;code&gt;extrait de code&lt;/code&gt;</code>
@deprecated	Précise que l'élément est déprécié. Informer à partir de quelle version.
@throws	Précise la ou les exceptions(s) pouvant être levées.
@param	Précise le nom et le rôle de chaque paramètre d'une méthode ou d'un constructeur.
@return	Précise et décrit la valeur de retour d'une méthode renvoyant autre chose que <i>void</i> .
@see	Précise un élément de codage en lien avec l'élément documenté.
@since	Précise à partir de quelle version l'élément concerné a été ajouté.
@version	Précise le numéro de version actuelle d'une classe ou interface.
{@link}	Génère un lien vers un élément de code dans la documentation ( classe, interface, méthode, constructeur, ...)

Figure 4 : Les principaux tags *javadoc*.

Il est recommandé de consulter la [documentation officielle d'Oracle](#) pour explorer l'ensemble complet des tags *javadoc*. Vous y découvrirez notamment que certains de ces tags ne sont applicables que sur certains éléments et pas d'autres.

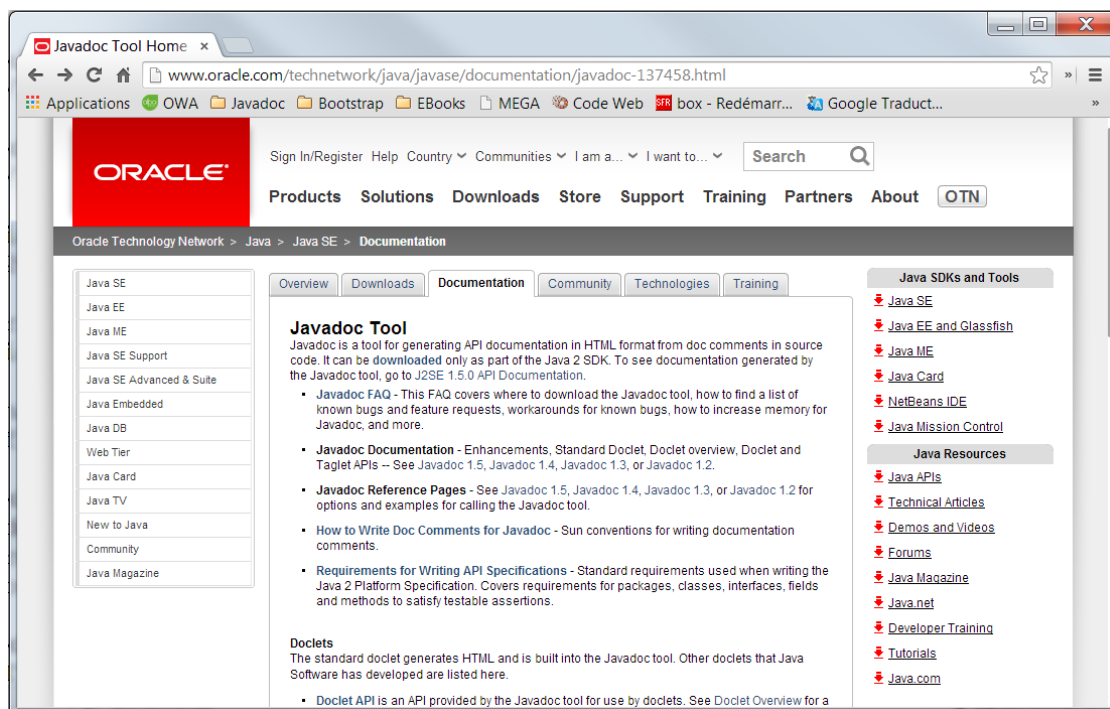


Figure 5 : Présentation de l'outil *javadoc* par Oracle.

## - 6 - Détails des principaux tags javadoc

Nous allons maintenant commenter, avec exemple à l'appui, le rôle des tags les plus usités .

### @author

- Précise l'auteur de l'entité commentée. Ne s'utilise que sur une **classe** ou une **interface**.
- Syntaxe : **@author** texte-descriptif.
- Exemple :

```
package com.michel.utilitaires;
/**
 * Cette classe fournit une de méthode de classe modélisant les quatre
 * opérations arithmétiques.<p>
 * Elle propose également une autre méthode de classe permettant le calcul
 * de la factorielle d'un nombre entre 0 et 20 !!!!!
 * @author Michel Dupont
 */
public class Calculatrice {
    ....
}
```

## @deprecated

- Précise le caractère déprécié de l'élément commenté (classe, interface, constructeur, méthode, propriété).
- L'usage veut que l'on précise depuis quelle version l'élément est déprécié et quelle solution de substitution est mise en place. Le tag complémentaire **@see** générera alors un lien vers l'élément à utiliser.
- Syntaxe : **@deprecated** texte-descriptif
- Génère une rubrique **Deprecated** dans la documentation résultante.
- Exemple :

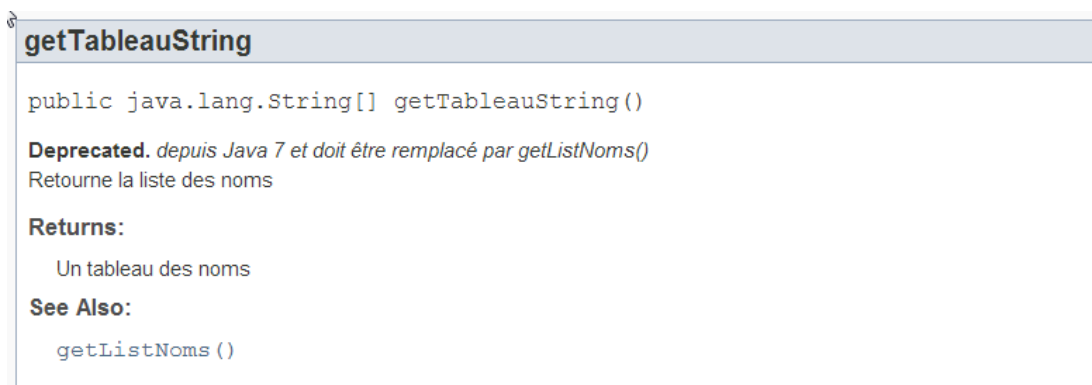
```
package com.michel.utilitaires;

public class Calculatrice {
    ....
    /**
     * Retourne la liste des noms.
     * @return un tableau des noms
     * @deprecated depuis Java 7 et doit être remplacé par getListNoms()
     * @see Calculatrice#getListNoms()
     */
    public String[] getTableauString() {
        return new String [1] ;
    }
    public List<String> getListNoms() {
        return new ArrayList<>();
    }
    ...
}
```

A noter que Java souligne la méthode pour montrer qu'elle est dépréciée. On peut , alternativement utiliser l'expression suivante :

```
* @deprecated veuillez utiliser {@link #getListNoms() } ()
```

... qui va générer un lien dans la documentation sur la méthode de remplacement préconisée `getListNoms()` dans la rubrique *See Also* .



```
getTableauString

public java.lang.String[] getTableauString()

Deprecated. depuis Java 7 et doit être remplacé par getListNoms()
Retourne la liste des noms

Returns:
    Un tableau des noms

See Also:
    getListNoms()
```

Figure 6 : Extrait de la documentation concernant l'exemple **@deprecated**.

## @throws

- Précise la ou (les) exceptions que peut lever la méthode ou le constructeur sur lequel le tag s'applique.
- Syntaxe : **@throws** *NomException* texte-descriptif
- Si plusieurs exceptions sont concernées, il faut préciser autant de **@throws**.
- Génère une rubrique **Throws** dans la documentation résultante.
- Exemple :

```
package com.michel.utilitaires;

public class Calculatrice {
    ....
    /**
     * Lit et affiche le fichier dont le nom est transmis en argument
     *
     * @param fichier Le nom du fichier à exploiter
     * @throws java.io.FileNotFoundException }Le fichier fourni n'existe pas
     * @throws MalformedURLException L'URL choisie est mal formée
     */
    public void lireFichier ( String fichier) throws FileNotFoundException,
        MalformedURLException{
        .....
    }
    ....
}
}
```

### lireFichier

```
public void lireFichier(java.lang.String fichier)
    throws java.io.FileNotFoundException,
           java.net.MalformedURLException
```

Lit et affiche le fichier dont le nom est transmis en argument

#### Parameters:

fichier - Le nom du fichier à exploiter

#### Throws:

java.io.FileNotFoundException - Le fichier fourni n'existe pas

java.net.MalformedURLException - L'URL choisie est mal formée

Figure 7 : Extrait de la documentation concernant l'exemple **@throws**.



## @param

- Précise le ou (les) paramètres reçus par une méthode ou un constructeur.
- **Syntaxe** : `@param param1` texte-descriptif
- Si plusieurs paramètres sont prévus, il faut préciser autant de `@param`.
- **NB** : Ne fournir que le nom du paramètre : sa classe sera automatiquement insérée dans la documentation résultante.
- Génère une rubrique **Parameters** dans la documentation résultante.
- **Exemple** :

```
package com.michel.utilitaires;

public class Calculatrice {
    ....
    /**
     * Cette méthode de classe permet d'effectuer les 4 opérations arithmétiques de base.
     * S'emploie par {@code Principale.calculer('+', 12, 4 )}
     * @param operateur : le type de calcul souhaité
     * @param var1 Le premier opérande
     * @param var2 Le second opérande
     * @return Le résultat du calcul
     * @throws com.michel.exceptions.CalculException
     */
    public static double calculer( char operateur, double var1, double var2) throws CalculException {
        ...
    }
    ...
}
```

### calculer

```
public static double calculer(char operateur,
                             double var1,
                             double var2)
    throws CalculException
```

Cette méthode de classe permet d'effectuer les 4 opérations arithmétiques de base. S'emploie par `Principale.calculer('+', 12, 4 )`

#### Parameters:

operateur - : le type de calcul souhaité  
var1 - Le premier opérande  
var2 - Le second opérande

#### Returns:

Le résultat du calcul

#### Throws:

CalculException

Figure 8 : Extrait de la documentation concernant l'exemple `@param`.

## @return

- Précise l'éventuelle valeur de retour d'une méthode qui renvoie autre chose que *void*.
- Syntaxe : **@return** texte-descriptif-décrivant-la-valeur-de-retour.
- Génère une rubrique **Returns** dans la documentation résultante.
- Exemple :

```
package com.michel.utilitaires;

public class Calculatrice {
    ....
    /**
     * Cette méthode de classe permet d'effectuer les 4 opérations arithmétiques de base.
     * S'emploie par {@code Principale.calculer('+', 12, 4 )}
     * @param operateur : le type de calcul souhaité
     * @param var1 Le premier opérande
     * @param var2 Le second opérande
     * @return Le résultat du calcul
     * @throws com.michel.exceptions.CalculException
     */
    public static double calculer( char operateur, double var1, double var2) throws
    CalculException {
        ...
    }
    ...
}
```

**calculer**

```
public static double calculer(char operateur,
    double var1,
    double var2)
    throws CalculException
```

Cette méthode de classe permet d'effectuer les 4 opérations arithmétiques de base. S'emploie par `Principale.calculer('+', 12, 4 )`

**Parameters:**

- operateur - : le type de calcul souhaité
- var1 - Le premier opérande
- var2 - Le second opérande

**Returns:**

- Le résultat du calcul

**Throws:**

- CalculException

Figure 9 : Extrait de la documentation concernant l'exemple **@return**.

## @see

- Génère un renvoi **sous forme de lien** vers un autre élément de la documentation ou bien encore vers une URL externe. Cet élément commenté peut être un package , une classe, une propriété de la classe courante, une propriété d'une autre classe, une méthode de la classe courante, une autre méthode d'une autre classe, ....
- Génère une rubrique **See Also** : dans la documentation résultante.
- Syntaxes possibles (parmi d'autres) :
  - @see package
  - @see package.MaClasse
  - @see #propriété
  - @see MaClasse#propriété
  - @see #méthode ( type 1, type 2, ...)
  - @see MaClasse#méthode ( type 1, type 2, ...)
  - @see <a href= « URL »> texte...</a>
  - .....
- Exemples :

```
package com.michel.utilitaires;

public class Calculatrice {
    ....
    /**
     * @see com.michel.exceptions
     * @see com.michel.application.Principale
     * @see #cpt
     * @see com.michel.application.Principale#main(java.lang.String[])
     * @see #lireFichier(java.lang.String)
     * @see <a href="http://docs.oracle.com/javase/7/docs/api/"> Vers la Javadoc Oracle ...</a>
     */
    public void testSee() {
        ....
    }
    ...
}
```

### testSee

```
public void testSee()
```

#### See Also:

```
com.michel.exceptions, Principale, cpt, Principale.main (java.lang.String[]),
lireFichier (java.lang.String), Vers la Javadoc Oracle ...
```

Figure 10 : Extrait de la documentation concernant l'exemple @see.

## @since

- Précise le numéro de version de classe depuis laquelle l'élément commenté a été ajouté.
- Syntaxe : `@since` texte-descriptif
- Génère une rubrique **Since** dans la documentation résultante.
- Exemple :

```
package com.michel.utilitaires;

public class Calculatrice {
    ....
    /**
     * @since V3.1.2
     */
    public void testSince() {
        ...
    }
    ....
}
}
```

### testSince

```
public void testSince()
```

#### Since:

V3.1.2

Figure 11 : Extrait de la documentation concernant l'exemple *@since*.

## @version

- Précise le numéro de version actuelle de la classe ou de l'interface.
- **Syntaxe** : `@version` texte-descriptif
- Génère une rubrique **Version** dans la documentation résultante.
- **Exemple** :

```
/**
 * Ce commentaire est dédié à javadoc. Il débute par /** et se termine par * /
 * Il est placé juste au-dessus de la classe Calculatrice : il la décrit.
 * Chaque ligne débute par une étoile.
 * On peut y insérer des tags HTML qui produiront leur effet dans la documentation
 * résultante.
 * Projet : <b>Calculatrice</b>
 * <p>Cette classe fournit une de méthode de classe modélisant les quatre
 * opérations arithmétiques<p>
 * Elle propose également une autre méthode de classe permettant le calcul
 * de la factorielle d'un nombre entre 0 et 20 !!!!!
 * @author Michel-HP
 * @version 2.1.2
 */
public class Calculatrice {
}
```

com.michel.utilitaires

### Class Calculatrice

java.lang.Object  
com.michel.utilitaires.Calculatrice

```
public class Calculatrice
extends java.lang.Object
```

Ce commentaire est dédié à javadoc. Il débute par /\*\* et se termine par \* / Il est placé juste au-dessus de la classe Calculatrice : il la décrit. Chaque ligne débute par une étoile. On peut y insérer des tags HTML qui produiront leur effet dans la documentation résultante. Projet : **Calculatrice**

Cette classe fournit une de méthode de classe modélisant les quatre opérations arithmétiques

Elle propose également une autre méthode de classe permettant le calcul de la factorielle d'un nombre entre 0 et 20 !!!!!

#### Version:

2.1.2

#### Author:

Michel-HP

Figure 12 : Extrait de la documentation concernant l'exemple `@version`.

## `{@link}`

- Génère un lien vers tout élément de la documentation résultante.
- Syntaxe : `{@link package.MaClasse#propriété}`
- La différence avec le tag `@see` c'est que le lien généré n'apparaît pas dans la rubrique « *See Also* : » mais où bon vous semble dans la documentation.
- Exemple :

```
public class Calculatrice {  
    /**  
     * Allez sur ...  
     * {@link com.michel.application.Principale#main(java.lang.String[])}  
     */  
    public static long factorielle ( int valeur) throws CalculException {  
        ...  
    }  
}
```

### factorielle

```
public static long factorielle(int valeur)  
                        throws CalculException
```

Allez sur ... `Principale.main(java.lang.String[])`

#### Throws:

`CalculException`

Figure 13 : Extrait de la documentation concernant l'exemple `{@link}`.

## - 7 - QUELQUES CONSEILS

Les commentaires **javadoc** sont particulièrement riches et permettent de générer une documentation rigoureuse et **extrêmement bien structurée**. Les quelques conseils suivants contribueront à viser une très bonne qualité de vos documentations techniques à venir :

- Le texte décrivant les classes et autres interfaces doit **clair, explicatif et concis**.
- Ce principe est également applicable aux autres éléments comme les constructeurs, méthodes, ...
- Systématisez la présence des tags **@param** et **@return** : ils permettront à vos lecteurs de connaître rapidement les arguments requis par votre méthode et la fonction de sa valeur de retour. Ce point est très important.
- Concernant le code technique de vos méthodes : si celui-ci, de part sa complexité, mérite d'être commenté, précisez les choix algorithmiques que vous avez mis en œuvre.
- **javadoc** utilisant **HTML** pour générer votre future documentation, vous pouvez y insérer des balises **HTML** pour formater votre texte. Néanmoins : la sobriété est de rigueur. Laissez de côté toute volonté d'utiliser des polices exotiques, des couleurs psychédéliques, ...
- Seuls, les tags **HTML** apportant des éléments structurant sont appréciés :  
`<b>, <i>, <ul>, <li>, <p>....`
- La feuille de styles **CSS** utilisée par **javadoc** suffit amplement pour présenter joliment votre documentation et garantit une certaine unité pour l'ensemble d'entre elles.

## - 8 - Les fichiers générés

Afin de générer votre documentation technique avec *NetBeans*, il vous suffit de cliquer-droit sur le nom de votre projet et de lancer la commande « *Generate Javadoc* ».

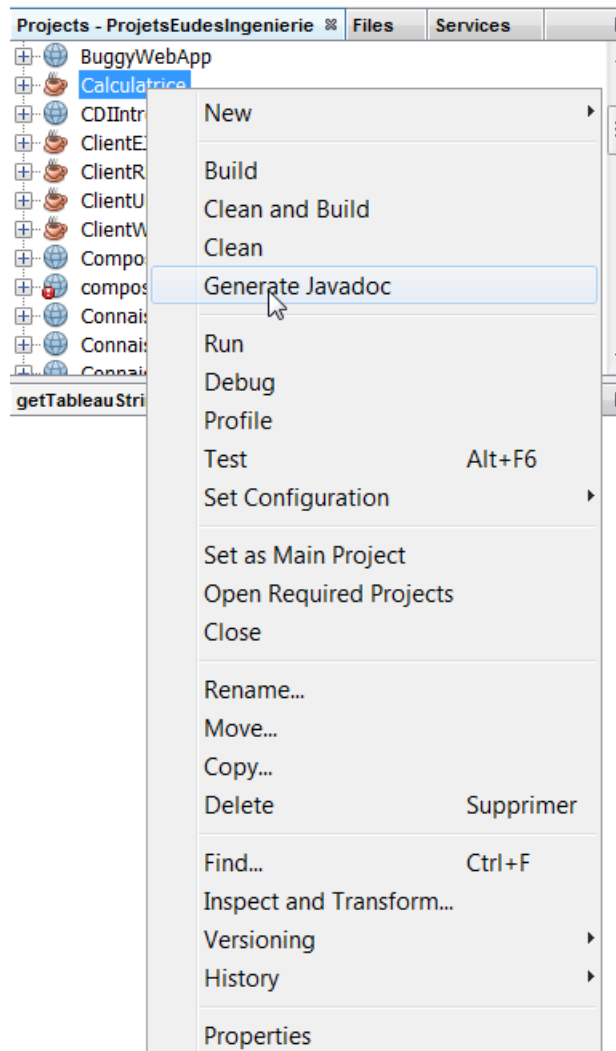


Figure 14 : Générer la *javadoc* d'un projet Java avec *NetBeans*.

Cette action va produire de nombreux fichiers et présenter spontanément une fenêtre d'index organisée comme celle de la *javadoc* de le l'API de *Java* elle-même, avec trois frames réparties de la façon suivante : en haut à gauche la liste des packages du projet, en-dessous : la liste des classes et interfaces du package choisi et enfin la composition de chaque classe/interface sur laquelle on a cliqué dans la frame de droite.

L'opération va générer un certain nombre de fichiers que l'on retrouve dans *NetBeans* à partir de l'onglet *Files* dans les répertoires *dist* et *javadoc*.



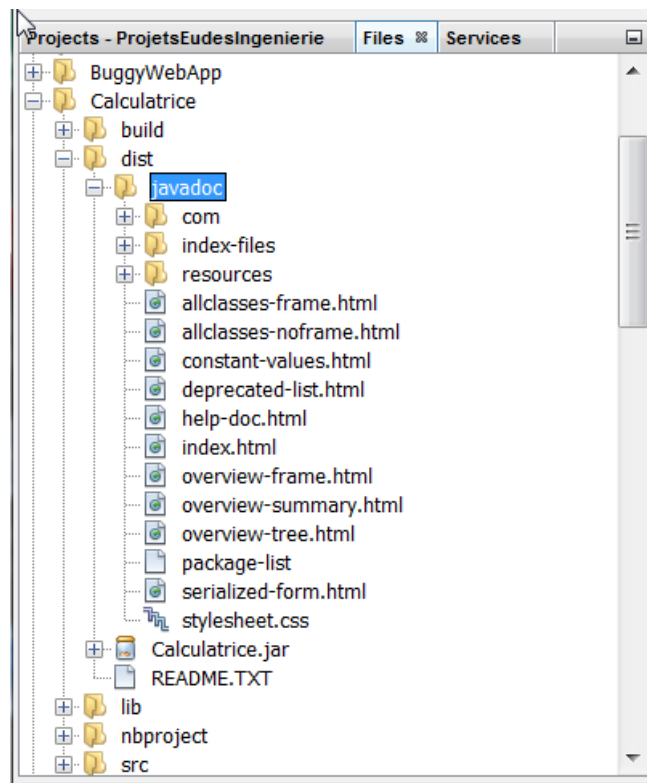


Figure 14 : Les fichiers générés par *javadoc*.

L'outil a créé beaucoup de fichiers parmi lesquels :

- un fichier html par classe/interface précisant chacun de ses éléments.
- un fichier html par package décrivant son contenu.
- un fichier *overview-summary.html*.
- un fichier *overview-tree.html*.
- un fichier *deprecated-list.html*.
- un fichier *serialized-form.html*.
- un fichier *overview-frame.html*.
- un fichier *all-classe.html*.
- un fichier *package-summary.html* pour chaque package.
- un fichier *package-frame.html* pour chaque package.
- un fichier *package-tree.html* pour chaque package.

Nous vous recommandons à partir de la page principale *index.html* de naviguer et d'apprécier la richesse des informations ainsi structurées.

## - 9 - Le projet *DemonstrationJavadoc*



Le projet *DemonstrationJavadoc* vous est fourni afin de résumer chaque aspect étudié dans ce support. Il intègre les meilleures pratiques pour documenter vos composants. Nous vous conseillons de produire la *javadoc relative* et de vous en servir en exemple pour vos projets à venir.

Ce projet met plusieurs classes d'entités en jeu. Chacune ayant des liens avec les autres. Toutes les classes, méthodes, constructeurs, sont javadoc-commentés et produisent une documentation très riche pouvant servir de référence pour vos développements Java .

Il est impératif de *mettre en œuvre ces commentaires dans vos développements*.

Voici quelques copies, parmi de nombreuses autres, des pages générées. Vous apprécierez les nombreuses références croisées matérialisées par des liens entre les constituants du projet.

Documentation Dem x

file:///C:/Users/Michel/Documents/NetBeansProjects/DemonstrationJavadoc/dist/javadoc/index.html

Prev Package Next Package Frames No Frames

### Package com.michel.entites

**Class Summary**

Class	Description
Stagiaire	Stagiaire est une classe visant à démontrer les principaux tags de javadoc.

**Enum Summary**

Enum	Description
NiveauExperience	NiveauExperience est un type énuméré.

Overview **Package** Class Use Tree Deprecated Index Help

Prev Package Next Package Frames No Frames

## Method Summary

### Methods

Modifier and Type	Method and Description
void	<b>ajouterAmi</b> (Stagiaire ami) Permet d'ajouter un ami à la liste des amis du stagiaire
int	<b>getId</b> () Retourne l'identifiant unique du stagiaire
java.util.List<Stagiaire>	<b>getListeAmis</b> () Retourne la liste des amis du stagiaire
NiveauExperience	<b>getNiveau</b> () Retourne le niveau du stagiaire à un instant dans sa formation
java.lang.String	<b>getNom</b> ()
java.lang.String	<b>getPrenom</b> () Retourne le prénom du stagiaire
java.lang.String	<b>getPseudo</b> () Retourne le pseudo actuel du stagiaire
java.util.Vector<Stagiaire>	<b>getVecteurAmis</b> () <b>Deprecated.</b> Depuis Java SE 1.4 , doit être remplacée par <code>getListeAmis()</code>
void	<b>retirerAmi</b> (Stagiaire ex_ami) Permet de retire un stagiaire de la liste des amis
void	<b>setNiveau</b> (NiveauExperience niveau)

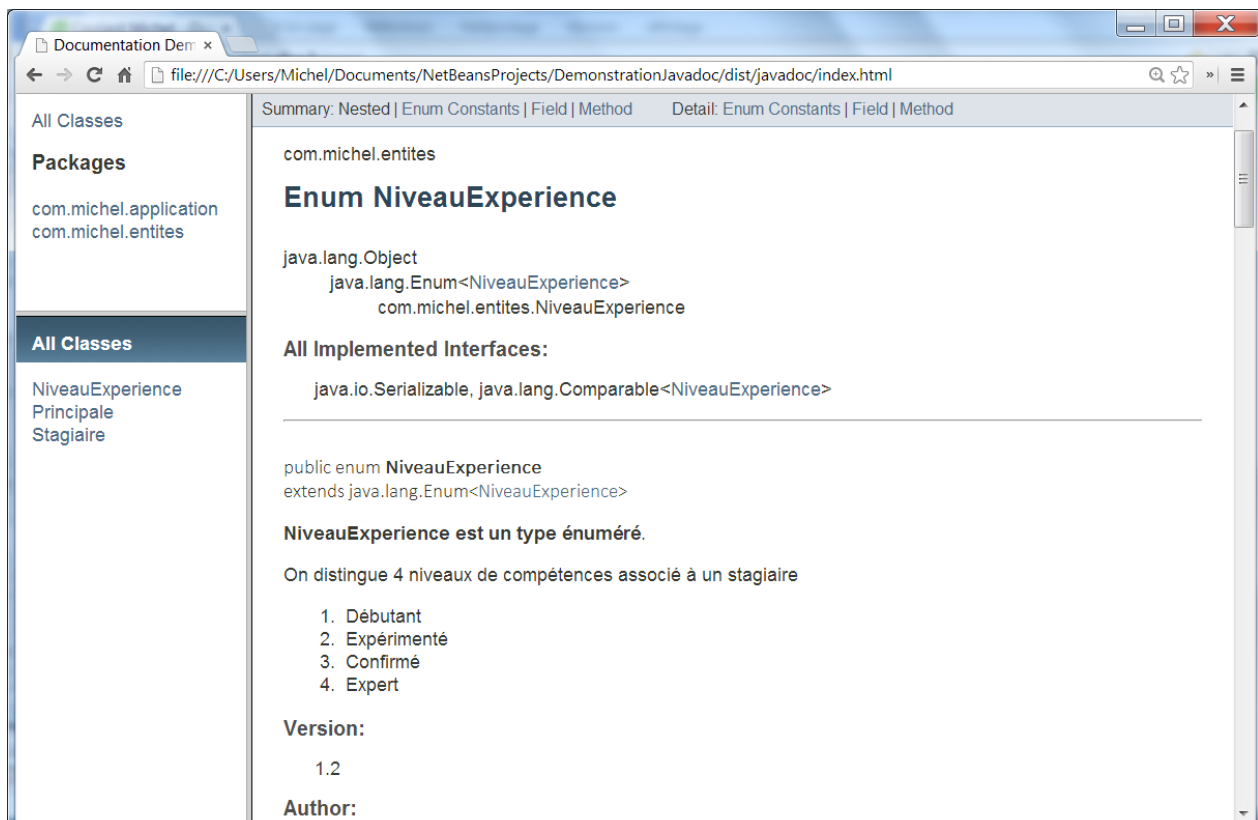


Figure 15 : Les fichiers générés par *javadoc* pour l'application de référence *DemonstrationJavadoc*.

## - 10 - RESUME

**javadoc** est un formidable outil fourni par défaut avec le **JDK**. Il permet de générer des fichiers de documentations à partir de votre code **Java**.

L'utilitaire **javadoc** analyse les fichiers sources pour rechercher les commentaires du type **/\*\*...\*/** dans les classes et les méthodes. Il produit des fichiers **HTML** au même format que la documentation **API**.

En fait, la documentation **API** n'est rien d'autre que la sortie de **javadoc** appliquée aux fichiers sources des classes **Java** standard.

En ajoutant des commentaires commençant par le délimiteur spécial **/\*\*** dans le code source, il est possible de produire facilement **une documentation d'aspect professionnel**.

Le problème majeur inhérent à la façon classique de documenter est lié à la divergence dans le temps de la génération du code et de celle de ses commentaires.

L'intérêt de l'approche **javadoc** est qu'elle permet de produire code et documentation **au même endroit et au même moment**.

Dès lors que les commentaires destinés à la documentation se trouvent dans le même fichier que le code source, il devient simple de mettre à jour les deux simultanément puis de relancer **javadoc**.

L'utilisation de l'outil **javadoc** est très simple. Il vous demande de lui spécifier un ensemble de fichiers **Java** à documenter. A partir de là, le reste se fait tout seul, pourvu que vous ayez dispersé un peu d'informations dans les fichiers de code.

Un certain nombre de fichiers sont générés et permettent la navigation parmi toutes les classes documentées. Le fichier de départ se nomme alors *index.html*.

La documentation générée par **javadoc** contient tout ce qui est nécessaire aux utilisateurs de vos classes.

Normalement, vos classes font partie d'un package. N'oubliez pas qu'en dehors d'un package, seuls les champs **public** sont accessibles. En conséquence, **javadoc** ne présentera que les champs ( méthodes ou attributs ) publics de vos classes.

► **Chef de projet (responsable du produit de formation)**

PERRACHON Chantal, DIIP Neuilly-sur-Marne

► **Ont participé à la conception**

COULARD Michel, CFPA Evry Ris-Orangis

► **Réalisation technique**

COULARD Michel, CFPA Evry Ris-Orangis

► **Crédit photographique/illustration**

Sans objet

► **Reproduction interdite / Edition 2014**

**AFPA Mai 2014**

**Association nationale pour la Formation Professionnelle des Adultes**

13 place du Général de Gaulle – 93108 Montreuil Cedex

[www.afpa.fr](http://www.afpa.fr)