Ing39: Javadoc (avec Eclipse)

M. V. Aponte
G. Levieux
S. Rosmorduc

Documentation en Javadoc

- but : fournir une documentation toujours à jour des méthodes d'une classe

- pratique courante dans la plupart des langages de programmation: documenter une procédure ou une fonction à l'aide de commentaires

- en pratique: formalisme permettant d'extraire *automatiquement* une documentation à partir des sources java

Programmation par contrat

- Idée importante: un sous-programme passe un « contrat » avec son utilisateur:
  - « si tu me passe tel ou tel argument, alors je ferai/je renverrai tel ou tel résultat »
- On y spécifie :
  - ce qui doit être vrai au moment de l'appel
  - ce qui est vérifié après l'appel
- ces informations doivent figurer dans la documentation du sous-programme.

Programmation par contrat

- pré-conditions: conditions qui doivent être remplies pour appeler le sous programme
  - généralement contraintes sur les arguments
    - si elles ne sont pas remplies, normalement: levée d'exception
- post-condition: conditions qui doivent être remplies après l'appel du sous programme. Elles précisent **ce que fait** le sous-programme
- Un contrat est une documentation précise d'un sous-programme!

Exemple

- Le contrat d'une méthode:
  - public static double moyenne(double [] tab)
- pré-condition: tab n'est ni null, ni vide (tab.length > 0)
- post-condition: la valeur retournée est la moyenne des valeurs comprises dans tab
- Attention: si la pré-condition n'est pas respectée, le contrat ne dit pas ce que fait la méthode. Son comportement reste non spécifié dans ce cas.

Javadoc en pratique

- La javadoc se présente comme une série de commentaires insérées dans le code.

- Un commentaire javadoc a le format

```
/**
 * texte
 */
```

  - placé avant une classe : documente la classe toute entière

  - placé avant une méthode : documente la méthode

- Via la ligne de commandes/IDE on génère du texte HTML correspondant à chaque commentaire javadoc.

Un exemple concret

```java
/**
 * Un ensemble de méthodes utilitaires pour travailler sur des tableaux.
 */
public class TableauxUtils {

    /**
     * Calcule la moyenne des éléments d'un tableau.
     * <p> Pré-condition : le tableau ne doit être ni null,
     * ni vide (tab.length &gt; 0) </p>
     * @param tab un tableau non vide
     * @return la moyenne des valeurs du tableau
     * @throws IllegalArgumentException si tab est null ou de taille 0
     */
    public static double moyenne(double tab[]) {
```

HTML généré

Problems Javadoc Declaration Search Console Coverage

double cours2_javadoc.TableauxUtils.moyenne(double[] tab)

Calcule la moyenne des éléments d'un tableau.

Pré-condition : le tableau ne doit être ni null, ni vide (tab.length > 0)

Parameters:
tab un tableau non vide
Returns:
la moyenne des valeurs du tableau
Throws:
IllegalArgumentException - si tab est null ou de taille 0

Circonscription | Courrier.java | Lettre.java | Colis.java | Calcul.java | Exo12_1.java | Exo12_2.java | TableauxUtils.j

```java
package cours2_javadoc;
/**
 * Un ensemble de méthodes utilitaires pour travailler sur des tableaux.
 */
public class TableauxUtils {

        /**
         * Calcule la moyenne des éléments d'un tableau.
         * <p> Pré-condition : le tableau ne doit être ni null,
         * ni vide (tab.length &gt; 0) </p>
         * @param tab un tableau non vide
         * @return la moyenne des valeurs du tableau
         * @throws IllegalArgumentException si tab est null ou de taille 0
         */
        public static double moyenne(double tab[]) {
            throw new UnsupportedOperationException ();
        }
}
```

Structure du commentaire javadoc

- 1ère ligne (terminée par un point): résumé de ce que fait la méthode,

- suite : texte documentant en détail la méthode. On peut utiliser des balises HTML

- ensuite: annotations pour décrire:
  - les paramètres de la méthode : @param
  - la valeur de retour @return
  - les exceptions @throws

```java
/**
 * Calcule la moyenne des éléments d'un tableau.
 * <p> Pré-condition : le tableau ne doit être ni null,
 * ni vide (tab.length &gt; 0) </p>
 * @param tab un tableau non vide
 * @return la moyenne des valeurs du tableau
 * @throws IllegalArgumentException si tab est null ou de taille 0
 */
public static double moyenne(double tab[]) {
```

Résumé (1 ligne terminée par « . »)

documentation

Description d'un paramètre

valeur de retour

Exceptions

liste des packages

Lire la javadoc

String (Java Platform SE 7)

docs.oracle.com/javase/7/docs/api/

Applications | 12.6 Machine Learni | informatique | Interactive Applicati | https://class.course | à lire | Autres favoris

Java™ Platform Standard Ed. 7

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

All Classes

Packages

java.applet

StreamFilter
StreamHandler
StreamPrintService
StreamPrintServiceFactory
StreamReaderDelegate
StreamResult
StreamSource
StreamTokenizer
StrictMath
String

classes

StringHolder
StringIndexOutOfBoundsExcep
StringMonitor
StringMonitorMBean
StringNameHelper
StringReader
StringRefAddr

java.lang

Class String

java.lang.Object
  java.lang.String

Documentation d'une classe

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

Lire la javadoc

Overview Package **Class** Use Tree Deprecated Index Help — Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, ar
instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.
Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

Position de cette classe dans la hiérarchie

Introduction (souvent assez détaillée)

Methods

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(CharSequence cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(StringBuffer sb)

index des méthodes et des champs

détail sur une méthode

indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring.

The returned index is the smallest value *k* for which:

```
this.startsWith(str, k)
```

If no such value of *k* exists, then `-1` is returned.

Parameters:

`str` - the substring to search for.

Returns:

the index of the first occurrence of the specified substring, or `-1` if there is no such occurrence.

Que fait-elle ?

Spécification

Cas particuliers

Paramètres

Valeur de retour, et cas particulier

Autre exemple

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.

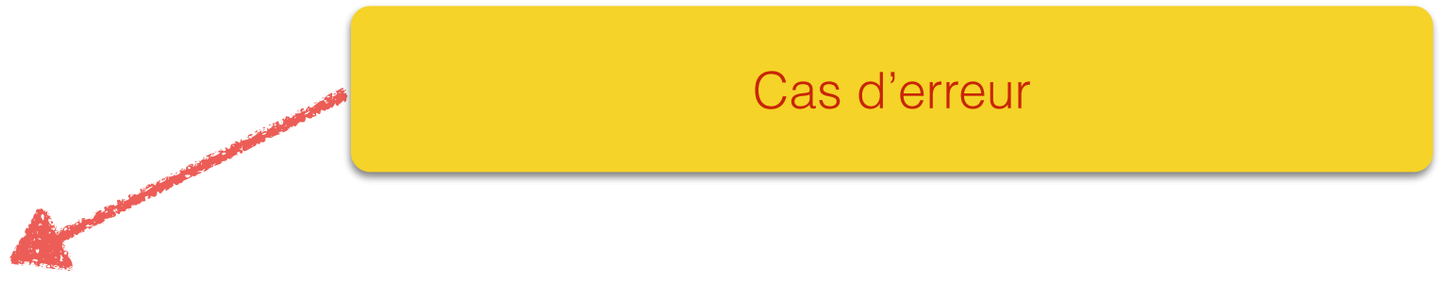
`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.



Retour sur l'écriture de javadoc: Quelques balises supplémentaires

- **@see** *NomClasse#methode(arguments)*

  renvoi à la documentation d'une autre méthode ou classe.
  Exemple:

  @see MiseEnPage#justifierGauche(String, int)

- **@author** *NOM*

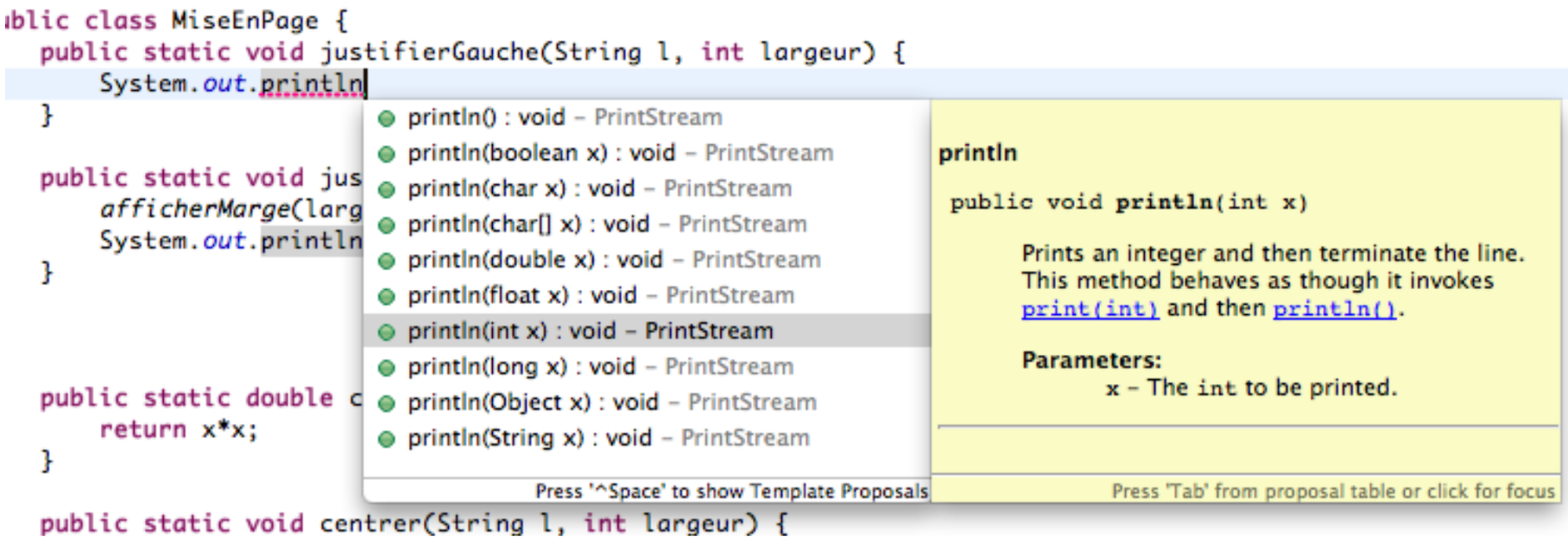
  - nom du créateur de la classe ou de la méthode

- {**@link** *NomClasse#methode(arguments)*} comme @see, mais @see donne un lien après la documentation, alors que @link se place dans le corps de la documentation.

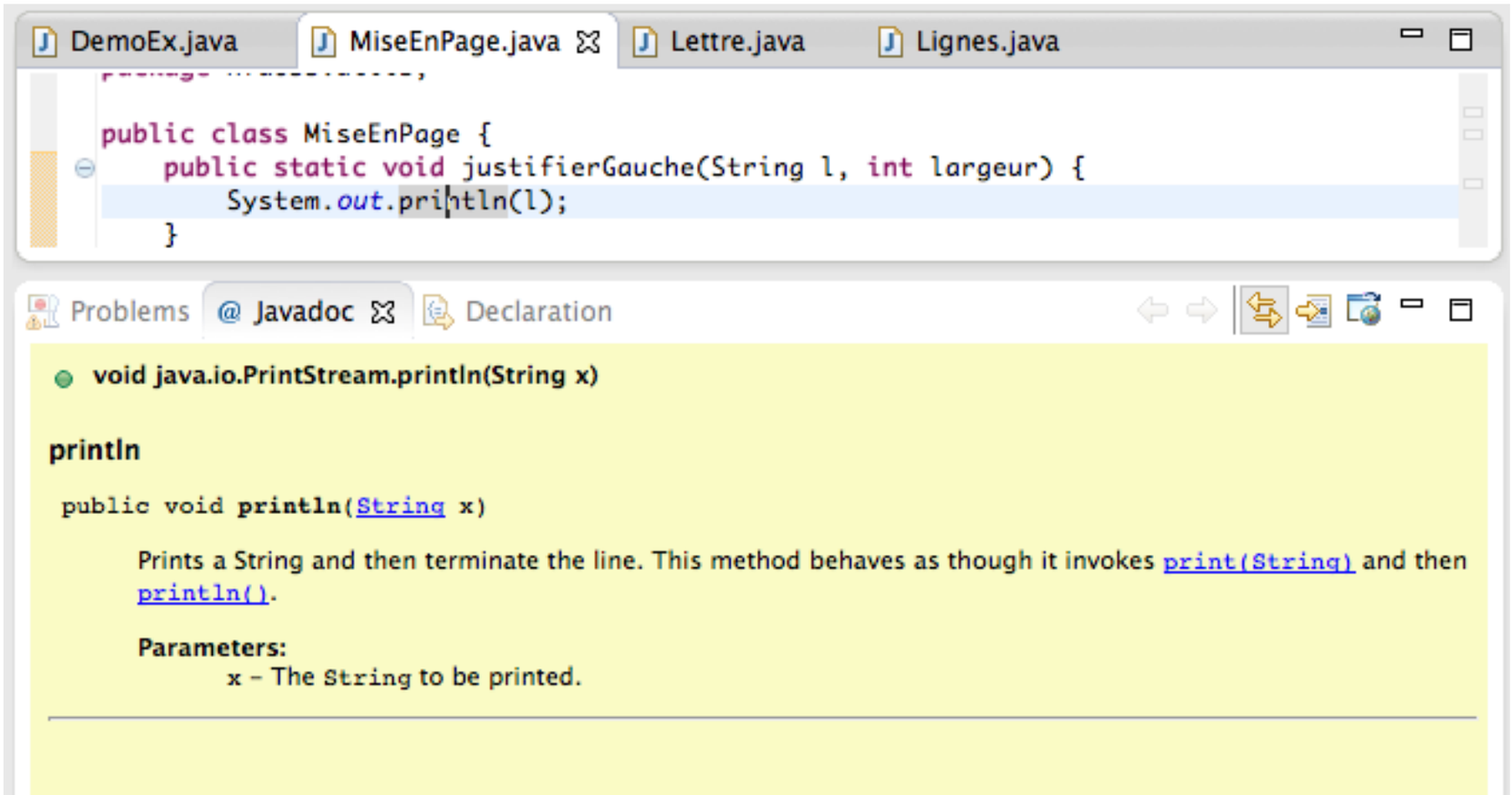
Intégration de javadoc dans eclipse

Aide lors de la complétion

On commence à écrire un nom de méthode possédant une documentation HTML. Eclipse propose des suggestions et affiche leur documentation.



Onglet javadoc



Aide à l'écriture

Juste avant l'entête d'une méthode….

```
    /**
    public static void justifierDroit(String l, int largeur) {
        afficherMarge(largeur- l.length());
        System.out.println(l);
    }
```

…on tape « entrée » et on obtient:

```
    /**
     *
     * @param l
     * @param largeur
     */
    public static void justifierDroit(String l, int largeur) {
        afficherMarge(largeur- l.length());
        System.out.println(l);
    }
```