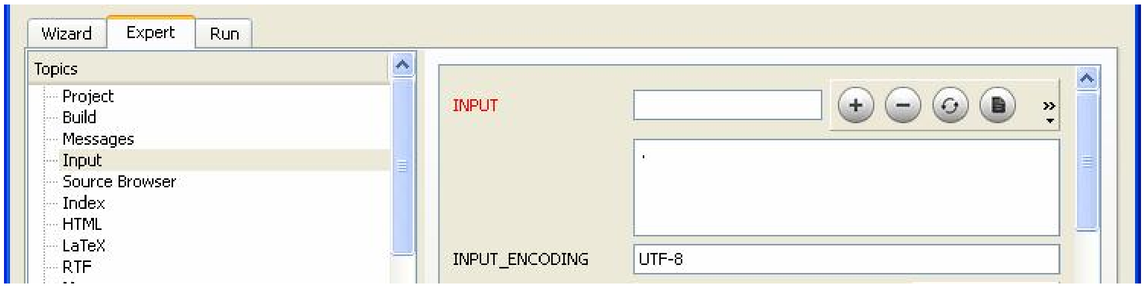
Bugs report :

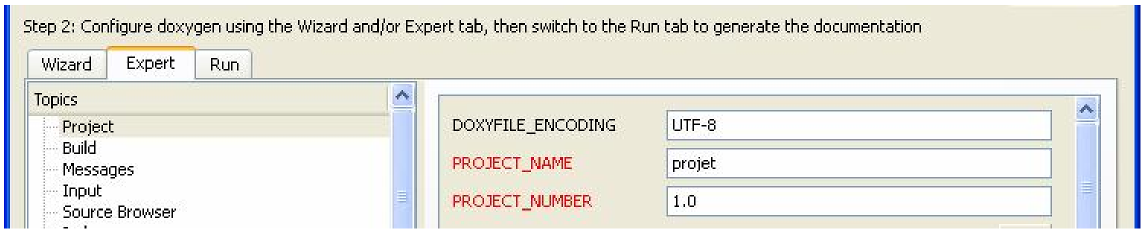
BUG 1:



INPUT .

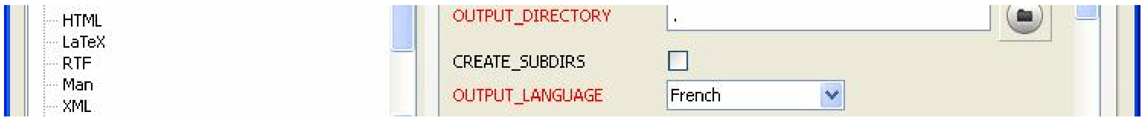
INPUT_ENCODING UTF-8

Input encoding on UTF-8


Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

DOXYFILE_ENCODING UTF-8
PROJECT_NAME projet
PROJECT_NUMBER 1.0

Doxywizard encoding on UTF-8



OUTPUT_DIRECTORY .
CREATE_SUBDIRS ☐
OUTPUT_LANGUAGE French

Ouput language on French

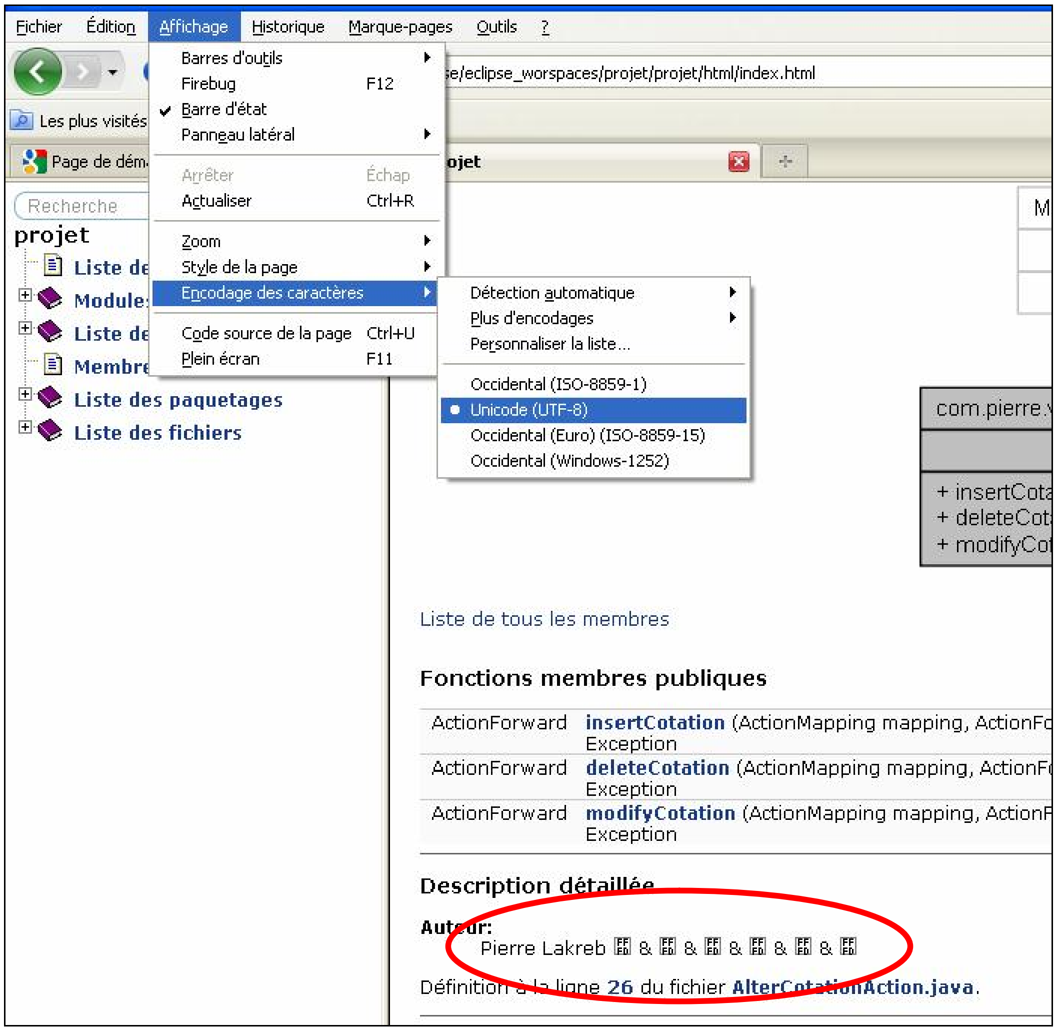
DoxyFile result:

```
DOXYFILE_ENCODING       = UTF-8
OUTPUT_LANGUAGE         = French
INPUT_ENCODING          = UTF-8
```

The code:

```
/**
 *
 * @author Pierre Lakreb
 *  è & è & à & ù & ê & î
 */
```

<u>But the HTML page result:</u>



Liste de tous les membres

Fonctions membres publiques

ActionForward	**insertCotation** (ActionMapping mapping, ActionFo Exception
ActionForward	**deleteCotation** (ActionMapping mapping, ActionF Exception
ActionForward	**modifyCotation** (ActionMapping mapping, ActionF Exception

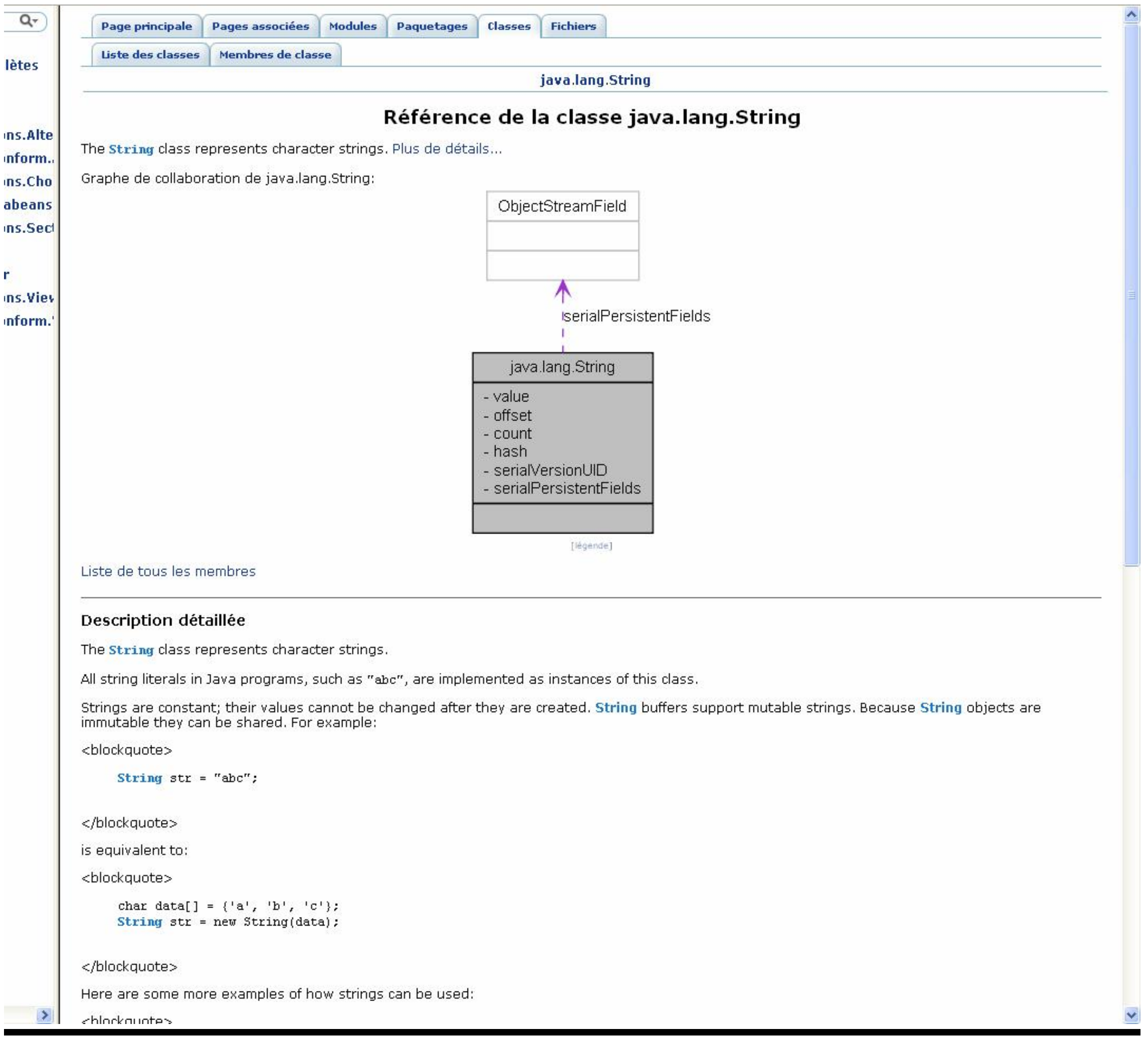
Description détaillée

Auteur:
Pierre Lakreb □ & □ & □ & □ & □ & □

Définition à la ligne **26** du fichier **AlterCotationAction.java**.

<u>BUG 2:</u>

Here is the result html page of the java.lang.String class (two screenshots).



Référence de la classe java.lang.String

The String class represents character strings. Plus de détails...

Graphe de collaboration de java.lang.String:

Liste de tous les membres

Description détaillée

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:

<blockquote>

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

</blockquote>

is equivalent to:

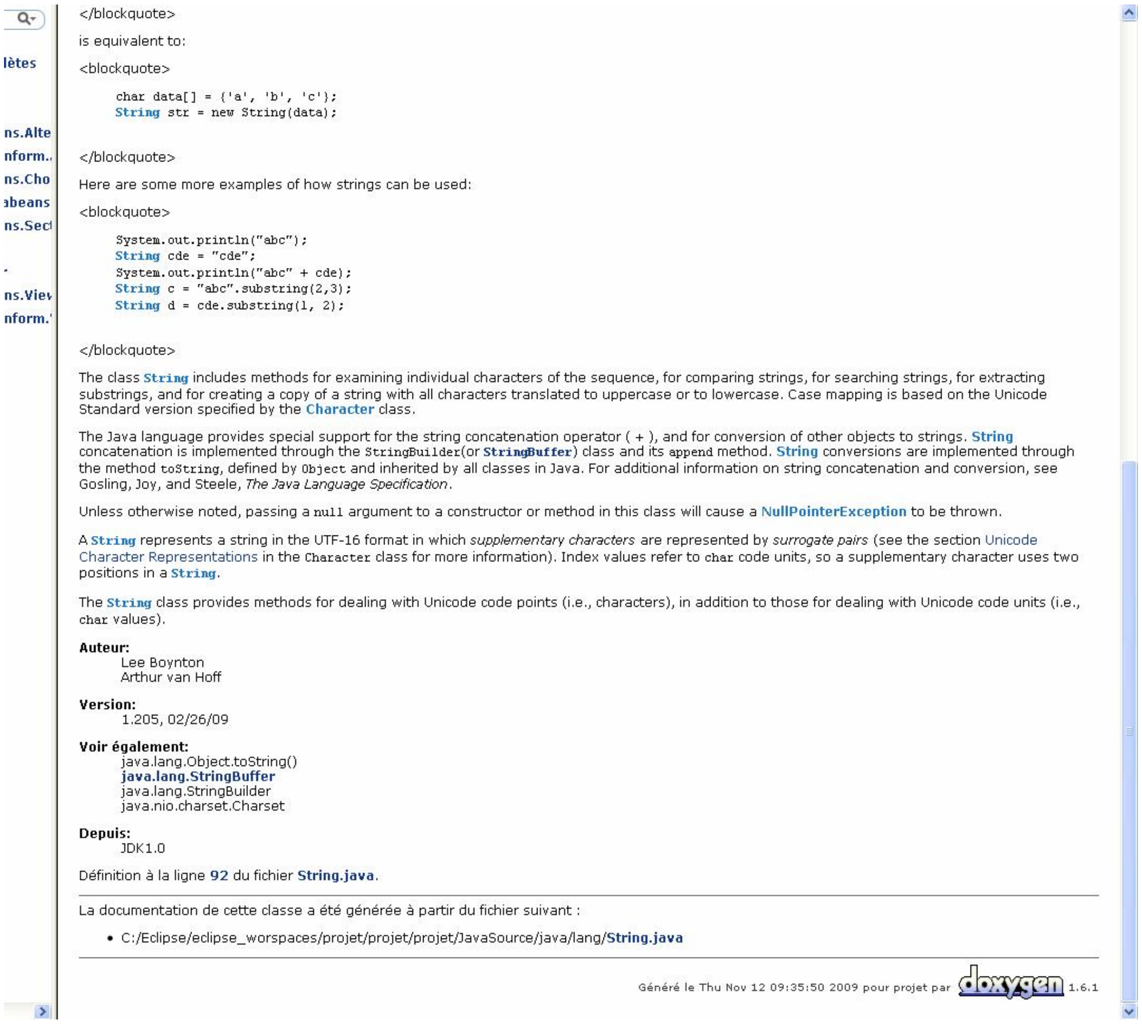
<blockquote>

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

</blockquote>

Here are some more examples of how strings can be used:

<blockquote>



</blockquote>

is equivalent to:

<blockquote>

```
     char data[] = {'a', 'b', 'c'};
     String str = new String(data);
```

</blockquote>

Here are some more examples of how strings can be used:

<blockquote>

```
     System.out.println("abc");
     String cde = "cde";
     System.out.println("abc" + cde);
     String c = "abc".substring(2,3);
     String d = cde.substring(1, 2);
```

</blockquote>

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the `StringBuilder`(or `StringBuffer`) class and its `append` method. String conversions are implemented through the method `toString`, defined by `Object` and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

Unless otherwise noted, passing a `null` argument to a constructor or method in this class will cause a NullPointerException to be thrown.

A `String` represents a string in the UTF-16 format in which *supplementary characters* are represented by *surrogate pairs* (see the section Unicode Character Representations in the `Character` class for more information). Index values refer to `char` code units, so a supplementary character uses two positions in a `String`.

The `String` class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., `char` values).

Auteur:
Lee Boynton
Arthur van Hoff

Version:
1.205, 02/26/09

Voir également:
java.lang.Object.toString()
java.lang.StringBuffer
java.lang.StringBuilder
java.nio.charset.Charset

Depuis:
JDK1.0

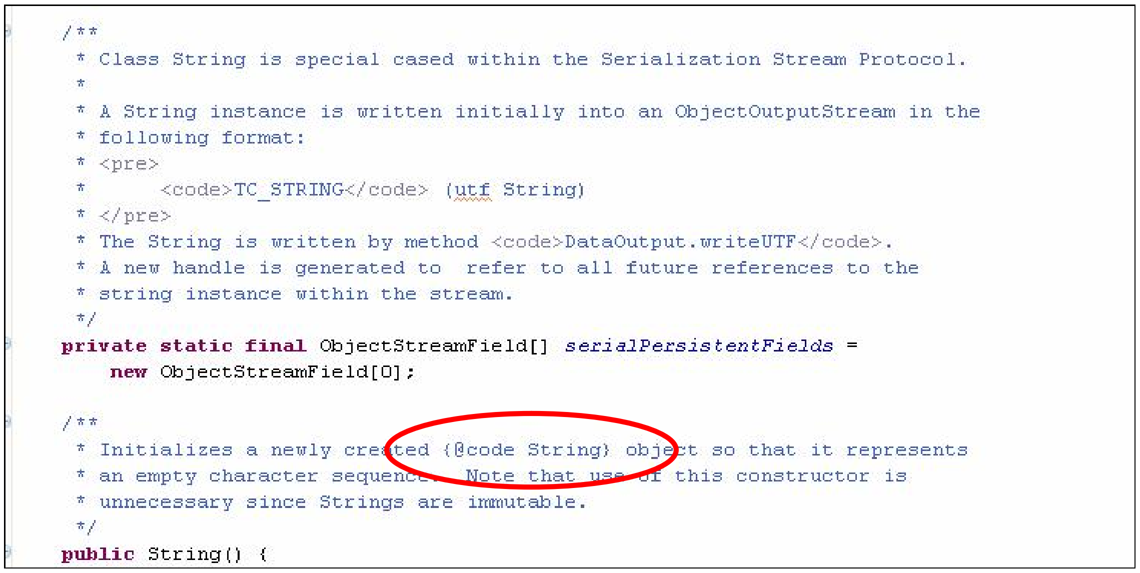
Définition à la ligne **92** du fichier **String.java**.

La documentation de cette classe a été générée à partir du fichier suivant :

- C:/Eclipse/eclipse_worspaces/projet/projet/projet/JavaSource/java/lang/**String.java**

Généré le Thu Nov 12 09:35:50 2009 pour projet par doxygen 1.6.1

As you can see the doc is really not complete. Here is the beginning of the class code:



```
    /**
     * Class String is special cased within the Serialization Stream Protocol.
     *
     * A String instance is written initially into an ObjectOutputStream in the
     * following format:
     * <pre>
     *      <code>TC_STRING</code> (utf String)
     * </pre>
     * The String is written by method <code>DataOutput.writeUTF</code>.
     * A new handle is generated to  refer to all future references to the
     * string instance within the stream.
     */
    private static final ObjectStreamField[] serialPersistentFields =
        new ObjectStreamField[0];

    /**
     * Initializes a newly created {@code String} object so that it represents
     * an empty character sequence.  Note that use of this constructor is
     * unnecessary since Strings are immutable.
     */
    public String() {
```

Of course, the javadoc annotation {@code} is in conflict with \code and \encode doxygen annotations. The generator is waiting for the close tag!! So the rest of the class is ignored.

More over, I don't know if you have seen the <blockquote> tag, it is not an error on my part (as I didn't wrote the String class ^^). This html tag is simply translated as text format in the html page. That's not all, all the javadoc annotation like {@annotation} is unusable.

I don't know if it is an oversight or an error but it is damageable for instance if you have old class utilities you want to document. It's not convenient for java developers, though it is a very powerful tool!

Best regards.