Les objets du point de vue de l'utilisateur

Serge Rosmorduc

CNAM-Paris

9 septembre 2015

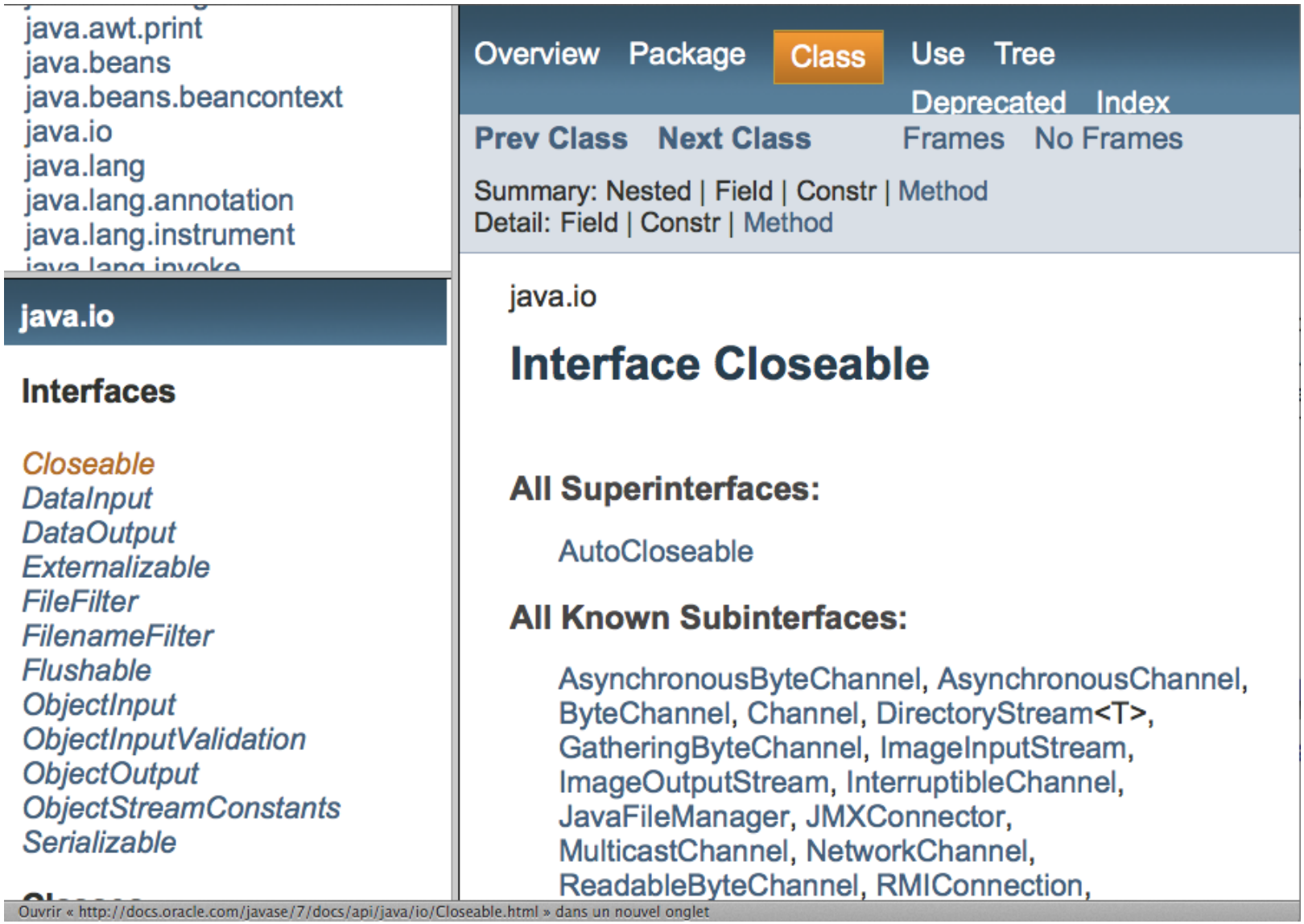
Introduction

- Un programmeur conçoit et implémente des classes ;
- mais surtout, il utilise énormément de classes écrites par d'autres ;
- 4240 classes dans l'API standard de java ;
- très (très) nombreuses bibliothèques disponibles
- dans ce cours, nous allons apprendre à utiliser un objet.

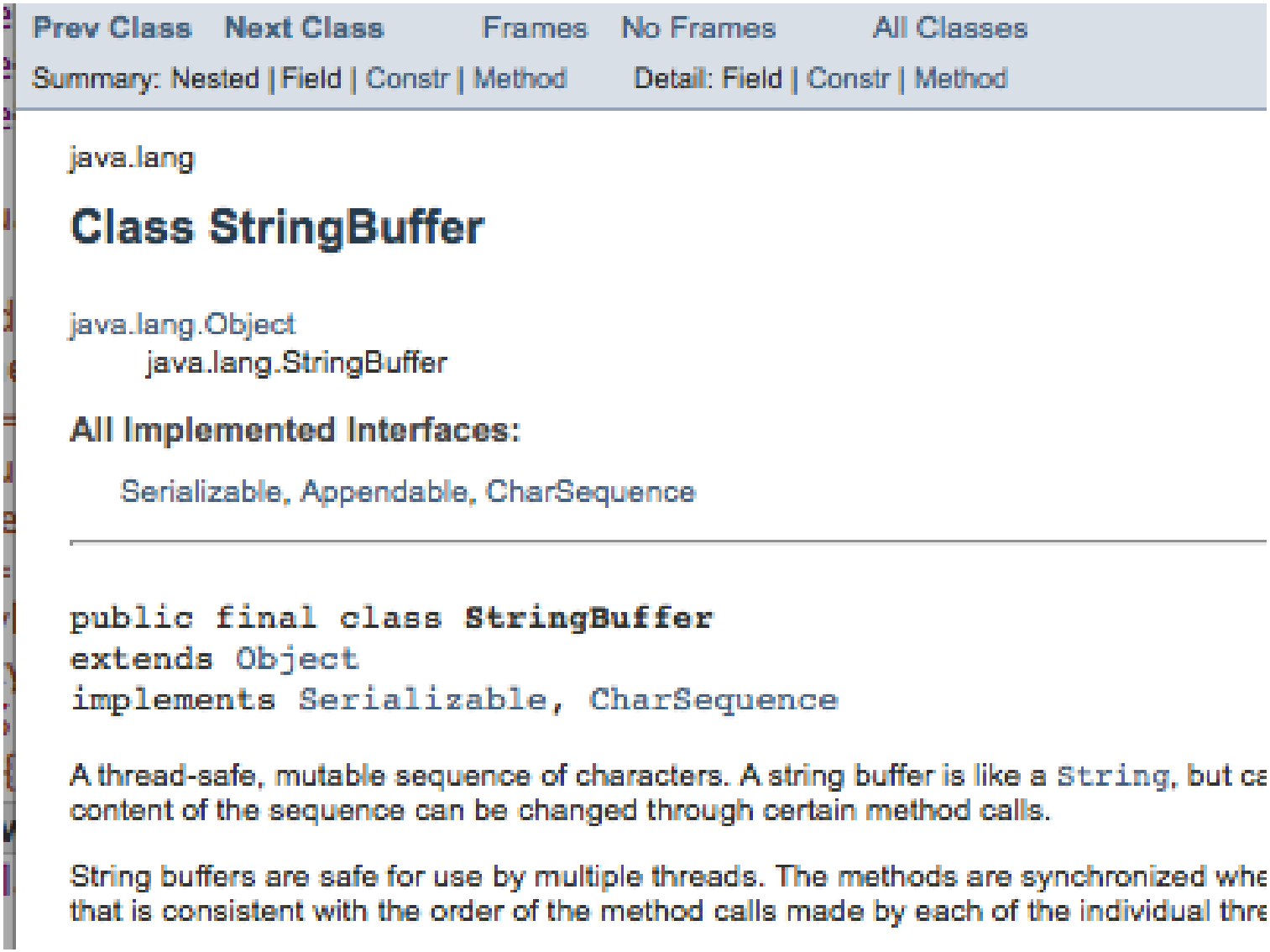
Programmation par contrat et interface

- La documentation d'une classe explique (dans l'idéal) :
- ce que *représente* un objet de la classe ;
- comment *créer* un objet de la classe :
- la liste des méthodes qui permettent de *manipuler* un objet d'une classe et définit son comportement.

La javadoc


java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.invoke
java.io
Interfaces
Closeable
DataInput
DataOutput
Externalizable
FileFilter
FilenameFilter
Flushable
ObjectInput
ObjectInputValidation
ObjectOutput
ObjectStreamConstants
Serializable
Overview Package Class Use Tree
Deprecated Index
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method
java.io
Interface Closeable
All Superinterfaces:
AutoCloseable
All Known Subinterfaces:
AsynchronousByteChannel, AsynchronousChannel,
ByteChannel, Channel, DirectoryStream<T>,
GatheringByteChannel, ImageInputStream,
ImageOutputStream, InterruptibleChannel,
JavaFileManager, JMXConnector,
MulticastChannel, NetworkChannel,
ReadableByteChannel, RMIConnection,
Ouvrir « http://docs.oracle.com/javase/7/docs/api/java/io/Closeable.html » dans un nouvel onglet

Structure de la javadoc



Prev Class Next Class Frames No Frames All Classes

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

java.lang

Class StringBuffer

java.lang.Object
    java.lang.StringBuffer

All Implemented Interfaces:

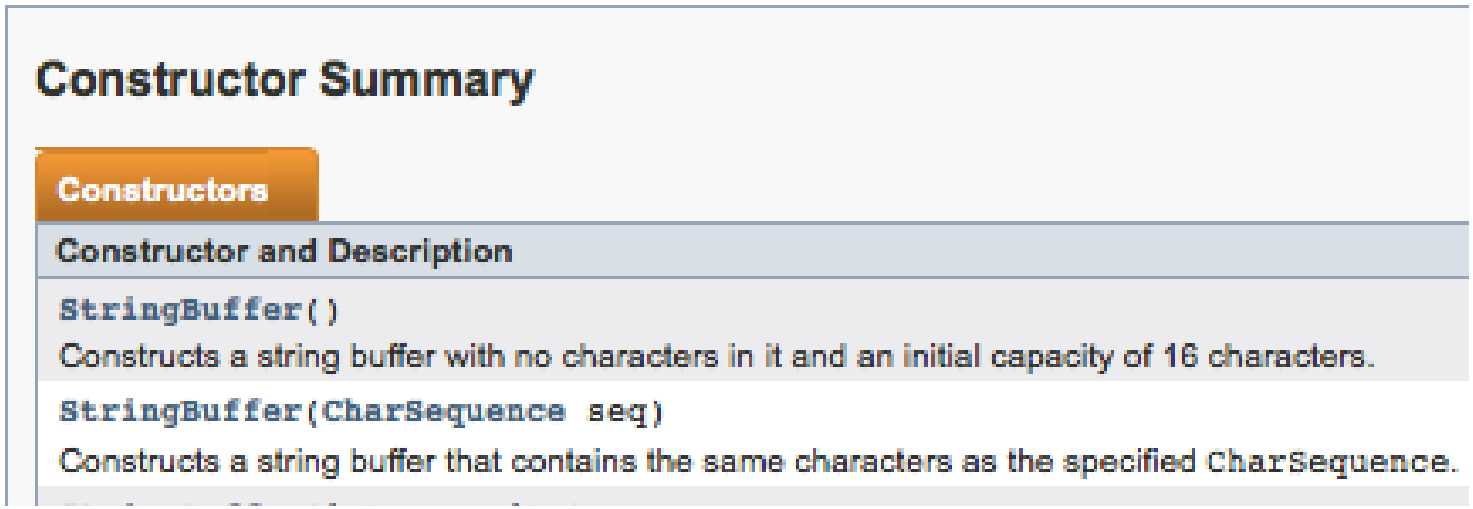
Serializable, Appendable, CharSequence

```
public final class StringBuffer
extends Object
implements Serializable, CharSequence
```

A thread-safe, mutable sequence of characters. A string buffer is like a `String`, but ca
content of the sequence can be changed through certain method calls.

String buffers are safe for use by multiple threads. The methods are synchronized whe
that is consistent with the order of the method calls made by each of the individual thre

Structure de la javadoc

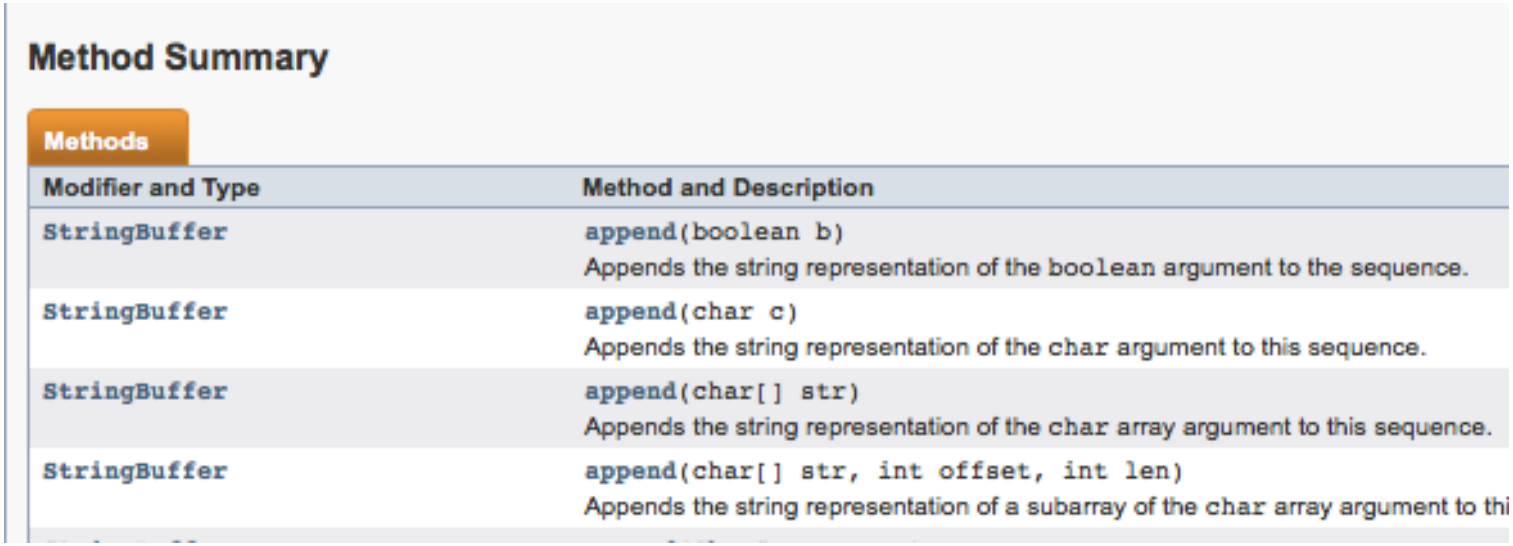


Constructor Summary

Constructors

Constructor and Description
`StringBuffer()` Constructs a string buffer with no characters in it and an initial capacity of 16 characters.
`StringBuffer(CharSequence seq)` Constructs a string buffer that contains the same characters as the specified `CharSequence`.

Structure de la javadoc



Method Summary

Methods

Modifier and Type	Method and Description
`StringBuffer`	`append(boolean b)` Appends the string representation of the `boolean` argument to the sequence.
`StringBuffer`	`append(char c)` Appends the string representation of the `char` argument to this sequence.
`StringBuffer`	`append(char[] str)` Appends the string representation of the `char` array argument to this sequence.
`StringBuffer`	`append(char[] str, int offset, int len)` Appends the string representation of a subarray of the `char` array argument to thi

Structure de la javadoc

charAt

`public char charAt(int index)`

Returns the `char` value in this sequence at the specified index. The first `char` value is at index `0`, the next at index `1`, and so on, as in array indexing.

The index argument must be greater than or equal to `0`, and less than the length of this sequence.

If the `char` value specified by the index is a surrogate, the surrogate value is returned.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the desired `char` value.

Returns:

the `char` value at the specified index.

Throws:

`IndexOutOfBoundsException` - if `index` is negative or greater than or equal to `length()`.

See Also:

`length()`

Documentation d'une méthode

- **Signature de la méthode** : nom de la méthode, type de retour, type des paramètres ;
- courte description de ce que fait la méthode ;
- spécification de la méthode ;
- description des paramètres ;
- description de la valeur retournée ;
- description des exceptions levées par la méthode.

Description d'une classe

La javadoc explique :

- quels sont les *invariants* des objets : quelles propriétés sont vraies sur les objets une fois créés.
- comment *créer un objet* ;
- comment le modifier ;
- comment connaître son état.
- comment utiliser l'objet.

Création d'objet

Plusieurs possibilités :

- créer l'objet directement avec new et un constructeur ;
- faire créer l'objet par une méthode statique de la classe de l'objet ;
- faire créer l'objet par un autre objet.

Création d'objet

constructeur

Exemple :

```
URL url = new URL("http://www.cnam.fr/");
```

Création d'objet

Constructeur nommé

Méthode statique qui sert à créer ou à récupérer une instance d'un objet.
C'est une variante du pattern *factory method.*
Exemple :

```
Calendar rightNow = Calendar.getInstance();
```

ou

```
Pattern p = Pattern.compile("a*b");
```

Création d'objet

Un objet B est construit par un objet A

L'objet A fonctionne alors comme une fabrique (factory).

```
URL url = new URL("http://www.cnam.fr/");
URLConnection connexion = url.openConnection();
```

Cache pas mal de complexité : ici, l'implémentation de l'objet `URLConnection` est différent selon le protocole choisi (http, https, file...)

Helper class (ou classes utilitaires)

- classes comportant des méthodes statiques, travaillant souvent sur des instances d'une autre classe ;
- normalement rares dans un programme bien conçu (ça n'est pas de l'objet) ;
- assez utilisées en java, pour des fonctionalités "transverses" :
  - classe `Arrays` : fournit des fonctionalités sur les tableaux ;
  - classe `Collections`, travaille sur les collections en général (voir plus tard) ;
  - classes `Integer, Double, Character...` : ont des instances, mais fournissent aussi des méthodes pour manipuler les types de base.
- à partir de java 8, devraient devenir moins fréquentes (pour les nouvelles classes).

Classes associées au types primitifs

- int, double, char, boolean ne sont pas des classes ;
- où mettre les méthodes qui travaillent sur ces types (exemple : méthode qui dit si un char est une majuscule) ?
- parfois, on a vraiment besoin d'un objet. Par exemple, dans une `ArrayList`, les éléments sont des objets.
- comment faire quand on a vraiment besoin d'un objet, mais que les données sont d'un type primitif ?

Classes associées au types primitifs

- Solution : on asssocie à chaque type primitif une classe.
  - char $\rightarrow$ Character ; double $\rightarrow$ Double ; int $\rightarrow$ Integer ; boolean $\rightarrow$ Boolean ;
  - ces classes comportent de très nombreuses méthodes utilitaires ;
  - On peut utiliser ces classes pour $\ll$ encapsuler $\gg$ des données primitives ; les enrober dans un objet.

    ```
    Character c= new Character('a');
    ```

  - java sait – plus ou moins – le faire automatiquement. On peut écrire : Character c= 'a' ; // autoboxing !

et

```
Integer id= new Integer(3);
int i= id;
```

Immutabilité

Un objet est immuable s'il n'est pas possible de le modifier après sa création.
Intérêt :

- sémantique simple : l'objet représente une *valeur* ;
- on peut utiliser l'objet sans risque. Il ne sera pas modifié sans qu'on en soit prévenu.

Intérêt de l'immutabilité

Exemple de problème :

```
public static boolean contientDoublons(String [] l) {
Arrays.sort(l);
for (int i= 0; i < l.length -1 ; i++) {
          if (l[i].equals(l[i+1])) return true;
}
return false;
}
```

- Un appel à `contientDoublon` *modifie* le tableau ;
- probablement pas ce qu'on veut ;
- solution 1 : copies défensives
  - avant l'appel ;
  - copie des arguments ;
  - on copie beaucoup. C'est cher.

Si on avait une liste non modifiable, ce serait plus simple.

Fluent API

Typiquement, création d'un « sous-langage » en utilisant l'astuce suivante : les méthodes de l'objet renvoient celui-ci, ce qui fait qu'on peut « chaîner » les appels.
Un exemple : la bibliothèque `jRTF`

```
rtf().section(
    p( "first␣paragraph" ),
    p( tab(),
       "␣second␣par␣",
       bold( "with␣something␣in␣bold" ),
       text( "␣and␣" ),
       italic( underline( "italic␣underline" ) )
     )
).out(out);
```

Cette manière de concevoir une classe est de plus en plus fréquente en java 8.

Exemple : Classe StringBuffer

Exemple : classe Date

Google est votre ami

- mais il faut faire un peu attention :-)
- pour des solutions ponctuelles :
  - forums informatiques : plus ou moins bons (risque de solutions bricolées/dépassées voire fausses) ;
  - stackoverflow est assez fiable (il attire des utilisateurs de bon niveau)

Les tutoriels d'oracle

Réflexions sur les classes Date, Calendar, et LocalDate