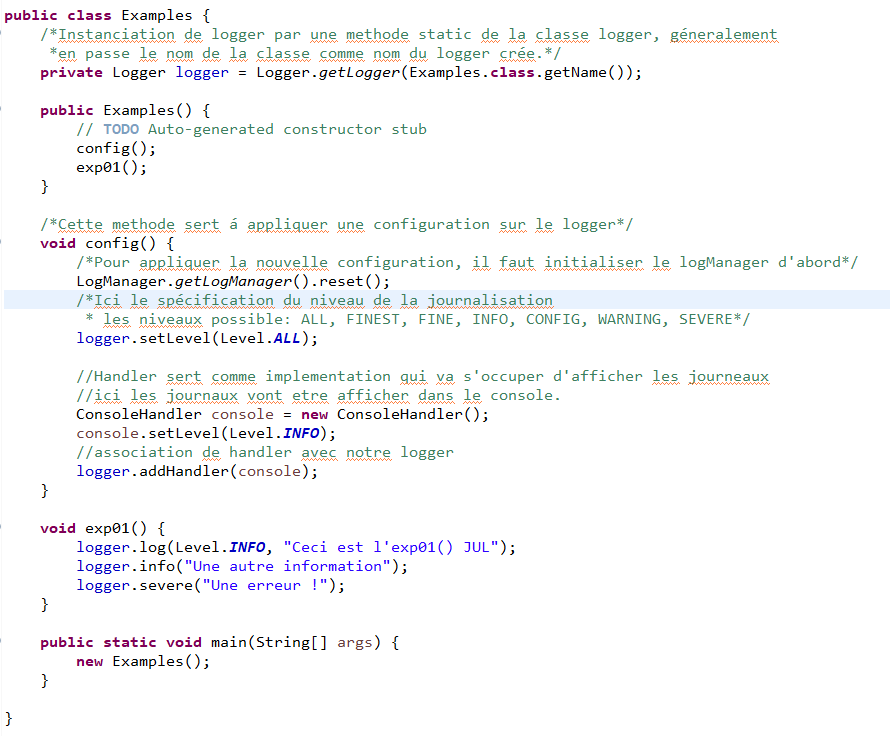
Journalisation:

**En utilisant JUL (Java Util Loggin)** : outil de journalisation qui est intégré en JDK, son configuration est faite dans un fichier nommé logging.propreties.

La journalisation se fait en général par l’ajoute d’une propriété Logger logger :

Ici un exemple de la classe Examples qui dépend d’un objet Logger du package java.util.loggin.Logger ;



**En utilisant LOG4J :**

Pour utiliser log4j qui est une bibliothèque extérieure il faut 2 fichiers .jar :

* Log4j-core-2.14.1.jar
* Log4j-api-2.14.1.jar

Le package de log4j est : org.apache.logging.log4j

La configuration de log4j se fait dans un fichier log4j2.xml qui se trouve au racine du package, il y’a les Appenders, qui indique ou les messages de journalisation vont être afficher (dans une console, ou bien enregistrer dans un fichier), et il y’a les Loggers où il y’a l’élément root qui a la configuration de base qui va s’appliquer sur tous les éléments du package, et si après on veut appliquer une spécifique configuration sur une classe on crée un logger qui porte le même nom de la classe.

<Console name = ‘’console’’ target = ‘’SYSTEM\_OUT’’ > </Console > 🡪Pour créer un appender de type console.

<File name = ‘’service-file’’ fileName = ‘’logs/service.log’’></File> 🡪 appender de type fichier.

Pour specifier le format de message de journalisation en utilise la balise PatternLayout :

<PatternLayout pattern = ‘’%5-p : %d{dd/MM/yyyy hh:mm:ss %C, %M, %t, %m%n} />

Quelque symbole utiliser dans le pattern:

* %p: level de Journalisation.
* %d{}: pour spécifier la date.
* %m : message de journal.
* %C : classe concerné par le journal.
* %M : méthode concernée par le journal.
* %t : thread qui a lancé le journal
* %n : retour á la ligne.

Avant tout logger il y’a Root qui est le logger père qui contient la configuration de base qui sera appliquer sur tous les logger qui vient après, ici additivity égal á true veut dire appliquer aussi la configuration de root, s’il est égal á false, il n’hérite pas la configuration de root, et pour spécifier l’appender pour un logger, il y’a la balise appederRef qui a l’attribut ref qui prend le nom d’appender qui est créé au préalable.

<Logger name = ‘’packageNameOuClasse’’ level = ‘’’’ additivity = ‘’true’’ >

<appenderRef ref = ‘’nomdAppender’’ />

</Logger>

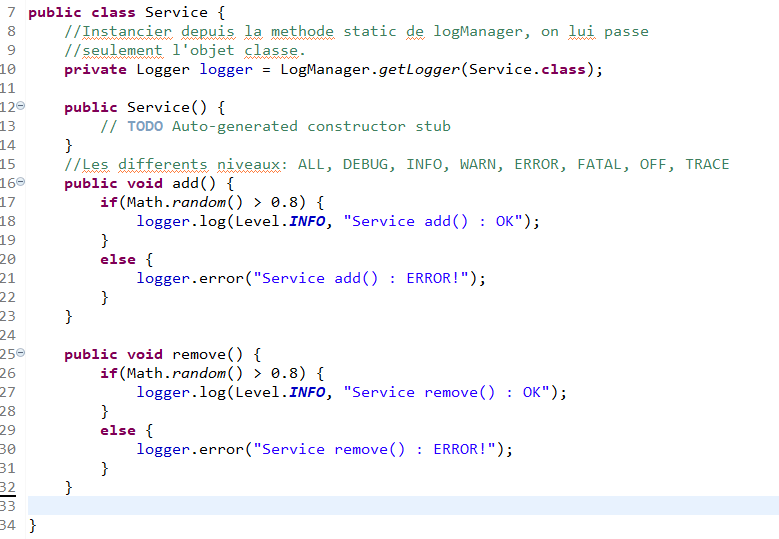
Exemple d’un fichier log4j2.xml:



Remarques :

* Lorsqu’on définit un autre logger en déçus de root, le root sera désactivé.
* Root a la portée package, donc tous logger dans le package sera affiché.
* Vous pouvez appliquer chaque logger sur chaque classe où il est déclaré un logger interne.
* Lorsque additivity = ‘’true’’ appeder de root aussi sera hérité le level qui sera afficher dans ce handler, c’est le level de logger et n’est pas de root.

Exemple d’une classe qui utilise la journalisation de log4j2 :



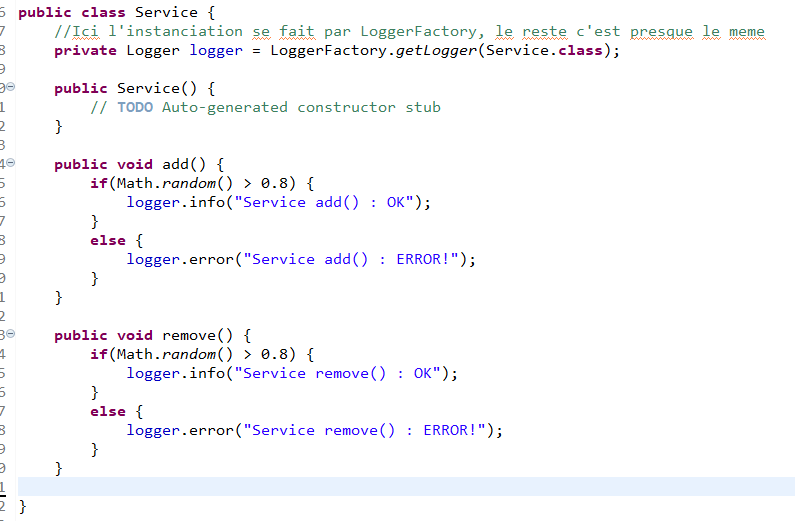
**En utilisant LOG4J comme implementation de SLF4J :**

Lorsqu’on utilise la spécification de SLF4, notre code sera inchangeable quoi que ce soit l’implémentation choisie (log4j ou logback), le seul changement dont on a besoin de faire, c’est de changer le fichier de configuration (log4j2.xml ou logback.xml).

Pour LOG4J avec SLF4J le fichier de configuration reste le même, seul le code qui va être écrit par la spécification SLFJ, et pour ça on a besoin de 4 fichiers :

* Log4j-core-2.14.1.jar
* Log4j-api-2.14.1.jar
* Log4j-sl4j-impl-2.14.jar 🡪 bridge
* Slf4j-api-1.7.32.jar

Le package du logger SLF4 c’est : **org.slf4j.Logger**



**En utilisant logBack comme implementation de SLF4J :**

Pour faire ça il suffit de changer le fichier de configuration par logback.xml, et ajouter les bibliothèques log Back au lieu des bibliothèques de slf5 suivant :

* + Logback-classic-1.2.6.jar
  + Logback-core-1.2.6.jar
  + Slf4j-api-1.7.32.jar

Exemple de logback.xml :

