**GESTION DES LOGS**

Table des matières

[Chapitre 1 DEFINITION 2](#_Toc105262062)

[Chapitre 2 Des recommandations lors de la mise en œuvre : 2](#_Toc105262063)

[Chapitre 3 Exemple de mise en œuvre & niveaux de gravité : 3](#_Toc105262064)

[Chapitre 4 4](#_Toc105262065)

[Titre 1 4](#_Toc105262066)

[Chapitre 5 4](#_Toc105262067)

[Titre 1 4](#_Toc105262068)

[Chapitre 6 4](#_Toc105262069)

[Titre 1 4](#_Toc105262070)

# 

# DEFINITION

La gestion des logs dans une application (un programme) c’est de pouvoir afficher des traces d’exécution de ses méthodes lors de la mise en fonction de ce programme)

Le logging est une activité technique utile et nécessaire dans une application pour :

* Déboguer : pratique lorsque la mise en oeuvre d'un débogueur n'est pas facile.
* Obtenir des traces d'exécution (démarrage/arrêt, informations, avertissements, erreurs d'exécution, ...
* Faciliter la recherche d'une source d'anomalie (stacktrace, ...)
* Comprendre ou vérifier le flux des traitements exécutés : traces des entrées/sorties dans les méthodes, affichage de la pile d'appels, ...

Une API de logging fait généralement intervenir trois composants principaux :

* Logger : invoqué pour émettre grâce au framework un message généralement avec un niveau de gravité associé
* Formatter : utilisé pour formater le contenu du message
* Appender : utilisé pour envoyer le message à une cible de stockage (console, fichier, base de données, email, ...)

***NB :*** Le logging est particulièrement important dans une application notamment côté serveur mais une utilisation à outrance ou une mauvaise utilisation de cette fonctionnalité peut dégrader les performances générales de l'application.

L'utilisation d'une API de Logging implique donc une surcharge de consommation de ressources (CPU, mémoire, ...) mais elle se justifie par l'apport des informations fournies en cas de problème sous réserve que ces informations aient été judicieusement choisies.

# Des recommandations lors de la mise en œuvre :

* Chaque message doit contenir la date/heure d'émission et la classe émettrice
* Ne jamais utiliser de System.out pour afficher des messages mais utiliser une API de Logging
* Ne jamais utiliser la méthode printStackTrace() de la classe Exception pour afficher des messages mais utiliser une API de Logging
* Eviter les messages émis trop fréquemment (par exemple dans une boucle avec un nombre important d'itérations ou dans une méthode fréquemment invoquée, ...)
* Utiliser le niveau de gravité en adéquation avec le message

Pour des traces d'exécution, il est pratique d'émettre un message en début d'une méthode qui affiche les paramètres en entrée et un message à la fin de la méthode avec la valeur de retour

# Exemple de mise en œuvre & niveaux de gravité :

## 1- Ecriture sur la console :

## 

1. On crée un projet maven,
2. On crée un fichier log4j.properties stocké dans le classpath de l'application : ce fichier contient la configuration de Log4j pour l'application.

log4j.rootLogger=**DEBUG**, **console**

log4j.**appender**.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=%d [%-5p] (%F:%M:%L) %m%n

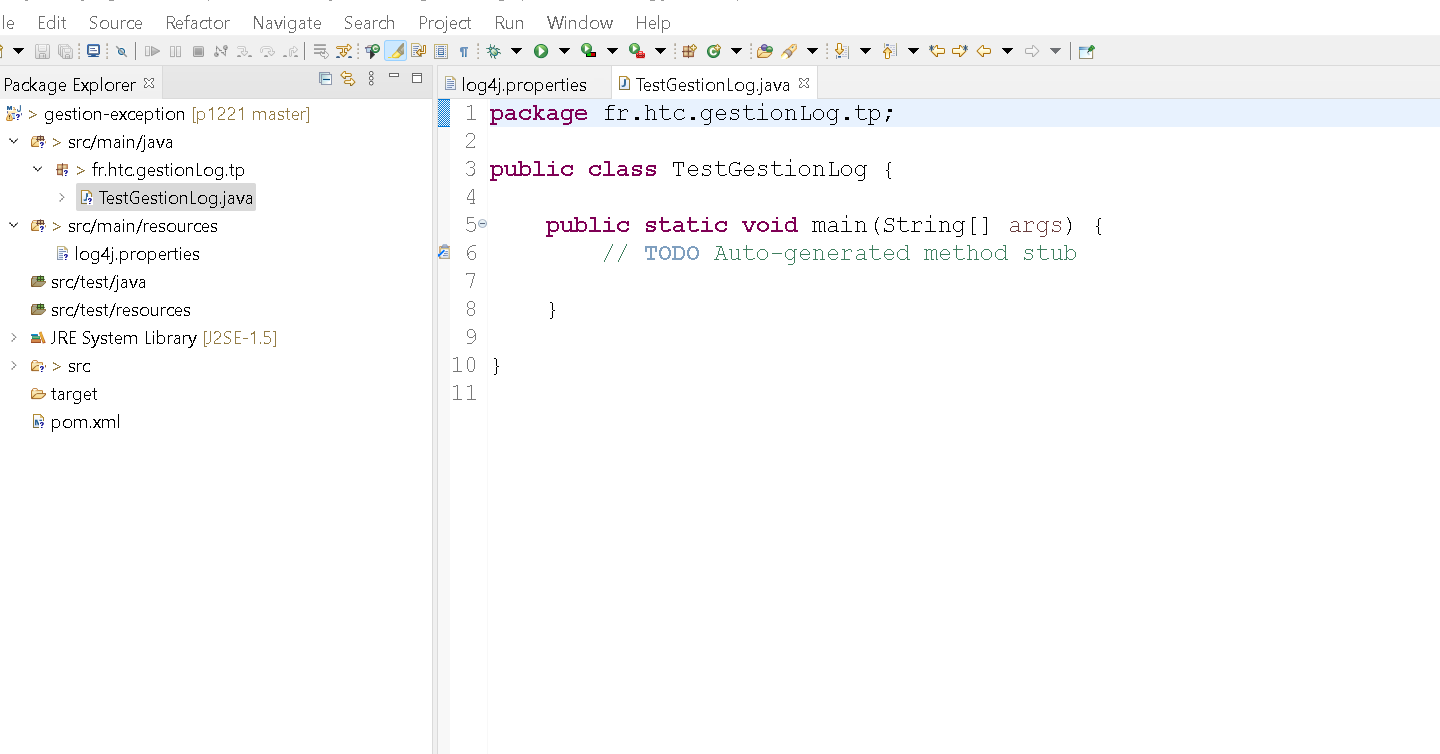
* Cette configuration définit le niveau de gravité **DEBUG** pour le logger racine et lui associe un logger nommé arbitrairement **console**. Par héritage, tous les loggers de l'application vont hériter de cette configuration.
* **L'appender** nommé console est de type ConsoleAppender : il envoie les messages sur la console standard.
* Un layout personnalisé est associé à l'appender nommé console pour formater les messages. Chaque séquence commençant par le caractère % sera remplacée dynamiquement par sa valeur correspondante. Par exemple : %d correspond à la date/heure, %p au niveau de gravité, %m le message, %n une nouvelle ligne, ...
* Pour mettre en oeuvre l'API dans le code source, il faut tout d'abord obtenir une instance du **logger** à utiliser en utilisant la méthode getLogger() de la classe Logger.
* Chaque message est émis en utilisant la méthode correspondant au niveau de gravité choisi de la classe Logger.
* Les niveaux de gravité, on peut les voir dans le tableau suivant  du moins grave au plus grave:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Affichage sur la console | | | | | | |
| Commandes lancées |  | debug | Info | Warn | Error | Trace | fatal |
| Info |  |  |  |  |  |  |
| Warn |  |  |  |  |  |  |
| Error |  |  |  |  |  |  |
| Trace |  |  |  |  |  |  |
| fatal |  |  |  |  |  |  |
| debug |  |  |  |  |  |  |

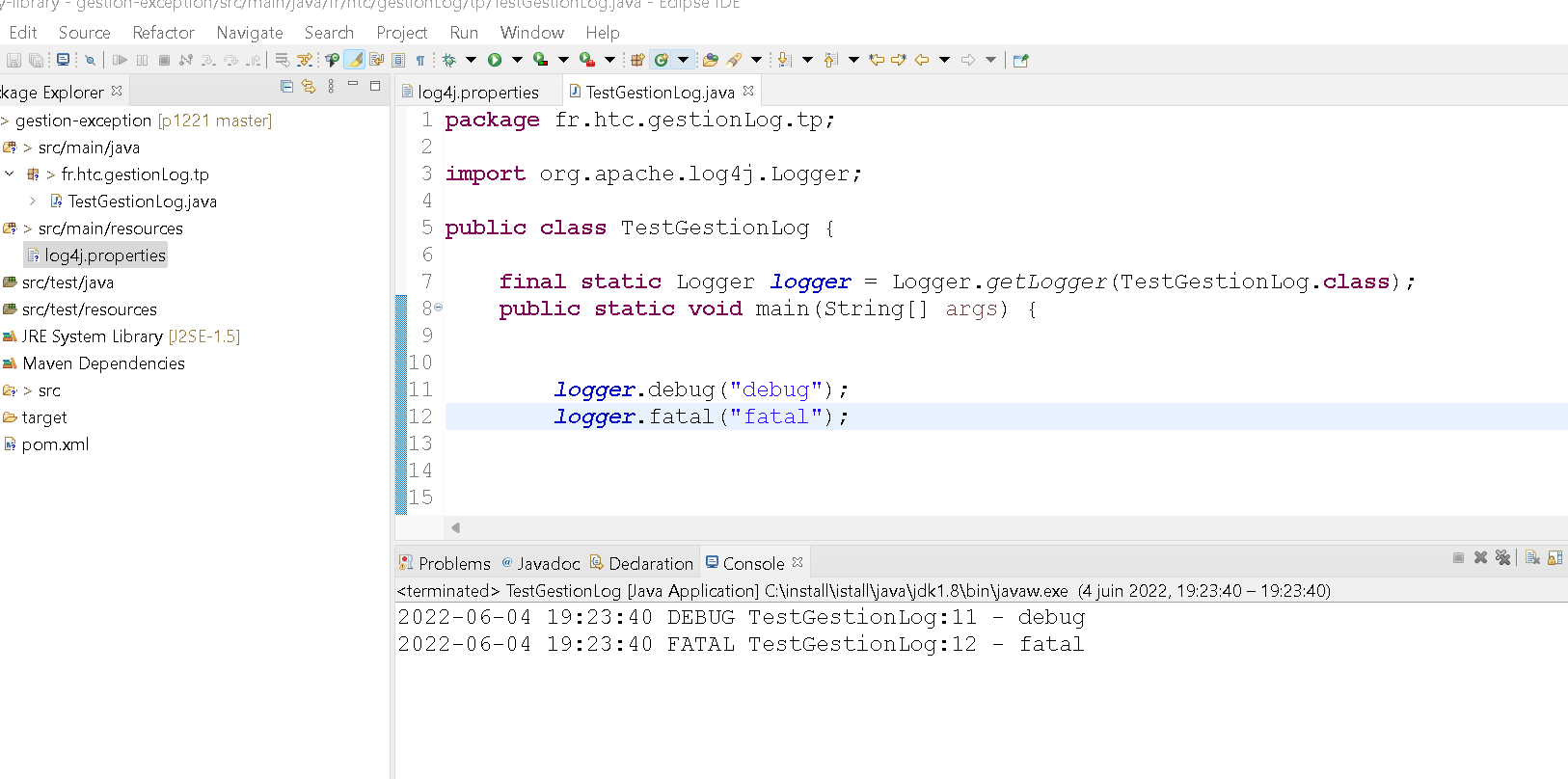
+ grave

- grave

**Trace, debug, info, warn, error, fatal**







## 1- Ecriture sur un fichier .log :

