Le 27 février 2015

Design Object

Framework de Logging

Equipe de projet :

Cécile Betmont

Adrien Frasca

Laura Hilliquin

Notre projet se trouve sur le répertoire Github à l’adresse  [https://github.com/danteAl/Betmont\_Frasca\_Hilliquin\_Architecture/commits/master](%20https://github.com/danteAl/Betmont_Frasca_Hilliquin_Architecture/commits/master).

Ce travail a été réalisé par Cécile Betmont, Adrien Frasca et Laura Hilliquin.

Sur le répertoire github, vous retrouverez les commits d’Adrien sous le nom de ‘AlDante’, ceux de Laura sous le nom de ‘Hilliquin’ et Cécile soit sous le pseudo de cecile-betmont ou bien « unknown » (en changeant sa clé ssh, le répertoire github ne la reconnait plus en tant que cecile-betmont).

**Installation**

Pour utiliser notre Logger, vous devez récupérer le dossier « » en effectuant un « git pull origin master à partir d’un dosser git.

Dans un second temps, vous devez charger le dossier contenant notre framework alors ainsi créé sur Eclipse. Ouvrez la classe « main » puis lancez le framework.

**Fonctionnement du Framework**

Le framework mis en place a pour but de récolter les différentes erreurs amenées par un programme et de les envoyer vers des cibles particulières, à un format spécifique. Le niveau et l’importance de ces erreurs sont classés par niveau.

L’utilisateur de notre framework se trouve en mesure de configurer ses logs à sa convenance. Nous avons choisi d’introduire des paramètres par défauts, pour un utilisateur ne souhaitant pas configurer minutieusement son logger.

L’utilisateur instancie son logger spécifique à une classe en utilisant la méthode « getLogger » de la classe « LoggerFactory ». Son logger est alors lié à sa classe et stockée dans une liste présente au sein de la classe « LoggerFactory ». Pour chaque logger, l’utilisateur spécifiera, grâce aux méthodes de la classe « Configuration », le niveau qu’il veut donner à sa classe, le format de logs ainsi que chacune des cibles vers lesquelles il veut envoyer ses logs. Si l’utilisateur ne spécifie pas ces informations, par défauts, le niveau de logs sera en debug, la cible, un fichier txt appelé « » enfin, le formateur sera de la forme : Wed Feb 25 19:29:20 CET 2015 [Name = Main, Level = DEBUG, Message = Test log1 ESIEA]

1. Ajouter un format de message

Pour cela, il se doit de créer son propre formateur : l’utilisateur crée sa propre classe héritée de la classe abstraite Formatter en redéfinissant la classe publique « format ».

L’exemple d’ajout d’un formateur dans notre classe main :

**private** **static** **class** MyCustomFormatter **extends** Formatter {

@Override

**public** String format(String message) {

StringBuffer sb = **new** StringBuffer();

sb.append("Prefix :\n");

sb.append(message + "\n");

sb.append("Suffix :\n");

**return** sb.toString();

}

Une fois le nouveau format spécifié, l’utilisateur l’ajoute aux configurations grâce à la ligne :

Configuration.setLayout(MaClasse.class, new MyCustomFormatter())

1. La liste des cibles (handlers)

Nous avons conçu notre logger afin qu’il puisse gérer des cibles d’extension .txt simples et rotatifs.

Chaque logger est muni d’une liste de fichiers cibles. L’utilisateur est alors en mesure de configurer cette liste de fichiers. Il peut alors, grâce à la methode addHandler de la classe « Configuration », ajouter une cible en spécifiant le fichier texte à compléter :

Configuration.addHandler(Maclasse.class, new FileHandler(String fichier);

On peut aussi spécifier la volonté d’un fichier rotatif : une fois la taille du fichier dépassée 1Ko, notre système crée automatiquement un nouveau fichier dans lequel sera sauvegardé la suite des logs. Pour cela on utilise Configuration.addHandler(MaClasse.class, new FileRotateHandler(String fichier);

L’utilisateur pourra, de plus, spécifier la taille maximum de son fichier rotatif en utilisant le constructeur FileRotateHandler(String fichier, int tailleMax).

1. Le niveau des logs

Notre logger comporte les 7 niveaux différents suivants, d’importance croissante :

All, Debug, Info, Warn, error, fatal.

L’utilisateur spécifie, grâce à la méthode addLevel de la classe configuration, le niveau qu’il veut donner à ses logs.

Une fois le logger configuré, l’utilisateur n’aura qu’à appeler les fonctions principales permettant l’envoie du message aux format, niveau et vers les cibles spécifiques au logger : « logger.debug(« message ») ».

**Document d’architecture**

1 - Diagramme UML

2 – Explication de l’architecture

Notre Framework de LOG s’organise autour de deux packages.

Le plus important formé de 10 classes, com.esiea et le deuxième, appelé test, contenant la classe main.