**Compte rendu du projet**

**d’Architecture Logicielle**

**FRAMEWORK**

**DE LOGGING**



*Enseignant :* Nicolas Andre

Sommaire

[Introduction 2](#_Toc412786505)

[Dossier de maintenance 3](#_Toc412786506)

[Diagramme de classes : 4](#_Toc412786507)

[Architecture des classes : 4](#_Toc412786508)

## Introduction

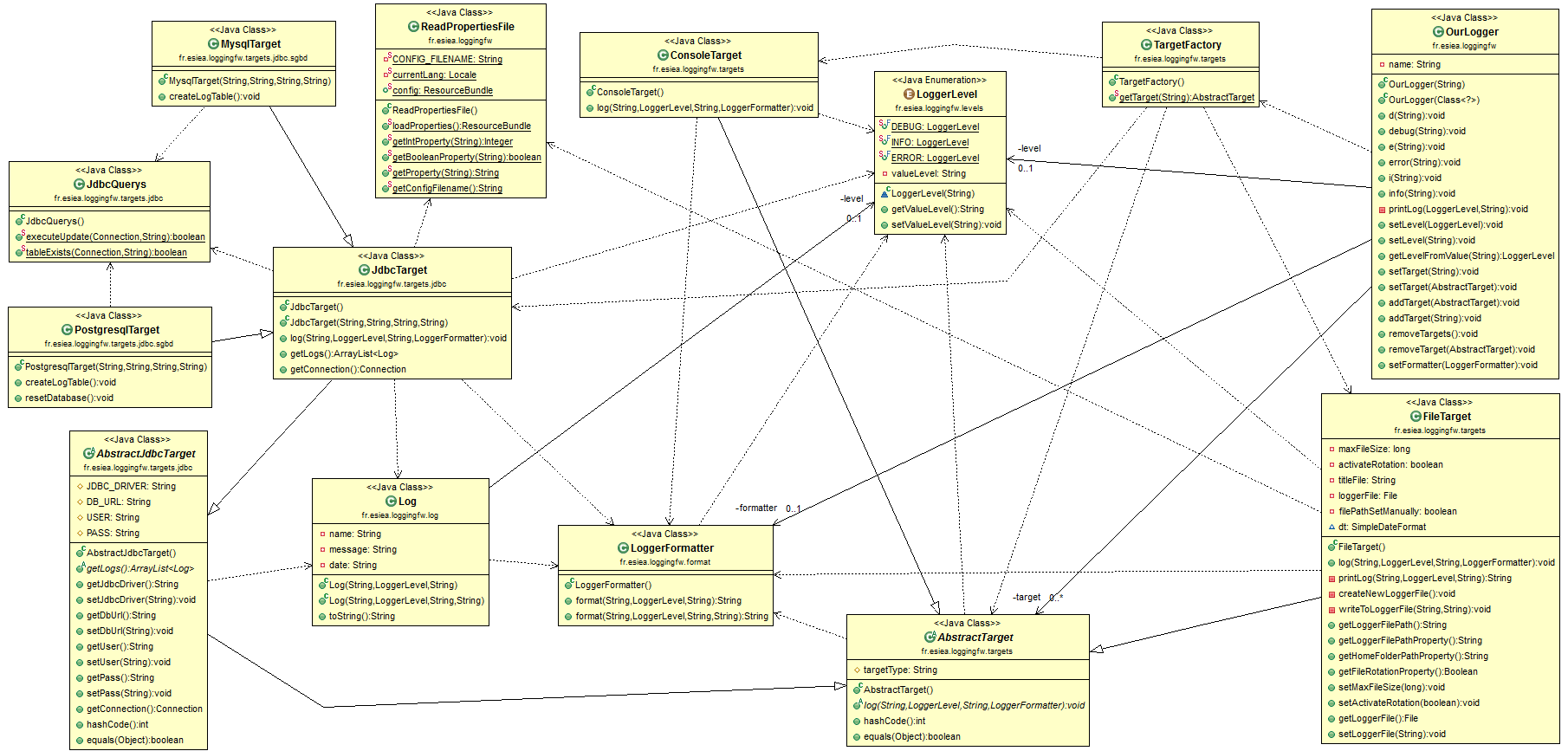
Dans le cadre du cours d’Architecture Logicielle, il nous a été demandé de concevoir et de développer un framework de logging.

Le but est de créer un système permettant de gérer des messages d’information, de debug ou d’erreur afin de mieux comprendre le fonctionnement d’un programme.

Ces messages générés pourront être afficher ou sauvegarder dans une console, dans un fichier ou dans une base de données.

## Dossier de maintenance

### Diagramme de classes :



### Architecture des classes :

* Package fr.esiea.loggingfw
  + Ourlogger : classe principale du logger.
    - Champs :
      * name : pour différencier les différents logs. généralement la source du log
      * level : le niveau de log (debug, error...)
      * target : la liste des cibles du logger
      * formatter : permet l'uniformisation de l'affichage des logs
    - Méthodes :
      * Les deux constructeurs qui instancient le logger, à partir d'une chaine de caractères ou à partir d'une classe
      * Les trois méthodes de debug, d'erreur et d'info qui définissent de quel level sera le log créé.
      * La méthode printLog() qui effectuera le log dans les différentes cibles du logger.
      * Les getters et setters pour le niveau de log ainsi que la priorité du log
      * Les méthodes setTarget(String), setTarget(AbstractTarget), addTarget(AbstractTarget), removeTargets() permettent d'ajouter des cibles au log ou au contraire à n'en définir qu'une seule.
  + ReadPropertiesFile : classe permettant de récupérer et d'utiliser le fichier properties du logger
    - Champs :
      * CONFIG\_FILENAME : chemin du fichier de configuration
      * currentLang : indique la localisation
      * config : de type ResourceBundle, appelle la méthode loadProperties()
    - Méthodes :
      * La méthode loadProperties permettant de charger les informations du fichier de properties.
      * getIntProperty(), getBooleanProperty(), getProperty() renvoient la valeur d'une propriété.
* Package fr.esiea.loggingfw.format
  + LoggerFormatter : classe permettant d'uniformiser l'affichage en console des logs
    - Champs :
      * result : retourne les informations du log mises en forme
    - Méthodes :
      * Deux méthodes, l'une affichant le name, le level et le message du log et l'autre y ajoutant la date.
* Package fr.esiea.loggingfw.levels
  + LoggerLevel : énumération contenant les différents niveaux de logs
    - Enum : DEBUG, INFO, ERROR
    - Champs :
      * valeurLevel : pour récupérer la bonne valeur de l'énum
    - Méthodes :
      * getter et setter du champ
* Package fr.esiea.loggingfw.log
  + Log : classe permettant de créer un log avec ses paramètres.
    - Champs :
      * name : pour différencier les différents logs. généralement la source du log
      * level : le niveau de log (debug, error...)
      * message : contenu du log donnant l'information
      * date : date et heure du log
    - Méthodes :
      * deux constructeurs, un avec tous les champs, l'autre sans la date
      * méthode toString() renvoyant vers un objet LoggerFormatter avec ses paramètres.
* Package fr.esiea.loggingfw.targets
  + AbstractTarget : classe abstraite de base pour les différentes cibles. Pour créer une nouvelle cible celle-ci doit obligatoirement hériter de cette classe.
    - Champs :
      * targetType : chaine de caractères servant à comparer deux cibles entre elles.
    - Méthodes :
      * méthode log(String name, LoggerLevel level, String message, LoggerFormatter formatter) utilisée par toutes les cibles (Target) lorsqu'elle effectue un log.
      * méthodes hashCode() et equals()
  + ConsoleTarget : classe de la cible correspondant à la console
    - Champs : aucun
    - Méthodes :
      * surcharge de la méthode log() de la classe abstraite mère
  + FileTarget : classe de la cible correspondant à l'écriture dans un fichier
    - Champs :
      * maxFileSize : taille en octets par défaut
      * activateRotation : appelle la méthode getFileRotationProperty()
      * titleFile et loggerFile
      * filePathSetManually : sert à forcer le fichier à se réécrire si on choisit le mode rotatif plutôt que de créer un autre fichier.
      * dt : format de la date
    - Méthodes :
      * surcharge de la méthode log() de la classe mère avec le mode rotatif
      * createNewLoggerFile(), création d'un nouveau fichier contenant les logs
      * méthode writeToLoggerFile() et writeToLogger() permettant l'écriture des logs dans un fichier.
      * méthodes getLoggerFilePath() et getLoggerFilePathProperty() récupérant le nom du fichier s'il existe ou sinon en crée un à partir du fichier properties
      * getter et setter
  + TargetFactory : classe permettant choisir la cible souhaitée.
    - Champs :
      * param : variable désignant le type de cible voulue
      * target : de type AbstractTarget, création de l'objet correspondant à la cible demandée.
    - Méthodes :
      * méthode getTarget(), teste la valeur de param afin de créer une instance de la cible correspondante.
* Package fr.esiea.loggingfw.targets.jdbc
  + AbstractJdbcTarget : classe abstraite de base pour les cibles utilisant une base de données. Pour créer une nouvelle cible de type JDBC celle-ci doit obligatoirement hériter de cette classe.
    - Champs :
      * JDBC\_DRIVER : driver jdbc utilisé suivant que l'on utilise la base de données voulue
      * DB\_URL : url de la base de données
      * USER : identifiant de l'utilisateur
      * PASS : mot de passe optionnel pour accéder à la BD
    - Méthodes :
      * getter et setter des champs, retournent les valeurs inscrit dans la base
      * méthode getConnection() permettant de se connecter à la base de données à l'aide des identifiants en paramètres.
      * méthodes hashCode() et equals()
  + JdbcQuerys : classe exécutant une mise à jour faite sur table de la base de données
    - Champs :
      * conn et connection : connexion à la base de données
      * query : requête que l'on souhaite effectuer
      * nomTable : nom de la table dont on vérifie l'existence
      * return : booléen validant ou non la requête
    - Méthodes :
      * méthode executeUpdate(Connection, String) exécutant la mise à jour de la table.
      * méthode tableExists() permet de vérifier si la table demandée est bien présente dans la base.
  + JdbcTarget : classe de la cible correspondant à la base de données
    - Champs : aucun
    - Méthodes :
      * deux constructeurs
      * surcharge de la méthode log() de la classe mère
      * méthode getLogs(), retourne une ArrayList contenant les logs
      * getConnection() permet la connexion à la BD
* Package fr.esiea.loggingfw.targets.jdbc.sgbd
  + MysqlTarget : classe héritant de JdbcTarget permettant d'utiliser une cible Mysql.
    - Champs : aucun
    - Méthodes :
      * constructeur de la classe
      * méthode createLogTable() permettant de créer la table de log et le trigger associé pour la date.
  + PostgresqlTarget : classe héritant de JdbcTarget permettant d'utiliser une cible Postgresql
    - Champs : aucun
    - Méthodes :
      * constructeur de la classe
      * méthode createLogTable() permettant de créer la table de log.
      * méthode resetDatabase() permettant de supprimer la table de log et la séquence associée. L'ensemble est ensuite recréé.