

# Projet OPL

Master 2 IAGL

---

---

## CheckCorrector : Automatic analysis and correction of Java code violations for a better Pull Request

---

**Réalisé par :**  
Jihad Mansour El  
Fakawy  
Badr Rahal

**Encadré par :**  
M. Martin Monperrus

**Date du rendu : 11/10/2016 pour :**  
M. Martin Monperrus

Formations En Informatique De Lille, FIL - Bât. M3 - Cité Scientifique  
59655 Villeneuve d'Ascq Cedex  
Tél. +33 (0) 3.20.43.44.94  
[www.fil.univ-lille1.fr/](http://www.fil.univ-lille1.fr/)

Année universitaire 2016-2017



# CheckCorrector : Automatic analysis and correction of Java code violations for a better Pull Request

Jihad Mansour El Fakawy

Badr Rahal

11 Octobre 2016





# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.2	Objectif . . . . .	3
1.3	Méthode . . . . .	4
<b>2</b>	<b>Travail technique</b>	<b>5</b>
2.1	But . . . . .	5
2.1.1	Analyse et correction du code . . . . .	5
2.1.2	Création d'une bonne pull request . . . . .	6
2.2	Overview et Workflows . . . . .	6
2.3	Architecture . . . . .	8
2.3.1	Langage . . . . .	8
2.3.2	Librairies utilisées . . . . .	8
2.3.3	Modules et classes . . . . .	10
2.4	Implémentation . . . . .	10
2.4.1	Analyse et Correction du code avec Spoon . . . . .	10
2.4.2	JGit et l'API Github . . . . .	12
2.5	Utilisation . . . . .	14
<b>3</b>	<b>Evaluation</b>	<b>15</b>
3.1	Efficacité . . . . .	15
3.2	Complexité . . . . .	15
3.3	Performance . . . . .	15
3.4	Facilité d'utilisation . . . . .	16
	<b>Conclusion</b>	<b>17</b>

# Chapitre 1

## Introduction

### 1.1 Contexte

Dans un milieu de développement collaboratif et pour la contribution à un projet, une Pull Request est une méthode de soumission des contributions à un projet de développement. C'est souvent la meilleure façon de soumettre des contributions à un projet à l'aide d'un système de contrôle de version distribué comme Git. Une demande de tirage se produit lorsqu'un développeur demande des changements déterminés à un référentiel externe pour l'inclusion dans le dépôt principal d'un projet. C'est donc l'ultime étape d'un collaborateur pour que sa contribution soit acceptée et reconnue. La qualité de la Pull Request doit donc être irréprochable, ainsi il est nécessaire de veiller à ce qu'elle soit bien soumise.

### 1.2 Objectif

Les développeurs font bien souvent des erreurs de style qui ne respectent pas les normes et standards. Ces erreurs impactent directement la performance du collaborateur, et la qualité des livrables. Eventuellement cela freine l'équipe de développement.

Notre objectif est d'aider le collaborateur à parfaire sa Pull Request, tout en apprenant de ces erreurs. Ceci augmentera la qualité des livrables et motivera le collaborateur à apporter le plus d'attention possible à son style.

### 1.3 Méthode

Le rôle de CheckCorrector est donc d'analyser et corriger le code du collaborateur, et essentiellement se charger de créer la Pull Request pour lui, avec la certitude que celle-ci respectera les normes et standards de codage imposés.



## Chapitre 2

# Travail technique

### 2.1 But

L'intérêt de ce travail est de nous permettre de développer un outil qui sera utile pour l'automatisation de différentes tâches du processus de soumission d'une contribution, pour enfin créer une bonne pull request.

Pour cela, nous nous sommes fixés un périmètre de travail :

- Analyse et correction du code.
- Création de la pull request.

#### 2.1.1 Analyse et correction du code

Pour cette partie, nous avons défini et limité clairement les règles et standard qu'analysera et corrigera notre outil avant de créer la Pull Request.

#### Règles de nommage :

- Variables locales, finales : Doit être sous la forme : strName, sinon on remplace le premier caractère en minuscule et le caractère avant dernier par une Majuscule, avec un commentaire TODO au collaborateur pour vérifier que la variable est bien en CamelCase .
- Constantes : Doit être en majuscules, sinon on remplace les caractères minuscules en Majuscules.

**Fonctionnalité :**

- Présence de la Javadoc , sinon on génère une javadoc par défaut avec un commentaire TODO pour le collaborateur afin de la compléter.
- Présence d'un commentaire spécifiant l'utilité de la classe sinon On ajoute un commentaire TODO pour le collaborateur.
- Pas de retour à la ligne successives, sinon on supprime les retours à la ligne et on ne laisse qu'1 seul.
- Longueur max d'une ligne : 80 caractères sinon on ajoute un commentaire TODO pour le collaborateur.
- Nombre max de paramètres : 7 sinon on ajoute un commentaire TODO pour le collaborateur.
- Détecter les méthodes vides et les supprimer.
- Vérifier que les switch on bien un défaut case sinon ajouter un par défaut

**2.1.2 Création d'une bonne pull request**

Pour cette partie, il fallait utiliser des bibliothèques et API disponible, afin de pouvoir créer une Pull Request à partir du résultat obtenu de l'analyse et la correction.

- Indexer les fichiers
- Commiter les changements
- Pousser le commit
- Créer la Pull Request
- Mettre à jour la Pull Request

**2.2 Overview et Workflows**

L'idée de notre projet est représentée par le workflow ci-dessous.

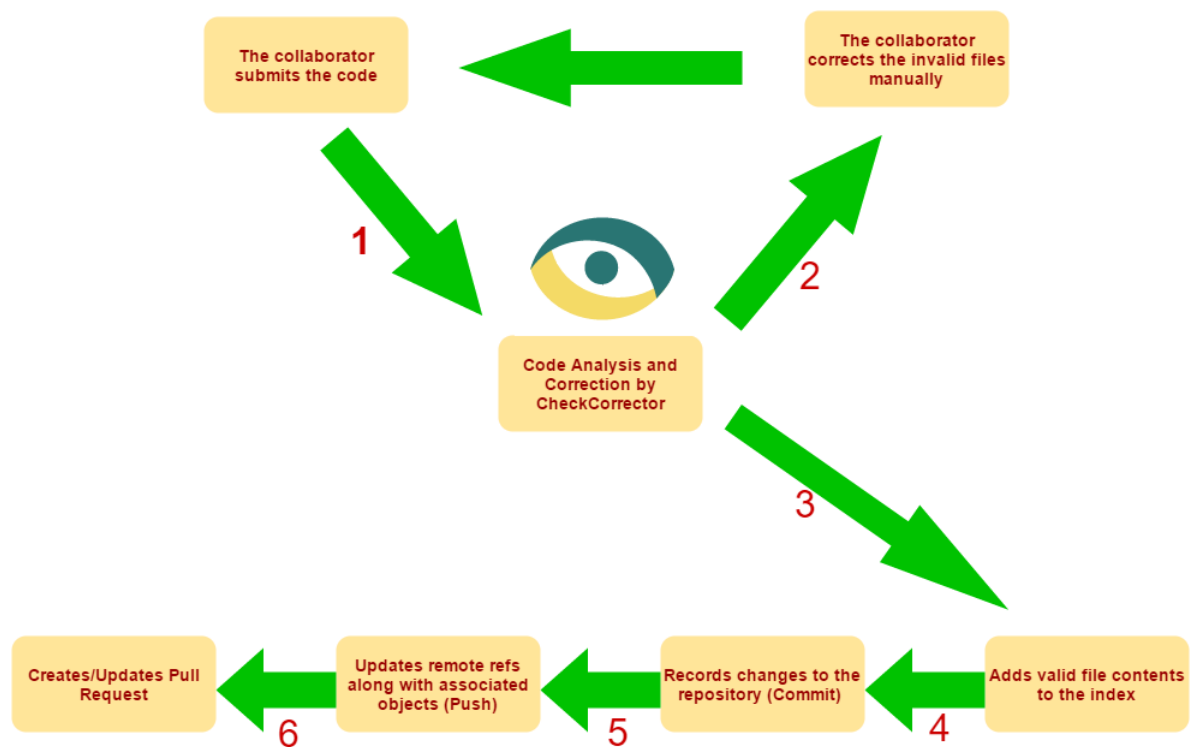


FIGURE 2.1 – CheckCorrector Workflow

1. Le collaborateur soumet son code à l'outil, qui va s'occuper d'analyser, corriger et créer la pull request
2. Dans le cas où le collaborateur n'as pas respecté les standards de code java, et qu'il y a un besoin de correction manuelle, dans ce cas-là il est obligé de le faire.
3. Quand 1 ou plusieurs fichiers respectent les normes, CheckCorrector les indexes.
4. Commiter les modifications.
5. Pousser le commit au repertoire distant.
6. Créer la pull request ou la mettre à jour.

## 2.3 Architecture

### 2.3.1 Langage

Pour développer notre outil, on a choisi le langage Java, un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982).

Il était nécessaire de le choisir pour pouvoir utiliser une certaine librairie d'analyse et manipulation de code source Java.



FIGURE 2.2 – Java

### 2.3.2 Bibliothèques utilisées

#### Spoon

Spoon est une bibliothèque open-source développée à l'INRIA, qui permet de transformer et d'analyser le code source Java. Spoon fournit un méta modèle complet où tout élément de programme (classes, méthodes, champs, déclarations, expressions ...) sont accessibles à la fois pour la lecture et la modification. Spoon prend pour entrée un code source, et produit le code source transformé prêt à compiler. Nous l'avons intégré dans Maven.



FIGURE 2.3 – Spoon

### JGit

Pour utiliser Git depuis notre programme Java, nous avons utilisés la bibliothèque JGit. JGit est une réalisation relativement complète de Git écrite en Java.

JGit propose deux niveaux généraux d'interfaçage logiciel : l'API Plomberie et l'API Porcelaine. Porcelaine est une interface de haut niveau pour des interactions de niveau utilisateur, tandis que Plomberie permet uniquement d'interagir directement avec les objets de bas niveau du dépôt.

Nous avons logiquement utilisés que Porcelaine, car notre but était de pouvoir implémenter le genre de choses qu'un utilisateur normal ferait en utilisant la ligne de commande.



FIGURE 2.4 – JGit

### GitHub Java API

GitHub Java API est une bibliothèque Java pour communiquer avec l'API GitHub. Le but de la bibliothèque est de soutenir l'API v3 GitHub. Elle est actuellement utilisée par le connecteur Mylyn GitHub pour travailler

sur des issues GitHub, Pull Request, gists, et repositories à partir d'Eclipse. (EGit extension git pour Eclipse)

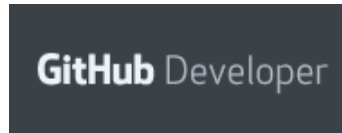


FIGURE 2.5 – Github API

### 2.3.3 Modules et classes

Les différents Package du projet :

- CheckClassOk : Package qui va vérifier les classes qui sont syntaxiquement correct.
- DuplicateCode : Package qui va vérifier la duplication de Code.
- Field : Package qui va vérifier la syntaxe des Attributs.
- Method : Package qui va vérifier la syntaxe des Méthodes.
- MethodUnused : Package qui va vérifier le code Mort.
- While : Package qui va étudier les boucles infinies.
- Catch : Package qui va étudier les Catches.
- PullRequestEngineering : Package qui va faire une pull Request des fichiers syntaxiquement Correctes.
- UserConfiguration : Package qui permettra à l'utilisateur de configurer l'outil.

## 2.4 Implémentation

### 2.4.1 Analyse et Correction du code avec Spoon

#### Processeurs Spoon

Dans le cas de notre développement. Nous utiliserons les « Processeurs » de Spoon. Les Processeurs nous permettront de parcourir le code d'un « élément » spécifique d'un fichier java.

```

10 import spoon.reflect.declaration.CtMethod;
11
12 public class MethodCheckProcessor extends AbstractProcessor<CtMethod<?>> {
13     private HashMap<String, Integer> map;
14     private ArrayList<String> useFile;
15     private HashMap<List<CtStatement>, ArrayList<String>> duplicate;
16
17     public MethodCheckProcessor(HashMap<String, Integer> map, ArrayList<String> useFile,
18                               HashMap<List<CtStatement>, ArrayList<String>> duplicate) {
19         this.map = map;
20         this.useFile = useFile;
21         this.duplicate = duplicate;
22     }
23
24     @Override
25     public boolean isToBeProcessed(CtMethod<?> candidate) {
26         return true;
27     }
28
29     @Override
30     public void process(CtMethod<?> element) {
31         new Duplicate(element, duplicate).verif();
32         String method = element.getReference().toString();
33         if (!method.contains(ConstantsMethod.SIGNATURE_MAIN)) {
34             if (!this.map.containsKey(method)) {
35                 this.map.put(method, 0);
36             }
37             new CheckMethod(element, useFile).verifier();
38         }
39     }
40 }

```

FIGURE 2.6 – MethodCheckProcessor

Dans cet Exemple de code, on remarque que notre classe étend `AbstractProcessor<CtMethod<?>>`, c'est ici un processeur qui va analyser les Méthodes des classes java. Les modifications se feront dans la méthode `process` qui contiendra l'élément en paramètre.

## Launcher Spoon

Le Launcher va nous permettre de lancer notre programme, il faudra donc lui indiquer les fichiers java à analyser, le chemin de sortie des fichiers, ajouter les processeurs qui analyseront le code et pour finir lancer le programme.

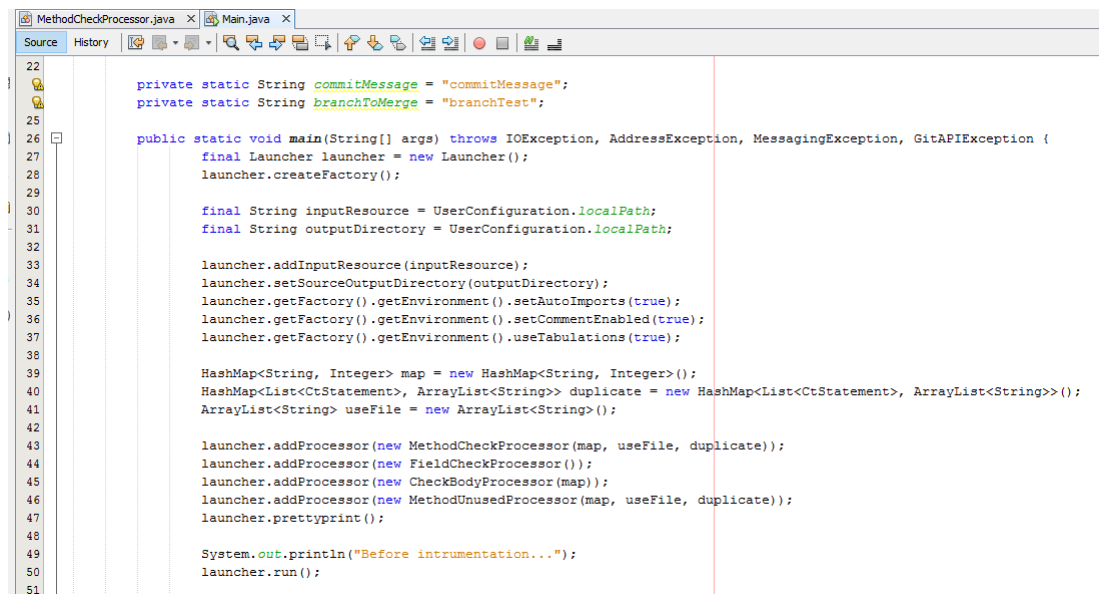


FIGURE 2.7 – Launcher

On retrouve donc :

- addInputResource(inputResource)-> ajoute les fichiers java à analyser.
- setSourceOutputDirectory(outputDirectory)-> Indiquer la sortie des fichiers.
- addProcessor(...)-> ajoute les processeurs.
- run()-> lancement des processeurs.

A la fin de l'analyse et la correction du code, on a configuré notre outil pour générer 2 tableaux : ValidFiles et InvalidFiles. L'intérêt est de pouvoir les utiliser pour notre logique métier afin de créer une pull request uniquement pour les fichiers valides.

## 2.4.2 JGit et l'API Github

### Jgit

Pour des activités fréquentes, JGit fournit un ensemble de fonctionnalités de haut niveau pour simplifier par exemple l'ajout de fichier à l'index ou la validation, et le point d'entrée c'est la classe **Git**

C'est la structure habituelle avec la classe Git, les méthodes renvoient



un objet commande qui permet d'enchaîner les appels de paramétrage qui sont finalement exécutés par l'appel à `.call()`.

On a utilisé JGit pour toutes les commandes basic git :

- Indexer les fichiers
- Commiter les changements
- Pousser le commit

Pour cela, on récupère le tableau des fichiers valides généré par Spoon, on le parcourt pour pousser les changements.

---

**Code Java 1** Git commands

---

```
for (String ValidFileName : ValidFile) {  
    GitImplementation gitCmd =  
    new GitImplementation(UserConfiguration.localPath, UserConfiguration.remotePath,  
    UserConfiguration.user, UserConfiguration.password);  
    gitCmd.addFile(ValidFileName);  
    gitCmd.commitChanges(commitMessage);  
    gitCmd.pushCommit();  
}
```

---

Pour l'authentification, il y'a un objet spécial nommé `CredentialsProvider`.

---

**Code Java 2** L'authentification Github avec `CredentialsProvider`

---

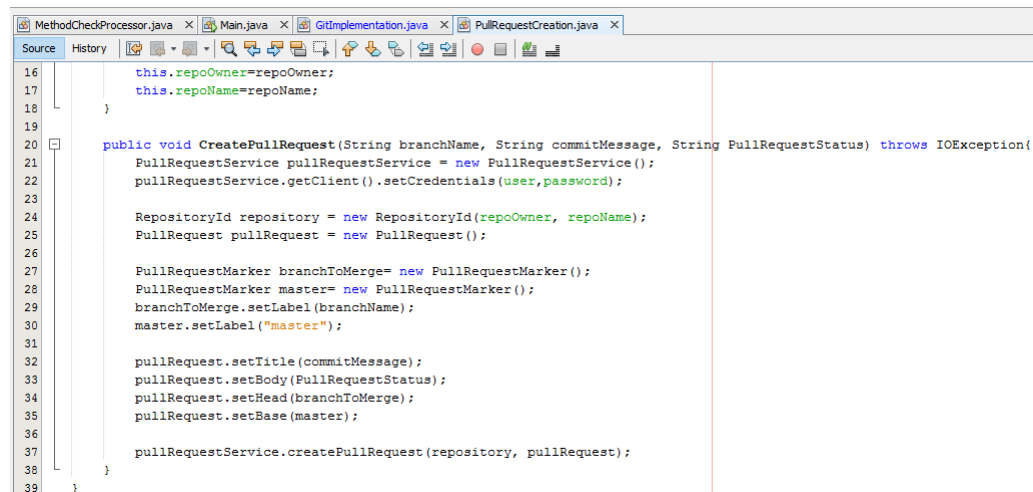
```
CredentialsProvider cp =  
new UsernamePasswordCredentialsProvider("username", "password");  
.setCredentialsProvider(cp);
```

---

## API Github

On a aussi utilisé un objet `CredentialsProvider` pour l'authentification Github.

Il fallait bien évidemment configuré correctement les données envoyées à travers l'API :



```

16         this.repoOwner=repoOwner;
17         this.repoName=repoName;
18     }
19
20     public void CreatePullRequest(String branchName, String commitMessage, String PullRequestStatus) throws IOException{
21         PullRequestService pullRequestService = new PullRequestService();
22         pullRequestService.getClient().setCredentials(user,password);
23
24         RepositoryId repository = new RepositoryId(repoOwner, repoName);
25         PullRequest pullRequest = new PullRequest();
26
27         PullRequestMarker branchToMerge= new PullRequestMarker();
28         PullRequestMarker master= new PullRequestMarker();
29         branchToMerge.setLabel(branchName);
30         master.setLabel("master");
31
32         pullRequest.setTitle(commitMessage);
33         pullRequest.setBody(PullRequestStatus);
34         pullRequest.setHead(branchToMerge);
35         pullRequest.setBase(master);
36
37         pullRequestService.createPullRequest(repository, pullRequest);
38     }
39 }

```

FIGURE 2.8 – CreatePullRequest

pullrequest

## 2.5 Utilisation

Pour utiliser CheckCorrector, l'utilisateur doit tout d'abord, configurer ses paramètres, ils sont disponibles dans la classe **UserConfiguration** du package UserConfiguration .

Ensuite il pourra exécuter l'outil :

- Si tous les fichiers respectent les normes : Création de la pull request avec tous les fichiers.
- Sinon : Création de la pull request avec uniquement les fichiers valides, après l'utilisateur est obligé de corriger manuellement quelques erreurs, par la suite la pull request sera mise à jour automatiquement.

## Chapitre 3

# Evaluation

### 3.1 Efficacité

CheckCorrector est un outil efficace pour contrer les erreurs comises par les collaborateurs, en effet, nous avons pu implémenter différentes règles de codage, ainsi la Pull Request créée respectera ces normes. Bien évidemment ceci aidera avec une grande efficacité le Reviewer de la Pull Request, il ne se souciera plus de vérifier si la contribution du collaborateur est dans les normes, l'outil aura déjà fait ça à sa place.

Ceci lui permettra de gagner du temps et de se concentrer plus sur les autres aspects.

### 3.2 Complexité

La création d'un Pull Request complète prendra plus de temps pour le collaborateur, il sera amené à réviser sa contribution. Il devra s'assurer de bien configurer l'outil ( Un répertoire locale pour l'input, données d'authentification ...)

### 3.3 Performance

CheckCorrector analyse et veille à ce que le code respecte les règles les plus utilisées, ainsi la contribution sera conforme aux attentes. La performance qu'il possède permet d'automatiser tout le processus de contribution à un projet en juste quelques configurations.

### 3.4 Facilité d'utilisation

En moyenne, le collaborateur devra exécuter CheckCorrector 2 fois pour que ça Pull Request soit complète, dans l'hypothèse qu'il a commit des erreurs de nommage ou autres. Ceci sera un peu complexe pour lui. Néanmoins, ça le poussera à bien respecter les normes, et par la suite sa pull request sera créer après 1 seule exécution désormais qu'il maitrise les règles et standards de codage java.

# Conclusion

CheckCorrector va permettre au développeur de faire une pull Request clean, et ainsi participer sans pour autant apporté des problèmes de qualités de code.

Cependant il y'a un risque de faire des modifications non souhaitées, mais il y'a toujours une possibilité d'amélioration. L'outil ne permet pas de vérifier toutes les règles et normes pour l'instant.