Sonar fixer

Clement Maxime - Piorun Jordan

November 9, 2015

1 Introduction

Pour de nombreux projets, nous avons recours a Sonar, un logiciel permettant de mesurer la qualité de notre code en continu. Le rapport de Sonar classe les améliorations a apporter en différentes catégories, des erreurs critiques aux simples informations.

Pour notre projet, nous somme parti d'un simple constat : les erreurs sonar sont souvent les mêmes, et un nombre considérable d'entre elles pourrait être corrigées de façon automatique. Cela passe par deux étapes : premièrement il faut repérer le morceau de code qui est à l'origine de l'erreur, puis il faut ensuite le remplacer avec du code équivalent.

pour ces deux besoins, nous avons choisis d'utiliser Spoon, une librairie développée à l'Inria permettant d'analyser et de transformer du code Java.

2 Analyse du code

2.1 Le scope

Avant même de s'intéresser à la partie technique de ce projet, nous avons du établir la liste des erreurs que nous voulions corriger.

Sonar classe les erreurs en 5 catégories : bloquantes, critiques, majeures, mineures et informatives. Parmi celles ci, nous avons jugé que les plus pertinentes, mais aussi les plus urgentes à corriger sont les erreurs majeures.

En effet, les deux catégories au dessus gênent le bon fonctionnement du programme et sont donc en toute logique détectée au préalable par le développeur.

Les niveaux de sévérité moins importants quant à eux relèvent plus de conventions de codages, de format de code ou autres problèmes pouvant gêner la compréhension du code. Nous gardons à l'idée que comprendre un code facilement est essentiel pour le maintenir, mais nous préférons mettre la priorité sur la catégorie restante : les erreurs majeures.

Ces alertes Sonar proviennent de mauvaises pratiques de codage, qui ne gênent pas le bon déroulement du code, mais qui pour certaines d'entre elles, pourraient entrainer des erreurs lors d'ajout de code. Après cette première réduction du scope initial, nous avons fais le tri entre les alertes que nous pouvons corriger de façon automatique, et celles qui dépendent du contexte. Suite a cette seconde analyse, nous avons dressé le tableau en annexe 1 qui reprend les différentes alertes ainsi que la solution apportée.

2.2 Détection d'erreur

Afin de détecter les erreurs que nous voulons corriger, nous nous sommes appuyé sur la librairie Spoon, qui nous apporte une solution pour analyser du code Java. Pour chaque alerte, nous avons donc un une classe Processor associée dans laquelle nous décrivons le patern à rechercher. Lorsque cette fonction trouve du code qui match, nous pouvons commencer sa transformation.

3 La correction du code

4 Test de notre projet

4.1 Les test unitaires

4.2 Test sur un projet de taille

A Annexe 1 - Tableaux des correctifs

Noncompliant	Compliant
`if (myClass.compareTo(arg) == -1) {...}`	`if (myClass.compareTo(arg) < 0) {...}`

`class Bar {`
`  public boolean doSomething(){...}`
`}`
`class Foo extends Bar {`
`  public boolean doSomething(){...}`
`}`
	`class Bar {`
	`  public boolean doSomething(){...}`
	`}`
	`class Foo extends Bar {`
	`  @Override`
	`  public boolean doSomething(){...}`
	`}`

`public void a(Map<String,Object> map) {`
`  for (String key : map.keySet()) {`
`  Object value = map.get(key);`
`  ...`
`  }`
`}`
	`public void a(Map<String,Object> map) {`
	`  for (Map.Entry<String,Object> entry :`
	`  map.entrySet()) {`
	`  String key = entry.getKey();`
	`  Object value = entry.getValue();`
	`  ...`
	`  }`
	`}`

`foo.equals(bar.toUpperCase());`	`foo.equalsIgnoreCase(bar);`

`switch (myVariable) {`
`  case 0: // 6 lines till next case`
`  methodCall1("");`
`  methodCall2("");`
`  methodCall3("");`
`  methodCall4("");`
`  break;`
`  case 1:`
`  ...`
`}`
	`switch (myVariable) {`
	`  case 0:`
	`  doSomething()`
	`  break;`
	`  case 1:`
	`  ...`
	`}`
	`...`
	`private void doSomething(){`
	`  methodCall1("");`
	`  methodCall2("");`
	`  methodCall3("");`
	`  methodCall4("");`
	`}`

`public class MyClass {`
`  private int foo = 42;`
 	
`  public int compute(int a) {`
`  return a * 42;`
`  }`
`}`
	`public class MyClass {`
	`  public int compute(int a) {`
	`  return a * 42;`
	`  }`
	`}`

`class A {`
`  private int field;`
`}`
	`class A {`
	`  private int field;`
	
	`  A(int field) {`
	`  this.field = field;`
	`  }`
	`}`