Software Architecture And Design
Detecting Code Smells

Ioan Luca
201638554
University of Strathclyde

October 22, 2018

1 Code Smells Overview

I attempted the "Middle Man" code smell from the Hard category, the "Long Method", "Switch Statements" and "Primitive Obsession" code smells from the Medium category as well as the "Long Class" and 'Long Parameter List" from the Easy category.

1.1 Middle Man

This code smell happens when a class delegates all of its responsibility to another class, usually when attempting to reduce high coupling. However, this way it essentially becomes useless.

The software identifies such behaviour by using 2 visitors. The first one traverses all the methods in scope to compile a set of all the method names. The second visitor identifies methods with a single statement or with 2 statements out of which the outer one is a return. If the statement is a method call that is part of the list, then the smell was found.

1.2 Long Method and Long Class

Long methods/classes are a sign of bad incremental design. Functionality has been extended over time without taking into account refactoring. They are hard to read and they usually denote breaking of the "Single responsibility principle". The software detects such methods/classes by counting the statements inside its body, including arbitrarily nested statements. If the number exceeds 10/100, the smell was found.

1.3 Switch Statements

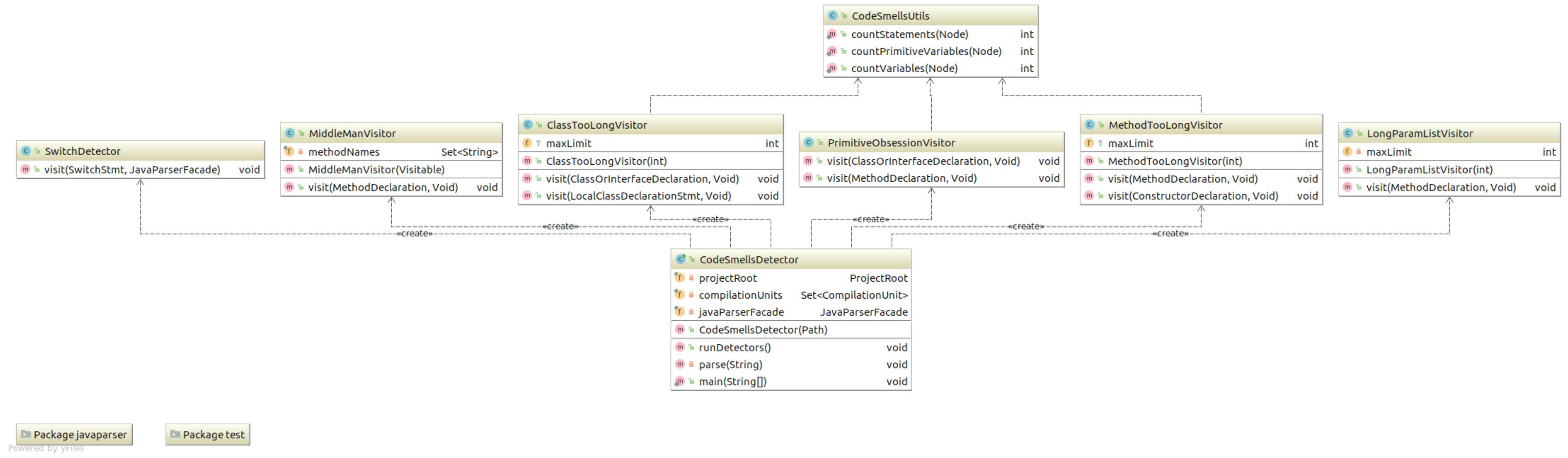
Switching on type codes is a bad practice in OOP and it should be handled by subclassing. This smell is detected when the expressions being switched on is of a user defined Enum type.

1.4 Primitive Obsession

This appears when sufficiently complex data is handled by primitive types instead of being abstracted away in a class. It favours the birth of Data Clumps smells as well. The software detects this smell inside method declarations and classes by counting all the total number of variables, parameters and fields. Then, the primitive declarations are selected from the total and ratio is calculated with $ratio = \frac{primitives}{total}$. If $ratio \geq 0.37$ and $primitive \geq 4$ then the code smell was discovered. The numbers came from a lot of trial and error and they seem to discover this code smell wherever it is present in the testing system.

1.5 Long Parameter List

Similar to the Primitive Obsession, having too many parameters breaks both encapsulation and abstraction and. In addition, such a code smell provides a clear clue that the implementation of the method is quite complex. The software detects parameters lists for both methods and constructors with more than 5 parameters.


CodeSmellsUtils
countStatements(Node) int
countPrimitiveVariables(Node) int
countVariables(Node) int
SwitchDetector
visit(SwitchStmt, JavaParserFacade) void
MiddleManVisitor
methodNames Set<String>
MiddleManVisitor(Visitable)
visit(MethodDeclaration, Void) void
ClassTooLongVisitor
maxLimit int
ClassTooLongVisitor(int)
visit(ClassOrInterfaceDeclaration, Void) void
visit(LocalClassDeclarationStmt, Void) void
PrimitiveObsessionVisitor
visit(ClassOrInterfaceDeclaration, Void) void
visit(MethodDeclaration, Void) void
MethodTooLongVisitor
maxLimit int
MethodTooLongVisitor(int)
visit(MethodDeclaration, Void) void
visit(ConstructorDeclaration, Void) void
LongParamListVisitor
maxLimit int
LongParamListVisitor(int)
visit(MethodDeclaration, Void) void
«create»
«create»
«create»
«create»
«create»
«create»
CodeSmellsDetector
projectRoot ProjectRoot
compilationUnits Set<CompilationUnit>
javaParserFacade JavaParserFacade
CodeSmellsDetector(Path)
runDetectors() void
parse(String) void
main(String[]) void
Package javaparser
Package test
Powered by yFiles
Figure 1: Each code smell is implemented using a visitor that extends either the generic JavaParser visitor or the the void one. The CodeSmellsDetector class is responsible for loading and parsing all the sources found in the test system. It also runs each visitor on compilation units and groups errors by classes. Finally, CodeSmellsUtil provides useful functions like counting nested statements.

2 High-level design

Each code smell is implemented using a visitor that extends either the generic JavaParser visitor or the the void one.

The CodeSmellsDetector class is responsible for loading and parsing all the sources found in the test system. It also runs each visitor on compilation units and groups errors by classes. Finally, CodeSmellsUtil provides useful functionality like counting nested statements.

The project was developed using Java 10 and Maven as a build tool plus git for version control. This report is written in LaTeX.

Newer versions of JavaParser provide convenience methods to traverse the AST in a functional style by using the Stream and Map-Reduce API and the source code tries to use them where possible.

The SwitchDetector is using SymbolParser to test whether the type switched on is an user defined Enum. SymbolParser is a new addition to JavaParser that implements various semantics based on the AST.

It is worth mentioning that when implementing Primitive Obsession, it is not enough to check whether a variable declaration is of a primitive type because the JavaParser provided method does not check for Strings and other boxed types. This is acceptable because they are in fact reference types in Java. This solution takes considers Strings as being primitives as well.

The runner class allows to specify a root folder for the Java source files that need to be analysed and a starting package — afterwards, it runs all the code smell detectors on each file.

The utility functions use a mix of Map-Reduce and Recursion to traverse the AST.

3 Results and Evaluation

The system seems to work reasonably well on the test system provided. I am not aware of any false positives. However, there are chances that this software would fail in more complex cases that I haven't thought about yet.

Moreover, there is uncertainty regarding whether the implementations that work well for typical method declarations behave the same on lambdas. They do for anonymous functions. Similarly, JavaParser has no support for the new *var* keyword that has recently been added to Java.

3.1 Program Output

Below is the output of generated by the program when run on the test system.

3.1.1 in type MorpionSolitairePanel->>

```
METHOD TOO LONG at mousePressed => it has 19 which is more than 10!
CONSTRUCTOR TOO LONG at MorpionSolitairePanel => it has 26 which is more than 10!
METHOD TOO LONG at run => it has 15 which is more than 10!
METHOD TOO LONG at start => it has 16 which is more than 10!
METHOD TOO LONG at paintComponent => it has 26 which is more than 10!
SWITCH ON ENUM test.Abusers.Switch.MorpionSolitairePanel.State at mousePressed
MIDDLEMAN at method start
```

3.1.2 in type ManOrBoy->>

```
LONG PARAMETER LIST at A => it has 6 which is more than 5!
```

3.1.3 in type Item->>

```
PRIMITIVE OBSESSION at CLASS Item -> 4 primitives out of 4 fields => that is 100.00 % primitives
```

3.1.4 in type Eertree->>

```
METHOD TOO LONG at eertree => it has 28 which is more than 10!
PRIMITIVE OBSESSION at METHOD eertree -> 8 primitives out of 9 variables => that is 88.89 %
primitives
```

3.1.5 in type BoxingTheCompass->>

METHOD TOO LONG at main => it has 13 which is more than 12!
PRIMITIVE OBSESSION at METHOD buildPoints –> 7 primitives out of 7 variables => that is 100.00 % primitives

3.1.6 in type FloodFill->>

METHOD TOO LONG at floodFill => it has 26 which is more than 10!
PRIMITIVE OBSESSION at METHOD floodFill –> 8 primitives out of 13 variables => that is 61.54 % primitives

3.1.7 in type Grid->>

LONG PARAMETER LIST at checkLine => it has 6 which is more than 5!
CLASS TOO LONG at Grid => it has 140 which is more than 100!
METHOD TOO LONG at newGame => it has 12 which is more than 10!
METHOD TOO LONG at draw => it has 36 which is more than 10!
METHOD TOO LONG at playerMove => it has 34 which is more than 10!
METHOD TOO LONG at checkLine => it has 12 which is more than 10!
METHOD TOO LONG at addLine => it has 11 which is more than 10!
PRIMITIVE OBSESSION at METHOD newGame –> 4 primitives out of 4 variables => that is 100.00 % primitives
PRIMITIVE OBSESSION at METHOD draw –> 14 primitives out of 16 variables => that is 87.50 % primitives
PRIMITIVE OBSESSION at METHOD playerMove –> 6 primitives out of 13 variables => that is 46.15 % primitives
PRIMITIVE OBSESSION at METHOD checkLine –> 11 primitives out of 12 variables => that is 91.67 % primitives
PRIMITIVE OBSESSION at CLASS Grid –> 26 primitives out of 29 fields => that is 89.66 % primitives

3.1.8 in type Luhn->>

METHOD TOO LONG at luhnTest => it has 12 which is more than 10!
PRIMITIVE OBSESSION at METHOD luhnTest –> 6 primitives out of 6 variables => that is 100.00 % primitives

3.1.9 in type AccountManager->>

MIDDLEMAN at method GetAccount

3.1.10 in type NBodySim->>

CONSTRUCTOR TOO LONG at NBody => it has 23 which is more than 10!
PRIMITIVE OBSESSION at METHOD decompose –> 5 primitives out of 5 variables => that is 100.00 % primitives
PRIMITIVE OBSESSION at CLASS NBody –> 4 primitives out of 7 fields => that is 57.14 % primitives

3.1.11 in type CipollasAlgorithm->>

METHOD TOO LONG at c => it has 29 which is more than 10!

3.1.12 in type BresenhamPanel->>

METHOD TOO LONG at paintComponent => it has 15 which is more than 10!
METHOD TOO LONG at drawLine => it has 28 which is more than 10!
PRIMITIVE OBSESSION at METHOD paintComponent –> 12 primitives out of 13 variables => that is 92.31 % primitives
PRIMITIVE OBSESSION at METHOD plot –> 10 primitives out of 11 variables => that is 90.91 % primitives
PRIMITIVE OBSESSION at METHOD drawLine –> 13 primitives out of 14 variables => that is 92.86 % primitives

3.1.13 in type HuffmanCode->>

METHOD TOO LONG at printCodes => it has 12 which is more than 10!

3.1.14 in type BarnsleyFern->>

METHOD TOO LONG at createFern => it has 19 which is more than 10!
PRIMITIVE OBSESSION at METHOD createFern –> 8 primitives out of 8 variables => that is 100.00 % primitives

3.1.15 in type Client->>

MIDDLEMAN at method something
MIDDLEMAN at method somethingElse

3.1.16 in type BarnsleyFernTwo->>

METHOD TOO LONG at createFernWithTemp => it has 18 which is more than 10!
PRIMITIVE OBSESSION at METHOD createFernWithTemp –> 6 primitives out of 6 variables => that is 100.00 % primitives

3.1.17 in type Test->>

PRIMITIVE OBSESSION at METHOD meanStdDev –> 4 primitives out of 5 variables => that is 80.00 % primitives
PRIMITIVE OBSESSION at METHOD showHistogram01 –> 5 primitives out of 6 variables => that is 83.33 % primitives
PRIMITIVE OBSESSION at CLASS Test –> 4 primitives out of 4 fields => that is 100.00 % primitives