Real World Idris
Project Scope and Outline

Ioan Luca
201638554

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 code 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.

2 High-level design

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

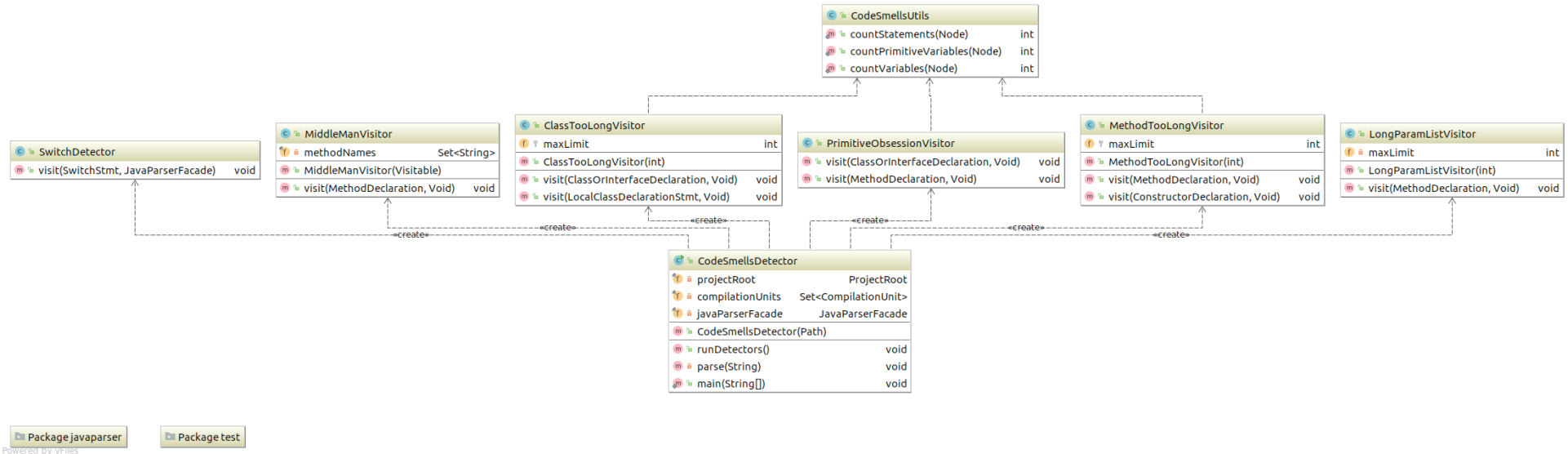

Figure 1: Each code smell is implemented using a visitor that extends either the generic JavaParser visitor or the the void one. The CodeSmellsDetector class is responsible for loading and parsing all the sources found in the test system. It also runs each visitor on compilation units and groups errors by classes. Finally, CodeSmellsUtil provides useful functions like counting nested statements.

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