# Project 1: Bad Smell Detection

Ardigò Susanna

November 6, 2020

# Contents

# 1 Ontology Creation

## 1.1 Goal and Input parameter

This part of the project consists of creating an ontology for Java Entities.
This file takes an optional argument which is the path of the python file that defines the Java Abstract Syntax Tree. If the argument is non supplied then a predefined path is used. For this project we used the file `tree.py` of the Javalang Python Library.

## 1.2 Description of the code

In order to efficiently parse this file we created a class named `Class` to store the name, superclasses and properties of each class. The function `get_classes(python_file_name)` reads the given file, parses into an Abstract Syntax Tree. We use the function `walk` to iterate the tree, create instances of `Class` with the class definition nodes and save them into an array.
The main function `start` creates an ontology using the library Owlready2. We then iterate the list of the parsed classes to create them in the ontology. This step needs to differentiate among three different contructions depending on the number of superclasses. If the current class has none, meaning that it has no super class, in the ontology it is created as a subclass of Thing which, in owl, is the top superclass of all classes. If the current class has one superclass, it is created as its subclass. If the current class has two superclass, it is created as subclass of both.
For each class we add the previously extracted properties and add them to the ontology. There are two different types: Object, which are only `body` and `parameters`, and Data, which are all other properties. Since the first type has only two possible values, we decided to add them only once at the end. To avoid conflict, when we create "name" properties we rename them to "jname".
The ontology is finally created and we can export it into an owl file.

## 1.3 Results

Figure 1 shows the hierarchy tree of the created classes and properties, table 2 shows their count.
We can see that `Statement` is the biggest superclass with 15 subclasses. The second biggest hierarchy is `Expression` with 8 direct subclasses some of which are superclasses themselves, with a total of 15 extra subclasses. `Documented` and `Declaration` share some subclasses.

| **Type** | **#** |
|---|---|
| Class | 78 |
| Object Property | 2 |
| Data Property | 65 |

Table 1: Count of created classes and properties

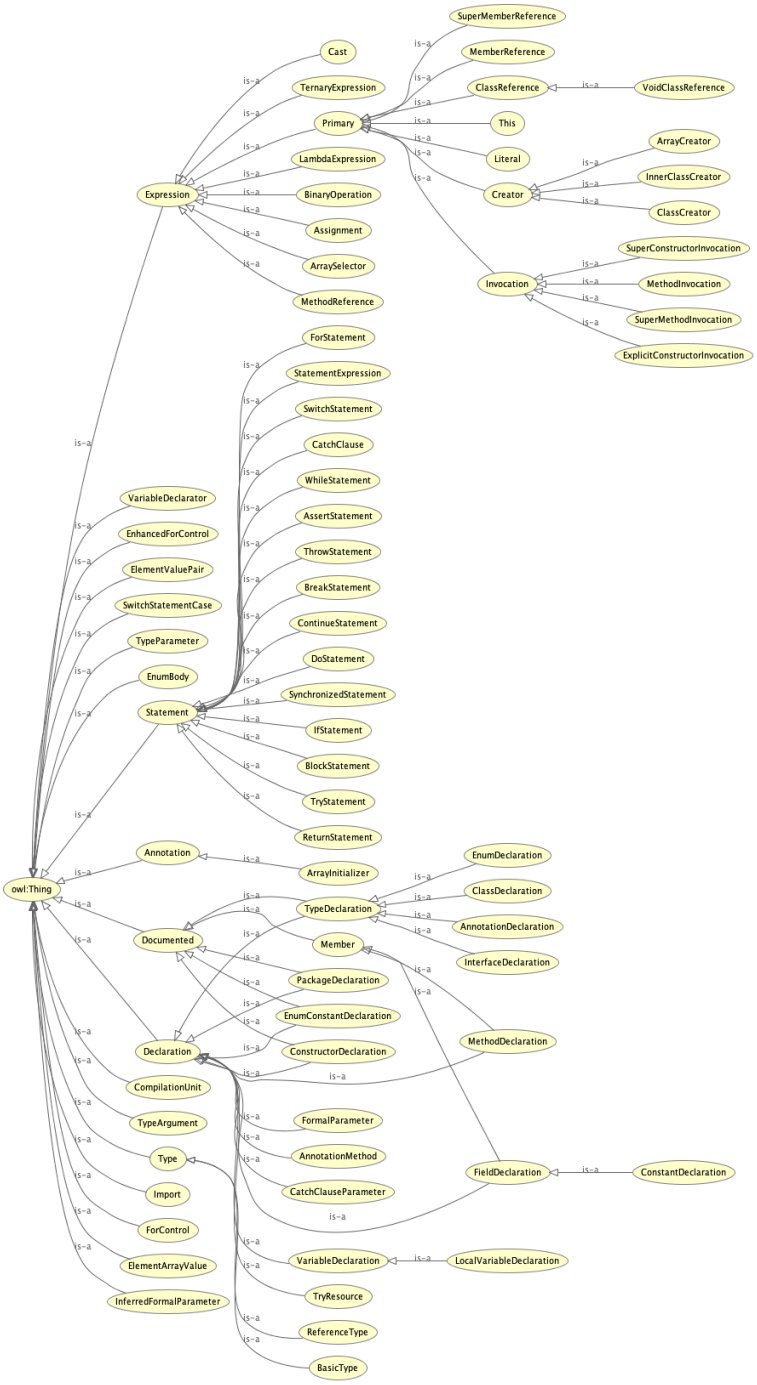


Figure 1: Class Hierarchy of the Ontology created

# 2 Populate the Ontology

## 2.1 Goal and input parameters

This part of the project consists in populating the previously created ontology with instances of `ClassDeclaration`, `MethodDeclaration`, `FieldDeclaration`, `Statements` (including its subclasses) and `FormalParameter`. .
This file takes as input the path of the folder containing the java files that we want to analyze. In this project we used the folder `res/android-chess/app/src/main/java/jwtc/chess/` of the project `android-chess` which has 11 classes in 9 java files.

## 2.2 Description of the code

Before we can start we need to load the ontology created in the previous step. In the first part we used the standard library `os` to list the files in the given directory and filtered the result on the extension to only open java files. We then use the function `parse.parse` of the Javalang Python Library to parse the content into an Abstract Syntax Tree. I decided to save all the trees, checking that are `ClassDeclaration`s into a default dictionary to allow duplicates multiple classes with the same name.
For each class we create a `ClassDeclaration` instance in the ontology with the class name. We then iterate through its methods and create a new instance of `MethodDeclaration` with the method name and add it to the class declaration. We add `FormalParameter` declarations for all the parameters of the method. Its body contains all the statements which we use to create an instance in the ontology taking the statement type for the tree. After the constructor we check the fields and create a `FieldDeclaration` for each. As the Constructor is a particular method, its logic only differers in type of declaration which is `ConstructorDeclaration`.
At last we can save the newly created instances in an owl file.

## 2.3 Results

We found 1344 instances. Table 2 shows statistics for the individuals created for the considered project directory.

| **Class** | # |
|---|---|
| ClassDeclaration | 11 |
| ConstructorDeclaration | 6 |
| MethodDeclaration | 152 |
| FieldDeclaration | 105 |
| FormalParameter | 165 |
| AssertStatement | 0 |
| BlockStatement | 143 |
| BreakStatement | 23 |
| CatchClause | 8 |
| ContinueStatement | 4 |
| DoStatement | 2 |
| ForStatement | 6 |
| IfStatement | 125 |
| ReturnStatement | 106 |
| StatementExpression | 446 |
| SwitchStatement | 8 |
| SynchronizedStatement | 1 |
| ThrowStatement | 15 |
| TryStatement | 8 |
| WhileStatement | 10 |

Table 2: Number of created instances per class

# 3 Find Bad Smell

## 3.1 Goal and input parameters

This last part of the project consists of finding bad smells in the Java classes previously parsed into ontology individuals.
In order to find bad smells we execute SparQL queries in our ontology.
This file takes an optional argument which is the path of the owl file previously created. If the argument is non supplied then a predefined path is used.

## 3.2 Description of the code

Before we can start we need to create a graph to be able to execute the queries. We create a world, load the ontology created in the previous steps and get the graph.
In order to efficiently parse the code smells we created two class named, respectively, `ClassSmell` and `MethodSmell` to store the name of the class, the occurrences and, in the second one, the name of the method. There are five different types of code smells which require five different queries:

- LongMethod and LongConstructors: methods and constructors that have 20 or more statements.
- LargeClass: classes that have 10 or more methods
- MethodWithSwitch, ConstructorWithSwitch: methods and constructors that have switch statements.
- MethodWithLongParameterList, ConstructorWithLongParameterList: methods and constructors that have 5 or more parameters
- DataClass: classes that have only setters and getters.

We have created a function for each type of query: `query_long`, `query_large_class`, `query_with_switch`, `query_with_long_parameter_list`. All these functions require the graph ad input. The functions that query both methods and constructors take as input also the type of query. These query functions are straight forward: in the first lines there is the query string, we run the query, filter the result based on a threshold described above and return an array with instances of `ClassSmell` of `MethodSmell` depending on the type of query. The results of all queries is then saved into a dictionay with the code smell name as a key. The function `prepare_query` is used to initialize the namespace of the queries. We created a function to print the results on the console.

## 3.3 Results

Table 3 shows the number of occurrences of each bad smell found in the code.
We can see that the constructors do not have any bad smells, there is only data class and the main

bad smell is long methods. I created different tables to show the results found for each query: table 4 reports the results of LongMethod, table 5 reports the results of LargeClass, table 6 reports the results of LongMethod, table 7 reports the results of MethodWithLongParameterList, table 8 reports the results of DataClass.

| **Query** | **#** |
|---|---|
| LongMethod | 10 |
| LongConstructor | 0 |
| LargeClass | 3 |
| MethodWithSwitch | 8 |
| ConstructorWithSwitch | 0 |
| MethodWithLongParameterList | 4 |
| ConstructorWithLongParameterList | 0 |
| DataClass | 1 |

Table 3: Bad smells (total)

| **Class** | **Method** | **# of statements** |
|---|---|---|
| PGNProvider | insert | 31 |
| ChessPuzzleProvider | query | 25 |
| | insert | 20 |
| GameControl | loadPGNHead | 26 |
| | loadPGNMoves | 96 |
| | requestMove | 76 |
| | getDate | 26 |
| JNI | newGame | 35 |
| | initFEN | 88 |
| | initRandomFisher | 87 |

Table 4: Long methods

| **Class** | **Method** | **# of parameters** |
|---|---|---|
| PGNProvider | query | 5 |
| ChessPuzzleProvider | query | 5 |
| GameControl | addPGNEntry | 5 |
| JNI | setCastlingsEPAnd50 | 6 |

Table 7: MethodWithLongParameterList query results

| Class | # of methods |
|---|---|
| GameControl | 63 |
| JNI | 44 |
| Move | 21 |

Table 5: Large class

| Class | Method | # of switch |
|---|---|---|
| PGNProvider | query | 1 |
| | getType | 1 |
| | delete | 1 |
| | update | 1 |
| ChessPuzzleProvider | query | 1 |
| | getType | 1 |
| | delete | 1 |
| | update | 1 |

Table 6: Methods with switches

| Class | # of methods |
|---|---|
| Valuation | 1 |

Table 8: Data classes

# A Python code

## A.1 Project

### A.1.1 Create Ontology

```
from sys import argv, exit
from ast import *
from owlready2 import *
from types import new_class


class Class:
    def __init__(self, name, super_classes, properties):
        self.name = name
        self.super_classes = super_classes
        self.properties = properties


def get_classes(python_file_name):
    with open(python_file_name, "r") as python_file:
        return [Class(node.name, [node_base.id for node_base in node.bases], [elt.s for elt
             in node.body[0].value.elts])
                for node in walk(parse(python_file.read())) if type(node) is ClassDef]


def start(python_file_name):
    ontology_file_name = "res/tree.owl"
    ontology_file = get_ontology("http://my.onto.org/tree.owl")
    with ontology_file:
        for current_class in get_classes(python_file_name):
            if len(current_class.super_classes) == 1:
                if current_class.super_classes[0] == "Node":
                    new_class(current_class.name, (Thing,))
                else:
                    new_class(current_class.name, (ontology_file[current_class.super_classes
                        [0]],))
            else:
                new_class(current_class.name, (ontology_file[current_class.super_classes[0]],
                                               ontology_file[current_class.super_classes
                                                   [1]],))

            for class_property in current_class.properties:
                if class_property != "body" and class_property != "parameters":
                    new_class("jname" if class_property == "name" else class_property, (
                        DataProperty,))

        new_class("body", (ObjectProperty,))
        new_class("parameters", (ObjectProperty,))

    ontology_file.save(file=ontology_file_name, format="rdfxml")


if __name__ == "__main__":
    start(argv[1] if len(argv) > 1 else "res/tree.py")
```

### A.1.2 Populate Ontology

```
from collections import defaultdict
from sys import argv, exit
import javalang as jl
import javalang.tree
from owlready2 import *


def start(project_path):
    ontology = populate_ontology(get_ontology("res/tree.owl").load(), get_classesAST(
        project_path))
    ontology.save(file="res/tree2.owl", format="rdfxml")


def get_classesAST(project_path):
    class_declarations = defaultdict()
    for file in os.listdir(project_path):
        if file.endswith(".java"):
            java_file = open(project_path + '/' + file, "r")
            for _, node in jl.parse.parse(java_file.read()):
                if type(node) is jl.tree.ClassDeclaration:
                    class_declarations.setdefault(node.name, []).append(node)
            java_file.close()
    return class_declarations


def populate_ontology(ontology, class_declarations):
    with ontology:
        for class_name, classesAST in class_declarations.items():
            for classAST in classesAST:
                class_declaration = ontology["ClassDeclaration"]()
                class_declaration.jname = [class_name]

                for method in classAST.methods:
                    if type(method) is javalang.tree.MethodDeclaration:
                        declaration = add_new_declaration(method, "Method", class_declaration
                            , ontology)
                        add_other_declarations(method, declaration, ontology)

                for field in classAST.fields:
                    if type(field) is javalang.tree.FieldDeclaration:
                        for decl in field.declarators:
                            add_new_declaration(decl, "Field", class_declaration, ontology)

                for constructor in classAST.constructors:
                    if type(constructor) is javalang.tree.ConstructorDeclaration:
                        declaration = add_new_declaration(constructor, "Constructor",
                            class_declaration, ontology)
                        add_other_declarations(constructor, declaration, ontology)
    return ontology


def add_new_declaration(node, declaration_type, class_declaration, ontology):
    declaration = ontology[declaration_type + "Declaration"]()
    declaration.jname = [node.name]
    class_declaration.body.append(declaration)
```

```
    return declaration


def add_other_declarations(node, declaration, ontology):
    for parameter in node.parameters:
        formal_declaration = ontology["FormalParameter"]()
        formal_declaration.jname = [parameter.name]
        declaration.parameters.append(formal_declaration)

    if node.body is not None:
        for _, statement in node:
            if type(statement).__bases__[0] is javalang.tree.Statement:
                declaration.body.append(ontology[type(statement).__name__]())


if __name__ == "__main__":
    if len(argv) < 2:
        print("Please give as input the path of the java class files to create the ontology")
        exit(1)
    start(argv[1])
```

### A.1.3 Find Bad Smell

```
from sys import argv
import rdflib.plugins.sparql as sq
from owlready2 import *


class ClassSmell:
    def __init__(self, row):
        self.class_name = str(row.class_name)
        self.counter = int(row.counter)


class MethodSmell(ClassSmell):
    def __init__(self, row):
        super().__init__(row)
        self.method_name = str(row.method_name)


def start(owl_path):
    world = World()
    world.get_ontology(owl_path).load()
    graph = world.as_rdflib_graph()
    print_queries(run_queries(graph))


def prepare_query(string):
    return sq.prepareQuery(string, initNs={"tree": "http://my.onto.org/tree.owl#"})


def query_long(query_type, graph):
    # >= 20
    query = f""" SELECT ?class_name ?method_name (COUNT(*) AS ?counter)
        WHERE {{
```

```
        ?c a tree:ClassDeclaration .
        ?c tree:jname ?class_name .
        ?c tree:body ?m .
        ?m a tree:{query_type}Declaration .
        ?m tree:jname ?method_name .
        ?m tree:body ?statements .
        }} GROUP BY ?m"""

    return [MethodSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter
        ) >= 20)]


def query_large_class(graph):
    # >= 10 methods
    query = f""" SELECT ?class_name (COUNT(*) AS ?counter)
        WHERE {{
        ?c a tree:ClassDeclaration .
        ?c tree:jname ?class_name .
        ?c tree:body ?m .
        ?m a tree:MethodDeclaration .
        }} GROUP BY ?c"""

    return [ClassSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter)
         >= 10)]


def query_with_switch(query_type, graph):
    # >= 1 switch statement in method/constructor body
    query = f""" SELECT ?class_name ?method_name (COUNT(*) AS ?counter)
        WHERE {{
        ?c a tree:ClassDeclaration .
        ?c tree:jname ?class_name .
        ?c tree:body ?m .
        ?m a tree:{query_type}Declaration .
        ?m tree:jname ?method_name .
        ?m tree:body ?s .
        ?s a tree:SwitchStatement
        }} GROUP BY ?m"""

    return [MethodSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter
        ) >= 1)]


def query_with_long_parameter_list(query_type, graph):
    # >= 5 parameters
    query = f""" SELECT ?class_name ?method_name (COUNT(*) AS ?counter)
        WHERE {{
        ?c a tree:ClassDeclaration .
        ?c tree:jname ?class_name .
        ?c tree:body ?m .
        ?m a tree:{query_type}Declaration .
        ?m tree:jname ?method_name .
        ?m tree:parameters ?param .
        }} GROUP BY ?m"""

    return [MethodSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter
        ) >= 5)]
```

```
86
87
88  def query_data_class(graph):
89      # class with only setters and getters
90      query0 = f""" SELECT ?class_name (COUNT(*) AS ?counter)
91          WHERE {{
92          ?c a tree:ClassDeclaration .
93          ?c tree:jname ?class_name .
94          ?c tree:body ?m .
95          ?m a tree:MethodDeclaration .
96          }} GROUP BY ?c"""
97
98      query1 = f""" SELECT ?class_name (COUNT(*) as ?counter)
99          WHERE {{ ?c a tree:ClassDeclaration .
100         ?c tree:jname ?class_name .
101         ?c tree:body ?m .
102         ?m a tree:MethodDeclaration .
103         ?m tree:jname ?method_name .
104         FILTER regex(?method_name , "^(get|set)", "i") .
105         }} GROUP BY ?c"""
106
107     large_class = [ClassSmell(row) for row in graph.query(prepare_query(query0)) if row.
            counter]
108     get_and_set = [ClassSmell(row) for row in graph.query(prepare_query(query1)) if row.
            counter]
109     return [method for large in large_class for method in get_and_set
110             if large.class_name == method.class_name and large.counter == method.counter]
111
112
113 def run_queries(graph):
114     return {
115         "LongMethod": query_long("Method", graph),
116         "LongConstructor": query_long("Constructor", graph),
117         "LargeClass": query_large_class(graph),
118         "MethodWithSwitch": query_with_switch("Method", graph),
119         "ConstructorWithSwitch": query_with_switch("Constructor", graph),
120         "MethodWithLongParameterList": query_with_long_parameter_list("Method", graph),
121         "ConstructorWithLongParameterList": query_with_long_parameter_list("Constructor",
                graph),
122         "DataClass": query_data_class(graph)
123     }
124
125
126 def print_queries(queries):
127     for key in queries:
128         if len(queries[key]) == 0:
129             print("No bad smell found for " + key)
130         else:
131             print(key, ":")
132             for element in queries[key]:
133                 string = '\t' + str(element.class_name) + ' '
134                 if type(element) == MethodSmell:
135                     string += str(element.method_name) + ' '
136                 string += str(element.counter)
137                 print(string)
138         print()
139
```

```
140
141 if __name__ == "__main__":
142     start(argv[1] if len(argv) > 1 else "res/tree2.py")
```

## A.2 Tests

### A.2.1 Create Ontology

```
1  import unittest
2  from onto_creator import *
3
4
5  class OntoCreatorTests(unittest.TestCase):
6
7      def __init__(self, *args, **kwargs):
8          super(OntoCreatorTests, self).__init__(*args, **kwargs)
9          self.path_file_python = "res/tree.py"
10         self.path_file_owl = "res/tree.owl"
11
12     def test_00(self):
13         classes = get_classes(self.path_file_python)
14         self.assertEqual(type(classes), type(list()), "Classes should be placed in an array")
15         self.assertEqual(len(classes), 77, "There are missing classes")
16
17     def test_01(self):
18         onto = get_ontology(self.path_file_owl).load()
19         cd = onto["ClassDeclaration"]
20         self.assertEqual(cd.name, "ClassDeclaration", "Should be a ClassDeclaration
               definition")
21         self.assertEqual(len(cd.is_a), 1, "The length of ClassDeclaration should be 1")
22         self.assertEqual(cd.is_a[0].name, "TypeDeclaration", "Should be a TypeDeclaration")
23
24     def test_02(self):
25         onto = get_ontology(self.path_file_owl).load()
26         cd = onto["TypeDeclaration"]
27         self.assertEqual(cd.name, "TypeDeclaration", "Should be a TypeDeclaration definition"
               )
28         self.assertEqual(len(cd.is_a), 2, "The length of TypeDeclaration should be 2")
29         self.assertEqual(cd.is_a[0].name, "Declaration", "Should be a Declaration")
30         self.assertEqual(cd.is_a[1].name, "Documented", "Should be a Documented")
31
32     def test_03(self):
33         onto = get_ontology(self.path_file_owl).load()
34         cd = onto["jname"]
35         self.assertEqual(cd.name, "jname", "Should be a TypeDeclaration definition")
36         self.assertEqual(cd.is_a, [owl.DatatypeProperty], "Should be an DatatypeProperty")
37
38     def test_04(self):
39         onto = get_ontology(self.path_file_owl).load()
40         cd = onto["body"]
41         self.assertEqual(cd.name, "body", "Should be a TypeDeclaration definition")
42         self.assertEqual(cd.is_a, [owl.ObjectProperty], "Should be an ObjectProperty")
43
44
45 unittest.main()
```

### A.2.2 Populate Ontology

```
import unittest
from individ_creator import *
from owlready2 import destroy_entity


class IndividCreatorTests(unittest.TestCase):

    def __init__(self, *args, **kwargs):
        super(IndividCreatorTests, self).__init__(*args, **kwargs)
        self.path_file_owl = "res/tree.owl"
        self.path_project = "res/android-chess/app/src/main/java/jwtc/chess/"

    def create_ontology(self, code):
        classes = defaultdict()
        for _, node in jl.parse.parse(code):
            if type(node) is jl.tree.ClassDeclaration:
                classes.setdefault(node.name, []).append(node)
        return populate_ontology(get_ontology(self.path_file_owl).load(), classes)

    def delete_ontology(self, onto):
        for e in onto["ClassDeclaration"].instances():
            destroy_entity(e)

    def test_10(self):
        classes = get_classesAST(self.path_project)
        self.assertEqual(type(classes), type(defaultdict()), "The Classes should be placed in
             a dictionary")
        self.assertEqual(len(classes), 10, "There are missing classes")

    def test_11(self):
        code = "class A { int x, y; }"
        ontology = self.create_ontology(code)
        instance = ontology['ClassDeclaration'].instances()[0]
        self.assertEqual(instance.body[0].is_a[0].name, "FieldDeclaration", "Should be a
            FieldDeclaration definition")
        self.assertEqual(instance.body[0].jname[0], 'x', "jname should be equal to x")
        self.assertEqual(instance.body[1].is_a[0].name, "FieldDeclaration", "Should be a
            FieldDeclaration definition")
        self.assertEqual(instance.body[1].jname[0], 'y', "jname should be equal to y")
        self.delete_ontology(ontology)

    def test_12(self):
        code = "class A { int x, y; public A() { } public int getX() { return x;} }"
        ontology = self.create_ontology(code)
        instance = ontology['ClassDeclaration'].instances()[0]
        self.assertEqual(instance.body[0].is_a[0].name, "MethodDeclaration", "Should be a
            MethodDeclaration definition")
        self.assertEqual(instance.body[0].jname[0], 'getX', "jname should be equal to getX")
        self.assertEqual(instance.body[1].is_a[0].name, "FieldDeclaration", "Should be a
            FieldDeclaration definition")
        self.assertEqual(instance.body[1].jname[0], 'x', "jname should be equal to x")
        self.assertEqual(instance.body[2].is_a[0].name, "FieldDeclaration", "Should be a
            FieldDeclaration definition")
        self.assertEqual(instance.body[2].jname[0], 'y', "jname should be equal to y")
        self.assertEqual(instance.body[3].is_a[0].name, "ConstructorDeclaration",
```

```
                        "Should be a MethodDeclaration definition")
        self.assertEqual(instance.body[3].jname[0], 'A', "jname should be equal to A")
        self.delete_ontology(ontology)

    def test_13(self):
        code = "class A { int f(int x, int y) { return 0; } }"
        ontology = self.create_ontology(code)
        instance = ontology['ClassDeclaration'].instances()[0]
        self.assertEqual(instance.body[0].is_a[0].name, "MethodDeclaration", "Should be a
            MethodDeclaration definition")
        self.assertEqual(instance.body[0].jname[0], 'f', "jname should be equal to f")
        self.assertEqual(instance.body[0].parameters[0].jname[0], 'x', "jname should be equal
             to x")
        self.assertEqual(instance.body[0].parameters[1].jname[0], 'y', "jname should be equal
             to y")
        self.assertEqual(instance.body[0].body[0].is_a[0].name, 'ReturnStatement',
                         "name should be equal to ReturnStatement")
        self.delete_ontology(ontology)


unittest.main()
```

### A.2.3 Find Bad Smell

```
import unittest

import rdflib
from bad_smells import *
from individ_creator import *
from owlready2 import destroy_entity


class BadSmellsTests(unittest.TestCase):

    def __init__(self, *args, **kwargs):
        super(BadSmellsTests, self).__init__(*args, **kwargs)
        self.path_file_owl = "res/tree.owl"

    def create_ontology(self, code):
        classes = defaultdict()
        for _, node in jl.parse.parse(code):
            if type(node) is jl.tree.ClassDeclaration:
                classes.setdefault(node.name, []).append(node)
        return populate_ontology(get_ontology(self.path_file_owl).load(), classes)

    def get_graph(self, ontology):
        ontology.save(file="res/test3.owl", format="rdfxml")
        graph = rdflib.Graph()
        graph.load("res/test3.owl")
        return graph

    def delete_ontology(self, onto):
        for e in onto["ClassDeclaration"].instances():
            destroy_entity(e)

```

```
32      def test31(self):
33          code = "class A { int f(int x) {  x++;x++;x++;x++;x++;x++;x++;x++;x++;x++;x++;x
                ++;x++;x++;x++;x++;x++;" \
34                 "x++;x++;x++; return x; } }"
35          ontology = self.create_ontology(code)
36          graph = self.get_graph(ontology)
37          self.assertEqual(len(query_long("Method", graph)), 1)
38          self.delete_ontology(ontology)
39
40      def test32(self):
41          code = "class A { public A(int x) {  x++;x++;x++;x++;x++;x++;x++;x++;x++;x++;x++;
                x++;x++;x++;x++;x++;" \
42                 "x++;x++;x++;x++; }}"
43          ontology = self.create_ontology(code)
44          graph = self.get_graph(ontology)
45          self.assertEqual(len(query_long("Constructor", graph)), 1)
46          self.delete_ontology(ontology)
47
48      def test33(self):
49          code = "class A { void a(){} void b(){} void c(){} int d() {return 1;} void e(){}
                void f() {} void g(){}" \
50                 " void h(){} void i(){} void l(){} }"
51          ontology = self.create_ontology(code)
52          graph = self.get_graph(ontology)
53          self.assertEqual(len(query_large_class(graph)), 1)
54          self.delete_ontology(ontology)
55
56      def test34(self):
57          code = "class A { void a(){ int i = 0; switch(i){ case 1: System.out.println(); break
                ; " \
58                 "case 2: System.out.println(); break; default: System.out.println(); } } }"
59          ontology = self.create_ontology(code)
60          graph = self.get_graph(ontology)
61          self.assertEqual(len(query_with_switch("Method", graph)), 1)
62          self.delete_ontology(ontology)
63
64      def test35(self):
65          code = "class A { public A() { int i = 0; switch(i){ case 1: System.out.println();
                break;" \
66                 "case 2: System.out.println(); break; default: System.out.println(); } } }"
67          ontology = self.create_ontology(code)
68          graph = self.get_graph(ontology)
69          self.assertEqual(len(query_with_switch("Constructor", graph)), 1)
70          self.delete_ontology(ontology)
71
72      def test36(self):
73          code = "class A { void a(int x, int y, int z, String args1, String args2){ } " \
74                 "int b(int x, int y, int z, String args1, String args2){ } }"
75          ontology = self.create_ontology(code)
76          graph = self.get_graph(ontology)
77          self.assertEqual(len(query_with_long_parameter_list("Method", graph)), 2)
78          self.delete_ontology(ontology)
79
80      def test37(self):
81          code = "class A { public A(int x, int y, int z, String args1, String args2) { } }"
82          ontology = self.create_ontology(code)
83          graph = self.get_graph(ontology)
```

```
84          self.assertEqual(len(query_with_long_parameter_list("Constructor", graph)), 1)
85          self.delete_ontology(ontology)
86
87      def test38(self):
88          code = "class A { private int x = 0; public int getX() { return x; } public void setX
                (int x) {this.x = x;} }"
89          ontology = self.create_ontology(code)
90          graph = self.get_graph(ontology)
91          self.assertEqual(len(query_data_class(graph)), 1)
92          self.delete_ontology(ontology)
93
94
95
96
97
98
99  unittest.main()
```

# B Bash Code

## B.1 Run Project

```
1  #!/bin/bash
2
3  python3 src/onto_creator/onto_creator.py res/tree.py
4
5  python3 src/individ_creator/individ_creator.py res/android-chess/app/src/main/java/jwtc/chess
       /
6
7  python3 src/bad_smells/bad_smells.py res/tree2.owl > res/bad_smells.txt
```

## B.2 Test Project

```
1  python3 src/onto_creator/onto_creator_tests.py
2  python3 src/individ_creator/individ_creator_tests.py
3  python3 src/bad_smells/bad_smells_tests.py
4  rm res/test3.owl
```