Project 1: Bad Smell Detection

Ardigò Susanna

November 5, 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 I used the file `tree.py` of the Javalang Python Library.

1.2 Parsing Classes

In order to efficiently parse this file I created a class named `Class` to store the name, superclass and property of each class. The function `get_classes(python_file_name)` reads the given file, parses into an Abstract Syntax Tree. I 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. I then iterate the list of the parsed classes, previously explained, to create create in in the ontology. This step needs to differentiate among three different contruction creations depending on the number of superclasses. If the current class has none, meaning that it has no super class, it is created as a subclass of Thing which, in owl, is the top superclass. 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 I 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, I decided to add them only once at the end. To avoid conflict, when I create "name" properties I rename them to "jname".
The ontology is finally created and I can export it into an owl file.

1.3 Results

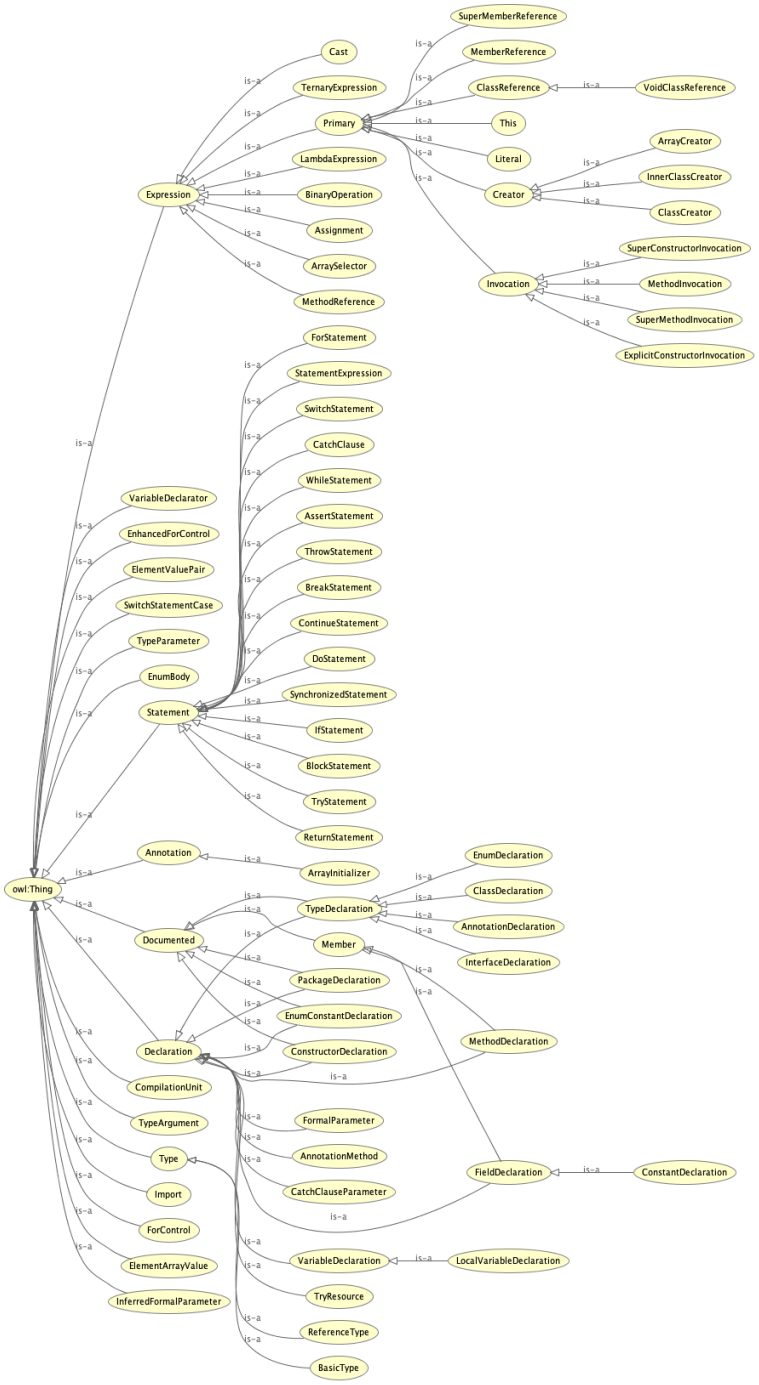

Figure 1: Class Hierarchy of the Ontology created

2 Populate the Ontology

2.1 Results

3 Find Bad Smell

3.1 Results

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 {{
```

```
33          ?c a tree:ClassDeclaration .
34          ?c tree:jname ?class_name .
35          ?c tree:body ?m .
36          ?m a tree:{query_type}Declaration .
37          ?m tree:jname ?method_name .
38          ?m tree:body ?statements .
39          }} GROUP BY ?m"""
40
41      return [MethodSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter
            ) >= 20)]
42
43
44  def query_large_class(graph):
45      # >= 10 methods
46      query = f""" SELECT ?class_name (COUNT(*) AS ?counter)
47          WHERE {{
48          ?c a tree:ClassDeclaration .
49          ?c tree:jname ?class_name .
50          ?c tree:body ?m .
51          ?m a tree:MethodDeclaration .
52          }} GROUP BY ?c"""
53
54      return [ClassSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter)
             >= 10)]
55
56
57  def query_with_switch(query_type, graph):
58      # >= 1 switch statement in method/constructor body
59      query = f""" SELECT ?class_name ?method_name (COUNT(*) AS ?counter)
60          WHERE {{
61          ?c a tree:ClassDeclaration .
62          ?c tree:jname ?class_name .
63          ?c tree:body ?m .
64          ?m a tree:{query_type}Declaration .
65          ?m tree:jname ?method_name .
66          ?m tree:body ?s .
67          ?s a tree:SwitchStatement
68          }} GROUP BY ?m"""
69
70      return [MethodSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter
            ) >= 1)]
71
72
73  def query_with_long_parameter_list(query_type, graph):
74      # >= 5 parameters
75      query = f""" SELECT ?class_name ?method_name (COUNT(*) AS ?counter)
76          WHERE {{
77          ?c a tree:ClassDeclaration .
78          ?c tree:jname ?class_name .
79          ?c tree:body ?m .
80          ?m a tree:{query_type}Declaration .
81          ?m tree:jname ?method_name .
82          ?m tree:parameters ?param .
83          }} GROUP BY ?m"""
84
85      return [MethodSmell(row) for row in graph.query(prepare_query(query)) if (int(row.counter
            ) >= 5)]
```

```
86
87
88  def query_constructor_with_long_parameter_list(graph):
89      # >= 5 parameters
90      return "TODO"
91
92
93  def query_data_class(graph):
94      # class with only setters and getters
95      query0 = f""" SELECT ?class_name (COUNT(*) AS ?counter)
96          WHERE {{
97          ?c a tree:ClassDeclaration .
98          ?c tree:jname ?class_name .
99          ?c tree:body ?m .
100         ?m a tree:MethodDeclaration .
101         }} GROUP BY ?c"""
102
103     query1 = f""" SELECT ?class_name (COUNT(*) as ?counter)
104         WHERE {{ ?c a tree:ClassDeclaration .
105         ?c tree:jname ?class_name .
106         ?c tree:body ?m .
107         ?m a tree:MethodDeclaration .
108         ?m tree:jname ?method_name .
109         FILTER regex(?method_name , "^(get|set)", "i") .
110         }} GROUP BY ?c"""
111
112     large_class = [ClassSmell(row) for row in graph.query(prepare_query(query0)) if row.
            counter]
113     get_and_set = [ClassSmell(row) for row in graph.query(prepare_query(query1)) if row.
            counter]
114     return [method for large in large_class for method in get_and_set
115             if large.class_name == method.class_name and large.counter == method.counter]
116
117
118 def run_queries(graph):
119     return {
120         "LongMethod": query_long("Method", graph),
121         "LongConstructor": query_long("Constructor", graph),
122         "LargeClass": query_large_class(graph),
123         "MethodWithSwitch": query_with_switch("Method", graph),
124         "ConstructorWithSwitch": query_with_switch("Constructor", graph),
125         "MethodWithLongParameterList": query_with_long_parameter_list("Method", graph),
126         "ConstructorWithLongParameterList": query_with_long_parameter_list("Constructor",
                graph),
127         "DataClass": query_data_class(graph)
128     }
129
130
131 def print_queries(queries):
132     for key in queries:
133         if len(queries[key]) == 0:
134             print("No bad smell found for " + key)
135         else:
136             print(key, ":")
137             for element in queries[key]:
138                 string = '\t' + str(element.class_name) + ' '
139                 if type(element) == MethodSmell:
```

```
                    string += str(element.method_name) + ' '
                string += str(element.counter)
                print(string)

        print()


if __name__ == "__main__":
    if len(argv) < 2:
        print("Please give as input the path of the owl file to create find bad smells")
        exit(1)
    start(argv[1])
```

A.2 Tests

A.2.1 Create Ontology

```
import unittest
from onto_creator import *


class OntoCreatorTests(unittest.TestCase):

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

    def test_00(self):
        classes = get_classes(self.path_file_python)
        self.assertEqual(type(classes), type(list()), "Classes should be placed in an array")
        self.assertEqual(len(classes), 77, "There are missing classes")

    def test_01(self):
        onto = get_ontology(self.path_file_owl).load()
        cd = onto["ClassDeclaration"]
        self.assertEqual(cd.name, "ClassDeclaration", "Should be a ClassDeclaration
            definition")
        self.assertEqual(len(cd.is_a), 1, "The length of ClassDeclaration should be 1")
        self.assertEqual(cd.is_a[0].name, "TypeDeclaration", "Should be a TypeDeclaration")

    def test_02(self):
        onto = get_ontology(self.path_file_owl).load()
        cd = onto["TypeDeclaration"]
        self.assertEqual(cd.name, "TypeDeclaration", "Should be a TypeDeclaration definition"
            )
        self.assertEqual(len(cd.is_a), 2, "The length of TypeDeclaration should be 2")
        self.assertEqual(cd.is_a[0].name, "Declaration", "Should be a Declaration")
        self.assertEqual(cd.is_a[1].name, "Documented", "Should be a Documented")

    def test_03(self):
        onto = get_ontology(self.path_file_owl).load()
        cd = onto["jname"]
        self.assertEqual(cd.name, "jname", "Should be a TypeDeclaration definition")
        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

```
1   import unittest
2   from individ_creator import *
3   from owlready2 import destroy_entity
4
5
6   class IndividCreatorTests(unittest.TestCase):
7
8       def __init__(self, *args, **kwargs):
9           super(IndividCreatorTests, self).__init__(*args, **kwargs)
10          self.path_file_owl = "res/tree.owl"
11          self.path_project = "res/android-chess/app/src/main/java/jwtc/chess/"
12
13      def create_ontology(self, code):
14          classes = defaultdict()
15          for _, node in jl.parse.parse(code):
16              if type(node) is jl.tree.ClassDeclaration:
17                  classes.setdefault(node.name, []).append(node)
18          return populate_ontology(get_ontology(self.path_file_owl).load(), classes)
19
20      def delete_ontology(self, onto):
21          for e in onto["ClassDeclaration"].instances():
22              destroy_entity(e)
23
24      def test_10(self):
25          classes = get_classesAST(self.path_project)
26          self.assertEqual(type(classes), type(defaultdict()), "The Classes should be placed in
                a dictionary")
27          self.assertEqual(len(classes), 10, "There are missing classes")
28
29      def test_11(self):
30          code = "class A { int x, y; }"
31          ontology = self.create_ontology(code)
32          instance = ontology['ClassDeclaration'].instances()[0]
33          self.assertEqual(instance.body[0].is_a[0].name, "FieldDeclaration", "Should be a
                FieldDeclaration definition")
34          self.assertEqual(instance.body[0].jname[0], 'x', "jname should be equal to x")
35          self.assertEqual(instance.body[1].is_a[0].name, "FieldDeclaration", "Should be a
                FieldDeclaration definition")
36          self.assertEqual(instance.body[1].jname[0], 'y', "jname should be equal to y")
37          self.delete_ontology(ontology)
38
39      def test_12(self):
40          code = "class A { int x, y; public A() { } public int getX() { return x;} }"
```

```
41          ontology = self.create_ontology(code)
42          instance = ontology['ClassDeclaration'].instances()[0]
43          self.assertEqual(instance.body[0].is_a[0].name, "MethodDeclaration", "Should be a
                MethodDeclaration definition")
44          self.assertEqual(instance.body[0].jname[0], 'getX', "jname should be equal to getX")
45          self.assertEqual(instance.body[1].is_a[0].name, "FieldDeclaration", "Should be a
                FieldDeclaration definition")
46          self.assertEqual(instance.body[1].jname[0], 'x', "jname should be equal to x")
47          self.assertEqual(instance.body[2].is_a[0].name, "FieldDeclaration", "Should be a
                FieldDeclaration definition")
48          self.assertEqual(instance.body[2].jname[0], 'y', "jname should be equal to y")
49          self.assertEqual(instance.body[3].is_a[0].name, "ConstructorDeclaration",
50                           "Should be a MethodDeclaration definition")
51          self.assertEqual(instance.body[3].jname[0], 'A', "jname should be equal to A")
52          self.delete_ontology(ontology)
53
54      def test_13(self):
55          code = "class A { int f(int x, int y) { return 0; } }"
56          ontology = self.create_ontology(code)
57          instance = ontology['ClassDeclaration'].instances()[0]
58          self.assertEqual(instance.body[0].is_a[0].name, "MethodDeclaration", "Should be a
                MethodDeclaration definition")
59          self.assertEqual(instance.body[0].jname[0], 'f', "jname should be equal to f")
60          self.assertEqual(instance.body[0].parameters[0].jname[0], 'x', "jname should be equal
                 to x")
61          self.assertEqual(instance.body[0].parameters[1].jname[0], 'y', "jname should be equal
                 to y")
62          self.assertEqual(instance.body[0].body[0].is_a[0].name, 'ReturnStatement',
63                           "name should be equal to ReturnStatement")
64          self.delete_ontology(ontology)
65
66
67  unittest.main()
```

A.2.3 Find Bad Smell

```
1   import unittest
2
3   import rdflib
4   from bad_smells import *
5   from individ_creator import *
6   from owlready2 import destroy_entity
7
8
9   class BadSmellsTests(unittest.TestCase):
10
11      def __init__(self, *args, **kwargs):
12          super(BadSmellsTests, self).__init__(*args, **kwargs)
13          self.path_file_owl = "res/tree.owl"
14
15      def create_ontology(self, code):
16          classes = defaultdict()
17          for _, node in jl.parse.parse(code):
18              if type(node) is jl.tree.ClassDeclaration:
19                  classes.setdefault(node.name, []).append(node)
```

```
20          return populate_ontology(get_ontology(self.path_file_owl).load(), classes)
21
22      def get_graph(self, ontology):
23          ontology.save(file="res/test3.owl", format="rdfxml")
24          graph = rdflib.Graph()
25          graph.load("res/test3.owl")
26          return graph
27
28      def delete_ontology(self, onto):
29          for e in onto["ClassDeclaration"].instances():
30              destroy_entity(e)
31
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++;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++;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
```