Remote repository

https://github.com/heyerique/Refactoring

# Smell Detection

## Bad Smell 1: Feature Envy

**Name: Feature Envy**

**Location:**

<controller.py>-<Controller>-<do\_import>-<between line 123 and 158>

**Reasons:**

1. The method is used for importing data and displaying result from the specific resource type by the keywords “-csv” and “-pk” from user input. However, the code for displaying import results uses many methods from View class.
2. The symptom matches the code smell of Feature Envy

**Strategies/ approaches:**

Move method

## Bad Smell 2: Primitive Obsession

**Name: Primitive Obsession**

**Location:**

<staff\_data.py>-<StaffData>-<get\_gender>-<between Line 101 and 121>

**Reasons:**

1. A data type of dictionary is used as return value of the method

2. The field names of the dictionary are string values

3. Above symptoms match the code smell of Primitive Obsession

**Strategies/ approaches:**

Replace data value with object

## Bad Smell 3: Long Method

**Name: Long Method**

**Location:**

<controller.py>-<Controller>-<do\_select>-<between Line 27 and 80>

**Reasons:**

1. The method process two functions which are processing user input and processing the keyword “-csv” by using several conditional statements.
2. The above symptoms indicate the method has two functions, which matches the symptom of the code smell of Long Method

**Strategies/ approaches:**

Extract method

**Refactoring steps:**

1. Create a new method in the same class for processing user input
2. Test the code to ensure no faults
3. Replace the code of processing user input in do\_select with the new method
4. Test the code to ensure no faults and the functionality doesn’t change
5. Create another method in the same class for processing the keyword “-csv”
6. Test the code to ensure no faults
7. Replace the code of processing the keyword “-csv” in do\_select with the new method.
8. Test the code to ensure no faults and the functionality doesn’t change
9. Inspect the code to assure no new code smells introduced.

## Bad Smell 4: Shotgun Surgery

**Name: Duplicated Code**

**Location:**

<data\_validator.py>-<DataValidator>-<between Line 17 and 154>

**Reasons:**

1. All the methods in the class have the code of matching regular expressions

2. If the change on the way to find the match, the change will be applied on all the methods in the class

**Strategies/ approaches:**

Extract method

# Testing before Refactoring

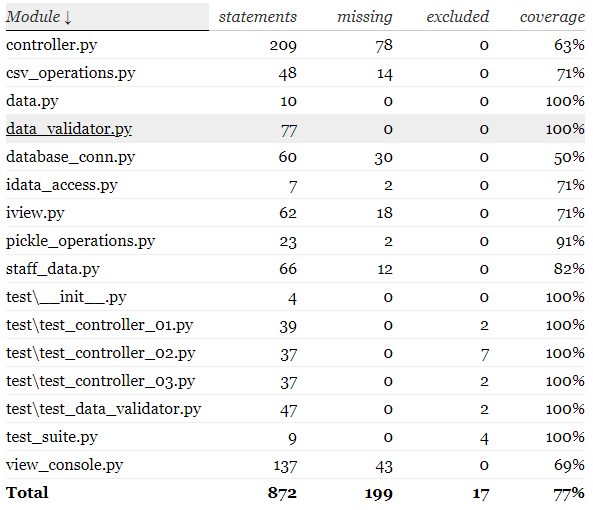
## Unit Test

A set of tests for the methods or classes encompassed by the bad smells previously identified has been carried out by unit test.

Test cases are created under the folder “test”

## Coverage

A test suite is created for running branch coverage test. The overall coverage of the project code is 77%. The branch coverage of the methods or classes, which are encompassed by the bad smells previously identified, is 100%.



The coverage test result HTML file can be found in the folder “htmlcov\_before”

## Code Inspection with PEP8

The code of the project has passed the PEP8 check.

# Refactoring

## Prioritising

The prioritising of the identified bad smells based on the rule that is the number of part that the smell affects in the program. The bad smells are listed below to compare against each other and they are prioritised with numbers from 1 to 4, which stand for the orders of refactoring.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Smell** | **Affected Methods** | **Affected Class** | **Priority** |
| Bad Smell 1 | do\_import | Controller, ViewConsole | 2 |
| Bad Smell 2 | get\_gender,  show\_pie,  show\_bar | StaffData, Controller | 1 |
| Bad Smell 3 | do\_select | Controller | 4 |
| Bad Smell 4 | check\_empid,  check\_gender,  check\_age,  check\_sales,  check\_bmi,  check\_salary,  check\_birthday | DataValidator | 3 |

## Refactoring

### Refactor Bad Smell 2: Primitive Obsession

#### Steps

1. Create class Counter and GenderCounter
2. Test the code and make sure it works
3. Create a new method get\_gender2 to achieve the same functionality of the original method get\_gender
4. Test the code to ensure the method get\_gender2 works and returns an object with correct values
5. Find one the place where get\_gender is called and modify the code for the new method get\_gender2
6. Test the code to ensure no faults and the functionality doesn’t change
7. Keep doing step 5 to 6 until all place of calling get\_gender is updated for using get\_gender2
8. Test the code to ensure no faults and the functionality doesn’t change
9. Remove the original method get\_gender, rename the new method get\_gender2 to get\_gender, replace all the places where get\_gender2 is called to get\_gender.
10. Test the code to ensure no faults and the functionality doesn’t change
11. Inspect the code to assure no new code smells introduced.
12. Pass the PEP8 check

#### Evaluation

Before the refactoring, the array was using strings as keys to specify each item, it is hard for other developers to use the array somewhere else, because it is difficult to remember the strings and easily make the fault in the program if wrong strings are used.

After the refactoring, the string keys have been substituted by properties in an object which is easy to maintain (maintain only one object class) and be used without remembering them.

### Reprioritise the identified code smells

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Smell** | **Affected Methods** | **Affected Class** | **Priority** |
| Bad Smell 1 | do\_import | Controller, ViewConsole | 1 |
| Bad Smell 3 | do\_select | Controller | 3 |
| Bad Smell 4 | check\_empid,  check\_gender,  check\_age,  check\_sales,  check\_bmi,  check\_salary,  check\_birthday | DataValidator | 2 |

### Refactor Bad Smell 1: Feature Envy

#### Steps

1. Create a method import\_result in class Controller and move the code under one conditional statement which has the bad smell to the method.
2. Test the code to ensure no faults and the functionality doesn’t change
3. Create a list to store all data returned by the method import\_row
4. Test the code to ensure no fault exists.
5. Create a method display\_import\_result in ViewConsole to achieve the same functionality as the method import\_result
6. Test the code to ensure no fault exists.
7. Replace the code of the method\_result by invoking the method display\_import\_result
8. Test the code to ensure no faults and the functionality doesn’t change
9. Replace the same code in the second conditional statement which is used for pickle which the new method import\_result
10. Test the code to ensure no faults and the functionality doesn’t change
11. Inspect the code to assure no new code smells introduced (Identified a new one, see Section 3.2.8)
12. Change variable name args to commands
13. Test the code to ensure no faults and the functionality doesn’t change
14. Remove unnecessary exception handling code
15. Test the code to ensure no faults and the functionality doesn’t change
16. Move the code under the conditional statements to a new method import\_csv and import\_pickle in the same class
17. Test the code to ensure no faults and the functionality doesn’t change
18. Pass the PEP8 check

#### Evaluate the refactored code

A new bad smell Primitive Obsession is identified in after the removal of the bad smell 1.

#### Bad Smell 1.1: Primitive Obsession

**Name: Primitive Obsession**

**Location:**

<controller.py>-<Controller>-<do\_import>-<between line 108 and 125>

**Reasons:**

1. A list is used for storing the commands which is input by the user

**Strategies/ approaches:**

Replace the list with an object

**Refactoring Steps:**

1. Create a new class ImportCommand for processing and storing the user command
2. Test the code to ensure no faults
3. Update the method do\_import by using the new class ImportCommand
4. Test the code to ensure no faults and the functionality doesn’t change
5. Reorder the conditional statements
6. Test the code to ensure no faults and the functionality doesn’t change
7. Inspect the code to assure no new code smells introduced.
8. Pass the PEP8 check

#### Evaluation

Before refactoring, the method used a lot of features from the class View which made the method very long and hard to understand the purpose of them.

After the refactoring, new bad smells are identified, therefore, several rounds of refactoring have been done to them. Compares to the code before the refactoring, the code is much more clearer and easy to understand, because functions have been extracted to separate methods, functions which are using many methods from the class View, have been extracted and keep them resided in the View class, meanwhile in terms of succinctness, code lines have been reduced significantly in the Controller class, and duplicated code is removed as well as a consequence of the refactoring.

### Reprioritise the identified code smells

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Smell** | **Affected Methods** | **Affected Class** | **Priority** |
| Bad Smell 3 | do\_select | Controller | 2 |
| Bad Smell 4 | check\_empid,  check\_gender,  check\_age,  check\_sales,  check\_bmi,  check\_salary,  check\_birthday | DataValidator | 1 |

### Refactor Bad Smell 4: Shotgun Surgery

#### Steps

1. Extract the code in a method, which is used for finding matches with a regular expression, to new methods regx\_match and regx\_match\_group
2. Test the code to ensure no faults and the functionality doesn’t change
3. Repeat step 1 – 2 until all code smells are gone
4. Inspect the code to assure no new code smells introduced.
5. Pass the PEP8 check

#### Evaluation

Before the refactoring, if a change need to be made on matching a string, then the change must be applied to every relevant methods in the class, which is time consuming and the program code is inflexible.

After the refactoring, the code for matching a string with a regular expression has been extracted from each method in the class. The code becomes more flexible that the modification applied to one place (new method) instead of changing all relevant methods.

### Refactor Bad Smell 3: Long Class

#### Steps

1. Create a new class SelectCommand for processing and storing the user input command.
2. Test the code to ensure no faults
3. Update the method do\_select by using the new class SelectCommand
4. Test the code to ensure no faults and the functionality doesn’t change
5. Move the code used for processing CSV files in the method do\_select to a new method select\_csv in the same class
6. Test the code to ensure no faults and the functionality doesn’t change
7. Remove unnecessary exception handling code
8. Test the code to ensure no faults and the functionality doesn’t change
9. Remove unnecessary conditional statements
10. Inspect the code to assure no new code smells introduced.
11. Pass the PEP8 check

#### Evaluation

Before the refactoring, the method is long, hard to read and understand, because it mingled with function of importing data from CSV files and pickle files.

After the refactoring, functions are extracted to separate methods with proper naming conventions. The method code lines have been significantly reduced, it is much easier to read and understand the code, and each extracted code has one function which can be identified by their method’s name.

# Testing after Refactoring

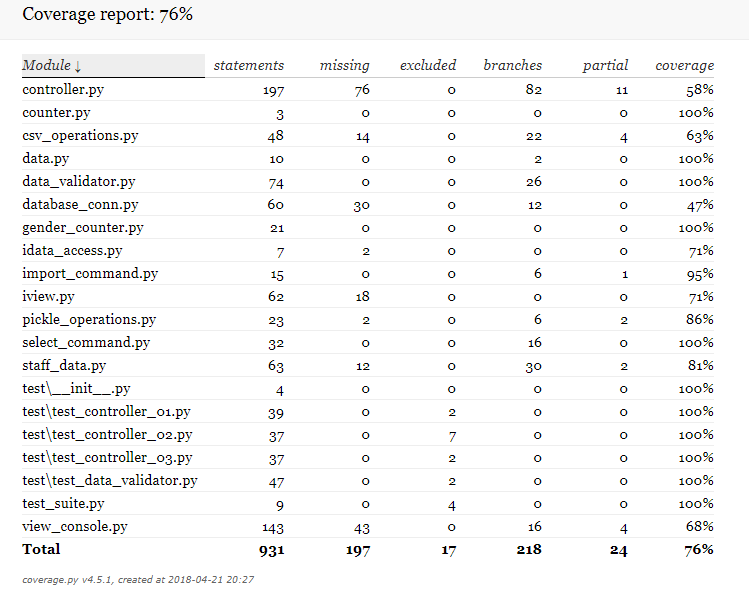
## Unit Test

The same tests for the methods or classes encompassed by the bad smells previously identified have been updated and were carried out by unit test.

Test cases are created under the folder “test”

## Coverage

A test suite is created for running branch coverage test. The overall coverage of the project code is 76%. The branch coverage of the methods or classes, which are encompassed by the bad smells previously identified, is nearly 100%.



The coverage test result HTML file can be found in the folder “htmlcov\_after”

## Code Inspection with PEP8

The code of the project has passed the PEP8 check.