# Smell Detection

#### 1. Name: duplicated code

Location: validator.py in some check methods.

Reasons:

* Check\_Id, age, gender, sales, bmi, salary methods has similar rules for formats, the code in these functions are redundant.
* The function is repeating the same code except it calls different rule name.

Strategy/Approach

* -If the same code is found in two or more methods in the same class: use Extract Method and place calls for the new method in both places.
* Use Extract Method, followed by Pull Up Field for the fields used in the method that needs pulling up: Create a new method and copy the relevant code fragment. Delete the fragment from its old location and put a call for the new method there instead. Define a new function of calling several functions instead of calling them repeatedly in different sub-classes.

#### 2. Name: Switch statement

Location: file\_handler.py in the open method

Reasons:

* Too many if and elif statement in the method makes it hard to add or edit another condition. Need to check the existing condition again every time adding/edit new condition.

Strategy/Approach

In filehandler class add dictionary of file types.

Replace Conditional with Polymorphism: make subclass of each file type and choose the relevant subclass based on file extension.

First define the file\_types by file’s extension in FileHandler and make them in a dictionary

And then in open method delete the if elif statement

Create an instance “extension” and store the file extension by splitting the filepath by “.” And then set the last ones as extension.

Create FileReader abstract class. And implemented by subclasses CSVReader, TXTTeader and XLSXReader.

#### 3. Name: large class

Location: validator.py in the Validator class

Reasons:

1. Validator class has too many methods inside and the code inside is redundant.

Strategy/Approach

-Split up the responsibilities of the class, Create a new class to contain the relevant functionality.,

-Extract method to make a subclass. In this case: extract check method and combine them together in a subclass. Call check methods in sub class, delete the method in the old class.

#### 4. Name: Temporary Field

Location: validator.py, in the check\_birthday methods

Reasons:

1. The temporary field only get their values when need to check birthday, other times it is no use.
2. They waste space in memory.
3. Oftentimes, temporary fields are created for use in an algorithm that requires a large amount of inputs. So instead of creating a large number of parameters in the method, the programmer decides to create fields for this data in the class. These fields are used only in the algorithm and go unused the rest of the time.

Strategy/Approach

Integrate it in place where needed instead of calling the temporary field.

THEN (after testing)

Remove the temporary field.

# Refactoring

## 1. Switch statement

Location: file\_handler.py FileHandler.open method.

### Why it's the worst:

The if and elif statement made the code hard to read, it is also hard to extend for future file type, we have to edit the elif statement every time we want to add an new file type. The statement is not closed for modify, we can add countless elif statements easily.

### Testing

Create refactoring\_test.py to test file\_reader functionality.

Use console to print out extension to see it is correct.

### Version Control

Github for version control to record of changing process.

### Evaluation

After refactoring the new method satisfy the open-closed principle. It can extend its file type by adding more sub-class to file\_reader abstract class. It is easy to test and it is easy to read.

### *2. Temporary Field*

Location: validator.py Validator.check\_birthday and check\_birthday\_against\_age method

### Why it's the worst:

It holds up memory space and it may slow down the computer. Code has lower readability.

### Testing

### Test\_validator.py unit tests are used for testing the functionality of new check\_birthday and check\_birthday\_against\_age in Validator.py.

### Version Control

Github for version control to record of changing process.

### Evaluation

It saves some line of code after deleting the holder for day, month and year. Check birthday method is now simpler.

## 3. Duplicate Code

Location: validator.py Validator.checkxxx methods.

### Why it's the worst:

If multiple places have the same code, every time a change made it needs to change these code in multiple places. It increase the work load and it makes code hard to read and maintain.

### Testing

Test\_validator.py unit tests are used for testing the functionality of check\_hud in Validator.py.

### Version Control

Github for version control to record of changing process.

### Evaluation

The new function check\_hud reduce a lot of duplicated code in the validator, check\_id, check\_age, check\_gender, check\_sales, check\_bmi function now calls check\_hud as a general solution. It is easier to maintain. If any change happens to validate rules, now by changing check\_hud method, I can change check\_id, check\_age, check\_gender, check\_sales, check\_bmi function at once. However, there are new duplicated code inside check methods created.

## 4. Switch statement

Location: validator.py Validator.check\_all method.

### Why it's the worst:

The if and elif statement made the code hard to read, it is also hard to extend for future file type, we have to edit the elif statement every time we want to add an new file type. The statement is not closed for modify, we can add countless elif statements easily.

### Testing

Use doctest to test the functionality of the new check\_all method

### Version Control

Github for version control to record of changing process.

### Evaluation

Modif check\_all method and remove the if statement, make it simpler. Better extensibility. Easy to add more check rules for future develop. After refactoring the new method satisfy the open-closed principle.