# Bad Smells

## 1) Middle Man

### Name:

Middle Man

### Location:

Interpreter - controller.py - Controller - load(), validate(), set\_local(), set\_remote(), set\_graph(), get\_local(), get\_remote(), set\_criteria(), set\_keys(), draw(), check\_data()  
- Between line 7 and 76

### Reasons:

1. Impacts code readability
2. No function other than passing information
3. More difficult to test and maintain
4. Increases overall code size unnecessarily

### Strategies:

1. Remove Middle Man
   1. Create a ‘getter’ for accessing the delegate class objects from the server class
   2. Replace calls to the delegate methods in the server class with direct calls to the required delegate functions

Remove the calls from the Shell() class to the Controller() class and replace them with calls to the delegate object needed

## 2) Switch Statements

### Name:

Switch Statements

### Location:

Interpreter - validator.py - Validator - checker() - Between line 137 and 184

### Reasons:

1. Reduces code readability
2. Adds too much responsibility to one method
3. Reduces code readability (to add new key – value matches)

### Strategies:

1. Replace Conditional with Polymorphism
   1. For each hierarchy subclass, redefine the method that contains the conditional and copy the code of the corresponding conditional branch to that location
   2. Delete this branch from the conditional
   3. Repeat replacement until the conditional is empty. Then delete the conditional and declare the method abstract

Remove each if and elif check for key and subclass

## 3) Duplicated Code

### Name:

Duplicated Code

### Location:

Interpreter - validator.py - Validator - check\_empid(), check\_gender(), check\_age(), check\_sales, check\_BMI(), check\_salary(), check\_birthday()  
- Between line 18 and 134

### Reasons:

1. Severely reduces code readability
2. Reduces code upgradability (to add new key – value matches)
3. Increases overall code size unnecessarily

### Strategies:

1. Extract Class
   1. Create a new class to contain the relevant functionality
   2. Create a relationship between the new class and the old class
   3. Use move field and move method and construct the new class with code from the old class

Create a subclass to check each value based on the key and run the corresponding check() function

## 4) Lazy Class

### Name:

Lazy Class

### Location:

Interpreter - unpickler.py - Unpickler - unpickle\_dictionary()  
- Between line 6 and 19

### Reasons:

1. Should be the same class (Pickler() and Unpickler()) with functions for each
2. Unnecessarily bloats code
3. More difficult to maintain and increases overall code size unnecessarily

### Strategies:

1. Inline Class
   1. In the recipient class, create the public fields and methods present in the donor class. Methods should refer to the equivalent methods of the donor class
   2. Replace all references to the donor class with references to the fields and methods of the recipient class
   3. Start using Move Method and Move Field to completely transplant all functionality to the recipient class from the original one
   4. Delete the original class

Add the methods in the Unpickler() class to the Pickler() class and resolve references to the Unpickler() class in other program components