BCPR301 – Advanced Programming

Contents

[1. NAME: BLOASTERS 4](#_Toc515046611)

[Location: 4](#_Toc515046612)

[Reason: 4](#_Toc515046613)

[Strategies / Approaches: 4](#_Toc515046614)

[Refactoring Step Taken: 4](#_Toc515046615)

[2. NAME: OBJECT ORIENTED ABUSDERS 4](#_Toc515046616)

[Location: 4](#_Toc515046617)

[Reason: 5](#_Toc515046618)

[Strategies / Approaches: 5](#_Toc515046619)

[Refactoring Step Taken: 5](#_Toc515046620)

[3. NAME: Large Class: Dispensable 5](#_Toc515046621)

[Location: 6](#_Toc515046622)

[Reason: 6](#_Toc515046623)

[Strategies / Approaches: 6](#_Toc515046624)

[Refactoring Step Taken: 6](#_Toc515046625)

[4. NAME: Large Class: Change Preventers 7](#_Toc515046626)

[Location: 7](#_Toc515046627)

[Reason: 7](#_Toc515046628)

[Strategies / Approaches: 7](#_Toc515046629)

[Refactoring Step Taken: 7](#_Toc515046630)

Assessment Four- Bad Smells Documentation

# NAME: BLOASTERS

Large Class:

# Location:

Folder: Employee Management System> Employee Database > Employee-Management-System > Refactoring

File: employee\_data.py

Line: 99 to 156

# Reason:

1. In my Assignment one, I think “Large Class” is the most-worst code smell as the “EmployeeData” class is too large.
2. Too many functions and arrays.
3. I found this is the code smell as class is very large, very hard to understand as well as any changes or update to the class will make the class more clumsy
4. it could also break the “Open /Closed principle”

# Strategies / Approaches:

Extract Subclass

# Refactoring Step Taken:

1. Create “GetEmployee(EmployeeData) class
2. Import employee\_get\_data module in InterpreterController(cms) class
3. Create Instance of the GetEmployee() in InterpreterController(cmd) class
4. Use the fields (instance variable) from the super class
5. Unit test
6. Run Test

# NAME: OBJECT ORIENTED ABUSDERS

Refuse Bequest

# Location:

Folder: Employee Management System> Employee Database > Employee-Management-System > Refactoring

File: employee\_get\_data.py

Line: 6 to 71

# Reason:

1. While try to remove first worst code smell “Large Class: EmployeeData”; I have abstract “Subclass:GetEmployee”.
2. The Subclass “GetEmployee” has nothing common with the super class “EmployeeData”
3. The super class and sub class is completely different
4. A subclass, that inherits from a parent class, but the subclass does not need all behaviour provided by the parent class. Because of that, the subclass refuses some behaviour (bequest) of the parent class. That's why this is a code smell.

# Strategies / Approaches:

Replace Inheritance with Delegation

# Refactoring Step Taken:

1. Remove the Inheritance from subclass
2. So both class are independent now, only the super class will be the help of sub class
3. Create a field in the subclass for holding the super class.
   1. Def \_\_init\_\_(self):

Super().\_\_init\_\_()

1. The super() builtin returns a proxy object, a substitute object that has ability to call method of the base class via delegation. This is called indirection (ability to reference base object with super())

# NAME: DISPENSABLE

Data Class

# Location:

Folder: Employee Management System> Employee Database > Employee-Management-System > Refactoring

File: data.py

Line: 5 to 16

# Reason:

1. In my Assignment1; I discover the third worst code smell is “Data(Enum) class” because this class only includes the public attributes and no operation (methods)
2. The attributes are used by other classes but not the class itself
3. The attributes are public, so that it can be accessible by any class and methods, which can cause risk of damaging the attributes

# Strategies / Approaches:

Encapsulate Field

# Refactoring Step Taken:

1. Create setter and getter method
2. def setData(self, Data):  
    self.Data = Data  
     
   def getData(self):  
    return self.Data  
     
   def setEmpID(self, EMPID):  
    self.\_\_EMPID = EMPID  
     
   def setGENDER(self, GENDER):  
    self.\_\_GENDER = GENDER  
     
   def setAGE(self, AGE):  
    self.\_\_AGE = AGE  
     
   def setSALES(self, SALES):  
    self.\_\_SALES = SALES  
     
   def setBMI(self, BMI):  
    self.\_\_BMI = BMI  
     
   def setSALARY(self, SALARY):  
    self.\_\_SALARY = SALARY  
     
   def setBIRTHDAY(self, BIRTHDAY):  
    self.\_\_BIRTHDAY = BIRTHDAY  
     
   def getEMPID(self):  
    return self.EMPID  
     
   def getGENDER(self):  
    return self.GENDER  
     
   def getAGE(self):  
    return self.AGE  
     
   def getSALES(self):  
    return self.SALES  
     
   def getBMI(self):  
    return self.BMI  
     
   def getSALARY(self):  
    return self.SALARY  
     
   def getBIRTHDAY(self):  
    return self.BIRTHDAY
3. So that other object class can use that fields

# NAME: Large Class: CHANGE PREVENTERS

Shotgun Surgery

# Location:

Folder: Employee Management System> Employee Database > Employee-Management-System > Refactoring

File: employee\_data.py

Line: 5 to 16

# Reason:

1. This code smell introduced when the requirement change
2. One changes could affect to many classes
3. That can cause another bed smell “Divergent Change” when many changes are made to a single class.
4. When try to remove the “Shotgun Surgery” bed smell, leaves the original classes almost empty, try to get rid of these now-redundant classes via [**Inline Clas**](https://refactoring.guru/inline-class)**s.**

# Strategies / Approaches:

Move method / Move Field

# Refactoring Step Taken:

1. Create a new class to contain the relevant functionality.