**REFACTORING**

**Refactoring**or **Code Refactoring**is defined as systematic process of improving existing computer code, without adding new functionality or changing external behaviour of the code. It is intended to change the implementation, definition, structure of code without changing functionality of software. It improves extensibility, maintainability, and readability of software without changing what it actually does.

**Why should we refactor our code when it works fine?**

The goal of refactoring is not to add new functionality or remove an existing one. The main goal of refactoring is to make code easier to maintain in future and to fight technical debt. We do refactor because we understand that getting design right in first time is hard and also you get the following benefits from refactoring:

* Code size is often reduced
* Confusing code is restructured into simpler code.

Both of the above benefits greatly improve maintainability which is required because requirements always keep changing.

**When do we refactor?**

* Before you add new features, make sure your design and current code is “good” this will help the new code be easier to write.
* When you need to fix a bug
* When you do a peer review
* During a code review

**Benefits:**

* Refactoring improves the design of software.
* Refactoring makes software easier to understand.
* Refactoring helps us finding bugs in the program.
* Refactoring helps us program faster.

**How to identify code to refactor?**

Martin Fowler proposed using “code smells” to identify when and where to refactor. Code smells are bad things done in code, just like bad patterns in the code. Refactoring and Code smells are a few techniques that help us identify problems in design and implementation. It also helps us in applying known solutions to these problems. **Refactoring Techniques :** There are more than 70 refactoring techniques that exist. But we will discuss only a few, more common ones.

1. **Extract Method –** When we have a code that can be grouped together. **Example:**

 def student():

 getgrades()

 # details

 name = input()

 class = input()

**This could be refactored as:**

def student():

 getgrades()

 getdetails()

def getdetails():

 name = input()

 class = input()

**2. Replace Temp with Query –** When we are using a temporary variable to hold the result of an expression.

**Example :**

SI = P \* R \* T / 100

if(SI > 100):

 return SI

else:

 return SI \* 1.5

**This could be refactored as:**

def SI():

return P \* R \* T / 100

if(SI() > 100):

 return SI()

else:

 return SI()\*1.5

**3. Encapsulate Field –** It involves providing methods that is used to read/write data rather than accessing it directly.

**Example :**

class A:

 variable

**This could be refactored as:**

class A:

 self.variable

 getvariable()

 setvariable()

class A:

 self.variable

 getvariable()

 setvariable()

**4.Inline Method –** When we have a method body which is more obvious than the method itself. **Example :**

class PizzaDelivery:

 def getgrades(self):

 return 'A' if self.moretheneight() else B

 def ismorethaneight(self):

 return self.number > 8

**This could be refactored as:**

class PizzaDelivery:

 def getgrades(self):

 return self.number > 8 ? A : B

**5. Move Method –** When a function class is used by other class more than the class in which it exists.

Class A:

 #...

 abc()

Class B:

 #...

**This could be refactored as:**

Class A:

 #...

Class B:

 #...

 abc()

**6. Replace Conditional with Polymorphism –** When we have a conditional that performs various actions depending on object type or properties.

 **Example :**

=class Bird:

 # ...

 def getSpeed(self):

 if self.type == EUROPEAN:

 return self.getBaseSpeed()

 elif self.type == AFRICAN:

 return self.getBaseSpeed() - self.getLoadFactor() \* self.numberOfCoconuts

 elif self.type == NORWEGIAN\_BLUE:

 return 0 if self.isNailed else self.getBaseSpeed(self.voltage)

 else:

 raise Exception("Should be unreachable")

**This could be refactored as**

class Bird:

 # ...

 def getSpeed(self):

 pass

class European(Bird):

 def getSpeed(self):

 return self.getBaseSpeed()

class African(Bird):

 def getSpeed(self):

 return self.getBaseSpeed() - self.getLoadFactor() \* self.numberOfCoconuts

class NorwegianBlue(Bird):

 def getSpeed(self):

 return 0 if self.isNailed else self.getBaseSpeed(self.voltage)

# Somewhere in client code

speed = bird.getSpeed()

**Note :**