Extract Method:

Problema: Cuantas más líneas se encuentren en un método, más difícil será descubrir qué hace el método.

```
def printOwing(self):
    self.printBanner()

    # print details
    print("name:", self.name)
    print("amount:", self.getOutstanding())
```

Refactorizado

```
def printOwing(self):
    self.printBanner()
    self.printDetails(self.getOutstanding())

def printDetails(self, outstanding):
    print("name:", self.name)
    print("amount:", outstanding)
```

Inline Method:

Problema: Cuando el cuerpo de un método sea más obvio que el método en sí.

```
class PizzaDelivery:
    # ...
    def getRating(self):
        return 2 if self.moreThanFiveLateDeliveries() else 1

    def moreThanFiveLateDeliveries(self):
        return self.numberOfLateDeliveries > 5
```

Refactorizado

```
class PizzaDelivery:
  # ...
  def getRating(self):
    return 2 if self.numberOfLateDeliveries > 5 else 1
```

Extract Variable:

Problema: Tienes una expresión que es difícil de entender.

```
def renderBanner(self):
    if (self.platform.toUpperCase().indexOf("MAC") > -1) and \
       (self.browser.toUpperCase().indexOf("IE") > -1) and \
       self.wasInitialized() and (self.resize > 0):
```

Refactorizado

```
def renderBanner(self):
    isMacOs = self.platform.toUpperCase().indexOf("MAC") > -1
    isIE = self.browser.toUpperCase().indexOf("IE") > -1
    wasResized = self.resize > 0

    if isMacOs and isIE and self.wasInitialized() and wasResized:
```

Replace temp with Query

Problema: Coloca el resultado de una expresión en una variable local para su uso posterior en su código.

```
def calculateTotal():
    basePrice = quantity * itemPrice
    if basePrice > 1000:
        return basePrice * 0.95
    else:
        return basePrice * 0.98
```

Refactorizado

```
def calculateTotal():
    if basePrice() > 1000:
        return basePrice() * 0.95
    else:
        return basePrice() * 0.98

def basePrice():
    return quantity * itemPrice
```

Remove assignments to parameters

Problema: Se asigna algún valor a un parámetro dentro del cuerpo del método.

```
def discount(inputVal, quantity):
    if quantity > 50:
        inputVal -= 2
```

Refactorizado

```
def discount(inputVal, quantity):
    result = inputVal
    if quantity > 50:
        result -= 2
```

Replace method with method object

Problema: Tiene un método largo en el que las variables locales están tan entrelazadas que no puede aplicar el método Extract .

```
class Order:
    # ...
    def price(self):
        primaryBasePrice = 0
        secondaryBasePrice = 0
        tertiaryBasePrice = 0
```

Refactorizado

```
class Order:
    # ...
    def price(self):
        return PriceCalculator(self).compute()


class PriceCalculator:
    def __init__(self, order):
        self._primaryBasePrice = 0
        self._secondaryBasePrice = 0
        self._tertiaryBasePrice = 0
        # Copy relevant information from the
        # order object.

    def compute(self):
```

Split Temporary Variable

Problema: Tiene una variable local que se usa para almacenar varios valores intermedios dentro de un método.

```
temp = 2 * (height + width)
print(temp)
temp = height * width
print(temp)
```

Refactorizado

```
perimeter = 2 * (height + width)
print(perimeter)
area = height * width
print(area)
```

Replace Conditional with Polymorphism

Problema: Tiene un condicional que realiza varias acciones según el tipo de objeto o las propiedades.

```
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")
```

Refactorizado:

```
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()
```