MAKE SCHOOL

Refactoring

Other Refactoring Techniques

SPD 2.31

Warm up

10 mins

MAKE SCHOOL

Study this script. What could be possibly wrong with it (in terms of readability and modularity).

Note: You do not need to refactor this code at this point (you do it later).

Learning Outcomes

MAKE SCHOOL

By the end of today, you should be able to…

1. Compare and contrast different refactoring techniques for moving methods and attributes between classes, extracting class and superclass and replacing magic numbers with constants.
2. Identify code smells and apply refactoring techniques to improve code quality.

Move Method

Move Method

MAKE SCHOOL

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.
from typing import List

class Person:
 def __init__(self, banks):
  self.banks: List[Bank] = banks

 def get_net_worth(self):
  sum = 0
  for i in range(0,len(self.banks)):
   sum += self.get_net_worth_per_bank(self.banks[i])
  return sum

def get_net_worth_per_bank(self, bank):
 for j in range(0,len(bank.accounts)):
  sum += bank.accounts[j].balance
  return sum


class Bank:
 def __init__(self, accounts):
  self.accounts:List[Account] = accounts


class Account:
 def __init__(self, balance):
  self.balance: float = balance
```

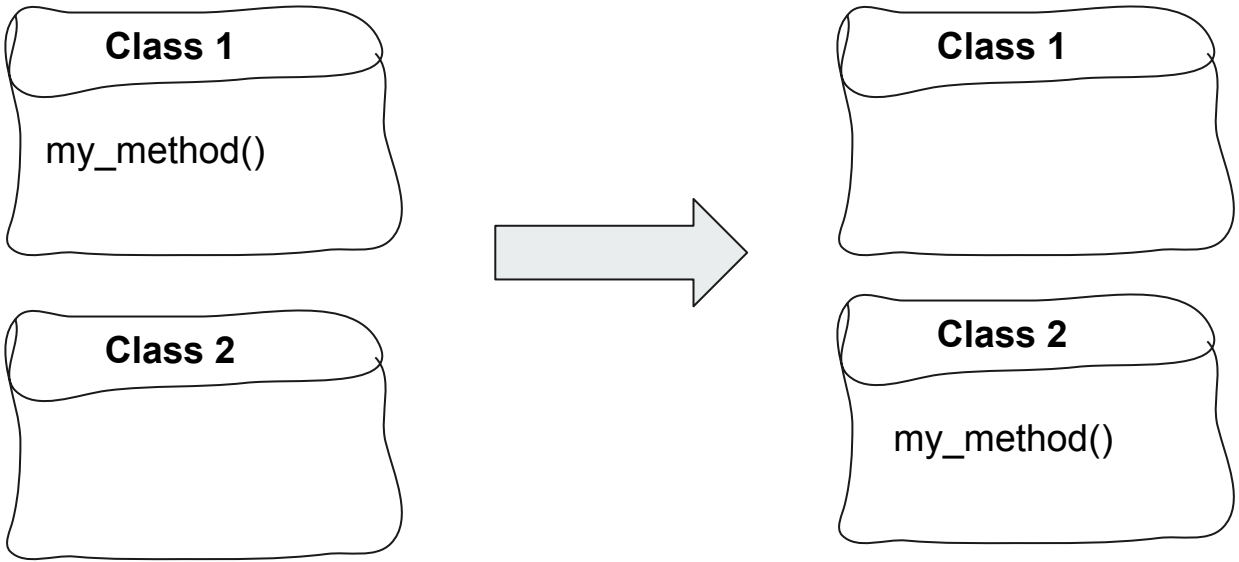
It's more about 'Bank' class than 'Person' class. So move it to 'Bank' class.

Move Method

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.
# Refactored.
from typing import List
class Person:
 def __init__(self, banks):
   self.banks: List[Bank] = banks

 def get_net_worth(self):
   sum = 0
   for i in range(0,len(self.banks)):
     sum += self.banks[i].getNetWorth()
   return sum


class Bank:
 def __init__(self, accounts):
   self.accounts:List[Account] = accounts

 def get_net_worth(self):
   sum = 0
   for j in range(0, self.accounts.Length):
     sum += self.accounts[j].balance
   return sum

class Account:
 def __init__(self, balance):
   self.balance: float = balance
```

Move Method

A method is used more by another class than the class that owns it → Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.



MAKE SCHOOL

Move Attribute

(Formal name: **Move Field**)

Move Field

MAKE SCHOOL

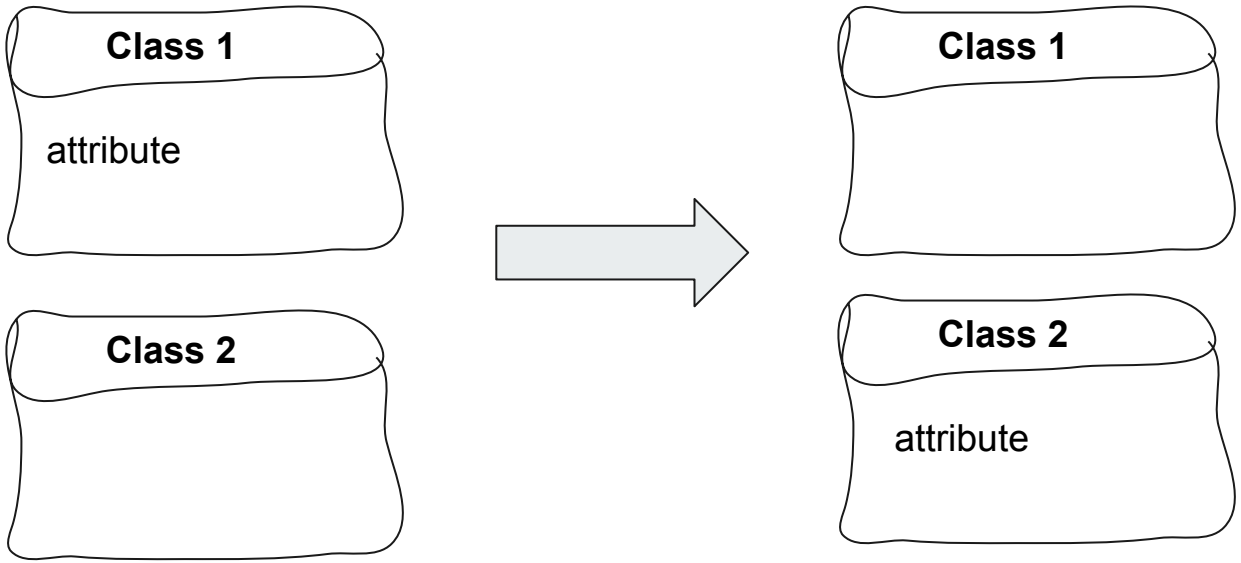
Code is too large to be displayed on a single slide. So let's go to Github:

Code → move_field.py

Refactored Code → move_field_refactored.py

Move Attribute

An attribute is used more by another class than the class who owns it → Create a new attribute in the target class, and change all its users.



Move Field

10 min

Solve Exercise 21: 'Move Attribute' Technique

MAKE SCHOOL

Extract Class

Extract Class

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.
class Person:

    def __init__(self, name, office_area_code, office_tel_number):
        self.name = name
        self.office_area_code  = office_area_code
        self.office_tel_Number = office_tel_number

    def get_telephone_number(self):
        return "(" + self.office_area_code + ") " + self.office_tel_Number


kami = Person('kami', '403', '2008412')
print(kami.get_telephone_number())
```

These pieces of info seem to work together. So let's extract/bundle them into a class.

Extract Class

MAKE SCHOOL

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.
# Refactored.
class Person:

  def __init__(self, name, office_area_code, office_tel_number):
    self.name = name
    self.telephone_number = TelephoneNumber(office_area_code,
                                            office_tel_number)

  def get_telephone_number(self):
    return self.telephone_number.get_telephone_number()


class TelephoneNumber:

  def __init__(self, office_area_code, office_tel_number):
    self.office_area_code  = office_area_code
    self.office_tel_number = office_tel_number

  def get_telephone_number(self):
    return "(" + self.office_area_code + ") " \
           + self.office_tel_number

kami = Person('kami', '403', '2008412)
print (kami.get_telephone_number())
```

Extract Class

MAKE SCHOOL

You have one class doing work that should be done by two. → *Create a new class* and move the relevant attributes (fields) and methods from the old class into the new class.

Extract Class

15 mins

Solve Exercise 22: 'Extract Class' Technique

MAKE SCHOOL

Break for 10 mins

Get up, stretch, get some water, and relax your mind. ☁️

MAKE SCHOOL

Replace Magic Number with Symbolic Constant

Replace Magic Number with Symbolic Constant

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.



def potential_energy(mass, height):
    return mass * 9.81 * height

print(potential_energy(10, 100))
```

Replace Magic Number with Symbolic Constant

MAKE SCHOOL

```
# Adapted from a Java code in the "Refactoring" book by Martin Fowler.
# Refactored.
GRAVITATIONAL_ACCELERATION = 9.81

def potential_energy(mass, height):
    return mass * GRAVITATIONAL_ACCELERATION * height

print(potential_energy(10, 100))
```

Replace Magic Number with Symbolic Constant

MAKE SCHOOL

You have a literal number with a particular meaning. → Create a constant, name it after the meaning, and replace the number with it.

Note: Numbers **0, 1 and 2** are not considered magic numbers (except if they have special meaning in your program). So you do not need to refactor them.

Replace Magic Number with Symbolic Constant

7 mins

MAKE SCHOOL

Solve Exercise 25: 'Replace Magic Number with Symbolic Constant' Technique

MAKE SCHOOL

Extract Superclass

Extract Superclass

MAKE SCHOOL

Code is too large to be displayed on a single slide. So let's go to Github:

Code → extract_superclass.py

Refactored Code → extract_superclass_refactored.py

Extract Superclass

MAKE SCHOOL

You have two classes with similar feature → Create a superclass and move the common features to the superclass.

Extract Superclass

15 mins

Resolve Exercise 26: 'Extract Superclass' Technique

Summary

1. Move method
2. Move field
3. Extract class
4. Replace magic number with symbolic constant
5. Extract superclass

References and Further Study

MAKE SCHOOL

1. "Refactoring: Improving the Design of Existing Code" (1st edition) by Martin Fowler
2. https://en.wikipedia.org/wiki/Code_refactoring
3. https://en.wikipedia.org/wiki/Magic_number_(programming)