SOLID Principles:

Single Responsibility Principle

By Noura Bensaber

In this series, I will highlight some areas where you can improve the quality of your code during the development process. In the first article, I explained why code quality is paramount. Then I introduced briefly some ideas that would help you improve the quality of your code. I will provide more details on these points in future articles. Some concepts will be illustrated with examples in Java and Python. I will begin this journey with you by introducing the SOLID principles in this article. They are some of the concepts that will help you improve the quality of your code and boost your abstract thinking in the object-oriented paradigm. I will elucidate the first principle (Single Responsibility Principle), with examples in Java and Python.

1. SOLID principles

The SOLID principles are a collection of object-oriented design concepts for building a solid object-oriented software architecture. These principles help developers create software that are more readable, flexible, extensible and maintainable.

It was Robert C. Martin who first introduced these principles. But it was Michael Feathers, who came up with the acronym SOLID and reworked these principles. The acronym SOLID stands for:

- **S:** Single responsibility Principle
- **O:** Open/closed Principle
- **L:** Liskov Substitution Principle
- **I:** Interface Segregation Principle
- **D:** Dependency Inversion Principle

2. Advantages of Using SOLID Principles

Compliance with SOLID principles has a big impact on the quality of your software. We can mention some benefits of using these principles as follows:

- Reduce the complexity of the code.
- Increase readability, extensibility, and maintenance.
- Increase flexibility and reusability.
- Reduce tight coupling.
- Reduce errors for better testability.

We'll show how using SOLID principles helps you develop a code with better quality that confirms the achievement of the advantages described above. In this series, we will explain each principle individually and illustrate with an example how each improves certain aspects of the software. But, we focus only on the Single Responsibility Principle (SRP) in this article.

3. Single Responsibility Principle (SRP):

This principle means that a class should only have one responsibility. In other words, each class should only contain one feature or behavior. This principle helps you ensure code quality by preserving some of the code quality features we mention via the following benefits:

Testability: it is easy to define test cases for a class with a single responsibility.

Loose coupling: a class implementing SRP has few dependencies because it only encapsulates a single functionality.

Readability: this principle ensures that classes adhering to it are small, well-organized, and easy to understand.

Separation of Concerns: this principle preserves the concept of separation of concerns.

Maintainability: the ability to update the software is high.

Extensibility: adding new features is easy.

4. Example of SRP in Java

To explain how a class complies with SRP or not, we create an illustrative class 'Calculator' with a few methods that we named 'add', 'div', 'isMultipleOf', and 'removeWSpace'.

We defined the first three methods for arithmetic operations and followed strong logic to put them together (strong cohesion).

The fourth operation removes white spaces from a variable of type string. That means it has a weak cohesion with the 'add', 'div', and 'isMultipleOf'.

Consider one responsibility according to our requirements, which is achieving mathematical operations. Therefore, this example violates the SRP because it does not encapsulate this sole responsibility. This could result in the loss of the benefits of the SRP mentioned above.

```
public class Calculator {
    public Calculator() { }
    public static int add(int x, int y) {
        return x + y;
    }
    public static int div(int x, int y) {
        return x / y;
    }
    public static boolean isMultipleOf(int x, int y) {
        if (x % y == 0) {
            return true;
        } else {
            return false;
        }
    }
    public static String removeWSpace(String inputStr) {
```

```
        return inputStr.replaceAll("\\s+","");
    }
}
```

4. Example of SRP in Python

Consider a class in Python that violates the SRP according to our requirements. We define a class named 'ListExtremum' which encapsulates four methods 'find_max', 'find_min', 'find_max_dict', and 'find_min_dict'.

The 'find_max' and 'find_min' methods look for the max and the min values respectively in a list. The 'find_max_dict' and 'find_min_dict' methods return the max and the min values respectively in a dictionary.

This example violates the SRP because it fulfills more than one responsibility. The first responsibility consists of calculating the extrema (minimum and maximum) of a list. The second one is to calculate the extrema of a dictionary.

```
class ListExtremum:
    def __init__(self, num_list, dict):
        self.num_list = num_list
        self.dict = dict

    def find_max(self):
        try:
```

```
        maximum = self.num_list[0]
        for number in self.num_list:
            if number > maximum:
                maximum = number
        return maximum
    except TypeError:
        print("Method find_max - Error empty list")
    except IndexError:
        print("Method find_max - Error empty list")


def find_min(self):
    try:
        minimum = self.num_list[0]
        for number in self.num_list:
            if number < minimum:
                minimum = number
        return minimum
    except TypeError:
        print("Method find_min - Error empty list")
    except IndexError:
        print("Method find_min - Error empty list")
```

```
def find_max_dict(self):
    try:
        max_key = max(self.dict, key=self.dict.get)
        max_value = self.dics[max_key]
        return max_value
    except AttributeError:
        print("Method find_max_dict - Error empty dictionary")
    except ValueError:
        print("Method find_max_dict - Error empty dictionary")


def find_min_dict(self):
    try:
        min_key = min(self.dics, key=self.dics.get)
        min_value = self.dics[min_key]
        return min_value
    except AttributeError:
        print("Method find_min_dict - Error empty dictionary")
    except ValueError:
        print("Method find_min_dict - Error empty dictionary")
```

We explain what each method does in this code.

Constructor '__init__': this code defines a constructor function (__init__) for the 'ListExtremum' class. It takes two arguments ('num_list' and 'dict') and assigns them to instance variables ('self.num_list' and 'self.dict').

Method 'find_max': this method searches for the maximum number in 'self.num_list'. It initializes the maximum variable with the first element of 'self.num_list'. Then, it iterates over the list and updates the maximum variable when it finds a number greater than the current maximum. It handles 'TypeError' and 'IndexError' exceptions and prints an error message if the list is None or empty.

Method 'find_min': this method returns the minimum value in 'self.num_list'. It initializes the minimum variable with the element of 'self.num_list' at the first index. After that, it iterates over the list and updates the minimum variable when it finds a number smaller than the current minimum. It handles potential errors for an empty list by catching 'TypeError' and 'IndexError' exceptions and printing an error message.

Method 'find_max_dict': this method finds the key-value pair with the highest value in a dictionary. It uses the max function with the key parameter set to 'self.dict.get' to find the key with the highest value. If the dictionary is empty, it catches the 'AttributeError' and 'ValueError' exceptions and prints an error message.

Method 'find_min_dict': this method aims to find the key-value pair with the minimum value in a dictionary. It uses the min function to find the minimum key based on the values. Then, it retrieves the corresponding value from the dictionary. Like 'find_max_dict', this method catches the 'AttributeError' and 'ValueError' exceptions and prints an error message, if the dictionary is empty.

We should split this code into parts to make this class adhere to the SRP. The first part will implement operations applied to a list. The second part contains the operations of dictionaries. In this case, we get two classes called 'ListExtremum' and 'DictExtremum'.

The 'ListExtremum' class contains the 'find_max' and 'find_min' methods to calculate the max and min value of a list. On the other hand, the 'DictExtremum' class contains the 'find_max_dict' and 'find_min_dict' methods to get the max and min value of a dictionary.

```
class ListExtremum:
    def __init__(self, num_list):
        self.num_list = num_list


    def find_max(self):
        try:
            maximum = self.num_list[0]
            for number in self.num_list:
```

```
            if number > maximum:
                maximum = number
        return maximum
    except TypeError:
        print("Method find_max - Error empty list")
    except IndexError:
        print("Method find_max - Error empty list")


def find_min(self):
    try:
        minimum = self.num_list[0]
        for number in self.num_list:
            if number < minimum:
                minimum = number
        return minimum


    except TypeError:
        print("Method find_min - Error empty list")
    except IndexError:
        print("Method find_min - Error empty list")
```

```
class DictExtremum:
    def __init__(self,  dict):
        self.dict = dict


    def find_max_dict(self):
        try:
            max_key = max(self.dict, key=self.dict.get)
            max_value = self.dict[max_key]
            return max_value
        except AttributeError:
            print("Method find_max_dict - Error empty dictionary")
        except ValueError:
            print("Method find_max_dict - Error empty dictionary")


    def find_min_dict(self):
        try:
            max_key = min(self.dics, key=self.dics.get)
            max_value = self.dics[max_key]
            return max_value
        except AttributeError:
```

```
            print("Method find_min_dict - Error empty dictionary")
        except ValueError:
            print("Method find_min_dict - Error empty dictionary")
```

We can call this code using some examples as follows:

```
print("***********************************************")
print("Calculate max and min values in None list and None dictionary")
print("")
list_extremum_calc = ListExtremum(None)
print("Max value in the list: ", list_extremum_calc.find_max())
print("Min value in the list: ", list_extremum_calc.find_min())

dict_extremum_calc = DictExtremum(None)
print("Max value in the dictionary: ", dict_extremum_calc.find_max_dict())
print("Min value in the dictionary: ", dict_extremum_calc.find_min_dict())

print("")
print("***********************************************")
print("Calculate max and min values in empty list and empty dictionary")
```

```
list_extremum_calc = ListExtremum([])
print("Max value in the list: ", list_extremum_calc.find_max())
print("Min value in the list: ", list_extremum_calc.find_min())
dict_extremum_calc = DictExtremum({})
print("Max value in the dictionary: ", dict_extremum_calc.find_max_dict())
print("Min value in the dictionary: ", dict_extremum_calc.find_min_dict())

print("")
print("**********************************************")

num_list = [18, 20, 25, -16, 27]
temp_dict = {'Vienna': 10, 'Paris': 11, 'Rome': 15, 'Algiers': 16}

print("num_list: ", num_list)
print("temp_dict: ", temp_dict)
print("Calculate max and min values in num_list and temp_dict")
print("")

list_extremum_calc = ListExtremum(num_list)
print("Max value in the list: ", list_extremum_calc.find_max())
print("Min value in the list: ", list_extremum_calc.find_min())
```

```
dict_extremum_calc = DictExtremum(temp_dict)
print("Max value in the dictionary: ", dict_extremum_calc.find_max_dict())
print("Min value in the dictionary: ", dict_extremum_calc.find_min_dict())
```

The execution of this code gives us the following result:

```
**********************************************
Calculate max and min values in None list and None dictionary

Method find_max - Error empty list
Max value in the list:  None
Method find_min - Error empty list
Min value in the list:  None
Method find_max_dict - Error empty dictionary
Max value in the dictionary:  None
Method find_min_dict - Error empty dictionary
Min value in the dictionary:  None

**********************************************
```

Calculate max and min values in empty list and empty dictionary

Method find_max - Error empty list

Max value in the list: None

Method find_min - Error empty list

Min value in the list: None

Method find_max_dict - Error empty dictionary

Max value in the dictionary: None

Method find_min_dict - Error empty dictionary

Min value in the dictionary: None

****

num_list: [18, 20, 25, -16, 27]

temp_dict: {'Vienna': 10, 'Paris': 11, 'Rome': 15, 'Algiers': 16}

Calculate max and min values in num_list and temp_dict

Max value in the list: 27

Min value in the list: -16

Max value in the dictionary: 16

Method find_min_dict - Error empty dictionary

Min value in the dictionary: None