FH University of Applied Sciences
TECHNIKUM WIEN

SEMINAR PAPER

In Computer Science

Interface segregation principle

By: Peter Oettl

Student Number: 1910257146

Supervisor: DI Wolfgang Berger

Wien, May 25, 2021

Contents

1 SOLID principles

These principles describe practices for software development with considerations for maintaining and extending as the project grows. Adopting these practices can also contribute to avoiding code smells, refactoring code, and Agile or Adaptive software development. [1]

1.1 S - Single Responsiblity Principle

The single responsibility principle describes that one class is only supposed to handle one topic. This principle ensures that a class exists only for a single reason but can have multiple methods to carry out distinct functions.[2]

1.2 O - Open Close Principle

This principle ensures that objects should be open for extension but closed for any modification. This principle ensures that the code is easily extensible without editing or rewriting the existing codebase. Designing software applications using this principle ensures extensibility and the reusability of objects.[2]

1.3 L - Liskov Substitution Principle

This principle describes that if a section of your code is extending a superclass, then all subclasses of the superclass should be able to replace the superclass in your code. A part of your code does not have to know which class it is, since it is a superclass subclass. The use-case of this principle ensures the easy integration of classes.[2]

1.4 I - Interface Segregation Principle

The interface segregation principle simply means that larger interfaces should be split into smaller ones. This ensures that implementing classes only need to be concerned about the methods that are of interest and necessary. [3]

1.5 D - Dependency Inversion Principle

The dependency inversion principle guarantees that classes should not depend on solid classes but should only depend on an abstraction of classes. By following this approach, makes the replacing and changing of modules and classes or services more simple. The dependency inversion principle makes changing dependencies easier.[2]

2 Interface segregation principle

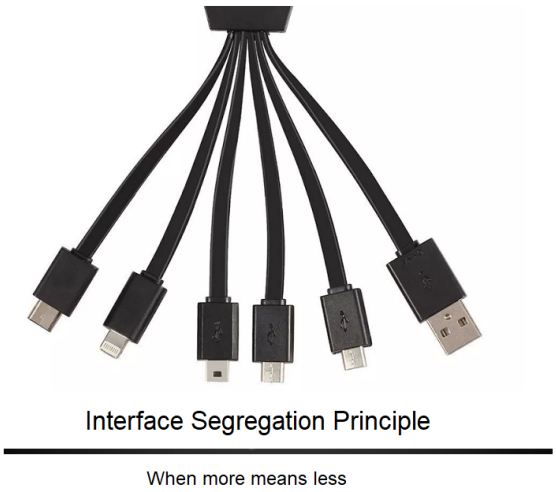

Figure 1: Interface segregation principle

2.1 Interface

An interface is a blueprint for an object and what it is capable of. In Object-Oriented Programming, an Interface is a description of all functions that an object must have to be a specified Interface. Again, as an example, anything that "ACTS LIKE" a light, should have a turnOn() method and a turnOff() method. The purpose of interfaces help developers to enforce these properties and to know that an object of TYPE T must have functions and properties called X, Y, Z, etc. [4]

2.2 ISP

> "Clients should not be forced to depend upon interfaces that they do not use."[3]

This principle was developed by Robert C. Martin while working at Xerox as a consultant. Martin was in charge of developing a new printing management system that could carry out tasks concurrently, which was new by the time.[5]

During the development process, Martin noticed that smaller changes in the design caused larger deployments. In some same cases caused deployments on modules out of the scope of the change. The problem was mainly caused by a large fat interface used throughout all the modules of the system.[5]

This principle is simple and clear, interfaces should not be forced to implement methods that they do not use. It has the main goal of limiting the scope of future changes. However, it is very difficult, in the first iterations, to identify the critical parts of your solution that will violate the principle.[5]

2.3 Problem with will be solved with the ISP

With ISP you can prevent overloaded interfaces that force the client to implement a method that is not required. The client ends up implementing a usefulness method, in other words, a method that has no meaning to the client. These circumstances decrease the readability of the code and also confuses every other developer who will work with the code. The client interface ends up violating SRP sometimes since it might perform some action that is not related to it.[6]

3 Practical Example

In the first example, the Ostrich class is forced to use the fly method. This clear violate the Interface Segregation Principle. As you can see in the second example, the fly and walk methods were separated into an extra interface. With this implementation, it is possible to use the fly and walk method at the Duck class and the walk method at the Ostrich class. Now no class is forced to implement an unused method.

```
interface Bird {
  fly(): void;
  walk(): void;
}
class Duck extends Bird{
    fly(){
        // Duck can fly
    }
    walk(){
        // Duck can walk
    }
}
class Ostrich extends Bird{
    fly(){
        // Ostrich cant fly... throw some error
    }
    walk(){
        // Ostrich can walk
    }
}
```

Listing 1: initial scenario

```
interface BirdFly{
    fly(): void;
}
interface BirdWalk{
    walk(): void;
}
class Duck extends BirdFly, BirdWalk{
    fly(){
        // Duck can fly
    }
    walk(){
        // Duck can walk
    }
}
class Ostrich extends BirdWalk{
    walk(){
        // Ostrich can walk
    }
}
```

Listing 2: ISP Implementation

Bibliography

[1] Digitalocean, “Solid: The first 5 principles of object oriented design,” 2021. [Online]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

[2] A. Ridwan, “Introduction to the solid principle,” 2021. [Online]. Available: https://www.section.io/engineering-education/introduction-to-solid-principle/

[3] T. JANSSEN, “Solid design principles explained: Interface segregation with code examples,” 2018. [Online]. Available: https://www.section.io/engineering-education/introduction-to-solid-principle/

[4] “Interfaces in object oriented programming languages.” [Online]. Available: https://www.cs.utah.edu/~germain/PPS/Topics/interfaces.html

[5] V. M. Pinzon, “Solid: Interface segregation principle.” [Online]. Available: https://dev.to/victorpinzon1988eng/solid-interface-segregation-principle-31c2

[6] “Interface segregation principle.” [Online]. Available: https://www.c-sharpcorner.com/UploadFile/pranayamr/overview-of-interface-segregation-principle/

List of Figures

List of source codes