SOLID
Interface Segregation Principle
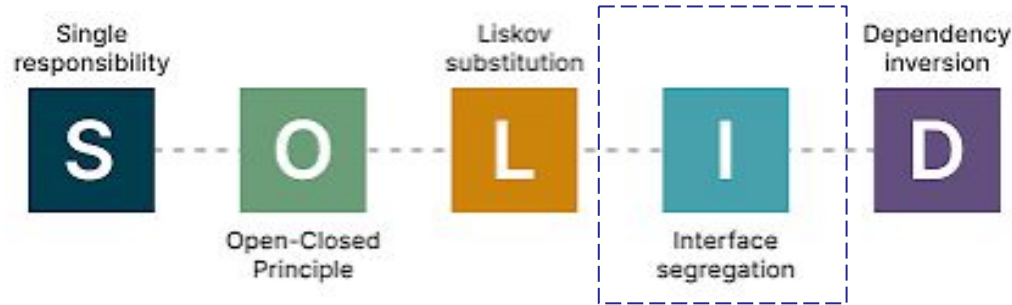
Upcode Software
Engineer Team

-2023-

CONTENT

1. What is Interface Segregation Principle (ISP) ?
2. Why needs to ISP ?
3. How to use ISP ?
4. Where to use ISP ?
5. SOURCE code (github)

1.What is Interface Segregation Principle (ISP) ?

- **Robert Martin** introduced them in the book Agile Software Development, Principles, Patterns, and Practices
- **SOLID** is a mnemonic for five design principles intended to make software designs more understandable, flexible and maintainable.

Interface segregation principle states:

- A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use. (*First defined by Robert C. Martin*)
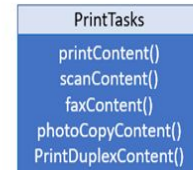
2. Why needs to ISP

By violating the ISP, we face the following problems in our code:
    1. Client developers are confused by the methods they don't need.
    2. Maintenance becomes harder because of side effects: a change in an interface forces us to change classes that don't implement the interface.
    3. Implementing FAT interface with its unwanted methods and values could lead to many side effects.
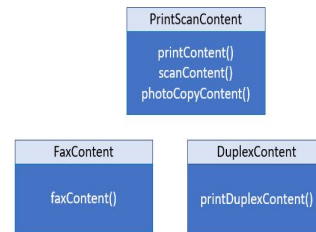

PrintTasks
printContent()
scanContent()
faxContent()
photoCopyContent()
PrintDuplexContent()

Fat Interface, doing too many things

* FAT Interface - is an interface which contains many unrelated values and methods inside it

3. How to use ISP :

1. Split large interfaces into smaller ones.

2. Inherit multiple small interfaces if required.

3. Use the adapter design pattern for the third-party large interface so that your code can work with the adapter.

PrintScanContent

printContent()
scanContent()
photoCopyContent()

FaxContent

faxContent()

DuplexContent

printDuplexContent()

4. Where to use ISP

Advantages:

1. **Flexibility:** In absence of ISP, you have one Generic FAT interface and many classes implementing it. Assume that you had 1 interface and 50 classes. If there is a change in interface, all 50 classes have to change their implementation. With ISP, you will divide generic FAT interface into fine granular small interfaces. If there is a change in small granular interface, only the classes implementing that interface will be affected.
2. **Maintainability and Ease of use:** Since changes are limited to fine granular interface instead of generic FACT interface, code maintenance is easier. Unrelated code is no longer part of implementation classes.

* FACT interface - is an interface which contains from only specific values and methods that are related to each other

5. SOURCE code

```csharp
public interface IPrinterTasks
{
    void Print(string PrintContent);
    void Scan(string ScanContent);
    void Fax(string FaxContent);
    void PrintDuplex(string PrintDuplexContent);
}
```

Forcing a class to provide the body of an interface method means violating the Interface Segregation Principle

```csharp
class HPLaserJetPrinter : IPrinterTasks
{
    public void Print(string PrintContent)
    {
        Console.WriteLine("Print Done");
    }

    public void Scan(string ScanContent)
    {
        Console.WriteLine("Scan content");
    }

    public void Fax(string FaxContent)
    {
        Console.WriteLine("Fax content");
    }

    public void PrintDuplex(string PrintDuplexContent)
    {
        Console.WriteLine("Print Duplex content");
    }
}
```

```csharp
class LiquidInkjetPrinter : IPrinterTasks
{
    public void Print(string PrintContent)
    {
        Console.WriteLine("Print Done");
    }

    public void Scan(string ScanContent)
    {
        Console.WriteLine("Scan content");
    }

    public void Fax(string FaxContent)
    {
        throw new NotImplementedException();
    }

    public void PrintDuplex(string PrintDuplexContent)
    {
        throw new NotImplementedException();
    }
}
```

5. SOURCE code

```csharp
public interface IPrinterTasks
{
    void Print(string PrintContent);
    void Scan(string ScanContent);
}
interface IFaxTasks
{
    void Fax(string content);
}
interface IPrintDuplexTasks
{
    void PrintDuplex(string content);
}
```

Splitting a big interface into smaller ones

Summary

- The ISP is a straightforward principle that is also easy to violate by adding methods to existing interfaces that the clients don't need.
- ISP is also closely related to other SOLID principles.
- **Advantages:**
  1. Flexibility
  2. Maintainability and Ease of use

Reference Resources

1. **Dive into design pattern compression (book)**
2. SOLID Design Principles Explained: Dependency Inversion Principle with Code [Examples](#)
3. Interface Segregation Principle in [Java](#)

Thank you!

Presented by **Sanjar Suyunov**

sanjarsuyunov1304@gmail.com