Interface Segregation Principle (ISP) - C++ Notes

1. Introduction

The Interface Segregation Principle (ISP) is one of the five SOLID principles of object-oriented design. It states that clients should not be forced to depend on interfaces they do not use. Instead of having one large interface with multiple responsibilities, it's better to split it into smaller, more specific interfaces.

2. Problems with Large Interfaces

When a single interface contains multiple responsibilities (such as printing, scanning, and faxing), it forces implementers to define methods they may not need. This leads to several problems:

- **Unnecessary Dependencies**: A class may be required to implement methods it does not use.
- **Code Complexity**: Implementers might throw exceptions or leave methods empty, making the code harder to maintain.
- **Tight Coupling**: Changes to the interface affect all implementers, even if they only need a subset of functionalities.

3. Solution: Segregating Interfaces

Instead of defining a large interface, we should create smaller, focused interfaces that define only the required functionalities. This allows implementers to choose only the interfaces they need.

4. Implementation in C++

Bad Example: Large Interface

```
struct IMachine
{
  virtual void print(Document& doc) = 0;
  virtual void fax(Document& doc) = 0;
  virtual void scan(Document& doc) = 0;
};

struct MFP : IMachine // Multi-Function Printer
{
  void print(Document& doc) override;
  void fax(Document& doc) override;
  void scan(Document& doc) override;
};
```

Problems:

- If a device only prints or only scans, it still has to implement unused methods.
- Implementers may throw exceptions or leave methods blank, which misleads clients.

Good Example: Segregated Interfaces

```
struct IPrinter
{
  virtual void print(Document& doc) = 0;
};

struct IScanner
{
  virtual void scan(Document& doc) = 0;
};

struct Printer : IPrinter
{
  void print(Document& doc) override;
};

struct Scanner : IScanner
{
  void scan(Document& doc) override;
};
```

Advantages:

- Each class only implements the methods it needs.
- Avoids empty or misleading methods.
- More modular and easier to maintain.

Combining Interfaces When Needed

```
struct IMachine: IPrinter, IScanner
{
};

struct Machine : IMachine
{
  IPrinter& printer;
  IScanner& scanner;

  Machine(IPrinter& printer, IScanner& scanner)
    : printer{printer},
      scanner{scanner}
  {
  }

  void print(Document& doc) override {
    printer.print(doc);
  }
  void scan(Document& doc) override {
    scanner.scan(doc);
  }
};
```

Advantages:

- Allows composition instead of forcing implementation.
- Encourages flexibility and modularity.

5. Benefits of ISP

- **Increased Flexibility**: Developers can pick and choose interfaces that match their needs.
- **Improved Code Maintainability**: Reduces unnecessary code and dependencies.
- **Better Readability**: Clearly communicates a class's responsibilities.
- **Encourages Reusability**: Smaller interfaces can be combined in different ways to create more complex structures.

6. Conclusion

The Interface Segregation Principle helps keep interfaces clean, modular, and specific. By splitting large interfaces into smaller ones, developers can create more maintainable and scalable systems.