Interface Segregation Principle

- Interfaces are a core part of Java and are used extensively to achieve abstraction and to support multiple inheritance of type (the ability of a class to implement more than one interface)
- This principle states that “Clients should not be forced to depend on methods that they do not use”. Here, the term “Clients” refers to the implementing classes of an interface.
- Basically, your interface shouldn't be bloated with methods that implementing classes don't require
- "Fat Interfaces" implement classes that are unnecessarily forced to provide implementations (dummy/empty) even for those methods that they don't need
- In addition, the classes are subject to change when the interface changes. An addition of a method or change to a method signature requires modifying all the implementation classes even if some of them don't use the method
- Break "Fat interfaces" into smaller and highly cohesive interfaces known as "role interfaces"
  - Each "role interface declares one or more methods for a speicific behavior,"

Violation (Bad) Example

Consider the requirements of an application that builds different types of toys. Each toy will have a price and color. Some toys, such as a toy car or toy train can additionally move, while some toys, such as a toy plane can both move and fly. An interface to define the behaviors of toys is this.

Toy.java

```
public interface Toy {
    void setPrice(double price);
    void setColor(String color);
    void move();
    void fly();
}
```

ToyHouse.java

```
public class ToyHouse implements Toy {
    double price;
    String color;
    @Override
    public void setPrice(double price) {
        this.price = price;
    }
    @Override
    public void setColor(String color) {
        this.color=color;
    }
    @Override
    public void move(){}
    @Override
    public void fly(){}
}
```

As you can see, it is useless to provide ToyHouse with implementations of move and fly. This also leads to violation of the open closed principle

Following the Interface Segregation Principle

Create interfaces for specific behaviors Toy.java

```
package guru.springframework.blog.interfacesegregationprinciple;

public interface Toy {
     void setPrice(double price);
     void setColor(String color);
}
```

Movable.java

```
package guru.springframework.blog.interfacesegregationprinciple;

public interface Movable {
```

```
    void move();
}
```

Flyable.java

```
package guru.springframework.blog.interfacesegregationprinciple;

public interface Flyable {
    void fly();
}
```

Now we can implement classes which only implement interfaces they are interested in.

ToyHouse.java

```
package guru.springframework.blog.interfacesegregationprinciple;

public class ToyHouse implements Toy {
    double price;
    String color;

    @Override
    public void setPrice(double price) {

        this.price = price;
    }
    @Override
    public void setColor(String color) {

        this.color=color;
    }
    @Override
    public String toString(){
        return "ToyHouse: Toy house- Price: "+price+" Color: "+color;
    }
}
```

ToyCar.java

```
package guru.springframework.blog.interfacesegregationprinciple;
```

```
public class ToyCar implements Toy, Movable {
    double price;
    String color;

    @Override
    public void setPrice(double price) {

        this.price = price;
    }

    @Override
    public void setColor(String color) {
     this.color=color;
    }
    @Override
    public void move(){
        System.out.println("ToyCar: Start moving car.");
    }
    @Override
    public String toString(){
        return "ToyCar: Moveable Toy car- Price: "+price+" Color: "+color;
    }
}
```

ToyPlane.java

```
package guru.springframework.blog.interfacesegregationprinciple;

public class ToyPlane implements Toy, Movable, Flyable {
    double price;
    String color;

    @Override
    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public void setColor(String color) {
        this.color=color;
```

```
    }
    @Override
    public void move(){

        System.out.println("ToyPlane: Start moving plane.");
    }
    @Override
    public void fly(){

        System.out.println("ToyPlane: Start flying plane.");
    }
    @Override
    public String toString(){
        return "ToyPlane: Moveable and flyable toy plane- Price: "+price+" Color: "+color;
    }
}
```

Summary of Interface Segregation Principle

Both the Interface Segregation Principle and Single Responsibility Principle have the same goal: ensuring small, focused, and highly cohesive software components. The difference is that Single Responsibility Principle is concerned with classes, while Interface Segregation Principle is concerned with interfaces.Interface Segregation Principle is easy to understand and simple to follow. But, identifying the distinct interfaces can sometimes be a challenge as careful considerations are required to avoid proliferation of interfaces. Therefore, while writing an interface, consider the possibility of implementation classes having different sets of behaviors, and if so, segregate the interface into multiple interfaces, each having a specific role.

Interface Segregation Principle in the Spring Framework

The Interface Segregation Principle becomes especially important when doing Enterprise Application Development with the Spring Framework.

As the size and scope of the application you’re building grows, you are going to need pluggable components. Even when just for unit testing your classes, the Interface Segregation Principle has a role. If you’re testing a class which you’ve written for dependency injection it is ideal that you write to an interface. By designing your classes to use dependency injection against an interface, any class implementing the specified interface can be injected into your class. In testing your classes, you may wish to inject a mock object to fulfill the needs of your unit test. But when the class you wrote is running in production, the Spring Framework would inject the real full featured implementation of the interface into your class.

The Interface Segregation Principle and Dependency Injection are two very powerful concepts to master when developing enterprise class applications using the Spring Framework.

Revision #1
Created 17 April 2022 00:53:08 by Elkip
Updated 17 April 2022 00:53:38 by Elkip