



O Princípio I do SOLID

Interface Segregation Principle - ISP

Introdução ao conceito I do SOLID

Imagine que você está em uma aula de arte, onde você pode escolher entre desenhar, pintar ou esculpir. Seria estranho se o professor obrigasse todo mundo a fazer as três coisas, mesmo que você só quisesse desenhar, certo? Se você fosse forçado a fazer tudo, talvez não ficasse feliz e nem conseguisse terminar sua obra.



O conceito I do SOLID é como essa aula: "Interface Segregation Principle" ou Princípio da Segregação de Interfaces. Ele diz que as interfaces devem ser pequenas e específicas para o que você precisa fazer, e não obrigar ninguém a implementar coisas que não fazem sentido para ele.

Em outras palavras, em vez de ter um "contrato" grande que força todos a fazerem tudo, é melhor ter "contratos" menores, onde cada um escolhe o que realmente precisa.

Agora, vamos ver isso na prática com exemplos práticos em Java.



Sem aplicar o princípio ISP:

```
public interface Animal {  
    void comer();  
    void voar();  
    void nadar();  
}
```

E, agora, tentamos criar uma classe Cachorro que implementa essa interface:

```
public class Cachorro implements Animal {  
    @Override  
    public void comer() {  
        System.out.println("Cachorro está comendo.");  
    }  
  
    @Override  
    public void voar() {  
        // Cachorros não voam!  
        throw new UnsupportedOperationException(  
            "Cachorros não podem voar.");  
    }  
  
    @Override  
    public void nadar() {  
        System.out.println("Cachorro está nadando.");  
    }  
}
```



Aqui, a classe **Cachorro** foi forçada a implementar o método `voar()`, mesmo que isso não faça sentido. Isso viola o Princípio da Segregação de Interfaces, porque **Cachorro** tem que "assinar" um contrato que inclui métodos desnecessários.



Aplicando o princípio ISP:

```
public interface Animal {  
    void comer();  
}
```

```
public interface Voar {  
    void voar();  
}
```

```
public interface Nadar {  
    void nadar();  
}
```

E vamos adaptar nossas classes para usar apenas o que elas precisam:



```
public class Cachorro implements Animal, Nadar {  
    @Override  
    public void comer() {  
        System.out.println("Cachorro está comendo.");  
    }  
  
    @Override  
    public void nadar() {  
        System.out.println("Cachorro está nadando.");  
    }  
}
```

```
public class Pato implements Animal, Voar, Nadar {  
    @Override  
    public void comer() {  
        System.out.println("Pato está comendo.");  
    }  
  
    @Override  
    public void voar() {  
        System.out.println("Pato está voando.");  
    }  
  
    @Override  
    public void nadar() {  
        System.out.println("Pato está nadando.");  
    }  
}
```



Agora, Cachorro só implementa os métodos que fazem sentido para ele, enquanto Pato pode implementar todos os métodos porque ele faz todas essas coisas.



CONCLUSÃO

Seguir o Princípio da Segregação de Interfaces é como ter aulas de arte em que você escolhe apenas o que deseja fazer. No código, isso significa criar interfaces menores e mais específicas para que as classes implementem apenas o que precisam.

Com isso, você torna o código mais flexível, fácil de entender e menos propenso a erros!



**Compartilha com aquele que quer
aprender.**



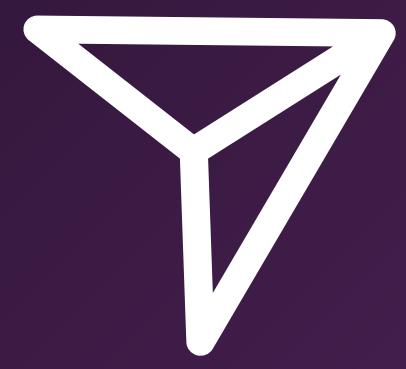
curti



comenta



compartilha



envia

Que a força do código esteja com você !