S.O.L.I.D - Princípio da Segregação de Interfaces (ISP)

Definição do ISP como parte dos princípios S.O.L.I.D.

Proposto por Robert C. Martin para promover o design de software mais flexível e coeso, o ISP, Princípio da Segregação de Interfaces, é um dos cinco princípios do S.O.L.I.D. Ele argumenta que nenhum cliente deve ser forçado a depender de interfaces que não utiliza.

Objetivo do ISP

- Busca coesão e flexibilidade ao segmentar interfaces em conjuntos mais específicos de funcionalidades.
- Evitar que uma classe seja forçada a implementar interfaces que contenham métodos que ela não utiliza.

Redução de Acoplamento

- Interfaces mais específicas significam menos dependências entre componentes, resultando em um acoplamento mais fraco.

Legibilidade e Manutenção

- Códigos mais especializados são mais fáceis de entender e manter.
- Com interfaces menores, as mudanças em uma parte do sistema são menos propensas a afetar outras partes.

```java
package Interfaces;

public interface trabalhador { // Interface original
    void trabalhar();

    void descansar();
}
```

Exemplo Prático - Interface Original

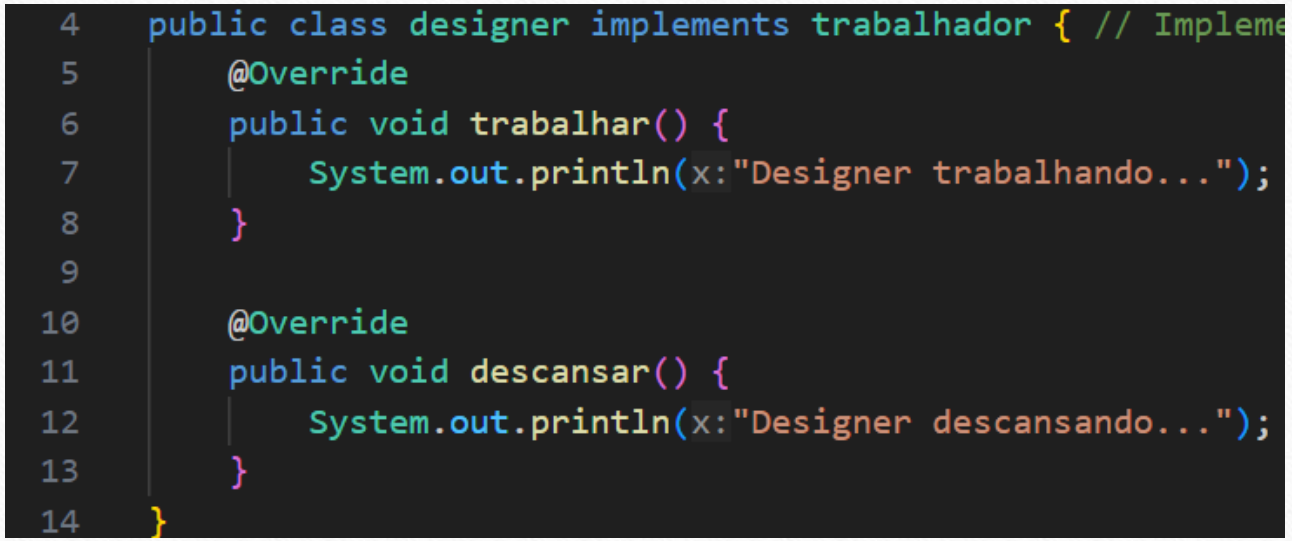
Exemplo Prático - Implementações

```java
public class designer implements trabalhador { // Impleme
    @Override
    public void trabalhar() {
        System.out.println(x:"Designer trabalhando...");
    }

    @Override
    public void descansar() {
        System.out.println(x:"Designer descansando...");
    }
}
```

```java
public class programador implements trabalhador {// Implemen
    @Override
    public void trabalhar() {
        System.out.println(x:"Programador trabalhando...");
    }

    @Override
    public void descansar() {
        System.out.println(x:"Programador descansando...");
    }
}
```

Exemplo Prático - Problema Detectado

A classe "Designer", em sua natureza, não exige ter o metódo "trabalhar". Portanto, representa uma violação da ISP.



Segregação de Interfaces

```
3   public interface trabalhadorDescanso {
4       void descansar();
5   }
6
```

```
3   public interface trabalhadorTrabalho {
4       void trabalhar();
5   }
6
```

Exemplo Prático - Implementações Segregadas

```
4   public class designerSegregado implements trabalhadorDescanso {
5       @Override
6       public void descansar() {
7           System.out.println(x:"Designer descansando...");
8       }
9   }
```