# Lista de Exercícios – Princípios SOLID

## **1. SRP – Single Responsibility Principle**

1. Explique, com suas palavras, o que significa o princípio da responsabilidade única.

R: É o principio que diz que toda classe precisa somente de uma resposabilidade para existir, muitas responsabilidades em uma classe pode gerar problemas futuros.

1. Quais problemas podem surgir quando uma classe possui múltiplas responsabilidades?

R:

**Exercício prático**  
Uma classe Funcionario realiza múltiplas operações: calcula salário, gera relatório de desempenho e envia e-mails de aniversário.

class Funcionario {

void calcularSalario() { /\* ... \*/ }

void gerarRelatorioDesempenho() { /\* ... \*/ }

void enviarEmailAniversario() { /\* ... \*/ }

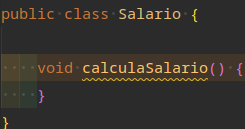
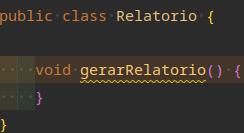
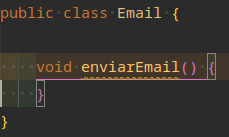
}

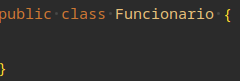
Responda:

1. Quantas responsabilidades essa classe possui?

R: 3

1. Reestruture-a em classes separadas, cada uma com uma única responsabilidade.





## **2. OCP – Open/Closed Principle**

1. O que significa uma classe estar "aberta para extensão e fechada para modificação"?

R: significa nao mudar o core da funcionalidade dela e sim adicionar uma funcionaldade

1. Cite um exemplo de sistema onde esse princípio seria essencial.

R: navegador e extensoes

**Exercício prático**

class CalculadoraDesconto {

double calcular(String tipoCliente, double valor) {

if (tipoCliente.equals("VIP")) return valor \* 0.8;

if (tipoCliente.equals("Estudante")) return valor \* 0.9;

return valor;

}

}

1. Onde o OCP é violado?

R: Na criacao de ifs para adaptar para novos clientes, ao inves de adicionar ele modificou o metodo calcular

1. Reestruture o código de forma que seja fácil adicionar novos tipos de cliente sem modificar a classe principal.

## **3. LSP – Liskov Substitution Principle**

**Perguntas teóricas**

1. O que significa "subtipos devem ser substituíveis por seus tipos base"?
2. Quais sinais indicam que o LSP está sendo violado em uma hierarquia de classes?

**Exercício prático**

class Ave {

void voar() { /\* ... \*/ }

}

class Pato extends Ave { /\* voa normalmente \*/ }

class Pinguim extends Ave { /\* não voa \*/ }

Responda:

1. Qual é o problema com essa hierarquia?

R:

1. Refaça a hierarquia respeitando o LSP.

## **4. ISP – Interface Segregation Principle**

1. Por que interfaces grandes e genéricas podem prejudicar a reutilização do código?

R: porque as classes nao sao obrigadas a implementar todos os metodos que a interface oferece

1. Como o ISP ajuda a reduzir o acoplamento entre classes?

R: Separando os metodos em diversas interfaces

**Exercício prático**

interface Impressora {

void imprimir();

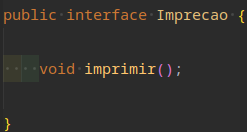
void escanear();

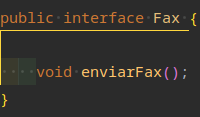
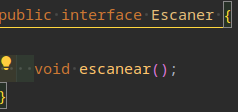
void enviarFax();

}

1. Quais problemas essa interface pode causar em classes que só imprimem (por exemplo, uma impressora térmica)?

R: Porque ela nao envia fax, nem escanea, estaria apenas ocupando memoria dentro da classe ImpressoraTermica() por exemplo

1. Reescreva as interfaces de forma mais coesa, aplicando o ISP.



## **5. DIP – Dependency Inversion Principle**

1. Qual a diferença entre depender de classes concretas e depender de abstrações?

R:

1. Como o DIP facilita a manutenção e a testabilidade de um sistema?

R:

**Exercício prático**

class EmailService {

void enviar(String mensagem) {

System.out.println("Enviando e-mail: " + mensagem);

}

}

class Notificacao {

private EmailService email = new EmailService();

void notificar(String mensagem) {

email.enviar(mensagem);

}

}

Responda:

1. Onde está a violação do DIP?

R:

1. Reestruture o código para que a classe Notificacao dependa de uma abstração, permitindo trocar o tipo de notificação (e-mail, SMS, push).