# Princípios em Projeto de Software

## Atividades de Aprendizado e Avaliação

**Aluno: João Marcos** **RA: 2264501**

**Data: 02/04/2023**

## Use esta cor no seu texto

1. Considerando o texto no link “Inversão de Controle & Injeção de Dependência”,

    a) A inversão de controle pode ser entendida como a mudança do conhecimento que uma classe tem em relação à outra.

    b) Na primeira versão da classe VendaDeProduto, o problema é o acoplamento que essa classe tem em relação à classe Log.

    c) Abrir o código fonte da classe VendaDeProduto para mudar o nome do arquivo de log? Comente (3 a 5 linhas)

    R: A classe VendaDeProduto não deveria ter como responsabilidade cuidar dos arquivos de log, e se modificarmos a classe Log teríamos de modificar em todas as outras classes que utilizassem a classe Log.

    d) O que a classe VendaDeProduto sabe sobre a classe Log? Comente (2 a 3 linhas)

    R: A classe VendaDeProduto não deveria saber nada sobre a classe log, apenas usa-la para fazer o log. Mas no exemplo dado a classe de venda cria e até mesmo dá o nome do arquivo que será salvo o log.

    e) A injeção de dependencia se dá pela mudança na estrutura do código, de modo que as responsabilidades passam a ser da classe Log. Assim, a classe VendaDeProduto não mais necessita de conhecimento sobre a instanciação da classe Log

    f) No padrão “*Constructor Injection*” as dependências são injetadas via construtor.

    g) A Inversão de controle torna possível e simples a escrita e execução de código que segue o princípio da injeção de dependência.

    h) A inserção de uma dependência definindo os serviços da classe Log reduziria ainda mais o acoplamento.

2. Considerando o conteúdo do vídeo “SOLID fica FÁCIL com Essas Ilustrações”

    a) Porque o ROBO MULTIFUNCIONAL quebra o princípio “S” do SOLID? Comente (2 a 3 linhas)

R: Porque esse robô faz diversas atividades distintas, e o principio de Single Responsability consiste em cada entidade/módulo ter uma única função/responsabilidade.

b) Com unidades independentes e isoladas você consegue

   i) Reaproveitar o código mais facilmente.

   ii) Refatorar o código mais facilmente.

   iii) Fazer testes automatizados.

   iv) Menos bugs, e mesmo que gere bugs você consegue isolar e consertar onde está o problema

c) Em um software com alto acoplamento, basta um componente no lugar errado para manchar todo o sistema com algum mal comportamento

d) O nome da função ou componente deve expressar tudo o que ele faz.

e) O princípio Open/Closed prescreve que deve ser possível adicionar novas funcionalidades sem alterar a camada de abstração principal.

f) No princípio Open/Closed, a classe deve estar aberta para extensão mas fechada para modificação.

g) Uma forma de garantir a extensão sem quebrar o princípio Open/Closed se dá pelo conceito de Plugins.

h) Respeitar o Princípio de Liskov força fazer abstrações no nível certo e ser mais consistente.

i) O exemplo do “pinguim” demonstra a abstração errada do Princípio da Substituição de Liskov. A abstração “Ave” não está correta, pois nem toda ave voa.

j) O Princípio da Interface Segregation promove a especificação de interfaces fragmentadas e mais especificas.

k) O Princípio da Injeção de Dependências defende que uma classe/módulo não deve depender diretamente de outra classe/módulo, mas sim dos serviços que este último oferece.

l) No contexto do *Dependency Injection Principle* a classe depende dos serviços definidos em uma interface, ou seja, ela não possui acoplamento com a classe que faz a implementação dos serviços, desconhecendo sua existência.

m) Os princípios SOLID foram especificados em 1996 por Robert C. Martin.