Relatório Explicativo dos Exemplos do Padrão SOLID

1. **Princípio da Responsabilidade Única (SRP)**

   Incorreto: A classe ‘Usuario’ lida tanto com o usuário quanto com os e-mails, e o padrão do Princípio de Responsabilidade Única é que cada classe precisa ter apenas uma responsabilidade. Como essa classe possui mais de uma responsabilidade ela não implementa o padrão.

   Correto: A classe ‘Usuario’ lida apenas com o usuário. A classe ‘UsuarioEmail’ lida apenas com os e-mails do usuário. A classe ‘UsuarioRepository’ lida apenas com a inserção do usuário no Banco.Por isso, essas classes são as corretas e implementam o padrão.

2. **Princípio Aberto/Fechado (OCP)**

   Incorreto: A classe ‘CalculadoraDesconto’ fere o Princípio Aberto/Fechado, pois para adicionar novos tipos de desconto é necessário alterar o código existente dentro da classe (adicionando if ou else). Isso viola o conceito de que uma classe deve estar aberta para extensão (novos comportamentos) mas fechada para modificação.

   Correto: É usada uma interface ‘Desconto’ e classes específicas para cada tipo de desconto, classe ‘DescontoEstudante’ e classe ‘DescontoIdoso’. Assim, para adicionar novos descontos basta criar uma nova classe que implemente a interface, sem mudar a classe ‘CalculadoraDesconto’. O código principal não precisa ser reescrito, respeitando o OCP.

3. **Princípio da Substituição de Liskov (LSP)**

   Incorreto: A classe ‘Pinguim’ estende ‘Ave’ mas não consegue cumprir o contrato da classe base (voar). Isso gera incoerência: um Pinguim é uma Ave, mas não pode voar como as outras. Isso quebra o LSP, pois uma subclasse não pode violar o comportamento esperado da superclasse.

   Correto: A classe ‘Ave’ define apenas comportamentos comuns a todas as aves (comer). Aves voadoras têm voar() em uma subclasse ‘AveVoadora’. Já ‘Pinguim’ só implementa o que faz sentido para ele (comer). Assim, qualquer ave pode ser substituída sem quebrar o contrato. Isso mantém o princípio de substituição de Liskov.

4. **Princípio da Segregação de Interface (ISP)**

   Incorreto: A interface ‘Trabalhador’ força todos os implementadores a terem métodos trabalhar() e comer()’. Isso é um problema para o ‘Robo’, pois um robô não precisa comer. Assim, o robô é forçado a implementar um método que não faz sentido, o que viola o ISP.

   Correto: ‘Trabalhavel’ define trabalhar(), e ‘Alimentavel’ define comer(). Assim, o ‘Trabalhador’ implementa ambas, pois faz sentido para ele, enquanto ‘Robo’ implementa apenas ‘Trabalhavel’. Cada classe implementa somente o que faz sentido para ela, respeitando o ISP.

5. **Princípio da Inversão de Dependência (DIP)**

   Incorreto: ‘Interruptor’ depende diretamente da classe ‘Lampada’. Isso viola o DIP, pois uma classe de alto nível, ‘Interruptor’, depende de uma implementação concreta, ‘Lampada’. Assim, o sistema fica rígido e difícil de estender: se quiser usar ‘Ventilador’, por exemplo, teria que reescrever ‘Interruptor’.

   Correto: ‘Interruptor’ depende de uma abstração, ‘Dispositivo’, e ‘Lampada’ implementa essa abstração. Assim, o ‘Interruptor’ pode controlar qualquer dispositivo que implementa a interface ‘Dispositivo’. A dependência é invertida: a classede alto nível depende de abstração, não da implementação. Isso deixa o sistema flexível, desacoplado e extensível, cumprindo o DIP.