Lista - Java
Questões que não envolvem código

Aluna: cxm@cin.ufpe.br

[Q2] R:

Classes concretas:

Quando usar: As classes concretas são apropriadas quando os tipos de veículos têm comportamentos específicos e podem ser totalmente instanciados por si mesmos.

Justificativa: Cada tipo de veículo (carro, moto ou bicicleta) pode ter características e comportamentos diferentes em uma locadora de automóveis. Por exemplo, carros podem ter portas e bagagens, enquanto motos podem ter cilindradas, e bicicletas podem ter marchas. Esses atributos específicos podem ser aplicados diretamente a cada tipo de veículo graças às classes concretas.

Classes abstratas:

Quando usar: Há diferenças que devem ser consideradas, mas classes abstratas são apropriadas quando há um conjunto de comportamentos comuns entre os tipos de veículos.

Justificativa: Por exemplo, todos os veículos, sejam carros, motos ou bicicletas, podem ter comportamentos semelhantes, como calcular a quilometragem ou verificar sua disponibilidade. Mas os detalhes individuais podem variar. Por exemplo, os cálculos de quilometragem para carros (com base no odômetro) e motos (com base na cilindrada) podem ser implementados de maneiras diferentes. Os métodos comuns podem ser definidos pela classe abstrata, enquanto os detalhes específicos podem ser reservados para as classes concretas.

Interface:

Quando usar: As interfaces são úteis quando você deseja estabelecer um contrato comum para vários tipos de veículos, independentemente de suas semelhanças ou diferenças.

Justificativa: Uma interface pode ser a melhor opção para as locadoras que desejam implementar certas rotinas ou funções para todos os tipos de carros, como calcular

preços de aluguel e verificar disponibilidade. Apenas a estrutura dos métodos é descrita pelas interfaces (sua assinatura), sem as implementações. Cada tipo de veículo implementaria então essas técnicas de acordo com suas especificações.

[EXTRA] R:

Esses conceitos em programação orientada a objetos são essenciais para a criação de código eficiente, legível e fácil de manter.

A *coesão* é o estado em que os componentes de uma classe estão conectados e cooperam para atingir um objetivo comum. Em classes altamente coesas, os componentes (métodos e atributos) estão intimamente relacionados e trabalham para o mesmo objetivo. Geralmente, isso resulta em um código mais organizado e fácil de entender. A coesão alta é ideal porque facilita a extensão e manutenção do código.

O termo *"acoplado"* refere-se à relação de dependência entre os componentes de um sistema. Isso mostra o grau de interações entre classes ou módulos. Um acoplamento baixo é bom porque as classes são independentes umas das outras e podem ser alteradas sem afetar outros componentes do sistema. Pode ser mais difícil manter e alterar o código com um alto acoplamento, pois as mudanças em uma classe podem ter efeitos cascata em outras classes.

O princípio do *encapsulamento* permite que uma interface interaja com um objeto enquanto oculta seus detalhes internos. Em termos práticos, isso significa que os atributos de uma classe devem ser privados e só podem ser acessados por métodos públicos, ou getters e setters. Isso permite uma administração mais precisa do processamento de dados e evita acesso direto e não permitido. Além de manter um código mais forte, o encapsulamento garante que os dados sejam seguros e intactos.

Coesão e Encapsulamento: Geralmente, classes altamente coesas também têm um bom encapsulamento. Uma classe bem planejada geralmente mantém seus membros encapsulados e fornece métodos públicos de interação. Isso garante que a classe mantenha um estado interno consistente.

Acoplamento e Encapsulamento: Ao ocultar os detalhes internos de uma classe, reduzimos a dependência de outras partes do sistema em relação a esses detalhes. Isso geralmente resulta em um acoplamento menor. Isso torna mais fácil manter e alterar uma classe sem afetar outras partes de todo o sistema.

Coesão e Acoplamento: Ambos os conceitos podem estar em desacordo. Por vezes, é preciso equilibrar acoplamento e coesão. Por exemplo, para facilitar a integração de diferentes componentes em um sistema altamente acoplado, pode ser necessário sacrificar um pouco da coesão.

Em resumo, o objetivo do projeto de código orientado a objetos é atingir um alto nível de coesão, um baixo nível de acoplamento e um bom encapsulamento. Esses princípios ajudam a criar código mais eficiente, legível, extensível e fácil de manter.