Programação Orientada a Objeto - Java

POO - Java

V1.0.2 (DEC/2023)

Sumário

1 POO - Java

1.1 Introdução

A programação orientada a objetos (POO) é um paradigma de programação que se baseia no conceito de “objetos”, que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos. Java é uma linguagem de programação fortemente orientada a objetos.

2 POO - Java

2.1 Abstração

A abstração é um dos princípios fundamentais da programação orientada a objetos (POO) e é uma maneira de simplificar e representar entidades do mundo real em um sistema de software. Ela é usada para modelar objetos do mundo real em termos de suas características essenciais, ignorando os detalhes menos importantes. Na programação orientada a objetos, a abstração é alcançada por meio de classes e objetos. Ao usar abstração, você modela objetos em seu sistema de software de forma a representar apenas as características e comportamentos essenciais para o propósito do seu sistema.

2.2 Classes e Objetos

Em Java, uma classe é uma estrutura fundamental que serve como modelo para criar objetos.

Exemplo de declaração de classe em Java:

```java
public class Carro {
    // Atributos
    String modelo;
    int ano;

    // Métodos
    void acelerar() {
        // Lógica para acelerar o carro
    }
}
```

- **Classe Abstrata:** Não pode ser instanciada. Só pode servir como progenitora.
- **Método Abstrato:** Declarado, mas não implementado na progenitora.
- **Classe Final:** Não pode ser herdada por outra classe. Obrigatoriamente.
- **Método Final:** Não pode ser sobrescrito pelas suas sub-classes. Obrigatoriamente herdado.

Objeto: Um objeto é uma instância de uma classe. Pode ser criado usando o operador new.

Exemplo de criação de um objeto:

```java
Carro meuCarro = new Carro();
```

2.3 Encapsulamento

O encapsulamento é um dos princípios fundamentais da programação orientada a objetos (POO) e refere-se à ideia de agrupar os dados (atributos) e os métodos que operam nesses dados em uma única unidade, chamada classe. Além disso, o encapsulamento implica em controlar o acesso aos membros da classe, permitindo que apenas certos métodos (métodos getters e setters, por exemplo) tenham acesso direto aos atributos da classe. Isso ajuda a proteger os dados e a implementação interna da classe, promovendo um design mais modular e robusto.

Em Java, o encapsulamento é implementado usando modificadores de acesso, que controlam a visibilidade dos membros da classe. Os principais modificadores de acesso são:

- **public:** O modificador public em Java é utilizado para declarar membros (atributos, métodos, classes) que são acessíveis globalmente, sem restrições de pacote ou herança. Membros marcados como public podem ser acessados de qualquer lugar no código Java, incluindo classes de outros pacotes. Não há limitações quanto ao escopo de acesso desses membros. Pode ser observado fora do pacote.
- **private:** O modificador private em Java é utilizado para declarar membros (atributos, métodos, etc.) que só são acessíveis dentro da própria classe onde foram declarados. Essa restrição promove o encapsulamento, impedindo o acesso direto de outras classes, mesmo que estejam no mesmo pacote ou sejam subclasses. A palavra-chave private é usada na declaração para indicar que o acesso está restrito à própria classe.
- **protected:** O modificador protected em Java é utilizado para controlar o acesso a membros (atributos, métodos, etc.) de uma classe. Suas principais características são:
- Acesso na mesma classe: Os membros protected são acessíveis dentro da mesma classe onde são declarados.
- Acesso em subclasses (herança): Os membros protected são acessíveis em classes que herdam da classe onde o membro foi declarado.
- Acesso no mesmo pacote: Os membros protected são acessíveis por classes no mesmo pacote, independentemente de serem subclasses ou não.
- Não acesso fora do pacote (exceto por herança): Os membros protected não são acessíveis fora do pacote, a menos que a classe que está tentando acessá-los seja uma subclasse da classe que contém os membros protected.

Isso permite a criação de hierarquias de classes onde certos membros devem ser visíveis para as subclasses, mas restritos em escopo fora desse contexto.

- **default:** O modificador de acesso padrão em Java, também chamado de “default”, é utilizado quando nenhum modificador é especificado. Suas principais características são:
- Acesso no Mesmo Pacote:
- Os membros com o modificador padrão são acessíveis apenas dentro do mesmo pacote em que foram declarados. Sem Acesso Fora do Pacote:
- Membros com o modificador padrão não podem ser acessados por classes que estão fora do pacote onde foram declarados. Automático se Nenhum Modificador For Especificado:

Se nenhum modificador for especificado, o modificador padrão é aplicado automaticamente. Essa restrição limita o acesso aos membros para o contexto do pacote em que estão declarados, proporcionando encapsulamento dentro desse escopo.

Use modificadores de acesso (public, private, protected) para controlar o acesso aos membros da classe.

O encapsulamento ajuda a garantir que o estado interno de um objeto seja manipulado de maneira controlada, promovendo a segurança e facilitando futuras mudanças na implementação sem afetar o código cliente.

2.4 Herança

A herança é um conceito fundamental na programação orientada a objetos (POO) que permite a criação de uma nova classe, chamada classe derivada ou subclasse, que herda características (atributos e métodos) de uma classe existente, chamada classe base ou superclasse. A ideia é reutilizar o código existente e estabelecer uma relação hierárquica entre as classes.

- A herança será aplicada tanto para as características quanto para os comportamentos.

A herança é uma ferramenta poderosa, mas é importante usá-la com cuidado para evitar hierarquias profundas e complexas. O princípio do “é um” deve ser respeitado; ou seja, a relação de herança deve fazer sentido semântico.

- **Herança de implementação:** situação em que uma classe derivada (subclasse) herda a implementação (código) dos métodos de uma classe base (superclasse).
- **Herança para diferença:** refere-se a uma abordagem na qual uma classe derivada (subclasse) herda uma classe base (superclasse) e, em seguida, modifica ou estende sua funcionalidade. Em outras palavras, a subclasse mantém parte da implementação da superclasse, mas também adiciona ou substitui comportamentos específicos.

2.5 Polimorfismo

O polimorfismo é um conceito fundamental na programação orientada a objetos (POO) que permite que objetos de diferentes classes sejam tratados de maneira uniforme quando compartilham uma mesma interface ou herança. A palavra “polimorfismo” vem do grego, onde “poly” significa muitos e “morphe” significa formas. Em POO, isso significa que um mesmo método pode ter diferentes implementações, dependendo do contexto em que é chamado. O polimorfismo permite escrever código mais flexível e extensível, pois você pode tratar objetos de maneira mais geral, utilizando interfaces ou classes base, e ainda assim chamar os métodos específicos de suas subclasses.

Assinatura do método: Quantidade e os tipos dos parâmetros.

- **Polimorfismo de Sobreposição:** O polimorfismo de sobreposição (ou polimorfismo de método) é uma forma específica de polimorfismo em que uma subclasse fornece uma implementação específica para um método que já está definido em sua superclasse. O método na subclasse deve ter a mesma assinatura (nome, tipo de retorno e tipos de parâmetros) que o método na superclasse. Esse é um conceito essencial para a aplicação de princípios como o polimorfismo em programação orientada a objetos.

(Mesma assinatura, classes diferentes, É SOBREPOSIÇÃO)

- **Polimorfismo de Sobrecarga:** O polimorfismo de sobrecarga ocorre quando uma classe possui múltiplos métodos com o mesmo nome, mas com diferentes listas de parâmetros. Isso permite que você tenha métodos com o mesmo nome, mas que realizam operações diferentes com base nos parâmetros que recebem. Esse tipo de polimorfismo é resolvido em tempo de compilação, pois é baseado na assinatura dos métodos (quantidade ou tipos de parâmetros). O polimorfismo de sobrecarga é útil para fornecer funcionalidades similares com interfaces diferentes ou tipos diferentes de dados, simplificando a chamada de métodos e melhorando a legibilidade do código.

(Assinaturas diferentes, Mesma classe, É SOBRECARGA)

3 Change log

3.1 Version 1.0.2

- Documento sobre POO - Java