CLASSES ABSTRATAS EM JAVA

No universo da programação orientada a objetos, as classes abstratas desempenham um papel crucial. Elas oferecem uma estrutura para definir comportamentos que devem ser compartilhados entre várias subclasses, ao mesmo tempo que permitem que cada classe derivada implemente suas próprias versões desses comportamentos.

Em essência, uma classe abstrata serve como uma "base" que estabelece contratos para métodos que devem ser implementados por suas subclasses, garantindo consistência e coerência em hierarquias complexas de classes. Ao mesmo tempo, elas permitem a flexibilidade necessária para que as classes filhas personalizem e adaptem os métodos abstratos às suas necessidades específicas.

Em Java, uma classe abstrata é declarada usando a palavra-chave `abstract`. Aqui está um exemplo:

```
public abstract class Animal {
    protected String nome;

    public Animal(String nome) {
        this.nome = nome;
    }

    // Método concreto
    public void dormir() {
        System.out.println(nome + " está dormindo.");
    }

    // Método abstrato
    public abstract void fazerSom();
}

// Subclasse concreta
public class Cachorro extends Animal {
    public Cachorro(String nome) {
        super(nome);
    }

    // Implementação do método abstrato
    @Override
    public void fazerSom() {
        System.out.println(nome + " faz Au Au!");
    }
}

// Uso
public class Main {
    public static void main(String[] args) {
        // Animal animal = new Animal("Bob"); // Isso causaria um
erro de compilação
        Cachorro cachorro = new Cachorro("Rex");
```

```
        cachorro.dormir();    // Saída: Rex está dormindo.
        cachorro.fazerSom();  // Saída: Rex faz Au Au!
    }
```

Neste exemplo:

1. `Animal` é uma classe abstrata que define uma estrutura comum para diferentes tipos de animais.
2. Ela tem um método concreto `dormir()` e um método abstrato `fazerSom()`.
3. `Cachorro` é uma subclasse concreta que estende `Animal` e implementa o método abstrato `fazerSom()`.
4. Não é possível instanciar diretamente a classe `Animal`, mas podemos criar objetos de suas subclasses concretas.
5. A classe abstrata permite definir um comportamento comum (como `dormir()`) e forçar as subclasses a implementar comportamentos específicos (como `fazerSom()`).

Classes abstratas em Java são usadas para criar uma estrutura base para outras classes, promovendo reutilização de código e garantindo que certas funcionalidades sejam implementadas nas subclasses.

REGRAS PARA UTILIZAR CLASSES ABSTRATAS

1. Somente métodos de instância podem ser marcados como abstratos

- **Explicação**: Um método abstrato é aquele que não tem corpo, ou seja, ele é declarado, mas não implementado. Em Java, você só pode marcar um método de instância (ou seja, um método que pertence a uma instância de uma classe) como abstrato. Não é permitido que variáveis, construtores ou métodos `static` sejam abstratos.

Exemplo:

```
// Correto: Método de instância abstrato
abstract class Animal {
    abstract void fazerSom(); // Método abstrato
}

// Incorreto: Tentando marcar um método estático como abstrato
abstract class AnimalIncorreto {
    abstract static void fazerSom(); // Isso causaria erro de compilação
}
```

2. Métodos abstratos só podem ser declarados em classes abstratas

- **Explicação**: Uma classe abstrata é uma classe que não pode ser instanciada diretamente, ou seja, você não pode criar objetos dela. Apenas dentro de uma classe abstrata é possível declarar métodos abstratos. Se você tentar declarar um método abstrato em uma classe que não seja abstrata, vai dar erro.

Exemplo:

```
// Correto: Método abstrato dentro de uma classe abstrata
abstract class Animal {
    abstract void fazerSom(); // Método abstrato
}

// Incorreto: Tentando declarar um método abstrato em uma classe
não abstrata
class AnimalIncorreto {
    abstract void fazerSom(); // Isso causaria erro de compilação
}
```

3. Classes não abstratas que estendem classes abstratas devem implementar todos os métodos herdados

- **Explicação**: Quando uma classe não abstrata (ou seja, uma classe que pode ser instanciada) herda de uma classe abstrata, ela é obrigada a implementar todos os métodos abstratos dessa classe. Se ela não implementar, também precisará ser declarada como abstrata. E sendo declarada abstrata, a classe concreta que a extender deverá implementar todos os métodos de ambas as classes abstratas.

Exemplo:

```
abstract class Animal {
    abstract void fazerSom(); // Método abstrato
}

// Correto: A classe Cachorro implementa o método abstrato
class Cachorro extends Animal {
    void fazerSom() {
        System.out.println("O cachorro late!");
    }
}

// Incorreto: A classe Gato não implementa o método abstrato
class Gato extends Animal {
    // Se você não implementar o método fazerSom(), terá que
declarar a classe Gato como abstrata
}
```

4. Todas as outras regras para sobrescrever métodos se aplicam

- **Explicação**: Quando uma classe não abstrata sobrescreve um método abstrato, ela precisa seguir as mesmas regras que se aplicam a qualquer outro método sobrescrito em Java. Por exemplo, a assinatura do método deve ser a mesma, e você não pode tornar o método menos acessível (por exemplo, mudar de `public` para `protected`).

Exemplo:

```
abstract class Animal {
    abstract void fazerSom(); // Método abstrato
}

class Cachorro extends Animal {
    @Override
```

```
    public void fazerSom() { // Correto: mesmo nível de acesso ou
maior
        System.out.println("O cachorro late!");
    }

    // Incorreto: Menor nível de acesso
    // protected void fazerSom() { // Isso causaria erro de
compilação
    //     System.out.println("O cachorro late!");
    // }
}
```

Mais alguns pontos importantes sobre classes abstratas:

As classes abstratas são semelhantes às classes comuns em muitos aspectos, inclusive na possibilidade de **terem construtores**. No entanto, há uma diferença crucial: os **construtores de classes abstratas só podem ser chamados através do método especial "super"** nas classes filhas. Isso ocorre porque não é possível criar instâncias diretas de classes abstratas.

É fundamental entender que classes ou **métodos abstratos não podem ser marcados como "final"**. Isso faz sentido, pois as classes abstratas são projetadas para serem estendidas, e os métodos abstratos para serem sobrescritos. Marcar qualquer um deles como "final" iria contra sua própria natureza.

Da mesma forma, **métodos abstratos não podem ser privados**. A ideia de um método abstrato é que ele seja implementado nas subclasses, o que seria impossível se fosse privado.

Por fim, a **combinação de "abstract" e "static" não é permitida para métodos**. Métodos estáticos não podem ser sobrescritos, apenas ocultados, o que contradiz o propósito de um método abstrato.

Mesmo que uma classe seja abstrata, **ela não precisa ter apenas métodos abstratos**. Na verdade, ela pode ter métodos com implementação concreta (ou seja, métodos normais, com corpo). Isso permite que você defina um comportamento padrão para as classes filhas, que podem optar por usar ou sobrescrever esse comportamento.

Nem toda classe abstrata precisa necessariamente ter métodos abstratos. Você pode ter uma classe abstrata que serve apenas para fornecer uma base comum para outras classes, sem que ela mesma defina métodos abstratos.

Uma **classe abstrata pode herdar de outra classe abstrata**. Nesse caso, a classe filha abstrata **não precisa implementar os métodos abstratos da classe mãe**; ela pode deixar essa tarefa para uma subclasse concreta mais abaixo na hierarquia. Neste caso tanto todos os metodos das classes abstratas (mãe e filha) devem ser implementados.

Classes abstratas são muito úteis em design de software, especialmente quando você está trabalhando com herança. Elas permitem que você defina uma interface comum para todas as subclasses, garantindo que todas as subclasses sigam uma estrutura específica, enquanto ainda permite que elas implementem detalhes específicos por conta própria.

Isso é muito comum em frameworks e bibliotecas, onde você tem uma arquitetura bem definida e espera que os desenvolvedores preencham certas lacunas.