Exercícios de Polimorfismo em Java

Exercício 1: Classe `Animal` e o Método `emitirSom()`

Conceitos Focados: Sobrescrita de método (*Method Overriding*) e polimorfismo simples.

1. Crie uma **classe abstrata** chamada `Animal`.
2. Nessa classe, declare um **método abstrato** chamado `emitirSom()` que não recebe parâmetros e não retorna nada.
3. Crie as classes `Cachorro` e `Gato` que **herdam** de `Animal`.
4. Em cada subclasse, **sobrescreva** o método `emitirSom()` para que:
    - `Cachorro` imprima: "Au Au!"
    - `Gato` imprima: "Miau!"
5. No método `main`, crie um `Cachorro` e um `Gato`.
6. Em seguida, crie um **array** ou **lista** de objetos do tipo `Animal` e adicione o cachorro e o gato.
7. Itere sobre a lista de `Animal` e chame o método `emitirSom()` para cada um. O polimorfismo deve garantir que o som correto seja executado.

É uma ótima ideia focar em **polimorfismo**! É um conceito fundamental da Orientação a Objetos em **Java**.

Aqui estão 5 exercícios que variam em complexidade para ajudar a praticar:

Exercícios de Polimorfismo em Java

Exercício 1: Classe `Animal` e o Método `emitirSom()`

Conceitos Focados: Sobrescrita de método (*Method Overriding*) e polimorfismo simples.

1. Crie uma **classe abstrata** chamada `Animal`.
2. Nessa classe, declare um **método abstrato** chamado `emitirSom()` que não recebe parâmetros e não retorna nada.
3. Crie as classes `Cachorro` e `Gato` que **herdam** de `Animal`.
4. Em cada subclasse, **sobrescreva** o método `emitirSom()` para que:
    - `Cachorro` imprima: "Au Au!"
    - `Gato` imprima: "Miau!"

5. No método `main`, crie um `Cachorro` e um `Gato`.
6. Em seguida, crie um **array** ou **lista** de objetos do tipo `Animal` e adicione o cachorro e o gato.
7. Itere sobre a lista de `Animal` e chame o método `emitirSom()` para cada um. O polimorfismo deve garantir que o som correto seja executado.

Exercício 2: Classe `Funcionario` e o Cálculo de Bônus

Conceitos Focados: Sobrescrita com diferentes lógicas de negócio.

1. Crie uma classe base chamada `Funcionario` com os atributos `nome` (String) e `salario` (double).
2. Adicione um método chamado `calcularBonus()` que retorna um `double`. Por padrão, na classe `Funcionario`, o bônus será **10%** do salário.
3. Crie duas subclasses: `Gerente` e `Programador`.
4. Na classe `Gerente`, **sobrescreva** `calcularBonus()` para que ele retorne **15%** do salário mais um valor fixo de R$ 1000,00.
5. Na classe `Programador`, **sobrescreva** `calcularBonus()` para que ele retorne **20%** do salário.
6. No `main`, crie uma lista que armazene objetos do tipo `Funcionario`.
7. Adicione um `Gerente` e um `Programador` à lista.
8. Itere sobre a lista e imprima o nome e o bônus calculado para cada funcionário.

Exercício 3: Interfaces Gráficas - `Desenhavel`

Conceitos Focados: Polimorfismo através de **interfaces**.

1. Crie uma **interface** chamada `Desenhavel` com um único método `void desenhar()`.
2. Crie três classes concretas que **implementam** essa interface: `Circulo`, `Quadrado` e `Linha`.
3. Em cada classe, implemente o método `desenhar()` para que ele imprima uma mensagem específica para a forma (Ex: "Desenhando um círculo colorido...").
4. Crie uma classe `AreaDeDesenho`. Esta classe deve ter um método chamado `adicionar(Desenhavel forma)` e um método `executarDesenho()` que itera sobre todas as formas adicionadas e chama o método `desenhar()` de cada uma.
5. No `main`, instancie `Circulo`, `Quadrado` e `Linha`, adicione-os à `AreaDeDesenho` e chame o método `executarDesenho()`.

Exercício 4: Polimorfismo de Inclusão e Casting

Conceitos Focados: Downcasting (*casting* para subtipo) e o operador `instanceof`.

1. Crie a classe `Veiculo` (classe base).
2. Crie as subclasses `Carro` e `Moto` que herdam de `Veiculo`.
3. Na classe `Carro`, adicione um método exclusivo chamado `ligarArCondicionado()`.
4. Na classe `Moto`, adicione um método exclusivo chamado `darGrau()`.
5. No `main`, crie um array de `Veiculo` e adicione alguns `Carro` e `Moto`.
6. Itere sobre o array. Para cada `Veiculo`:
    - Verifique se o objeto é um **`instanceof`** `Carro`. Se for, faça o *downcasting* para `Carro` e chame `ligarArCondicionado()`.
    - Se for um **`instanceof`** `Moto`, faça o *downcasting* para `Moto` e chame `darGrau()`.
    - (Opcional) Chame um método `acelerar()` que deve estar definido em `Veiculo` e sobrescrito em ambos os filhos, mostrando o polimorfismo de sobrescrita.

Exercício 5: Sobrecarga de Método (*Method Overloading*) e Coerção

Conceitos Focados: Polimorfismo de **Sobrecarga** (Overloading) e a forma como o Java escolhe o método.

1. Crie uma classe chamada `Calculadora`.
2. Nesta classe, crie o método `somar` **sobrecarregado** (*overloaded*) três vezes:
    - `int somar(int a, int b)`
    - `double somar(double a, double b)`
    - `String somar(String a, String b)` (que concatena as strings)
3. No método `main`, chame a função `somar` com diferentes tipos de argumentos e observe como o compilador decide qual método usar:
    - Chame com dois inteiros: `calculadora.somar(5, 3)`
    - Chame com dois doubles: `calculadora.somar(5.5, 3.2)`
    - Chame com duas strings: `calculadora.somar("Olá, ", "Mundo!")`
4. **Desafio:** Chame a função com um `int` e um `double` (Ex: `calculadora.somar(10, 5.5)`). Qual versão do método `somar` é chamada? Explique o comportamento (coerção de tipo).

Exercício 2: Classe `Funcionario` e o Cálculo de Bônus

Conceitos Focados: Sobrescrita com diferentes lógicas de negócio.

1. Crie uma classe base chamada `Funcionario` com os atributos `nome` (String) e `salario` (double).
2. Adicione um método chamado `calcularBonus()` que retorna um `double`. Por padrão, na classe `Funcionario`, o bônus será **10%** do salário.
3. Crie duas subclasses: `Gerente` e `Programador`.
4. Na classe `Gerente`, **sobrescreva** `calcularBonus()` para que ele retorne **15%** do salário mais um valor fixo de R$ 1000,00.
5. Na classe `Programador`, **sobrescreva** `calcularBonus()` para que ele retorne **20%** do salário.
6. No `main`, crie uma lista que armazene objetos do tipo `Funcionario`.
7. Adicione um `Gerente` e um `Programador` à lista.
8. Itere sobre a lista e imprima o nome e o bônus calculado para cada funcionário.

Exercício 6: Processamento de Pagamento

Conceitos Focados: Polimorfismo com classes abstratas e delegação de comportamento.

1. Crie uma **classe abstrata** chamada `MetodoPagamento` com um atributo `valor` (double).
2. Defina um **método abstrato** `processarPagamento()` que retorna um `boolean` (indicando sucesso ou falha).
3. Crie as subclasses `PagamentoCartaoCredito` e `PagamentoBoleto`.
4. **Sobrescreva** `processarPagamento()` em cada subclasse com uma lógica simulada:
    - `PagamentoCartaoCredito`: Retorna `true` se o `valor` for menor que R$ 5000,00. Imprima "Transação de Crédito Autorizada.".
    - `PagamentoBoleto`: Retorna `true` se o `valor` for menor que R$ 1000,00. Imprima "Boleto Gerado e Pago.".
5. Crie uma classe `Loja` com um método `finalizarCompra(MetodoPagamento metodo)`. Dentro deste método, chame `metodo.processarPagamento()` e imprima o resultado.
6. No `main`, crie instâncias de ambos os tipos de pagamento, atribua um valor diferente para cada e passe-os para o método `finalizarCompra`.

Exercício 7: Sistema de Impressão de Documentos

Conceitos Focados: Polimorfismo via Interfaces e Injeção de Dependência (simples).

1. Crie uma **Interface** chamada `Imprimivel` com um método `imprimir()` que não retorna nada.
2. Crie duas classes que implementam a interface: `DocumentoPDF` e `RelatorioFinanceiro`.
3. Em `DocumentoPDF`, o método `imprimir()` deve imprimir: "Imprimindo PDF com alta resolução.".
4. Em `RelatorioFinanceiro`, o método `imprimir()` deve imprimir: "Imprimindo Relatório em modo rascunho.".
5. Crie uma classe chamada `FilaDeImpressao`. Ela deve ter um método `adicionarDocumento(Imprimivel doc)` e um método `executarImpressao()` que itera sobre a lista de documentos (`ArrayList<Imprimivel>`) e chama o método `imprimir()` para cada um.
6. No `main`, crie uma `FilaDeImpressao`, adicione instâncias de `DocumentoPDF` e `RelatorioFinanceiro`, e chame `executarImpressao()`.

Exercício 8: Jogo de Luta Simples

Conceitos Focados: Herança e Sobrescrita de método para alterar *comportamento*.

1. Crie uma classe base chamada `Personagem` com os atributos `nome` (String) e `vida` (int).
2. Adicione um método chamado `atacar(Personagem alvo)` que diminui a vida do alvo em 10. Imprima quem atacou quem.
3. Crie as subclasses `Guerreiro` e `Mago`.
4. **Sobrescreva** o método `atacar` em `Guerreiro` para que ele diminua a vida do alvo em **15**.
5. **Sobrescreva** o método `atacar` em `Mago` para que ele diminua a vida do alvo em **5**, mas adicione uma mensagem especial (Ex: "Mago lança feitiço fraco, mas rápido!").
6. No `main`, crie instâncias de um `Guerreiro` e um `Mago`. Faça-os atacar um ao outro em uma sequência.
    - Crie uma variável do tipo `Personagem` (o polimorfismo) para simular o combate.

Exercício 9: Classe `Notificador` e Canais de Mensagem

Conceitos Focados: Interfaces com diferentes implementações de métodos.

1. Crie uma **Interface** chamada `CanalNotificacao` com um único método `void notificar(String mensagem)`.

2. Crie as classes `NotificacaoEmail`, `NotificacaoSMS` e `NotificacaoPush` que implementam a interface.
3. Implemente o método `notificar` em cada classe para que ele imprima a mensagem formatada para aquele canal:
    - `NotificacaoEmail`: "[EMAIL] Enviando: [mensagem]"
    - `NotificacaoSMS`: "[SMS] Enviando: [mensagem]"
    - `NotificacaoPush`: "[PUSH] Enviando: [mensagem]"
4. Crie uma classe `SistemaNotificacao` que possui um método `enviarTodas(String mensagem, List<CanalNotificacao> canais)`.
5. No `main`, crie uma lista de `CanalNotificacao`, adicione um objeto de cada tipo de notificação (Email, SMS, Push) e chame o método `enviarTodas` para que a mesma mensagem seja entregue de formas diferentes.