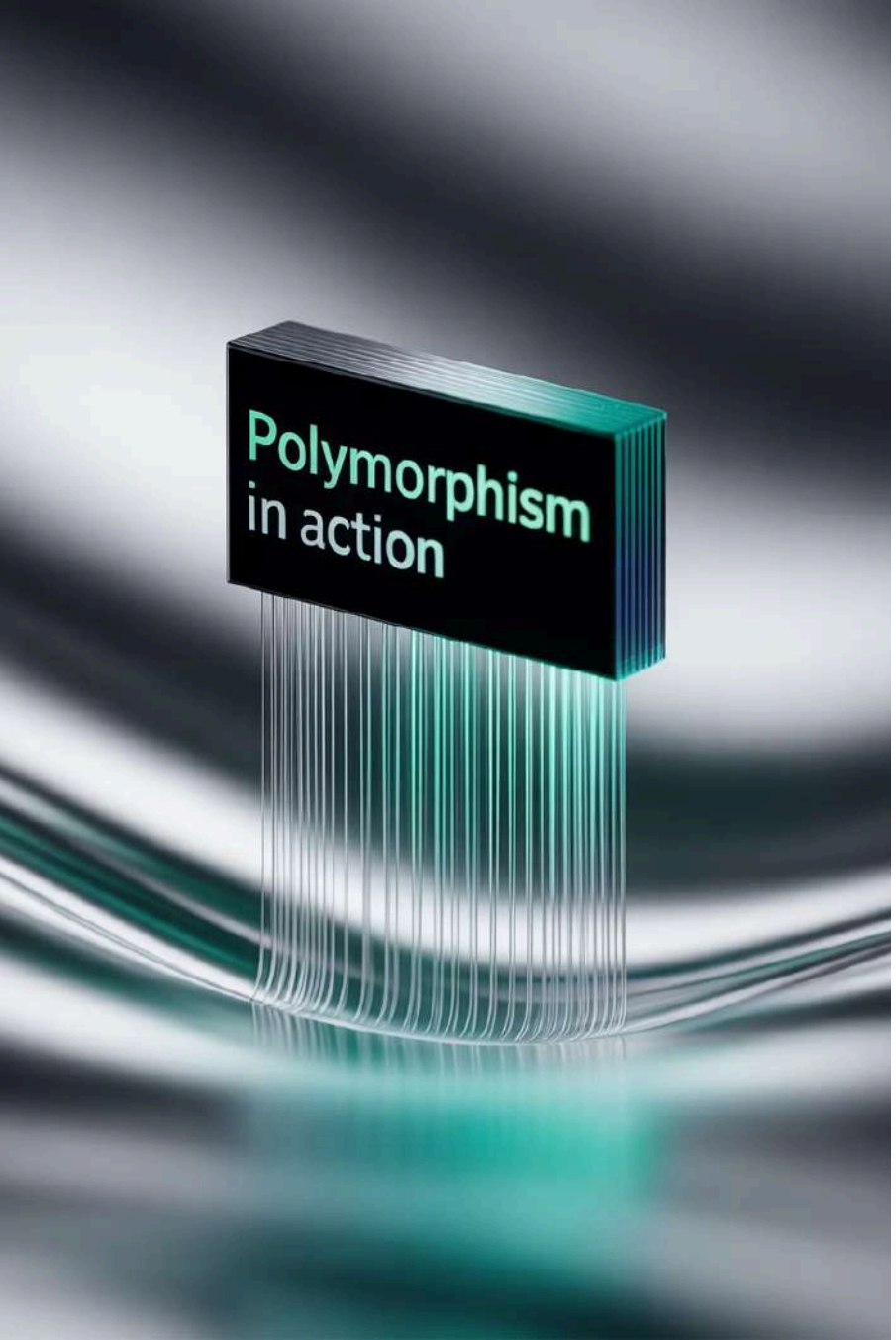


Pilares da POO – Polimorfismo

UNIVERSIDADE DO OESTE DE SANTA CATARINA - UNOESC

Ciência da Computação | Programação II

Prof.: Leandro Otavio Cordova Vieira

Objetivos da Aula

Compreender Polimorfismo

Entender o conceito fundamental do polimorfismo na programação orientada a objetos e sua aplicação prática no desenvolvimento de software.

Diferenciar Conceitos

Distinguir claramente entre sobrecarga e sobrescrita de métodos, compreendendo quando e como aplicar cada conceito.

Aplicar na Prática

Resolver exercícios práticos que demonstrem o uso do polimorfismo em cenários reais de programação.

Ao final desta aula, você será capaz de implementar polimorfismo em suas aplicações, tornando seu código mais flexível, reutilizável e elegante.

O que é Polimorfismo?

Conceito Fundamental e Etimologia

A palavra polimorfismo tem suas raízes no grego antigo, onde "poli" significa **"muitas"** e "morfos" significa **"formas"**. Literalmente, significa "ter muitas formas". Na programação orientada a objetos (POO), esse termo descreve a incrível capacidade de um objeto assumir múltiplas formas ou comportamentos, dependendo do contexto em que é utilizado.

É um dos pilares fundamentais da POO, ao lado da herança e do encapsulamento. O polimorfismo permite que objetos de diferentes classes respondam de maneira única a uma mesma mensagem ou chamada de método, sem que o código cliente precise saber o tipo exato do objeto com o qual está interagindo. Isso é alcançado através da implementação de métodos com o mesmo nome em classes relacionadas.

Polimorfismo no Mundo Real: Analogias

Imagine um controle remoto universal: ele possui botões como "Ligar", "Desligar", "Aumentar Volume". Dependendo do aparelho (TV, DVD, Som) que ele está controlando, a ação "Ligar" pode ser executada de forma diferente (iniciando o sistema operacional da TV, carregando um disco no DVD, etc.), mas a intenção e a chamada são as mesmas. Outro exemplo é uma pessoa que pode ser um "Pai", um "Funcionário", um "Amigo" – em cada papel, ela assume comportamentos diferentes.

O que é Polimorfismo?

Como Funciona Internamente?

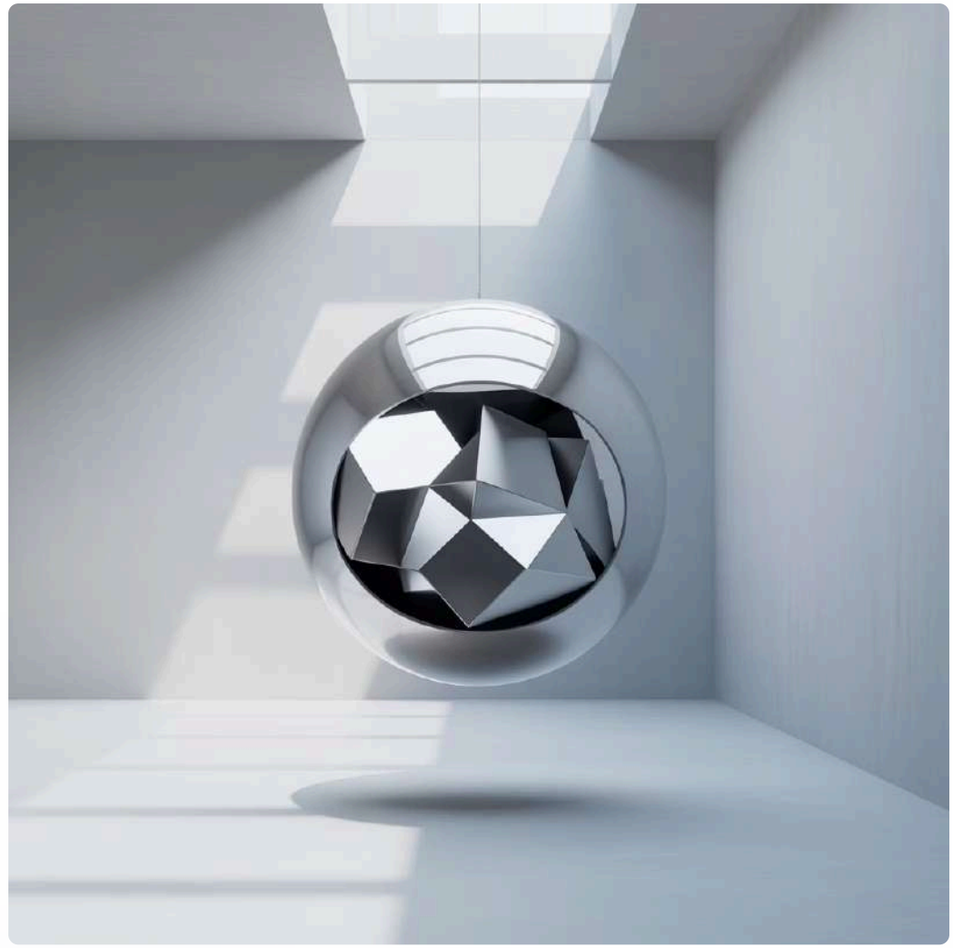
Internamente, o polimorfismo é frequentemente implementado através de mecanismos como a sobrescrita (override) de métodos. Isso significa que uma subclasse pode fornecer uma implementação específica para um método que já foi definido em sua superclasse. Quando o método é invocado em um objeto, o sistema de tempo de execução (runtime) determina qual versão do método deve ser executada com base no tipo real do objeto, e não no tipo da referência. Este processo é conhecido como late binding ou despacho dinâmico (dynamic dispatch).

Para que o polimorfismo ocorra, geralmente é necessária uma hierarquia de classes, onde uma classe base (ou interface) define um contrato (assinatura de métodos) e as classes derivadas fornecem suas próprias implementações desses métodos. O tipo da variável que referencia o objeto pode ser o tipo da classe base, mas o comportamento executado será o da subclasse concreta.

O que é Polimorfismo?

Benefícios Práticos do Polimorfismo

- **Flexibilidade e Extensibilidade:** Permite adicionar novas classes (novas "formas") sem modificar o código existente que utiliza o contrato da classe base. Isso torna o sistema mais adaptável a mudanças e novos requisitos.
- **Reutilização de Código:** Desenvolvedores podem escrever código genérico que opera sobre uma variedade de objetos relacionados, em vez de ter que escrever código específico para cada tipo de objeto.
- **Manutenibilidade:** A centralização do comportamento em hierarquias de classes simplifica a depuração e as atualizações, pois as modificações em um comportamento específico de uma "forma" afetam apenas essa subclasse.
- **Clareza e Legibilidade:** O código se torna mais intuitivo e fácil de entender, pois a mesma "mensagem" (chamada de método) tem um significado consistente, mesmo que suas implementações variem.



Por exemplo, em um sistema de processamento de formas geométricas, você pode ter uma classe base `Forma` com um método `calcularArea()`. Classes como `Circulo`, `Retangulo` e `Triangulo` herdariam de `Forma` e implementariam seu próprio `calcularArea()`. Um algoritmo que calcula a área de uma coleção de `Forma`s pode chamar `calcularArea()` em cada objeto, e o polimorfismo garante que a implementação correta seja invocada para cada tipo de forma.

> "O polimorfismo permite escrever código mais genérico e flexível, sem se preocupar com os tipos específicos dos objetos, focando no 'o quê' fazer, e não no 'como'."

Exemplo Clássico: Hierarquia Animal

Considere uma classe `Animal` com o método `emitirSom()`. As subclasses `Cachorro` e `Gato` herdam esta estrutura, mas cada uma implementa o som de forma específica.

1

Classe Animal

Define a estrutura básica com o método `emitirSom()`

2

Cachorro

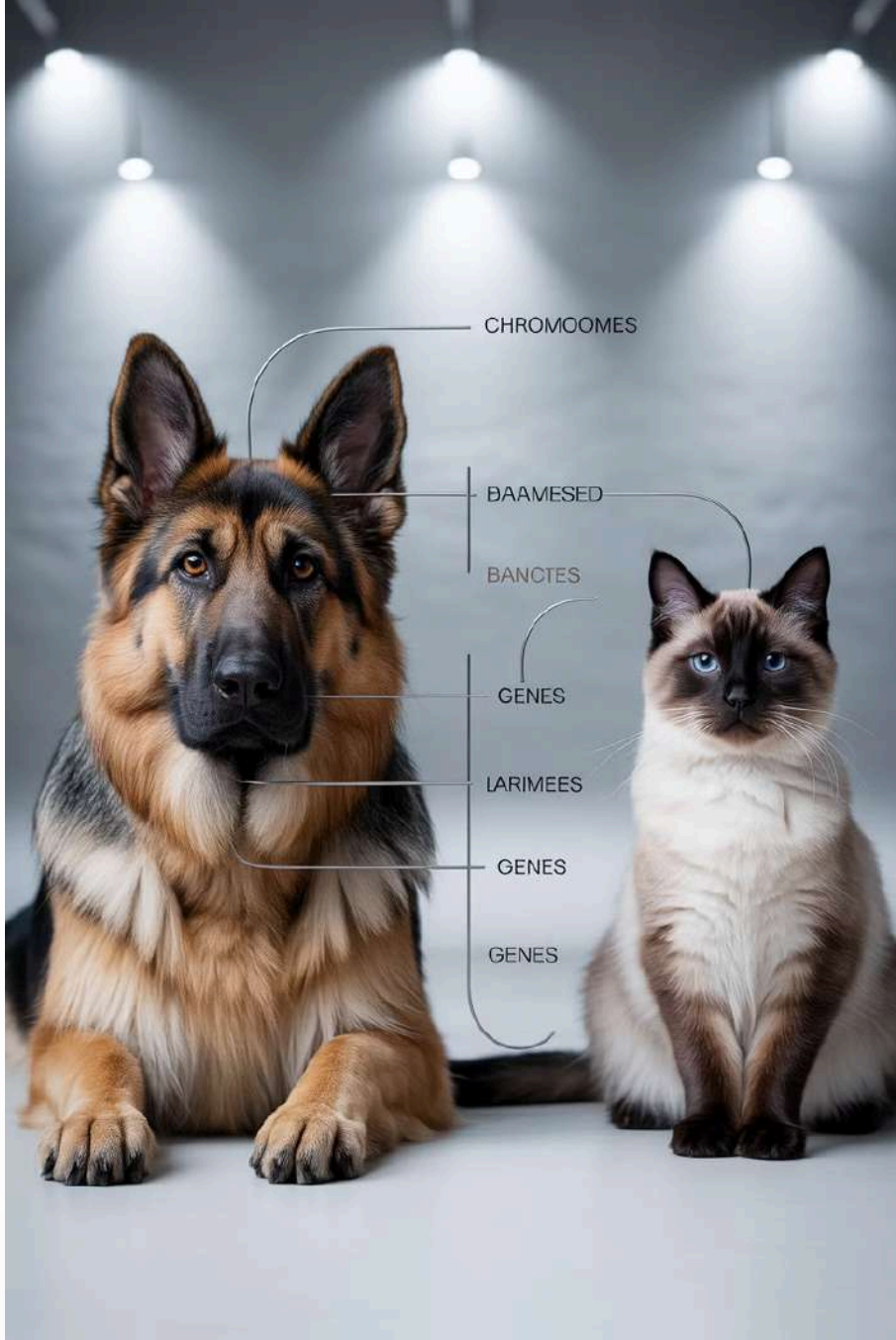
Implementa `emitirSom()` retornando "Au au!"

3

Gato

Implementa `emitirSom()` retornando "Miau!"

Este é o polimorfismo em ação: um mesmo método com comportamentos diferentes dependendo do objeto que o executa.

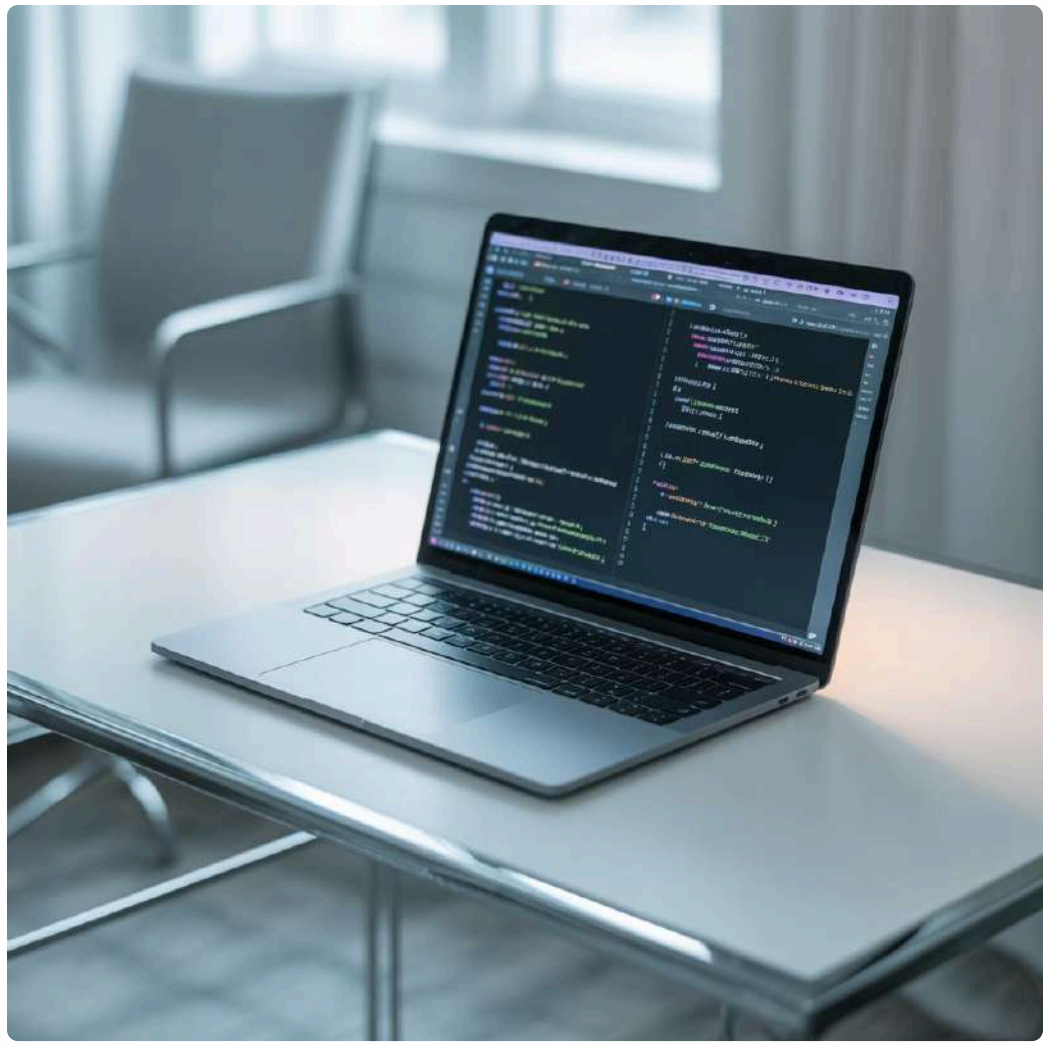


Sobrecarga de Métodos (Method Overloading)

Definição

A sobrecarga de métodos é um recurso da programação orientada a objetos que permite definir **múltiplos métodos com o mesmo nome** dentro da mesma classe, desde que suas **assinaturas sejam diferentes**. A assinatura de um método é definida pelo seu nome e pela lista de parâmetros (tipo, quantidade e ordem).

- **Mesmo nome do método:** Todos os métodos sobrecarregados compartilham o mesmo identificador.
- **Assinaturas diferentes:** A distinção ocorre através do número, tipo ou ordem dos parâmetros. O tipo de retorno do método não faz parte da assinatura para fins de sobrecarga.
- **Comportamentos específicos:** Cada versão sobrecarregada pode ter uma implementação ligeiramente diferente, adaptada aos parâmetros que recebe.
- **Facilita o uso da API:** Torna o código mais intuitivo e legível, pois permite que uma única operação seja expressa de várias maneiras.



A sobrecarga melhora a usabilidade do código, permitindo que o desenvolvedor use o método mais conveniente para cada situação, sem precisar lembrar de nomes de métodos diferentes para operações similares.

Em termos mais teóricos, a sobrecarga é uma forma de polimorfismo estático (ou polimorfismo em tempo de compilação). O compilador decide qual método sobrecarregado será chamado com base nos tipos e no número de argumentos passados durante a chamada do método. Esta decisão é feita antes da execução do programa.

Sobrecarga em PHP

Ao contrário de linguagens como Java ou C#, que permitem definir diretamente vários métodos com o mesmo nome e diferentes assinaturas (tipos e número de parâmetros), o PHP **não suporta sobrecarga de métodos de forma nativa e tradicional** em tempo de compilação. Isso significa que você não pode ter duas funções ou métodos com o mesmo nome na mesma classe ou escopo, mesmo que tenham listas de parâmetros diferentes. O PHP simplesmente sobrescreveria a primeira definição pela última. No entanto, o PHP oferece mecanismos poderosos que permitem **simular** o comportamento da sobrecarga, alcançando resultados semelhantes por meio de abordagens flexíveis e dinâmicas.

1. Parâmetros Opcionais e Valores Padrão

A forma mais comum e recomendada para simular sobrecarga em PHP é utilizando parâmetros opcionais com valores padrão. Isso permite que um método seja chamado com diferentes números de argumentos, e a lógica interna do método pode se adaptar com base nos parâmetros fornecidos.

```
class Calculadora {
    public function somar($a, $b, $c = null, $d = null) {
        $soma = $a + $b;
        if ($c !== null) {
            $soma += $c;
        }
        if ($d !== null) {
            $soma += $d;
        }
        return $soma;
    }

    public function apresentarMensagem($nome, $saudacao = "Olá") {
        return "$saudacao, $nome!";
    }
}

$calc = new Calculadora();
echo $calc->somar(5, 3); // Saída: 8
echo $calc->somar(5, 3, 2); // Saída: 10
echo $calc->somar(1, 2, 3, 4); // Saída: 10
echo $calc->apresentarMensagem("Maria"); // Saída: Olá, Maria!
echo $calc->apresentarMensagem("João", "Bom dia"); // Saída: Bom dia, João!
```

Esta abordagem é simples, legível e possui boa performance, sendo a preferida para casos onde a variação de parâmetros é previsível e limitada. A desvantagem é que a lógica condicional dentro do método pode se tornar complexa se houver muitas variações de parâmetros.

Sobrecarga em PHP

2. Método Mágico __call() para Sobrecarga Dinâmica

O PHP oferece o método mágico `__call()`, que é invocado quando um método inacessível ou inexistente é chamado em um objeto. Este é o mecanismo mais próximo da sobrecarga dinâmica que o PHP oferece, permitindo criar comportamentos que variam com base no nome do método chamado e nos argumentos passados.

```
class Logger {
    public function __call($nomeMetodo, $argumentos) {
        if (strpos($nomeMetodo, 'log') === 0) {
            $tipo = strtoupper(substr($nomeMetodo, 3)); // Extrai INFO, WARN, ERROR
            $mensagem = $argumentos[0];
            echo "[$tipo] " . date('Y-m-d H:i:s') . ": $mensagem\n";
        } else {
            throw new \BadMethodCallException("Método $nomeMetodo não existe.");
        }
    }
}

$log = new Logger();
$log->logInfo("Este é um log informativo.");
$log->logError("Ocorreu um erro crítico!");
$log->logWarn("Atenção: algo inesperado aconteceu.");
// $log->logDebug("Isso geraria um erro se não fosse tratado.");
```

O `__call()` é extremamente flexível, mas pode impactar a performance devido à sua natureza dinâmica e à necessidade de parsing do nome do método em tempo de execução. Deve ser usado com cautela e apenas quando a sobrecarga tradicional não é viável ou quando se busca um alto grau de dinamismo.

Sobrecarga em PHP

3. Usando func_get_args() e func_num_args()

Para cenários onde o número e/ou o tipo de parâmetros podem variar muito e de forma imprevisível, as funções `func_get_args()` e `func_num_args()` permitem inspecionar os argumentos passados para uma função ou método em tempo de execução. Isso pode ser combinado com a verificação de tipo para simular uma sobrecarga mais robusta.

```
class ManipuladorDeDados {
    public function processar() {
        $numArgs = func_num_args();
        $args = func_get_args();

        if ($numArgs === 1 && is_array($args[0])) {
            echo "Processando array de dados.\n";
            return count($args[0]);
        } elseif ($numArgs === 2 && is_string($args[0]) && is_numeric($args[1])) {
            echo "Processando string e número.\n";
            return $args[0] . "-" . $args[1];
        } elseif ($numArgs >= 2 && is_numeric($args[0])) {
            echo "Processando múltiplos números.\n";
            return array_sum($args);
        } else {
            echo "Tipo de dados não suportado.\n";
            return null;
        }
    }
}

$manipulador = new ManipuladorDeDados();
echo $manipulador->processar([10, 20, 30]) . "\n"; // Saída: Processando array de dados. 3
echo $manipulador->processar("Item", 123) . "\n"; // Saída: Processando string e número. Item-123
echo $manipulador->processar(1, 2, 3, 4, 5) . "\n"; // Saída: Processando múltiplos números. 15
echo $manipulador->processar("Olá", "Mundo") . "\n"; // Saída: Tipo de dados não suportado.
```

Esta técnica oferece grande flexibilidade, mas a legibilidade do código pode ser comprometida, e a lógica interna pode se tornar complexa. Além disso, a falta de verificação estática de tipos pode levar a erros em tempo de execução que seriam capturados em tempo de compilação em outras linguagens.

Sobrecarga em PHP

Comparação e Melhores Práticas

Enquanto linguagens como Java ou C# beneficiam-se da sobrecarga nativa para criar APIs intuitivas com verificação de tipo em tempo de compilação, o PHP exige abordagens mais dinâmicas. Em C#, por exemplo, você pode ter:

```
public class Calculator {
    public int Add(int a, int b) { return a + b; }
    public double Add(double a, double b) { return a + b; }
    public int Add(int a, int b, int c) { return a + b + c; }
}
```

O compilador C# sabe qual método `Add` chamar com base nos tipos e número de argumentos. No PHP, a decisão é sempre em tempo de execução.

Vantagens e Desvantagens das Abordagens em PHP:

- **Parâmetros Opcionais:**
  - **Vantagens:** Simples, legível, boa performance, fácil de implementar.
  - **Desvantagens:** Menos flexível para grandes variações de tipos/número de parâmetros, a lógica interna pode ficar aninhada.
- `__call()`**:**
  - **Vantagens:** Permite sobrecarga verdadeiramente dinâmica baseada no nome do método, útil para DSLs (Domain-Specific Languages) ou comportamentos de proxy.
  - **Desvantagens:** Potencial impacto na performance, erros são detectados apenas em tempo de execução, menos legível para quem não conhece bem a lógica do `__call()`.
- `func_get_args()`**:**
  - **Vantagens:** Máxima flexibilidade para lidar com qualquer número e tipo de argumentos.
  - **Desvantagens:** Código mais verboso e complexo, difícil de manter, performance ligeiramente inferior, menos verificação estática.

Recomendações de Performance e Boas Práticas:

Para a maioria dos casos, a abordagem com **parâmetros opcionais** é a mais indicada devido à sua simplicidade, clareza e eficiência. Use `__call()` apenas quando for realmente necessário um comportamento dinâmico complexo, como em frameworks ou bibliotecas que exigem tal flexibilidade. Evite `func_get_args()` a menos que você esteja lidando com um número arbitrário de argumentos e a tipagem não possa ser resolvida de outra forma, pois pode tornar o código mais difícil de depurar e manter. Sempre documente claramente as diferentes formas de chamar seus métodos para garantir que outros desenvolvedores (ou seu futuro eu) entendam o comportamento esperado.

Exemplo Prático: Calculadora com Sobrecarga

Vamos implementar uma classe `Calculadora` que simula sobrecarga para diferentes operações de soma:

```
class Calculadora {
 public function somar($a, $b = null, $c = null) {
 if ($c !== null) {
 return $a + $b + $c; // Soma três números
 } elseif ($b !== null) {
 return $a + $b; // Soma dois números
 } else {
 return $a; // Retorna o próprio número
 }
 }
}

$calc = new Calculadora();
echo $calc->somar(5); // Resultado: 5
echo $calc->somar(5, 3); // Resultado: 8
echo $calc->somar(5, 3, 2); // Resultado: 10
```

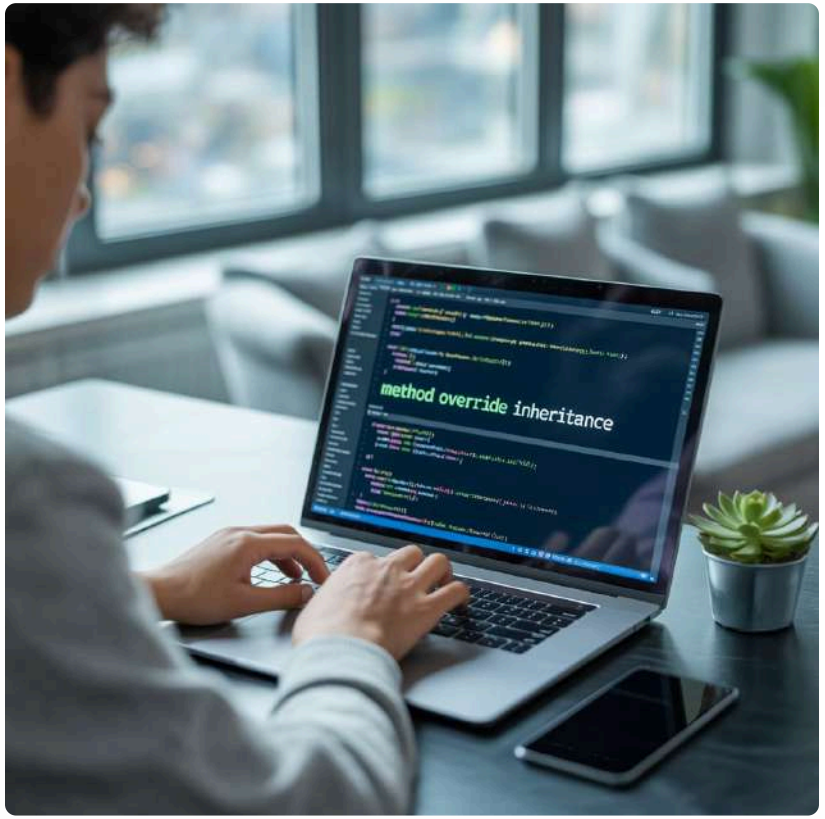
Sobrescrita de Métodos

O que é Sobrescrita?

A sobrescrita ocorre quando uma classe filha redefine um método herdado da classe pai, fornecendo uma **implementação específica** para suas necessidades.

Características Principais:

- Mesmo nome e assinatura do método pai
- Implementação personalizada
- Uso da palavra-chave `override` (em algumas linguagens)
- Mantém o contrato da classe pai



A sobrescrita é essencial para o polimorfismo, pois permite que objetos de diferentes classes respondam de forma única ao mesmo método.

Exemplo: Sobrescrita do Método falar()

Implementação prática de sobrescrita na hierarquia Animal:

Classe Animal (Pai)

```
abstract class Animal {
    abstract public function falar();
}
```

Classe Cachorro

```
class Cachorro extends Animal {
    public function falar() {
        return "Au au! Woof!";
    }
}
```

Classe Gato

```
class Gato extends Animal {
 public function falar() {
 return "Miau! Meow!";
 }
}
```

Diagrama de Classes: Polimorfismo em Ação

Visualização da estrutura hierárquica e do polimorfismo:

Polimorfismo

Sobrescrita e uso via Animal

Gato

Implementa falar(): "Miau"

Cachorro

Implementa falar(): "Auau"

Animal

Método abstrato falar()

Cenário Real: Sistema de Pagamento

Aplicação prática do polimorfismo em um sistema de e-commerce:

Cartão de Crédito

Implementa `processarPagamento()` com validação de cartão, autorização bancária e cobrança em parcelas.

PIX

Processa pagamento instantâneo via chave PIX, validando conta e transferindo valor em tempo real.

Boleto Bancário

Gera código de barras, define data de vencimento e aguarda confirmação de pagamento.

Cada método de pagamento implementa `processarPagamento()` de forma única, mas o sistema pode tratá-los uniformemente.

Cenário Real: Sistema de Notificações

Polimorfismo aplicado em sistema de comunicação multicanal:

Implementação Polimórfica

Cada canal de notificação implementa o método `enviar()` com suas particularidades:

- **Email:** formatação HTML, anexos, headers SMTP
- **WhatsApp:** integração com API, formatação de mensagem
- **Push:** payload JSON, tokens de dispositivo

```
interface Notificacao {
    public function enviar($mensagem, $destinatario);
}

$notificacoes = [
    new EmailNotificacao(),
    new WhatsAppNotificacao(),
    new PushNotificacao()
];

foreach ($notificacoes as $notif) {
    $notif->enviar("Bom dia!", $usuario);
}
```

Benefícios do Polimorfismo

Reuso de Código

Permite escrever código genérico que funciona com diferentes tipos de objetos, eliminando duplicação e reduzindo a manutenção necessária.

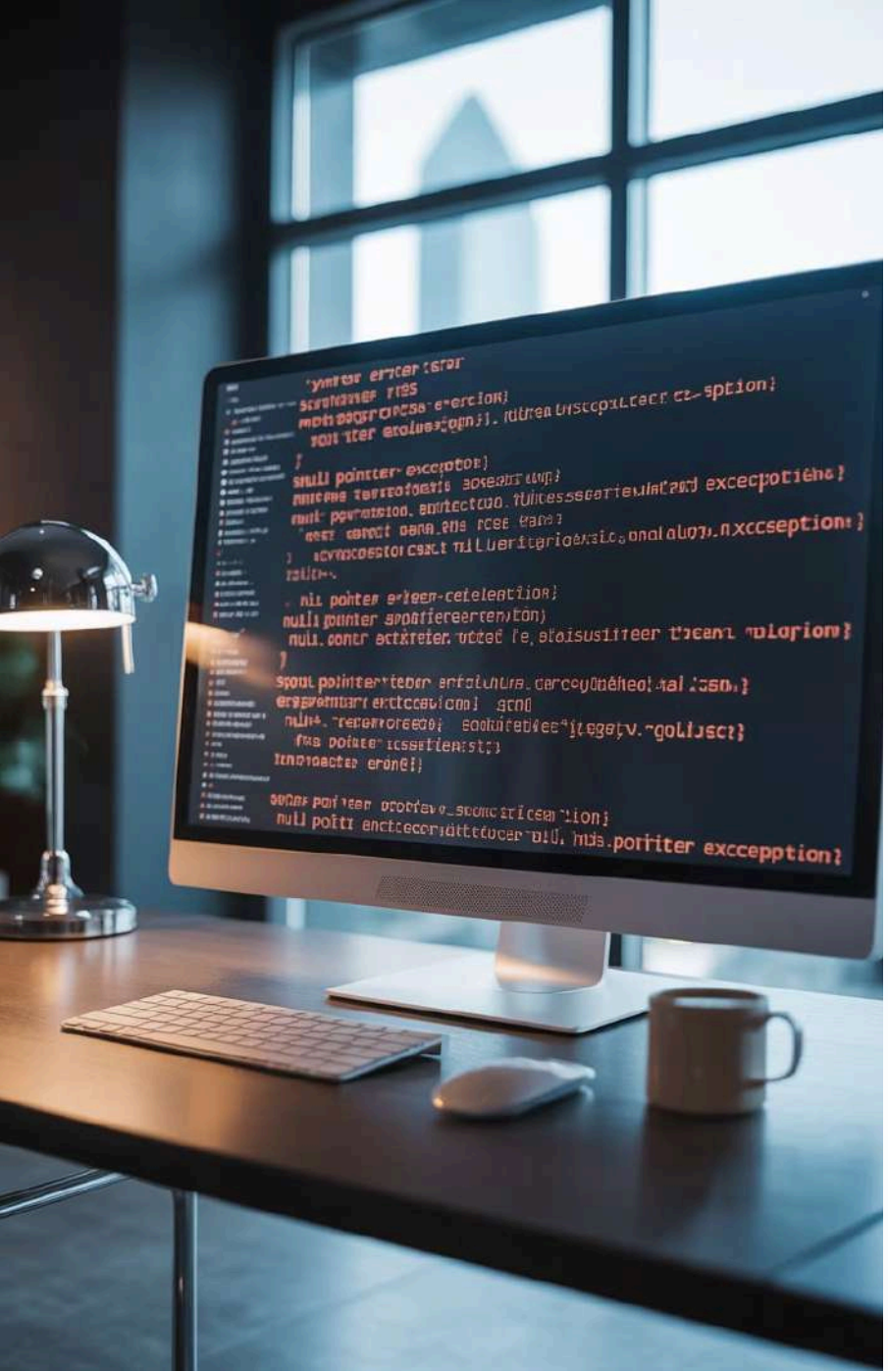
Flexibilidade

Facilita mudanças e adaptações no sistema. Novos tipos podem ser adicionados sem modificar o código existente que os utiliza.

Extensibilidade

O sistema pode crescer de forma orgânica. Novas implementações seguem contratos estabelecidos, garantindo compatibilidade e funcionalidade.

O polimorfismo torna o código mais limpo, organizando e fácil de manter, seguindo o princípio DRY (Don't Repeat Yourself).



Erros Comuns no Polimorfismo

Armadilhas frequentes que desenvolvedores iniciantes enfrentam:

Confusão Conceitual

Problema: Misturar sobrecarga com sobrescrita.

Solução: Lembrar que sobrecarga = mesmo nome, parâmetros diferentes; sobrescrita = mesmo método, implementação diferente.

Violação de Encapsulamento

Problema: Expor métodos internos desnecessariamente.

Solução: Manter métodos privados quando possível, expor apenas interfaces necessárias.

Quebra do Contrato

Problema: Sobrescrita que muda comportamento esperado.

Solução: Respeitar o princípio de substituição de Liskov - objetos filhos devem funcionar onde objetos pai funcionam.

Atividade em Grupo

Vamos colocar a teoria em prática!

01

Formação dos Grupos

Organizem-se em grupos de 3-4 pessoas para resolver os exercícios colaborativamente.

02

Análise dos Problemas

Leiam cuidadosamente cada exercício e identifiquem onde aplicar polimorfismo.

03

Implementação

Desenvolvam as soluções discutindo as melhores abordagens dentro do grupo.

04

Apresentação

Preparem-se para apresentar e discutir suas soluções com a turma.

Exercício 1: Hierarquia Animal

Objetivo: Implementar polimorfismo através de sobrescrita na hierarquia Animal.

Requisitos

1. Criar classe abstrata `Animal` com método `falar()`
2. Implementar classes `Cachorro`, `Gato` e `Passaro`
3. Cada animal deve ter som específico
4. Criar um array de animais e chamar `falar()` para cada um
5. Adicionar método `mover()` com comportamentos diferentes

> Como demonstrar que diferentes objetos respondem de forma única à mesma chamada de método?

Saída Esperada

```
Cachorro fala: Au au!
Gato fala: Miau!
Pássaro fala: Piu piu!

Cachorro move: Correndo
Gato move: Caminhando
Pássaro move: Voando
```

Exercício 2: Calculadora com Sobrecarga

Objetivo: Simular sobrecarga de métodos em PHP para diferentes operações matemáticas.

Parte A: Método somar()

- Somar dois números
- Somar três números
- Somar array de números

Parte B: Método calcular()

- Operação com dois operandos
- Múltiplas operações em sequência
- Operações com precisão decimal

Parte C: Validações

- Tratar divisão por zero
- Validar tipos de entrada
- Retornar mensagens de erro apropriadas

Dica: Use parâmetros opcionais e o método mágico `__call()` para simular sobrecarga real.

Discussão em Grupo

Momento de compartilhar conhecimento e comparar soluções!

Apresentação

Cada grupo apresenta sua solução explicando as decisões de design tomadas.

Comparação

Analisamos diferenças de implementação e discutimos prós e contras de cada abordagem.

Aprendizado

Identificamos padrões comuns e melhores práticas emergentes das discussões.

Feedback

Professores e colegas oferecem sugestões de melhoria e validam soluções.

Resumo da Aula

Recapitulando os conceitos fundamentais aprendidos hoje:

Polimorfismo

Capacidade de objetos assumirem **múltiplas formas**, respondendo de maneira única à mesma chamada de método. Fundamental para código flexível e reutilizável.

Sobrecarga

Múltiplos métodos com o **mesmo nome** mas **parâmetros diferentes**. Em PHP, simulada através de parâmetros opcionais e métodos mágicos.

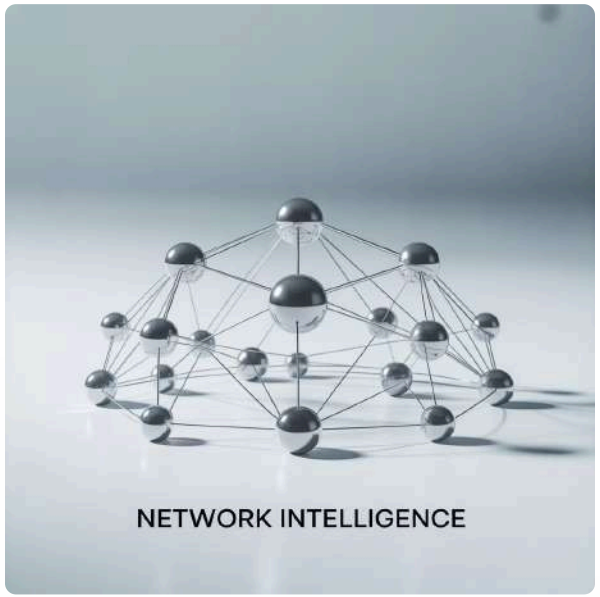
Sobrescrita

Redefinição de métodos herdados para **implementação específica**. Permite que classes filhas personalizem comportamentos mantendo a interface.

> "O polimorfismo é a chave para escrever código elegante, extensível e de fácil manutenção na programação orientada a objetos."

Próxima Aula

Relacionamentos entre Classes



Tópicos da Próxima Aula

01

Associação

Como classes se relacionam e colaboram entre si através de referências e dependências.

02

Agregação

Relacionamento "tem-um" onde objetos existem independentemente uns dos outros.

03

Composição

Relacionamento mais forte onde objetos não existem sem o objeto container.

Preparem-se para explorar como diferentes classes podem trabalhar juntas de forma coordenada!