A photograph of a computer lab with several students working at desks. The room is dimly lit with a strong blue light source, likely from the ceiling lights and the computer monitors. The students are seated in rows, facing away from the camera, focused on their work. The desks are equipped with multiple monitors and keyboards. The overall atmosphere is professional and academic.

Métodos e Atributos: Construtores e Modificadores de Acesso

UNIVERSIDADE DO OESTE DE SANTA CATARINA - UNOESC

Ciência da Computação | Programação II

Prof. Leandro Otavio Cordova Vieira

Objetivos da Aula

1

Construtores no PHP

Compreender o propósito e a implementação de métodos construtores na criação de objetos.

2

Modificadores de Acesso

Diferenciar os níveis de visibilidade **public**, **private** e **protected** e suas aplicações práticas.

3

Encapsulamento

Aplicar técnicas de encapsulamento através de getters e setters para proteger e controlar o acesso aos atributos.

Ao final desta aula, você estará apto a criar classes com estruturas robustas e seguindo as boas práticas de programação orientada a objetos.

Revisão: Classes e Objetos

Classe = Molde

Define um tipo de dado, especificando:

- Estrutura (atributos)
- Comportamento (métodos)
- Estado inicial (construtor)

É o modelo a partir do qual os objetos são criados.

Objeto = Instância

Representação concreta de uma classe:

- Ocupa espaço na memória
- Possui valores específicos para seus atributos
- É criado usando a palavra-chave new



Revisão: Métodos Básicos

Métodos são **funções definidas dentro de uma classe** que determinam o comportamento dos objetos.

Definição

```
public function nomeDaFuncao($parametro) {  
    // código  
    return $resultado;  
}
```

Chamada

```
$objeto = new MinhaClasse();  
$objeto->nomeDaFuncao("valor");
```

Propósitos

- Manipular atributos
- Executar operações
- Interagir com outros objetos

Os métodos encapsulam a lógica de funcionamento do objeto, permitindo reuso e organização do código.

O que são Construtores?

O construtor é um **método especial** que é executado **automaticamente** quando um novo objeto é criado.

Características:

- Nome fixo: `__construct()` em PHP
- Executado no momento do `new`
- Não possui retorno (nem void)
- Pode receber parâmetros

Finalidades:

- Inicializar atributos
- Garantir estado inicial válido
- Executar operações necessárias na criação
- Receber dependências



Sintaxe do __construct()

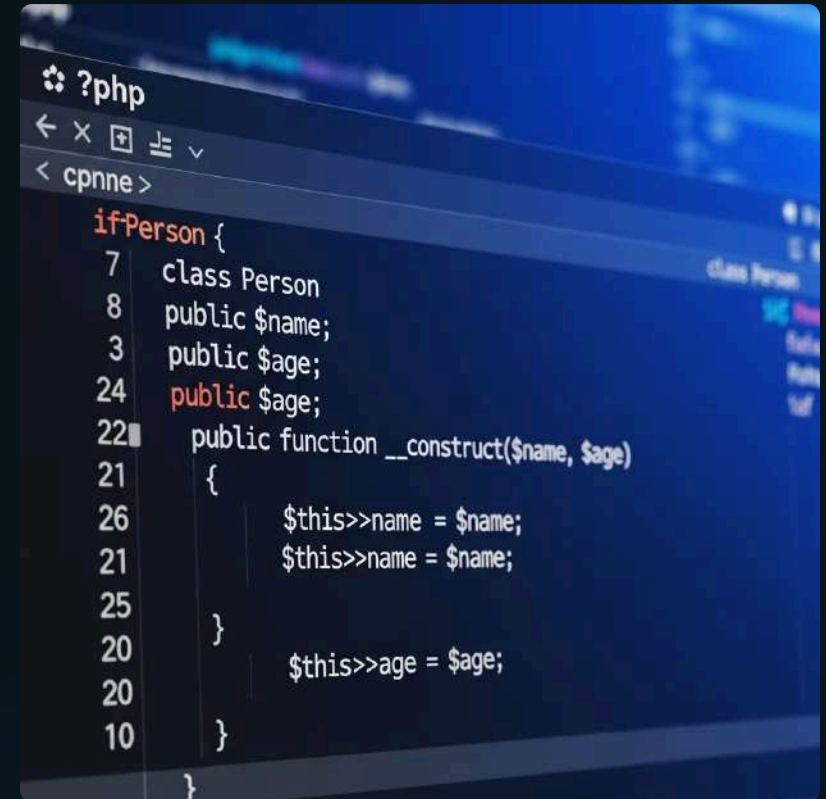
Em PHP, o construtor é definido através do método mágico `__construct()`:

```
class MinhaClasse {  
    private $atributo1;  
    private $atributo2;  
  
    public function __construct($valor1, $valor2) {  
        $this->atributo1 = $valor1;  
        $this->atributo2 = $valor2;  
        echo "Objeto criado com sucesso!";  
    }  
}  
  
// Criando um objeto  
$objeto = new MinhaClasse("Valor inicial 1", "Valor inicial 2");
```

❗ **Observação:** Em versões do PHP anteriores à 8.0, os construtores podiam ser definidos com o mesmo nome da classe, mas essa prática está obsoleta.

Exemplo Prático: Classe Pessoa com Construtor

```
class Pessoa {  
    private $nome;  
    private $idade;  
    private $email;  
  
    public function __construct($nome, $idade, $email) {  
        $this->nome = $nome;  
        $this->idade = $idade;  
        $this->email = $email;  
        echo "Pessoa {$this->nome} criada!";  
    }  
  
    public function apresentar() {  
        return "Olá! Sou {$this->nome}, " .  
            "tenho {$this->idade} anos.";  
    }  
}  
  
// Criando uma pessoa  
$pessoa1 = new Pessoa("Ana Silva", 22, "ana@email.com");  
echo $pessoa1->apresentar();
```

A screenshot of a code editor window titled '?php' with a file path '< cpnne >'. The code defines a 'Person' class with public properties '\$name' and '\$age', and a public constructor method '__construct(\$name, \$age)'. The constructor method contains two lines of code: '\$this->>name = \$name;' and '\$this->>name = \$name;', followed by '\$this->>age = \$age;'. The code is highlighted with a blue background and white text.

```
?php  
< cpnne >  
if Person {  
7   class Person  
8   public $name;  
3   public $age;  
24  public $age;  
22  public function __construct($name, $age)  
21  {  
26      $this->>name = $name;  
21      $this->>name = $name;  
25  }  
20      $this->>age = $age;  
20  }  
10  }
```

Saída:

Pessoa Ana Silva criada!
Olá! Sou Ana Silva, tenho 22 anos.

Encapsulamento – Conceito

Esconder para Proteger

O encapsulamento é um dos **quatro pilares da Programação Orientada a Objetos**, ao lado de herança, polimorfismo e abstração.

Consiste em **esconder os detalhes de implementação** e expor apenas o necessário para uso do objeto.



Benefícios

Controle sobre como os dados são acessados e modificados, evitando estados inválidos.

Implementação

Através de modificadores de acesso e métodos específicos (getters e setters).

Modificadores de Acesso

Os modificadores de acesso controlam a visibilidade de atributos e métodos em PHP:



public

Acesso livre de qualquer lugar

```
public $nome;
```

Pode ser acessado por qualquer código, dentro ou fora da classe.



private

Acesso restrito à própria classe

```
private $senha;
```

Só pode ser acessado por métodos da mesma classe onde foi definido.



protected

Acesso na classe e subclasses

```
protected $id;
```

Acessível na classe onde foi definido e em todas as classes que herdam dela.

A escolha correta do modificador é fundamental para implementar o encapsulamento.

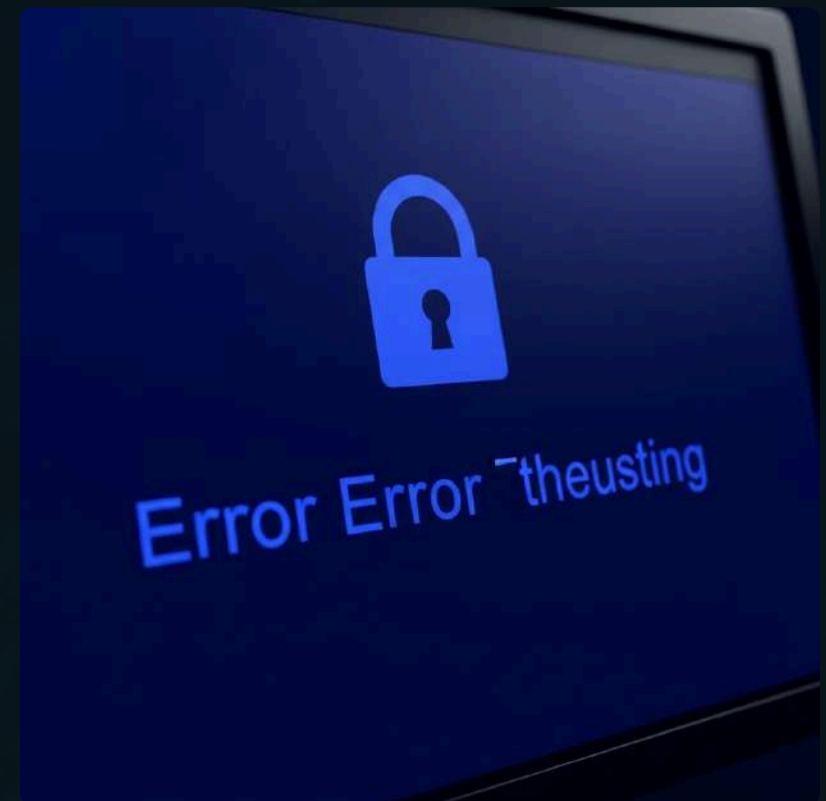
Exemplo de Atributo Privado

Quando tentamos acessar diretamente um atributo privado, ocorre um erro:

```
class ContaBancaria {  
    private $saldo = 1000;  
    private $senha = "1234";  
  
    public function fazerSaque($valor, $senhaInformada) {  
        if ($senhaInformada != $this->senha) {  
            return "Senha incorreta!";  
        }  
        if ($valor > $this->saldo) {  
            return "Saldo insuficiente!";  
        }  
        $this->saldo -= $valor;  
        return "Saque de R$ {$valor} realizado!";  
    }  
}  
  
$conta = new ContaBancaria();  
// Tentativa incorreta - erro!  
echo $conta->saldo;  
echo $conta->senha;
```

⊗ Erro!

Fatal error: Uncaught Error: Cannot access private property ContaBancaria::\$saldo



Os atributos privados só podem ser acessados dentro da própria classe.

Getters e Setters

Getters e setters são **métodos especiais** que permitem acessar e modificar atributos privados de forma controlada:

Getters (Acessores)

- Nomenclatura:
`getNomeDoAtributo()`
- Retornam o valor do atributo
- Permitem leitura controlada
- Podem formatar dados na leitura

```
public function getNome() {  
    return $this->nome;  
}
```

Setters (Modificadores)

- Nomenclatura:
`setNomeDoAtributo()`
- Alteram o valor do atributo
- Permitem validações antes da atribuição
- Garantem integridade dos dados

```
public function  
setIdade($idade) {  
    if ($idade > 0) {  
        $this->idade = $idade;  
    }  
}
```



Exemplo Prático: Classe Produto com Get/Set

```
class Produto {  
    private $nome;  
    private $preco;  
    private $estoque;  
  
    public function __construct($nome, $preco, $estoque) {  
        $this->nome = $nome;  
        $this->setPreco($preco); // Usando setter no construtor  
        $this->setEstoque($estoque);  
    }  
  
    public function getNome() {  
        return $this->nome;  
    }  
  
    public function setNome($nome) {  
        $this->nome = $nome;  
    }  
  
    public function getPreco() {  
        return $this->preco;  
    }  
  
    public function setPreco($preco) {  
        if ($preco > 0) {  
            $this->preco = $preco;  
        } else {  
            throw new Exception("Preço deve ser maior que zero");  
        }  
    }  
  
    public function getEstoque() {  
        return $this->estoque;  
    }  
  
    public function setEstoque($estoque) {  
        if ($estoque >= 0) {  
            $this->estoque = $estoque;  
        } else {  
            throw new Exception("Estoque não pode ser negativo");  
        }  
    }  
}
```


Encapsulamento Aplicado

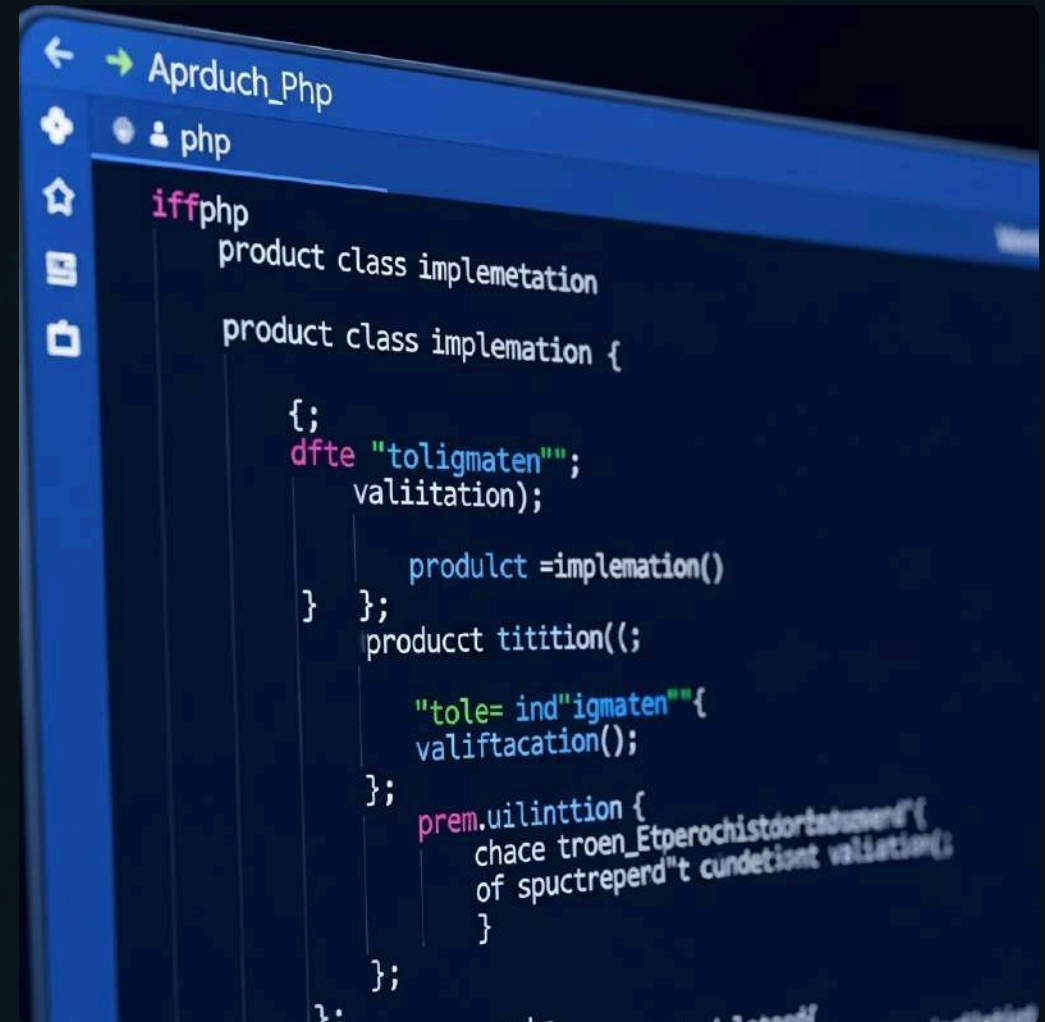
Veja como o uso de getters e setters permite maior controle sobre os dados:

```
// Criando um produto
$celular = new Produto(
    "Smartphone X",
    1999.90,
    50
);

// Usando getters
echo "Produto: " . $celular->getNome();
echo "Preço: R$ " . $celular->getPreco();
echo "Estoque: " . $celular->getEstoque();

// Usando setters com validação
try {
    $celular->setPreco(-100); // Erro!
} catch (Exception $e) {
    echo "Erro: " . $e->getMessage();
}

// Alteração válida
$celular->setPreco(1899.90);
echo "Novo preço: R$ " . $celular->getPreco();
```



Vantagens demonstradas

- Validação de dados (preço > 0)
- Controle de acesso
- Tratamento de erros
- Possibilidade de implementar lógica na leitura/escrita

Vantagens do Encapsulamento

Segurança

Protege atributos contra alterações indevidas

- Evita estados inválidos
- Impede acesso direto a dados sensíveis
- Controla como os valores são modificados

Elegância

Código mais limpo e profissional

- Interface bem definida
- Separação de responsabilidades
- Conformidade com padrões de projeto



Flexibilidade

Permite modificar a implementação interna

- Alterações na classe sem afetar o uso
- Adaptação a novos requisitos
- Implementação de cache ou logging

Manutenção

Facilita alterações e correções

- Código mais organizado
- Localização centralizada de lógica
- Facilidade para depurar problemas

Exercício 1

Classe Conta com Construtor

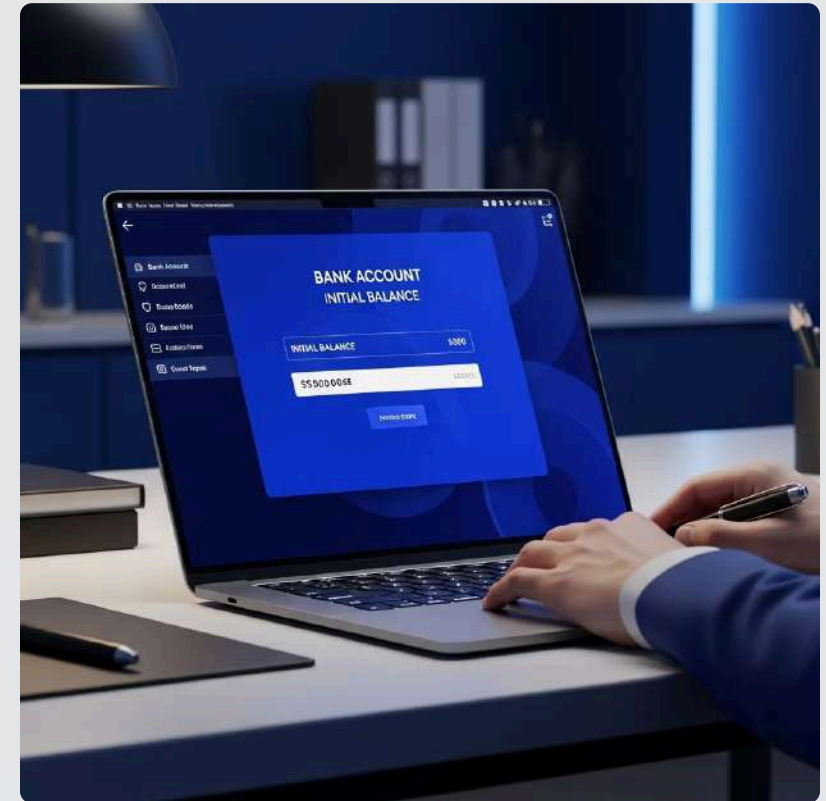
Crie uma classe **Conta** que tenha um construtor para inicializar o titular e o saldo inicial.

Requisitos:

- Atributos: \$titular, \$saldo, \$numero
- Construtor que receba titular e saldo inicial
- Número da conta deve ser gerado automaticamente
- Método `exibirDados()` para mostrar informações
- Método `depositar($valor)` que adicione ao saldo

Teste sua implementação:

```
$conta1 = new Conta("Carlos Silva", 500);  
$conta2 = new Conta("Ana Oliveira", 1200);  
  
echo $conta1->exibirDados();  
$conta1->depositar(300);  
echo $conta1->exibirDados();
```



❏ Dica: Para gerar um número de conta aleatório, você pode usar `rand(1000, 9999)` ou alguma outra estratégia de sua preferência.

Exercício 2

Modificadores de Acesso na Classe Aluno

Crie uma classe **Aluno** com os seguintes requisitos de acesso:

Atributos:

- `$nome` – Público (pode ser lido e alterado)
- `$matricula` – Somente leitura (privado com getter)
- `$notas` – Privado (acessível apenas por métodos)
- `$media` – Privado (calculado internamente)

Métodos:

- Construtor para inicializar nome e matricula
- Getter para matricula (`getMatricula()`)
- `adicionarNota($nota)` para adicionar à lista de notas
- `calcularMedia()` – método privado
- `situacao()` para retornar "Aprovado" ou "Reprovado"

Teste a classe criando dois alunos, adicionando notas diferentes e verificando a situação de cada um.

Student Profile

UNLOCKED



 Name: [Student's Name]

 Student ID: [ID Number]

 Major: [Major Name]

 Academic Record: [Locked [Date]]

 Contact Information: [Locked [Date]]

 Emergency Contact: [Locked]

Exercício 3

Getters e Setters na Classe Livro

Implemente uma classe **Livro** com getters e setters que façam validações:

Atributos (todos privados):

- `$titulo`
- `$autor`
- `$anoPublicacao`
- `$numeroPaginas`
- `$disponivel` (booleano)

Validações necessárias:

- Ano de publicação não pode ser futuro
- Número de páginas deve ser positivo
- Título e autor não podem ser vazios



Teste sua implementação:

```
$livro = new Livro();
$livro->setTitulo("Dom Casmurro");
$livro->setAutor("Machado de Assis");
$livro->setAnoPublicacao(1899);
$livro->setNumeroPaginas(256);
$livro->setDisponivel(true);

echo $livro->getTitulo();
// Teste de validação
try {
    $livro->setAnoPublicacao(2030);
} catch (Exception $e) {
    echo "Erro: " . $e->getMessage();
}
```

Atividade em Duplas



Formação

Forme uma dupla com um colega próximo



Implementação

Resolva os exercícios anteriores com seu parceiro



Testes

Crie casos de teste para validar sua implementação



Comparação

Compare sua solução com outras duplas

Orientações para a atividade:

- Tempo de desenvolvimento: 30 minutos
- Foco especial na validação de dados nos setters
- Explore diferentes formas de implementar os construtores
- Documente seu código com comentários explicativos

Durante a atividade, o professor estará disponível para esclarecer dúvidas e orientar sobre as implementações.



Discussão Coletiva

Compartilhamento de Soluções

Vamos discutir as diferentes abordagens e soluções encontradas pelas duplas:

- Apresentação de 2-3 soluções diferentes
- Comparação de estratégias de validação
- Discussão sobre escolhas de design
- Identificação de padrões comuns

Erros Comuns e Boas Práticas

- Esquecer validações nos setters
- Acesso direto a atributos privados
- Construtores sem inicialização adequada
- Métodos getters/setters incompletos
- Nomes de métodos inconsistentes

A discussão coletiva ajuda a consolidar o conhecimento e explorar diferentes perspectivas sobre o mesmo problema.

Resumo e Próximos Passos

Conceitos Aprendidos:

- Construtores: método `__construct()` para inicialização
- Modificadores de acesso: `public`, `private`, `protected`
- Encapsulamento através de getters e setters
- Validação de dados em métodos setters
- Boas práticas de orientação a objetos

Próximo Tema:

Herança e Reuso de Código

Estudaremos como criar hierarquias de classes, reutilizar código e implementar comportamentos especializados.



Leitura Complementar:

- Documentação PHP: Construtores e Destrutores
- Clean Code: Capítulo sobre Classes
- Design Patterns em PHP