CURSOS
TÉCNICOS

DESENVOLVIMENTO DE SISTEMAS WEB II

Eixo Informática para Internet

UNIDADE 4

QI
Faculdade & Escola Técnica

SUMÁRIO

UNIDADE 4

1. PHP WEB e Orientação a objetos

Nesta unidade, o foco recai, essencialmente, sobre o paradigma orientado a objetos, classe e diagrama de classe.

1.1 Paradigma orientado a objetos

"Como a maioria das atividades que fazemos no dia a dia, programar também possui modos diferentes de se fazer. Esses modos são chamados de **paradigmas de programação** e, entre eles, estão a programação orientada a objetos (POO) e a programação estruturada. Quando começamos a utilizar linguagens como Java, C#, Python e outras que possibilitam o paradigma orientado a objetos, é comum errarmos e aplicarmos a programação estruturada achando que estamos usando recursos da orientação a objetos. " (HENRIQUE, 2023).

Na programação estruturada, temos basicamente três tipos de estruturas:

a) **Sequências:** são os comandos que serão executados.

b) **Condicionais**: sequências de código, que só devem ser executadas mediante a um retorno verdadeiro de uma condicional (Exemplo: if-else, switch).

c) **Repetições**: sequências de código, que se repetem com a declaração de uma estrutura de repetição, e se encerram após a uma condição for satisfeita (Exemplo: for, while, do-while...).

Conforme Henrique (2023), utilizamos estas estruturas para processar a entrada de um programa, e alterar seus dados até que o resultado esperado seja alcançado. Outro ponto é que, com o paradigma funcional/estruturado (que é o que estamos utilizando até o momento), não temos recursos necessários para separar, restringir ou encapsular partes do nosso código. Por exemplo, se desejássemos utilizar um modelo para gerar outras partes do nosso código? Com isso, poderíamos ter uma estrutura padrão, regras para esse contexto e até mesmo uma maior confiabilidade.

O fato é que, com o paradigma de programação funcional/estruturado, não é possível realizar isto. E neste momento, temos de recorrer à programação orientada a objetos, que é uma excelente alternativa para superar os gargalos da programação estruturada.

Esta Programação orientada a objetos (POO), tem como base dois conceitos chave: classes e objetos. Portanto, a partir de agora, iremos analisar cada um dos outros conceitos que surgem a partir destes.

1.2 Classes

Uma classe funciona como um "molde" para a definição de outras estruturas. Classes, geralmente, são compostas pelo agrupamento de atributos ou características, e métodos ou ações. Uma classe define agrupamentos de atributos e métodos que são correlacionados, e podem ser reaproveitados (FONSECA, 2022).

Por exemplo, imagine que você precisa criar uma aplicação que controla um estoque de TV's (televisão). Nessa aplicação, com certeza será necessário manipular informações sobre as TV's, lembrando que todas as TV's geralmente possuem um "molde" padrão, com características e ações que são comuns a todas TVs ou que tornam uma TV única.

Ressalta-se, ainda, que toda TV possui características como modelo; marca; tamanho; cor; etc. Algumas ações em comum das TV's podem ser manipular volume; trocar canal; ligar; desligar; dentre outras funcionalidades, e tendo em mente esta estrutura padrão poderíamos construir uma classe, que servirá como base/molde para cada TV que fossemos registrar no nosso sistema.

Segue exemplo de código PHP com a definição de classe TV:

```php
<?php

class Tv
{
    //Atributos ou características de uma TV
    public string $modelo;
    public string $marca;
    public float $tamanho;
    public string $cor;

    //Métodos ou ações de uma TV
    public function manipularVolume()
    {
        //Código para manipular o volume
    }

    public function trocarCanal()
    {
        //Código para trocar canal
    }

    public function ligar()
    {
        //Código para ligar TV
    }

    public function desligar()
    {
        //Código para desligar TV
    }
}
```

Perceba que traduzimos a estrutura padrão de uma TV criando uma classe, neste exemplo. Com isso, compreendemos que:

- ✓ **Atributos** → atributos são as características que descrevem o estado de um objeto, armazenados como variáveis na classe.

- ✓ **Métodos** → são as ações que os objetos podem executar, definidos como funções na classe.

Porém, perceba que nesta estrutura temos o uso da palavra *public* mais de uma vez neste código. Veja sobre modificadores de acesso.

1.3 Encapsulamento: modificadores de acesso

Os modificadores de acesso em Programação Orientada a Objetos (POO) são palavras-chave que determinam o nível de visibilidade e acessibilidade de atributos e métodos em uma classe. Os principais modificadores de acesso são:

- ✓ **public** → atributos e métodos marcados como públicos são acessíveis de qualquer lugar, tanto dentro quanto fora da classe.

- ✓ **private** → atributos e métodos marcados como privados só podem ser acessados dentro da própria classe. Eles são ocultos do mundo exterior, ou seja: Não é possível acessar o atributo de uma classe diretamente.

- ✓ **protected** → atributos e métodos protegidos são semelhantes aos privados, mas também podem ser acessados por subclasses (herança) da classe que os define. Os modificadores de acesso, são um grande acerto deste paradigma, pois permitem controlar o encapsulamento e a segurança de dados e funcionalidades em um programa orientado a objetos, ajudando a manter a integridade e a estrutura do código.

Na maioria dos casos é recomendado utilizar **private** em atributos de uma classe, pois dizem respeito a aquela classe em específico, e se fosse preciso **recuperar** ou **alterar** algum valor poderíamos utilizar **funções públicas** que podem ser chamadas em outros contextos da aplicação. Qual a vantagem? Podemos executar validações e controlar o fluxo de acordo com o que definirmos no escopo destas funções.

Exemplo de atributos private

```php
<?php

class Tv
{
    //Atributos ou características de uma TV
    private string $modelo;
    private string $marca;
    private float $tamanho;
    private string $cor;

}
```

E agora, como recuperar, por exemplo o modelo da TV ou a Marca se os atributos estão privados? Confira: “**getter e setters**”.

1.4 Encapsulamento: getters e setters

Getters e setters são métodos especiais em Programação Orientada a Objetos (POO) usados para acessar (get) e modificar (set) os atributos de uma classe de forma controlada.

Getter(acessador):

→ um getter é um método que permite recuperar o valor de um atributo privado de uma classe.

→ geralmente, os getters têm nomes como "**get**NomeDoAtributo()" e retornam o valor do atributo.

→ eles são úteis para permitir que outras partes do programa acessem os dados de um objeto sem modificar diretamente os atributos privados.

→ getters oferecem um nível de encapsulamento, pois controlam o acesso aos atributos.

Exemplo de getter:

```php
public function getModelo(): string
{
    return $this->modelo;
}
no usages
public function getMarca(): string
{
    return $this->marca;
}
```

Setter(Mutador):

→ um setter é um método que permite definir (alterar) o valor de um atributo privado de uma classe.

→ geralmente, os setters têm nomes como "**set**NomeDoAtributo()" e recebem um parâmetro que define o novo valor do atributo.

→ eles são usados para garantir que a modificação de um atributo seja feita de acordo com as regras estabelecidas pela classe, como validações ou limites.

Exemplo de setter:

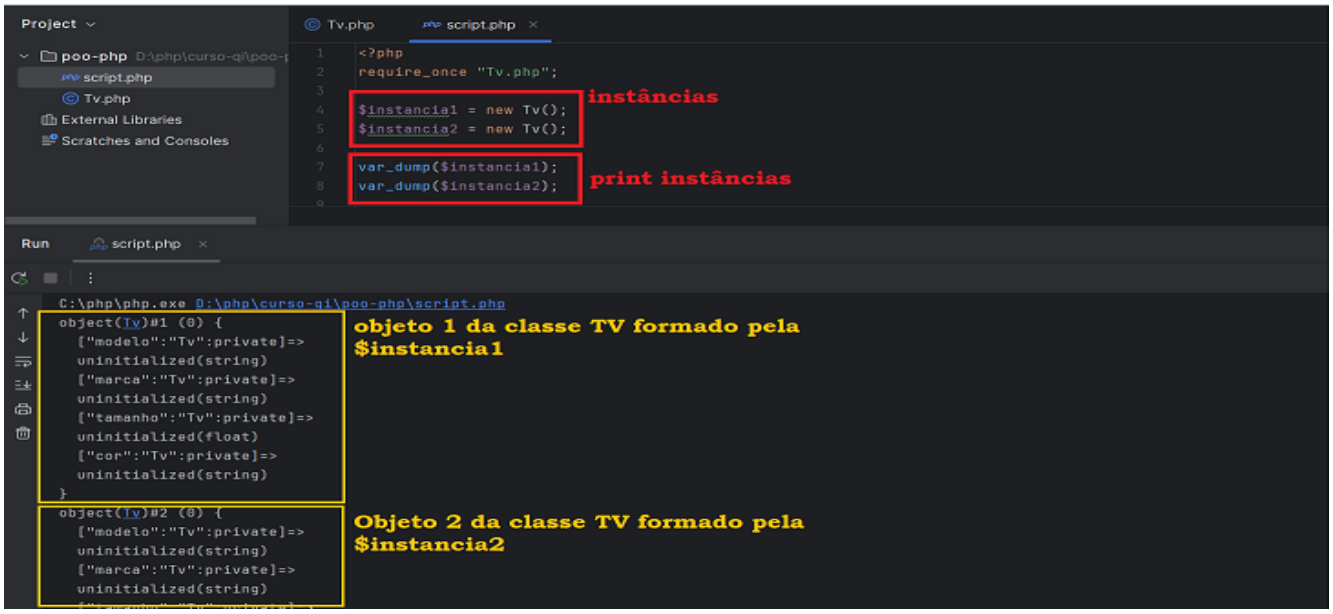
```php
public function setModelo($nomeModelo): void
{
    if(strlen($nomeModelo) < 5) {
        echo "O nome de modelo é muito curto" . PHP_EOL;
        return;
    }

    $this->modelo = $nomeModelo;
}
```

1.5 Instanciação de classes

Para dar "vida" a uma classe precisamos fazer uma **instância** desta. **O resultado de uma instância é a criação de um objeto.** Para instanciar uma classe usamos a palavra reservada "**new**" + **Nomeclasse** + **parâmetros** da função construtora.

Exemplo de instanciação:

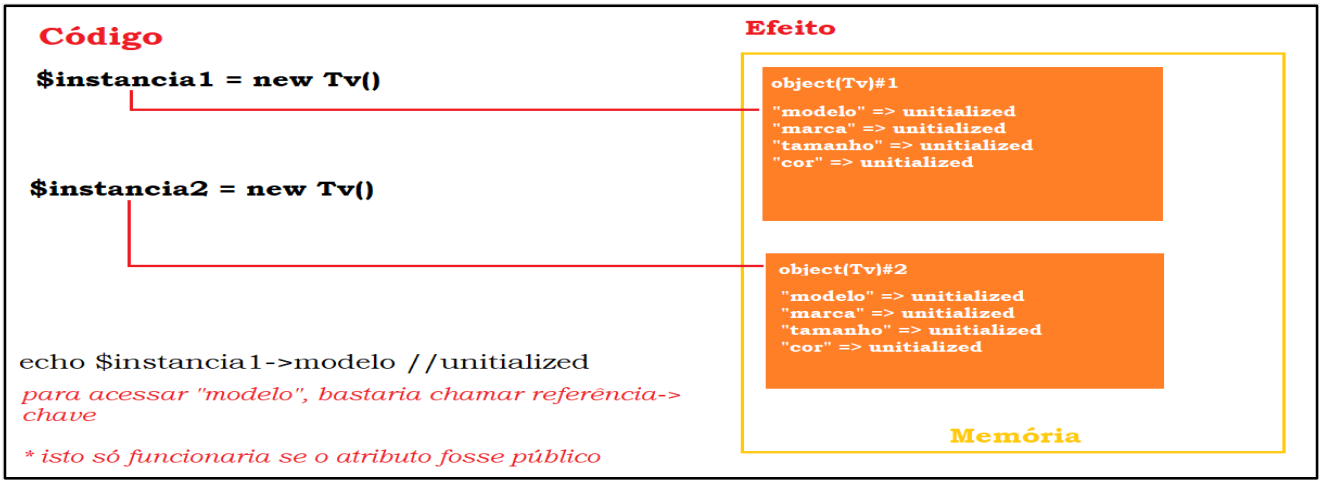


Perceba que, para cada vez que rodamos o comando **new Tb()** é gerado um novo objeto. a variável **$instancia1** e **$instancia2** armazenam a **referência** deste **novo objeto criado** em **memória.**

Um exemplo disso é: quando declaramos o **var_dump()** nas **linhas 7 e 8** com a variável que guarda a referência das instâncias, então tivemos em tela um objeto novo criado com seus **atributos não inicializados.**

Em memória são criados dois objetos. Estes dois objetos estão linkados a variável que armazena a referência. Então sempre que quisermos acessar um método ou atributo de uma classe basta utilizar: **referencia->nomeAtributo | nomeMetodo;**

Exemplo gráfico da instanciação de uma classe Tv



2. Construtores e destrutores

Em PHP, as funções construtoras e destrutoras são métodos especiais que são executados, automaticamente, quando um objeto de uma classe é criado (construtor) ou destruído (destrutor).

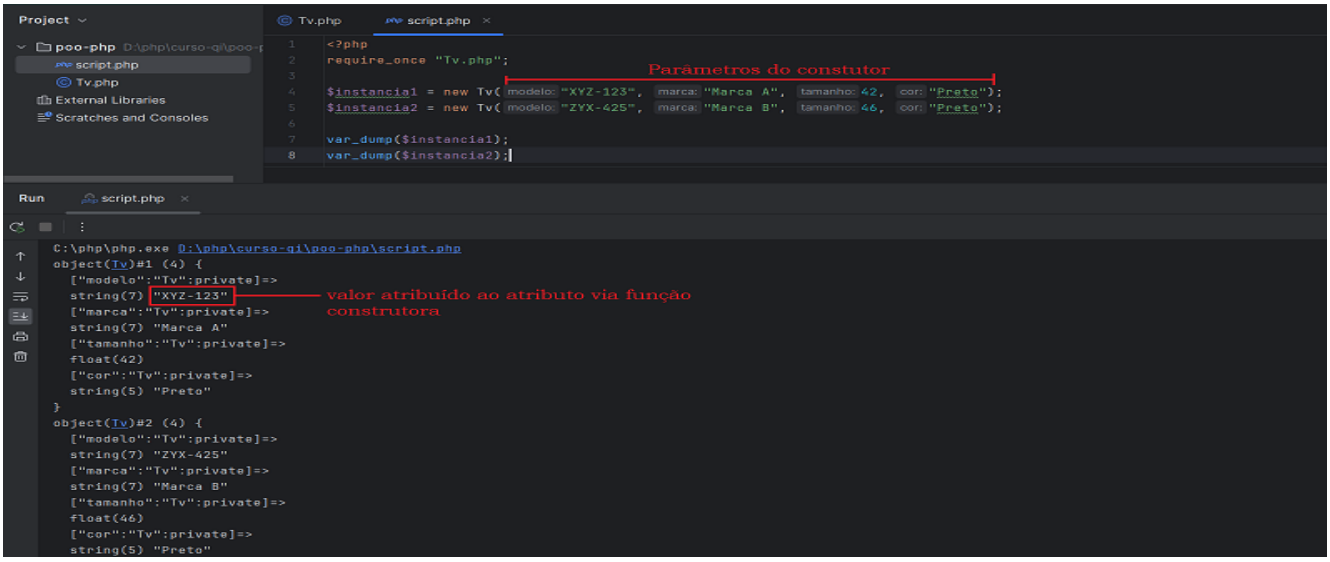
2.1 Construtor

Um construtor é um método especial chamado **__construct()**, que é executado automaticamente quando um objeto é instanciado a partir de uma classe. O construtor é usado para inicializar os atributos ou realizar outras tarefas de inicialização necessárias para o objeto. Confira o exemplo:

```
2 usages
public function __construct(string $modelo, string $marca, string $tamanho, string $cor)
{
    $this->modelo = $modelo;
    $this->marca = $marca;
    $this->tamanho = $tamanho;
    $this->cor = $cor;
}
```

Todo o código envolto no escopo da função **__construct** será executado no momento que um objeto for criado, ou seja, no momento em que a classe for instanciada. Neste exemplo, usamos todos atributos para terem seus valores inicializados de acordo com o que foi passado no construtor.

Instanciação com passagem de parâmetros



Observe que, agora ao executar **var_dump** nas duas instâncias que receberam parâmetros na instanciação, os atributos possuem já valores que foram definidos na chamada da função **__construct**.

É possível atribuir um valor padrão (default) **a um atributo**. Isso pode ser feito via função construtora ou na própria declaração do atributo.

Exemplo: atribuição de valor padrão a um atributo na criação do objeto:

```php
<?php

class Conta
{
    //Atributos ou características de uma TV
    private string $nomePessoa;
    private string $cpf;
    private float $saldo = 0;

    public function __construct(string $nomePessoa, string $cpf)
    {
        $this->nomePessoa = $nomePessoa;
        $this->cpf = $cpf;
    }
}
```

Atributo saldo inicia com valor 0 na criação do objeto

2.2 Destrutor

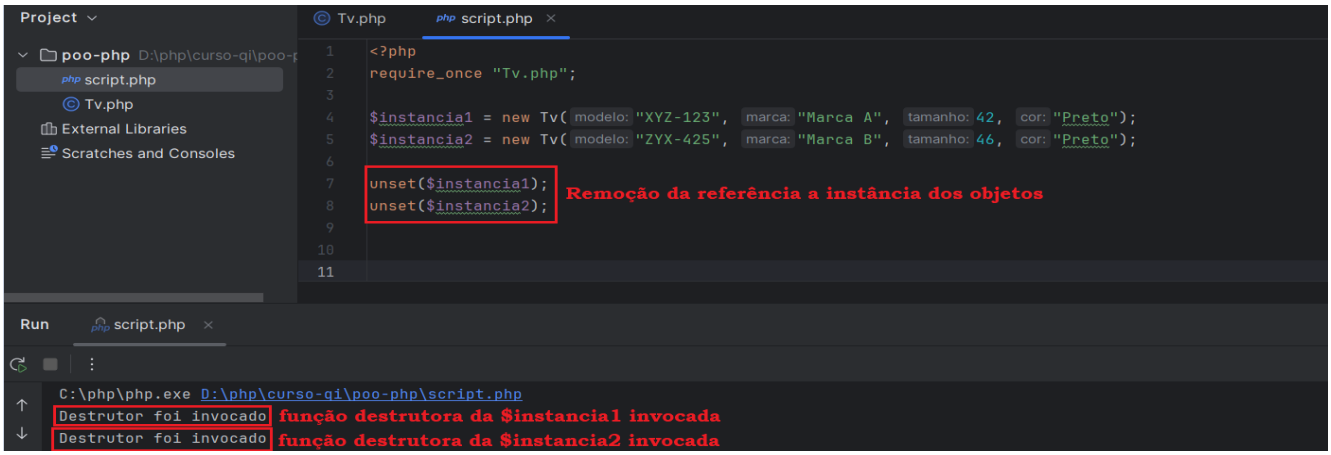
Um destrutor é um método especial chamado **__destruct()** que é executado automaticamente quando um objeto é destruído, ou seja, quando não há mais referências ao objeto. O destrutor pode ser usado para realizar tarefas de limpeza, como fechar conexões de banco de dados, liberar recursos, etc. Confira os exemplos:

Exemplo - Declaração de função destrutora:

```php
no usages
public function __destruct()
{
    echo "Destrutor foi invocado" . PHP_EOL;
}
```

Para que esta função seja invocada, vamos **remover a referência de um objeto instanciado.**

Exemplo - Removendo referência de objeto instanciado e invocando destrutor.



Perceba que no momento em que, foi dado o **unset** de cada uma das instâncias, a função destrutora foi invocada e executou o código que nesse caso era somente exibir a mensagem "destrutor invocado".

3. *Palavras chave this*

No PHP, **this** é uma palavra-chave e é usada para fazer referência a um atributo ou método de acordo com a instância da classe em que é chamado.

O $this é usado **em contexto de objetos dentro de uma classe** e se **refere à instância atual da classe.** Ele é **usado para acessar os membros da classe,** como **atributos** e **étodos**, dentro da própria classe.

Exemplo - $this valores diferentes no this de acordo com a instância.

instância 1

```
object(Pessoa)#1 {
  "nome" => "Pedro"

public function getNome(): string
{
      echo $this->nome; //Pedro
}
}
```

instância 2

```
object(Pessoa)#2 {
  "nome" => "João"

public function getNome(): string
{
      echo $this->nome; //João
}
}
```

Por mais que o código de **getNome()** seja o mesmo, **$this** vai acessar valores diferentes de nome, pois diz respeito **ao valor do atributo da instância.**

4. Métodos estáticos e "self"

Métodos estáticos **são métodos que pertencem a uma classe** e **não a uma instância específica dessa classe**. Isso significa que você **pode chamar um método estático diretamente na classe,** sem criar uma instância do objeto.

Os métodos estáticos são úteis quando você precisa executar uma ação que não depende do estado de um objeto, mas está relacionada à classe como um todo.

Exemplo - Declaração de atributo e método estáticos.

```php
<?php

class Exemplo{

    public static $propriedadeEstatica = 'Valor estático';

    public static function metodoEstatico() {
        echo "Este é um método estático.";
    }
}
```

Note que, para adicionar um atributo ou método estático basta adicionar a palavra reservada **static** após a definição do modificador de acesso.

E como chamar o recurso estático da classe?

Exemplo - Chamada de método e atributo estático com "self"

```php
<?php

class Conta
{
    //Atributos ou características de uma TV
    private string $nomePessoa;
    private string $cpf;
    private float $saldo = 0;
    private static int $numeroContas = 0;


   public function __construct(string $nomePessoa, string $cpf)
   {
       $this->nomePessoa = $nomePessoa;
       $this->cpf = $cpf;
       self::$numeroContas++;
   }

   public function __destruct()
   {
       self::$numeroContas--;
   }

    public static function printaContaEstatico():int
    {
        return self::$numeroContas;
    }
}

echo Conta::printaContaEstatico(); // 0 obs: Este número aumenta a cada chamada do construtor.
```

Perceba que no fim da classe, chamamos a função **printaContaEstatico()** que retorna o valor de um atributo estático e, utilizamos **self::nomeAtributo** para acessar atributos estáticos.

4.1 Herança

"Quando dizemos que uma classe A é um tipo de classe B, dizemos que a classe A herda as características da classe B e que a classe B é mãe da classe A, estabelecendo então uma relação de herança entre elas " (HENRIQUE, 2023).

Em um exemplo prático: uma **Smart TV** é uma evolução das TVs tradicionais, com **funcionalidades adicionais** que permitem acesso à Internet, aplicativos e streaming de conteúdo. No entanto, **uma Smart TV ainda mantém muitas características de uma TV convencional**, como marca, tamanho e capacidade de ligar e desligar. Assim, podemos criar uma **classe base chamada TV** e uma **classe derivada chamada SmartTV**. A classe **SmartTV herda as propriedades** e **métodos** da **classe TV**, mas também inclui funcionalidades específicas, como navegar na Internet e assistir a serviços de streaming.

Exemplo - Herança da classe SmartTV em TV

```php
1 usage  1 inheritor
class Tv{...}

2 usages
class SmartTv extends Tv
{
    1 usage
    private $sistemaOperacional;

    2 usages
    public function __construct(string $modelo, string $marca, string $tamanho, string $cor, $sistemaOperacional)
    {
        parent::__construct($modelo, $marca, $tamanho, $cor); //Atributos herdados da classe pai.
        $this->sistemaOperacional = $sistemaOperacional; //Atributo da classe/
    }

    no usages
    public function navegarInternet()
    {
        echo "Navegando na internet...";
    }
```

Classe que pai

Usamos: NomeClasse **extends** NomeClassePai para estabelecer uma relação de herança/

```php
<?php
require_once "Tv.php";

$instancia1 = new SmartTv(modelo: "XYZ-123", marca: "Marca A", tamanho: 42, cor: "Preto", sistemaOperacional: "Android");
$instancia2 = new SmartTv(modelo: "YZX-555", marca: "Marca B", tamanho: 46, cor: "Preto", sistemaOperacional: "IOS");


var_dump($instancia1);
var_dump($instancia2);
```

Parâmetros do construtor pai

parâmetro do construtor filho

```
C:\php\php.exe D:\php\curso-qi\poo-php\script.php
object(SmartTv)#1 (5) {
  ["modelo":"Tv":private]=>
  string(7) "XYZ-123"
  ["marca":"Tv":private]=>
  string(7) "Marca A"
  ["tamanho":"Tv":private]=>
  float(42)
  ["cor":"Tv":private]=>
  string(5) "Preto"
  ["sistemaOperacional":"SmartTv":private]=>
  string(7) "Android"
}
object(SmartTv)#2 (5) {
  ["modelo":"Tv":private]=>
```

Atributos herdados

Atributo da classe SmartTv

Note que, no **var_dump()** da instância, todos atributos da classe Tv foram herdados, e também, os métodos, ou seja, poderíamos acessar **$instancia1->getModelo**, assim como, **$instancia1->navegarInternet()**, pois tanto os atributos da classe pai, quanto os atributos da classe filha estão disponíveis, mesmo não estando declarados na própria classe. Isto só é possível por conta da herança.

4.2 Interface

Quando duas (ou mais) classes possuem **comportamentos comuns** que podem ser separados em uma outra classe, dizemos que a "classe comum" é uma interface, que pode

ser "herdada" pelas outras classes. Note que colocamos a interface como "classe comum", que pode ser "herdada", porque **uma interface não é exatamente uma classe,** mas sim **um conjunto de métodos que todas as classes que herdam dela devem possuir (implementar)**. Neste caso, uma interface não é "herdada" por uma classe, mas sim implementada. No mundo do desenvolvimento de software, dizemos que **uma interface é um "contrato"**: uma classe que implementa uma interface deve fornecer uma implementação a todos os métodos que a interface define. (HENRIQUE, 2023).

Exemplo:

Uma interface é uma forma de definir um conjunto de métodos que uma classe deve implementar. No contexto de TVs, podemos criar uma interface chamada Conectividade para representar dispositivos com capacidades de conexão à Internet. Em seguida, implementamos essa interface na classe SmartTV, que é uma especialização da classe base TV. Isso permite que a SmartTV forneça funcionalidades específicas de conexão à Internet, enquanto ainda mantém suas características como uma TV convencional.

Exemplo - Criação de interface

```
1 usage   1 implementation
interface Conectividade
{
    no usages   1 implementation
    public function navegarInternet();
    no usages   1 implementation
    public function assistirStreamingDeVideo(string $servico);
}
```

Dentro do escopo da interface definimos quais métodos devem existir, e inclusive, se algum método recebe um parâmetro específico.

Exemplo - Utilização de interface

```php
class SmartTv extends Tv implements Conectividade
{
    1 usage
    private $sistemaOperacional;

    2 usages
    public function __construct(string $modelo, string $marca, string $tamanho, string $cor, $sistemaOperacional)
    {
        parent::__construct($modelo, $marca, $tamanho, $cor); //Atributos herdados da classe pai.
        $this->sistemaOperacional = $sistemaOperacional; //Atributo da classe/
    }

    no usages
    public function navegarInternet()
    {
        echo "Navegando na internet...";
    }

    no usages
    public function assistirStreamingDeVideo(string $sistema)
    {
        echo "Asistindo streaming de vídeo";
    }
}
```

Perceba que utilizamos a palavra-chave **implements** junto do nome da interface. Se por algum acaso, não existissem o método **navegarInternet()** ou **assistirStreamingDeVideo(string $sistema)** a IDE já apontaria o erro.

Exemplo - Erro na classe por não conter métodos de uma interface

```php
2 usages
class SmartTv extends Tv implements Conectividade
{
    Class must be declared abstract or implement method 'navegarInternet'
    Add method stubs Alt+Shift+Enter   More actions... Alt+Enter

    1 usage
    private $siste

    2 usages
    public function __construct(string $modelo, string $marca, string $tamanho, string $cor, $sistemaOperacional)
    {
        parent::__construct($modelo, $marca, $tamanho, $cor); //Atributos herdados da classe pai.
        $this->sistemaOperacional = $sistemaOperacional; //Atributo da classe/
    }

    no usages
    public function assistirStreamingDeVideo(string $sistema)
    {
        echo "Asistindo streaming de vídeo";
    }
}
```

Perceba que retiramos o método **navegarInternet()**, com isso obtivemos o seguinte erro: “A classe deve implementar o método ‘navegarInternet()’ ”.

5. Diagrama de classe UML

A **linguagem** de **modelagem unificada (UML)**, auxilia na modelagem de sistemas. Um dos tipos mais populares do uso do UML é na elaboração de diagrama de classes. Esta prática de elaboração de diagramas e prototipação não só auxiliam no planejamento do software como também é uma prática bastante recorrente nos trabalhos formais e geralmente é elaborada por engenheiros de software.

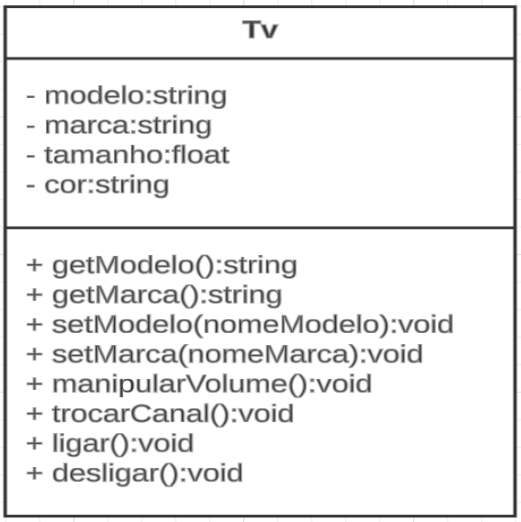
Os diagramas de classes mapeiam de forma clara a estrutura de um sistema ao modelar suas classes, atributos, métodos e a relação entre cada objeto. Saiba mais em: lucidchart.com - O que é um diagrama de classe UML

5.1 Componentes básicos de um diagrama de classes

Um diagrama de classes pode ser composto por três partes.

1. **Parte superior:** contém informações como o nome da classe.
2. **Parte do meio:** contém informações sobre os atributos de uma classe, e também sobre a visibilidade e tipo de cada um dos atributos.
3. **Parte inferior:** inclui as operações (métodos) de uma classe. Exibido em um formato de lista, cada operação por linha. Também é especificada a visibilidade dos métodos.

Exemplo - Diagrama de classe TV:



Perceba que identificamos o **nome** dos atributos e métodos, **tipo** e **visibilidade**. Ou seja:

(+/-/#/~) + **nome** + **:tipo**

Os símbolos que representam a visibilidade de um método ou atributo são:

+ → **Público**

- → **Privado**

→ **Protegido (*protected*)**

~ → **Pacote**

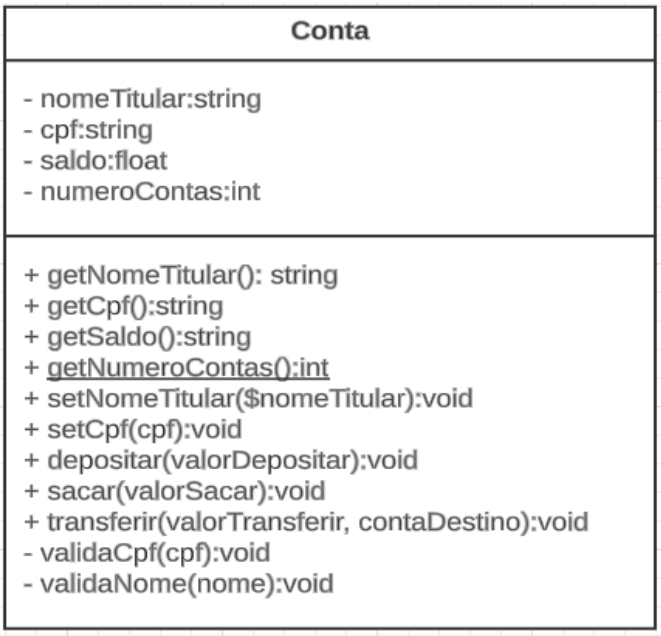
sublinhado → **Estático**

Neste caso, todos os atributos têm a visibilidade privada, enquanto todos métodos desta classe possuem visibilidade pública.

6. Fixando Conhecimento: Desafio

Desenvolva um código utilizando dos conhecimentos da programação orientada a objetos (POO).

1. Tome como base o seguinte diagrama de classe (é possível tomar outra abordagem):



2. Atenção a algumas regras de negócio:

a) O CPF de uma conta deve ser obrigatoriamente no formato "000.000.000-12", qualquer entrada diferente do formato esperado deve ser informada. Dica: pode-se utilizar a função **filter_var()** do PHP com um **regexp**. Pesquise sobre!

b) Um usuário não pode sacar um valor maior que o saldo que possui

c) Um usuário não pode depositar um valor <= (menor igual) 0.

d) Um usuário deve iniciar com saldo = 0 E não pode atribuir um saldo a não ser pelas funções sacar e depositar.

e) Só é possível transferir um valor para um conta válida E deve conter saldo suficiente para realizar a transferência

f) O nome do titular deve conter no mínimo 3 caracteres.

Repositório com possível solução (foi adotada uma abordagem diferente do diagrama das classes)

→ **https://github.com/RafaelR4mos/php-desafios-III**

7. Referências

Múltiplos autores: PHP: **Manual do PHP**, 2023. Disponível em: https://www.php.net/manual/pt_BR/. Acesso em: 10 de outubro de 2023.

HENRIQUE, João: **"POO: o que é programação orientada a objetos?"**, 2023. Disponível em: <https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos>. Acesso em: 10 de outubro de 2023.

FONSECA, Elton: "**Orientação a objetos em PHP**", 2022. Disponível em: <https://www.treinaweb.com.br/blog/orientacao-a-objetos-em-php>. Acesso em: 22 de outubro de 2023.

equipe LUCIDCHART: **"Tutorial de Diagramas de Classes UML"**, 2018. Disponível em: <https://www.youtube.com/watch?v=rDidOn6KN9k>. Acesso em: 23 de outubro de 2023.

equipe LUCIDCHART: **"O que é um diagrama de classe UML?"**, ano não informado. Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-classe-uml>. Acesso em: 23 de outubro de 2023.

Wikipedia: **"Expressão regular"**, ano não informado.
Disponível em: <https://pt.wikipedia.org/wiki/Express%C3%A3o_regular>. Acesso em: 23 de outubro de 2023.