

CURSO COMPLETO DE



85bits
<developer/>

Versão Vídeo Aula disponível em:



<https://www.youtube.com/channel/UC0aGMws3O0PEhxng-CxwLFq>

Sobre o Professor (Dez 2019)

- Celso Araujo Fontes
- Mestre em Sistemas e Computação (IME-RJ)
- Coordenador de Projetos Estratégicos na PGE RJ
- Ex-Professor na graduação do Instituto Infnet

linkedin: <https://www.linkedin.com/in/celsowm/>

email: celsowm@gmail.com

Tópicos do Curso

1. PHP Básico (sintaxe, variáveis, estruturas de controle...)
2. PHP Funções
3. PHP Arrays
4. PHP Orientado a Objetos
5. PHP Entrada de Dados e Formulários
6. Serialização
7. Tratamento de Exceções
8. PHP e Banco de Dados (pdo)
9. Estruturas de Dados com SPL (pilha, fila...)
10. Computação Gráfica com PDO
11. Design Patterns

PHP Básico

- Introdução ao PHP
- Sintaxe Básica
- Variáveis
- Operadores (string, aritméticos, comparação, lógicos, incremento/decremento)
- Estruturas de Controle (if, switch, for, while)
- Função
- Constante

PHP

- Criado por Rasmus Lerdorf in 1994
- Originalmente *Personal Home Page* hoje *PHP: Hypertext Preprocessor*
- Linguagem script (*server-side*)
- Tipagem dinâmica
- Multiparadigma (procedural, reflexiva, orientada a objetos, funcional, imperativa)
- Influenciada por Perl, C, C++, Java e Tcl.
- Cross-platform (Windows, Linux, Mac OSX...)

PHP

Server-side Programming Languages

Most popular server-side programming languages

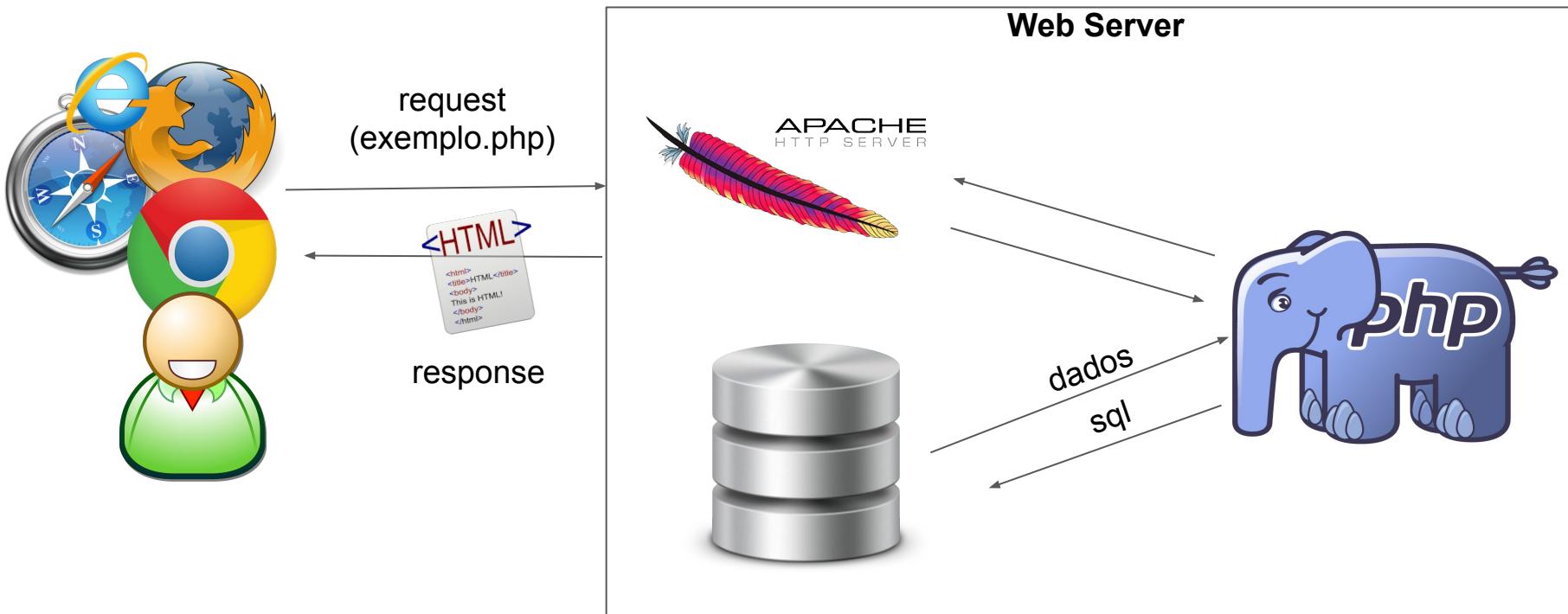
© W3Techs.com	usage	change since 1 February 2017
1. PHP	82.5%	+0.1%
2. ASP.NET	15.2%	-0.1%
3. Java	2.7%	
4. static files	1.5%	
5. ColdFusion	0.6%	

percentages of sites

Ferramentas (sugestões)

- Ferramentas Web:
 - <http://ideone.com/>
 - <https://3v4l.org/>
 - <http://phpfiddle.org/>
 - <http://phptester.net/>
- Ferramentas desktop:
 - <http://www.wampserver.com/>
 - <https://notepad-plus-plus.org/>
 - <https://netbeans.org/>

Exemplo de Arquitetura com PHP



Arquitetura PHP

- Executável que interpreta e executa scripts PHP
- Suporta extensões escritas em C
 - Projetos PECL (PHP Extension Community Library) outros
- Pode utilizar bibliotecas de terceiros escritas em PHP
 - Projeto PEAR PHP Extension and Application Repository) e outros
- Precisa, normalmente, de um serviço HTTP para responder requisições Web

Sintaxe básica (HTML5 + PHP)

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

O objetivo final da maioria dos documentos web PHP será exibir dinamicamente algum conteúdo HTML, pois esta é a linguagem de marcação utilizada pelos navegadores para exibição e formatação dos elementos de uma página web.

Este é um exemplo de uma página web em HTML5 com um trecho de instruções para interpretação do PHP.

O cliente não terá acesso ao código PHP, apenas ao conteúdo final escrito pelo mesmo.

Separação de instruções (; ?>)

```
<?php  
    echo 'Isto é um teste';
```

```
?>
```

```
<?php echo 'Isto é um teste' ?>
```

```
<?php echo 'Nós omitimos a última  
tag de fechamento';
```

Basicamente um código **PHP** começa com o símbolo de menor seguido de um sinal de interrogação e o termo **PHP**.

Cada instrução é separada pelo caractere ponto e vírgula.

Para fechar um bloco de códigos **PHP** utilizamos o símbolo de interrogação e o sinal de menor.

É válido salientar que a instrução de fechamento do **PHP** é opcional quando não existe nenhum outro tipo de código após o bloco de instruções

Comentários (//, /* */, #)

```
<?php  
  
    echo 'Isto é um teste'; // Estilo de  
comentário de uma linha em c++  
  
    /* Este é um comentário de múltiplas linhas  
       ainda outra linha de comentário */  
  
    echo 'Isto é ainda outro teste';  
  
    echo 'Um teste final'; # Este é um comentário  
de uma linha no estilo shell  
  
?>
```

Assim como outras linguagens, o PHP permite a escrita de comentários, que permitem ao desenvolvedor escrever textos que possam explicar a outros desenvolvedores o que significa aquele código ou até documentar de forma estruturada elementos da aplicação através do PHPdoc.

require_once & include

- funcionam com caminhos
 - absolutos:
 - “c:\\apache24\\htdocs\\aula\\teste.php”
 - relativos:
 - “teste.php” “..\\aula2\\teste2.php”
- quando o arquivo não existe:
 - include: gera erro mas o script continua
 - require: erro fatal e o script é encerrado
- require_once:
 - A instrução require_once () é idêntica a require (), exceto que o PHP verificará se o arquivo já foi incluído e, em caso afirmativo, não incluirá (require) novamente.

Variáveis e Tipos “básicos”

```
<?php  
  
$minha_var; // null  
$texto = "Ola Mundo !"; // string  
$x = 5; // integer  
$y = 10.5; // float ou double  
$verdadeiro = true; // boolean  
$nulo = null; // null
```

O elemento básico de qualquer linguagem de programação é variável.

Variáveis são basicamente locais onde as informações são definidas ou referenciadas. No PHP as variáveis começam com o símbolo do cifrão (\$) seguido pelo nome da variável.

Se nenhum valor é definido para esta variável, ela começa automaticamente com o valor NULL ou nulo.

Para definir um valor basta utilizar o símbolo de igual seguido do valor que pode ser um destes tipos primitivos ou também um array ou uma referência a um objeto.

Como o PHP é uma linguagem de tipagem dinâmica, não é necessário que o desenvolvedor defina um tipo para a variável, pois a mesma poderá ter seu tipo alterado de acordo com os valores a serem definidos na execução do código.

Variáveis (diferença entre caixas)

```
<?php  
  
//duas variáveis diferentes  
$verdadeiro = true;  
$Verdadeiro = false;
```

É válido salientar que os nomes de variáveis no PHP fazem distinção entre letras maiúsculas e minúsculas (caixas alta e baixa).

Portanto, neste exemplo, existem duas variáveis distintas pois uma possua uma letra em caixa diferente da outra.

Echo e Print

```
<?php  
echo("Olá Mundo!");  
echo "Olá Mundo !";  
echo "Olá ", "Mundo!";
```

```
<?php  
print("Olá Mundo!");  
print "Olá Mundo!";  
$ret = print "Hello World";  
print("Olá mundo! ".$ret);
```

Outro recurso fundamental de qualquer ambiente de desenvolvimento é a **saída padrão**, isto é, uma forma para se exibir conteúdo para o usuário. No PHP podemos utilizar tanto o echo e como o print. Como nenhum deles é exatamente uma função PHP, o uso do parêntese é opcional.

As diferenças entre eles são que o echo é ligeiramente mais rápido e pode aceitar mais de um parâmetro enquanto o print tem valor de retorno 1 e pode ser usado como uma função.

Strings ('Single quoted' e “Double quoted”)

```
<?php  
  
$versao = 7;  
  
echo 'PHP versão $versao';  
//saída: PHP versão $versao  
  
echo "PHP versão $versao";  
//saída: PHP versão 7
```

Para definir uma *string* ou texto, o PHP permite utilizar dois diferentes tipos de caracteres delimitadores, o apóstrofo ou as aspas.

Quando uma string é delimitada por apóstrofos, nomes de variáveis serão considerados partes integrantes do texto.

Já com o uso das aspas, nomes de variáveis serão interpretados e seus valores e não seus nomes é que serão considerados para integrar a *string*.

'NOWDOC' e HEREDOC

```
<?php  
  
$versao = '7';  
  
$nowdoc = <<<'NOW'  
    PHP versão $versao!  
NOW;  
  
$herec = <<<'HERE'  
    PHP versão $versao!  
HERE;  
  
echo $nowdoc;  
echo $herec;
```

NOWDOC são para cadeias de caracteres com aspas simples o que HEREDOC são para cadeias de caracteres com aspas duplas.

Um NOWDOC é especificado de forma semelhante a um HEREDOC, mas nenhuma análise é feita dentro de um NOWDOC.

A construção é ideal para incorporar código PHP ou outros grandes blocos de texto sem a necessidade de usar “escape” (escapar).

Operadores de String (concatenação “.” e “.=”)

```
<?php  
  
$a = "Olá ";  
  
$b = $a . "Mundo!";  
  
$a = "Hello ";  
  
$a .= "World!";
```

Um importante recurso para a manipulação de duas ou mais strings é o operador de concatenação.

O sinal de ponto no PHP é utilizado com este propósito: permitir a união de uma ou mais strings.

Ele também pode ser utilizado com o símbolo de igual para permitir que o PHP faça a concatenação do conteúdo já presente na variável à esquerda do ponto com o conteúdo a direita do sinal de igual.

Operadores Aritméticos

Operador	Nome	Exemplo	Resultado
+	Adição	$\$x = 2 + 2;$	4
-	Subtração/Negação	$\$x = 4 - 2;$	2
*	Multiplicação	$\$x = 2 * 10;$	20
/	Divisão	$\$x = 15 / 5$	3
%	Módulo (sobra da divisão)	$\$x = 5 \% 2$	1
**	Exponencial	$\$x = 4 ** 4;$	256

Outro conjunto de operadores comumente disponível nas linguagens de programação são os operadores aritméticos.

Ressaltando que além das quatro operações básicas da aritmética, no PHP, existe também o operador de módulo, que permite obter o valor da sobra de uma divisão e o operador de exponencial.

Ordem de precedência

```
<?php  
//PHP usa a ordem de precedência  
conhecida como BODMAS
```

```
$x = 1 + 2 * 3;
```

```
$y = (1 + 2) * 3;
```

```
echo $x; //7
```

```
echo $y; //9
```

B	RACKETS	() [] { }
O	RDER POWER OF	$\sqrt{ } ()^2$
D	IVIDE	/ \div
M	ULTIPLY	* X
A	DDITION	+
S	UBTRACTION	-

unset()

```
<?php  
  
$a = 1;  
$b = $a;  
unset($a);  
echo $a; //Notice: Undefined variable: a  
echo $b;
```

O unset() destrói variáveis previamente especificadas.

O comportamento de unset() dentro de uma função pode variar dependendo do tipo de variável que você está tentando destruir.

Estrutura de Controle (If)

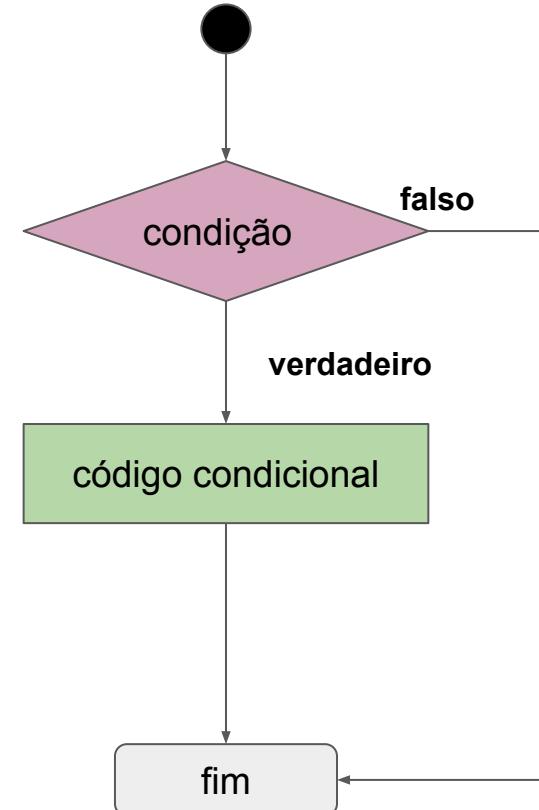
- Como toda linguagem de alto nível, PHP possui um conjunto estruturas de controle que permite à aplicação reagir a diferentes ações ou resultados.
- A primeira e mais importante é o IF (se).
- O if permite testar uma ou mais condições e definir quais serão os códigos para executar caso a condição seja verdadeira ou falsa.
- Neste pequeno exemplo testamos se uma determinada variável possui o conteúdo igual a string “85 bits”, se a condição for verdadeira o código do bloco “então” será executado.

Estrutura de Controle (If)

- Sintaxe básica:

```
if (condição) {  
    //código para executar se a  
    //condição for verdadeira;  
}
```

then/então



fim do código condicional

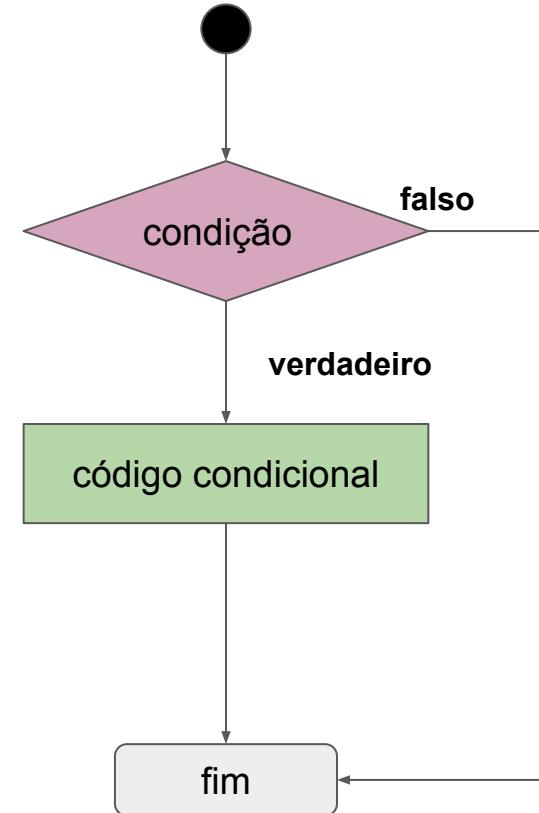
Estrutura de Controle (If)

Neste pequeno exemplo testamos se uma determinada variável possui o conteúdo igual a string “85 bits”, se a condição for verdadeira o código.

Estrutura de Controle (If)

- Exemplo:

```
<?php  
  
if ($nome == "85 bits") {  
    echo "Nome correto !";  
}
```



Operadores de comparação

Operador	Nome	Exemplo (<code>\$a = 5; \$b = 3;</code>)	Resultado
<code>==</code>	Igual	<code>if (\$a == \$b)</code>	false
<code>===</code>	Idêntico	<code>if (\$a === \$b)</code>	false
<code>!=</code>	Diferente	<code>if (\$a != \$b)</code>	true
<code><></code>	Diferente	<code>if (\$a <> \$b)</code>	true
<code>!==</code>	Não idêntico	<code>if (\$a !== \$b)</code>	true
<code>></code>	Maior que	<code>if (\$a > \$b)</code>	true
<code><</code>	Menor que	<code>if (\$a < \$b)</code>	false

Nesta tabela é descrito os operadores de comparação disponíveis no PHP.

Apesar da tipagem dinâmica do PHP é possível fazer uma comparação mais criteriosa utilizando os comparadores idêntico e não idênticos onde não apenas os valores são comparados mas o tipo também.

Operadores de comparação

Operador	Nome	Exemplo (<code>\$a = 5; \$b = 3; \$c = 0</code>)	Resultados
<code>>=</code>	Maior ou Igual	<code>if (\$a >= \$b)</code>	true
<code><=</code>	Menor ou Igual	<code>if (\$a <= \$b)</code>	false
<code><=></code>	Spaceship	<code>if (\$a <=> \$b)</code>	1
<code>??</code>	Diferente (Null coalescing)	<code>\$a ?? \$b ?? \$c</code>	5

Spaceship: Retornar 0 se os valores de ambos os lados são iguais. Retornar 1 se o valor à esquerda é maior. Retornar -1 se o valor da direita é maior.

Já o operador **diferente** ou **Null Coalescing** funciona avaliando as variáveis da esquerda para a direita, e o primeiro valor não nulo será o resultado da comparação.

Operadores Lógicos

Operador	Nome	Exemplo (<code>\$a = 5; \$b = 3; \$c = 0;</code>)	Resultados
<code>&&</code>	E	<code>if (\$a && \$b)</code>	true
<code> </code>	OU	<code>if (\$a \$b)</code>	true
<code>xor</code>	Xor “ou exclusivo”	<code>if (\$a xor \$b)</code>	1
<code>and</code>	E	<code>if (\$a > \$b AND \$c < \$a)</code>	true
<code>or</code>	OU	<code>if (\$a > \$b OR \$a < \$c)</code>	true
<code>!</code>	Not	<code>if(!(\$a > \$b))</code>	false

Operadores Lógicos (“>” maior, igual “==” e “&&”)

Exemplo:

```
<?php  
  
$compra = 500;  
$saldo  = 300;  
$tem_cheque_especial = false;  
  
if( $compra >= $saldo && $tem_cheque_especial == false ) {  
    print 'compra negada';  
}
```

Outro conjunto de operadores do PHP são os operadores lógicos, estes operadores são utilizados para combinar e testar mais de uma condição.

Neste exemplo utilizamos o operador lógico E, representado pelos símbolos duplos de e comercial para testar se ambas as condições são verdadeiras.

Operadores Lógicos (ou “||” e “>=” maior ou igual)

Exemplo 2:

```
<?php  
  
$tempo_trabalho = 24;  
$idade = 66;  
  
if( $idade > 65 || $tempo_trabalho >= 25 ) {  
    echo 'Aposentadoria !';  
}
```

Neste exemplo utilizamos o operador lógico OU, representado pelos símbolos duplos de *pipe* para testar se uma das condições são verdadeiras.

Operadores Lógicos (diferente “!=”)

Exemplo 3:

```
<?php  
  
$convite = 'econômico';  
  
if($convite != 'VIP') {  
    echo 'Acesso negado!';  
}
```

Neste exemplo utilizamos o operador lógico de diferente, representado pelos símbolos de exclamação e igual para testar se uma variável ou expressão é diferente do valor testado à direita.

Operadores Lógicos (igual “==” e idêntico “===”)

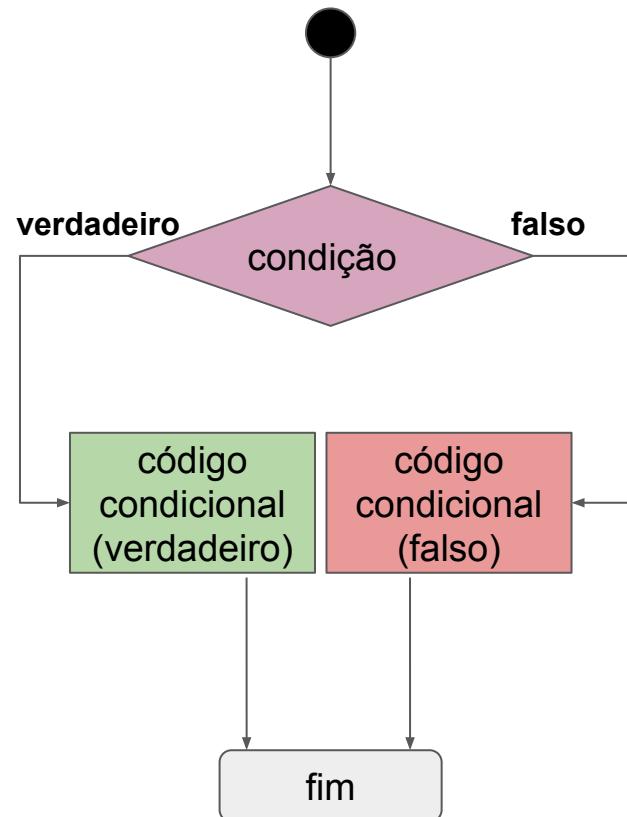
Exemplo 4:

```
<?php  
$x = '300.00';  
$y = 300.00;  
$z = 300.00;  
  
if($x == $y){  
    echo 'iguais'; //são idênticos  
}  
if($x === $y){  
    echo 'idênticos'; //não são idênticos  
}  
if($y === $z){  
    echo 'idênticos'; //são idênticos  
}
```

```
<?php  
$x = " ";  
$y = null;  
$z = 0;  
  
if($x == $y){  
    echo 'iguais'; //são idênticos  
}  
if($x === $y){  
    echo 'idênticos'; //não são idênticos  
}  
if($y === $z){  
    echo 'idênticos'; //não são idênticos  
}
```

Estrutura de Controle (if...else)

Na estrutura de controle IF também é possível definir qual conjunto de códigos será executado se as condições do if não forem verdadeiras, para isto, podemos utilizar a cláusula “else”.

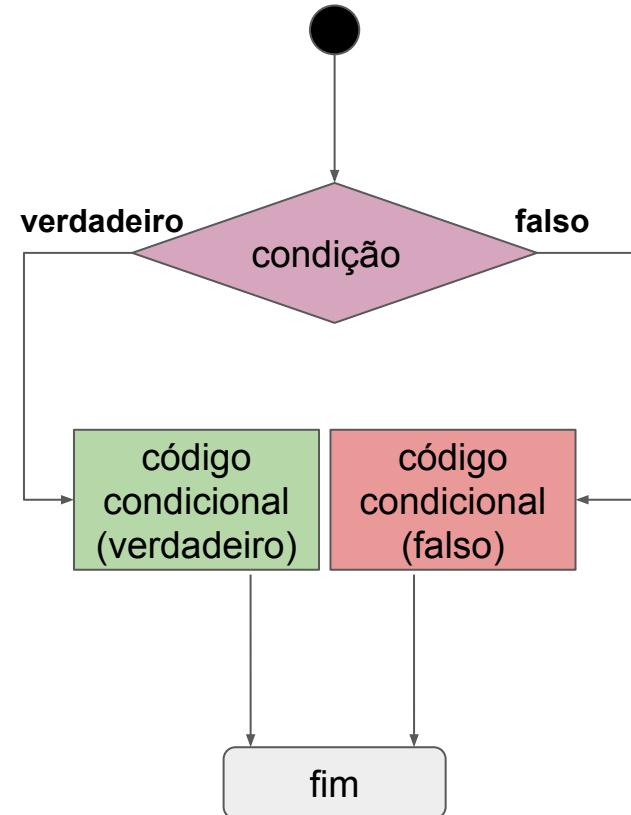


Estrutura de Controle (if...else)

```
<?php  
$a = 1; $b = 2;
```

```
if ($a > $b) {  
    echo "A é maior que B";  
} else {  
    echo "A não é maior do que B";  
}
```

THEN/ENTÃO



Estrutura de Controle (if...elseif)

```
<?php  
if ($a > $b) {  
    echo "a é maior que b";  
} elseif ($a == $b) {  
    echo "a é igual a b";  
} else {  
    echo "a não é maior do que b";  
}
```

Também podemos combinar mais uma estrutura de controle if utilizando a cláusula **elseif**.

Neste exemplo utilizamos apenas um **elseif** mas diversos elseif podem ser usados.

Operador Ternário (?:)

```
<?php
```

```
$a = 40;  
$b = 50;
```

Outra forma de escrita do **if** no PHP é o operador ternário. Nesta forma simplificada utilizamos o **sinal de interrogação** para definir bloco do **então** e o sinal de **dois pontos** para o **else**. O valor de resultado da operação pode ser impresso ou armazenado em uma variável, conforme fizemos neste exemplo.

then/então

else/senão

```
$resultado = ($a >= $b ? 'maior ou igual a b' : 'menor que b');
```

```
echo $resultado;
```

Operadores de Incremento/Decremento

Operador	Nome
++	Incremento
--	Decremento

Incremento e decremento são operadores que as linguagens oferecem para que o desenvolvedor possa manipular o valor de uma variável de forma rápida, permitindo que o seu valor acrescido ou subtraído em 1.

Pré Incremento e Decremento

```
<?php
```

```
$x = 10;  
$y = 10;
```

```
//pré incremento
```

```
echo "valor de x:".++$x; //11 - Incrementa $x mais 1 e então retorna $x  
echo "valor de x:".$x; //11
```

```
//pré decremento
```

```
echo "valor de x":--$y; //9 - Decrementa $y menos 1 e então retorna $y  
echo "valor de x":$y; //9
```

Pós Incremento e Decremento

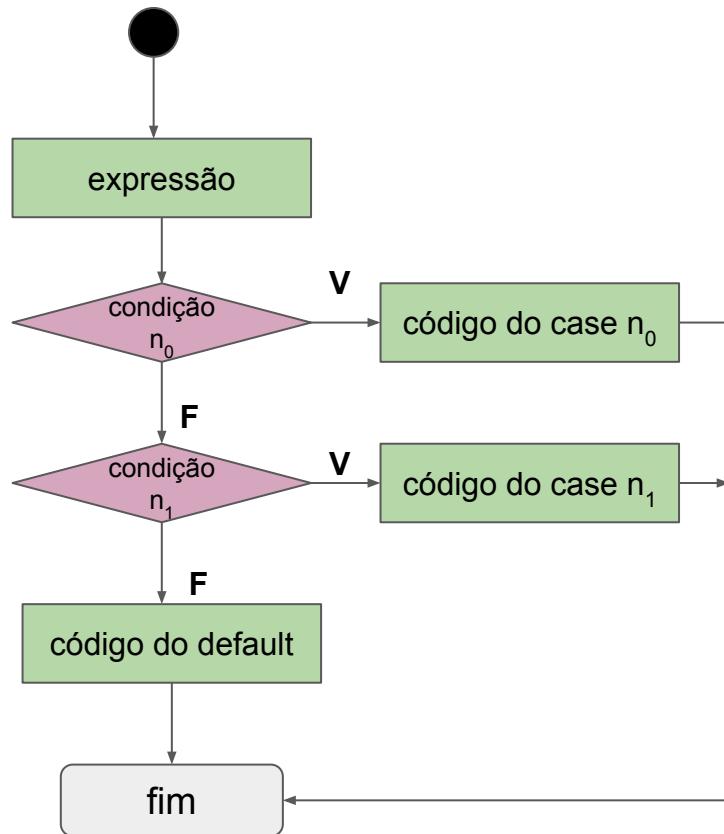
```
<?php  
  
$x = 10;  
$y = 10;  
  
//pós incremento  
echo "valor de x:". $x++; //10 - Retorna $x e então incrementa $x mais 1  
echo "valor de x:". $x; //11  
  
//pós decremento  
echo "valor de x:". $y--; //10 - Retorna $y e então decrementa $y menos 1  
echo "valor de x:". $y; //9
```

Estrutura de Controle (switch)

Apesar das facilidades do if muitas vezes nos deparamos com um número grande de possibilidades para uma determinada variável, então podemos usar uma outra estrutura de controle chamada de switch.

Nela podemos definir um conjunto de rotinas para cada resultado e até um conjunto para se caso nenhum dos resultados for alcançado.

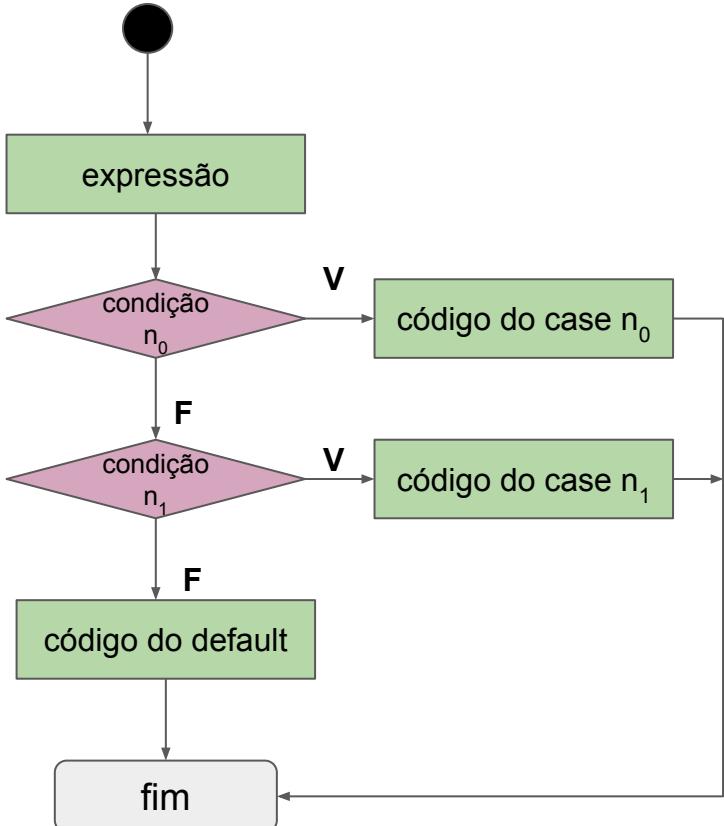
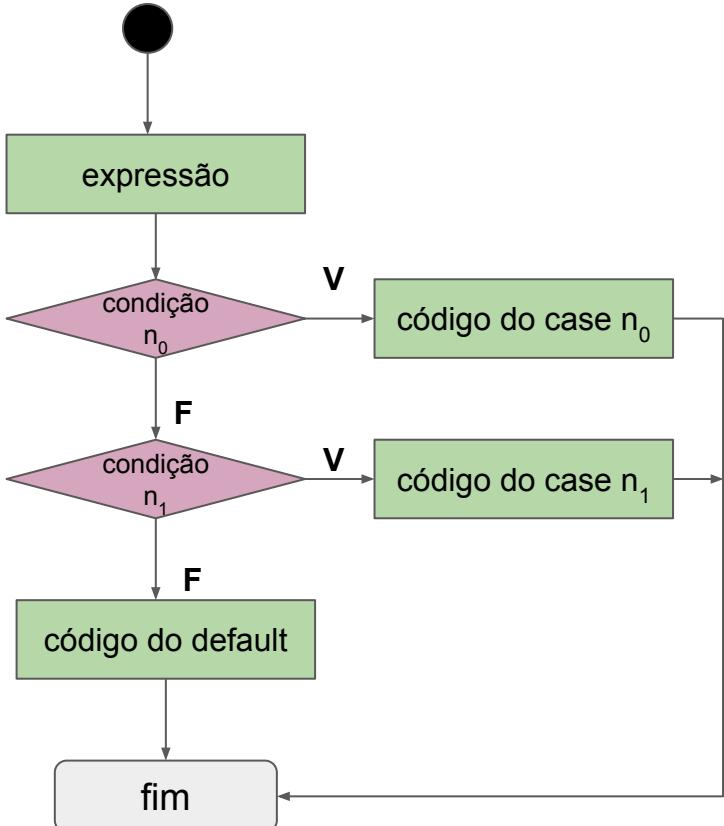
É válido ressaltar a importância do uso do break no final de cada bloco pois se não utilizarmos esta cláusula o PHP continuará testando as outras condições inclusive a *default*.



Estrutura de Controle (switch)

```
<?php
switch ($i) {
    case 0:
        echo "i igual a 0";
        break;
    case 1:
        echo "i igual a 1";
        break;
    default:
        echo "i não é igual nem a 0 ou 1";
}
```

ATENÇÃO !



Estrutura de Controle “loop”

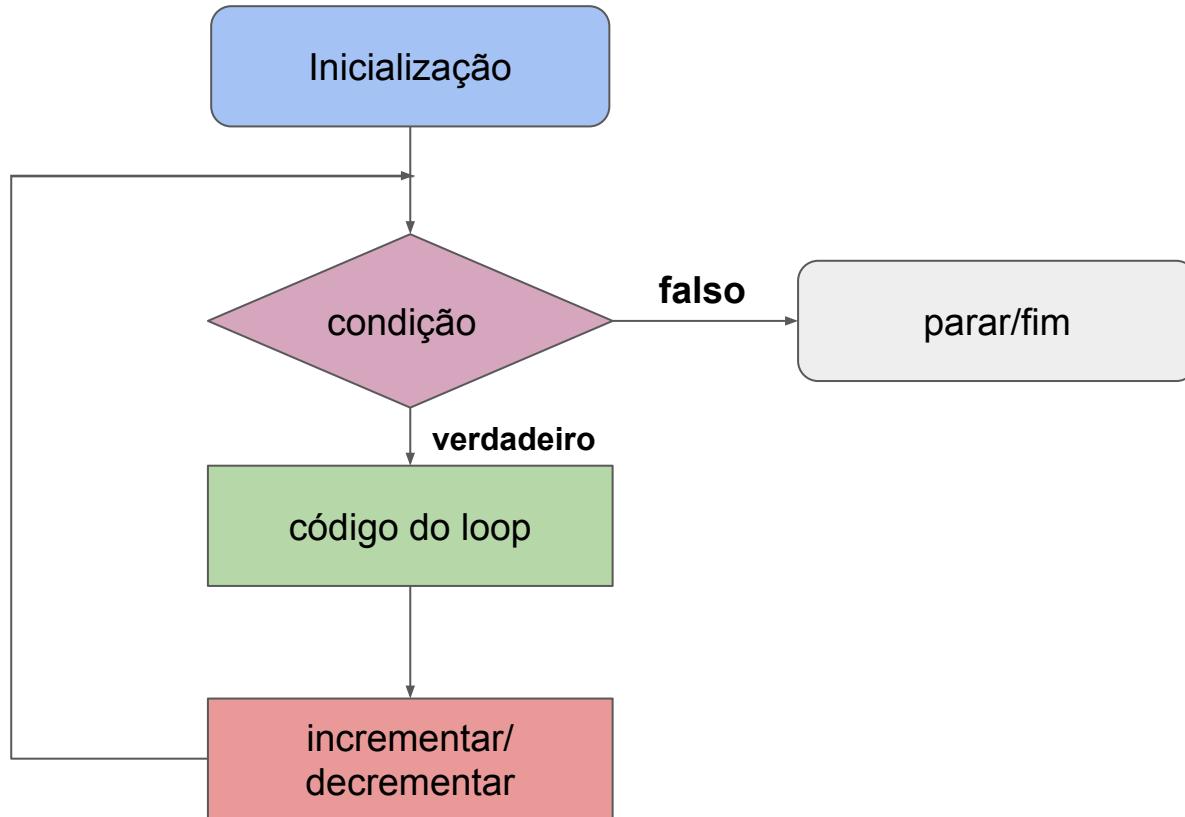
Durante o desenvolvimento de uma aplicação é comum trabalhar com instruções repetitivas, e na maioria das vezes para trabalhar com um conjunto de dados.

Para otimizar este tipo de tarefa utilizamos estruturas de controle conhecidas popularmente como “laço” ou loop.

Este fluxograma representa um exemplo de loop onde o primeiro passo é a inicialização, onde muitas vezes uma variável é usada para o controle do loop. Em seguida, uma condição é testada, e enquanto esta condição for verdadeira todo o bloco do código será executado.

Quando a condição não é mais atendida, o laço é interrompido e as instruções posteriores serão executadas.

Estrutura de Controle “loop”



Estrutura de Controle “loop” (for)

Sintaxe:

```
for (inicialização; condição; incremento) {  
    //código  
} //fim
```

Uma das estruturas de controle de laço do PHP é o **FOR**.

Esta é sintaxe para sua utilização de forma mais básica.

Estrutura de Controle “loop” (for)

```
<?php
```

```
for ($x = 0; $x <= 10; $x++) {  
    echo "O número é: $x <br />";  
}
```

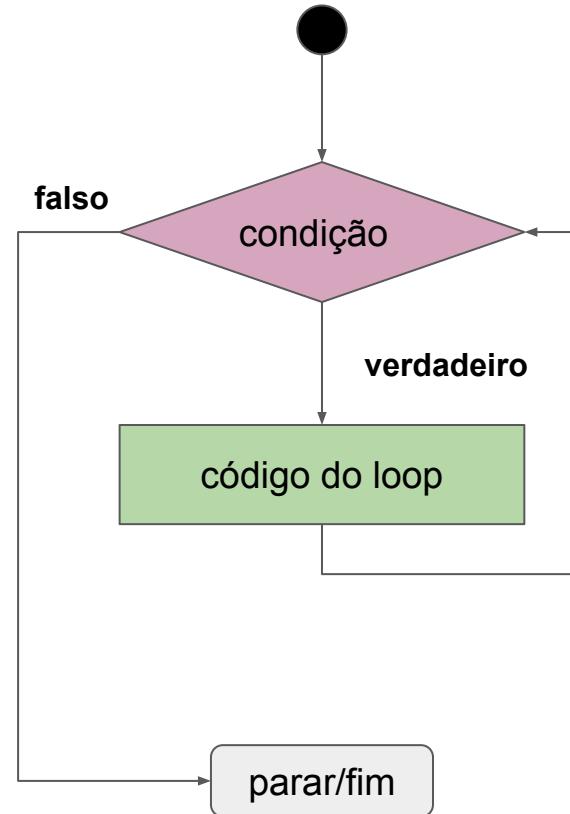
```
0 número é: 0  
0 número é: 1  
0 número é: 2  
0 número é: 3  
0 número é: 4  
0 número é: 5  
0 número é: 6  
0 número é: 7  
0 número é: 8  
0 número é: 9  
0 número é: 10
```

* substituir o </ br> por \r\n se vc estiver usando o *ideone* ou *php cli*.

Estrutura de Controle “loop” (while)

Outra estrutura de laço é o **while** (enquanto), ele permite que um conjunto de instruções sejam repetidas inúmeras vezes “enquanto” uma condição for verdadeira.

Por isso, é comum utilizar alguma variável de controle que será modificada dentro do escopo do while para limitar o número de repetições.



Estrutura de Controle “loop” (while)

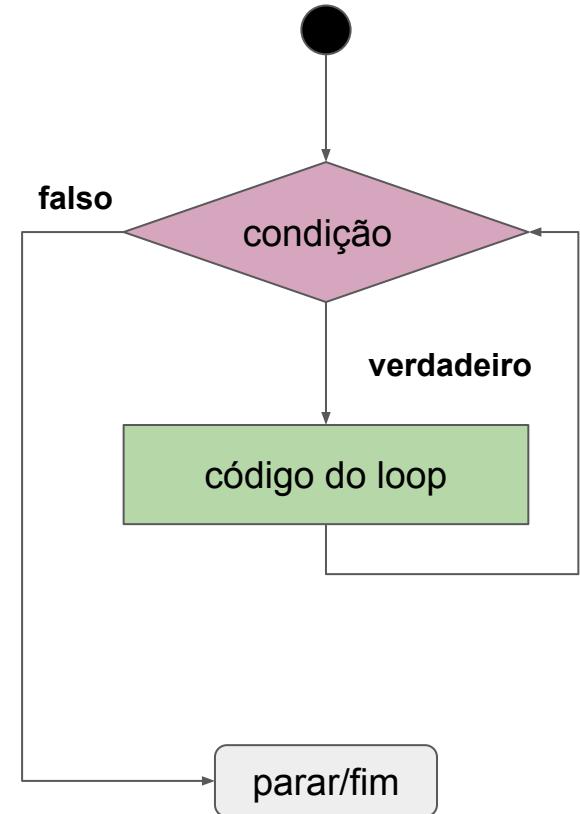
```
<?php
```

```
$i = 1;
```

```
while ($i <= 10) {
```

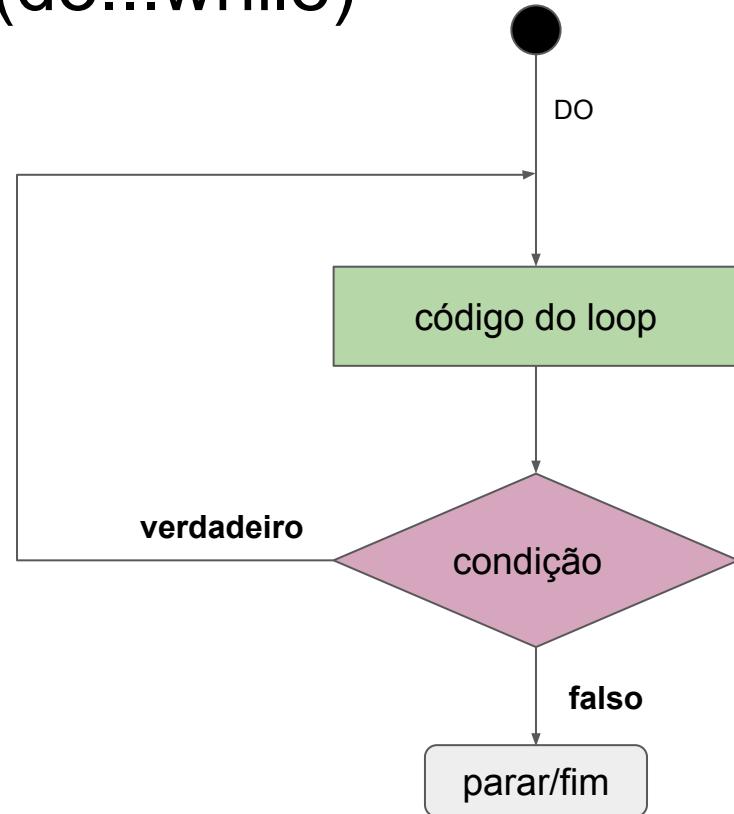
```
    echo $i++;
```

```
}
```



Estrutura de Controle “loop” (do...while)

```
<?php  
  
$x = 1;  
  
do {  
    echo "O número é: $x </ br>";  
    $x++;  
} while ($x <= 5);
```



While (sintaxe alternativa)

```
<?php  
$i = 1;  
while ($i <= 10) {  
    echo $i++;  
}
```

```
<?php  
$i = 1;  
while ($i <= 10):  
    echo $i;  
    $i++;  
endwhile;
```

Função

Sintaxe básica:

```
<?php  
function nomeDaFuncao() {  
    //código  
}
```

Uma função é basicamente um bloco de instruções que podem ser utilizados repetidamente num programa.

Função

```
<?php  
function escreverMensagem() {  
    echo "Olá Mundo!";  
}  
  
escreverMensagem();  
// executar/chamar a função
```

Diferente das outras instruções básicas, uma função nunca será executada imediatamente quando uma página ou script é carregado.

Uma função só será executada apenas quando é feito uma chamada pela mesma.

Função (argumentos/parâmetros)

```
<?php  
function imprimeNome($nome) {  
    echo "Meu nome é $nome !";  
}  
  
imprimeNome("João");  
imprimeNome("Maria");
```

Uma informação pode ser passada para as funções por meio de argumentos. Um argumento é como uma variável (inclusive utilizamos o \$ antes do nome).

Argumentos são especificados após o nome da função, dentro dos parênteses.

Função (argumentos)

```
<?php  
function imprimeNome($nome, $sobrenome) {  
    echo "Meu nome é $nome $sobrenome !";  
}  
  
imprimeNome("João","Oliveira");  
imprimeNome("Maria","das Lurdes");
```

Podemos adicionar quantos argumentos quisermos, basta separá-los com uma vírgula.

Função (argumentos com valor padrão “opcional”)

```
<?php  
function imprimeNome($nome = "Fulano") {  
    echo "Meu nome é $nome !";  
}  
  
imprimeNome();  
imprimeNome("João");  
imprimeNome("Maria");
```

Muitas vezes é necessário que alguns parâmetros de nossa função sejam opcionais.

Para isso, podemos definir valores padrões para os parâmetros que desejamos que se tornem opcionais.

Esta atribuição é feita através de forma muito similar a atribuição de um valor para uma variável usando o sinal de igual seguido do valor desejado.

Lista de argumentos variável (Splat Operator “...”)

```
<?php  
function soma(...$numeros) {  
    echo array_sum($numeros);  
}  
  
soma(1, 2, 3, 4);
```

A partir do PHP 5.6 tornou-se possível incluir o token reticências para indicar uma lista de argumentos e assim informar que a função aceita um número variável de argumentos.

Os argumentos serão passados na forma de um array e por isso, neste exemplo, utilizamos uma função do php chamada array_sum para somar os valores dentro deste array.

Argument Unpacking (Splat Operator “...”)

```
<?php  
function soma($a, $b, $c) {  
    echo $a + $b + $c;  
}  
  
$args = array(2, 3);  
soma(1, ...$args);
```

O splat operator também pode ser utilizado para “desempacotar” um array coleção em argumentos separados.

Função (return)

```
<?php  
function soma($x, $y) {  
    $z = $x + $y;  
    return $z;  
}  
  
echo "5 + 10 = " . soma(5, 10)  
;
```

O *return* permite que uma função possa retornar algum valor.

O *return* retorna o controle do programa para o trecho que executou a chamada.

Função Recursiva

- É basicamente uma função que chama a si mesma.
- Neste exemplo utilizaremos o resultado de cada iteração para multiplicar este resultado vezes ele mesmo menos 1 em uma cadeia de recursividade que será controlada por um “if” que irá garantir que a recursividade não seja eterna.
- **Atenção:** Chamadas de função / método recursivas com mais de 100 a 200 níveis de recursão podem estourar a pilha e causar o encerramento do script atual.
- A recursão infinita é considerada um erro de programação.

Função Recursiva

```
<?php
```

```
function factorial($numero) {
```

```
    echo 'Numero atual'. $numero.' ';
```

```
    if ($numero < 2) {
```

```
        return 1;
```

```
    } else {
```

```
        return ($numero * factorial($numero-1));
```

```
    }
```

```
}
```

```
echo 'Resultado:'.factorial(4);
```

$$= 4 * 3$$

$$= 4 * (3 * 2)$$

$$= 4 * (3 * (2 * 1))$$

$$= 4 * (3 * 2)$$

$$= 4 * 6$$

$$= 24$$

Type Hint

```
<?php
```

```
function soma(int $a, int $b) {  
    return $a + $b;  
}  
  
echo soma(1,2);
```

Tipos usados no
TypeHint:

- bool
- float (php >=7)
- int (php >=7)
- string (php >=7)
- Classe/Interface
- callable
- self
- array

Type Hint + Splat Operator (...)

```
<?php  
function soma(int ...$numeros) {  
    echo array_sum($numeros);  
}  
  
soma(1, 2, 3, 4);  
  
soma(1, 2, 3, 4, 5, 6, 7);
```

O *type hint* ainda pode ser usado em conjunto com o Splat Operator para especificar qual será o tipo dos valores de um determinado conjunto.

Função (declaração de tipo de retorno “：“)

```
<?php  
function soma($a, $b): int {  
    return $a + $b;  
}  
  
echo soma(1, 2.5);
```

A partir do PHP 7 também se tornou possível definir o tipo do valor do retorno de uma função.

Para utilizar este recurso, basta definir tipo da saída (retorno) antecedido pelo símbolo de dois pontos.

declare (strict_types=1)

```
<?php  
  
declare(strict_types=1);  
  
function soma(int $a, int $b) {  
    return $a + $b;  
}  
  
echo soma(1,2);  
echo soma(1,2.5); //se o strict_types  
// for 1 teremos um "TypeError"
```

No PHP também é possível controlar o nível de restrição do Type Hint, isto é, definir se a restrição estará ligada “1” ou desligada “0”.

Vale a pena salientar que o strict_types precisar ser a primeira instrução no script

Nullable types (php 7.1 >=) “?”

```
<?php
```

```
function soma(int $x, ?int $y): ?string
```

```
{
```

```
    return $x+$y;
```

```
}
```

```
echo soma(2,2).PHP_EOL;
```

```
echo soma(2,null).PHP_EOL;
```

```
echo soma(2).PHP_EOL; /*Too few
```

```
arguments to function soma() */
```

Com o advento do PHP 7, foi possível determinar uma nova condição de valor para o tipo de parâmetro (ou retorno de função), usando o “nullable type”.

O Nullable Type permite que além de um valor com o tipo solicitado, o parâmetro também seja capaz de receber null (ou um retorno null).

Passagem por referência “&”

```
<?php  
function somar(&$var) {  
    $var++;  
}  
  
$x=5;  
somar($x);  
echo $x;
```

Com o uso do e comercial, é possível determinar que um parâmetro possa modificar uma variável “mantendo a referência” para a mesma dentro do escopo da variável.

Função Anônima (Closure)

```
<?php  
  
$x = function ($txt) {  
    echo("Hello $txt !");  
};  
  
$x('World');  
$x('PHP');
```

Funções anônimas, também conhecidas como *closures*, permitem a criação de funções que não tem o nome especificado.

Elas são mais úteis como o valor de parâmetros **callback**, mas podem ter vários outros usos.

Vale apenas salientar que toda função anônima atribuída a uma variável termina com ponto e vírgula.

Função Anônima (“use”)

```
<?php  
  
$total = 40;  
  
$soma = function ($x) use ($total) {  
    echo $x+$total;  
};  
  
$soma(50);
```

O termo “use” permite a utilização de variáveis externas dentro do escopo de uma *closure*.

Função Anônima como parâmetro

```
<?php  
  
function add($x,$y){  
    return $x+$y();  
}  
  
echo add (3, function(){return 5;}  );
```



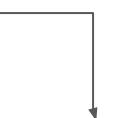
A diagram consisting of a rectangular callout box with a pointer line originating from the word 'add' in the 'echo' statement. The pointer line points down to the start of the 'add' function definition.

Função Anônima como parâmetro

```
<?php
```

```
function teste_closure(callable $a) {  
    $b = 20;  
    return $a($b);  
}
```

```
teste_closure(function($x){  
    echo($x+10);  
});
```



Função Anônima como parâmetro (com iteração de coleção)

```
<?php

function calculaImposto(callable $formula){
    $impostos = [10,22,34,46,58];

    for($i = 0; $i < count($impostos); $i++){
        $impostos[$i] = $formula($impostos[$i]);
    }
    return $impostos;
}

$resultado = calculaImposto(function($valor){
    return $valor * 0.1;
});

var_dump($resultado);
```

Arrow Function (fn)

```
?php  
  
$z = 4;  
function somar(int $x,callable $y){  
    return $x+$y(10);  
}  
  
echo somar(3,function($v) use ($z){return $v*$z;});  
echo PHP_EOL;  
echo somar(3, fn($v)=> $v*$z);
```

Recurso introduzido no PHP 7.4, as arrow functions são uma sintaxe alternativa mais concisa para funções anônimas.

As diferenças de sintaxes são:

- **function** é substituído por **fn**
- **return** é substituído por **=>**
- **{ } e ;** são completamente eliminados.

Funções anônimas e funções de seta são implementadas usando a classe Closure.

Arrow functions tem como vantagem o acesso automático ao escopo de variáveis do pai, eliminando assim a necessidade da clausura “use”.

Entretanto, uma desvantagem das arrow function é sua limitação a apenas uma expressão.

Constantes

```
<?php  
  
define("MINHA_CONSTANTE", "nunca vai mudar");  
define("OUTRA_CONSTANTE", "não importa o case", true);  
define("PI", 3.141592);  
  
echo MINHA_CONSTANTE;  
echo outra_constante;  
echo PI;
```

Diferente das variáveis, o valor de um constante nunca muda e acessamos seu valor sem o uso do cifrão.

Para definir uma constante utilizamos a função define, onde o primeiro parâmetro será um string com o nome da constante, o segundo parâmetro será o valor da constante e por último podemos definir se a constante será insensível ao case, isto é, se a constante pode ser chamada através de letras maiúsculas ou minúsculas independente de como foi definido.

Date e time

```
<?php  
//definindo TimeZone  
date_default_timezone_set('America/Sao_Paulo');  
  
//data  
echo date("d");  
echo date("d/m/y");  
  
//hora  
echo date("h:i:s");  
echo date("d/m/y h:i:s"); //combinando data e hora
```

Função date permite recuperar data (e hora) em diversos formatos.

PHP Array

- Um array em PHP é na verdade um mapa ordenado. Um mapa é um tipo que associa **valores a chaves**.
- Esse tipo é otimizado para vários usos diferentes; ele pode ser tratado como uma **matriz**, **lista** (vetor), **tabela de hash** (uma implementação de um mapa), **dicionário**, **coleção**, **pilha**, **fila** e provavelmente mais.
- Como os valores de array podem ser outros arrays, árvores e arrays **multidimensionais** também são possíveis.
- Pode crescer dinamicamente e ser **iterado** em diferentes direções.

Array (sintaxe básica)

```
<?php
```

```
//No primeiro exemplo é feito utilizando a sintaxe tradicional do php até 5.3
```

```
$array = array("maria", "joão", 2, 3.5);
```

```
/*A partir do php 5.4 é possível utilizar esta sintaxe simplificada conforme o  
segundo exemplo*/
```

```
$array = ["maria", "joão", 2, 3.5];
```

0	1	2	3
"maria"	"joão"	2	3.5

Array (chaves)

```
<?php
```

```
//definindo manualmente as chaves
```

```
$array = [ 1=>"maria" , 2=>"joão" , 'valor'=>2 , null => 3.5 ];
```

```
echo $array[1];
echo $array[2];
echo $array['valor'];
echo $array[null];
```

1	2	'valor'	null
"maria"	"joão"	2	3.5

```
//sem definição das chaves
```

```
$array2 = [ 'maria' , 'joao' ];
```

```
echo $array2[0];
echo $array2[1];
```

0	1
"maria"	"joão"

Arrays Multidimensionais

```
<?php
```

```
$shop = [  
    ["camisa", 9.25 , 15],  
    ["short", 19.75 , 25],  
    ["tênis", 89.15 , 7]  
];
```

0	1	2
0	1	2
camisa	9.25	15
short	19.75	25
tênis	89.15	7

```
<?php  
echo $shop[0][0]." custa ".$shop[0][1]." e existem ".$shop[0][2]." em estoque";  
echo $shop[1][0]." custa ".$shop[1][1]." e existem ".$shop[1][2]."em estoque";
```

Arrays (Multidimensionais e Associativos)

```
<?php  
$shop = [  
    [  
        "Produto" => "camisa",  
        "Preco" => 9.25,  
        "Quantidade" => 15  
    ],  
    [  
        "Produto" => "short",  
        "Preco" => 19.75,  
        "Quantidade" => 25,  
    ],  
    [  
        "Produto" => "tênis",  
        "Preco" => 89.15,  
        "Quantidade" => 7  
    ];  
];
```

0	1	2
Produto	Preco	Qua ntida de
camisa	9.25	15
short	19.75	25
tênis	89.15	7

```
echo $shop[0]['Produto'];
```

Iteração/loop de Arrays (for)

```
<?php
```

```
$array = ["Brasil","Argentina","Bolívia","Venezuela"];
```

```
for ($i = 0; $i < count($array); $i++) {
```

```
    print $array[$i];
```

```
}
```

Iteração de Arrays (foreach)

```
<?php
```

```
$array = ["Brasil","Argentina","Bolívia","Venezuela"];
```

```
foreach ($array as $pais) {  
    print $pais;  
}
```

Iteração de Array (foreach chave - valor)

```
<?php
```

```
$array = [ 10 => "Brasil", 20 => "Argentina", 30 => "Bolívia", 99 => "Venezuela"];
```

```
foreach ($array as $key => $pais) {  
    print "$key - $pais";  
}
```

Iteração de Arrays Multidimensionais

```
<?php
```

```
$pessoas = [  
    ["nome" => "João", "cpf" => 123],  
    ["nome" => "Maria", "cpf" => 456]  
];
```

```
foreach ($pessoas as $pessoa) {  
    print "nome: ".$pessoa["nome"];  
}
```

Iteração de Arrays Multidimensionais

```
<?php  
$turmas = [  
    ["nome" => "PHP", "alunos" => [  
        "João", "Maria", "Lucas"]],  
    ["nome" => "BD", "alunos" => [  
        "Pedro", "Thiago", "João"]]  
];
```

0			1		
nome	alunos		nome	alunos	
"João"	0	1	2	"João"	0
	"João"	"Maria"	"Lucas"	"João"	"Maria"

Iteração de Arrays Multidimensionais

```
<?php  
  
foreach ($turmas as $turma) {  
    print "nome: ". $turma["nome"]."</ br>";  
    print "alunos:";  
    foreach($turma["alunos"] as $aluno){  
        print $aluno."</ br>";  
    }  
}
```

Debug (print_r, var_dump, var_export)

```
<?php
```

```
$var1 = [ " , false , 42 , ['42'] ];
```

```
print_r($var1);
```

```
var_dump($var1);
```

```
var_export($var1);
```

print_r, var_dump, var_export

print_r()

```
Array
(
    [0] =>
    [1] =>
    [2] => 42
    [3] => Array
        (
            [0] => 42
        )
)
```

var_dump()

```
array(4) {
    [0]=>
    string(0) ""
    [1]=>
    bool(false)
    [2]=>
    int(42)
    [3]=>
    array(1) {
        [0]=>
        string(2) "42"
    }
}
```

var_export()

```
array (
    0 => '',
    1 => false,
    2 => 42,
    3 =>
    array (
        0 => '42',
    ),
)
```

Ordenação de Arrays

```
$carros = ["Chevete", "BMW", "Ferrari"];
```

- Ordem crescente (sort):

```
sort($carros);
```

- Ordem descrecente (rsort):

```
rsort($carros);
```

Ordenação de Arrays Associativos

```
$pessoas = ["Mateus"=>"35", "Marcos"=>"37", "Lucas"=>"43"];
```

- Ordem crescente “valor” (sort):

```
asort($pessoas);
```

- Ordem descrecente “valor” (rsort):

```
arsort($pessoas);
```

Ordenação de Arrays Associativos

```
$pessoas = ["Mateus"=>"35", "Marcos"=>"37", "Lucas"=>"43"];
```

- Ordem crescente “chave” (sort):

```
ksort($pessoas);
```

- Ordem descrecente “chave” (rsort):

```
krsort($pessoas);
```

Array para String (implode)

```
<?php
```

```
$array = ['uva', 'pera', 'abacate'];  
$texto = implode(", ", $array);
```

```
echo $texto; // uva,pera,abacate
```

String para Array (explode)

```
<?php
```

```
$texto = "uva pera abacate";  
$array = explode(" ", $texto);
```

```
echo $array[0]; // uva  
echo $array[1]; // pera
```

Checagem de valor no Array (in_array)

```
<?php
```

```
$frutas = ["uva", "pera", "abacaxi", "laranja"];
```

```
if (in_array("uva", $frutas)) {  
    echo "Tem uva !";  
}
```

Checagem de chave no Array (array_key_exists)

```
<?php
```

```
$frutas = ["uva" => 10.99, "manga" => 4.00];
```

```
if (array_key_exists("uva", $frutas)) {  
    echo "a chave uva se encontra no array !";  
}
```

Executar uma função em cada elemento (array_map)

```
<?php  
$array = [  
    'PHP',  
    'Arrays',  
    'Funções',  
];  
  
function hashtag($x){  
    return "#".$x;  
}  
  
print_r(array_map('hashtag', $array));
```

```
$y = array_map(function ($x){  
    return "#".$x;  
}, $array);  
  
print_r($y);
```

Filtrar elementos do array por condição (array_filter)

```
<?php  
  
$alunos = [  
    ['nome' => 'João', 'nota' => 8],  
    ['nome' => 'Maria', 'nota' => 4.5],  
    ['nome' => 'Thiago', 'nota' => 7]  
];  
$alunosAprovados = array_filter($alunos, function($aluno) {  
    return $aluno['nota'] >= 7;  
});  
var_dump($alunosAprovados);
```

Variáveis para array (compact)

```
<?php
```

```
$nome  = "João";  
$cpf   = "123";  
$rg    = "8888";  
$filhos = ["Maria", "Thiago"];
```

```
$result = compact("nome", "cpf", "rg", "filhos");  
print_r($result);
```

Array para variáveis (list)

```
<?php
```

```
$info = ['Camiseta', 'Azul', 30.65];
```

```
// Listando todas as variáveis
```

```
list($produto, $modelo, $preco) = $info;
```

```
echo "A $produto $modelo custa $preco.\n";
```

Array column (recuperar dados por key)

```
<?php  
$pessoas = [  
    [  
        'id' => 1,  
        'nome' => 'João',  
        'idade' => 34,  
    ],  
    [  
        'id' => 2,  
        'nome' => 'Maria',  
        'idade' => 22,  
    ],  
    [  
        'id' => 3,  
        'nome' => 'Lucas',  
        'idade' => 60,  
    ]  
];
```

```
$nomes = array_column($pessoas, 'nome');  
print_r($nomes);
```

0	1	2
“João”	“Maria”	“Lucas”

Array column (recuperar dados por key com index)

```
<?php  
$pessoas = [  
    [  
        'id' => 6789,  
        'nome' => 'João',  
        'idade' => 34,  
    ],  
    [  
        'id' => 7089,  
        'nome' => 'Maria',  
        'idade' => 22,  
    ],  
    [  
        'id' => 3000,  
        'nome' => 'Lucas',  
        'idade' => 60,  
    ]  
];
```

```
$nomes = array_column($pessoas, 'nome', 'id');  
print_r($nomes);
```

6789	7089	3000
“João”	“Maria”	“Lucas”

Array como Pilha (array_push & array_pop)

```
<?php
```

```
$alunos = ["Maria", "João"]; //1
```

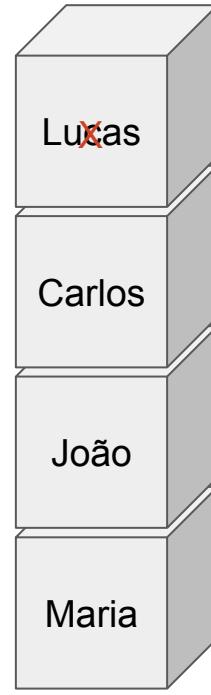
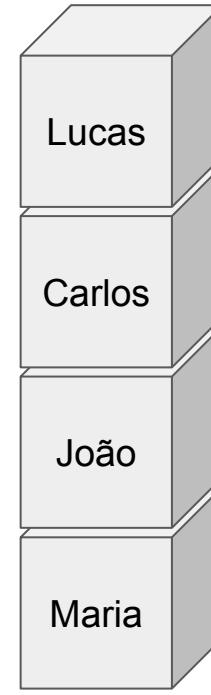
```
array_push($alunos, "Carlos", "Lucas"); //2  
print_r($alunos);
```

```
array_pop($alunos); //3  
print_r($alunos);
```

//2

//3

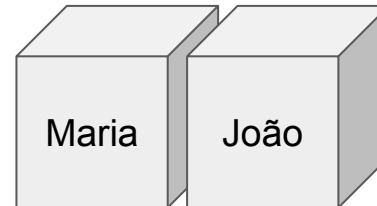
//1



Array como Fila (array_shift)

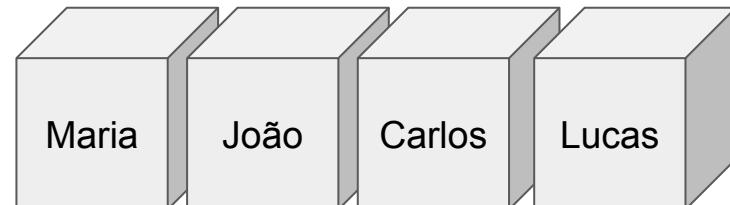
```
<?php  
  
$alunos = ["Maria", "João"];
```

//1



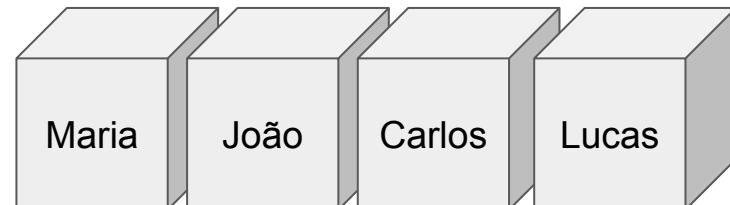
```
array_push($alunos,"Carlos", "Lucas");  
print_r($alunos);
```

//2



```
array_shift($alunos);  
print_r($alunos);
```

//3

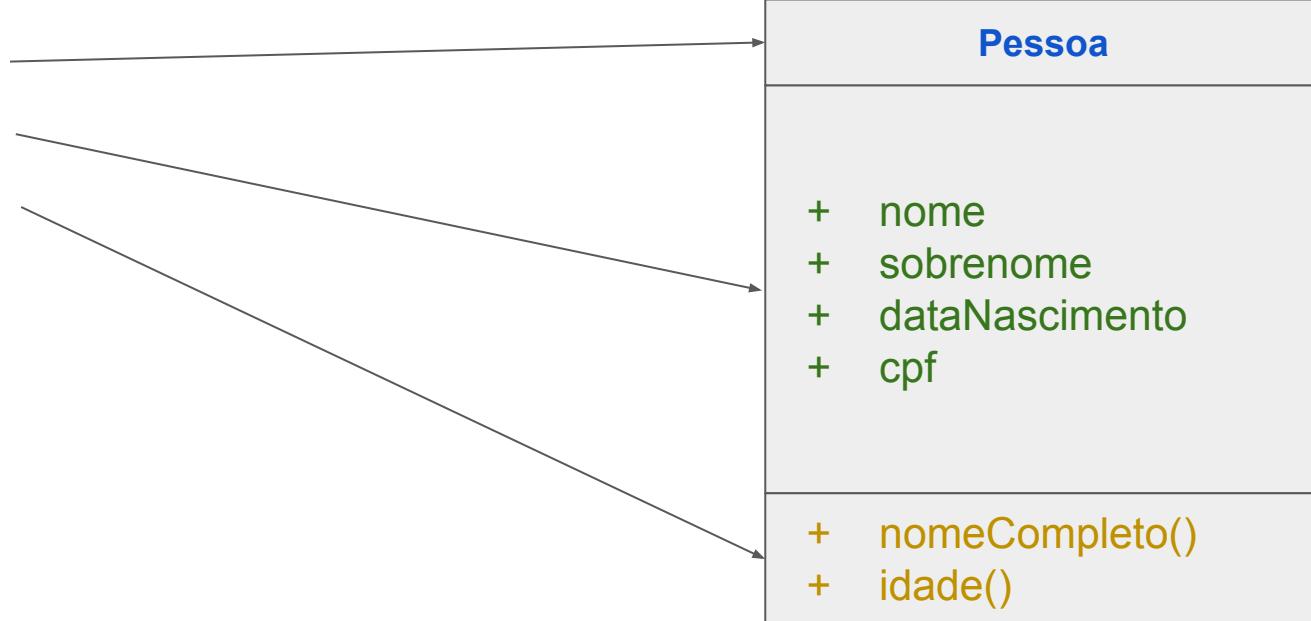


PHP OO

- A programação orientada a objetos (POO) é um paradigma baseado no conceito de "objetos", que podem conter dados, na forma de campos (atributos/propriedades) e instruções de código, na forma de funções (métodos).
- Consolidado no PHP 5, os recursos de Orientação a Objetos suportados pela linguagem estão a inclusão de visibilidade, classes e métodos abstratos e final, métodos mágicos, interfaces, clonagem e *type hint*.
- O PHP trata objetos da mesma maneira que referências ou manipuladores, onde cada variável contém uma referência a um objeto ao invés de uma cópia de todo o objeto.

Classe

- nome
- atributos
- métodos



Classe PHP

```
<?php  
class Pessoa {  
  
    public $nome;  
    public $sobrenome;  
    public $dataNascimento;  
    public $cpf;  
  
    public function nomeCompleto(){  
  
        return $this->nome." ".$this->sobrenome;  
  
    }  
}
```

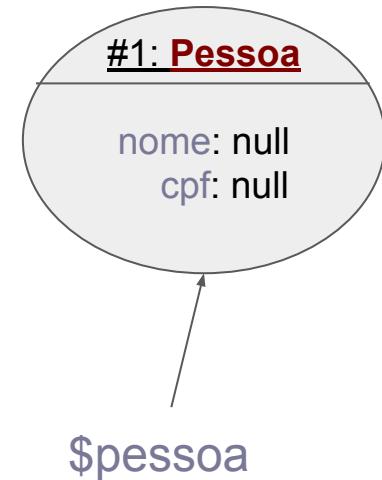
Pessoa

- + nome
- + sobrenome
- + dataNascimento
- + cpf

- + nomeCompleto()

Instanciando (new)

```
<?php  
  
class Pessoa {  
  
    public $cpf;  
    public $nome;  
  
}  
  
$pessoa = new Pessoa();
```



Object Operator (arrow) “->”

```
<?php
```

```
class Pessoa {
```

```
    public $cpf;
```

```
    public $nome;
```

```
}
```

```
$pessoa = new Pessoa();
```

```
$pessoa->nome = 'maria';
```

```
$pessoa->cpf = '123';
```

Objetos (Instâncias)

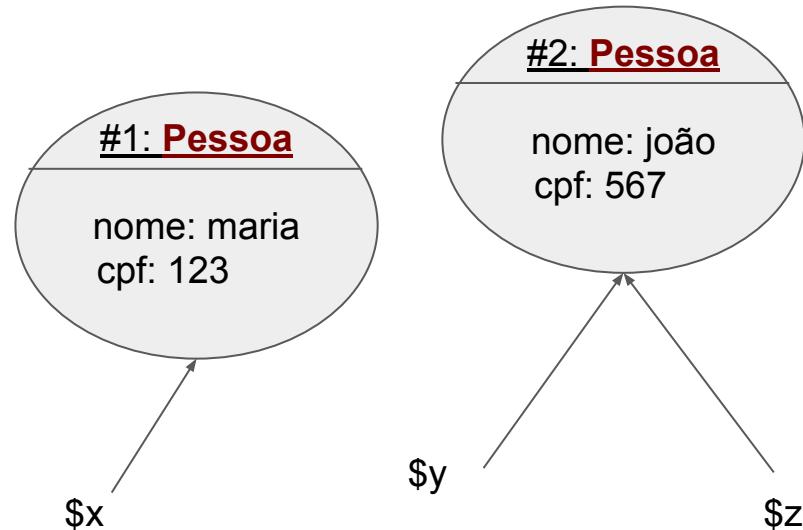
```
<?php  
class Pessoa {  
    public $cpf;  
    public $nome;  
}
```

```
$x = new Pessoa();  
$y = new Pessoa();
```

```
$x->nome = 'maria';  
$x->cpf = '123';
```

```
$y->nome = 'joão';  
$y->cpf = '345';
```

```
$z = $y;  
$z->cpf = '567';
```



```
var_dump($x);  
var_dump($y);  
var_dump($z);
```

clone

```
<?php  
  
class Aluno {  
    public $nome;  
    public $turma;  
}  
$a = new Aluno();  
$a->nome = 'Maria';  
$a->turma = 'PHP';
```

```
$b = clone $a;  
$b->nome = 'Juliana';
```

```
var_dump($a);  
var_dump($b);
```

Métodos

```
<?php  
  
class Pessoa {  
  
    public $cpf;  
    public $nome;  
  
    function exemplo() {  
        return 'olá mundooo!';  
    }  
}  
  
$pessoa = new Pessoa();  
  
echo $pessoa->exemplo();
```

Pessoa

+ cpf
+ nome

+ exemplo()

Referência ao Objeto (\$this)

```
<?php
```

```
class Pessoa {  
  
    public $cpf;  
    public $nome;  
    public $sobreNome;  
  
    function nomeCompleto(){  
        return $this->nome." ".$this->sobreNome;  
    }  
}
```

```
$pessoa = new Pessoa();  
$pessoa->nome      = 'João';  
$pessoa->sobreNome = 'da Silva';  
  
echo $pessoa->nomeCompleto();
```

Método Construtor (__construct)

```
<?php  
class Pessoa {  
  
    public $cpf;  
    public $nome;  
  
    function __construct() {  
        echo "nova pessoa";  
    }  
}
```

```
$pessoa = new Pessoa();  
$pessoa->nome = 'maria';  
$pessoa->cpf = '123';  
print $pessoa->nome;
```

Método Construtor com parâmetros (`__construct`)

```
<?php
```

```
class Pessoa {  
  
    public $cpf;  
    public $nome;  
  
    function __construct($cpf, $nome){  
        $this->cpf = $cpf;  
        $this->nome = $nome;  
    }  
}
```

```
$pessoa = new Pessoa('123','João');  
echo $pessoa->cpf;  
echo $pessoa->nome;
```

Método Destruitor (`__destruct`)

```
<?php  
class Pessoa {  
  
    public $cpf;  
    public $nome;  
  
    function __construct() {  
        echo "nova pessoa";  
    }  
  
    function __destruct() {  
        echo "objeto removido";  
    }  
}
```

```
$pessoa = new Pessoa();  
$pessoa->nome = 'maria';  
$pessoa->cpf = '123';  
print $pessoa->nome;
```

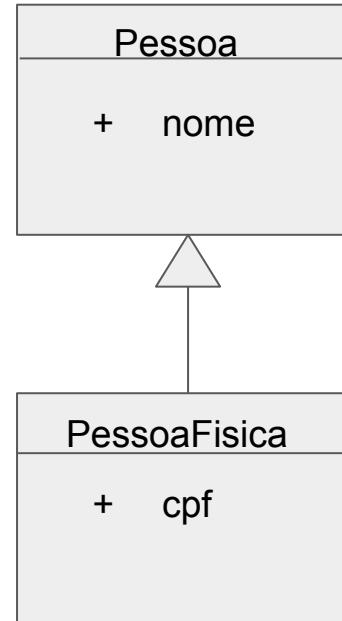
Herança (extends)

```
<?php  
class Pessoa {  
    public $nome;  
}
```

```
class PessoaFisica extends Pessoa {  
    public $cpf;  
}
```

```
$a = new PessoaFisica();  
$a->nome = 'Maria';  
$a->cpf = 123;
```

```
var_dump($a);
```



Method Overriding e Parent

```
<?php  
class Pai {  
  
    public function falar() {  
        echo 'sou seu pai!';  
    }  
  
}  
class Filho extends Pai{  
  
    public function falar() {  
        parent::falar();  
        echo 'nãoooooooo';  
    }  
  
}  
  
$filho1 = new Filho();  
echo $filho1->falar();
```

A substituição de métodos (method overriding), em é um recurso que permite que uma subclasse ou classe filha forneça uma implementação específica de um método que já é fornecido por uma de suas superclasses ou classes pai.

A implementação na subclasse substitui (override) a implementação da superclasse, fornecendo um método que possui o mesmo nome, mesmos parâmetros ou assinatura, e o mesmo tipo de retorno que o método na classe pai.

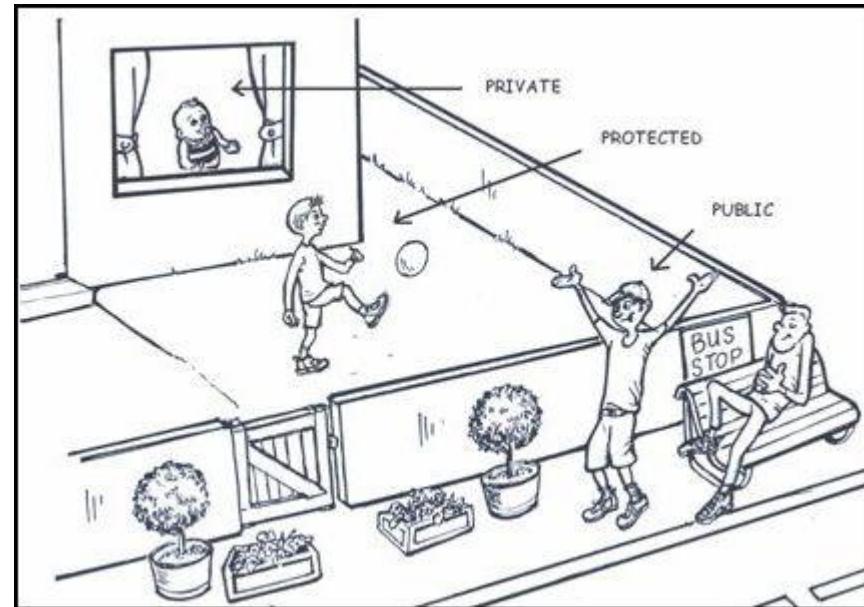
Apesar do recurso de overriding, ainda é possível que a classe filha “execute” um método da classe pai através do recurso **parent**.

PHP OO (visibilidade)

public: torna uma variável / método disponível de qualquer lugar, outras classes e instâncias do objeto.

protected: escopo quando você quiser fazer o seu / função variável visível em todas as classes que estendem a classe atual, incluindo a classe pai.

private: escopo quando você quer que seu / função variável a ser visível apenas em sua própria classe.



Public (Visibilidade)

```
<?php  
class Pai {  
    public $publico;  
    protected $protulado;  
    private $privado;  
}  
  
class Filho extends Pai {}
```

```
$pai = new Pai();  
$filho = new Filho();  
$pai->publico = 'abc';  
$filho->publico = 123;
```

```
var_dump($pai);  
var_dump($filho);
```

- Mesma classe que foi declarada
- Classes que herdam
- As classes que herdam a classe declarada acima.
- Quaisquer elementos estrangeiros fora dessa classe também podem acessar estes elementos

Protected (Visibilidade)

```
<?php  
class Pai {  
    public $publico;  
    protected $protegido;  
    private $privado;  
}  
  
class Filho extends Pai {  
    public function alteraProtegido($valor){  
        $this->protegido = $valor;  
    }  
}  
  
$pai = new Pai();  
$filho = new Filho();  
  
// $pai->protegido = 'xyz'; // Erro !  
// $filho->protegido = 456; // Erro !  
  
$filho->alteraProtegido(456);  
  
var_dump($filho);
```

Quando você declara um método (função) ou uma propriedade (atributo) como protegido, esses métodos e propriedades podem ser acessados por a mesma classe que declarou isso.

As classes que herdam a classe declarada acima. Os membros de outsiders não podem acessar essas variáveis. "Outsiders" no sentido de que eles não são instâncias de objetos da própria classe declarada.

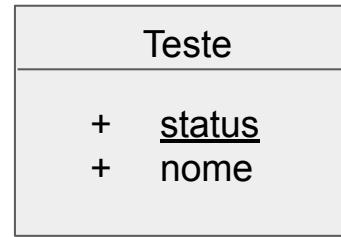
Private (Visibilidade)

```
<?php  
class Pai {  
    public $publico;  
    protected $protulado;  
    private $privado;  
  
    public function exibePrivado(){  
        return $this->privado;  
    }  
    public function alteraPrivado($valor){  
        $this->privado = $valor;  
    }  
}  
  
class Filho extends Pai {}  
  
$filho = new Filho();  
$filho->alteraPrivado(789);  
  
echo $filho->exibePrivado();
```

Quando você declara um método (função) ou uma propriedade (variável) como privada, esses métodos e propriedades podem ser acessados por a mesma classe que declarou isso.

Atributos Estáticos

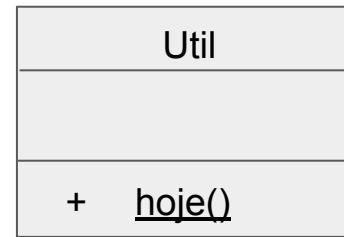
```
<?php  
class Teste {  
  
    public static $status = 'online';  
    public $nome;  
}  
  
$t1 = new Teste();  
$t1->nome = 'Teste 1';  
Teste::$status = 'offline';  
  
var_dump($t1);  
var_dump(Teste::$status);
```



Na programação orientada a objetos, atributo estático é conhecido como "variável de classe". Um atributo estático de uma determinada classe tem seus valores compartilhados em todas as instâncias (objetos) ou diretamente na classe utilizando o operador ::

Métodos Estáticos

```
<?php  
  
class Util {  
  
    public static function hoje() {  
        return date('d/m/y');  
    }  
}  
  
echo Util::hoje();  
$util = new Util();  
echo $util->hoje();
```



De maneira similar, um método estático é “procedimento da classe”. Um método estático de uma determinada classe pode ser acessado em todas as instâncias (objetos) ou diretamente na classe utilizando o operador ::

É válido salientar que é **proibido** utilizar o **\$this** dentro (no contexto) de um método estático.

Class Constants

```
<?php  
  
class Matematica  
{  
    const PI = 3.14;  
  
    function valorDePi() {  
        echo self::PI . "\n";  
    }  
}
```

```
echo Matematica::PI . "\n";
```

```
$mat = new Matematica();  
$mat->valorDePi();
```

Similar as constantes usando o define(), o valor de uma constante nunca pode ser alterado.
O acesso é similar aos atributos estáticos, utilizando o :: porém sem o \$.

Class Constants (visibilidade)

```
<?php

class Matematica
{
    private const PI = 3.14; //disponível a partir do PHP 7.1

    function valorDePi() {
        echo self::PI . "\n";
    }
}

//echo Matematica::PI . "\n"; erro !

$mat = new Matematica();
$mat->valorDePi();
```

Final Method

```
<?php  
class ClasseBase {  
    public function teste() {  
        echo "teste";  
    }  
    final public function maisTeste() {  
        echo "teste";  
    }  
}  
  
class ClasseFilha extends ClasseBase {  
  
//erro:  
    public function maisTeste() {  
        echo "mais teste";  
    }  
}
```

O PHP 5 introduziu a palavra-chave `final`, que previne que classes filhas *sobrescrevam* um método que esteja prefixado em sua definição com a palavra `final`.

Final Class

```
<?php  
  
class Pessoa {  
    public $nome;  
}  
  
final class Professor extends Pessoa {  
    public $matricula;  
}  
  
//erro:  
class ProfessorSubstituto extends Professor {}
```

Se a própria classe estiver definida como **final**, ela não pode ser estendida.

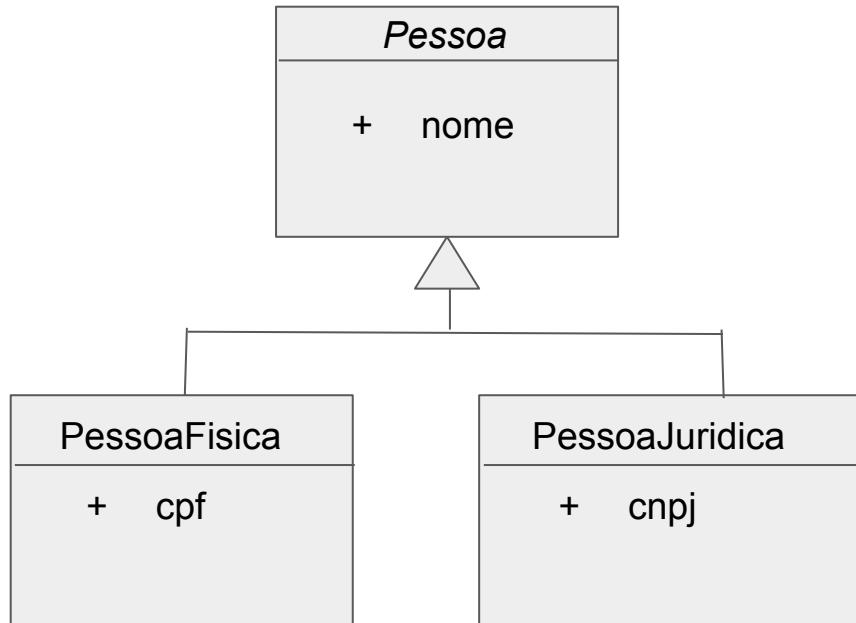
Classe Abstrata

```
<?php  
abstract class Pessoa {  
    public $nome;  
}
```

```
class PessoaFisica extends Pessoa {  
    public $cpf;  
}
```

```
class PessoaJuridica extends Pessoa {  
    public $cnpj;  
}
```

```
$pf = new PessoaFisica();  
$pj = new PessoaJuridica();  
//$p = new Pessoa(); //erro !
```



Método Abstrato

```
<?php

abstract class Pessoa{
    public $nome;
    abstract function exibirDocumento();
}

class PessoaFisica extends Pessoa {
    public $cpf;

    //Métodos abstratos precisam ser implementados nas classes filhas
    public function exibirDocumento(){
        return $this->cpf;
    }
}
```

```
$pf = new PessoaFisica();
$pf->cpf = 123;
echo $pf->exibirDocumento();
```



Métodos abstratos precisam ser implementados nas classes filhas.

Interface

- Interfaces permitem a criação de códigos que especificam quais métodos uma classe deve implementar, sem definir como esses métodos serão tratados.
- Interfaces permitem também que classes de diferentes hierarquias possam ter comportamentos similares.
- Interfaces são definidas da mesma forma que classes, mas com a palavra-chave ***interface*** substituindo ***class*** e com nenhum dos métodos tendo seu conteúdo definido.
- Todos os métodos declarados em uma interface devem ser públicos, essa é a natureza de uma interface.
- Interfaces PHP suportam *herança múltipla* e *constantes*

Interface

```
<?php
```

```
interface Formatador {
```

```
    public function formatar();
```

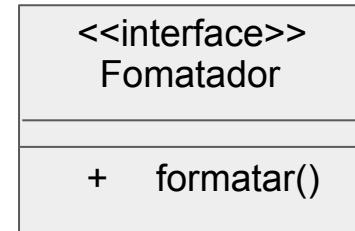
```
}
```

```
class Html implements Formatador {
```

```
    //erro: é obrigatório implementar o método formatar !
```

```
    //obs.: exceção se a classe Html fosse abstrata
```

```
}
```



Interface (herança múltipla)

```
<?php

interface Paginador{
    public function paginar();
}

interface Arvore{
    public function getNoRaiz();
}

interface ArvorePaginavel extends Paginador, Arvore {

}
```

```
class TabelaPessoa implements ArvorePaginavel
{
    //deverá implementar dois métodos
}
```

Interface (usando typehint)

```
<?php

interface Sepultavel {
    public function calcularValorDoCaixao(float $valor): float;
}

class Felino implements Sepultavel {
    public $peso;
    public function calcularValorDoCaixao(float $valor): float{
        return $valor * ($this->peso * 0.3);
    }
}

class Pessoa implements Sepultavel {
    public $altura;
    public $peso;
    public $circunferenciaAbdominal;
    public function calcularValorDoCaixao(float $valor): float{
        return $valor * (($this->altura * $this->peso) / 2 +
                        $this->circunferenciaAbdominal);
    }
}
```

Neste exemplo, classes de diferentes Taxonomias implementam uma mesma Interface

Interface (usando typehint - parte 2)

```
<?php

class Sepultamento {

    public $cotacao = '4.05';

    public function enterrar(Sepultavel $sepultavel): float{
        return
        $sepultavel->calcularValorDoCaixao($this->cotacao);
    }

    }

    $gato = new Felino();
    $gato->peso = 3.4;
    $s = new Sepultamento();
    echo $s->enterrar($gato);
}
```

Um método de uma classe (ou uma função) pode usar o type int com o nome de uma interface para que seja garantido que os comportamentos de um objeto (que implementam esta interface) possam ser evocados.

Trait

- Traits são mecanismos para reutilização de código em linguagens de herança únicas, como PHP.
- Uma Trait destina-se a reduzir algumas limitações de herança única, permitindo que um desenvolvedor reutilizar conjuntos de métodos livremente em várias classes independentes que podem viver em diferentes taxonomias de classes.
- Não é possível instanciar um Trait por conta própria.
- É uma adição à herança tradicional e permite a composição horizontal do comportamento, isto é, a aplicação de membros de classe sem exigir herança.
- Traits podem conter métodos abstratos

Trait

```
<?php  
trait MinhaTrait {  
    public $nome = 'trait';  
    public function ola() {  
        return 'ola '.$this->nome."!!!";  
    }  
}  
  
class Teste {  
    use MinhaTrait;  
}
```

```
$x = new Teste();  
echo $x->nome;  
echo $x->ola();
```

\$y = new MinhaTrait(); //Erro! Não é possível instanciar uma Trait

Trait - (múltiplo uso)

```
<?php

trait Telefone {
    public function ligar() {
        echo 'alo...';
    }
}

trait Camera {
    public function filmar() {
        echo 'gravando...';
    }
}
```

```
class Smartphone {
    use Telefone, Camera;
    public function videoChamada() {
        echo $this->filmar().$this->ligar();
    }
}
```

```
$o = new Smartphone();
$o->ligar();
$o->filmar();
$o->videoChamada();
```

Trait (precedência/instead of)

```
<?php

trait Video {
    public function tag() {
        echo '<video> ';
    }
}

trait Audio {
    public function tag() {
        echo '<audio>';
    }
}
```

```
class Media {

    use Audio, Video {
        Video::tag insteadof Audio;
    } //sem esta resolução ocorrerá um Fatal Error.
}

$media = new Media();
$media->tag(); //resultado: <video>
```

Trait (alias “as”)

```
<?php

trait Video {
    public function tag() {
        echo '<video> ';
    }
}

trait Audio {
    public function tag() {
        echo '<audio>';
    }
}
```

```
class Media {
    use Audio, Video {
        Video::tag insteadof Audio;
        Audio::tag as tagAudio;
    }
}

$media = new Media();
$media->tag();
$media->tagAudio();
```



Trait (sintaxe para resolução de conflitos)

- Precedência:
 - ClasseA::metodo insteadof ClasseB
- Alias:
 - ClasseA::metodo as metodoN2

Trait (método abstrato)

```
<?php

trait MinhaTrait {
    abstract function fazAlgo();
    function mensagem(){
        echo "método concreto";
    }
}

class MinhaClass {
    use MinhaTrait;
    function fazAlgo(){
        echo "implementação de fazAlgo em MinhaClass";
    }
}
```

Classe Anônima

```
<?php  
  
$pessoa = new class {  
  
    public $nome = 'Maria';  
    public $cpf   = '754';  
  
    public function getNome(){  
        return strtoupper($this->nome);  
    }  
};
```

```
var_dump($pessoa);  
echo $pessoa->nome;  
echo $pessoa->cpf;  
echo $pessoa->getNome();
```

Classe Anônima (novas instâncias)

```
<?php  
  
$pessoa = new class{  
  
    public $nome = 'Fulano';  
  
    public function oi(){  
        return "Oi {$this->nome} !";  
    }  
};
```

```
echo $pessoa->oi();  
$p1 = new $pessoa();  
$p1->nome = 'Maria';  
echo $p1->oi();
```

Classe Anônima (via function)

```
<?php

function pessoa_oo() {

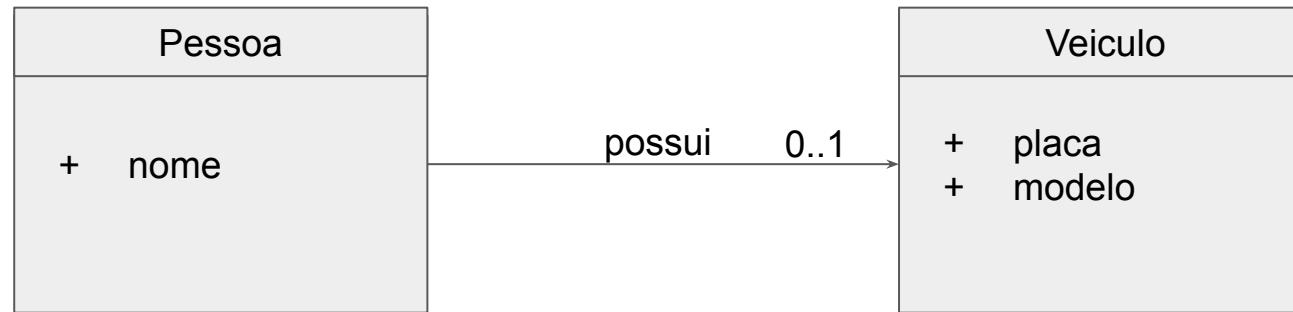
    return new class {
        public $nome = 'Maria';
        public $cpf   = '754';

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

```
$pessoas = pessoa_oo();
echo $pessoas->nome;
echo $pessoas->cpf;
echo $pessoas->getNome();
```

Associação entre objetos

Na programação orientada a objetos, a associação define um relacionamento entre classes de objetos que permite que uma instância de objeto faça com que outra execute uma ação em seu nome. Essa relação é estrutural, porque especifica que objetos de um tipo estão conectados a objetos de outro e não representam comportamento.



Associação entre objetos

```
<?php  
  
class Pessoa {  
    public $nome;  
    public $veiculo;  
}  
  
class Veiculo{  
    public $placa;  
    public $modelo;  
}
```

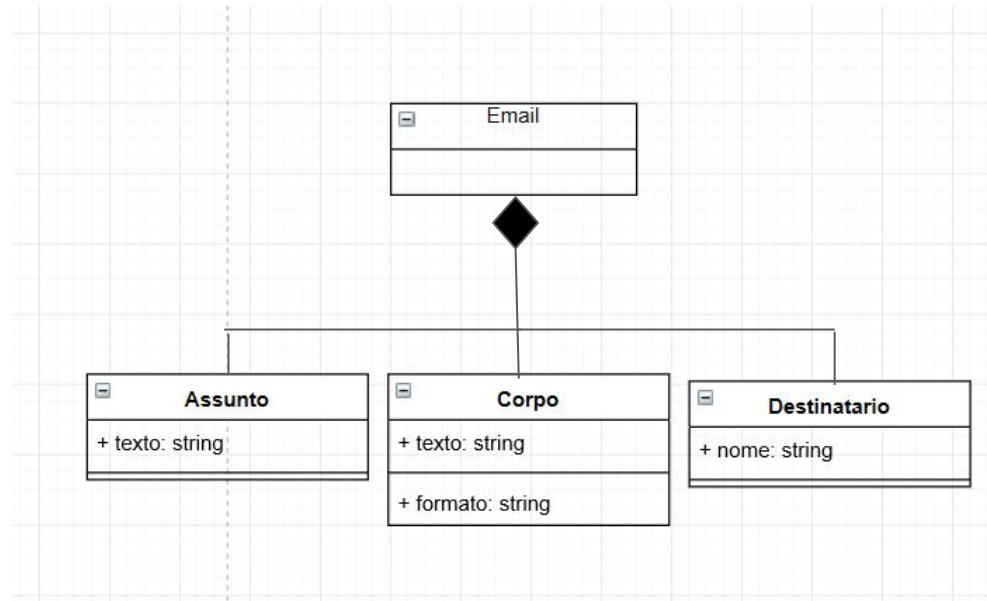
```
$p1 = new Pessoa();  
$p1->nome = 'Maria';  
$p1->veiculo = new Veiculo();  
  
$p1->veiculo->placa = 'ABC123';  
$p1->veiculo->modelo = 'Tesla X1';  
  
var_dump($p1);
```

Composição

- Composição é um tipo de associação onde o objeto composto é o único responsável pela disposição das partes componentes.
- O relacionamento entre o composto e o componente é um relacionamento forte "tem um", pois o objeto composto apropria-se do componente.
- Isso significa que o composto é responsável pela criação e destruição das partes componentes. Um objeto só pode fazer parte de um composto. Se o objeto composto for destruído, todas as partes componentes devem ser destruídas.
- A composição impõe o encapsulamento, pois as partes do componente geralmente são membros do objeto composto.
- A composição é caracterizada na UML pelo uso do losango preenchido.

Composição

```
<?php  
class Assunto {  
    public $texto;  
}  
class Corpo {  
    public $texto;  
    public $formato;  
}  
class Destinatario {  
    public $nome;  
}  
class Email {  
    private $assunto;  
    private $corpo;  
    private $destinatario;  
    public function __construct() {  
        $this->assunto = new Assunto();  
        $this->corpo = new Corpo();  
        $this->destinatario = new Destinatario();  
    }  
}
```



Composição (Versão com classes anônimas)

```
<?php

class Email {
    private $assunto;
    private $corpo;
    private $destinatario;

    public function __construct() {
        $this->assunto = new class {
            public $texto;
        };
        $this->corpo = new class {
            public $texto;
            public $formato;
        };
        $this->destinatario = new class {
            public $nome;
        };
    }
}

public function setAssunto($texto){
    $this->assunto->texto = $texto;
}
//implementar os próximos getters & setters
}

$email = new Email();
$email->setAssunto('Curso de PHP');
```

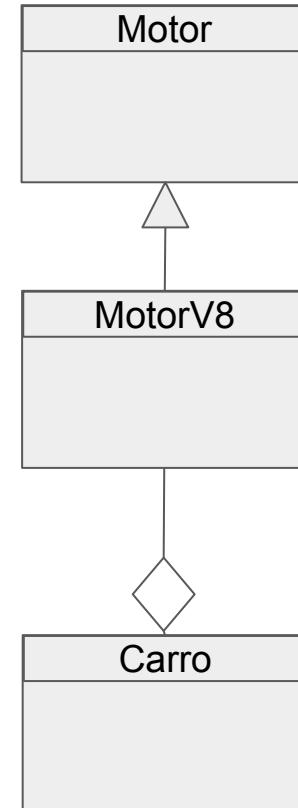
Agregação

- Agregação é um tipo de associação que especifica um relacionamento de todo / parte entre a parte agregada (todo) e o componente.
- Esse relacionamento entre o agregado e o componente é um relacionamento "tem-um" fraco, pois o componente pode sobreviver ao objeto agregado.
- O objeto componente pode ser acessado através de outros objetos sem passar pelo objeto agregado. O objeto agregado não participa do ciclo de vida do objeto de componente, o que significa que o objeto de componente pode sobreviver ao objeto agregado. O estado do objeto componente ainda faz parte do objeto agregado.
- A agregação é caracterizada na UML pelo uso do losango sem vazio.

Agregação

```
<?php
abstract class Motor {}
class MotorV8 extends Motor {

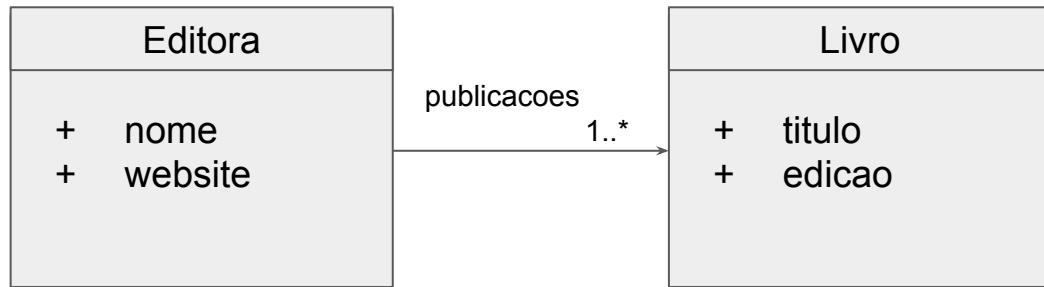
}
class Carro {
    private $motor;
    public function __construct(Motor $motor) {
        $this->motor = $motor;
    }
}
$motorV8 = new MotorV8();
$carro1 = new Carro($motorV8);
```



Associação (1..*)

Uma associação um-para-muitos (ou têm muitos) é um tipo de associação que estabelece uma vinculação onde uma instância da classe do lado esquerdo tem zero ou mais instâncias de outra classe.

Para representar esta multiplicidade, podemos utilizar uma coleção (ex.: array) para “armazenar” / “intermediar” esta relação.



Associação (1..*)

```
<?php

class Livro {
    public $titulo;
    public $dedicao;
}

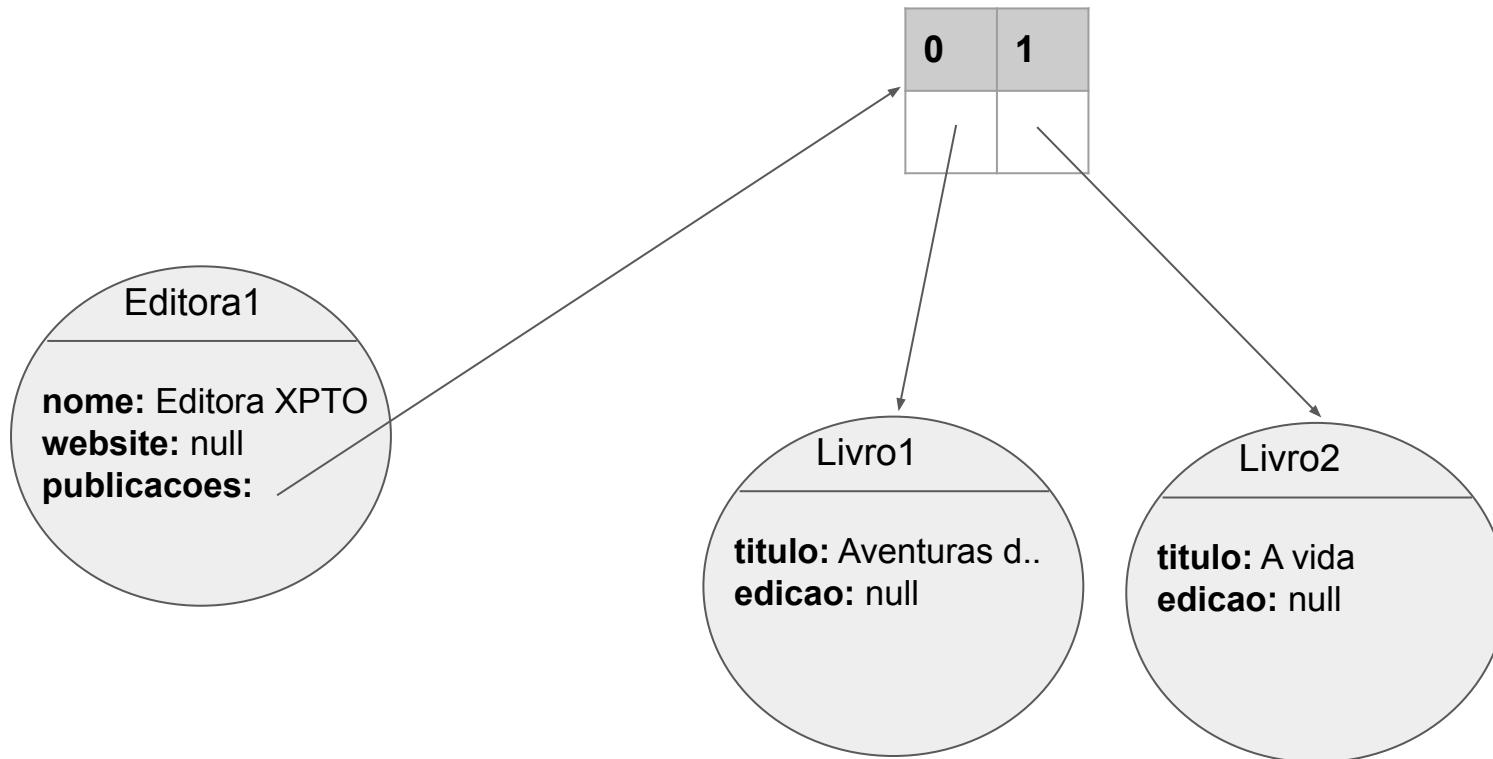
class Editora {
    public $nome;
    public $website;
    public $publicacoes = [];
}
```

```
$livro1 = new Livro();
$livro1->titulo = 'Aventuras do PHP';

$livro2 = new Livro();
$livro2->titulo = 'A vida';

$editora1 = new Editora();
$editora1->nome = 'Editora XPTO';
$editora1->publicacoes[] = $livro1;
$editora1->publicacoes[] = $livro2;
```

Associação (1..*)



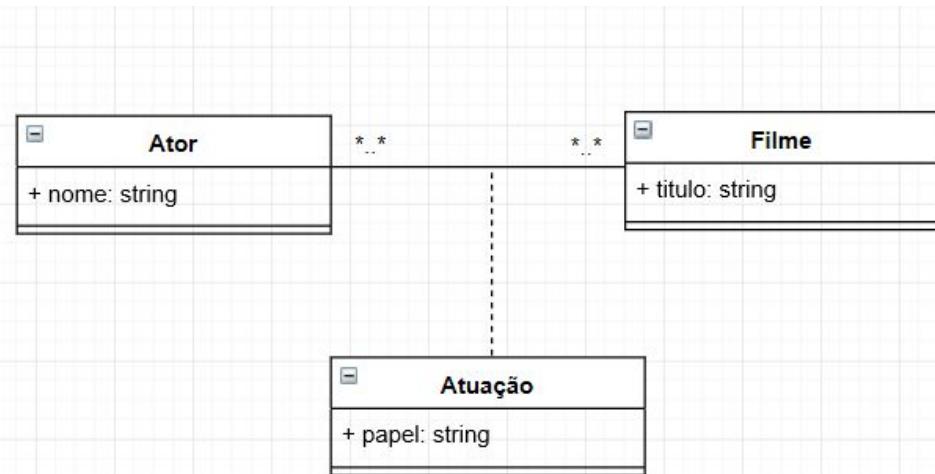
Classe Associativa

Uma classe associativa é uma classe que faz parte de um relacionamento de associação entre duas outras classes.

É possível “anexar” uma classe associativa a um relacionamento para fornecer informações adicionais sobre o relacionamento. Uma classe de associação é idêntica a outras classes e pode conter operações, atributos e outras associações.

Por exemplo, um **Autor** pode atuar em muitos **Filmes** e um **Filme** pode contar com a atuação de vários **Atores**. Em cada **Filme**, o **Autor** pode atuar com um papel distinto. Por isso, neste caso, uma classe associativa chamada **Atuação** pode definir melhor esta associação fornecendo informação adicional do papel que o **Autor** desempenhou em um determinado **Filme**.

Como a figura a seguir ilustra, uma classe de associação é conectada a uma associação por uma linha pontilhada.



Classe Associativa

```
<?php

class Ator {
    public $nome;
    public $atuacoes = [];
}

class Atuacao {
    public $ator;
    public $filme;
    public $papel;

    public function __construct(Ator $ator, Filme $filme, string $papel){
        $this->ator = $ator;
        $this->filme = $filme;
        $this->papel = $papel;
    }
}

class Filme {
    public $titulo;
    public $atuacoes = [];
}
```

```
$ator1 = new Ator();
$ator1->nome = 'José da Silva';

$ator2 = new Ator();
$ator2->nome = 'Maria das Dores';

$filme1 = new Filme();
$filme1->titulo = 'Aventuras do barulho';

$filme1->atuacoes[] =
    new Atuacao($ator1, $filme1, 'Protagonista');
$filme1->atuacoes[] =
    new Atuacao($ator2, $filme1, 'Coadjuvante');
```

Autoload (Carregamento automático de Classes)

- Quando trabalhamos com mais de uma Classe, Interface e/ou Trait, é interessante utilizar algum recurso que possibilite o carregamento dinâmico/automático das estruturas em seus respectivos arquivos *.php.
- O PHP possui duas funções de registro automático: o `__autoload()` e o `spl_autoload_register()`
- O `__autoload()` foi depreciado à partir do PHP 7.2.
- O `spl_autoload_register()` permite que vários carregadores automáticos sejam registrados, que serão executados por sua vez até que uma classe, interface ou trait correspondente seja localizada e carregada, ou até que todas as opções de carregamento automático tenham sido esgotadas. Isso significa que, se você estiver usando algum framework ou outras bibliotecas que implementam seus próprios carregadores automáticos, não é necessário se preocupar com a possibilidade de conflitos.

Autoload (spl_autoload_register)

classes\Pessoa.php

```
<?php

class Pessoa {
    public $nome;
}
```

classes\Veiculo.php

```
<?php

class Veiculo {
    public $placa;
}
```

exemplo.php

```
<?php

function meu_autoloader($class) {
    include 'classes/' . $class . '.php';
}

spl_autoload_register('meu_autoloader');

$pessoa1 = new Pessoa();
$veiculo1 = new Veiculo();
```

Reflection

- Permite introspecção no código Orientado a objeto.
- O Reflection auxilia os desenvolvedores a criar bibliotecas genéricas de software para exibir dados, processar diferentes formatos de dados, realizar serialização, agrupar dados e muito mais.
- O Reflection pode ser usado para observar e modificar a execução do programa em tempo de execução.
- Em linguagens de programação orientada a objeto, como PHP, o Reflection permite a inspeção de classes, interfaces, campos e métodos em tempo de execução sem conhecer os nomes das interfaces, campos, métodos em tempo de execução.
- Também permite a instanciação de novos objetos e a invocação de métodos de forma dinâmica.

Reflection (class)

```
<?php

class Animal {}

class Felino extends Animal {}

class Gato extends Felino {}

$class = new ReflectionClass(Gato::class);

while ($parent = $class->getParentClass()) {
    $parents[] = $parent->getName();
    $class = $parent;
}

echo "Parents: " . implode(", ", $parents);
```

```
<?php

class Aluno {
    public $nome;
    protected $endereco;
    private $telefone;
}

$aluno1 = new Aluno();
$reflect = new ReflectionClass($aluno1);
$props = $reflect->getProperties(ReflectionProperty::IS_PUBLIC | ReflectionProperty::IS_PROTECTED);

foreach ($props as $prop) {
    print $prop->getName() . "\n";
}

var_dump($props);
```

Reflection (alterando visibilidade)

```
<?php
class Teste {
    private $propriedade = 'ola';

    private function dizSegredo(){
        echo $this->propriedade.'! 1234';
    }
}
$class = new ReflectionClass("Teste");
$property = $class->getProperty("propriedade");
$property->setAccessible(true);

$teste1 = new Teste();
//echo $teste1->propriedade; // Não funciona aqui
echo $property->getValue($teste1);
$property->setValue($teste1, 'legal');

$method = new ReflectionMethod("Teste", 'dizSegredo');
$method->setAccessible(true);
echo $method->invoke($teste1);
```

Com o uso do Reflection é possível “modificar” a visibilidade de um atributo ou método em tempo de execução através do `setAccessible(true)`;

Entretanto, o acesso aos valores/retornos precisam ser feitos através dos métodos `getValue()` ou `invoke()`.

No caso dos atributos é possível utilizar o `setValue()` para modificar o valor de um atributo também.

Reflection (gerar instâncias e invocar métodos)

```
<?php

class Pessoa {
    public function digaOi(){
        return 'oi!';
    }
}

//com reflection
$reflector = new ReflectionClass('Pessoa');
$instance = $reflector->newInstance();
$metodo = $reflector->getMethod('digaOi');
 $metodo->invoke($instance);
```

Reflection (recuperar comentários)

```
<?php
/**
 * Esta é uma classe de exemplo
 */
class Exemplo {
    /**
     * Esta é uma função de exemplo
     */
    public function fazNada(){}
}

$reflector = new ReflectionClass(Exemplo::class);

echo $reflector->getDocComment();
echo
$reflector->getMethod('fazNada')->getDocComment();
```

Com Reflection também é possível recuperar comentários das classes e métodos através do getDocComment().

Namespace

- Os namespaces são basicamente uma maneira de organizar suas classes PHP e evitar conflitos de código.
- Usamos a palavra reservada **namespace**
- Namespace não estão relacionados diretamente com a estrutura de diretórios.
- Namespace usam “backslash” (\) como separador
- É possível referenciar a classe usando o **use** para evitar escrever o namespace completo novamente
- Apelidos e Conflitos podem ser resolvidos com o uso do **as**

Namespace

Pessoa.php

```
<?php
```

```
namespace Modelo;
```

```
class Pessoa {
```

exemplo.php

```
<?php
```

```
require "Pessoa.php";
```

```
// não funciona !
```

```
//$pessoa1 = new Pessoa();
```

```
$pessoa1 = new Modelo\Pessoa();
```

Namespace (modelos homônimos)

\Modelo\Literatura\Manga.php

```
<?php  
  
namespace Modelo\Literatura;  
  
class Manga {  
    public $autor;  
    public $editora;  
}  
}
```

\Modelo\Fruta\Manga.php

```
<?php  
  
namespace Modelo\Fruta;  
  
class Manga {  
  
    public $vitaminas = ['A','C'];  
}  
}
```

exemplo.php

```
<?php
```

```
require "Modelo\Fruta\Manga.php";  
require "Modelo\Literatura\Manga.php";
```

```
$manga1 = new Modelo\Literatura\Manga();  
$manga2 = new Modelo\Fruta\Manga();
```

Namespace (**use**)

```
<?php  
  
require "Modelo\Fruta\Manga.php";  
  
use Modelo\Fruta\Manga;  
  
$manga2 = new Manga();
```

Com a palavra reservada *use*, é possível invocar o nome completo de uma classe para que a mesma possa ser utilizada sem a necessidade do namespace após o *use*.

Namespace (**use** + **as**)

```
<?php
```

```
require "Modelo\Fruta\Manga.php";
```

```
use Modelo\Fruta\Manga as FrutaManga;
```

```
$manga2 = new FrutaManga();
```

Namespace (resolução de conflitos)

```
<?php  
  
require "Modelo\Fruta\Manga.php";  
require "Modelo\Literatura\Manga.php";  
  
use Modelo\Fruta\Manga as FrutaManga;  
use Modelo\Literatura\Manga;  
  
$manga1 = new Manga();  
$manga2 = new FrutaManga();
```

Namespace + Autoloader

```
<?php

use classes\Pessoa;
use classes\Veiculo;

function meu_autoloader($class) {
    include_once $class . '.php';
}

spl_autoload_register('meu_autoloader');

$pessoa1 = new Pessoa();
$veiculo1 = new Veiculo();
```

O nome do namespace (e consonância com o nome dos diretórios) pode ser utilizado para incluir automaticamente as classes no código.

Isto não exime o desenvolvedor de utilizar o use ou nome completo da classe (FQN - Fully Qualified Name).

Namespace (Group use PHP >= 7.0)

```
<?php  
  
use classes\{Pessoa, Veiculo};  
  
function meu_autoloader($class) {  
    include_once $class . '.php';  
}  
  
spl_autoload_register('meu_autoloader');  
  
$pessoa1 = new Pessoa();  
$veiculo1 = new Veiculo();
```

A partir do PHP 7, tornou-se possível agrupar classes de um mesmo namespace utilizando a sintaxe de *group use*.

Iteração de atributos do Objeto

```
<?php

class Pessoa {
    public $nome      = 'Maria da Silva';
    private $cpf       = '123456';
    protected $idade = 30;

    function iterar() {
        foreach ($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$pessoa = new Pessoa();
```

//Aqui só é possível iterar os atributos públicos

```
foreach($pessoa as $key => $value) {
    print "$key => $value\n";
}
echo "\n";
```

//no método usando o \$this é possível iterar todos os valores

```
$pessoa->iterar();
```

Interface Iterator

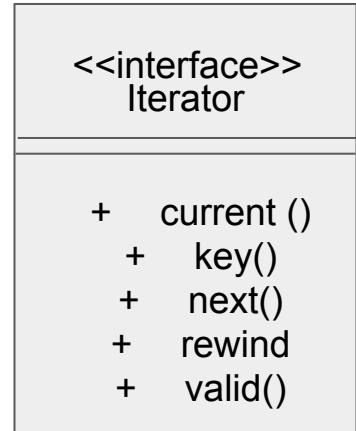
- Interface Iterator possui um conjunto abstrato de métodos para iteradores externos ou objetos que podem ser iterados internamente.
- Quando uma nova classe implementa esta interface, suas instâncias tornam-se iteráveis, similar ao uso de um array.

<<interface>>
Iterator
+ current () + key() + next() + rewind + valid()

Interface Iterator

```
<?php  
  
class Fibonacci implements Iterator {  
    private $anterior = 1;  
    private $atual = 0;  
    private $key = 0;  
  
    public function __construct (int $max = 100) {  
        $this->max = $max;  
    }  
  
    public function current() {  
        return $this->atual;  
    }  
  
    public function key() {  
        return $this->key;  
    }  
}
```

```
public function next() {  
    $novo_atual = $this->atual;  
    $this->atual += $this->anterior;  
    $this->anterior = $novo_atual;  
    $this->key++;  
}  
  
public function rewind() {  
    $this->anterior = 1;  
    $this->atual = 0;  
    $this->key = 0;  
}  
  
public function valid() {  
    if($this->atual > $this->max)  
        return false;  
    return true;  
}
```



Fibonacci

Interface Iterator

```
$seq = new Fibonacci();

foreach ($seq as $f) {

    echo $f.PHP_EOL;

}

}
```

Quando um objeto implementa o Iterator, o mesmo torna-se passível de ser “iterado” de forma transparente, similar a um array, por uma estrutura de laço dinâmica como o “foreach”, graças aos métodos que “garantem” o processo da iteração.

Generator

```
<?php

function meuGenerator() {

    echo "Um".PHP_EOL;
    yield 1;
    echo "Dois".PHP_EOL;
    yield 2;
    echo "Três".PHP_EOL;
    yield 3;
}

$iterator = meuGenerator();

$value = $iterator->current();
$value = $iterator->next();
$value = $iterator->next();
```

Quando uma função contém a palavra reservada `yield`, o PHP automaticamente interpreta a mesma como um objeto do tipo Generator.

Similar ao uso do **Iterator**, um generator permite que o objeto possa ser iterado.

Além disso, o `yield` “pausa” o fluxo natural do código da função e permite retorna um valor similar ao `return`.

Generator (yield from)

```
<?php

function contar10() {
    yield 1;
    yield 2;
    yield from [3, 4];
    yield from new ArrayIterator([5, 6]);
    yield from sete_oito();
    yield 9;
    yield 10;
}

function sete_oito() {
    yield 7;
    yield from oito();
}
```

```
function oito() {
    yield 8;
}

foreach (contar10() as $num) {
    echo "$num ";
}
```

Com o uso ***yield from*** é possível recuperar os valores de outros generators, iteradores ou arrays.

Iterator to Array

```
<?php  
  
$iterator = meuGenerator();  
$seq     = new Fibonacci();  
$array_fibonacci = iterator_to_array($seq);  
$array_generator = iterator_to_array($iterator);  
var_dump($array_fibonacci);  
var_dump($array_generator);
```

Com o uso da função iterator_to_array é possível converter um objeto iterável para array.

Classe DateTime

```
<?php  
  
$dateTime = new DateTime('2016-12-31');  
echo $dateTime->format('Y-m-d H:i:s');  
  
$dateTimeAgora = new DateTime();  
echo $dateTimeAgora->format('Y-m-d H:i:s');
```

Classe DateTime é uma alternativa O.O. à função date().

Ela possui um conjunto de métodos que permitem realizar operações complexas com datas e horários.

Além disso, por sua natureza de Classe, é possível estender a mesma e criar classes customizadas de Data e Hora.

DateTimeZone (Fuso Horário)

```
<?php  
  
date_default_timezone_set('America/Sao_Paulo');  
  
$dateTimeAgora = new DateTime();  
  
echo $dateTimeAgora->format('d/m/Y H:i:s');  
  
  
$dateTime = new DateTime('2016-12-31 12:03:00',  
                      new DateTimeZone('America/Sao_Paulo'));  
  
echo $dateTime->format('d/m/Y H:i:s');
```

É possível setar o fuso horário no PHP através da função `date_default_timezone_set()` passando como valor o fuso horário desejado.

Também é possível determinar o fuso horário individualmente no momento da criação do objeto `DateTime` passando uma instância de `DateTimeZone`

Classe DateTime (createFromFormat)

```
<?php
```

```
$dateBr = DateTime::createFromFormat('d/m/Y', '31/12/2003');  
echo $dateBr->format('d/m/Y');
```

Formato para
exibição
(string)

Formato
da data

Data no
formato

Classe DateTime (modificando data)

Adicionar um dia

```
<?php  
  
$dateTime = new DateTime('2018-01-31');  
$dateTime->modify('+1 day');  
echo $dateTime->format('d/m/Y');
```

Adicionar dez dias úteis

```
<?php  
  
$dateTime = new DateTime();  
$dateTime->modify('+10 weekdays');  
echo $dateTime->format('d/m/Y');
```

Segunda Feira desta semana

```
<?php  
  
$dateTime = new DateTime();  
$dateTime->modify('monday this week');  
echo $dateTime->format('d/m/Y');
```

Adicionar um mês e “menos três dias”

```
<?php  
  
$dateTime = new DateTime();  
$dateTime->modify('+1 month -3 days');  
echo $dateTime->format('d/m/Y');
```

Classe DateTime (diferença entre datas)

```
<?php  
  
$date1 = new DateTime('2018-08-11 05:02:00');  
$date2 = new DateTime('2018-10-13 10:12:34');  
$interval = $date1->diff($date2);  
  
echo "Meses: {$interval->m} dias: {$interval->d} horas: {$interval->h} minutos:  
{$interval->i} segundos: {$interval->s}";
```

DateTime (comparação)

```
<?php  
  
$hoje = new DateTime('today');  
$ontem = new DateTime('yesterday');  
var_dump($hoje > $ontem);  
var_dump($hoje < $ontem);  
var_dump($hoje == $ontem);
```

É possível usar operadores de comparação para comparar datas.

Typed Properties (PHP 7.4 >=)

```
<?php

class Veiculo {
    public string $placa;
    public string $modelo;
}

class Pessoa {
    public string $nome;
    public Veiculo $veiculo;
}

$pessoa1 = new Pessoa();
$pessoa1->nome = "Maria da Silva";
$veiculo1 = new Veiculo();
$veiculo1->placa = "ABC 123";
$pessoa1->veiculo = 1; //erro !
$pessoa1->veiculo = $veiculo;
```

Com o advento das typed properties no PHP 7.4, é possível especificar os tipos dos atributos de uma classe e restringir os valores dos mesmos.

O comportamento é similar ao type hint em parâmetros de métodos (funções) e ambos são afetados pelo nível do “strict_types” onde, o PHP poderá (ou não) converter determinados tipos dos valores em tempo de execução para o tipo determinado na declaração na classe.

StdClass e Type Cast

```
<?php  
//convertendo array para objeto  
$array = ['nome'=>'José', 'idade'=>34];  
$objeto = (object) $array;  
var_dump($objeto);  
  
//convertendo json para objeto  
$json = '{"nome":"Maria", "idade":45}';  
$objeto2 = json_decode($json);  
var_dump($objeto2);  
  
//criando um objeto genérico  
$objeto3 = new StdClass();  
$objeto3->nome = 'Legal';  
$objeto3->vivo = true;  
var_dump($objeto3);
```

A classe nativa do PHP StdClass (Standard Class) tem o propósito de ser uma classe para objetos frutos de coerções.

Também permite que objetos genéricos possam ser criados.

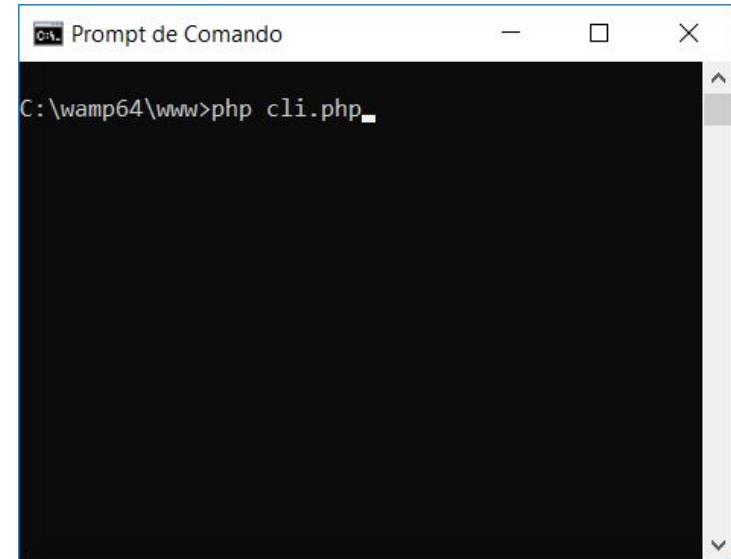
Entrada de dados

- Para que um aplicação seja dinâmica, faz-se necessário permitir que os usuários possam interagir com a mesma através de algum método de input (entrada) de dados.
- Antes do surgimento dos sistemas operacionais gráficos (GUI), a entrada de dados era feita através de interface de linha de comando (CLI).
- Com o surgimento da web, os dados são, em sua maioria, imputados através de formulários HTML.

Entrada de dados (CLI)

cli.php

```
<?php  
  
echo "Qual é o seu nome? " . PHP_EOL;  
$input = fgets(STDIN);  
echo "Seja bem vindo(a), $input";
```



Entrada de Dados (web)

formulario.htm

```
<html>
<form action='cadastrar.php'>
    <label>Nome:</label>
    <input type="text" name="nome"/>
    <label>Email:</label>
    <input type="email" name="email"/>
    <input type="submit" value="Cadastrar"/>
</form>
</html>
```

cadastrar.php

```
<?php
var_dump($_REQUEST);
```

Formulários e Requisições (Variáveis)

- **`$_POST`**: é um array de variáveis com os valores enviados via HTTP POST como, por exemplo, o utilizado por formulários HTML.
- **`$_GET`**: é um array de variáveis com os parâmetros da URL, também conhecido como “query string”.
- **`$_REQUEST`**: por default possui o conteúdo de `$_GET`, `$_POST` (e `$_COOKIE` também).
- **`$_FILES`**: um array com as informações de arquivos enviados via HTTP POST (utilizar o “multipart/form-data” no enctype do formulário)

Entrada de Dados (GET)

http://localhost/curso/get/index.php?nome=Maria&idade=33

URL

campo

valor

atribuição
de valor

separador de
campos

Separador de URL e Query String

Formulário (Array como parâmetro)

sanduba_formulario.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Formulário</title><meta charset="utf-8" />
</head>
<body>
<form action="sanduba.php" method="post">
    Quais adicionais você deseja?<br />
    <input type="checkbox" name="adicionais[]" value="queijo" />Queijo<br />
    <input type="checkbox" name="adicionais[]" value="bacon" />Bacon<br />
    <input type="checkbox" name="adicionais[]" value="ovo" />Ovo<br />
    <input type="submit"/>
</form>
</body>
</html>
```

sanduba.php

```
<?php
foreach($_REQUEST['adicionais'] as $adicional){
    echo $adicional."<br/>";
}
```

Formulários (upload - `$_FILES` e multipart/form-data)

```
<!DOCTYPE html>
<html>
<head>
<title>Formulário com upload</title>
<meta charset="utf-8" />
</head>
<body>
<form action="email_com_anexo.php"
      method="post" enctype="multipart/form-data">
    Name: <input type="text" name="nome"><br>
    E-mail: <input type="email" name="email"><br>
    Anexo: <input type="file" name="anexo"><br>
    <input type="submit"/>
</form>
</body>
</html>
```

email_com_anexo.php

```
<?php
$diretorio = "tmp".DIRECTORY_SEPARATOR;

foreach($_FILES as $arquivo){
    $nome    = $arquivo['name'];
    $conteudo = file_get_contents($arquivo['tmp_name']);
    //salvando no disco
    file_put_contents($diretorio.$nome, $conteudo);
}
```

Cookie

```
<!DOCTYPE html>
<?php
$cookie_name = "usuario";
$cookie_value = "João da Silva";
setcookie($cookie_name, $cookie_value, mtime(24), "/");
?>
<html>
<body>
<?php
if (!isset($_COOKIE[$cookie_name])) {
    echo "Cookie (" . $cookie_name . ") sem valor";
} else {
    echo "Cookie " . $cookie_name . " com valor!<br>";
    echo "valor atual é: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Um cookie é um pequeno arquivo que o servidor incorpora no computador (navegador) do usuário. Cada vez que o mesmo computador solicita uma página com um navegador, ele também enviará o cookie.

Com o PHP, é possível criar e recuperar valores de cookie. O nome do cookie é automaticamente atribuído a uma variável com o mesmo nome.

A função `setcookie()` cria o cookie enquanto a variável global `$_COOKIE` acessa os valores.

O cookie pode ser substituído por soluções mais modernas como o `localStorage` do Javascript.

Session

```
<?php session_start(); ?>
<!DOCTYPE html>
<html>
  <body>
    <?php
      $_SESSION["usuario"] = "admin";
      $_SESSION["senha"] = "1234";
    ?>
  </body>
</html>
```

```
<?php
session_start();

if(isset($_SESSION['usuario'])){
  echo "Seja bem vindo ! {$_SESSION['usuario']} !";
} else{
  header("Location: index.php");
}
```

Session (sessão) é uma maneira de armazenar informações temporariamente que podem ser utilizadas em várias páginas da aplicação;

Ao contrário de um cookie, as informações não são armazenadas no computador do usuário e sim no servidor.

Apesar da informação (dado) estar no servidor, um pequeno cookie é criado no navegador do usuário para vincular a este dado(s) na sessão.

É possível destruir a sessão utilizando a função: **session_destroy()**;

PHP Assíncrono (PHP + Javascript)

- Com o fluxo tradicional “síncrono” do HTTP, faz-se necessário “atualizar/carregar” a página toda vez que um informação é enviada do cliente para o servidor (ou o inverso), o que, em muitas situações, pode prejudicar a interação do usuário com a aplicação.
- Com o advento do Ajax em 1999, os apps web tornaram-se capaz de enviar e recuperar dados do servidor de forma assíncrona (em segundo plano) sem interferir na exibição e no comportamento da página existente. Ao separar a camada de troca de dados da camada de apresentação, o Ajax permite que uma página web possa alterar seu conteúdo dinamicamente sem a necessidade de recarregar a página inteira.
- A partir de 2017, uma nova alternativa surgiu ao Ajax: o fetch, que permite que requisições assíncronas possam ser feitas usando o conceito de *promises*.

PHP Assíncrono (Ajax - XMLHttpRequest)

index.php

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>Teste de Ajax</title>
    <script src="curso.js"></script>
  </head>
  <body>
    <div id="mensagem"></div>
    <button onclick="meuAjax()">Ver mensagem</button>
  </body>
</html>
```

meu_ajax.php

```
<?php

echo "Meu nome é ".$_REQUEST['nome'];
echo " ".(new DateTime())->format('h:i:s');
```

Com o uso do XMLHttpRequest do Javascript é possível recuperar enviar dados e receber conteúdos de documentos web de forma assíncrona.

curso.js

```
function meuAjax() {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', 'meu_ajax.php?nome=Maria');
  xhr.onload = function () {
    if (xhr.status === 200) {
      document.getElementById('mensagem')
        .innerHTML = xhr.responseText;
    } else {
      alert('Erro! Status: ' + xhr.status);
    }
  };
  xhr.send();
}
```

PHP Assíncrono (Fetch JS)

index.php

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>Teste de Ajax</title>
    <script src="curso.js"></script>
  </head>
  <body>
    <div id="mensagem"></div>
    <button onclick="meuFetch()">Ver mensagem</button>
  </body>
</html>
```

meu_fetch.php

```
<?php

echo "Meu nome é ".$_REQUEST['nome'];
echo " ".(new DateTime())->format('h:i:s');
```

A partir de 2017, o fetch permite que requisições assíncronas possam ser feitas usando o conceito de promises.

curso.js

```
function meuFetch() {
  window.fetch("meu_fetch.php?nome=Maria")
    .then(response => response.text())
    .then(data => {
      document.getElementById('mensagem')
        .innerHTML = data;
    })
    .catch(error => alert('Erro!' + error));
}
```

PHP Assíncrono (FormData + Fetch JS)

index.php

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>Formulário Assíncrono</title>
    <script src="curso.js"></script>
  </head>
  <body>
    <form id="meu_form" action="meu_form_data.php">
      <label>Nome:</label>
      <input type="text" name="nome"/>
      <label>Email:</label>
      <input type="email" name="email"/>
      <input type="submit" value="Cadastrar"/>
    </form>
    <div id="mensagem"></div>
    <script>
      const form = document.getElementById('meu_form');
      form.addEventListener('submit', meuFormData);
    </script>
  </body>
</html>
```

curso.js

```
function meuFormData(event) {
  event.preventDefault();
  const formData = new FormData(this);
  window.fetch(this.getAttribute("action"), {
    method: 'post',
    body: formData
  }).then(function (response) {
    return response.text();
  }).then(function (text) {
    document.getElementById('mensagem')
      .innerHTML = text;
  });
}
```

meu_form_data.php

```
<?php
echo "Email {$_REQUEST['email']} cadastrado com
sucesso !";
```

PHP Assíncrono (Upload Múltiplo com Fetch)

index.php

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head> <title>Formulário Assíncrono</title>
    <script src="curso.js"></script> </head>
  <body>
    <form id="meu_form" action="meu_form_data.php"
      enctype="multipart/form-data">
      <label>Nome:</label>
      <input type="text" name="nome"/>
      <label>Email:</label>
      <input type="email" name="email"/>
      <input type="file" name="anexo" multiple="multiple"/>
      <input type="submit" value="Cadastrar"/>
    </form>
    <div id="mensagem"></div>
    <script>
      const form = document.getElementById('meu_form');
      form.addEventListener('submit', meuFormData);
    </script>
  </body>
</html>
```

curso.js

```
function meuFormData(event) {
  event.preventDefault();
  const formData = new FormData(this);

  var fileInput = document.querySelector('input[type="file"]');
  formData.delete('anexo');
  for (var i=0; i < fileInput.files.length; i++){
    formData.append('anexo'+i, fileInput.files.item(i));
  }

  window.fetch(this.getAttribute("action"), {
    method: 'post',
    body: formData
  }).then(function (response) {
    return response.text();
  }).then(function (text) {
    document.getElementById('mensagem')
      .innerHTML = text;
  });
}
```

PHP Assíncrono (Upload Múltiplo com Fetch)

meu_form_data.php

```
<?php  
  
echo "Email {$_REQUEST['email']} cadastrado com sucesso ! <br/>";  
  
$diretorio = "anexos".DIRECTORY_SEPARATOR;  
  
foreach($_FILES as $arquivo){  
    $nome    = $arquivo['name'];  
    $conteudo = file_get_contents($arquivo['tmp_name']);  
    file_put_contents($diretorio.$nome, $conteudo);  
    if(file_exists($diretorio.$nome))  
        echo "Arquivo {$arquivo['name']} salvo com sucesso ! <br/>";  
}  
}
```

PHP Assíncrono (JSON + <select>)

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>Formulário Assíncrono</title> <script src="curso.js"></script>
  </head>
  <body>
    <form>
      <label>Região:</label>
      <select id="regioes">
        <option value="">Selecione...</option>
        <option value="centro-oeste">Centro Oeste</option>
        <option value="sul">Sul</option>
      </select>
      <label>Estado:</label>
      <select id="estados"></select>
    </form>
    <script>
      const select = document.getElementById('regioes');
      select.addEventListener('change', selectEstados.bind(this,'regioes', 'estados'), false);
    </script>
  </body>
</html>
```

index.php

PHP Assíncrono (JSON + <select>)

```
function selectEstados(fonte_id, alvo_id) {  
  
    fonte = document.getElementById(fonte_id);  
    alvo = document.getElementById(alvo_id);  
    alvo.length = 0;  
  
    let regiao_selecionada = fonte.options[fonte.selectedIndex].value;  
    if (regiao_selecionada == "")  
        return;  
    window.fetch("estados.php?regiao_selecionada=" + regiao_selecionada)  
        .then(response => response.json())  
        .then(data => {  
            for (var i = 0; i < data.length; i++) {  
                var option = document.createElement("option");  
                option.innerHTML = data[i].nome;  
                option.value = data[i].id;  
                alvo.options.add(option);  
            }  
        })  
        .catch(error => alert('Erro!' + error));  
}
```

curso.js

PHP Assíncrono (JSON + <select>)

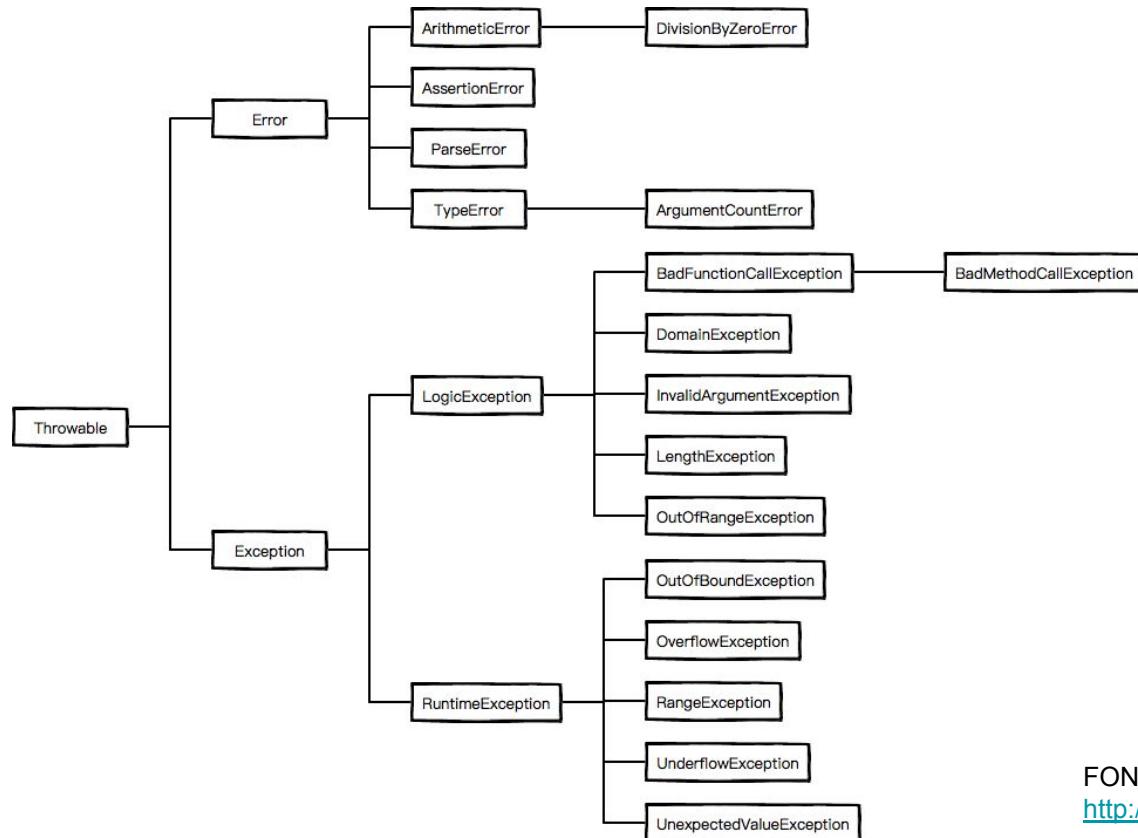
```
<?php  
  
header('Content-Type: application/json');  
  
$regiao_selecionada = $_REQUEST['regiao_selecionada'];  
$estados = [  
    'centro-oeste' => [  
        ['id'=>'DF', 'nome' => 'Distrito Federal'],  
        ['id'=>'GO', 'nome' => 'Goiás'],  
        ['id'=>'MT', 'nome' => 'Mato Grosso'],  
        ['id'=>'MS', 'nome' => 'Mato Grosso do Sul']  
    ],  
    'sul' => [  
        ['id'=>'PR', 'nome'=> 'Paraná'],  
        ['id'=>'RS', 'nome'=> 'Rio Grande do Sul'],  
        ['id'=>'SC', 'nome'=> 'Santa Catarina']  
    ]  
];  
  
echo json_encode($estados[$regiao_selecionada]);
```

estados.php

Tratamento de Exceções/Erros

- O tratamento de exceção é usada para alterar o fluxo normal da execução de código se ocorrer uma condição de erro específico (excepcional). Essa condição é chamada de exceção.
- No PHP 7>= existem dois principais tipos de erros: os throwables e o não throwables.
- Os throwables podem ser lançados via `throw` e tratados via `try/catch`. Nesta categoria existem dois sub tipos: `Exceptions` e `Errors`.
- Os não throwables podem, em alguns casos, ser suprimidos via `@`, `error_reporting` ou tratados utilizando o `set_error_handler()`

Taxonomia de Erros e Exceções (throwables)

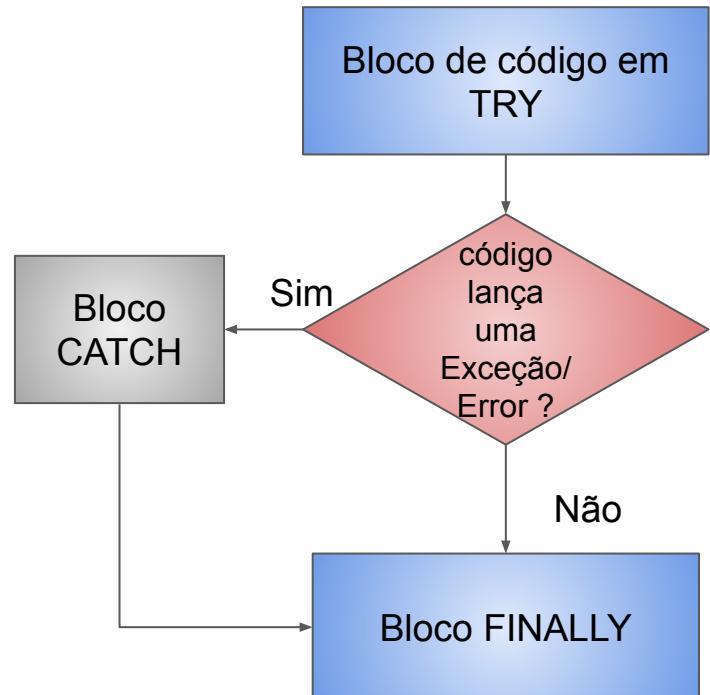


Com o advento do PHP 7, foi definido uma Interface chamada `Throwable`, que se tornou a interface base para qualquer objeto “lançável”, isto é, objetos passíveis de serem disparados via `throw` e/ou capturados via `catch`.

FONTE:
<http://asika.windspeaker.co/post/3503-php-exceptions>

Manipulação de Throwables

```
<?php  
  
try {  
    // código aqui  
}  
  
catch (\Throwable $t) {  
    echo $t->getMessage();  
}
```



Throwable

```
<?php
// "capturando" exceptions
try {
    throw new Exception('exception');
} catch (Exception $e) {
    echo('capturando exception: '. $e->getMessage()).PHP_EOL;
}
// capturando errors (PHP 7>=)
try {
    $obj = new \StdClass();
    $obj->not_a_method();
} catch (Error $e) {
    echo('capturando error: '. $e->getMessage()).PHP_EOL;
}
// ambos (Exception e Error)
try {
    throw new Exception('exception');
} catch (Throwable $e) {
    echo('capturando Throwable: '. $e->getMessage()).PHP_EOL;
}
```

Utilizando a Interface base como typehint do catch, é possível capturar tanto as Exceptions como os Errors.

Tratamento de Throwables (Finally [PHP 5.5>=])

```
<?php  
try {  
    print "bloco de código";  
  
} catch (\Throwable $t) {  
    print "Erro!";  
  
} finally {  
    print "Terminou!";  
}
```

Com o advento do PHP 5.5, tornou-se possível colocar uma condição final que será executada tanto no caso da ocorrência de uma Exception/Error quanto no caso de sucesso das instruções do bloco try.

Lançamento de Throwables (throw new)

```
<?php

function escolhaMcOferta (int $opcao) : ?string {

    $ofertas = ['bic mac', 'mc cheddar',
                'quarteirão', 'mc fish', 'mc chicken'];

    if (!in_array($opcao, range(1,5))) {
        throw new OutOfRangeException("Oferta inválida");
    }
    return $ofertas[--$opcao];
}

echo escolhaMcOferta(6);
```

Com o uso do operador `throw` seguido de uma instância de `Throwable` é possível “lançar” uma exceção/erro para que o usuário de uma função/método possa receber as informações necessárias do erro e tratá-lo da melhor maneira possível;

Se o desenvolvedor tentar usar uma instância de um objeto que não implementa `throwable` ocorrerá um erro fatal do tipo “`Uncaught Error`” descrevendo que o objeto não implementa a interface;

Tratamento de Throwables (múltiplos)

```
<?php

function teste($x):int{
    return $x;
}

try {
    intdiv(10,0);
    teste('ss');
}
catch(DivisionByZeroError $e){
    echo "não pode fazer essa divisão";
}
catch(TypeError $e){
    echo "Erro de tipo!";
}
```

É possível especificar diferentes tratamentos de Exceptions/Errors através do aninhamento de catchs

Tratamento de Throwables (grupo com pipe)

```
<?php

function teste($x):int{
    return $x;
}

try {
    intdiv(10,0);
    teste('ss');
}

catch(DivisionByZeroError | TypeError $e){
    echo "Erro de divisão ou de tipo";
}
```

Com o advento do PHP 7.1 é possível especificar um único tratamento para um grupo de diferentes tratamentos de Exceptions/Errors utilizando o operador pipe “|”;

Criando Exceções/Erros customizados (extends Exception)

```
<?php
class MeuException extends Exception {
    public function __construct($message = null, $code = 0){
        parent::__construct($message, $code);
        file_put_contents('/tmp/log.txt',
            $this->getTraceAsString().PHP_EOL ,
            FILE_APPEND | LOCK_EX);
    }
}

function legal($x){
    if($x == 0){
        throw new MeuException();
    }
}

try{
    legal(0);
}catch(Exception $e){
    print('erro');
}
```

É possível criar um Throwable customizado com uma especialização de Exception ou Error. (Pois não é possível implementar a interface Throwable).

Principais Tipos de Erros (não Throwables)

Tipo	Constante	Significado	Interrompe a execução do script?	Pode ser suprimido com @?	Tempo de:
NOTICE	E_NOTICE	Aviso para indica que o script encontrou alguma coisa que pode indicar um erro;	NÃO	SIM	Execução
WARNING	E_WARNING	Erro não fatal;	NÃO	SIM	Execução
DEPRECATED	E_DEPRECATED	Aviso de um recurso depreciado e será futuramente removido nas próximas versões do PHP.	NÃO	SIM	Execução
ERROR	E_ERROR	Erro fatal em tempo de execução. Estes indicam erros que não podem ser recuperados.	SIM	NÃO	Execução
PARSER ERROR	E_PARSER	Erros gerados pelo interpretador devido a erro de sintaxe no script	SIM	NÃO	Compilação

Operador de controle de erro (supressão @)

```
<?php
echo @(10 / 0);
// suprimiu "Warning: Division by zero"
$c = @$_POST["nome"] . @$_POST["sobrenome"];
// suprimiu "Notice: Undefined index: nome"
// suprimiu "Notice: Undefined index: sobrenome"
@$newfunc = create_function('$a', 'return;');
// suprimiu "Deprecated: Function create_function() is
deprecated"
@ $i / 0;
// suprimiu "Notice: Undefined variable: i"
// não suprimiu o "Warning: Division by zero"

$c = @funcaoNaoExiste(); //não suprimiu erro fatal
echo 'fim';
```

O PHP suporta um operador de controle de erro: o sinal 'arroba' (@).

Com uso do @ é possível suprimir um notice ou um erro não fatal.

Controle de nível de erros não throwables

```
<?php  
  
// Desligando todos os avisos de erros  
error_reporting(0);  
  
// Ligando apenas para warning  
error_reporting(E_WARNING);  
  
// com o uso do pipe | é possível criar uma lista  
// fixa de avisos para erros  
error_reporting(E_ERROR | E_WARNING | E_PARSE);  
  
// E_ALL é o equivalente a todos os tipos  
error_reporting(E_ALL);  
  
// Com o uso do ^ (not) é possível remover  
// um item da lista (todos menos notice)  
error_reporting(E_ALL ^ E_NOTICE);
```

Com o uso da função `error_reporting()` podemos controlar, em tempo de execução quais avisos de erros o PHP poderá imprimir na tela.

Essas mesmas constantes podem ser setadas de uma maneira global no `php.ini`.

set_error_handler() e restore_error_handler()

```
<?php
set_error_handler("manipuladorCustomizadoDeErros");

function manipuladorCustomizadoDeErros
($severity, $mensagem, $arquivo, $linha) {
    if (error_reporting() & $severity) {
        throw new Exception($mensagem, 0);
    }
}

$array = ['maria','jósé'];

try {
    $b = $array[2];
} catch (Exception $e) {
    echo "Posição não encontrada !";
} finally {
    restore_error_handler();
}
```

Com o uso do set_error_handler() é possível criar um manipulador customizado de erros não throwables.

É possível, inclusive, lançar uma exceção dentro de um manipulador customizado permitindo assim que erro possa ser tratado com um throwable, podendo ser capturado via catch.

O ideal é sempre no final do bloco de instruções restaurar o manipulador original do programa com o restore_error_handler().

Serialização

- É o processo de traduzir estruturas de dados ou estado de objeto em um formato que pode ser armazenado (por exemplo, em um arquivo ou buffer de memória) ou transmitido e reconstruído posteriormente (possivelmente em um ambiente de computador diferente).
- Quando a série resultante de bits é relida de acordo com o formato de serialização, ela pode ser usada para criar um clone semanticamente idêntico do objeto original.
- Esse processo de serializar um objeto também é chamado de ***marshalling*** (empacotamento de um objeto).
- A operação oposta, extraíndo uma estrutura de dados de uma série de bytes, é a desserialização (também chamada de *unmarshalling*).

Serialização (serialization)

Aluno.php

```
<?php

class Aluno {

    public $nome;
    public $matricula;

    public function __construct(string
$nome, int $matricula) {
        $this->nome = $nome;
        $this->matricula = $matricula;
    }
}
```

Turma.php

```
<?php

class Turma {

    public $nome;
    public $data;
    public $alunos;

    public function __construct(string
$nome, \DateTime $data, array $alunos = []) {
        $this->nome = $nome;
        $this->data = $data;
        $this->alunos = $alunos;
    }
}
```

Serialização (escrita)

```
<?php
include_once 'Turma.php';
include_once 'Aluno.php';

$turmas = [];
$aluno1 = new Aluno('José', 123);
$aluno2 = new Aluno('Maria', 456);
$aluno3 = new Aluno('Thiago', 789);

$turmas[] = new Turma('PHP',
    new \DateTime('today'), [$aluno1, $aluno2]);
$turmas[] = new Turma('CakePHP',
    new \DateTime('-2 days'), [$aluno1, $aluno2, $aluno3]);
$turmas[] = new Turma('MySQL',
    new \DateTime('yesterday'), [$aluno1, $aluno3]);

$serializacao = serialize($turmas);
file_put_contents('dados.db', $serializacao);
```

A função **serialize** converte toda estrutura de arrays e objetos em uma string utilizando um formato passível de recuperação pelo próprio PHP.

Já a função **file_put_contents** persiste uma string em um arquivo texto.

Se as classes não forem carregadas o PHP irá gerar instâncias de **__PHP_Incomplete_Class**.

Desserialização (leitura)

```
<?php
include 'Turma.php';
include 'Aluno.php';

$serializacao = file_get_contents('dados.db');
$turmas = unserialize($serializacao);

echo "<table border>";
foreach ($turmas as $turma) {
    echo "<tr>";
    echo "<td> {$turma->nome} </td>";
    echo "<td> {$turma->data->format('d/m/Y')} </td>";
    echo "<td>". implode(", ",
        array_column($turma->alunos, 'nome')). "</td>";
    echo "</tr>";
}
echo "</table>";
```

A função **file_get_contents** recupera o conteúdo de um arquivo texto em uma string.

A função **unserialize** interpreta o conteúdo serializado em uma string e reconstrói os objetos e arrays.

PHP e Bancos de Dados Relacionais

- Até o PHP 5.0, era necessário utilizar extensões PECL com conjuntos diferentes de funções para cada acessar cada banco de dados (ex.: `mysql_connect`, `mssql_connect` etc...).
- Com o advento do PHP 5.1 surgiu o PDO (*PHP Data Objects*): uma camada de abstração de acesso a banco de dados (DBAL). Uma DBAL (Database abstraction layer) é uma API que visa unificar a comunicação entre um aplicações e bancos de dados.
- As DBALs reduzem a quantidade de trabalho para acessar diferentes bancos de dados fornecendo uma API consistente ao desenvolvedor ocultando o máximo possível as especificidades do banco de dados por trás dessa interface.
- O PDO fornece uma DBAL, o que significa que, independentemente de qual banco de dados, o desenvolvedor utiliza as mesmas funções para emitir consultas e buscar dados. O PDO não fornece uma abstração de SQL e nem emula os recursos ausentes de um banco de dados.

Modelo Relacional

- O modelo relacional é uma abordagem para gerenciar dados usando uma estrutura e linguagem consistente com lógica de predicados de primeira ordem, descrita pela primeira vez em 1969 pelo cientista inglês Edgar F. Codd, onde os dados são representados em termos de tuplas (linhas), agrupadas em relações (tabelas).
- Um banco de dados organizado em termos do modelo relacional é um banco de dados relacional.
- A maioria dos bancos de dados relacionais utiliza o SQL para definição de dados e a linguagem de consulta; esses sistemas implementam o que pode ser considerado como uma aproximação de engenharia ao modelo relacional.
- Além disso, os banco de dados permitem também criar relacionamento entre as tabelas (entidades) através de chaves estrangeiras.

Modelo Relacional: Tabela, colunas, tipos e chave primária

- Uma tabela é composta de colunas (campos).
- Cada coluna possui um nome e um tipo de dado, ex.: VARCHAR (string), INT, DOUBLE etc.
- É possível definir também se um campo poderá permitir valores nulos ou não.
- Para garantir o unicidade de cada tupla (registro) é possível utilizar o recurso da **chave primária** onde o SGBD garante que aquele valor não possa ser repetido

Relacionamentos e restrições de integridade

- Um relacionamento, no contexto de bancos de dados relacionais, é uma situação que existe entre duas tabelas quando uma possui uma **chave estrangeira** que faz referência à chave primária da outra tabela. Os relacionamentos permitem que bancos de dados relacionais dividam e armazenem dados em diferentes tabelas, enquanto ligam itens de dados distintos.
- Existem os seguintes tipos de relacionamentos entre tabelas:
 - **1..1 (um para um)**: onde uma das duas tabelas faz referência para a outra.
 - **1..N (um para muitos)**: onde o lado N recebe faz referência para o lado um.
 - **N..N (muitos para muitos)**: onde uma terceira tabela (associativa) precisa ser criada para permitir que está faça referência para as duas tabelas da relação.
 - **Auto relacionamento**: onde uma tabela faz referência para ela mesma.

Drive PDO

- Um drive/extensão PDO define uma interface leve e consistente para acessar bancos de dados no PHP. Cada driver de banco de dados que implementa a interface do PDO pode expor recursos específicos do banco de dados como funções de extensão regulares.
- Os drivers PDO também são extensões PECL e podem ser habilitadas no php.ini, caso as mesmas estejam disponíveis em forma de *.dll ou *.so no diretório de extensões do PHP.

PDO (getAvailableDrivers())

```
<?php  
  
print_r(PDO::getAvailableDrivers());
```

O método estático `getAvailableDrivers` permite verificar quais os drivers de PDO estão instalados no PHP.

PDO Drivers

Banco (SGBD)	Drive (.dll ou .so)	Incluído no pacote PHP Windows?	Download adicional
MySQL 	php_pdo_mysql	SIM	-
Postgres 	php_pdo_pgsql	SIM	-
SQL Server 	php_pdo_sqlsrv	<u>NÃO</u>	Drive ODBC
Oracle 	php_pdo_oci	SIM	Instant Client

Classe PDO (instanciando)

```
<?php
```

```
$pdo = new PDO($dsn, $usuario, $senha, $opcoes);
```

- **dsn**: conexão com a fonte de banco de dados
- **usuario**: usuário do banco de dados
- **senha**: senha deste usuário
- **opções**: conjunto de opções em forma de chave e valores utilizando um conjunto de constantes, alguns comuns entre os drivers e outros específicos de cada um.

PDO Options (opções recomendadas)

Opção	Valor Recomendado	Resultado	Suportado pelos Drivers
PDO::ATTR_ERRMODE	PDO::ERRMODE_EXCEPTION	Lança Exceptions toda vez que uma instrução SQL falhar.	MySQL, Postgres, Sql Server e Oracle
PDO::ATTR_EMULATE_PREPARES	false	O motor do banco de dados fará o <i>prepared statement</i> ao em vez do PDO e assim, consulta e os dados reais são enviados separadamente, aumentando a segurança.	MySQL, Postgres e Oracle
PDO::ATTR_DEFAULT_FETCH_MODE	PDO :: FETCH_ASSOC	É conveniente configurá-lo de forma global e depois omiti-lo em buscas específicas.	MySQL, Postgres, Sql Server e Oracle

DSN (Data Source Name)

- DSN ou data source name (nome de fonte de dados, algumas vezes conhecido como nome de fonte de banco de dados, apesar de fontes de dados não serem limitadas a bancos de dados), é uma estrutura de dados usada para descrever uma conexão a uma fonte de dados.
- Cada Driver possui seu conjunto específico de parâmetros. Exemplos:
 - **MYSQL:** mysql:host=localhost;dbname=livraria;port=3306;charset=utf8mb4
 - **POSTGRES:** pgsql:host=localhost;port=5432;dbname=livraria;
 - **SQL SERVER:** sqlsrv:Server=localhost;Database=livraria
 - **ORACLE:**
`oci:dbname=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT _DATA=(SERVICE_NAME=LIVRARIA)))`

- Nome ou IP da máquina onde o Banco (SGBD) está executando.
- Nome da base onde as tabelas se encontram.
- Porta de conexão do SGBD

PDO (Conexão MySQL)

conexao.php

```
<?php
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false, //para funcionar bind no limit
];
$servidor = "localhost";
$banco    = "livraria";
$usuario  = "root";
$senha    = "";
$porta    = 3306;
$dsn      = "mysql:host=$servidor;port=$porta;dbname=$banco;charset=utf8";

$pdo = new PDO($dsn, $usuario, $senha, $options);
```

PDO (Conexão Postgres)

conexao_postgres.php

```
<?php

$options = [
    PDO::ATTR_ERRMODE           => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];

$servidor = "localhost";
$banco    = "livraria";
$usuario  = "postgres";
$senha    = "admin";
$porta    = 5432;
$dsn      = "pgsql:host=$servidor;port=$porta;dbname=$banco;";

$pdo = new PDO($dsn, $usuario, $senha, $options);
```

PDO (Conexão SQL Server)

conexao_sqlserver.php

```
<?php

$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION, //ver erros de query
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
];

$servidor = "localhost";
$banco    = "livraria";
$usuario  = "85bits";
$senha    = "admin";
$dsn      = "sqlsrv:Server=$servidor;Database=$banco";

$pdo = new PDO($dsn, $usuario, $senha, $options);
```

PDO (Conexão Oracle)

conexao_oracle.php

```
<?php
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_EMULATE_PREPARES   => false,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_CASE                => PDO::CASE_LOWER
];

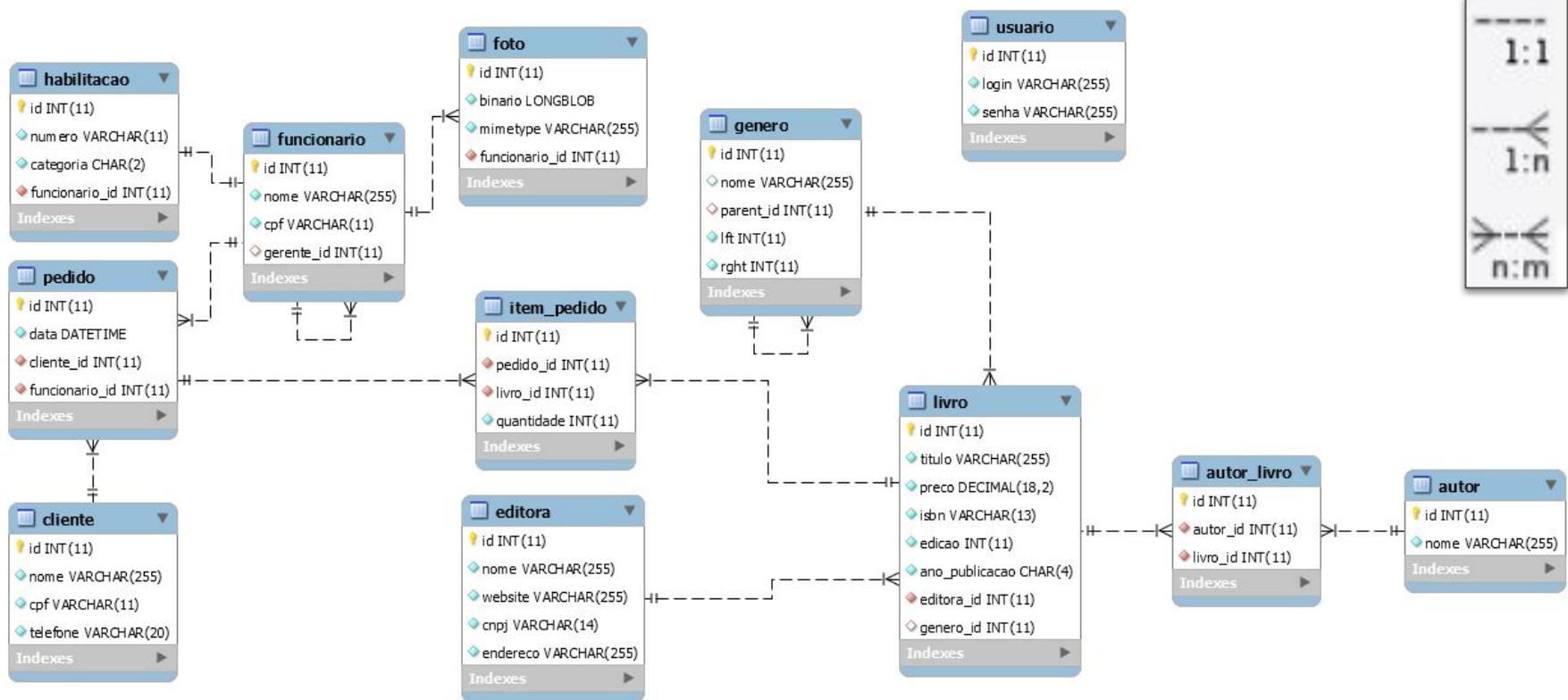
$servidor = "localhost";
$usuario = "php";
$senha  = "admin";
$service_name = "XE";
$sid     = "XE";
$port   = 1521;
$dbtns = "(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = $servidor)(PORT = $port)) (CONNECT_DATA =
(SERVICE_NAME = $service_name) (SID = $sid)))";

$pdo = new PDO("oci:dbname=".$dbtns . ";charset=utf8", $usuario, $senha, $options);
```

Modelo de Banco de Dados Livraria

- Para os exemplos deste curso utilizaremos um banco de dados do domínio de uma Livraria. O banco está disponível para 4 SGBDS diferentes e já conta com um número considerável de registros nas tabelas.
- Além disso, este banco possui todos os tipos de associações:
 - a. (1..1): Funcionário tem uma Habilidade.
 - b. (1..N): Editora tem muitos Livros.
 - c. (N..1): Livros pertencem a uma Editora.
 - d. (N..N): Livros têm muitos Autores e Autores têm muitos Livros.
- E algumas Associações especiais:
 - a. **Auto relacionamento:** Funcionário tem gerente Gerente (Funcionário).
 - b. **N..N com dados associativos:** Pedido tem muitos Livros (através de ItemPedido e seus dados).
 - c. **Árvore:** Gênero tem nós filhos, irmãos e pais.

Modelo de Exemplo (livraria)



SQL (livraria v0.3 - DUMP do Banco de Dados)

Disponível em:



<https://gist.github.com/celsowm/9d0ffd735e92dc4fdff854f8847fc39>



<https://gist.github.com/celsowm/067fe51dfa612697895c8ec3b5cb436d>



<https://gist.github.com/celsowm/b139713d65d6c42df084269b3f150a2d>



<https://gist.github.com/celsowm/219c130a18289b9378fa7642508c473b>

PDO Query (PDO->query() e PDOStatement->fetch())

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query("SELECT nome FROM funcionario");  
$funcionario = $statement->fetch();  
  
echo $funcionario['nome'];
```

O método query permite submeter uma query SQL para o SGBD e retorna um objeto do tipo PDOStatement.

Para recuperar (a próxima/primeira) linha do resultado da query podemos utilizar o método fetch(), que, no caso do fetch padrão associativo, retorna um array onde as chaves são as colunas e os valores de cada linha os valores do array.

PDO Query (PDO->query() e PDOStatement->fetch())

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query("SELECT nome FROM funcionario");  
  
while($funcionario = $statement->fetch()){  
    echo $funcionario['nome']."<br>";  
}  
}
```

Podemos iterar o resultado de uma query invocando o método `fetch` até o mesmo retornar nulo, isto é, até esgotar o número de registros.

PDO Query (Transversible)

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query('SELECT nome FROM funcionario');  
foreach ($statement as $linha){  
    echo $linha['nome'] . "<br>";  
}
```

A classe PDOStatement implementa Traversable, o que permite que objetos desta classe possam ser iterados de forma transparente.

SQL Injection

"SQL Injection" é um subconjunto da vulnerabilidade de entrada do usuário não verificada / não-tratada cujo propósito é forçar o aplicativo a executar um código SQL que não foi planejado.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Login</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <form action="logar.php">
      <label>login:</label>
      <input type="text" name="login" />
      <label>senha:</label>
      <input type="password" name="senha" />
      <input type="submit" value="logar" />
    </form>
  </body>
</html>
```

```
<?php
include_once '../conexao.php';

$login = $_REQUEST['login'];
$senha = $_REQUEST['senha'];

$query = "SELECT * FROM usuario WHERE login = '$login' AND senha = '$senha'";
//var_dump($query);

$statement = $pdo->query($query);
$usuario = $statement->fetch();
if($usuario){
  echo "Usuário {$usuario['login']} logado com sucesso !";
}
```

Exemplo:
' or '1'='1

Prepared Statements

- Para evitar SQL Injection podemos utilizar o recurso de ***prepared statement*** (declaração/instrução preparada/parametrizada).
- Os ***prepared statements*** são resilientes à SQL injections porque os valores que são transmitidos posteriormente usando um protocolo diferente e não são compilados/interpretados com o código SQL original.
- Para utilizar este recurso com o PDO devemos utilizar o método `prepare()` com a query desejada substituindo os valores por placeholders (caracteres substitutos).
- Antes de recuperar os dados (`fetch`) faz-se necessário vincular os valores aos seus respectivos placeholders e executar (`execute`).
- Os Drivers PDO podem utilizar ***prepared statements*** de forma nativa (quando suportado) ou emulados pelo PDO (`PDO::ATTR_EMULATE_PREPARES`)

PDO Binding (usando parâmetros no SQL)

```
include_once "conexao.php";

$nome = 'Edson Wander';
$cpf  = '54698715324';

//posicional
$statement = $pdo->prepare('SELECT * FROM funcionario WHERE nome = ? AND cpf = ?');
$statement->execute([$nome, $cpf]);
$funcionario = $statement->fetch();

//usando key e value
$statement = $pdo->prepare('SELECT * FROM funcionario WHERE email = :email AND status=:status');
$statement->execute(['nome' => $nome, 'cpf' => $cpf]);
$funcionario = $statement->fetch();
```

Prepared Statement (evitando SQL Injection)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Login</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <form action="logar_prepared.php">
      <label>login:</label>
      <input type="text" name="login" />
      <label>senha:</label>
      <input type="password"
name="senha" />
      <input type="submit" value="logar" />
    </form>
  </body>
</html>
```

```
<?php
include_once '../conexao.php';

$login = $_REQUEST['login'];
$senha = $_REQUEST['senha'];

$query = "SELECT * FROM usuario WHERE login = ? AND
senha = ? ";
//var_dump($query);

$statement = $pdo->prepare($query);
$statement->execute([$login, $senha]);
$usuario = $statement->fetch();
if($usuario){
  echo "Usuário {$usuario['login']} logado com sucesso !";
}
```

PDO Binding (bindParam)

```
<?php

include 'conexao.php';

$nome  = 'Edson Wander';
$cpf   = '54698715324';

$stmt = $pdo->prepare('SELECT * FROM funcionario WHERE nome = ? AND
cpf = ?');
$stmt->bindParam(1, $nome,PDO::PARAM_STR);
$stmt->bindParam(2, $cpf,PDO::PARAM_STR);
$stmt->execute();
$funcionario = $stmt->fetch();

var_dump($funcionario);
```

Podemos utilizar o método bindParam para passar por referência o valor de variáveis para uma prepared statement.

No primeiro parâmetro especificamos o nome ou posição do placeholder; no segundo, a variável de referência e no terceiro o tipo desejado do valor (usando constantes do PDO).

PDO Binding (bindValue)

```
<?php  
  
include 'conexao.php';  
  
$stmt2 = $pdo->prepare('SELECT * FROM funcionario WHERE nome = ? AND  
cpf = ?');  
$stmt2->bindValue(1, 'Edson Wander', PDO::PARAM_STR);  
$stmt2->bindValue(2, '54698715324', PDO::PARAM_STR);  
$stmt2->execute();  
$funcionario2 = $stmt2->fetch();  
  
var_dump($funcionario2);
```

Podemos utilizar o método `bindValue` para passar por um valor de variáveis para uma prepared statement.

No primeiro parâmetro especificamos o nome ou posição do placeholder; no segundo, a variável de referência e no terceiro o tipo desejado do valor (usando constantes do PDO).

PDO (bindColumn)

```
<?php  
  
include 'conexao.php';  
  
$query = "SELECT id, nome, cpf FROM funcionario";  
$statement = $pdo->query($query);  
$statement->bindColumn(1, $id);  
$statement->bindColumn(2, $nome);  
$statement->bindColumn('cpf', $cpf);  
  
while ($row = $statement->fetch(PDO::FETCH_BOUND)) {  
    echo "$id:" . $nome . " " . $cpf . "<br>";  
}  
}
```

Com o uso do bindColumn é possível passar por referência o valor de uma coluna para uma variável para cada fetch realizado.

A indicação da coluna pode ser feita de maneira posicional ou pelo nome da coluna.

O ideal é utilizar sempre o estilo PDO::FETCH_BOUND que permite que o PDO possa designar valores para variáveis que foram “vinculadas” anteriormente usando bindColumn.

PDO Prepared Statement (com SQL like)

```
<?php  
  
include 'conexao.php';  
  
$sql = "SELECT * FROM livro WHERE titulo LIKE ?";  
$statement = $pdo->prepare($sql);  
$statement->execute(['%do%']);  
foreach($statement as $livro){  
    echo $livro['titulo']."</br>";  
}
```

O operador like do SQL, que permite procurar por um determinado “padrão” em um texto, pode ser utilizado também como prepared statement.

Só é válido salientar que os operadores curingas precisam ser utilizados “fora” da query, isto é, precisam ser enviados como valores para os binds.

PDO Prepared Statement (com SQL IN() “literal”)

```
<?php
include 'conexao.php';

$filtro      = ["preco_minimo" => "1.98"];
$edicoes    = [1,2,10];

$edicoes = array_combine(
    array_map(function($i){ return ':id'.$i; }, array_keys($edicoes)),
    $edicoes
);
$in_placeholders = implode(',', array_keys($edicoes));
$sql = "SELECT * FROM livro WHERE preco >= :preco_minimo AND edicao IN
($in_placeholders)";
$statement = $pdo->prepare($sql);
$statement->execute(array_merge($filtro,$edicoes));
foreach($statement as $livro){
    echo $livro['titulo']."</br>";
}
```

Infelizmente o uso de array como placeholders não é suportado nativamente no PDO, então, faz-se necessário “replicar” um conjunto de placeholders que possa representar cada valor do conjunto do IN().

É válido salientar que muitos SGBDs possuem limitação no número de valores literais em um IN.

PDO (fetchColumn)

```
<?php  
  
include_once "conexao.php";  
  
//PDO fetchColumn  
$statement = $pdo->query("SELECT id, titulo FROM livro");  
var_dump($statement->fetchColumn());  
var_dump($statement->fetchColumn(1));
```

O método `fetchColumn` retorna uma única coluna da próxima linha de um conjunto de resultados ou FALSE se não houver mais linhas.

PDO getColumnData (Introspecção)

```
<?php  
  
include 'conexao.php';  
  
$statement = $pdo->query('SELECT titulo, preco FROM livro');  
$metadados = $statement->getColumnMeta(0);  
  
echo "<pre>";  
var_dump($metadados);  
echo "</pre>";
```

O método getColumnData permite que recuperar os metadados dos campos (colunas) de uma query (inclusive campos virtuais).

O parâmetro do método é a posição da coluna em relação do descrito na query.

Infelizmente o drive do Oracle (pdo_oci) não suporta/implementa este método.

PDO FetchAll

- PDOStatement::fetchAll() retorna um array contendo todas as linhas restantes no conjunto de resultados. O array representa cada linha como uma matriz de valores de coluna ou um objeto com propriedades correspondentes a cada nome de coluna.
- Uma array vazio é retornado se houver zero resultados a serem obtidos ou retorna FALSE em caso de falha.
- Usar esse método para buscar conjuntos de resultados grandes resultará em uma grande demanda no sistema e possivelmente nos recursos da rede. Em vez de recuperar todos os dados e manipulá-los no PHP, considere o uso do servidor de banco de dados para manipular os conjuntos de resultados. Por exemplo, use as cláusulas WHERE ou **Paginação** no SQL para restringir os resultados antes de recuperá-los e processá-los com o PHP.

FetchAll (exemplo)

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query('SELECT * FROM funcionario');  
$funcionarios = $statement->fetchAll();  
foreach ($funcionarios as $funcionario) {  
    echo $funcionario['nome']."</br>";  
}
```

O resultado do fetchAll() é um array com o resultado da query.

Fetch & Fetch All (estilos)

- Ao utilizar os métodos `fetch` ou `fetchAll` de um `PDOStatement` é possível determinar qual o “formato” dos dados resultantes da query.
- Alguns estilos possuem argumentos (similar a uma função). E os valores destes argumentos influenciam na formatação dos dados.
- O estilo dos fetchs podem ser determinados das seguintes maneiras :
 - De maneira global no options do PDO (`PDO::ATTR_DEFAULT_FETCH_MODE`)
 - Através do segundo parâmetro dos métodos `fetch` e `fetchAll` do `PDOStatement`
 - Utilizando o método `setFetchMode()` em cada `PDOStatement` antes do `fetch/fetch/execute`.

PDO::FETCH_ASSOC

```
<?php  
  
include 'conexao.php';  
  
$statement = $pdo->query('SELECT nome FROM funcionario');  
$funcionarios = $statement->fetchAll(PDO::FETCH_ASSOC);  
foreach ($funcionarios as $funcionario) {  
    echo $funcionario['nome']."<br>";  
}  
  
$statement = $pdo->query('SELECT nome FROM funcionario');  
$funcionario = $statement->fetch(PDO::FETCH_ASSOC);  
echo $funcionario['nome']."<br>";
```

PDO::FETCH_ASSOC:

retorna um array indexado pelo nome da coluna conforme o retorno da query.

PDO::FETCH_NUM

```
<?php  
  
include 'conexao.php';  
  
$statement = $pdo->query("SELECT nome, id FROM funcionario");  
// $statement = $pdo->query("SELECT titulo, id FROM livro");  
$dados = $statement->fetchAll(PDO::FETCH_NUM);  
foreach ($dados as $dado) {  
    echo "nome: $dado[0] | id: $dado[1] <br/>";  
}
```

PDO::FETCH_NUM:

retorna um array 2D onde as chaves assumem as posições das colunas e os valores são os dados de cada registro nesta coluna.

PDO::FETCH_BOTH

```
<?php  
  
include 'conexao.php';  
  
$statement = $pdo->query('SELECT * FROM funcionario');  
$funcionario = $statement->fetch(PDO::FETCH_BOTH);  
var_dump($funcionario);
```

PDO::FETCH_BOTH:
retorna um array indexado duplicando o número colunas onde permitindo que os dados sejam acessados tanto pelo nome da coluna como pelo índice (número) da mesma.

PDO::FETCH_KEY_PAIR

```
<?php

include_once "conexao.php";

$statement = $pdo->query("SELECT id, nome FROM funcionario");
$dados = $statement->fetchAll(PDO::FETCH_KEY_PAIR);
echo "<select>";
foreach($dados as $key => $dado){
    echo "<option value='$key'$dado</option>";
}
echo "</select>";
```

PDO::FETCH_KEY_PAIR:

retorna um array 2D onde as chaves assumem os valores da primeira coluna e os valores são os dados da segunda.

Se uma terceira coluna for colocada na projeção, o PDO lançará uma exceção

PDO::FETCH_UNIQUE

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query("SELECT id,nome,cpf FROM funcionario");  
$funcionarios = $statement->fetchAll(PDO::FETCH_UNIQUE);  
foreach ($funcionarios as $id => $funcionario) {  
    echo $id."/".$funcionario['nome']."<br>";  
}
```

PDO::FETCH_UNIQUE:
retorna um array de arrays onde as chaves assumem os valores da primeira coluna e os valores (com colunas em keys) em subarrays.

PDO::FETCH_NAMED

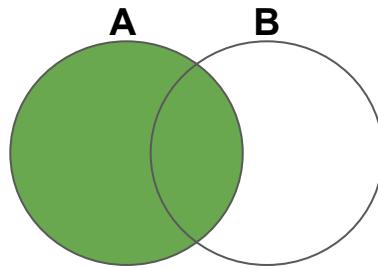
```
<?php  
  
$statement = $pdo->query('  
SELECT * FROM editora  
LEFT JOIN livro ON livro.editora_id = editora.id');  
  
$registros = $statement->fetchAll(PDO::FETCH_NAMED);  
  
foreach ($registros as $registro) {  
    echo "editora id {$registro['id'][0]} e  
          livro {$registro['id'][1]} <br/>";  
}  
}
```

PDO::FETCH_NAMED:

retorna um array multidimensional onde campos com nomes repetidos são colocados dentro de um mesmo array onde, a ordem dos mesmos é referente a ordem de suas respectivas tabelas na própria query.

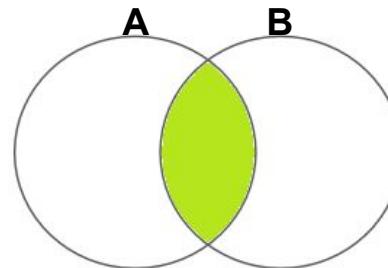
JOIN SQL

LEFT JOIN: junção de A e B, porém, recupere tudo de A mesmo se não houver uma referência em B (ficando nulos valores para B sem referência para A).



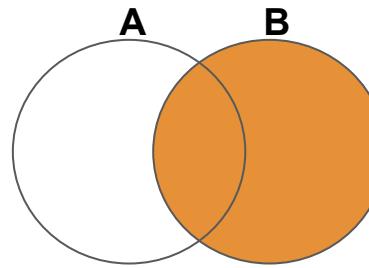
```
SELECT * FROM funcionario  
LEFT JOIN habilitacao ON  
funcionario.id =  
habilitacao.funcionario_id
```

INNER JOIN: junção de A e B, porém recupere apenas se houver registro A e B com referências.



```
SELECT * FROM pedido  
INNER JOIN cliente ON  
cliente.id = pedido.cliente_id
```

RIGHT JOIN: junção de A e B, porém, recupere tudo de B mesmo se não houver uma referência em A (ficando nulos valores para A sem referência para B).



```
SELECT * FROM livro  
RIGHT JOIN editora ON editora.id  
= livro.editora_id
```

PDO::FETCH_GROUP

```
<?php

include 'conexao.php';

$statement = $pdo->query("SELECT edicao, id, titulo
FROM livro order by edicao");

$livrosPorEdicao = $statement->fetchAll(PDO::FETCH_GROUP);

foreach ($livrosPorEdicao as $key => $livros) {
    echo "Livros na $key ª edição: <br>";
    foreach ($livros as $livro) {
        echo "-- {$livro['titulo']} <br>";
    }
    echo "<br/>";
}
```

PDO::FETCH_GROUP:
retorna os registros agrupados pela primeira coluna da query em arrays multidimensionais.

PDO::FETCH_LAZY

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query('SELECT * FROM livro');  
$livro = $statement->fetch(PDO::FETCH_LAZY);  
echo "ISBN: {$livro['isbn']} : {$livro->título} <br>";
```

PDO::FETCH_LAZY: Faz um busca tardia/lenta/por demanda e retornar a próxima linha como um objeto anônimo com nomes de colunas como atributos. O desempenho costuma ser notável ao recuperar uma grande massa de dados (linhas).



Devido à própria natureza de busca “tardia” (e por demanda) dos registros do banco de dados, o **FETCH_LAZY não funciona com o fetchAll()**.

PDO::FETCH_BOUND

```
<?php

include 'conexao.php';

$query = "SELECT id, nome, cpf FROM funcionario";
$statement = $pdo->query($query);
$statement->bindColumn(1, $id);
$statement->bindColumn(2, $nome);
$statement->bindColumn('cpf', $cpf);

while ($row = $statement->fetch(PDO::FETCH_BOUND)) {
    echo "$id:" . $nome . " " . $cpf . "<br>";
}
```

PDO::FETCH_BOUND:

Permite que o PDO possa designar valores para variáveis que foram “vinculadas” anteriormente usando bindColumn.

PDO::FETCH_COLUMN (Estilos com argumentos)

```
<?php
include_once "conexao.php";

$statement = $pdo->query('SELECT nome, id FROM funcionario');
$nomes    = $statement->fetchAll(PDO::FETCH_COLUMN);
foreach ($nomes as $nome) {
    echo $nome."</br>";
}

$statement = $pdo->query('SELECT id, nome, cpf FROM funcionario');
$cpfs     = $statement->fetchAll(PDO::FETCH_COLUMN, 2);
foreach ($cpfs as $cpf) {
    echo $cpf."</br>";
}
```

PDO::FETCH_COLUMN:
retorna um array 2D onde os valores são os dados de uma única coluna e uma chave sequencial.

Seu parâmetro é similar o [fetchColumn](#), onde um inteiro é utilizado para indicar a coluna desejada.

PDO::FETCH_FUNC (Estilos com argumentos)

```
<?php

include_once "conexao.php";

$statement = $pdo->query('SELECT preco, titulo FROM livro');
$livros = $statement->fetchAll(PDO::FETCH_FUNC, function($preco, $titulo){
    $preco_no_cartao = round($preco + ($preco * 0.1),2);
    return "$titulo: R$ {$preco} à vista e no cartão R$ {$preco_no_cartao}";
});

foreach ($livros as $preco) {
    echo $preco."</br>";
}
```

PDO::FETCH_FUNC:

Permite que uma função/closure receberá em forma de parâmetros as colunas da query.

PDO e ORM (Object Relational Mapper)

- Alguns estilos do PDO permitem um mapeamento primitivo entre registros de uma tabela (modelo relacional) e o instâncias/objetos de uma classe (modelo orientado a objetos).
- As maiores limitações começam a surgir quando é necessário representar/mapear relacionamentos entre objetos.
- Para tarefas mais complexas de mapeamento é recomendável utilizar bibliotecas ORM de terceiros.

PDO (Fetch Class)

Livro.php

```
<?php  
class Livro {  
    public $id;  
    public $titulo;  
    private $dedicao; }
```

```
//recuperando dados  
include_once "conexao.php";  
include_once "Livro.php";  
  
$sth = $pdo->prepare("SELECT * FROM livro");  
$sth->execute();  
  
$result = $sth->fetchAll(\PDO::FETCH_CLASS, 'Livro');  
print_r($result);
```

O comportamento padrão do `FETCH_CLASS` é chamar o construtor depois de colocar os valores nos atributos.

PDO FETCH_CLASS + FETCH_CLASSTYPE

```
<?php

include_once "conexao.php";

class Autor {}
class Funcionario {}

$stmt = $pdo->query(
"SELECT 'Autor', nome FROM autor
UNION
SELECT 'Funcionario', nome FROM funcionario");
$objeto = $stmt->fetchAll(PDO::FETCH_CLASS |
PDO::FETCH_CLASSTYPE);
print_r($objeto);
```

PDO::FETCH_CLASSTYPE:

Combinado com FETCH_CLASS permite que a primeira coluna da query seja utilizada para definir qual classe utilizar para instância o registro.

Pode ser muito útil para o caso de associações polimórficas com simulação de herança no modelo relacional.

PDO::FETCH_CLASS + PDO::FETCH_PROPS_LATE

```
<?php
class Funcionario {

    public $id;
    public $nome;
    public $habilitacao;

    public function __construct() {
        $this->habilitacao = new Habilitacao();
    }

    public function __set($name, $value) {
        if (array_key_exists($name, get_object_vars($this->habilitacao))) {
            $this->habilitacao->$name = $value;
        } else {
            $this->$name = $value;
        }
    }
}
```

```
<?php
class Habilitacao {

    public $numero;
    public $categoria;

}
```

Podemos utilizar a flag `FETCH_PROPS_LATE` para que o PDO comece a “colocar” os valores do objeto após a chamada do método construtor.
Com isso podemos decorar nosso objeto e mapear um relacionamento 1..1, como neste exemplo.

PDO::FETCH_CLASS + PDO::FETCH_PROPS_LATE

```
include 'conexao.php';
include 'Funcionario.php';
include 'Habilitacao.php';

$statement = $pdo->prepare(
    "SELECT * FROM funcionario "
    . "LEFT JOIN habilitacao "
    . "ON funcionario.id = habilitacao.funcionario_id");
$statement->execute();

$funcionarios = $statement->fetchAll(\PDO::FETCH_CLASS | \PDO::FETCH_PROPS_LATE,
Funcionario::class);

foreach ($funcionarios as $funcionario) {
    echo "{$funcionario->nome} com Habilidade nº {$funcionario->habilitacao->numero}<br/>";
}
```

PDO::FETCH_OBJ

```
<?php  
  
include_once "conexao.php";  
  
$statement = $pdo->query('SELECT * FROM livro');  
$livros = $statement->fetchAll(PDO::FETCH_OBJ);  
  
foreach ($livros as $livro) {  
    echo "instância de ".get_class($livro)."::";  
    echo $livro->título."</br>";  
}  
}
```

PDO::FETCH_OBJ:

Retorna os registros em forma de objetos genéricos (instâncias de StdClass).

fetchObject

```
<?php

include 'conexao.php';

class Livro {

    public function __construct($etiqueta = null) {
        $this->etiqueta = $etiqueta;
    }

}

$statement = $pdo->query("SELECT * FROM livro");
$livro = $statement->fetchObject(Livro::class, [123456]);
var_dump($livro);
```

Vale a pena salientar que o fetchObject cria a instância e seta os valores antes mesmo do método construtor ser executado.

PDO:FETCH_INTO

```
<?php  
class Livro {  
    public $titulo;  
    public $preco;  
    public $isbn;  
    public $edicao;  
    public $ano_publicacao;  
}  
  
$livro = new Livro();  
  
$statement = $pdo->query('SELECT * FROM livro');  
$statement->setFetchMode(PDO::FETCH_INTO, $livro);  
$statement->fetch();  
  
var_dump($livro);
```

PDO::FETCH_INTO:

Permite atualizar uma instância existente da classe solicitada, mapeando as colunas do conjunto da query para os atributos nomeadas na classe.



Atenção: Diferente do **FETCH_CLASS**, o **FETCH_INTO** não é capaz de modificar valores de atributos privados ou protegidos

PDO::FETCH_CLASS + PDO::FETCH_SERIALIZE

```
<?php  
  
include 'conexao.php';  
  
class Log implements \Serializable {  
    public $id;  
    public $dados;  
  
    public function serialize() {  
        return serialize((array) $this);  
    }  
    public function unserialize($serialized): void {  
        foreach (unserialize($serialized) as $p => $v) {  
            $this->{$p} = $v;  
        }  
    }  
}
```

```
$statement = $pdo->query('SELECT dados FROM log');  
$statement->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_SERIALIZE,  
Log::class);  
$logs = $statement->fetchAll();  
foreach ($logs as $log) {  
    echo "id: $log->id | ";  
    echo "dados: " . var_export($log->dados, true);  
    echo "<br/>";  
}
```

PDO::FETCH_SERIALIZE: Tem efeito similar ao FETCH_INTO porém é utilizada os comportamentos da serialização (interface) nos objetos.

PDO (operações com cursor)

- Quando um cursor é criado para uma consulta, é possível iterar sobre o conjunto de linhas sem obter o resultado inteiro da mesma de uma só vez.
- Com um cursor não rolável (forward-only) é possível efetuar FETCH em cada linha no máximo uma vez, e o cursor se move automaticamente para a linha seguinte.
- Já os cursores roláveis (scrollable) permitem iterar o resultado em diversas direções, inclusive para trás e por isso podem acessar a mesma linha no conjunto do resultados várias vezes. Assim, modificações de dados (inserir, atualizar, excluir operações) de outras transações podem ter um impacto no conjunto de resultados.

PDO Cursos (Exemplo Pedidos)

```
<?php
include_once '../conexao_postgres.php'; //sqlserver, oracle e postgres
$query = "select * from pedido order by id";
$statement = $pdo->prepare($query, [PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL]);
$statement->execute();

$pedido = $statement->fetch(PDO::FETCH_LAZY, PDO::FETCH_ORI_FIRST);
print "Primeiro pedido (id:{$pedido['id']}) ocorreu em : {$pedido['data']}<br/>";

$pedido = $statement->fetch(PDO::FETCH_LAZY, PDO::FETCH_ORI_NEXT);
print "Próximo pedido (id:{$pedido['id']}) ocorreu em : {$pedido['data']}<br/>";

$pedido = $statement->fetch(PDO::FETCH_LAZY, PDO::FETCH_ORI_LAST);
print "Último pedido (id:{$pedido['id']}) ocorreu em: {$pedido['data']}<br/>";

$pedido = $statement->fetch(PDO::FETCH_LAZY, PDO::FETCH_ORI_PRIOR);
print "Penúltimo pedido (id:{$pedido['id']}) ocorreu em: {$pedido['data']}<br/>";

$pedido = $statement->fetch(PDO::FETCH_LAZY, PDO::FETCH_ORI_ABS, 3);
print "Terceiro pedido (id:{$pedido['id']}) em: {$pedido['data']}<br/>";
```

Em SGBDs que suportam o recurso de cursor scroll, é possível, no momento do método fetch(), posicionar o cursor em diferentes pontos do resultado, bem como movimentá-lo para frente ou para trás.

No SQL Server o ABS começa com zero.

PDO Exception

```
<?php  
  
include_once 'conexao.php';  
  
try{  
  
    $pdo->query('SELECT * FROM nao_existe');  
  
} catch (\PDOException $t) {  
  
    echo "mensagem:".$t->getMessage()."<br/>".  
        "código:".$t->getCode();  
}
```

O PDO também faz parte da Taxonomia de Throwable e possui uma Exception com informações diretamente específicas do SGBD que podem auxiliar no tratamento do erro.



Atenção: Para que o PDO lance exceções, é necessário utilizar o atributo de conexão PDO::ATTR_ERRMODE com o valor PDO::ERRMODE_EXCEPTION.

SQL DML: Inserção, Atualização e Remoção

- Os recursos da linguagem SQL são normalmente divididos em conjuntos onde cada um destes possui um propósito específico.
- Um destes conjuntos é chamado de DML (Data Manipulation Language / Linguagem de Manipulação de Dados).
- O DML é composto pelas seguintes instruções (statements): INSERT, UPDATE e DELETE.
- É fortemente recomendado utilizar *prepared statement* ao executar qualquer instrução DML no PDO.

PDO (Inserção)

```
<?php  
  
include_once "conexao.php";  
  
try {  
    $statement = $pdo->prepare("INSERT INTO funcionario (nome, cpf)"  
        . "VALUES (?,?)");  
    $statement>execute([$nome, $cpf]);  
  
} catch (\PDOException $t) {  
  
    echo "mensagem:".$t->getMessage()."<br/>".  
    "código:".$t->getCode();  
}
```

A instrução (statement) **INSERT** SQL permite adicionar um ou mais registros a qualquer tabela única em um banco de dados relacional.

PDO (Atualização)

```
<?php  
  
include_once "conexao.php";  
  
try {  
  
    $stmt = $pdo->prepare("UPDATE livro SET preco += 1 WHERE id= :id ");  
    $stmt->execute(['id'=>$id]);  
  
} catch (\PDOException $t) {  
  
    echo "mensagem:".$t->getMessage()."<br/>".  
        "código:".$t->getCode();  
}
```

A instrução (statement) **UPDATE** SQL permite alterar os dados de um ou mais registros em uma tabela. Todas as linhas podem ser atualizadas ou um subconjunto pode ser escolhido usando uma condição (WHERE).

PDO (Remoção)

```
<?php  
  
include_once "conexao.php";  
  
try {  
    $stmt = $pdo->prepare("DELETE FROM livro WHERE id= :id ");  
    $stmt->execute(['id'=>$id]);  
  
} catch (\PDOException $t) {  
  
    echo "mensagem:".$t->getMessage()."<br/>".  
        "código:".$t->getCode();  
}
```

A instrução (statement) **DELETE** SQL permite remover um ou mais registros de uma tabela. Um subconjunto pode ser definido para exclusão usando uma condição (WHERE), caso contrário, todos os registros serão removidos. Alguns SGBDs, como o MySQL, permitem a exclusão de linhas de várias tabelas com uma instrução DELETE (multi-table delete)

Conjunto de DMLs (sem transação)

```
UPDATE conta_corrente SET saldo = saldo - 200 WHERE cliente_id = 1;  
select sleep(30); -- se durante este tempo o SGBD cair, a próxima instrução nunca ocorrerá  
UPDATE conta_corrente SET saldo = saldo + 200 WHERE cliente_id = 2;
```

É muito comum a utilização de duas ou mais instruções SQL DML de forma sequencial para cumprir a lógica de um caso de uso da aplicação.

Um exemplo seria o caso de uso: “transferência bancária”

Neste exemplo, R\$ 200 serão “transferidos” da conta do cliente 1 para o cliente 2.

PDO x Transação

- Uma transação, no contexto de um banco de dados, é uma unidade lógica que é executada de forma independente para recuperação ou atualização de dados.
- No SQL ANSI uma transação começa com BEGIN TRANSACTION e, após a mesma, todas as instruções serão parte desta transação.
- Para “confirmar” a transação, faz-se necessário utilizar a instrução COMMIT
- Para “cancelar” todas as instruções da transação e retornar o banco ao estado original, faz-se necessário utilizar o comando ROLLBACK
- Em bancos de dados relacionais, as transações do banco de dados devem ser atômicas, consistentes, isoladas e duráveis - resumidas como o acrônimo ACID.
- Alguns SGBDs como o MySQL, utiliza outra sintaxe (START ao invés de BEGIN). Utilizando o PDO estas diferenças são suprimidas pois o mesmo possui um método único para todos.

Transação ACID

- **Atomicidade:** Atomicidade garante que cada transação seja tratada de forma "unitária" e, se alguma das instruções que constituem uma transação não for concluída, a transação inteira falhará e o banco de dados permanecerá inalterado;
- **Consistência:** A consistência garante que uma transação só pode trazer o banco de dados de um estado válido para outro, mantendo as regras do banco: quaisquer dados gravados no banco de dados devem ser válidos de acordo com todas as regras definidas, incluindo restrições, triggers e qualquer combinação dos mesmos. Isso evita corrupção do banco de dados por uma transação ilegal, mas não garante que uma transação esteja correta;
- **Isolamento:** As transações geralmente são executadas simultaneamente (por exemplo, ler e gravar em várias tabelas ao mesmo tempo). O isolamento garante que a execução simultânea de transações deixe o banco de dados no mesmo estado que teria sido obtido se as transações fossem executadas sequencialmente. O isolamento é o principal objetivo do controle de concorrência;
- **Durabilidade:** A durabilidade garante que uma vez que uma transação tenha sido confirmada, ela permanecerá comprometida mesmo no caso de uma falha no sistema (por exemplo, falta de energia ou falha). Isso geralmente significa que as transações concluídas (ou seus efeitos) são registradas na memória não volátil.

PDO Transaction (Inserção)

```
<?php  
  
include_once "conexao.php";  
  
try {  
    $pdo->beginTransaction();  
    $stmt = $pdo->prepare("INSERT INTO funcionario (nome) VALUES (?)");  
    foreach(['José da Silva','Maria das Dores'] as $name){  
        $stmt->execute([$name]);  
    }  
    $pdo->commit();  
}catch (Exception $e){  
    $pdo->rollback();  
    throw $e;  
}
```

Neste exemplo, dois inserts são executados dentro de uma transação, porém, apenas com a confirmação feita com o commit, é que as mesmas serão de facto executadas.

Para que a transação não fique aberta em caso de erro, utilizaremos o rollback dentro do catch

PDO Transaction (Atualização)

```
<?php

include_once "conexao.php";

try {
    $pdo->beginTransaction();
    $stmt = $pdo->prepare("UPDATE livro SET preco += 1 WHERE id= :id ");
    $stmt->execute(['id'=>$id]);
    $pdo->commit();
} catch (Exception $e){
    $pdo->rollback();
    throw $e;
}
```

PDO Transaction (Remoção)

```
<?php

include_once "conexao.php";

try {
    $pdo->beginTransaction();
    $stmt = $pdo->prepare("DELETE FROM livro WHERE id= :id ");
    $stmt->execute(['id'=>$id]);
    $pdo->commit();
} catch (Exception $e){
    $pdo->rollback();
    throw $e;
}
```

PDO::lastInsertId

- Muitas vezes, na lógica da aplicação, faz-se necessário persistir mais de uma entidade e manter o relacionamento entre elas.
- Para isso, é preciso recuperar a chave primária da entidade forte e utilizar como chave estrangeira na entidade fraca.
- O PDO possui um método chamado `lastInsertId()` que permite recuperar a última chave primária persistida no banco. Infelizmente nem todos os bancos/drivers suportam esse recurso e faz-se necessário utilizar algum recurso proprietário dos mesmos.

PDO::lastInsertId

```
<?php
include 'conexao.php';
try {

    $pdo->beginTransaction();
    $sql = "INSERT INTO funcionario (nome, cpf) VALUES(:nome,:cpf) ";
    if($pdo->getAttribute(PDO::ATTR_DRIVER_NAME) == 'oci'){
        $sql .= 'RETURNING id INTO :last_id';
    }

    $statement = $pdo->prepare($sql);
    $statement->bindValue('nome','Jaqueline');
    $statement->bindValue('cpf','12269736044');

    if($pdo->getAttribute(PDO::ATTR_DRIVER_NAME) == 'oci'){
        $statement->bindParam('last_id', $funcionario_id,
            PDO::PARAM_INT, 8);
    }
    $statement->execute();
}
```

```
if($pdo->getAttribute(PDO::ATTR_DRIVER_NAME) != 'oci'){

    $funcionario_id = $pdo->lastInsertId();

}

$sql2 = "INSERT INTO habilitacao (numero, categoria,
funcionario_id) VALUES(:numero,:categoria,:funcionario_id)";

$statement2 = $pdo->prepare($sql2);
$statement2->execute(['95685512398','B',$funcionario_id]);
$pdo->commit();

} catch (PDOException $e) {
    $dbh->rollback();
    print "Error! " . $e->getMessage() . "<br>";
}
```

PDO x BLOB

- A maioria dos SGDS tem suporte ao armazenamento de BLOBs
- Um Binary Large Object (BLOB) é uma coleção de dados binários armazenados como uma entidade única em um sistema de gerenciamento de banco de dados. Os blobs são tipicamente imagens, áudio ou outros objetos multimídia, embora às vezes o código executável binário seja armazenado como um blob.
- Com o PDO o bind (vinculação) de um valor BLOB é feito utilizando a tipagem como LOB através do PARAM_LOB
- Alguns bancos como o SQL Server exigem o uso de um constante proprietária para indicar corretamente ao Drive ODBC o uso recurso.
- Outros bancos como o Oracle exigem a necessidade da criação de um BLOB vazio via SGBD e, através de um ponteiro virtual de arquivo (resource) a “persistência” dos dados.

PDO: Inserção de Binário (formulário)

```
<!DOCTYPE html>
<html>
<head>
<title>Incluir Nova Foto</title>
<meta charset="utf-8"/>
</head>
<body>
<form action="blob_action.php" method="post" enctype="multipart/form-data">
    Funcionário: <select name="funcionario_id">
        <?php include 'conexao.php';
        $funcionarios = $pdo->query('SELECT id, nome FROM funcionario')
            ->fetchAll(PDO::FETCH_KEY_PAIR);
        foreach($funcionarios as $key => $value){
            echo "<option value='$key'>$value</option>";
        }
        ?>
    </select>
    Foto: <input type="file" name="foto" accept="image/*"><br>
    <input type="submit" value="cadastrar"/>
</form>
</body>
</html>
```

Neste exemplo, será submetido via formulário o id do funcionário e uma imagem que será persistida na tabela foto.

PDO: Inserção de Binário (action - parte 1)

```
<?php

include 'conexao.php';

try {
    $pdo->beginTransaction();

    $funcionario_id = $_REQUEST['funcionario_id'];
    $arquivo = $_FILES['foto']['tmp_name'];
    $binario = file_get_contents($arquivo);
    $mimetype = $_FILES['foto']['type'];

    $sql = "INSERT INTO foto (binario, mimetype, funcionario_id)";
    $sql_values = " VALUES (:binario, :mimetype, :funcionario_id)";

    $driver = $pdo->getAttribute(PDO::ATTR_DRIVER_NAME);
    if ($driver == 'oci') {
        $sql_values = " VALUES (EMPTY_BLOB(), :mimetype, :funcionario_id)"
            . " RETURNING binario INTO :binario";
    }
}
```

Devido às peculiaridades do Oracle, faz-se necessário particularizar o bind dos parâmetros, primeiramente, criando um BLOB vazio com a função Oracle EMPTY_BLOB() e recuperar o resource do mesmo no placeholder :binario

PDO: Inserção de Binário (action - parte 2)

```
$statement = $pdo->prepare($sql . $sql_values);
switch ($driver) {
    case 'sqlsrv':
        $statement->bindParam('binario', $binario, PDO::PARAM_LOB, 0, PDO::SQLSRV_ENCODING_BINARY);
        break;
    case 'oci':
        $statement->bindParam('binario', $binario_resource, PDO::PARAM_LOB);
        $binario_resource = fopen($arquivo, 'rb');
        break;
    default:
        $statement->bindParam('binario', $binario, PDO::PARAM_LOB);
        break;
}
$statement->bindValue('mimetype', $mimetype, PDO::PARAM_STR);
$statement->bindValue('funcionario_id', $funcionario_id, PDO::PARAM_INT);
$statement->execute();
$pdo->commit();
} catch (\PDOException $ex) {
    $pdo->rollback();
    echo $ex->getMessage();
}
```

Tipo Resource

- Um **resource** é uma variável especial, mantendo uma referência a um recurso externo. Recursos são criados e usados por funções especiais.
- Resources normalmente funcionam como identificadores (referências) especiais para arquivos abertos, conexões de banco de dados, áreas de tela de imagem e semelhantes.
- Devido a natureza dos resources, a conversão dos mesmos não é possível.

Recuperação de binários do Banco (blob/varbinary)

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <title>Foto de BLOB</title>
    <meta charset="UTF-8">
  </head>
  <body>
    
  </body>
</html>
```

Neste exemplo recuperamos uma imagem armazenada no banco. Alguns drivers utilizam um link em forma de resource (recurso) para acessar o binário de forma otimizada. Para estes casos, utilizamos o `stream_get_contents()` para recuperar o conteúdo binário.

`ob_clean()` é uma função do PHP para ajudar a limpar o “output buffer” e evitar algum caráter indesejável na composição do echo do binário, o que poderia corromper o mesmo

```
<?php
include 'conexao.php';

$id = $_REQUEST['funcionario_id'];
try {
    $statement = $pdo->prepare("SELECT * FROM foto WHERE funcionario_id = ? ");
    $statement->execute([$id]);
    $foto = $statement->fetch(PDO::FETCH_ASSOC);
} catch (Exception $exc) {
    echo $exc->getTraceAsString();
}

ob_clean();
header("Content-type: {$foto['mimetype']}");
if (is_resource($foto['binario'])) {
    echo stream_get_contents($foto['binario']);
} else {
    echo ($foto['binario']);
}
```

PDO - Formatação de datas (dd/mm/YYYY)

```
<?php
include 'conexao.php';
switch ($pdo->getAttribute(PDO::ATTR_DRIVER_NAME)) {
    case 'mysql':
        $data = "DATE_FORMAT(data, '%d/%m/%Y')";
        break;
    case 'sqlsrv': //>= 2012
        $data = "FORMAT(data, 'd', 'pt-BR')";
        break;
    case 'pgsql':
    case 'oci':
        $data = "TO_CHAR(data, 'dd/mm/yyyy') as";
        break;
}
$statement = $pdo->query("SELECT $data data FROM pedido");
$pedidos  = $statement->fetchAll();

foreach ($pedidos as $pedido) {
    echo $pedido['data']."<br/>";
}
```

Normalmente, os SGBDs armazenam as datas utilizando o padrão YYYY-MM-DD.

Existem diferentes maneiras para recuperar datas no formato brasileiro, dentre elas, podemos utilizar funções específicas de SQL de cada SGBD (t-sql, plsql etc.) para já recuperar, diretamente da query, a data formatada.

Lembrando que esta formatação também pode ser feita utilizando o format() da classe Date conforme demonstrado [aqui](#).

PDO (paginação)

- O recurso/técnica de paginação em repositórios de dados permite que uma parte dos dados seja carregada por demanda do usuário.
- Com a paginação do lado do servidor, o número de registros de uma consulta também será limitada pelo limite estabelecido na paginação.
- O número e o tamanho das páginas digitais em um documento são limitados pela quantidade de dados no repositório de dados, não pelos dispositivos de vídeo ou pela quantidade de “papel”.

PDO paginação (parte 1)

```
<?php
include_once 'conexao.php';

$pagina = (isset($_REQUEST['pagina'])) ? $_REQUEST['pagina'] : 1;
$limit = 5;
$inicio = $pagina * $limit;
$offset = ($pagina - 1) * $limit;

$query = "SELECT id, titulo, preco, isbn, edicao, ano_publicacao FROM livro ";
$query_total = $pdo->query("SELECT COUNT(*) FROM ($query) q");
$total = $query_total->fetchColumn();
$statement = limit($pdo, $query, $limit, $offset);
$livros = $statement->execute();
```

PDO (paginação parte 2)

```
function limit($pdo, string $query, int $limit, int $offset, string $order = 'id') : \PDOStatement{  
    $query_limit = "";  
    switch ($pdo->getAttribute(PDO::ATTR_DRIVER_NAME)) {  
        case 'sqlsrv': //>= 2012  
        case 'oci': //>= 12c  
            $query_limit = "$query ORDER BY $order OFFSET :offset ROWS FETCH NEXT :limit ROWS ONLY";  
            break;  
        default: //mysql e postgres  
            $query_limit = "$query ORDER BY $order LIMIT :limit OFFSET :offset";  
            break;  
    }  
    $statement = $pdo->prepare($query_limit);  
    $statement->bindValue(':offset', (int) $offset, PDO::PARAM_INT);  
    $statement->bindValue(':limit', (int) $limit, PDO::PARAM_INT);  
    return $statement;  
}
```

PDO paginação (parte 3)

```
function montaLinha(array $row, $tag = 'td'){
    return "<tr>".implode("",array_map(function($row) use ($tag){
        return "<$tag>" . $row . "</{$tag}>";
    }, $row))."</tr>";
}

echo "<table border>";
echo montaLinha(['ID','Título','Preço','ISBN','Edição','Ano'], 'th');
while ($row = $statement->fetch()){
    echo montaLinha($row);
}
echo "</table>";

echo (($pagina-1) > 0) ? "<a href='index.php?pagina=".($pagina-1)."'">Anterior</a>" : "Anterior";
echo "&nbsp";
echo (($pagina)*$limit < $total) ? "<a href='index.php?pagina=".($pagina+1)."'">Próximo</a>" : "Próximo";
```

PDO - Paginação com Filtro

- Muitas vezes é necessário permitir que o usuário possa recuperar um subconjunto específico dos dados persistidos pelo sistema
- Com o recurso do formulário é possível permitir que o usuário possa escolher e/ou digitar determinados valores que, aplicados à query, afetará o resultado final exibido na lista/tabela
- Utilizando paginação síncrona, faz-se necessário persistir o filtro nos links da paginação permitindo que, via get, os valores possam ser reutilizados na próxima página pela query.

PDO Paginação com Filtro (parte 1)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
<form>
<label>Nome:</label>
<input name="nome" value=<?php echo isset($_REQUEST['nome']) ? $_REQUEST['nome'] : '';?>">
<label>CPF:</label>
<input name="cpf" value=<?php echo isset($_REQUEST['cpf']) ? $_REQUEST['cpf'] : '';?>">
<input type="submit" value="filtrar" />
</form>
```

Neste formulário, usaremos o PHP para recuperar os valores dos campos de pesquisa, caso os mesmos tenham sido usados (e repassados pelos links da paginação e/ou submissão do próprio formulário).

PDO Paginação com Filtro (parte 2)

```
<?php
// put your code here
include 'conexao_oracle.php';

function limit(\PDO $pdo, string $query, int $limit, int $offset, string $order = 'id'): \PDOStatement {
    $query_limit = "";
    switch ($pdo->getAttribute(PDO::ATTR_DRIVER_NAME)) {
        case 'sqlsrv':
        case 'oci':
            $query_limit = "$query ORDER BY $order OFFSET :offset ROWS FETCH NEXT :limit ROWS ONLY";
            break;
        default:
            $query_limit = "$query ORDER BY $order LIMIT :limit OFFSET :offset";
            break;
    }
    $statement = $pdo->prepare($query_limit);
    $statement->bindValue('offset', (int) $offset, PDO::PARAM_INT);
    $statement->bindValue('limit', (int) $limit, PDO::PARAM_INT);
    return $statement;
}
```

PDO Paginação com Filtro (parte 3)

```
$pagina = (isset($_REQUEST['pagina'])) ? $_REQUEST['pagina'] : 1;
unset($_REQUEST['pagina']);
$parametros = [];
$nome = "";
$cpf = "";
$limit = 5;
$inicio = $pagina * $limit;
$offset = ($pagina - 1) * $limit;
$query = "SELECT * FROM funcionario WHERE 1=1 ";
if(!empty($_REQUEST['nome'])){
    $nome = $_REQUEST['nome'];
    $parametros['nome'] = $nome;
    $query .= " AND nome LIKE :nome";
}
if(!empty($_REQUEST['cpf'])){
    $cpf = $_REQUEST['cpf'];
    $parametros['nome'] = $cpf;
    $query .= " AND cpf LIKE :cpf";
}
```

PDO Paginação com Filtro (parte 4)

```
$query_total = $pdo->prepare("SELECT COUNT(*) FROM ($query) q");
$query_total->execute($parametros);
$total = $query_total->fetchColumn();
$statement = limit($pdo, $query, $limit, $offset);
(!$nome) ?: $statement->bindValue('nome', "%$nome%");
(!$cpf) ?: $statement->bindValue('cpf', "%$cpf%");
$funcionarios = $statement->execute();
echo "<table border>";
echo "<tr><th>Nome</th><th>CPF</th></tr>";
while($funcionario = $statement->fetch()){
    echo "<tr>";
    echo "<td>{$funcionario['nome']}
```

PDO e Information Schema

- Com o Information Schema é possível fazer uma introspecção nas tabelas do SGBD.
- Informações de metadados como campos obrigatórios, tipos e ou até limites de tamanho podem ser obtidos para, por exemplo, validar entrada dos usuários de forma dinâmica.
- Tal recurso é primordial para tornar dinâmico algumas tarefas repetitivas como validação de campos e essencial na construção de uma biblioteca ORM.

PDO e Information Schema

```
<?php
include 'conexao_oracle.php';

$driver = $pdo->getAttribute(\PDO::ATTR_DRIVER_NAME);

$table = 'editora';
$colunas = 'COLUMN_NAME, IS_NULLABLE, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH';
$from = 'information_schema.COLUMNS';
if($driver == 'oci'){
    $table = strtoupper($table);
    $from = 'all_tab_columns';
    $colunas = 'column_name, nullable as IS_NULLABLE, data_type, data_length as
CHARACTER_MAXIMUM_LENGTH';
}
$query = "SELECT $colunas "
. " FROM $from WHERE table_name = ?";
$stmt = $pdo->prepare($query);
$stmt->execute([$table]);
$schema = $stmt->fetchAll();
```

Neste exemplo será recuperado os metadados de cada coluna da tabela editora.

Como o Oracle não suporta o Information Schema ANSI, faz-se necessário uma adaptação para utilizar seu equivalente.

PDO e Information Schema

```
echo "<table>";
echo "<tr><th>name</th><th>is null?</th><th>type</th><th>size</th></tr>";
foreach($schema as $coluna_schema){
    $coluna_schema = array_change_key_case($coluna_schema,
CASE_UPPER);
    echo "<tr>";
    echo "<td>{$coluna_schema['COLUMN_NAME']}
```

Para compatibilizar a caixa utilizada por alguns SGBDs, o array_change_key_case com a constante CASE_UPPER manterá todos os índices do resultado do fetchAll() em caixa alta (padrão ansi do information schema)

Estruturas de Dados com Classes SPL (Standard PHP Library)

- A Biblioteca Padrão do PHP (SPL) é uma coleção de interfaces e classes que se destinam a solucionar problemas comuns.
- Nenhuma biblioteca externa é necessária para construir essa extensão e está disponível e compilada por padrão desde o PHP 5.0.
- A SPL fornece um conjunto de:
 - Estrutura de dados (array, pilha, fila etc.)
 - Iteradores para percorrer objetos,
 - Interfaces,
 - Exceptions

SplFixedArray

- A classe SplFixedArray fornece as principais funcionalidades do array. A principal diferença entre um SplFixedArray e um array PHP normal é que o SplFixedArray é de tamanho fixo e permite apenas inteiros como valores para os índices.
- Notoriamente o SplFixedArray consome um número menor de memória RAM e, muitas vezes, também é mais rápido quando comparado ao array “tradicional” do PHP.

SplFixedArray

```
<?php  
  
$fixed = new SplFixedArray(3);  
  
$fixed[0] = 'A';  
$fixed[1] = 'B';  
$fixed[2] = 'C';  
  
foreach ($fixed as $item) {  
    echo $item, PHP_EOL;  
}  
  
$fixed->setSize(2); //diminuindo o array  
  
foreach ($fixed as $item) {  
    echo $item, PHP_EOL;  
}
```

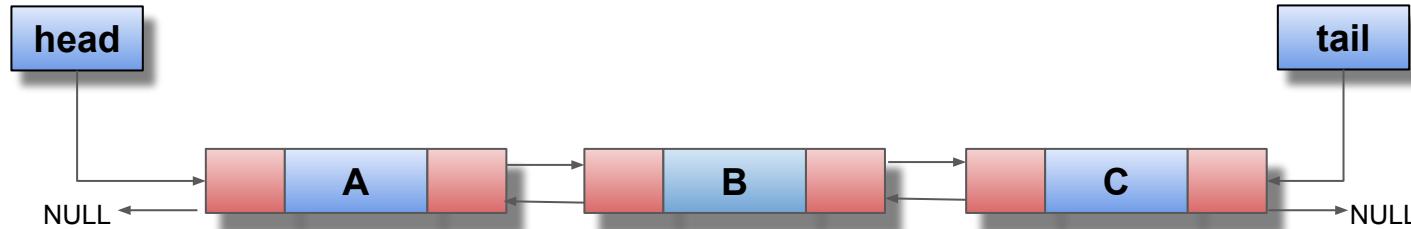
0	1	2

0	1	2
A	B	C

0	1
A	B

SplDoublyLinkedList (Lista Duplamente Encadeada)

- Classe genérica que representa a estrutura de uma fila duplamente encadeada.
- Permite a iteração bidirecional da lista e implementa as mesmas interfaces que o SplFixedArray.
- Ela também serve como a classe base para as classes **SplStack** e **SplQueue**, que também implementam as estruturas de dados de **pilha** e **fila**.
- Através do método `setIteratorMode`, o Lista encadeada pode ser organizada usando **FIFO** ou **LIFO**.
- Esta implementação não é circular, por isso, não existe uma referência entre os elementos das extremidades.



SplDoublyLinkedList (iteração, cabeça e cauda)

```
<?php
$dll = new \SplDoublyLinkedList();

$dll->push("laranja");
$dll->push("banana");
$dll->push("limão");
$dll->push("maçã");
$dll->push("uva");
$dll->push("abacaxi");

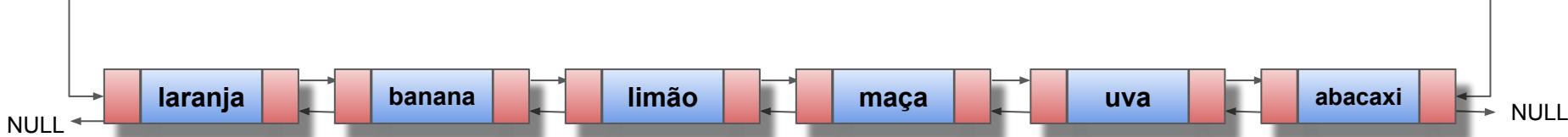
echo "Cabeça: ". $dll->bottom(). "<br/>";
echo "Cauda: ". $dll->top(). "<br/>";
```

head

```
$prev = null;
$dll->rewind(); //rebobinando

while ($dll->valid()) {
    $current = $dll->current();
    echo 'Anterior: '.$prev, "<br/>";
    echo 'Atual: '.$current, "<br/>";
    $prev = $current;
    $dll->next();
    $next = $dll->current();
    echo 'Próximo: '.$next. "<br/>";
    echo "<br/>";
}
```

tail



SplDoublyLinkedList (métodos)

```
<?php
$dll = new \SplDoublyLinkedList();
$dll->unshift(200); //no início
imprimir($dll);
$dll->unshift(100); //no início
imprimir($dll);
$dll->push(34); //no fim
imprimir($dll);
$dll->push(35); //no fim
imprimir($dll);
$dll->add(2, 3); //posição específica
imprimir($dll);
$dll->unshift(670); //no início
imprimir($dll);
$dll->add(3, 450); //posição específica
imprimir($dll);
$dll->pop(); //remove o último
imprimir($dll);
$dll->shift(); //remove o primeiro
imprimir($dll);
$dll->offsetUnset(1); //remove de posição específica (2a)
imprimir($dll);
```

```
function imprimir(\SplDoublyLinkedList &$dll) {

    $dll->rewind();
    $values = [];
    while ($dll->valid()) {
        $values[] = $dll->current();
        $dll->next();
    }
    echo "[" . implode(', ', $values) . "] <br>";
}
```



FIFO e LIFO

- FIFO: Comportamento First in, first out (primeiro a entrar, primeiro a sair) [Fila / Queue];
- Exemplos de FIFO:
 - Scheduling de CPU e Disco.
 - Fila da impressora: os trabalhos enviados para a impressora são impressos na ordem de chegada.
 - Transferindo dados de maneira assíncrona entre dois processos (E / S de Arquivo, Pipes, buffers de E / S, etc.)
 - App de gerenciamento fila em bilheteiras, hospitais etc.
- Comportamento Last in, first out (última a entrar, primeiro a sair). [Pilha / Stack]
- Exemplos de LIFO:
 - Mecanismo Desfazer / Refazer em editores de texto e outras aplicações (CTRL+Z);
 - Verificação de sintaxe do compilador para chaves correspondentes
 - Uma pilha de pratos ou livros ou cadeiras.
 - Backtracking: Este é um processo quando você precisa acessar o elemento de dados mais recente em uma série de elementos.
 - Inverter uma palavra

SplDoublyLinkedList (Modo FIFO)

```
<?php

//FIFO
$fifo = new \SplDoublyLinkedList();

fifo->setIteratorMode(\SplDoublyLinkedList::IT_MODE_FIFO);

$fifo->push("laranja");
$fifo->push("banana");
$fifo->push("limão");
$fifo->push("maçã");
$fifo->push("uva");
$fifo->push("abacaxi");

foreach ($fifo as $value) {
    echo $value."<br>";
}
```



SplDoublyLinkedList (Modo LIFO)

```
<?php

//FIFO
$fifo = new \SplDoublyLinkedList();

fifo->setIteratorMode(\SplDoublyLinkedList::IT_MODE_LIFO);

$fifo->push("laranja");
$fifo->push("banana");
$fifo->push("limão");
$fifo->push("maçã");
$fifo->push("uva");
$fifo->push("abacaxi");

foreach ($fifo as $value) {
    echo $value."<br>";
}
```

Top



Animation Completed

SplDoublyLinkedList (Modo Delete)

```
<?php
$dll = new \SplDoublyLinkedList();

$dll->setIteratorMode(\SplDoublyLinkedList::IT_MODE_DELETE);

$dll->push("laranja");
$dll->push("banana");
$dll->push("limão");
$dll->push("maçã");
$dll->push("uva");
$dll->push("abacaxi");

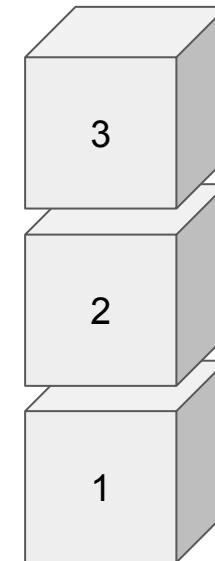
foreach ($dll as $value) {
    echo $value."<br/>";
    echo "Tamanho atual: ".$dll->count();
}
echo "Tamanho atual: ".$dll->count();
```

Com o modo de Iteração Delete, os item são removidos da lista duplamente encadeadas, conforme os mesmos são iterados.

O modo default é KEEP, que mantém os itens na lista.

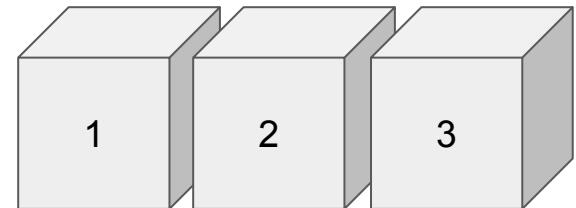
Pilha (SplStack) / LIFO

```
<?php  
  
$stack = new SplStack();  
  
$stack[] = 1;  
$stack[] = 2;  
$stack[] = 3;  
  
foreach ($stack as $item) {  
    echo $item, PHP_EOL;  
}
```



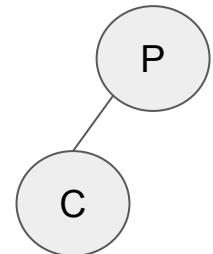
Fila (SplQueue) / FIFO

```
<?php  
  
$queue = new SplQueue();  
  
$queue[] = 1;  
$queue[] = 2;  
$queue[] = 3;  
  
foreach ($queue as $item) {  
    echo $item, PHP_EOL;  
}
```



SplHeap

- Heap é uma estrutura de dados baseada em **árvore especializada** que é essencialmente uma **árvore quase completa** que satisfaz a propriedade heap: *em um heap máximo, para qualquer nó dado C, se P é um nó pai de C, então a chave (o valor) de P é maior que ou igual à chave de C. Em uma pilha mínima (MinHeap), a chave de P é menor ou igual à chave de C.*
- O nó no "topo" do heap (sem pais) é chamado de nó raiz.
- O heap é uma implementação maximamente eficiente de um tipo de dados abstrato chamado de **fila de prioridade** e, na verdade, as filas de prioridade são geralmente chamadas de "heaps", independentemente de como elas podem ser implementadas. Em um heap, o elemento de prioridade mais alto (ou mais baixo) é sempre armazenado na raiz.
- No entanto, um heap não é uma estrutura classificada; pode ser considerado parcialmente ordenado.
- Um heap é uma estrutura de dados útil quando é necessário remover repetidamente o objeto/item com a prioridade mais alta (ou mais baixa).



Spl Heap (Exemplo)

```
<?php
class CampeonatoBrasileiro extends SplHeap {

    protected function compare($value1, $value2): int {

        if ($value1['pontuacao'] == $value2["pontuacao"]) {
            return $value1['vitorias'] <=> $value2['vitorias'];
        }
        return $value1['pontuacao'] <=> $value2['pontuacao'];
    }

}
```

```
$heap = new CampeonatoBrasileiro();
$heap->insert(['nome' => 'Remo', 'pontuacao' => 22, 'vitorias' => 6]);
$heap->insert(['nome' => 'Santa Cruz', 'pontuacao' => 28, 'vitorias' => 7]);
$heap->insert(['nome' => 'Atlético AC', 'pontuacao' => 30, 'vitorias' => 9]);
$heap->insert(['nome' => 'Botafogo PB', 'pontuacao' => 26, 'vitorias' => 6]);
$heap->insert(['nome' => 'Náutico', 'pontuacao' => 31, 'vitorias' => 9]);
$heap->insert(['nome' => 'Confiança', 'pontuacao' => 23, 'vitorias' => 5]);
$heap->insert(['nome' => 'Globo', 'pontuacao' => 22, 'vitorias' => 4]);

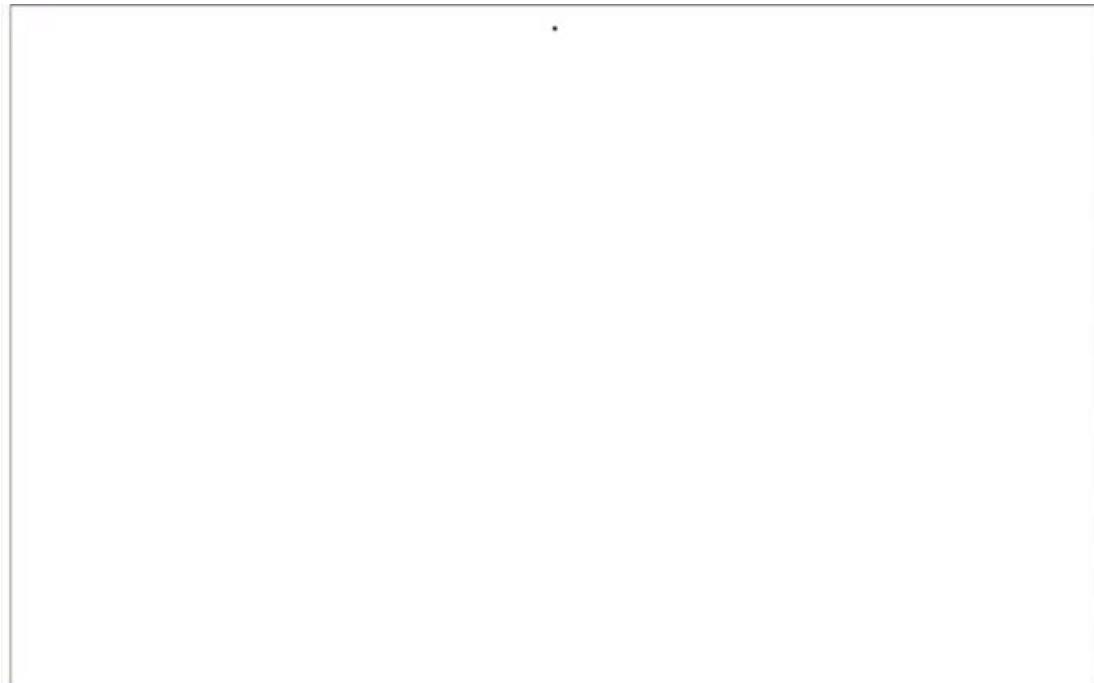
//Campeão:
echo "Campeão da Série C 2018: {$heap->top()['nome']} <br/>";
//Resultados
$total = $heap->count();
foreach ($heap as $key => $time) {
    echo ($total - $key) . " ° {$time['nome']} <br/>";
}
```

MinHeap (SplMinHeap)

```
<?php

$minHeap = new \SplMinHeap();
$minHeap->insert(30);
$minHeap->insert(50);
$minHeap->insert(10);
$minHeap->insert(105);
$minHeap->insert(99);
$minHeap->insert(88);

echo "mínimo:". $minHeap->top();
echo "<br/>";
foreach ($minHeap as $value) {
    echo $value."<br/>";
}
```

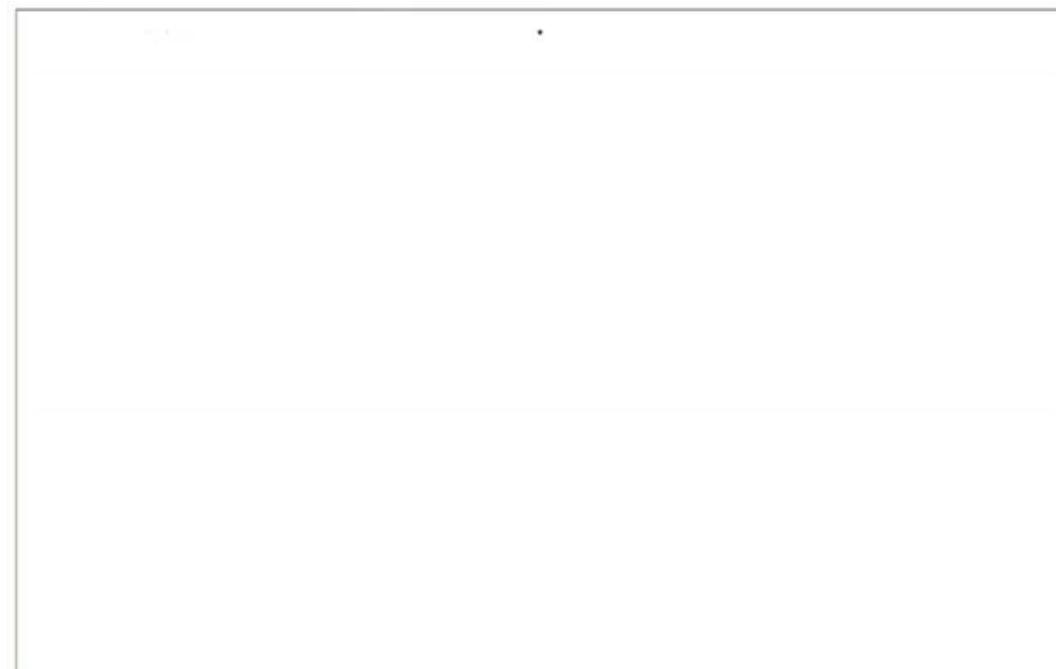


MaxHeap (SplMaxHeap)

```
<?php

$maxHeap = new \SplMaxHeap();
$maxHeap->insert(30);
$maxHeap->insert(50);
$maxHeap->insert(10);
$maxHeap->insert(105);
$maxHeap->insert(99);
$maxHeap->insert(88);

echo "max:". $maxHeap->top();
echo "<br/>";
foreach ($maxHeap as $value) {
    echo $value. "<br/>";
}
```

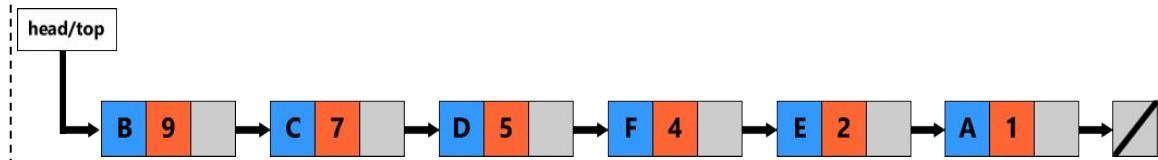


SplPriorityQueue (Fila de Prioridade)

- Fila de prioridade é uma extensão da fila com as seguintes características:
 - Cada item tem uma prioridade associada a ele.
 - Um elemento com alta prioridade é retirado da fila antes de um elemento com baixa prioridade.
 - Se dois elementos tiverem a mesma prioridade, eles serão atendidos de acordo com sua ordem na fila.
- Filas de prioridade são estruturas de dados úteis em simulações, particularmente para manter um conjunto de eventos futuros ordenados pelo tempo, para que possamos recuperar rapidamente o que acontecerá. Eles são chamados filas de prioridade porque permitem recuperar itens não pelo tempo de inserção (como em uma pilha ou fila), nem por uma correspondência de chave (como em um dicionário), mas por qual item tem a prioridade mais alta de recuperação.

SplPriorityQueue

```
<?php  
  
$prQueue = new SplPriorityQueue();  
  
$prQueue->insert('A',1);  
$prQueue->insert('B',9);  
$prQueue->insert('C',7);  
$prQueue->insert('D',5);  
$prQueue->insert('E',2);  
$prQueue->insert('F',4);  
  
foreach ($prQueue as $item) {  
    echo $item, <br/>;  
}
```



SplPriorityQueue (Flags)

```
<?php
//flags
$prQueue = new SplPriorityQueue();
$prQueue->insert('A', 1);
$prQueue->insert('B', 9);
$prQueue->insert('C', 7);
$prQueue->insert('D', 5);
$prQueue->insert('E', 2);
$prQueue->insert('F', 4);
$prQueue->insert('G', 5);

$prQueue->setExtractFlags(SplPriorityQueue::EXTR_PRIORITY);
foreach ($prQueue as $item) {
    echo $item."<br>";
}
```

```
<?php
//flags
$prQueue = new SplPriorityQueue();
$prQueue->insert('A', 1);
$prQueue->insert('B', 9);
$prQueue->insert('C', 7);
$prQueue->insert('D', 5);
$prQueue->insert('E', 2);
$prQueue->insert('F', 4);
$prQueue->insert('G', 5);

$prQueue->setExtractFlags(SplPriorityQueue::EXTR_BOTH);
foreach ($prQueue as $item) {
    echo "prioridade: {$item['priority']} ; dado:{$item['data']}<br/>";
}
```

Com o uso do método `setExtractFlags` é possível determinar qual será o resultado extraído ao recuperar os dados de um `SplPriorityQueue`. Seja com os dados (default), apenas o valor da priorização (priority) ou ambos (both).

SplStorageObject

- A classe SplObjectStorage fornece um mapa de objetos para dados ou, ignorando dados, um conjunto de objetos. Esse duplo propósito pode ser útil em muitos casos, envolvendo a necessidade de identificar objetos de forma exclusiva.
- Diferente de um hashmap, o SplObjectStorage não atua como um armazenamento de chave-valor, mas apenas um conjunto de objetos. Algo está no set ou não, mas sua posição não é importante.
- A "chave" de um elemento no SplObjectStorage é, na verdade, o hash do objeto. Isso faz com que não seja possível adicionar várias cópias da mesma instância de objeto a um SplObjectStorage, para que você não precise verificar se já existe uma cópia antes de adicionar.
- A principal vantagem do SplObjectStorage é o fato de que você ganha muitos métodos para lidar e interagir com diferentes conjuntos (contains(), removeAll(), removeAllExcept () etc).

SplObjectStorage

```
<?php

class Pessoa {
    public $nome;
}

$storage = new SplObjectStorage();

$o1 = new Pessoa();
$o2 = new Pessoa();
$o3 = new Pessoa();
$o4 = new Pessoa();

$o1->nome = 'João';
$o2->nome = 'Maria';
$o3->nome = 'Thiago';
$o4->nome = 'Maria';
```

```
$storage->attach($o1);
$storage->attach($o2);
$storage->attach($o3);
$storage->attach($o1); //já existe, mesmo hash
$storage->attach($o4); //estados iguais, objetos diferentes
$storage->detach($o3); //removendo

//var_dump($storage[0]); //nulo !
echo "contém o1? ".var_export($storage->contains($o1), true)."<br>";
foreach($storage as $key => $o){
    var_dump($o); //aqui funciona!
}
```

SPLEnum (Conjunto finito de identificadores)

```
<?php
class MesesDoAno extends SplEnum {

    const __default = self::Janeiro;
    const Janeiro = 1; const Fevereiro = 2;
    const Marco = 3; const Abril = 4;
    const Maio = 5; const Junho = 6;
    const Julho = 7; const Agosto = 8;
    const Setembro = 9; const Outubro = 10;
    const Novembro = 11; const Dezembro = 12;
}

echo new MesesDoAno(MesesDoAno::Junho) . "<br/>";
echo MesesDoAno::Novembro . "<br/>";
print_r((new MesesDoAno())->getConstList(true));

try {
    new MesesDoAno(13);
} catch (UnexpectedValueException $uve) {
    echo $uve->getMessage() . PHP_EOL;
}
```

Enum é um tipo de dado definido pelo usuário que consiste em um conjunto de constantes nomeadas chamadas enumeradores.

O SPLEnum faz parte de uma extensão PECL chamada SPLTypes. Infelizmente esta extensão não é mais compatível com PHP 7 >=, por isso, faz-se necessário utilizar algum fork (e compilar) ou um polyfill como por exemplo o [duck-projects](#).

Manipulação de String

- Na maioria das aplicações existe a necessidade de editar, manipular e/ou comparar textos;
- O PHP fornece muitas funções internas para manipular strings, como calcular o tamanho de uma string, encontrar substrings ou caracteres, substituir parte de uma string por caracteres diferentes, separar uma string e muitos outros;
- Além disso, o PHP também possui implementações nativas de algoritmos avançados de comparação/métrica de string como, por exemplo, o Levenshtein;

Caixa Baixa e Alta - (Manipulação de String)

```
<?php
$nome = "José da Silva";
echo "caixa baixa:".mb_strtolower($nome);
echo "caixa alta:".mb_strtoupper($nome);
echo "caixa alta 1º char :".ucfirst($nome);
echo "caixa baixa 1º char :".lcfirst($nome);
echo "caixa alta 1º char de cada palavra :".ucwords($nome);
```

O PHP possui um conjunto de funções que permite alterar as caixa (case) de uma string, seja em sua totalidade ou em partes dela.

Posição e Tamanho de string

```
<?php  
  
$nome = "José da Silva Silveira";  
echo "tamanho:".strlen($nome);  
echo "existe:".strstr($nome, "s");  
echo "existe:".stristr($nome, "silva");  
echo "encontrar:".strrpos($nome, "s");  
echo "encontrar:".stripos($nome, "O");  
  
$digitos = '0123456789';  
$telefone = '02199999999a';  
  
if (strlen($telefone) != strspn($telefone,$digitos)){  
    echo "telefone inválido";  
}  
}
```

strlen: retorna o tamanho de uma string

strrpos: encontra a posição da última ocorrência de um caractere em uma string

stripos: o mesmo do strrpos, porém, insensível à caixa.

strspn: encontra o comprimento do segmento inicial combinando com a máscara.

strrchr: Encontra a última ocorrência de um caractere em uma string

Substring

```
<?php  
  
$string = "85 bits";  
$frase = "o sabiá sabia assobiar";  
$path = '/www/public_html/index.html';  
  
echo substr($string, 1)."\n"; // 5 bits  
echo substr($string, 0, 2)."\n"; // 85  
echo substr($string, -4, 4). "\n"; // bits  
echo substr_count($frase, 'bi'); // 3  
$arquivo = substr($path, strpos($path, '/') + 1);  
echo $arquivo;
```

Uma substring é uma seqüência contígua de caracteres dentro de uma string. Por exemplo, "o melhor dos" é uma substring de "Foi o melhor dos tempos".

substr: Retorna uma parte de uma string

substr_count: Conta o número de ocorrências de uma substring

strpos: retorna a posição da ultima ocorrência de uma substring

Substituição de String

```
<?php

$noticias      = str_replace("aberto", "fechado", "Hoje o comércio está aberto !");
$vogais        = array("a", "e", "i", "o", "u");
$titulo         = str_replace($vogais, "", "Olá Seja Bem Vindo ao PHP !");
$conselho      = "Você deveria comer frutas, vegetais e fibra todos os dias.";
$saudavel       = array("frutas", "vegetais", "fibra");
$gorduroso     = array("pizza", "bacon", "sorvete");
$novo_conselho = str_replace($saudavel, $gorduroso, $conselho);
$var = 'ola chuva !';
echo substr_replace($var, 'guarda', 4) . "<br />\n";
echo substr_replace($var, 'guarda-', 4, 0) . "<br />\n";
```

Funções para Strings com HTML

```
<?php  
  
$original = "Eu <i>gosto</i> de estudar <b>PHP</b> !";  
$original = "Muito &lt;b&gt;bom!&lt;/b&gt;";  
  
echo htmlspecialchars($original);  
echo htmlspecialchars_decode($str);  
echo htmlspecialchars_decode($str, ENT_NOQUOTES);  
$a = htmlentities($original);  
$b = html_entity_decode($a);  
echo $a."<br>";  
echo $b;  
$str = '<p>this -&gt; "</p>';  
echo nl2br("Vamos quebrar\n linhas !");  
$html = '<p>Um parágrafo</p><!-- Comentário --> <a href="#">Link</a>';  
echo strip_tags($html) . "<br>";  
echo strip_tags($html, '<p><a>');
```

htmlentities: converte todos os caracteres aplicáveis em entidades html.

html_entity_decode: é o oposto da função `htmlentities()` no que converte todas as entidades HTML para os seus caracteres de string.

Formatação e limpeza

```
<?php  
  
$string = "Uma palavra enorme em português: inconstitucionalíssimamente";  
echo wordwrap($string,20,"<br>\n")."<br>\n";  
$string_com_espacos = " olá ";  
echo ltrim($string_com_espacos)."<br>\n";  
echo rtrim($string_com_espacos)."<br>\n";  
echo trim($string_com_espacos)."<br>\n";
```

wordwrap: permite “quebrar” string limitando o número de caracteres em grupos, comumente linhas, através de um separador.

ltrim: remove os espaços à esquerda

rtrim: remove os espaços à direita

trim: remove os espaços à esquerda e à direita

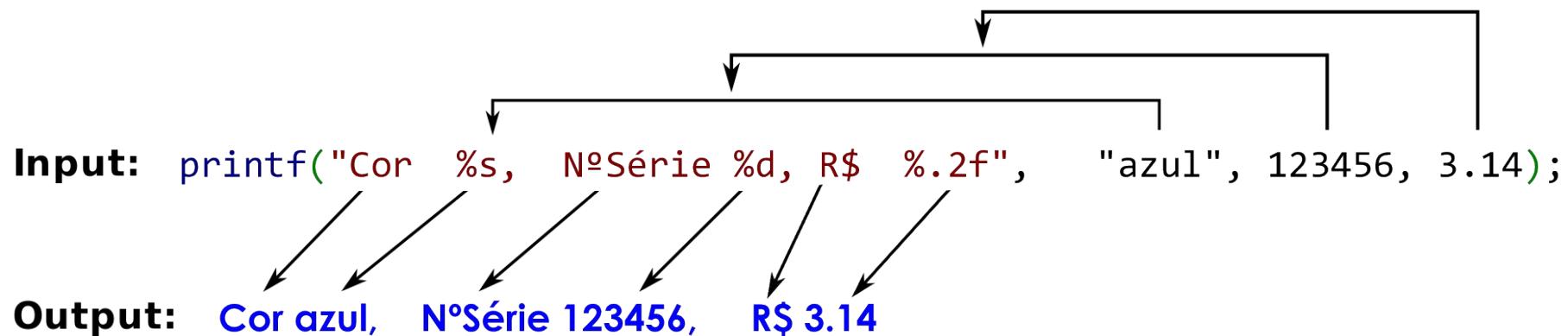
Formatação de String

- "Printf" é o nome de uma das principais funções de saída da linguagem C e significa "impressão formatada".
- O PHP, assim como outras linguagens de programação, também possui uma implementação do printf, mantendo uma compatibilidade razoável com printf do C.
- Essa função utiliza o conceito de um parâmetro de “formato”, que é composto de um conjunto de um ou mais especificadores que começam com um caractere “%”, indicando o local e o método para converter os dados de entrada no formato desejado de saída em uma string.
- Importante reforçar que incompatibilidades entre os especificadores de formato e o tipo de dados podem causar falhas e outras vulnerabilidades.

Formatação (printf)

especificador	Tratado como	Resultado
e	notação científica	notação científica em caixa baixa
E		notação científica em caixa alta
f	float	número de ponto flutuante (reconhecimento de localidade)
F		número de ponto flutuante (sem reconhecimento de localidade)
s	string	string
u	Inteiro	decimal sem sinal
x		hexadecimal (caixa baixa)
X		hexadecimal (caixa alta)
b		binário
c		caractere ASCII correspondente
d		decimal com sinal de positivo ou negativo
o		número octal

Printf (Exemplo)



printf (preenchimento)

```
<?php  
  
printf( "%04d", 12 );           // Resultado: "0012"  
printf( "%04d", 1234 );         // Resultado: "1234"  
printf( "%04d", 12345 );        // Resultado: "12345"  
printf( "% 10s", "Hello" );     // Resultado: "    Hello"  
printf( "%10s", "Hello" );      // Resultado: "    Hello"  
printf( "%'*10s", "Hello" );    // Resultado: "*****Hello"  
printf( "%'*-10s", "Hello" );   // Resultado: "Hello*****"
```

Troca de argumento

```
<?php  
//ordem natural:  
printf('Existe uma diferença entre o %s and %s', 'bem', 'mal');  
//informando posição usando % e a formatação com $:  
printf( 'O Brasil possui %2$d regiões e %1$d estados', 27, 5 );  
//retornando uma string (e não printando diretamente no output)  
$string = sprintf('%3$02d/%2$02d/%1$04d', 1998, 2, 1);  
print($string);
```

Comparação Avançada de Strings

```
<?php  
  
$nome1 = "tiago da silva";  
$nome2 = "thyago da silva";  
echo "distâncias: ".levenshtein($nome1, $nome2)."  
echo "comparação fonética 1:"  
    .soundex($nome1), " ", soundex($nome2)."  
echo "comparação fonética 2:"  
    .metaphone($nome1), " ", metaphone($nome2)."  
similar_text($nome1, $nome2, $porcentagem);  
echo "similaridade de textos: "  
    .$porcentagem." % <br/>";
```

levenshtein: calcula a “distância” entre duas strings. Esta distância é definida como o mínimo de caracteres (passos) necessários para transformar a string1 na string2. Sua complexidade é $O(m \cdot n)$;

soundex: calcula a “chave sonora” de uma string.

metaphone: similar ao soundex, porém mais preciso pois utiliza os princípios básicos da pronúncia de palavras em inglês.

similar_text: calcula a distância entre as duas strings, utilizando o algoritmo de Oliver. Sua complexidade é $O(n^3)$.

Expressões Regulares

- Comumente conhecidas como "regex" ou "RegExp", expressões regulares são sequências de caracteres especialmente formatadas para **encontrar padrões em textos**.
- Expressões regulares são uma das ferramentas mais poderosas disponíveis atualmente para processamento e manipulação de textos eficazes e eficientes.
- Podem ser usadas, por ex., para verificar se o formato dos dados, ou seja, nome, email, número de telefone etc. digitado pelo usuário estava correto ou não, localizar ou substituir a sequência correspondente no conteúdo de texto.
- Com o advento do PHP 5.3, a linguagem passou a dar suporte a expressões regulares no estilo Perl através de sua família preg_ de funções.

Regex no PHP

- O PHP possui um conjunto de funções que trabalham com expressões regulares, todas com o prefixo “preg”
- PREG é um mnemônico para Perl REGular expression.
- A sintaxe dos padrões usados nessas funções se parece muito com o Perl. A expressão deve estar entre os delimitadores, uma barra (/), por exemplo.
- O suporte a expressões regulares utilizando a sintaxe POSIX com funções de prefixo “ereg” foram depreciadas no PHP 7.

Regex (Delimitadores)

```
<?php
```

```
$regex_valido = "/a/";  
$regex_valido = "%a%";  
$regex_valido = "*a*";  
$regex_valido = "!a!";
```

Os delimitadores podem ser qualquer caractere ASCII não alfanumérico e sem espaço em branco, exceto a barra invertida (\) e o byte nulo. Se o caractere delimitador precisar ser usado na própria expressão, ele precisará ser escapado por uma barra invertida.

Delimitadores correspondentes ao estilo Perl (), {}, [] e <> também podem ser usados.

Funções de Regex no PHP

```
<?php
```

```
//busca
```

- `preg_match($pattern, $subject, $matches);`
- `preg_match_all($pattern, $subject, $matches);`
- `preg_grep($pattern, $input);`
- `preg_split($pattern, $subject, $limit, $flags);`

```
//substituição
```

- `preg_filter($pattern, $replacement, $subject);`
- `preg_replace_callback($pattern, $callback, $subject);`
- `preg_replace_callback_array($patterns_and_callbacks, $subject);`

```
//auxiliares
```

- `preg_last_error();`
- `preg_quote($str, $delimiter);`

Regex: preg_match

```
<?php

$regex = "/85/";
$subject = "Bem vindo ao 85 bits developer";
$matches = [];

if (preg_match($regex, $subject)) {
    echo "encontrado";
} else {
    echo "não encontrado";
}

//passando array por referência
preg_match($regex, $subject, $matches);
var_dump($matches);
```

preg_match: busca um padrão (\$pattern) específico em uma string (\$subject) e, quando o padrão é encontrado pela primeira vez, ele para de procurá-lo. Ele é capaz de preencher um array por referência (&\$matches), onde \$matches[0] conterá o texto que corresponde ao padrão completo, \$matches[1] terá o texto que correspondeu ao primeiro sub-padrão capturado entre parênteses, e assim por diante.

Regex: preg_match_all

```
<?php  
  
$regex = "/w+(ala)"/;  
$subject = "uma mala com uma bala jogada em uma cala";  
$matches = [];  
preg_match_all($regex, $subject, $matches);  
var_dump($matches);
```

preg_match_all: busca todas as correspondências em uma string e as produz em um array multidimensional (\$matches) ordenado de acordo com as sinalizações (\$flags).

Regex: preg_grep

```
<?php  
  
$regex = "/Silva/i";  
$array = [  
    'Maria da Silva', 'Luiz Oliveira',  
    'José da Silva', 'Thiago Silva'  
];  
var_dump(preg_grep($regex, $array));
```

preg_grep: retorna os itens do array que combinaram com o regex.

Regex: preg_split

```
<?php  
  
$regex = "/./";  
$string = "pudim.com.br";  
var_dump(preg_split($regex, $string));
```

preg_split: divide a string em substrings dentro de um array utilizando uma expressão regular.

Regex: preg_replace

```
<?php  
$regex      = '/[aeiou]/';  
$replacement = 'e';  
$string     = "O sapo não lava o pé, não lava porque não quer";  
var_dump(preg_replace($regex, $replacement, $string));
```

preg_replace: realiza uma pesquisa por uma expressão regular e a substitui.

Regex: preg_filter

```
<?php  
  
$regex = '/a/';  
$replacement = 'o';  
$subject = 'luiz';  
  
var_dump(preg_filter($regex, $replacement, $subject));  
var_dump(preg_replace($regex, $replacement, $subject));
```

preg_filter: é idêntico a preg_replace(), exceto que apenas retorna o \$subject (possivelmente transformado) onde houve uma correspondência.

Regex: preg_replace_callback

```
<?php

$string = '2014-02-22';
$regex = '/(\d{4})-(\d{2})-(\d{2})/';
$resultado = preg_replace_callback($regex, 'dataBr', $string);

function dataBr ($matches) {
    return $matches[3].'.'.$matches[2].'.'.$matches[1];
}

echo $resultado;
```

O comportamento desta função é similar ao da `preg_replace()`, exceto pelo fato que ao invés do parâmetro `$replacement`, é utilizado uma função ou closure com a responsabilidade de executar a transformação/substituição.

Regex: preg_quote

```
<?php  
  
$regex = 'etc... / legal';  
$regex = preg_quote($regex, '/');  
echo $regex;
```

Regex: preg_last_error()

PREG_PATTERN_ORDER: 1

PREG_SET_ORDER: 2

PREG_OFFSET_CAPTURE: 256

PREG_SPLIT_NO_EMPTY: 1

PREG_SPLIT_DELIM_CAPTURE: 2

PREG_SPLIT_OFFSET_CAPTURE: 4

PREG_NO_ERROR: 0

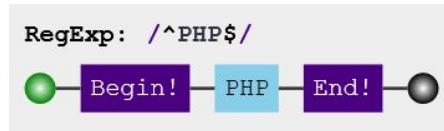
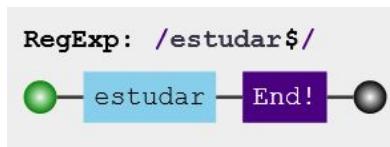
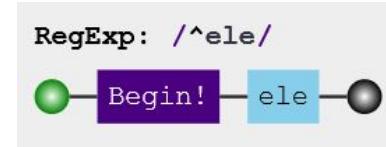
PREG_INTERNAL_ERROR: 1

PCRE

- O motor de expressões regulares do PHP é a implementação da biblioteca PCRE (Perl Compatible Regular Expressions).
- A biblioteca PCRE possui um conjunto de recursos que implementam a correspondência de padrões de expressão regular usando uma mesma sintaxe e semântica que o Perl 5, com apenas algumas poucas diferenças.
- Este conjunto de recursos é em sua vasta maioria representado por caracteres especiais (*tokens*) que, em caso de necessidade de utilizá-los como parte integrante da string de busca, as mesmas deverão ser “escapadas”.

Regex (âncoras)

```
<?php  
//^ começa com...  
$subject = "eles gostam de PHP";  
$pattern = "/^ele/"; //começa com  
var_dump(preg_match($pattern, $subject));  
  
//$ termina com...  
$subject = "Eu gosto de estudar";  
$pattern = "/estudar$/";  
var_dump(preg_match($pattern, $subject));  
  
//começa e termina com...  
$subject = "PHP";  
$pattern = "/^PHP$/";  
var_dump(preg_match($pattern, $subject));
```

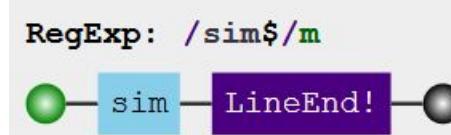


Âncoras ou asserções atômicas de largura zero, especificam uma posição na cadeia de caracteres em que uma correspondência deve ocorrer.

Quando utilizamos âncora em uma expressão de pesquisa, o mecanismo de expressões regulares não avança pela cadeia de caracteres ou consome caracteres, ele procura uma correspondência apenas na posição especificada.

Regex (modificador multi linha)

```
<?php  
//multi linha  
$regex = '/^ele/m'; $matches = [];  
$string = "eles disseram:\nnele gosta sim !";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);  
  
//apenas inicio da string (mesmo multi linha)  
$regex = '/^Aele/m'; $matches = [];  
$string = "eles disseram:\nnele gosta sim !";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);  
  
//termina com (multi linha)  
$regex = '/sim$/m'; $matches = [];  
$string = "gosto sim\n gosto disso sim";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);
```



Com o uso o modificador m (após os delimitadores da regex), é possível usar âncoras e outros tokens mesmo quando existir quebra de linhas na string.

Regex (modificador multi linha)

```
//z string termina com (mesmo multi linha)
$regex = '/sim\nz/m';
$string = "gosto sim\n gosto disso sim";
$matches = [];
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

Com o uso do modificador m (após os delimitadores da regex), é possível usar âncoras e outros tokens mesmo quando existir quebra de linhas na string.

Regex (grupos de captura/ subpattern)

```
<?php

//sem grupos
$string = 'comi muita uva';
preg_match('/uva/', $string, $matches);
var_dump($matches);

//com grupos
preg_match('/(u)(v)(a)/', $string, $matches);
var_dump($matches);

//grupo "nomeado"
$string = 'havia 2 homens e 4 crianças no local';
$regex = '/(?P<numeros>\d)/';
preg_match_all($regex, $string, $matches);
var_dump($matches['numeros']);
```

Ao colocar parte de uma expressão regular entre parênteses, é possível agrupar essa parte da expressão regular. Isso permite aplicar um quantificador a todo o grupo ou restringir a alternância a parte da regex.

Também é possível nomear/identificar um grupo através do uso do **?P<nome_do_grupo>**.

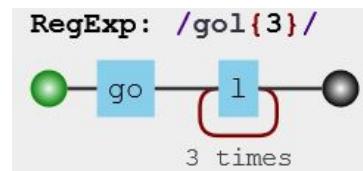
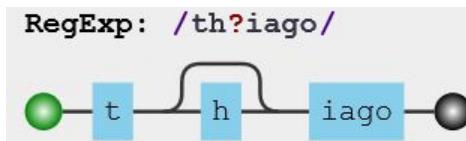
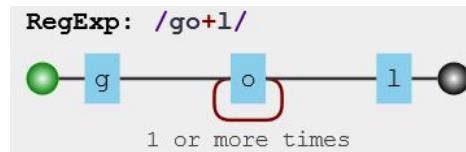
Regex (quantificadores)

```
<?php  
  
/* zero ou mais do caracter à esquerda de *  
$subject = "goal"; //ou gol  
$pattern = "/goa*l/";  
var_dump(preg_match($pattern, $subject));
```

```
// + um ou mais do caracter à esquerda de +  
$subject = "goooooool";  
$pattern = "/go+l/";  
var_dump(preg_match($pattern, $subject));
```

```
// ? zero ou um à esquerda de ?  
$subject = "thiago"; // ou tiago  
$pattern = "/th?iago/";  
var_dump(preg_match($pattern, $subject));
```

```
/* {n} N repetições do caracter à esquerda  
$subject = "gollll";  
$pattern = "/gol{3}/";  
var_dump(preg_match($pattern, $subject));
```



Regex (quantificadores)

```
// {nx, ny} mínimo nx e máximo ny repetições do caracter à esquerda
$subject = "goll";
$pattern = "/gol{1,3}/"; //começa com
var_dump(preg_match($pattern, $subject));

// {nx, } mínimo nx de repetições do caracter à esquerda
$subject = "gol|||||||||||||||";
$pattern = "/gol{1,}/"; //começa com
var_dump(preg_match($pattern, $subject));

// * zero ou mais repetições da string dentro dos parênteses
$subject = "muahahahahaha";
$pattern = "/mu(ha)*/"; //começa com
var_dump(preg_match($pattern, $subject));

// * zero ou mais repetições da string dentro dos parênteses
$subject = "muahahaha";
$pattern = "/mu(ha){1,3}$/";
var_dump(preg_match($pattern, $subject));
```

Greedy vs Lazy

```
<?php  
  
$html = '<b>Eu sou bold</b> <i>Eu sou itálico</i> <b>Eu também sou bold</b>';  
  
//greedy  
preg_match_all('#<b>(.)</b>#', $html, $bolds);  
print_r($bolds[1]);  
  
//lazy  
preg_match_all('#<b>(.*?)</b>#', $html, $bolds);  
print_r($bolds[1]);  
  
//lazy com modificador U  
preg_match_all('#<b>(.)</b>#U', $html, $bolds);  
print_r($bolds[1]);
```

Um quantificador greedy (ganancioso) tenta corresponder a um elemento tantas vezes quanto possível.

Um quantificador não greedy/ Lazy (lento) tenta corresponder a um elemento o menor número de vezes possível.

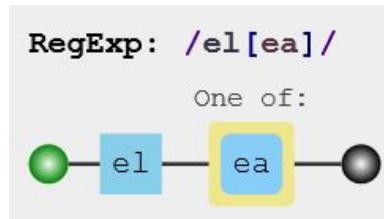
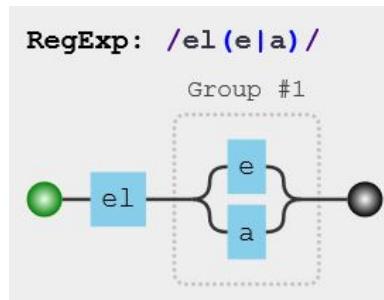
É possível transformar um quantificador greedy em um quantificador lazy simplesmente adicionando um token ? ou utilizar o modificador **U** depois do delimitador final.

A busca lazy é bastante útil ao tentar executar uma análise simplista de HTML.

Regex (Operador “ou”)

```
<?php  
  
// | seguido pela string a ou b  
$string = "ele";  
$regex = "/el(e|a)/"; //começa com  
var_dump(preg_match($pattern, $string));
```

```
// mesmo que o anterior porém sem grupo  
$string = "ela";  
$regex = "/el[ea]/";  
var_dump(preg_match($regex, $string));
```



O operador “ou” permite criar a alternância de matches.

Com uso do operador pipe é possível alternar entre caracteres dentro de um grupo.

Sem o uso do pipe, basta colocar os caracteres de forma contígua dentro de brackets

Regex (classe de caractere)

```
<?php
// \d dígito (mesmo que [0-9])
$subject = "tem numero 1 aqui";
$pattern = "/\d/"; //começa com
var_dump(preg_match($pattern, $subject));

// \w caractere alfanumérico + underscore (mesmo que [a-zA-Z0-9_])
$subject = "número ou letra ou underscore 1_";
$pattern = "/\w/"; //começa com
var_dump(preg_match($pattern, $subject));

// \s caractere de espaço (incluindo tab e quebra de linha)
$subject = "tem espaço aqui";
$pattern = "/\s/"; //começa com
var_dump(preg_match($pattern, $subject));

// . qualquer caractere
$subject = "oi1 _";
$pattern = "/./"; //começa com
var_dump(preg_match($pattern, $subject));
```

Regex (classe de caractere [negação])

```
//--- Negação ---  
  
// \D precisa ter um não dígito  
$subject = "precisa ter um 'não' dígito aqui 123";  
$pattern = "/\D/"; //começa com  
var_dump(preg_match($pattern, $subject));  
  
// \W precisa ter um não alfa  
$subject = "precisa ter um 'não' alfa aqui 123";  
$pattern = "/\W/"; //começa com  
var_dump(preg_match($pattern, $subject));  
  
// \S precisa ter um não espaço  
$subject = "precisa ter um 'não' espaço aqui";  
$pattern = "/\S/"; //começa com  
var_dump(preg_match($pattern, $subject));  
  
// \W precisa ter um não espaço  
$subject = "precisa ter um 'não' espaço aqui";  
$pattern = "/\S/"; //começa com  
var_dump(preg_match($pattern, $subject));
```

Regex (ranges “-”)

```
<?php  
//range para verificar se contém números 0 à 9 (todos os números)  
$string = "3 quartos, 2 banheiros, 4 tvs e 18 copos";  
$regex = "/[0-9]/";  
var_dump(preg_match($regex, $string));
```

```
//range para verificar se contém números de 0 à 3  
$regex = "/[0-3]/";  
var_dump(preg_match($regex, $string));
```

```
//range para verificar se contém letras (caixa baixa e não acentuadas)  
$regex = "[a-z]"/";  
var_dump(preg_match($regex, $string));
```

```
//range para verificar se contém letras (caixa alta e não acentuadas)  
$regex = "[A-Z]"/";  
var_dump(preg_match($regex, $string));
```

```
//range para letras acentuadas independente da caixa  
$string = "Último açaí às 18 horas";  
$regex = "/[À-ú]/u";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);
```

Regex Unicode

```
<?php
//categorias em https://www.fileformat.info/info/unicode/category/index.htm
//Símbolo de moeda
$string = "Sua dívida era de R$ 40,78 e £ 200,00";
$regex = "/\p{Sc}/u";
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

```
//Símbolo matemático
$string = "solidão + auto-estima ÷ zombaria ÷ condenação × culpa ...";
$regex = "/\p{Sm}/u";
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

```
//Símbolos de pontuação de abertura (Ps) e fechamento (Pe)
$string = 'tem coisas (muitas) e outras também [1] e assim {bom demais}';
$regex = "/\p{Ps}(.*?)\p{Pe}/u";
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

```
//Símbolo de pontuação "outros"
$string = "Ele gritou: – Você fez isso ?!?";
$regex = "/\p{Po}/u";
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

Regex \b (word bondaury)

```
<?php  
  
$regex = '/\bflor\b/';  
$string = "uma flor na janela";  
$string2 = "hoje visitei uma floricultura";  
var_dump(preg_match($regex, $string)); //1  
var_dump(preg_match($regex, $string2)); //0
```

Asserções

Uma asserção é um teste para os caracteres que seguem ou precedem o ponto de correspondência atual que realmente não consome nenhum caractere.

Asserções avançadas são codificadas como subpadrões. Existem dois tipos: aqueles que olham à frente da posição atual na string e aqueles que olham atrás dela. Um subpadrão de asserção é correspondido da maneira normal, exceto que não faz com que a posição de correspondência atual seja alterada.

Os sub-padrões de asserção não estão capturando sub-padrões e podem não ser repetidos, porque não faz sentido afirmar a mesma coisa várias vezes. Se qualquer tipo de asserção contiver subpadrões de captura dentro dele, eles serão contados com a finalidade de numerar os subpadrões de captura em todo o padrão. No entanto, a captura de substring é realizada apenas para afirmações positivas, porque não faz sentido para afirmações negativas.

Regex Asserção lookahead

```
<?php  
  
$string = "achei quebra; tem aqui;";  
  
//(?) positive lookahead  
$regex = "/\w+(?=;)/";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);  
  
//(?! negative lookahead /  
$regex = "\b\w+\b(?!;)"/";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);
```

Regex Asserção lookbehind

```
//(?) - positive lookbehind
```

```
$string = "dra Julia \ndr Marcos \ndr Mateus \ndra Ana";
$regex = "/(?(?=dra\s)(w+)/im";
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

```
//(?) - negative lookbehind
```

```
$regex = "/(?(?!^dra\h)\b\w+\b\h*/im";
preg_match_all($regex, $string, $matches);
var_dump($matches);
```

Regex Recursividade

```
<?php  
  
$string = "kkk!!!";  
$regex = "/k(?R)?!/";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);
```

Regex Recursividade

palíndromos:

```
^((\w((((\w)(?5)\5?)*)|(?1)|\w?))\2)$
```

com três ou mais consoantes seguidas:

```
\b\w*?([b-df-hj-np-tv-z])\{3,\}\w*?\b
```

Conditional Subpattern

```
<?php
```

```
$regex = "/(fato)??(?(1) é verdadeiro| é falso)/";  
$string = "este fato é verdadeiro";  
preg_match_all($regex, $string, $matches);  
var_dump($matches);
```

Se o grupo de captura 1 foi correspondido até agora, corresponde ao padrão antes da barra vertical.

Caso contrário, corresponde ao padrão após a barra vertical.

Biblioteca GD (Manipulação de Imagem)

- Nas aplicações modernas, muitas vezes, faz-se necessário manipular imagens ou até criar novas em tempo de execução.
- Para esta tarefa podemos utilizar uma extensão do php, comumente presente em suas distribuições chamada de PHP GD.
- GD (Graphical Draw) Graphics é uma biblioteca de software gráfico criada por Thomas Boutell para manipulação dinâmica de imagens. É nativamente escrito em ANSI C, mas possui interfaces para muitas outras linguagens de programação.



Instalação e Verificação

- No Windows, faz-se necessário habilitar a DLL do GD2 php_gd2.dll no php.ini (Esta DLL já é disponibilizada no download windows oficial)
 - Bundles como Wamp e Xamp já incluem o php_gd2.dll por default
- No linux, dependendo da distribuição (e da versão do PHP) a instalação poderá ser feita via gerenciadores de pacotes como:
 - apt-get install php7.3-gd
 - yum install php-gd
- Independente do S.O., sempre será necessário reiniciar o Apache após a instalação/habilitação do php_gd.
- Após os procedimentos acima, basta executar o `php_info()` e verificar a inclusão da extensão GD no mesmo.

GD: criação de imagem e preenchimento de fundo

```
<?php

// cria uma imagem de 800x600 pixels
$image = imagecreatetruecolor(800, 600);

// preenche com fundo branco (R,G,B)
$branco = imagecolorallocate($image, 255, 255, 255);
imagefill($image, 0, 0, $branco);

// envia a imagem
header("Content-type: image/png");
imagepng($image);
imagedestroy($image);
```

GD: Tipos de formato para geração de imagens

gif, jpeg e png

wbmp foi depreciado

GD Texto (imagestring)

```
<?php

// cria uma imagem de 800*600
$image = imagecreatetruecolor(800, 600);

// fundo branco e texto azul
$branco = imagecolorallocate($image, 255, 255, 255);
$azul = imagecolorallocate($image, 0, 0, 255);
imagefill($image, 0, 0, $branco);

// escreve a string em cima na esquerda
// por default, sem font específica, o tamanho é de 1 à 5
imagestring($image, 5, 0, 0, "85 Bits", $azul);

// envia a imagem
header("Content-type: image/png");
imagepng($image);
imagedestroy($image);
```

GD Texto Vertical (imagecharup)

```
<?php

$string = '85 Bits';
$font_size = 5;
$img = imagecreate(20,90);
$bg = imagecolorallocate($img,225,225,225);
$preto = imagecolorallocate($img,0,0,0);

$len = strlen($string);
for ($i=1; $i<=$len; $i++) {
    imagecharup($img, $font_size, 5,
imagesy($img)-($i*imagefontwidth($font_size)), $string, $preto);
    $string = substr($string,1);
}
header('Content-type: image/png');
imagepng($img);
imagedestroy($img);
```

GD Texto com fonte (imagettfttext)

```
<?php
//esse path só funciona com windows :(
$font_path = getenv('WINDIR') . DIRECTORY_SEPARATOR . "Fonts" .
DIRECTORY_SEPARATOR;
$font = 'arial.ttf';
// Criando a imagem
$image = imagecreatetruecolor(800, 600);

// fundo branco e texto azul
$branco = imagecolorallocate($image, 255, 255, 255);
$azul = imagecolorallocate($image, 0, 0, 255);
imagefill($image, 0, 0, $branco);

// escreve a string em cima na esquerda
imagettfttext($image, 50, 0, 100, 200, $azul, $font_path.$font, "85 bits");

// envia a imagem
header("Content-type: image/png");
imagepng($image);
imagedestroy($image);
```

GD Texto com fonte (ângulos)

```
<?php

//esse path só funciona com windows :
$font_path = getenv('WINDIR') . DIRECTORY_SEPARATOR . "Fonts" .
DIRECTORY_SEPARATOR;
$font = 'arial.ttf';
// Create the image
$image = imagecreatetruecolor(800, 600);
// fundo branco e texto azul
$branco = imagecolorallocate($image, 255, 255, 255);
$azul = imagecolorallocate($image, 0, 0, 255);
imagefill($image, 0, 0, $branco);
// escreve a string em cima na esquerda em 90º
imagettfttext($image, 50, 90, 100, 200, $azul, $font_path.$font,
"85 bits");
// envia a imagem
header("Content-type: image/png");
imagepng($image);
```

GD retângulo

```
<?php
$image = imagecreatetruecolor(800, 600);
$branco = imagecolorallocate($image, 255, 255, 255);
$verde = imagecolorallocate($image, 0, 153, 0);

//retangulo ou quadrado sem preenchimento
imagerectangle($image, 20, 100, 400, 400, $branco);
//retangulo ou quadrado com preenchimento
imagefilledrectangle($image, 410, 100, 800, 400, $verde);

header('Content-Type: image/png');
imagepng($image);
imagedestroy($image);
```

GD Círculo

```
<?php

$image = imagecreatetruecolor(800, 600);
$branco = imagecolorallocate($image, 255, 255, 255);
$verde = imagecolorallocate($image, 0, 153, 0);

imageellipse($image, 200, 150, 300, 300, $branco);

imagefilledellipse($image, 510, 150, 300, 300, $verde);

// desenhando a imagem
header("Content-type: image/png");
imagepng($image);
imagedestroy($image);
```

GD Linha

```
<?php  
  
$image = imagecreate(800, 600);  
imagecolorallocate($image, 15, 142, 210);  
$preto = imagecolorallocate($image, 0, 0, 0);  
imagesetthickness($image, 5);  
imageline($image, 0, 0, 800, 600, $preto);  
  
header('Content-type: image/png');  
imagepng($image);  
imagedestroy($image);
```

GD Linha tracejada

```
<?php

$image = imagecreatetruecolor(150, 150);
$branco = imagecolorallocate($image, 255, 255, 255);
//espessura da linha
imagesetthickness($image, 5);
imagedashedline($image, 150, 0, 0, 150, $branco);

header('Content-type: image/png');
imagepng($image);
imagedestroy($image);
```

GD Polígono

```
<?php

$image = imagecreatetruecolor(800, 600);
$branco = imagecolorallocate($image, 255, 255, 255);

// Polígono sem preenchimento
imagepolygon($image, [
    0, 0,
    100, 200,
    300, 200
],
3,
$branco);

header('Content-type: image/png');
imagepng($image);
imagedestroy($image);
```

GD Polígono aberto

```
<?php

$image = imagecreatetruecolor(800, 600);
$branco = imagecolorallocate($image, 255, 255, 255);

imageopenpolygon($image, [
    0, 0,
    100, 200,
    300, 200
], 3, $branco);

header('Content-type: image/png');
imagepng($image);
imagedestroy($image);
```

GD (imagefilledarc)

```
<?php
$image = imagecreatetruecolor(800, 600);
$branco = imagecolorallocate($image, 255, 255, 255);

//tipos de arcos
imagefilledarc($image, 50, 50, 300, 300, 0, 45, $branco, IMG_ARC_ROUNDED);
imagefilledarc($image, 310, 50, 300, 300, 0, 45, $branco, IMG_ARC_CHORD);
imagefilledarc($image, 50, 210, 300, 300, 0, 45, $branco, IMG_ARC_EDGED);
imagefilledarc($image, 310, 210, 300, 300, 0, 45, $branco, IMG_ARC_NOFILL);
imagefilledarc($image, 510, 210, 300, 300, 0, 45, $branco, IMG_ARC_PIE);

header('Content-type: image/png');
imagepng($image);
imagedestroy($image);
```

GD gráfico de barras

```
<?php

$largura = 800;
$saltura = 600;

$font_path = getenv("WINDIR") . DIRECTORY_SEPARATOR . "Fonts" .
DIRECTORY_SEPARATOR;
$font = 'arial.ttf';

$dados = ['jan' => 30, 'fev' => 40, 'mar' => 90, 'abr' => 77];

$colunas = count($dados);
$padding = ($largura + $saltura) / 100;

$largura_colunas = $largura / $colunas;

$image = imagecreate($largura, $saltura);
$cinza = imagecolorallocate($image, 0xcc, 0xcc, 0xcc);
$preto = imagecolorallocate($image, 0, 0, 0);
$cinza_claro = imagecolorallocate($image, 0xee, 0xee, 0xee);
$cinza_escuro = imagecolorallocate($image, 0x7f, 0x7f, 0x7f);
$branco = imagecolorallocate($image, 0xff, 0xff, 0xff);
```

GD Gráfico de barras (parte 2)

```
imagefilledrectangle($image, 0, 0, $largura, $altura, $branco);
$maxv = max($dados);

$array_values = array_values($dados);
$array_keys = array_keys($dados);
for ($i = 0; $i < $colunas; $i++) {

    $coluna_altura = ($altura / 100) * (( $array_values[$i] / $maxv) * 100);
    $string = $array_keys[$i];

    $x1 = $i * $largura_colunas;
    $y1 = $altura - $coluna_altura;
    $x2 = (( $i + 1 ) * $largura_colunas) - $padding;
    $y2 = $altura - ($padding * 4);
    $stamanho_fonte = ($altura - $y2) / 2.5;
    $maxChars = ($stamanho_fonte * 2) / $padding;

    if (strlen($string) > ($maxChars)) {
        $string = substr($string, 0, $maxChars) . "...";
    }
}
```

```
imagefilledrectangle($image, $x1, $y1, $x2, $y2,
$cinza);
imagettfttext($image, $stamanho_fonte, 0, $x1 + 2, $y2 +
$stamanho_fonte + $padding, $preto, $font_path . $font,
$string);
//efeito de sombra
imageline($image, $x1, $y1, $x1, $y2, $cinza_claro);
imageline($image, $x1, $y2, $x2, $y2, $cinza_claro);
imageline($image, $x2, $y1, $x2, $y2, $cinza_escuro);
}

header("Content-type: image/png");
imagepng($image);
imagedestroy($image);
```

GD captcha

```
<?php
$largura = 142;
$altura = 40;
$max_chars = 6;
$padding = ($largura / $altura);
$image = imagecreatetruecolor($largura, $altura);
$espaco_por_char = $largura / $max_chars;

$background = imagecolorallocate($image, 0x66, 0xCC, 0xFF);
imagefill($image, 0, 0, $background);
$azul_escuro = imagecolorallocate($image, 0x33, 0x99, 0xCC);
$preto = imagecolorallocate($image, 0, 0, 0);
$branco = imagecolorallocate($image, 255, 255, 255);

$random_char = function () {
    return chr(rand(65, 90));
};

$path = realpath('') . DIRECTORY_SEPARATOR . 'fonts' .
DIRECTORY_SEPARATOR;
$fonts = array_diff(scandir($path), array('.', '..'));
$captcha = "";
```

```
$tamanho_fonte = floor((40 / 100) * $altura);
$i = 0;

foreach (range(0, $largura, $espaco_por_char) as $x) {
    if ($x < $largura) {
        $char = $random_char();
        $captcha .= $char;
        $angulo = rand(0, 45);
        imagettftext($image, $tamanho_fonte, $angulo,
                     $x + ($tamanho_fonte / 2), ($tamanho_fonte * rand(1.5, 2)) +
$padding,
                     rand(0, 1) ? $preto : $branco,
                     $path . $fonts[array_rand($fonts)], $char);
    }
}

for ($i = 0; $i < 3; $i++) {
    imagesetthickness($image, rand(1, 2));
    imageline($image, rand(0, $largura), 0, rand(0, $largura), $altura,
             $azul_escuro);
}
session_start();
$_SESSION['captcha'] = $captcha;
header('Content-type: image/png');
imagepng($image);
imagedestroy($image);
```

Manipulação de imagens

- Além da possibilidade de criar imagens, o php-gd também permite manipular imagens pré-existentes.
- Para recuperar dinamicamente informações sobre os arquivos de imagens, o PHP possui um API EXIF que permite extrair um conjunto significativo de metadados.
- Com uma imagem carrega como resource, é possível rotacionar, redimensionar, cortar e muito mais.
- O GD também conta com recursos de filtros que permitem aplicar uma vasta gama de efeitos em imagens.

Exif

- *Exchangeable image file format (Exif) é uma especificação seguida por fabricantes de câmeras digitais que gravam informações sobre as condições técnicas de captura da imagem junto ao arquivo da imagem propriamente dita na forma de metadados etiquetados. A especificação usa os formatos de imagem JPEG, TIFF rev.6.0 e o formato de áudio wave RIFF. O Exif não está suportado nos formatos JPEG 2000, PNG, GIF e BMP.* (WIKIPEDIA, 2019).
- O PHP possui um conjunto de funções que permitem recuperar os metadados EXIFs em arquivos de imagens.

Exif (metadados)

```
<?php  
  
$metadata = exif_read_data("levi.jpg");  
var_dump($metadata);
```

Tipo da imagem (exif_imagetype)

```
<?php  
  
$fileType = exif_imagetype("levi_bubble_waffle.jpg");  
  
header("Content-type: ".image_type_to_mime_type($fileType));  
  
echo file_get_contents("levi_bubble_waffle.jpg");
```

Thumbnail (exif)

```
<?php  
  
$filename = "levi_bubble_waffle.jpg";  
$fileType = exif_imagetype($filename);  
$thumbnail = exif_thumbnail($filename);  
if($thumbnail != false){  
    header("Content-type: ".  
        image_type_to_mime_type($fileType));  
    echo $thumbnail;  
}else{  
    echo "Sem thumbnail disponível";  
}
```



Imagecreatefrom

GD (Rotação de imagem)

```
<?php
$nome_arquivo = 'levi.jpg';
$grau_rotacao = 90;
$fileType = exif_imagetype($nome_arquivo);
header("Content-type: ".image_type_to_mime_type($fileType));
switch ($fileType) {
    case IMAGETYPE_PNG:
        $image_source = imagecreatefrompng($nome_arquivo);
        $fundo_alpha = imagecolorallocatealpha($image_source, 255, 255, 255, 127);
        $image_rotate = imagerotate($image_source, $grau_rotacao, $fundo_alpha);
        imagesavealpha($image_rotate, true);
        imagepng($image_rotate);
        break;
    default:
        $image_source = imagecreatefromjpeg($nome_arquivo);
        $image_rotate = imagerotate($image_source, $grau_rotacao, 0);
        imagejpeg($image_rotate);
        break;
}
imagedestroy($image_source);
imagedestroy($image_rotate);
```

GD (Virar imagem)

```
<?php
$nome_arquivo = 'logo.png';
$flip_mode = IMG_FLIP_HORIZONTAL;
$fileType = exif_imagetype($nome_arquivo);
header("Content-type: ".image_type_to_mime_type($fileType));

switch ($fileType) {
    case IMAGETYPE_PNG:
        $image = imagecreatefrompng($nome_arquivo);
        $fundo_alpha = imagecolorallocatealpha($image, 255, 255, 255, 127);
        imageflip($image, $flip_mode);
        imagesavealpha($image, true);
        imagepng($image);
        break;
    case IMAGETYPE_JPEG:
        $image = imagecreatefromjpeg($nome_arquivo);
        imageflip($image, $flip_mode);
        imagejpeg($image);
        break;
}
imagedestroy($image);
```

GD (Recorte de Imagem)

```
<?php
$nome_arquivo = 'levi.jpg';
$corte_rect = ['x' => 80, 'y' => 100,
               'width' => 220, 'height' => 300];
$fileType = exif_imagetype($nome_arquivo);
header("Content-type: ".image_type_to_mime_type($fileType));

switch ($fileType) {
    case IMAGETYPE_PNG:
        $image_source = imagecreatefrompng($nome_arquivo);
        $output = function($image_cropped){imagepng($image_cropped);};
        break;

    case IMAGETYPE_JPEG:
        $image_source = imagecreatefromjpeg($nome_arquivo);
        $output = function($image_cropped){imagejpeg($image_cropped);};
        break;
}
$image_cropped = imagecrop($image_source, $corte_rect);
$output($image_cropped);
imagedestroy($image_source);
```

GD (Ampliação e Redução de imagem)

```
<?php
$nome_arquivo = 'levi_bubble_waffle.jpg';
$nova_altura = 80; $nova_largura = 80;
$fileType = exif_imagetype($nome_arquivo);
header("Content-type: ".image_type_to_mime_type($fileType));
switch ($fileType) {
    case IMAGETYPE_PNG:
        $image = imagecreatefrompng($nome_arquivo);
        $image_resized = imagescale($image, $nova_largura, $nova_altura);
        imagealphablending($image_resized, false);
        imagesavealpha($image_resized, true);
        imagepng($image_resized);
        break;
    case IMAGETYPE_JPEG:
        $image = imagecreatefromjpeg($nome_arquivo);
        $image_resized = imagescale($image, $nova_largura, $nova_altura);
        imagejpeg($image_resized);
        break;
}
imagedestroy($image);
imagedestroy($image_resized);
```

GD (gerando thumbnail)

```
<?php
$nome_arquivo = 'ultrawide.jpg';
$largura_maxima = 1000; $altura_maxima = 300;
$fileType = exif_imagetype($nome_arquivo);
header("Content-type: ".image_type_to_mime_type($fileType));
list($largura, $altura) = getimagesize($nome_arquivo);
if ($largura > $largura_maxima || $altura > $altura_maxima) {
    $proporcao = min($largura_maxima / $largura, $altura_maxima / $altura);
    $largura = round($largura * $proporcao, 0);
    $altura = round($altura * $proporcao, 0);
}
switch ($fileType) {
    case IMAGETYPE_PNG:
        $image = imagecreatefrompng($nome_arquivo);
        $image_resized = imagescale($image, $largura, $altura);
        imagealphablending($image_resized, false);
        imagesavealpha($image_resized, true);
        imagepng($image_resized);
        break;
    case IMAGETYPE_JPEG:
        $image = imagecreatefromjpeg($nome_arquivo);
        $image_resized = imagescale($image, $largura, $altura);
        imagejpeg($image_resized);
        break;
}
imagedestroy($image);
imagedestroy($image_resized);
```

GD (Sobreposição de imagens)

```
<?php
$image = imagecreatefromjpeg('levi.jpg');
$image_sobrepor = imagecreatefromjpeg('pencil.jpg');

// Copiar e sobrepor
imagecopy($image, $image_sobrepor,
    10, 10, //posição inicial da sobreposição
    0, 0, //posição inicial da cópia
    100, 200); //tamanho da cópia

header('Content-Type: image/jpg');
imagejpeg($image);
imagedestroy($image);
imagedestroy($image_sobrepor);
```

GD (Merge de Imagens)

```
<?php
$image = imagecreatefromjpeg('levi.jpg');
$image_sobrepor = imagecreatefromjpeg('pencil.jpg');
$opacidade = 50;

// Copiar e mesclar
imagecopy($image, $image_sobrepor,
    10, 10, //posição inicial da sobreposição
    0, 0, //posição inicial da cópia
    100, 200, //tamanho da cópia
    $opacidade); // opacidade

header('Content-Type: image/jpg');
imagejpeg($image);
imagedestroy($image);
imagedestroy($image_sobrepor);
```

GD (marca d'água com transparência)

```
<?php
$nome_arquivo = 'levi.jpg';
$marca      = 'logo.png';
$opacidade   = 50;
$imagem = imagecreatefromjpeg($nome_arquivo);
$imagem_marca = imagecreatefrompng($marca);
list($largura_arquivo, $altura_arquivo) = getimagesize($nome_arquivo);
list($largura_marca, $altura_marca) = getimagesize($marca);
$margem = 20;
$centerX = $largura_arquivo - $largura_marca - $margem;
$centerY = $altura_arquivo - $altura_marca - $margem;

$cut = imagecreatetruecolor($largura_marca, $altura_marca);
imagecopy($cut, $imagem, 0, 0, $centerX, $centerY, $largura_marca,
$altura_marca);
imagecopy($cut, $imagem_marca, 0, 0, 0, 0, $largura_marca, $altura_marca);
imagecopymerge($imagem, $cut, $centerX, $centerY, 0, 0, $largura_marca,
$altura_marca, $opacidade);

header('Content-type: image/png');
imagepng($imagem);
imagedestroy($imagem);
```

GD (Filtros)



GD (Filtro de Brilho, Contraste e Inversão de cores)

```
<?php  
  
imagefilter($image, IMG_FILTER_BRIGHTNESS, 100); //positivo ou negativo  
imagefilter($image, IMG_FILTER_CONTRAST, 50); //positivo ou negativo  
imagefilter($image, IMG_FILTER_NEGATE);
```

GD (Filtro de Coloração e Escala de Cinza)

```
<?php  
$nome_arquivo = 'levi.jpg';  
$image = imagecreatefromjpeg($nome_arquivo);  
  
imagefilter($image, IMG_FILTER_GRAYSCALE);  
  
header('Content-type: image/jpeg');  
imagejpeg($image);  
imagedestroy($image);
```

GD (Filtros de realce de arestas)

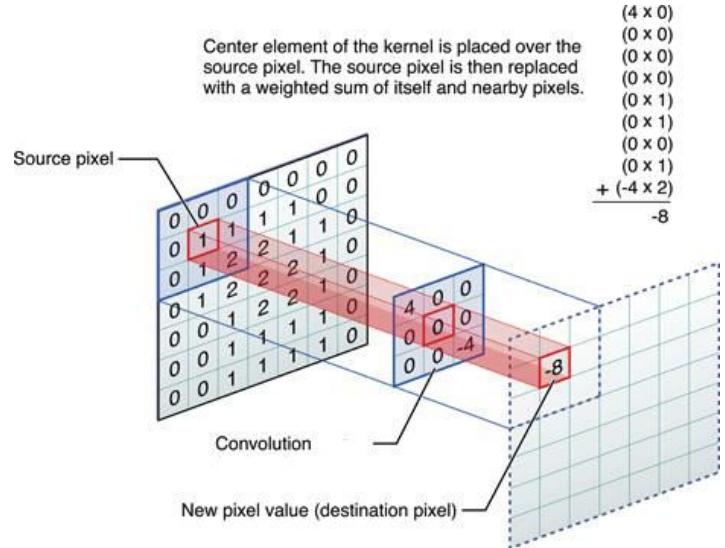
GD (Filtros de desfoque e suavização)

GD (Filtros de Esboço e Pixelização)

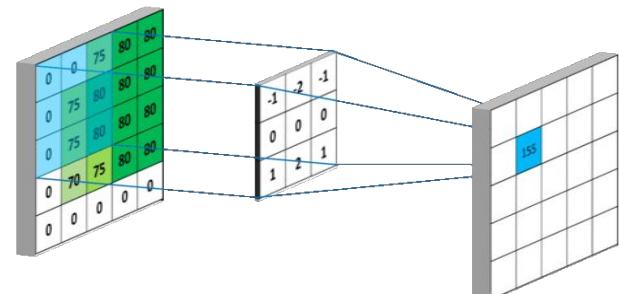
GD (Filtro de Sépia

Filtro matriz de convolução

- A convolução vem do latim *convolvere*, que significa 'rolar juntos'. Na computação gráfica, a convolução é uma maneira de calcular um novo valor para cada pixel na imagem para então transformá-lo.
- O filtro de convolução permite modificar uma imagem utilizando como parâmetro um array (matriz) multidimensional de 3x3. Este array é chamado de kernel (configuração) ou matriz convolução.
- Ao multiplicar um conjunto de pixels com esse kernel, o filtro produz a saída para o pixel "focado" (o pixel no meio do conjunto).
- Com essa matriz, é possível aplicar efeitos de desfoque, nitidez, relevo, detecção de bordas e muito mais.
- Um filtro de convolução passa por todos os pixels da imagem de tal maneira que, em um determinado momento, é recuperado um 'produto escalar' do filtro de convolução e os pixels da imagem.



fonte:
https://www.researchgate.net/figure/Matrix-convolution-filter-algorithm-as-a-graphic-diagram-12_fig18_304526867



GD (função imageconvolution)

- O GD fornece a função `imageconvolution()` para aplicar uma matriz de convolução 3x3 a um resource de imagem.
- O parâmetro `$matrix` é uma matriz de três matrizes, cada uma das quais contém três valores flutuantes - ou seja, é uma matriz 3x3. O primeiro elemento da primeira matriz é multiplicado pelo valor da cor do pixel superior esquerdo. Da mesma forma, o segundo elemento da primeira matriz é multiplicado pelo valor da cor do pixel diretamente na parte superior do pixel central. A cor final do pixel é obtida adicionando o resultado de todas essas multiplicações e dividindo-o por `$div` para normalização.
- O parâmetro `$div` é usado como um divisor para que o resultado da convolução normalize seu valor. A normalização geralmente mantém o valor final da cor abaixo de 255.
- O parâmetro `$offset`, por outro lado, é usado para especificar um valor de deslocamento para todas as cores.

GD (Image Convolution)

```
<?php
$image = imagecreatefromjpeg("levi.jpg");
$sharpen = [[0, -1, 0], [-1, 5, -1], [0, -1, 0]];
$emboss = [[-2, -1, 0], [-1, 1, 1], [0, 1, 2]];
$edge_detection = [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]];
imageconvolution($image, $sharpen, 1, 0);
header("Content-type: image/jpg");
imagejpeg($image);
```

Design Patterns & Princípios de Design de Software

- Durante o ciclo de desenvolvimento de software, umas etapas/atividades mais importantes é o design.
- Nesta atividade o arquiteto de software pode identificar um problema comum de design que já tenha sido resolvido no passado;
- A resolução deste problema comum de design de software, quando é amplamente testada e utilizada por muitos desenvolvedores, pode ser promovida para o status de um padrão.
- A reutilização destes padrões pode ajudar a acelerar o processo de desenvolvimento de software, além de reduzir custos também (WIKIPEDIA, 2019)

Design e Design de Software

- Design é “*uma especificação de um objeto, manifestada por algum agente, tendendo a atingir objetivos, em um ambiente específico, usando um conjunto de componentes primitivos, satisfazendo um conjunto de requisitos, sujeito a algumas restrições*” (Ralph e Wand, 2009);
- Segundo a ISO 24765 (2017), no contexto de engenharia de software, design é:
 - “*processo de definição da arquitetura, componentes, módulos, interfaces e dados de software para um sistema de software para atender aos requisitos especificados*”;
 - “*o resultado desse processo*”;

Design de Software e Princípios

- “[...]princípio é usado como sinônimo de valor fundamental (“É uma questão de princípio”), como um elemento da noção básica (os princípios da ética, da matemática, da física etc.) ou como uma abstração progressiva generalizada de uma série de dados e casos particulares (GUIDO, 1994);
- “Os princípios de design de software estão preocupados em fornecer meios para lidar com a complexidade do processo de design de maneira eficaz. O gerenciamento eficaz da complexidade não apenas reduzirá o esforço necessário para o design, mas também reduzirá o escopo da introdução de erros durante o design” (JAVATPOINT, 2018);

5 Princípios de Design de Software OO (S.O.L.I.D)

- SOLID é um acrônimo mnemônico para cinco princípios de design, destinados a tornar os projetos de software mais compreensíveis, flexíveis e sustentáveis.
- Os princípios foram introduzidos pelo engenheiro e instrutor de software americano Robert C. Martin em seu artigo de 2000 intitulado *Design Principles and Design Patterns*.
- Os cinco princípios são:
 - a. Princípio da responsabilidade Única
 - b. Princípio aberto/fechado
 - c. Princípio da substituição de Liskov
 - d. Princípio da segregação de Interface
 - e. Princípio da inversão de dependência

Single Responsibility Principle



SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

***“Princípio da responsabilidade única:
só porque você pode, não significa que
você deve”***

Single Responsibility Principle

- “*Uma classe deve ter apenas um motivo para mudar*”. (MARTIN, 2003);
- O princípio da responsabilidade única afirma que toda classe deve ter responsabilidade sobre uma única parte da funcionalidade fornecida pelo software e que a responsabilidade deve ser totalmente encapsulada pela classe. Todos os seus métodos devem estar estritamente alinhados com essa responsabilidade.
- “*Se uma classe tem mais de uma responsabilidade, esta responsabilidade fica incorporada a classe. Mudanças em uma responsabilidade podem prejudicar ou inibir a capacidade da classe de atender as outras*”. (MARTIN, 2003);

Single Responsibility Principle (antes)

```
<?php
class Funcionario {
    private $nome;
    private $matricula;
    private $salarioBruto;
    public function __construct(string $nome, string $matricula, float $salarioBruto) {
        $this->nome = $nome;
        $this->matricula = $matricula;
        $this->salarioBruto = $salarioBruto;
    }
    public function calcularINSS(): float{
        $inss = 0;
        switch(true){
            case($this->salarioBruto <= 1751.81):
                $inss = 8;
                break;
            case($this->salarioBruto >= 1751.82 && $this->salarioBruto <= 2919.72):
                $inss = 9;
                break;
            default:
                $inss = 11;
                break;
        }
        return round(($inss/100)*$this->salarioBruto,2);
    }
}
$f1 = new Funcionario('José', '123', 3000.00);
echo $f1->calcularINSS();
```

Neste exemplo a classe Funcionário possui um acúmulo de responsabilidades pois, além dos comportamentos do Funcionário diretamente em si, a mesma possui responsabilidades sobre o cálculo de desconto sobre o valor do salário do funcionário.

Single Responsibility Principle (depois)

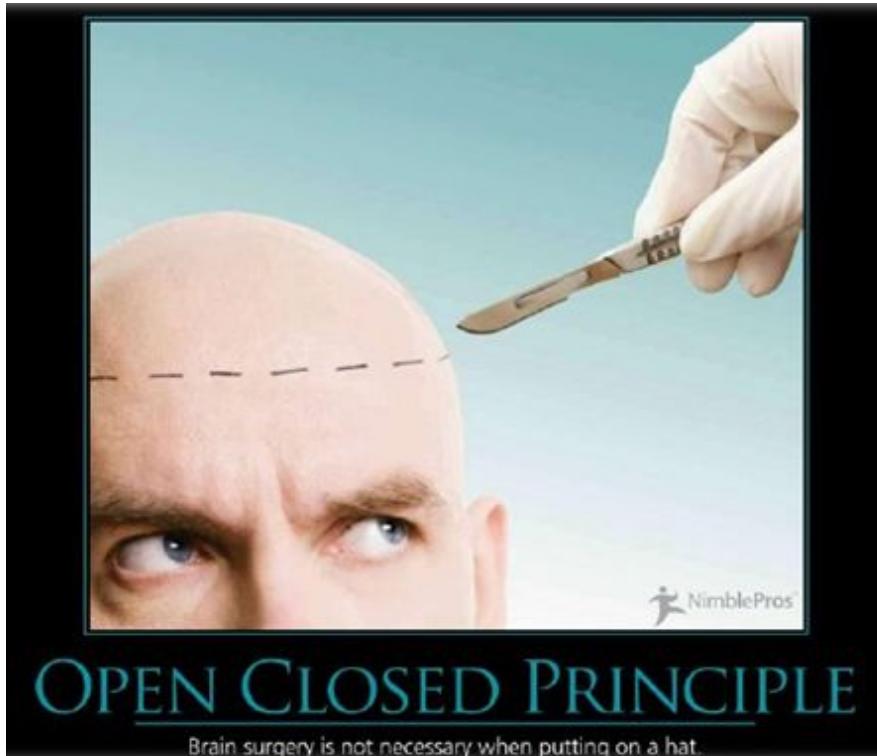
```
<?php
class Funcionario {
    private $nome;
    private $matricula;
    private $salarioBruto;
    public function __construct(string $nome, string $matricula, float
$salarioBruto) {
        $this->nome = $nome;
        $this->matricula = $matricula;
        $this->salarioBruto = $salarioBruto;
    }
    public function getSalarioBruto(): float{
        return $this->salarioBruto;
    }
}
```

```
$f1 = new Funcionario('José', '123', 3000.00);
$c1 = new CalculadoraSalarial($f1);
echo $c1->calcularINSS();
```

Aplicando o princípio SRP, separamos a responsabilidade para outra classe;

```
class CalculadoraSalarial {
    private $funcionario;
    public function __construct(Funcionario $funcionario) {
        $this->funcionario = $funcionario;
    }
    public function calcularINSS(): float{
        $salarioBruto = $this->funcionario->getSalarioBruto();
        $inss = 0;
        switch(true){
            case($salarioBruto <= 1751.81):
                $inss = 8;
                break;
            case($salarioBruto >= 1751.82 && $salarioBruto <= 2919.72):
                $inss = 9;
                break;
            default:
                $inss = 11;
                break;
        }
        return round((($inss/100)*$salarioBruto),2);
    }
    public function calcularIRRF(): float{
        //...
    }
}
```

Open/Closed Principle



***“Princípio aberto fechado:
cirurgia no cérebro não é necessária ao
colocar um chapéu”***

OPEN CLOSED PRINCIPLE

Brain surgery is not necessary when putting on a hat.

Open/Closed Principle

- “Classes deve ser abertas para extensões mas fechadas para modificações”. (MARTIN, 2003);
- Princípio estabelecido por Bertrand Meyer nos anos 90 seu livro *Object-Oriented Software Construction*:
 - “A abertura é uma preocupação natural dos desenvolvedores de software, pois eles sabem que é quase impossível prever todos os elementos - dados, operações - que um módulo precisará em sua vida útil; portanto, eles desejam manter o máximo de flexibilidade possível para futuras alterações e extensões”.
 - “Se nunca fechamos um módulo até termos certeza de que ele inclui todos os recursos necessários, nenhum software com vários módulos chegaria à conclusão: todo desenvolvedor sempre aguardaria a conclusão do trabalho de outra pessoa. Com técnicas tradicionais, os dois objetivos são incompatíveis”. (MEYER, 1997);

Open/Closed Principle (antes)

```
<?php
class Item {
    public float $valor;
    public float $peso;
    public function __construct(float $valor, float $peso) {
        $this->valor = $valor;
        $this->peso = $peso;
    }
}
class Pedido {
    private \DateTimeInterface $data;
    private array $items;
    private string $tipoFrete;

    public function __construct(\DateTime $data, array $items, string $tipoFrete) {
        $this->data = \DateTimeImmutable::createFromMutable($data);
        $this->items = $items;
        $this->tipoFrete = $tipoFrete;
    }
    public function getPesoTotal(): float{
        return array_reduce($this->items, function($soma, Item $item){
            return $soma + $item->peso;
        });
    }
    public function getValorItems(){
        return array_reduce($this->items, function($soma, Item $item){
            return $soma + $item->valor;
        });
    }
}
```

```
public function getValorFrete(): float{
    if($this->tipoFrete == 'PAC'){
        if($this->getValorItems() > 120.00){
            return 0;
        }
        return ($this->getPesoTotal() * 1.5);
    }
    if($this->tipoFrete == 'SEDEX'){
        return ($this->getPesoTotal() * 3);
    }
}
public function getDataEntrega(): \DateTimeImmutable {
    if($this->tipoFrete == 'PAC'){
        return $this->data->modify('+ 15 days');
    }
    if($this->tipoFrete == 'SEDEX'){
        return $this->data->modify('+5 days');
    }
}
$pedido = new Pedido(new \DateTime(), [
    new Item(40.50, 2), new Item(90.20, 18)], 'SEDEX');
echo $pedido->getValorFrete()."<br/>";
echo $pedido->getDataEntrega()->format('d/m/Y');
```

Open/Closed Principle (depois - parte1)

```
class Pedido {
    private \DateTimeInterface $data;
    private array $items;
    private Frete $frete;
    public function __construct(\DateTime $data, array $items, Frete $frete) {
        $this->data = \DateTimeImmutable::createFromMutable($data);
        $this->items = $items;
        $this->frete = $frete;
        $this->frete->setPedido($this);
    }
    public function getPesoTotal(): float{
        return array_reduce($this->items, function($soma, Item $item){
            return $soma + $item->peso;
        });
    }
    public function getValorItems(){
        return array_reduce($this->items, function($soma, Item $item){
            return $soma + $item->valor;
        });
    }
    public function getData() {
        return $this->data;
    }
    public function getItems() {
        return $this->items;
    }
    function getFrete(){
        return $this->frete;
    }
}
```

```
abstract class Frete {
    protected Pedido $pedido;
    function getPedido(): Pedido {
        return $this->pedido;
    }
    function setPedido(Pedido $pedido): void {
        $this->pedido = $pedido;
    }
    public abstract function getValorFrete(): float;
    public abstract function getDataEntrega(): \DateTimeImmutable;
}
```

```
class Item {
    public float $valor;
    public float $peso;
    public function __construct(float $valor, float $peso) {
        $this->valor = $valor;
        $this->peso = $peso;
    }
}
```

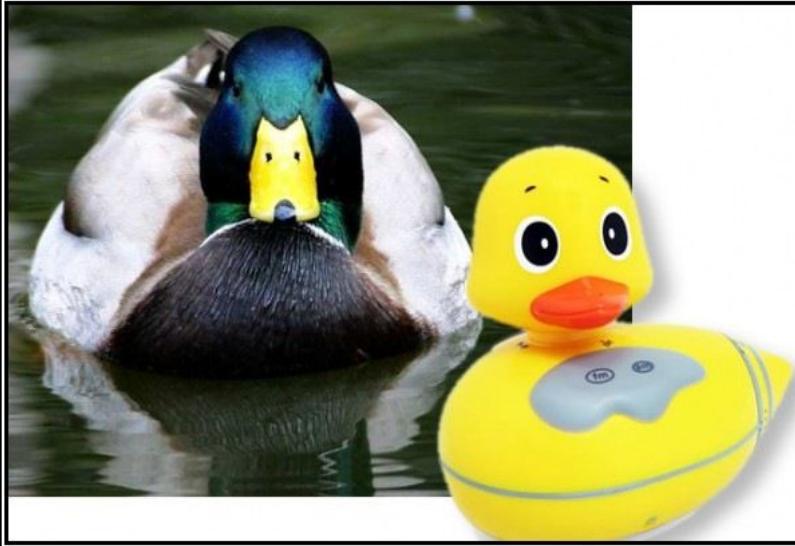
Open/Closed Principle (depois - parte 2)

```
class Pac extends Frete {
    public function getDataEntrega(): \DateTimeImmutable {
        return $this->pedido->getData()->modify('+15 days');
    }
    public function getValorFrete(): float {
        if ($this->pedido->getValorItems() > 120.00) {
            return 0;
        }
        return ($this->pedido->getPesoTotal() * 1.5);
    }
}

class Sedex extends Frete {
    public function getDataEntrega(): \DateTimeImmutable {
        return $this->pedido->getData()->modify('+5 days');
    }
    public function getValorFrete(): float {
        return ($this->pedido->getPesoTotal() * 3);
    }
}

$pedido = new Pedido(new \DateTime(),
    [new Item(40.50, 2), new Item(90.20, 18)], new Sedex());
echo $pedido->getFrete()->getValorFrete()."<br/>";
echo $pedido->getFrete()->getDataEntrega()->format('d/m/Y');
```

Liskov (Princípio de Substituição de Liskov)



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You
Probably Have The Wrong Abstraction

*“Princípio de Substituição de Liskov:
se parece com um pato, granya como um
pato mas precisa de baterias: você
provavelmente tem a abstração errada”*

Liskov (Princípio de Substituição de Liskov)

- Este princípio criado por Barbara Liskov, ganhadora do prêmio Turing de 2008, onde os “*Subtipos devem ser substituíveis por seus tipos de base*”;
- “*Se para cada objeto o1 do tipo S houver um objeto o2 do tipo T, de modo que, para todos os programas P definidos em termos de T, o comportamento de P seja inalterado quando o1 for substituído para o2 então S é um subtipo de T*”;
- Este princípio também impõe alguns padrões de requisitos tipos de entrada e saída de método e também no caso de exceções;
- Além dos requisitos de assinatura, o subtipo deve atender a várias condições comportamentais. Eles são detalhados em uma terminologia semelhante à da metodologia do **Design por contrato**.



Liskov (Checklist)

- Objetos de subclasses podem ser substituídos por objetos de suas classes de base?
- A contravariância não é violada nos parâmetros dos métodos?
- A covariância não é violada nos tipos de retorno dos métodos?
- Nenhuma nova exceção é lançada, a menos que as exceções sejam subtipos de exceções lançadas pelo pai?
- Nenhuma pré-condição foi reforçada ?
- Nenhuma pós-condição foi enfraquecida ?
- Todos os Invariantes foram preservados nas subclasses?
- O Histórico de Restrição não é violado ?

Covariância (PHP 7.4 >=)

```
<?php

abstract class Animal {
    protected string $nome;
    public function __construct(string $nome) {
        $this->nome = $nome;
    }
    abstract public function fazerSom();
}

class Cachorro extends Animal {
    public function fazerSom() {
        echo $this->nome . ": au au !";
    }
}

class Gato extends Animal {
    public function fazerSom() {
        echo $this->nome . ": miau !";
    }
}

interface AbrigoAnimal {
    public function adotar(string $nome): Animal;
}
```

```
class AbrigoGato implements AbrigoAnimal {
    public function adotar(string $nome): Gato {
        return new Gato($nome);
    }
}

class AbrigoCachorro implements AbrigoAnimal {
    public function adotar(string $nome): Cachorro {
        return new Cachorro($nome);
    }
}

$gatinho = (new AbrigoGato)->adotar("Darwin");
$gatinho->fazerSom();
echo "<br />";
$doguin = (new AbrigoCachorro)->adotar("Bolota");
$doguin->fazerSom();
```

No âmbito do Orientação a Objetos, covariância permite que um método de uma classe filha retorne um tipo mais específico do que o tipo de retorno do método de seu pai.

Também existe a possibilidade de covariância de tipos de parâmetros, porém, no PHP, o mesmo só é possível no método construtor.

Contravariância (PHP 7.4 >=)

```
<?php
class Alimento {}
class AlimentoAnimal extends Alimento {}

abstract class Animal {
    protected string $nome;
    public function __construct(string $nome) {
        $this->nome = $nome;
    }
    public function comer(AlimentoAnimal $alimento){
        echo $this->nome . ": nhac nhac comendo".
get_class($alimento);
    }
}
class Cachorro extends Animal {
    public function comer(Alimento $alimento){
        echo $this->nome . ": chomp chomp comendo " .
get_class($alimento);
    }
}

class Gato extends Animal {}
interface AbrigoAnimal {
    public function adotar(string $nome): Animal;
}
```

```
class AbrigoGato implements AbrigoAnimal {
    public function adotar(string $nome): Gato {
        return new Gato($nome);
    }
}

class AbrigoCachorro implements AbrigoAnimal {
    public function adotar(string $nome): Cachorro {
        return new Cachorro($nome);
    }
}

$gatinho = (new AbrigoGato)->adotar("Darwin");
$whiskas = new AlimentoAnimal();
$gatinho->comer($whiskas);
echo "<br />";
$doguin = (new AbrigoCachorro)->adotar("Bolota");
$arroz = new Alimento();
$doguin->comer($arroz);
```

Contravariância, por outro lado, permite que um tipo de parâmetro de um método seja menos específico em um método filho do que o de seu pai.

É o caso do método comer na classe Filha Cachorro, que utiliza um tipo mais genérico que no original de sua classe Pai.

Exceptions com herança

```
<?php  
class ComidaBaixaQualidadeException extends Exception {}  
class ComidaGatoBaixaQualidadeException extends  
ComidaBaixaQualidadeException {}  
class ComidaSemQualidadeException extends Exception {}  
  
class Animal{}  
  
class Comida {  
    private \DateTime $dataValidade;  
    public function __construct(\DateTime $dataValidade){  
        $this->dataValidade = $dataValidade;  
    }  
    public function isConsumivel(): bool {  
        return ($this->dataValidade >= (new DateTime('today')));  
    }  
}  
  
class Dono {  
    public function alimentar(Animal $animal, Comida $comida){  
        if (!$comida->isConsumivel())  
            throw new ComidaBaixaQualidadeException();  
    }  
}
```

```
class DonoFelino extends Dono {  
    public function alimentar(Animal $animal, Comida $comida){  
        if (!$comida->isConsumivel())  
            throw new ComidaGatoBaixaQualidadeException();  
    }  
}  
class DonoGenerico extends Dono {  
    public function alimentar(Animal $animal, Comida $comida){  
        if (!$comida->isConsumivel())  
            throw new ComidaSemQualidadeException();  
    }  
}  
function testar(Dono $dono){  
    try{  
        $dono->alimentar(new Animal(), new Comida(new \DateTime('yesterday')));  
    }catch(ComidaBaixaQualidadeException $e){  
        echo "ocorreu uma exception !";  
    }  
}  
  
$dono = new DonoFelino();  
$dono2 = new DonoGenerico();  
testar($dono2);
```

No princípio de Liskov: nenhuma nova exceção deve ser lançada pelos métodos herdados nas sub classes, exceto quando essas exceções são iguais ou subtipos das exceções lançadas pelos métodos da super classe.

Pré-condição em herança (enfraquecimento e reforço)

```
<?php
class CalculadoraPrazo {
    public function data(int $dias): DateTimeInterface {
        if($dias > 0)
            return (new DateTime())->modify("+$dias days");
        throw new \InvalidArgumentException("Prazo precisa ser maior que zero ");
    }
}
class CalculadoraPrazoCLT extends CalculadoraPrazo {
    public function data(int $dias): DateTimeInterface {
        if ($dias >= 0)
            return (new DateTime())->modify("+$dias days");
        throw new \InvalidArgumentException("Prazo precisa ser igual ou maior que zero");
    }
}
class CalculadoraPrazoCPC extends CalculadoraPrazo {
    public function data(int $dias): DateTimeInterface {
        if (in_array($dias, range(1,30)))
            return (new DateTime())->modify("+$dias days");
        throw new \InvalidArgumentException("Prazo precisa ser entre 1 e 30 ");
    }
}
echo (new CalculadoraPrazo)->data(31)->format("d/m/Y")."<br/>";
echo (new CalculadoraPrazoCLT)->data(31)->format("d/m/Y")."<br/>";
echo (new CalculadoraPrazoCPC)->data(31)->format("d/m/Y");
```

“Quando alguém substitui um método, não deve reforçar a pré-condição.” (BREUGEL, 2007).

<http://www.cse.yorku.ca/~buildit/notes/6/pre.pdf>

Pós-condição em herança (reforço e enfraquecimento)

```
<?php
class ContaBancaria {
    protected float $saldo;
    public function __construct(float $saldoinicial){
        $this->saldo = $saldoinicial;
    }
    public function sacar(float $valor) : float { //retorna apenas valores positivos
        if(($this->saldo - $valor) >= 0)
            $this->saldo -= $valor;
        return $this->saldo;
    }
}
class ContaBancariaVip extends ContaBancaria {
    private const TAXA = 10.00;
    public function sacar(float $valor) : float {
        if(($this->saldo - $valor) >= self::TAXA)
            $this->saldo -= $valor;
        return $this->saldo;
    }
}
class ContaBancariallimitada extends ContaBancaria {
    public function sacar(float $valor) : float {
        $this->saldo -= $valor;
        return $this->saldo;
    }
}
```

“Uma subclasse que enfraquece uma pós-condição está dizendo que ela não pode fazer tudo o que sua superclasse pode fazer”. (MA, 2009)

Invariância

```
<?php
class Pessoa {
    protected string $nome;
    protected string $apelido;
    protected function invariant(){
        assert(($this->nome != $this->apelido));
    }
    public function __construct(string $nome, string $apelido){
        $this->nome = $nome;
        $this->apelido = $apelido;
    }
    public function __set($nome, $value){
        $this->invariant();
        $this->$nome = $value;
    }
    public function __get($nome){
        $this->invariant();
        return $this->$nome;
    }
}
class BoaPessoa extends Pessoa {
    public function getNomeCompleto(){
        return $this->nome." ".$this->apelido. "!!!";
    }
}
class MalvadaPessoa extends Pessoa {
    protected function invariant(){
        assert(($this->nome != ""));
    }
}
```

```
$gp = new BoaPessoa("João", "Joãozinho");
echo $gp->apelido;
echo $gp->getNomeCompleto();
$bp = new MalvadaPessoa("João", "João");
echo $bp->apelido;
```

- Na matemática, um invariante é uma propriedade de um objeto matemático (ou uma classe de objetos matemáticos) que permanece inalterada, após operações ou transformações de um determinado tipo serem aplicadas aos objetos (WIKIPEDIA, 2019).
- Na OOP, invariância é um conjunto de asserções que sempre devem ser verdadeiras durante a vida de um objeto para que o programa seja válido (NODET, 2010)
- Neste exemplo foi utilizado uma assertion, que é uma ferramenta de desenvolvimento e um recurso de linguagens usadas para verificar se uma expressão condicional é avaliada como verdadeira quando o programa é executado
- Segundo Liskov, Invariantes do supertipo devem ser preservados em um subtipo.
- Este princípio é similar ao conceito da programação por contrato ou design por contrato.
- Algumas linguagens de programação como o D possuem recursos explícitos para implementação de invariância ([link deste exemplo em D](#))

History Constraint or History Rule

```
<?php
class Funcionario {
    protected string $matricula;
    public   string $nome;
    public function __construct(string $nome){
        $this->nome = $nome;
        $this->matricula = uniqid();
    }
    public function getMatricula(): string {
        return $this->matricula;
    }
}
class Gerente extends Funcionario {
    //violação
    public function setMatricula(string $matricula){
        $this->matricula = $matricula;
    }
}
$f1 = new Gerente('José');
$f1->setMatricula('123');
echo $f1->getMatricula();
```

Liskov (antes - parte 1)

```
<?php
class RacaoPato {
    public $sabor;
    public function __construct(string $sabor) {
        $this->sabor = $sabor;
    }
}
class Pato {
    protected $cores = [];
    public function nadar(float $metros) {
        echo "nadando distância de $metros metros";
    }
    public function grasar() {
        return "quack! quá, quá, quá! quac! quac!";
    }
    public function comer(RacaoPato $racao) {
        echo "comendo ração de sabor {$racao->sabor}";
    }
}
class PatoCayuga extends Pato {
    protected $cores = ['verde', 'preto'];
}
```

```
class PatoBrinquedo extends Pato {

    protected $cores = ['amarelo'];
    private $audios = ['pato1.mp3', 'pato2.mp3'];

    public function nadar(float $metros) {
        echo "Flutando $metros metros e gastando pilha";
    }

    public function grasar() {
        return $audios; //array e não string
    }

    public function comer(RacaoPato $racao) {
        throw \BadMethodCallException('Método não implementado !');
    }
}
```

Excesso de overrides para se adequar a situações específicas podem levar a problemas de manutenção.

Liskov (antes - parte 2)

```
function testar_pato(Pato $pato){
    echo "nadar 10 metros: </br>";
    $pato->nadar(10);
    $racao = new RacaoPato('xpto');
    echo "<br/> comer ração de pato xpto: <br/>";
    $pato->comer($racao);
}

$o1 = new Pato();
$o2 = new PatoBrinquedo();
testar_pato($o1);
```

Interface Segregation Principle



INTERFACE SEGREGATION

Tailor interfaces to individual clients' needs.

“Princípio da Segregação de Interface:
Adapte interfaces para as necessidades individuais dos clientes”.

Interface Segregation

- Segundo MARTIN (2003):
 - “Os clientes não devem ser forçados a depender dos métodos que não usam”.
 - “*Quando os clientes são forçados a depender dos métodos que não usam, esses clientes estão sujeitos a alterações nesses métodos. Isso resulta em um acoplamento inadvertido entre todos os clientes. Dito de outra maneira, quando um cliente depende de uma classe que contém métodos que o cliente não usa, mas que outros clientes usam, esse cliente será afetado pelas alterações que esses outros clientes impõem à classe. Gostaríamos de evitar esses acoplamentos sempre que possível e, portanto, queremos separar as interfaces.*”

Interface Segregation (antes)

```
<?php
interface Leitura {
    public function getConteudo();
    public function getTamanho(): int;
    public function getDono(): string;
}
class LogTexto implements Leitura {
    private $nome_arquivo;
    public function __construct(string $nome_arquivo) {
        if (is_file($nome_arquivo)) {
            $this->nome_arquivo = $nome_arquivo;
        } else {
            throw new \InvalidArgumentException('arquivo não existe !');
        }
    }
    public function getConteudo(): string {
        return file_get_contents($this->nome_arquivo);
    }
    public function getDono(): string {
        return fileowner($this->nome_arquivo);
    }
    public function getTamanho(): int {
        return filesize($this->nome_arquivo);
    }
}
```

```
class CSVDengue implements Leitura {
    const URL =
        'https://zenodo.org/api/files/613e9628-aa46-4a8e-bb95-ad6a4740c5a9/W_Table_dat
a.csv';
    //problemas com o princípio de liskov aqui também
    public function getConteudo(): array {
        return array_map('str_getcsv',
            explode(PHP_EOL, file_get_contents(self::URL))
        );
    }
    public function getDono(): string {
        throw \BadMethodCallException('Método não implementado !');
    }
    public function getTamanho(): int {
        throw \BadMethodCallException('Método não implementado !');
    }
}
$t = new LogTexto('log.txt');
var_dump($t->getDono());
$c = new CSVDengue();
var_dump($c->getConteudo());
```

Interface Segregation (depois)

```
<?php  
interface Leitura {  
    public function getConteudo(): string;  
}  
interface ArrayCSV {  
    public function getAsArray(): array;  
}  
interface Metadados {  
    public function getTamanho(): int;  
    public function getDono(): string;  
}  
class LogTexto implements Leitura, Metadados {  
    private $nome_arquivo;  
    public function __construct(string $nome_arquivo) {  
        if (is_file($nome_arquivo)) {  
            $this->nome_arquivo = $nome_arquivo;  
        } else {  
            throw new \InvalidArgumentException('arquivo não existe !');  
        }  
    }  
    public function getConteudo(): string {  
        return file_get_contents($this->nome_arquivo);  
    }  
    public function getDono(): string {  
        return fileowner($this->nome_arquivo);  
    }  
    public function getTamanho(): int {  
        return filesize($this->nome_arquivo);  
    }  
}
```

```
class CSVDengue implements Leitura, ArrayCSV {  
  
    const URL =  
        'https://zenodo.org/api/files/613e9628-aa46-4a8e-bb95-ad6a4740c5a9/  
        W_Table_data.csv';  
  
    public function getConteudo(): string {  
        return file_get_contents(self::URL);  
    }  
  
    public function getAsArray(): array {  
        return array_map('str_getcsv',  
            explode(PHP_EOL, $this->getConteudo())  
        );  
    }  
}
```

Dependency Inversion Principle



DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

***“Princípio da Inversão de dependência:
Você soldaria uma lâmpada diretamente
na fiação elétrica em uma parede?”***

Dependency Inversion Principle

Segundo MARTIN (2003):

1. *Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.*
2. *As abstrações não devem depender de detalhes. Os detalhes é quem devem depender da abstração*

Spasojevic (2019) descreve que “A idéia básica por trás do Princípio de Inversão de Dependência é que devemos criar os módulos de nível superior com sua lógica complexa de forma a ser reutilizável e não afetada por qualquer alteração dos módulos de nível inferior em nosso aplicativo. Para atingir esse tipo de comportamento em nossos aplicativos, apresentamos abstrações que separam mais alto dos módulos de nível inferior”.

Dependency Inversion Principle

Segundo Spasojevic (2019):

Os módulos de **baixo nível** contêm componentes individuais mais específicos, com foco em detalhes e em partes menores do aplicativo. Esses módulos são usados dentro dos módulos de alto nível em nosso aplicativo.

Os módulos de **alto nível** descrevem as operações em nosso aplicativo que têm natureza mais abstrata e contêm lógica mais complexa. Esses módulos orquestram os módulos de baixo nível em nosso aplicativo.

Dependency Inversion Principle (antes)

```
<?php  
class Funcionario {  
    public string $nome;  
    public string $sexo;  
    public string $cargo;  
    public function __construct(string $nome, string $sexo, string $cargo) {  
        $this->nome = $nome;  
        $this->sexo = $sexo;  
        $this->cargo = $cargo;  
    }  
}  
  
class Departamento {  
    private array $funcionarios;  
    public function __construct() {  
        $this->funcionarios = [];  
    }  
    public function addFuncionario(Funcionario $funcionario) {  
        $this->funcionarios[] = $funcionario;  
    }  
    public function getFuncionarios(): array {  
        return $this->funcionarios;  
    }  
}
```

```
class Relatorio {  
  
    private Departamento $departamento;  
    public function __construct(Departamento $departamento) {  
        $this->departamento = $departamento;  
    }  
  
    public function getTotalGerentesSexoFeminino(): int {  
        return array_reduce($this->departamento->getFuncionarios(),  
            function ($v, Funcionario $f) {  
                return $v + ($f->sexo == 'F' && $f->cargo == 'Gerente');  
            }, 0);  
    }  
}  
  
$departamento = new Departamento();  
$departamento->addFuncionario(new Funcionario('Maria','F', 'Gerente'));  
$departamento->addFuncionario(new Funcionario('Luiza','F', 'Gerente'));  
$departamento->addFuncionario(new Funcionario('José','M', 'Gerente'));  
$empSt = new Relatorio($departamento);  
echo $empSt->getTotalGerentesSexoFeminino();
```

Dependency Inversion Principle (depois)

```
<?php  
interface Enumeravel {  
    public function getTotalPorGeneroECargo(  
        string $genero, string $cargo) : Iterable;  
}  
  
class Funcionario {  
    public string $nome; public string $sexo;  
    public string $cargo;  
    public function __construct(string $nome, string $sexo, string $cargo){  
        $this->nome = $nome;  
        $this->sexo = $sexo;  
        $this->cargo = $cargo;  
    }  
}  
  
class Departamento implements Enumeravel {  
    private array $funcionarios;  
    public function __construct() {  
        $this->funcionarios = [];  
    }  
    public function addFuncionario(Funcionario $funcionario) {  
        $this->funcionarios[] = $funcionario;  
    }  
}
```

```
public function getTotalPorGeneroECargo(string $sexo, string $cargo):  
Iterable {  
    return array_filter($this->funcionarios, fn($emp) =>  
        ($emp->sexo == $sexo && $emp->cargo == $cargo));  
}  
  
class Relatorio {  
    private Enumeravel $enumeravel;  
    public function __construct(Enumeravel $enumeravel){  
        $this->enumeravel = $enumeravel;  
    }  
    public function getTotalGerentesMulheres() : int {  
        return count($this->enumeravel->getTotalPorGeneroECargo('F',  
'Gerente'));  
    }  
}  
$departamento = new Departamento();  
$departamento->addFuncionario(new Funcionario('Maria','F', 'Gerente'));  
$departamento->addFuncionario(new Funcionario('Luiza','F', 'Gerente'));  
$departamento->addFuncionario(new Funcionario('José','M', 'Gerente'));  
$empSt = new Relatorio($departamento);  
echo $empSt->getTotalGerentesMulheres();
```

Design Patterns - Definições

- “Os padrões de design são **soluções típicas para problemas comuns no design de software**. Eles são como modelos pré-fabricados que você pode personalizar para resolver um problema de design recorrente em seu código”. (SHVETS, 2019);
- “Um pattern é uma apresentação de uma **solução para um problema recorrente**. A solução é apenas a solução. O pattern é a apresentação, que leva você a uma discussão válida sobre a qualidade da apresentação”. (COOKBURN, 2004);
- Design Patterns são templates que auxiliam na construção de softwares melhores e com maior escalabilidade e extensibilidade, permitindo que futuras alterações sejam aplicadas de forma menos custosa, mais rápidas e mais organizadas;

Gang of Four Design Patterns

O conjunto mais famoso de Design Patterns é compilado do livro de 1994, o *Design Patterns: Elements of Reusable Object-Oriented Software*. Seus autores são:

Erich Gamma, que trabalha desde 2003 na Microsoft como líder de desenvolvimento do VSCode;

Richard Helm, que atualmente trabalha como consultor de TI;

Ralph Johnson, professor na Universidade de Illinois;

John Vlissides, pesquisador na IBM que infelizmente faleceu em 2005 vítima de um tumor no cérebro;



Grupos de Design Patterns GoF

1. **Criacionais** (5):
Abstract Factory, Builder, Factory Method, Prototype e Singleton;
2. **Estruturais** (7):
Adapter, Bridge, Composite, Decorator, Facade, Flyweight e Proxy;
3. **Comportamentais** (11):
Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento
Observer, State, Strategy, Template e Visitor;

Design Patterns Criacionais

São padrões de design de software orientado a objetos cuja finalidade é prover mecanismos para criação de novos objetos encapsulando detalhes da instanciação dos mesmos;

Inúmeros domínios podem exigir um conjunto de passos, restrições ou até associações na criação de determinados objetos de classes ou família de classes;

Ao utilizar um pattern criacional, o designer evitará erros do desenvolvedor se o mesmo tentasse instanciar, em todas as situações, utilizando o operador tradicional “new”.

Factory Method

O Factory Method é um padrão criacional que utiliza “métodos” para lidar com o problema ao criar objetos sem precisar especificar a classe exata do objeto que será criado.

Isso é feito criando objetos chamando um método “factory” especificado em uma interface e implementado por classes filhas, ou implementado em uma classe base e opcionalmente substituído por classes derivadas - em vez de chamar o construtor (WIKIPEDIA, 2020).



Factory Method (Conceitual)

```
<?php
interface Produto{
    public function fazerAlgo();
}

class ProdutoConcretoA implements Produto {
    public function fazerAlgo() {
    }
}

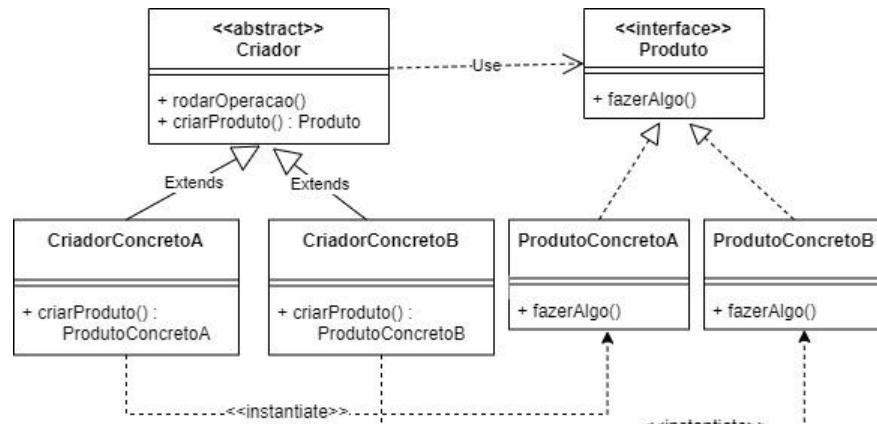
class ProdutoConcretoB implements Produto {
    public function fazerAlgo() {
    }
}

abstract class Criador {
    public function rodarOperacao(){
        $this->criarProduto()->fazerAlgo();
    }

    public abstract function criarProduto(): Produto;
}

class CriadorConcretoA extends Criador {
    public function criarProduto(): ProdutoConcretoA {
    }
}

class CriadorConcretoB extends Criador {
    public function criarProduto(): ProdutoConcretoB {
    }
}
```

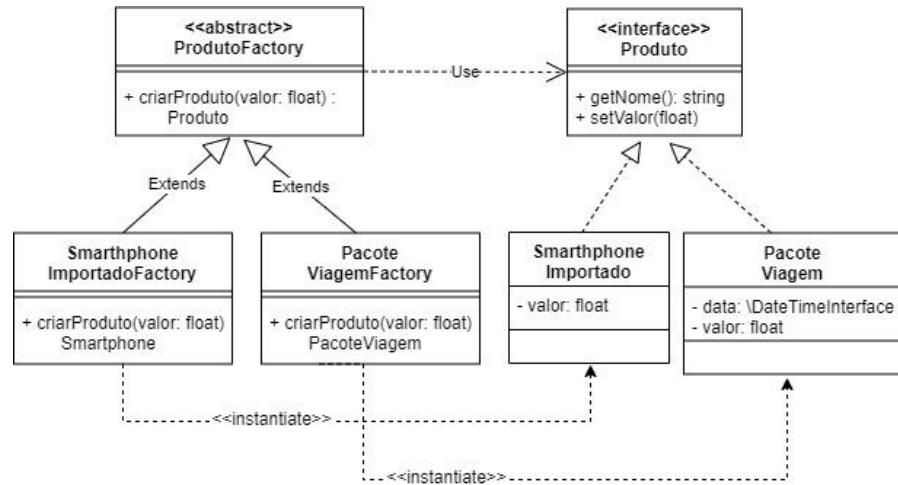


Factory Method (Exemplo)

```
<?php
interface Produto {
    public function getName(): string;
    public function setValor(float $valor);
}

class SmartphoneImportado implements Produto {
    private float $valor;
    public function getName(): string {
        return "Smartphone Importado XPTO";
    }
    public function setValor(float $valor) {
        $this->valor = $valor;
    }
}

class PacoteViagem implements Produto {
    private \DateTimeInterface $data;
    private float $valor;
    public function __construct(\DateTimeInterface $data) {
        $this->data = $data;
    }
    public function getName(): string {
        return "Pacote de Viagem";
    }
    public function setValor(float $valor) {
        $this->valor = $valor;
    }
}
```



Factory Method (Exemplo - continuação)

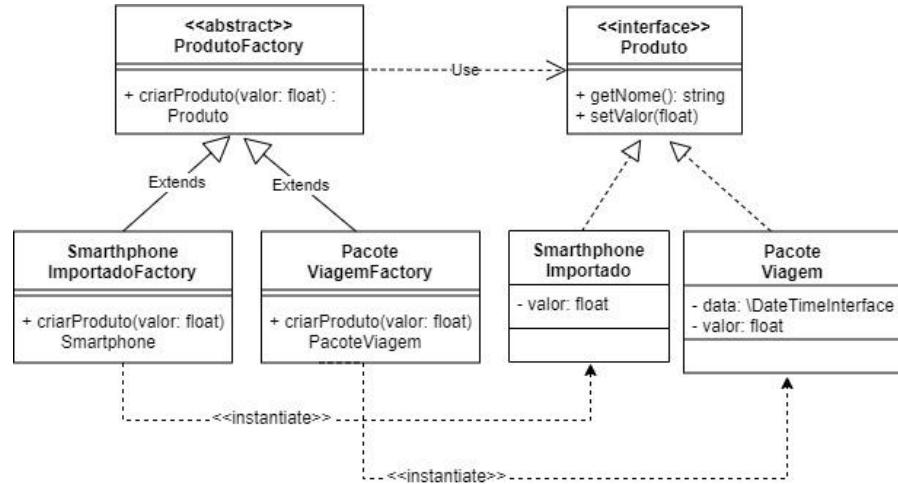
```
abstract class ProdutoFactory {
    public abstract function criarProduto(float $valor): Produto;
}

class SmartphoneImportadoFactory extends ProdutoFactory {
    private const URI_API =
'https://api.exchangeratesapi.io/latest?base=USD&symbols=BRL';
    public function criarProduto(float $valor): SmartphoneImportado {
        $cotacao_json = json_decode(file_get_contents(self::URI_API));
        $cotacao = $cotacao_json->rates->BRL;
        $produto = new SmartphoneImportado();
        $produto->setValor($valor * $cotacao);
        return $produto;
    }
}

class PacoteViagemFactory extends ProdutoFactory {
    const MESES_ALTA_TEMPORADA = [1, 7, 12];
    public function criarProduto(float $valor): PacoteViagem {
        $hoje = new \DateTime();
        $produto = new PacoteViagem($hoje);
        if (in_array($hoje->format('m'), self::MESES_ALTA_TEMPORADA)) {
            $produto->setValor($valor * 2);
        } else {
            $produto->setValor($valor);
        }
        return $produto;
    }
}
```

```
$factory = new SmartphoneImportadoFactory();
$produto = $factory->criarProduto(100);
var_dump($produto);

$factory2 = new PacoteViagemFactory();
$produto2 = $factory2->criarProduto(100);
var_dump($produto2);
```



Abstract Factory

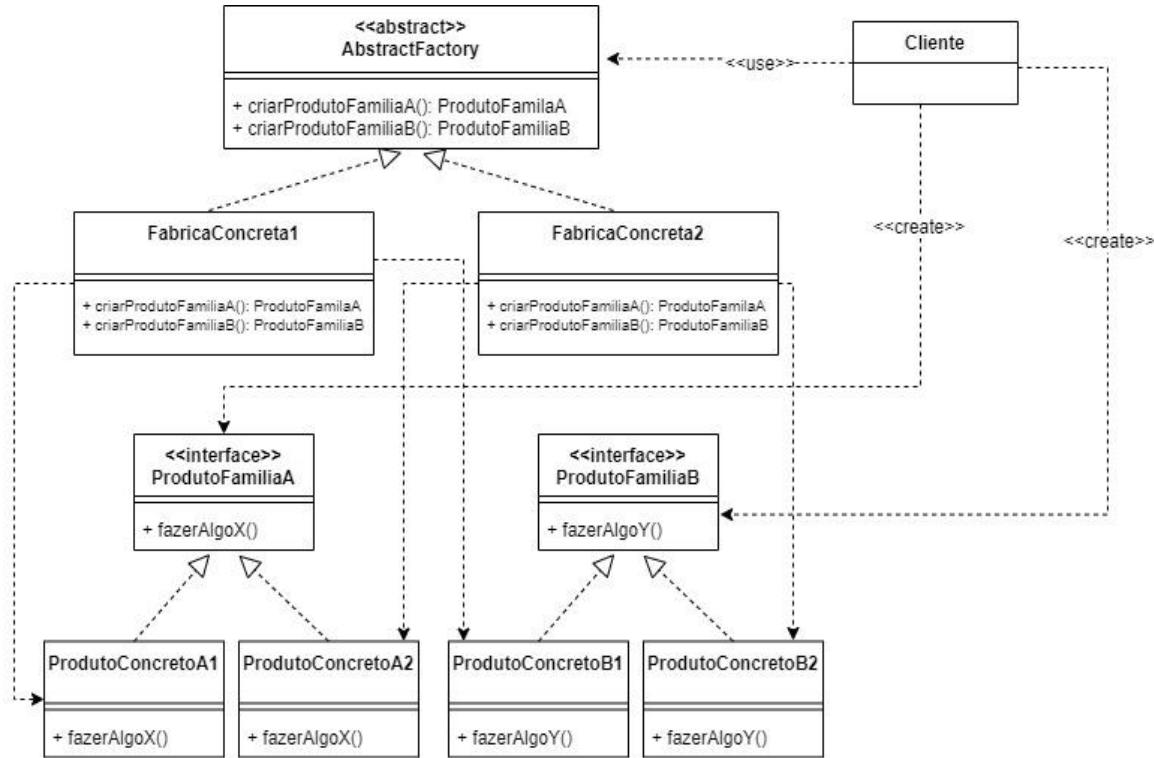
Abstract Factory permite produzir famílias de objetos relacionados sem especificar suas classes concretas.

O cliente não sabe (ou se importa) quais objetos concretos obtém de cada uma dessas fábricas internas, pois usa apenas as interfaces genéricas de seus produtos.



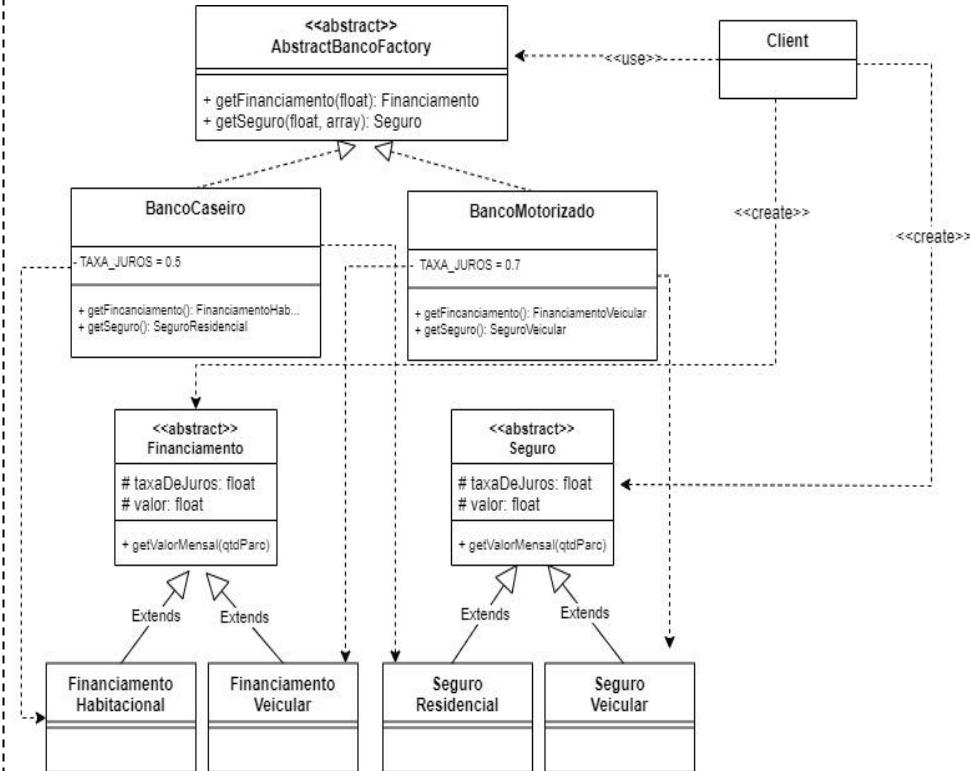
Abstract Factory (conceitual)

```
<?php
interface ProdutoFamiliaA {
    public function fazerAlgoX();
}
class ProdutoConcretoA1 implements ProdutoFamiliaA {
    public function fazerAlgoX() {}
}
class ProdutoConcretoA2 implements ProdutoFamiliaA {
    public function fazerAlgoX() {}
}
interface ProdutoFamiliaB {
    public function fazerAlgoX();
}
class ProdutoConcretoB1 implements ProdutoFamiliaB {
    public function fazerAlgoX() {}
}
class ProdutoConcretoB2 implements ProdutoFamiliaB {
    public function fazerAlgoX() {}
}
abstract class AbstractFactory {
    public abstract function criarProdutoA(): ProdutoFamiliaA;
    public abstract function criarProdutoB(): ProdutoFamiliaB;
}
class FabricaConcreta1 extends AbstractFactory {
    public function criarProdutoA(): \ProdutoConcretoA1 {}
    public function criarProdutoB(): \ProdutoConcretoB1 {}
}
class FabricaConcreta2 extends AbstractFactory {
    public function criarProdutoA(): \ProdutoConcretoA2 {}
    public function criarProdutoB(): \ProdutoConcretoB2 {}
}
class Client {}
```



Abstract Factory (Exemplo)

```
<?php
abstract class Financiamento { //Família A
    protected float $taxaDeJuros;
    protected float $valor;
    public function __construct(float $taxaDeJuros, float $valor) {
        $this->taxaDeJuros = $taxaDeJuros;
        $this->valor = $valor;
    }
    public abstract function getValorMensal(int $quantidadeParcelas): float;
}
class FinanciamentoHabitacional extends Financiamento { //concreto A1
    public function getValorMensal(int $quantidadeParcelas): float {
        $taxa = $this->taxaDeJuros / 100;
        $valParcela = $this->valor * pow((1 + $taxa), $quantidadeParcelas);
        $resultado = $valParcela / $quantidadeParcelas;
        return $resultado;
    }
}
class FinanciamentoVeicular extends Financiamento { //concreto A2
    public function getValorMensal(int $quantidadeParcelas): float {
        $taxa = $this->taxaDeJuros / 100;
        $m = $this->valor * (1 + $taxa * $quantidadeParcelas);
        $resultado = $m / $quantidadeParcelas;
        return $resultado;
    }
}
```

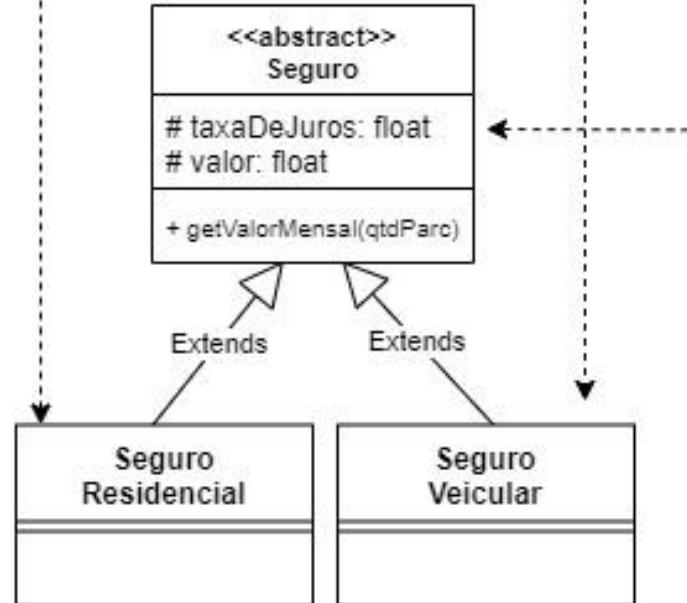


Abstract Factory (Exemplo - parte 2)

```
//Família B
abstract class Seguro {
    protected float $valorAvaliado;
    protected array $coberturas;
    function __construct(float $valorAvaliado, array $coberturas) {
        $this->valorAvaliado = $valorAvaliado;
        $this->coberturas = $coberturas;
    }
    public abstract function getValorMensal();
}

//concreto B1
class SeguroResidencial extends Seguro {
    public function getValorMensal() {
        $adicionais = count($this->coberturas) * 100;
        return (($this->valorAvaliado * 0.01) + $adicionais) / 12;
    }
}

//concreto B2
class SeguroVeicular extends Seguro {
    public function getValorMensal() {
        $adicionais = count($this->coberturas) * 80;
        return (($this->valorAvaliado * 0.1) + $adicionais) / 12;
    }
}
```



Abstract Factory (Exemplo - parte 3)

```
//abstract factory
interface AbstractBancoFactory {
    public function getFinanciamento(float $valor): Financiamento;
    public function getSeguro(float $valorAvaliado, array $coberturas): Seguro;
}

//fábricas concreta A
class BancoCaseiro implements AbstractBancoFactory {
    private const TAXA_JUROS = 0.5;
    public function getFinanciamento(float $valor): FinanciamentoHabitacional {
        return new FinanciamentoHabitacional(self::TAXA_JUROS, $valor);
    }
    public function getSeguro(float $valorAvaliado, array $coberturas): SeguroResidencial {
        return new SeguroResidencial($valorAvaliado, $coberturas);
    }
}

//fábricas concreta B
class BancoMotorizado implements AbstractBancoFactory {
    private const TAXA_JUROS = 0.7;
    public function getFinanciamento(float $valor): FinanciamentoVeicular {
        return new FinanciamentoVeicular(self::TAXA_JUROS, $valor);
    }
    public function getSeguro(float $valorAvaliado, array $coberturas): SeguroVeicular {
        return new SeguroVeicular($valorAvaliado, $coberturas);
    }
}
```

```
class Cliente {
    private AbstractBancoFactory $factory;
    public function __construct(string $tipo) {
        switch ($tipo) {
            case 'casa':
                $this->factory = new BancoCaseiro();
                break;
            case 'veiculo':
                $this->factory = new BancoMotorizado();
                break;
            default:
                new InvalidArgumentException('Opção inválida');
        }
    }
    function getFactory(): AbstractBancoFactory {
        return $this->factory;
    }
}
$cliente = new Cliente('casa');
echo $cliente->getFactory()
    ->getFinanciamento(1000)
    ->getValorMensal(10);
echo "<br/>";
$cliente2 = new Cliente('casa');
echo $cliente2->getFactory()
    ->getSeguro(200000, ['enchente','terremoto'])
    ->getValorMensal();
```

Builder

Builder é um pattern que separa a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar representações diferentes.

Os participantes deste pattern são:

Builder: fornece ao Diretor uma interface para construir um Produto. A interface permite que o Construtor (Builder) oculte a representação e a estrutura interna do produto. Ele também oculta como o produto é montado.

Builder Concreto: contém todo a lógica para criar e montar algum um tipo específico ou variação do Produto.

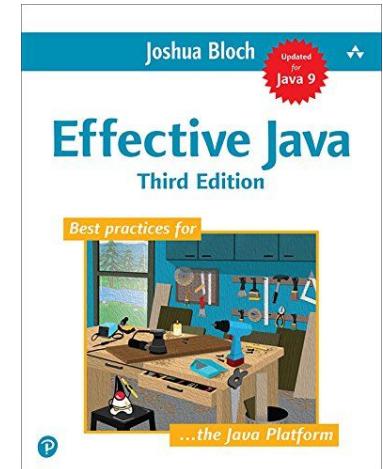
Diretor: responsável por utilizar os Construtores e construir o produto passo a passo sob controle do Diretor, que sabe a ordem certa destes passos pois é ele que notifica o Construtor sempre que uma parte do produto deve ser construída.

Cliente: solicita ao Construtor o Produto ou Resultado da Variação de acordo com o Construtor e o Diretor utilizados.



Builder GoF x Builder Bloch

- Em seu livro intitulado de “Java Effective” (2008), Joshua Bloch propôs uma versão própria de “Builder” baseado no Builder GoF.
- Esta versão tem um foco maior na eliminação de métodos construtores “telescópicos” e o uso de “interface fluente” para a construção de objetos complexos.
- Devido à sinonímia de ambos, é comum à confusão na implementação dos mesmos.



Builder (Conceitual)

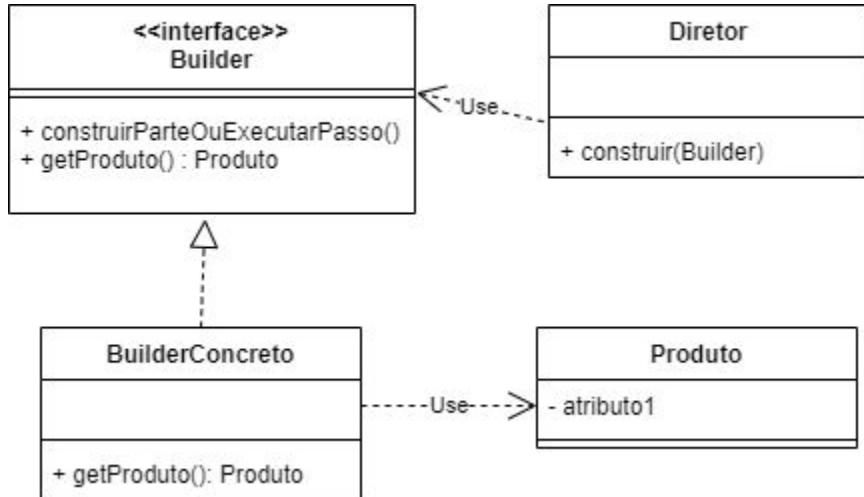
```
<?php
interface Builder {
    public function construirParteOuExecutarPasso();
    public function getProduto() : Produto;
}

class Diretor {
    public function construir(Builder &$builder){
        //executar os passos
    }
}

class BuilderConcreto implements Builder {
    private Produto $produto;
    public function __construct() {
        $this->produto = new Produto();
    }
    public function construirParteOuExecutarPasso(){
        //produto->variacao
    }
    public function getProduto(): Produto {
        return $this->produto;
    }
}

class Produto {
    private $atributo;
}

$builder = new BuilderConcreto();
$diretor = new Diretor();
$diretor->construir($builder);
var_dump($builder->getProduto());
```



Builder (Exemplo - parte 1)

```
abstract class Builder {  
  
    protected string $resultado = "";  
  
    abstract function incluirCabecalho(array $header);  
    abstract function incluirLinha(array $line);  
    abstract function finalizar();  
  
    public function getResultado() : string {  
        return $this->resultado;  
    }  
}
```

```
abstract class Diretor {  
  
    protected Builder $builder;  
  
    public function __construct(Builder $builder) {  
        $this->builder = $builder;  
    }  
    public abstract function construir(string $inputFileName);  
}
```

Builder (Exemplo - parte 2)

```
class HtmlBuilder extends Builder {

    private \DOMDocument $document;
    private \DOMElement $table;

    public function __construct() {
        $this->document = new \DOMDocument('1.0', 'utf-8');
        $this->document->appendChild($this->document->createElement('html'));
        $this->table = $this->document->createElement('table');
        $this->table->setAttribute('border', 1);
        $this->document->firstChild->appendChild($this->table);
    }

    private function criarTableRow(array $line, $tipo = 'td'){
        $tr = $this->document->createElement('tr');
        array_map(fn($v) =>
            $tr->appendChild($this->document->createElement($tipo, $v)),
            $line);
        $this->table->appendChild($tr);
    }

    public function incluirCabecalho(array $header) {
        $this->criarTableRow($header, 'th');
    }

    public function incluirLinha(array $line) {
        $this->criarTableRow($line);
    }

    public function finalizar() {
        $this->resultado = $this->document->saveHTML();
    }
}
```

```
class CsvBuilder extends Builder {

    private array $csvArray = [];

    public function incluirCabecalho(array $header) {
        $this->csvArray[] = $header;
    }

    public function incluirLinha(array $line) {
        $this->csvArray[] = $line;
    }

    public function finalizar() {
        foreach ($this->csvArray as $line) {
            $this->resultado.= implode(",",$line).PHP_EOL;
        }
    }
}
```

Builder (Exemplo - parte 3)

```
class DiretorXml extends Diretor {  
  
    public function construir(string $inputFileName) {  
        $document = new \DOMDocument();  
        $document->preserveWhiteSpace = false;  
        $document->load($inputFileName);  
        $root = $document->firstChild;  
        $item1 = iterator_to_array($root->firstChild->childNodes);  
        $this->builder->incluirCabecalho(array_column($item1, 'tagName'));  
        foreach($root->childNodes as $child){  
            $item = iterator_to_array($child->childNodes);  
            $this->builder->incluirLinha(array_column($item1, 'nodeValue'));  
        }  
        $this->builder->finalizar();  
    }  
}
```

```
class DiretorJson extends Diretor {  
  
    public function construir(string $inputFileName) {  
        $jsonArray = json_decode(file_get_contents($inputFileName));  
        $this->builder->incluirCabecalho(array_keys((array) $jsonArray[0]));  
        foreach ($jsonArray as $jsonObject) {  
            $this->builder->incluirLinha((array) $jsonObject);  
        }  
        $this->builder->finalizar();  
    }  
}
```

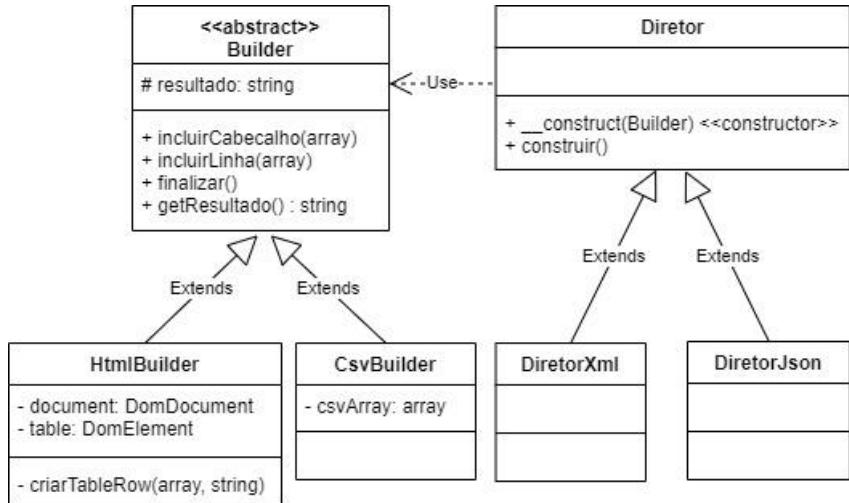
Builder (Exemplo - parte 4)

```
//json to html
https://gist.github.com/celsowm/45a9e92eb29c4580f971d528a0b61ab9
$input = "clientes.json";
$builder = new HtmlBuilder();
$diretor = new DiretorJson($builder);
$diretor->construir($input);
file_put_contents("clientes.html", $builder->getResultado());

//json to csv
$builder = new CsvBuilder();
$diretor = new DiretorJson($builder);
$diretor->construir($input);
file_put_contents("clientes.csv", $builder->getResultado());

//xml to html
https://gist.github.com/celsowm/4ad318f0217953e550e9694baab8aa37
$input2 = "clientes.xml";
$builder = new HtmlBuilder();
$diretor = new DiretorXml($builder);
$diretor->construir($input2);
file_put_contents("clientes2.html", $builder->getResultado());

//xml to csv
$builder = new CsvBuilder();
$diretor = new DiretorJson($builder);
$diretor->construir($input);
file_put_contents("clientes2.csv", $builder->getResultado());
```

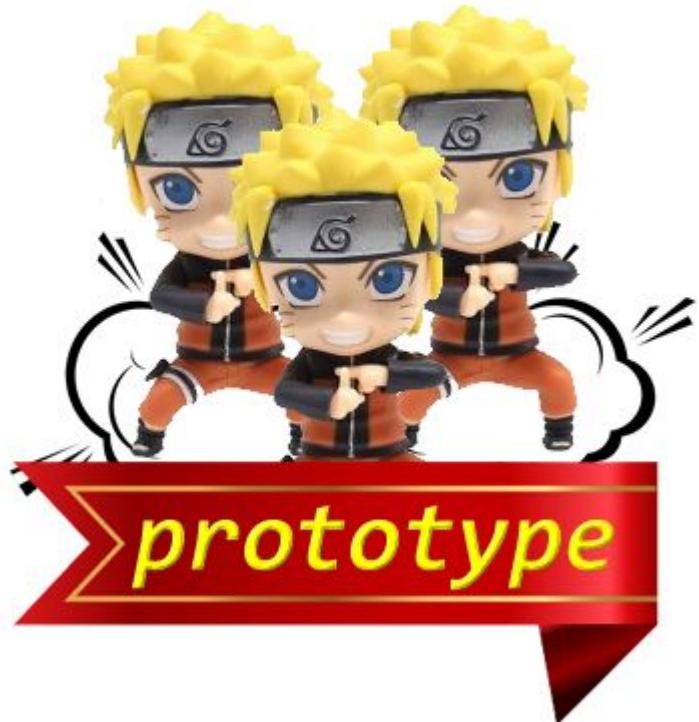


Prototype

Protótipo é um pattern que visa permitir copiar (clonar) objetos existentes sem tornar seu código dependente de suas classes (SHVETS, 2019).

O padrão Prototype delega o processo de clonagem para os objetos reais que estão sendo clonados. O padrão declara uma interface comum para todos os objetos que suportam a clonagem. Essa interface permite clonar um objeto sem acoplar seu código à classe desse objeto. (SHVETS, 2019).

Em muitas aplicações, faz-se necessário criar cópias de um mesmo objetos inúmeras vezes. Então, faz sentido economizar na criação dos mesmos, principalmente se a criação exigir muito tempo ou recursos.



Método “mágico” `__clone`

```
<?php

class Pessoa {

    public string $cpf;
    public string $nome;

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

    public function __clone(){
        $this->nome.= " (cópia)";
    }
}

$p1 = new Pessoa("José","123");
$p2 = clone $p1;
var_dump($p2);
```

- Além da instrução `clone`, o PHP permite que instruções customizadas possam ser executadas durante o processo de clonagem através do método mágico `__clone()`.
- Quando o mesmo é implementado em um determinada classe, o código contido neste método será executado no momento que algum objeto desta classe for clonada.
- Se o método mágico `__clone` for definido como abstrato em uma classe abstrata, as classes concretas derivadas serão obrigadas a implementá-lo.

Shallow Copy vs Deep Copy

```
<?php
class Aluno {
    public string $nome;
    public string $matricula;
    public function __construct(string $nome, string
$matricula){
        $this->nome = $nome;
        $this->matricula = $matricula;
    }
}
abstract class Turma {
    public string $codigo;
    public $alunos = [];
    public function __construct(string $codigo){
        $this->codigo = $codigo;
    }
}
class ShallowTurma extends Turma {}
class DeepTurma extends Turma {
    public function __clone(){
        $this->alunos = array_map(fn ($o) => clone $o,
$this->alunos);
    }
}
```

Existem duas estratégias básicas no processo de copiar um objeto:

Cópia rasa (shallow): as possíveis referências para outros objetos que o objeto original dispunha são simplesmente duplicadas e repassadas ao novo objeto clonado.

Cópia profunda (deep): os objetos que se relacionam com o objeto a ser clonado também são clonados e não apenas referenciados em duplicidade.

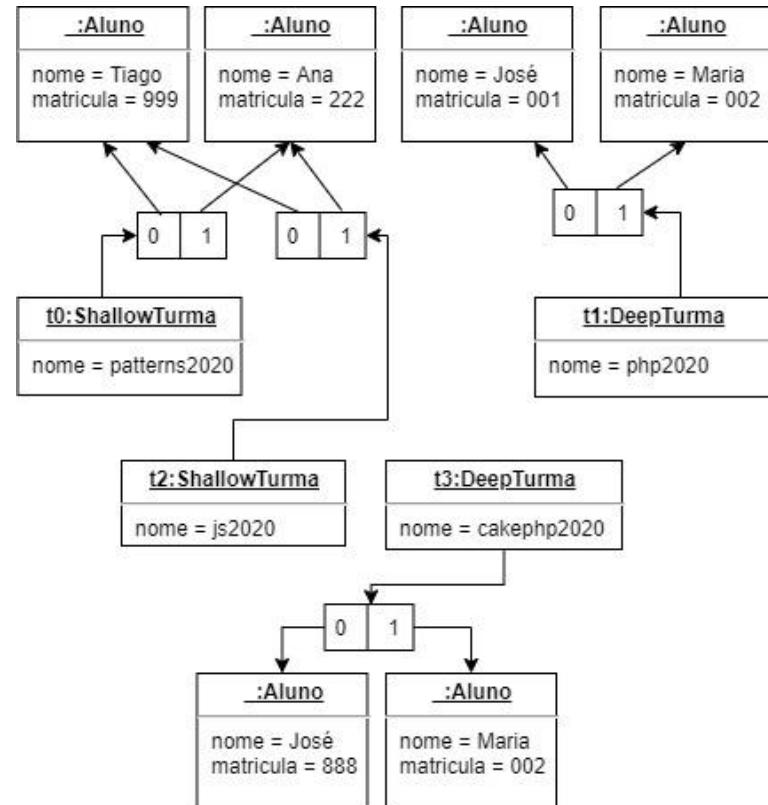
Shallow Copy vs Deep Copy (parte 2)

```
$t0 = new ShallowTurma("patterns2020");
$t1 = new DeepTurma("php2020");
$t0->alunos = [
    new Aluno("Tiago", "111"),
    new Aluno("Ana", "222")
];
$t1->alunos = [
    new Aluno("José", "001"),
    new Aluno("Maria", "002")
];

$t2 = clone $t0;
$t3 = clone $t1;

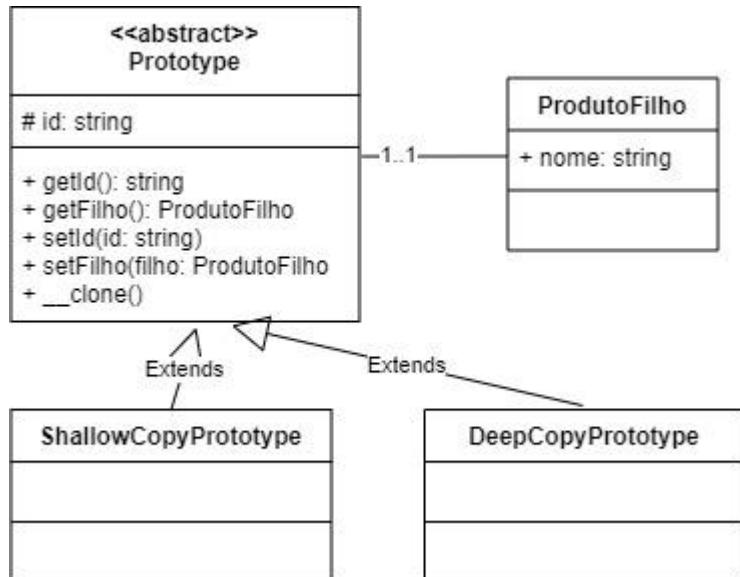
$t2->codigo = "js2020";
$t2->alunos[0]->matricula = "999";

$t3->codigo = "cakephp2020";
$t3->alunos[0]->matricula = "888";
```



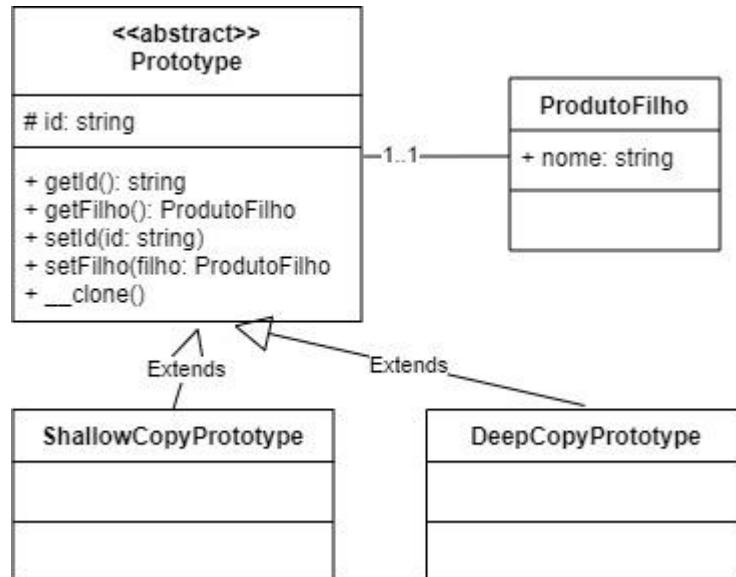
Prototype (conceitual)

```
<?php
class ProdutoFilho {
    public string $nome;
    function __construct(string $nome) {
        $this->nome = $nome;
    }
}
abstract class Prototype {
    protected string $id;
    protected ProdutoFilho $filho;
    function __construct(string $id, ProdutoFilho $filho) {
        $this->id = $id;
        $this->filho = $filho;
    }
    function getId(): string {
        return $this->id;
    }
    function getFilho(): ProdutoFilho {
        return $this->filho;
    }
    function setId(string $id): void {
        $this->id = $id;
    }
    function setFilho(ProdutoFilho $filho): void {
        $this->filho = $filho;
    }
    abstract function __clone();
}
```



Prototype (conceitual - parte 2)

```
class ShallowCopyPrototype extends Prototype {  
    public function __clone() {}  
}  
  
class DeepCopyPrototype extends Prototype {  
    public function __clone() {  
        $this->setFilho(clone $this->getFilho());  
    }  
}  
  
$p1 = new ShallowCopyPrototype('p001', new ProdutoFilho('filho1'));  
$p2 = new DeepCopyPrototype('p002', new ProdutoFilho('filho2'));  
$p3 = clone $p1;  
$p4 = clone $p2;  
$p3->getFilho()->nome = 'filho 3';  
$p4->getFilho()->nome = 'filho 4';  
var_dump($p1);  
var_dump($p2);  
var_dump($p3);  
var_dump($p4);
```



Prototype (exemplo - parte 1)

```
<?php
abstract class TermoContratual {

    private string $nomeContratada;
    private string $numeroContrato;
    private string $template;

    function __construct(string $nomeContratada, string $numeroContrato) {
        $this->nomeContratada = $nomeContratada;
        $this->numeroContrato = $numeroContrato;
        $this->template = file_get_contents($this->getURITemplate());
    }

    function getNomeContratada(): string {
        return $this->nomeContratada;
    }

    function setNomeContratada(string $nomeContratada) {
        $this->nomeContratada = $nomeContratada;
    }

    function setNumeroContrato(string $numeroContrato) {
        return $this->numeroContrato = $numeroContrato;
    }

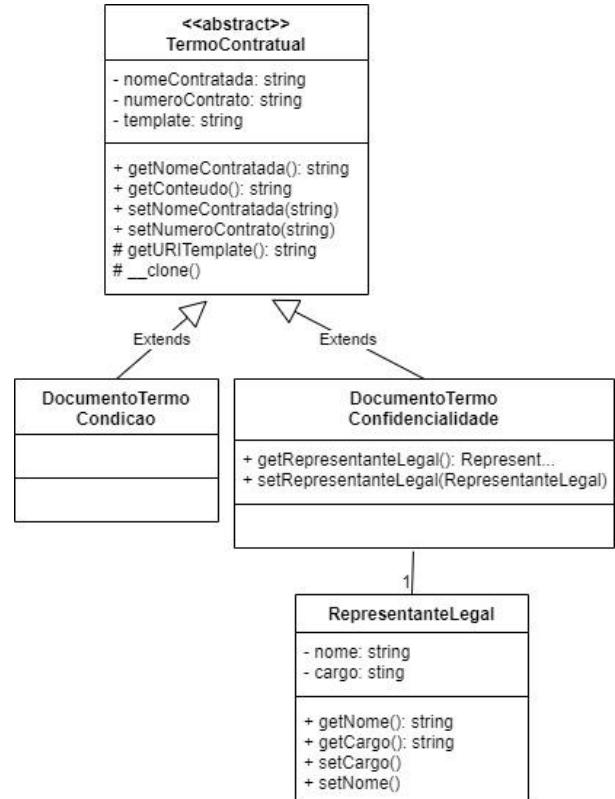
    function getConteudo(): string {
        $hashes = ["#contratada", "#numero_contrato", "#data"];
        $replaces = [$this->nomeContratada, $this->numeroContrato, (new \DateTime())->format("d/m/Y")];
        return str_replace($hashes, $replaces, $this->template);
    }

    abstract protected function getURITemplate() : string;
    public abstract function __clone();
}
```

<<abstract>>	
TermoContratual	
-	nomeContratada: string
-	numeroContrato: string
-	template: string
+	getNomeContratada(): string
+	getConteudo(): string
+	setNomeContratada(string)
+	setNumeroContrato(string)
#	getURITemplate(): string
#	__clone()

Prototype (exemplo - parte 2)

```
class DocumentoTermoCondicao extends TermoContratual {  
  
    public function __clone() {}  
    protected function getURITemplate(): string {  
        return "https://pastebin.com/raw/JNc3NVDy";  
    }  
  
}  
  
class RepresentanteLegal {  
    private string $nome;  
    private string $cargo;  
    function __construct(string $nome, string $cargo) {  
        $this->nome = $nome;  
        $this->cargo = $cargo;  
    }  
    function getName(): string {  
        return $this->nome;  
    }  
    function getCargo(): string {  
        return $this->cargo;  
    }  
    public function setNome(string $nome): void {  
        $this->nome = $nome;  
    }  
    public function setCargo(string $cargo): void {  
        $this->cargo = $cargo;  
    }  
}
```



Prototype (exemplo - parte 3)

```
class DocumentoTermoConfidencialidade extends TermoContratual {

    private RepresentanteLegal $representanteLegal;

    public function __construct(string $nomeContratada, string $numeroContrato, RepresentanteLegal $representanteLegal) {
        parent::__construct($nomeContratada, $numeroContrato);
        $this->representanteLegal = $representanteLegal;
    }

    public function setRepresentanteLegal(RepresentanteLegal $representanteLegal) {
        $this->representanteLegal = $representanteLegal;
    }

    public function getRepresentanteLegal(): RepresentanteLegal{
        return $this->representanteLegal;
    }

    public function getConteudo(): string {
        return str_replace("#representante",$this->representanteLegal->getNome(),parent::getConteudo());
    }

    public function __clone(){
        $this->representanteLegal = clone $this->representanteLegal;
    }

    protected function getURITemplate(): string {
        return "https://pastebin.com/raw/fvxp0W6Z";
    }
}
```

Prototype (exemplo - parte 4)

```
<?php
$docTC = new DocumentoTermoCondicao("Lorem Ipsum", "0000000");

$docConf = new DocumentoTermoConfidencialidade("Lorem Ipsum", "11111111",
    new RepresentanteLegal("Fulano da Silva", "Cargo de Exemplo"));

$cloneTC  = clone $docTC;
$cloneConf = clone $docConf;

$cloneTC->setNumeroContrato("2222222");
$cloneConf->setNumeroContrato("3333333");
$cloneConf->getRepresentanteLegal()->setNome("Beltrano da Silva");

echo "originais:";
var_dump($docTC);
var_dump($docConf);
echo "cópias:";
var_dump($cloneTC);
var_dump($cloneConf);
```

Singleton

O Singleton é um padrão de design de software que restringe a instanciação de uma classe a uma "única instância". Isso é algo que pode ser considerado útil quando exatamente um objeto é necessário para coordenar diversas ações no sistema.

Normalmente, isso é feito através da declaração de todos os construtores da classe como **privados**; e fornecendo um método estático que retorna uma referência à instância.

(WIKIPEDIA, 2020)



Singleton (conceitual)

```
<?php

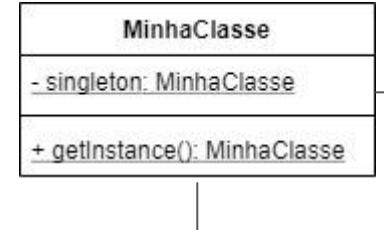
class MinhaClasse {

    private static $singleton;

    private function __construct() {}

    public static function getInstance(): ?self {
        if (self::$singleton == null) {
            self::$singleton = new self();
        }
        return self::$singleton;
    }

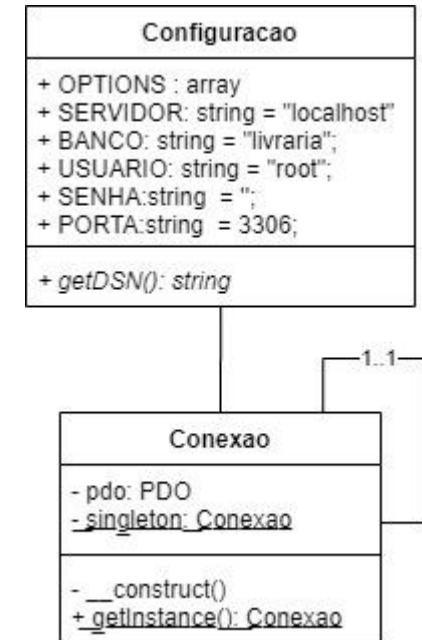
}
```



1..1

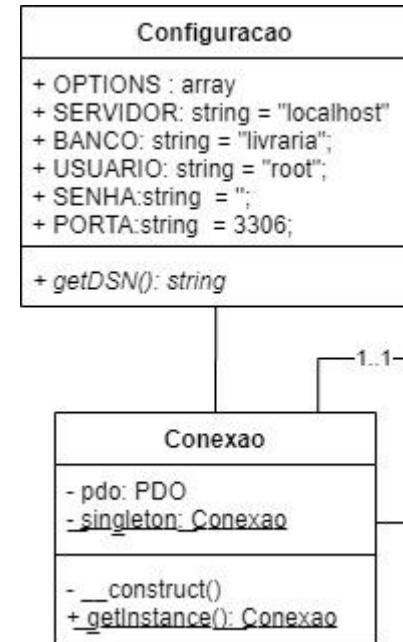
Singleton (exemplo - parte 1)

```
<?php
class Configuracao {
    const OPTIONS = [
        \PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION,
        \PDO::ATTR_DEFAULT_FETCH_MODE => \PDO::FETCH_ASSOC,
        \PDO::ATTR_EMULATE_PREPARES => false
    ];
    const SERVIDOR = "localhost";
    const BANCO = "livraria";
    const USUARIO = "root";
    const SENHA = "";
    const PORTA = 3306;
    public static function getDSN(): string {
        return "mysql:host=".self::SERVIDOR.";port=".self::PORTA.";dbname=".self::BANCO.";charset=utf8";
    }
}
```



Singleton (exemplo - parte 2)

```
class Conexao {
    private static $singleton;
    private \PDO $pdo;
    private function __construct(\PDO $pdo) {
        $this->pdo = $pdo;
    }
    public static function getInstance(): ?self {
        if (self::$singleton == null) {
            self::$singleton = new self(
                new \PDO(
                    Configuracao::getDSN(),
                    Configuracao::USUARIO,
                    Configuracao::SENHA,
                    Configuracao::OPTIONS
                ));
        }
        return self::$singleton;
    }
    public function getPdo(): ?\PDO{
        return $this->pdo;
    }
}
var_dump(Conexao::getInstance()->
    getPdo()->query("SELECT * FROM livro")->fetchAll());
```



Design Pattern: Padrões Estruturais

- Os padrões de design estrutural auxiliam na maneira de como as classes e objetos podem ser **compostos** para formar estruturas maiores, mantendo essas estruturas **flexíveis e eficientes** (SHVETS, 2019).
- Os padrões de design estrutural simplificam a estrutura, identificando os **relacionamentos** (JAVAPOINT, 2020).

Adapter

O Adapter é um pattern estrutural que visa permitir que objetos com interfaces incompatíveis colaborem.

Um adaptador (adapter) “empacota” um dos objetos para ocultar a complexidade da conversão que ocorre nos bastidores. O objeto empacotado nem mesmo tem conhecimento da adaptação.

(SHEVTS, 2019).



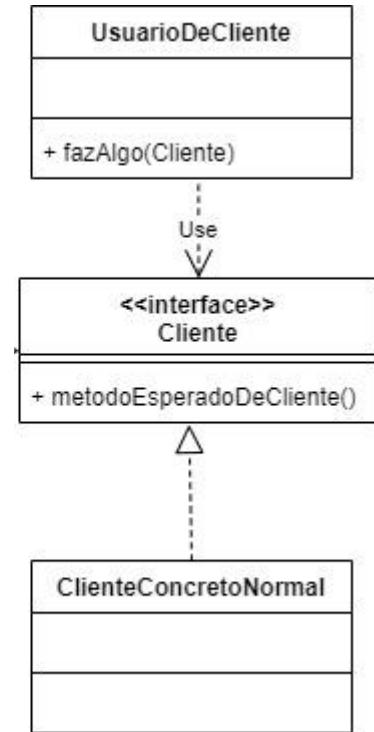
Adapter (conceitual - parte 1)

```
<?php

interface Cliente {
    public function metodoEsperadoDeCliente();
}

class UsuarioDeCliente {
    public static function fazAlgo(Cliente $cliente){
        $cliente->metodoEsperadoDeCliente();
    }
}

class ClienteConcretoNormal implements Cliente {
    public function metodoEsperadoDeCliente() {
        echo "oi!";
    }
}
```



Adapter (conceitual - parte 2)

```
<?php
class ClasseTerceiro {

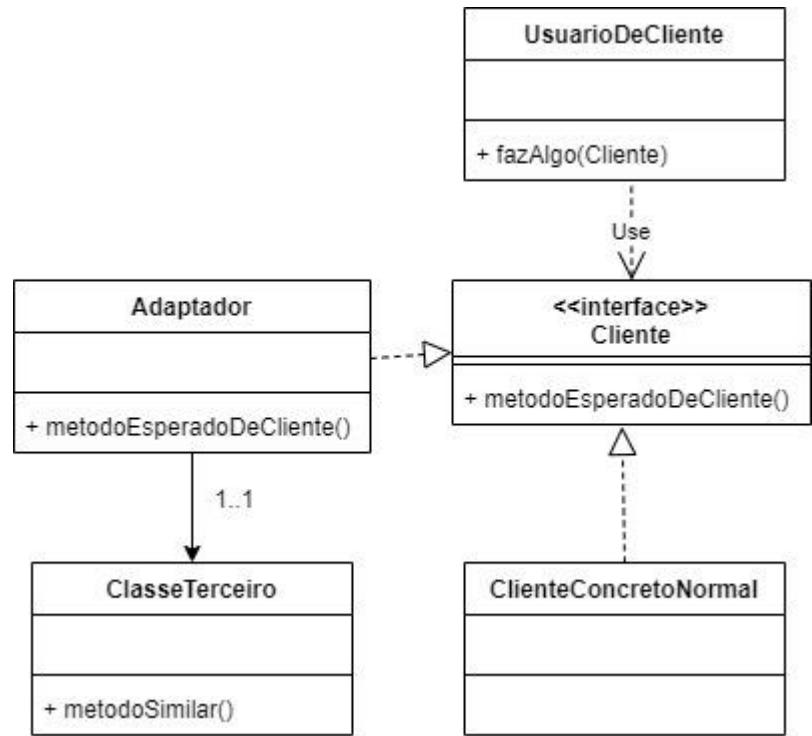
    public function metodoSimilar(){
        echo "oiii!!!";
    }
}

class Adaptador implements Cliente {
    private ClasseTerceiro $adaptado;

    public function __construct(Cliente $adapatado){
        $this->adaptado = $adapatado;
    }

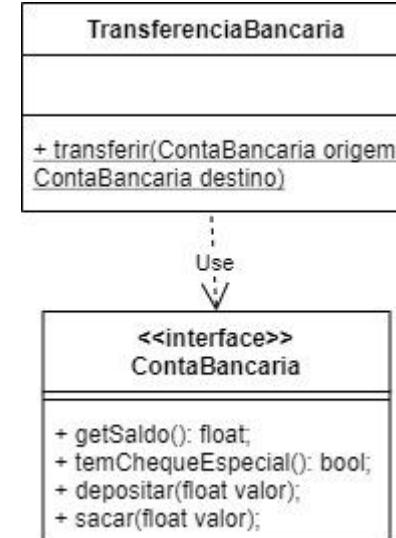
    public function metodoEsperadoDeCliente() {
        $this->adaptado->metodoSimilar();
    }
}

$normal      = new ClienteConcretoNormal();
$incompativel = new ClasseTerceiro();
$adaptador   = new Adaptador($incompativel);
UsuarioDeCliente::fazAlgo($normal); //ok
UsuarioDeCliente::fazAlgo($adaptador); //ok tb
```



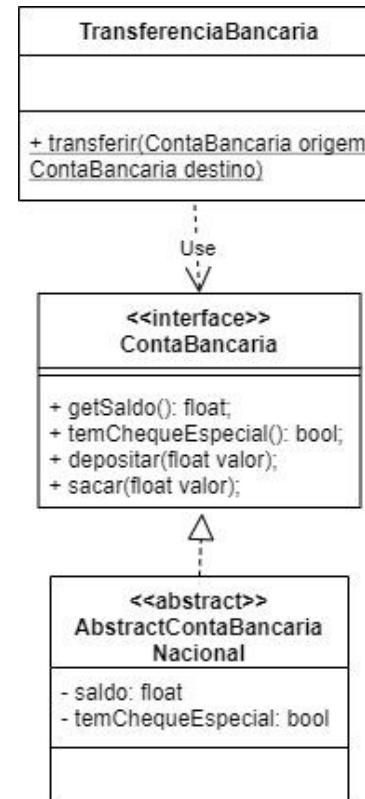
Adapter (exemplo - parte1)

```
<?php  
interface ContaBancaria {  
    public function getSaldo(): float;  
    public function temChequeEspecial(): bool;  
    public function depositar(float $valor);  
    public function sacar(float $valor);  
}  
  
class TransferenciaBancaria {  
    public static function transferir(float $valor, ContaBancaria  
$origem, ContaBancaria $destino){  
        if($origem->getSaldo() >= $valor){  
            $origem->sacar($valor);  
            $destino->depositar($valor);  
        }  
    }  
}
```



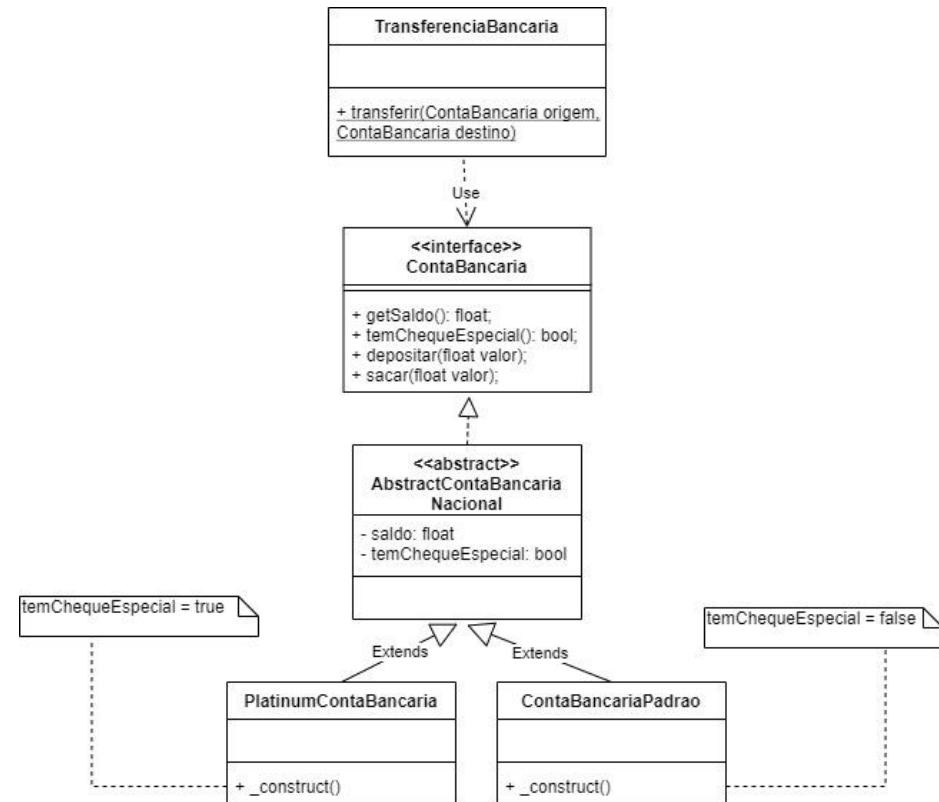
Adapter (exemplo - parte 2)

```
abstract class AbstractContaBancariaNacional implements ContaBancaria {  
  
    private float $saldo;  
    private bool $temChequeEspecial;  
  
    public function __construct(float $saldo) {  
        $this->saldo = $saldo;  
    }  
    public function getSaldo(): float {  
        return $this->saldo;  
    }  
    public function temChequeEspecial(): bool {  
        return $this->temChequeEspecial;  
    }  
    public function setChequeEspecial(bool $temChequeEspecial) {  
        $this->temChequeEspecial = $temChequeEspecial;  
    }  
    public function depositar(float $valor) {  
        $this->saldo += $valor;  
    }  
    public function sacar(float $valor) {  
        if ($this->saldo >= $valor || $this->temChequeEspecial()) {  
            $this->saldo -= $valor;  
        }  
    }  
}
```



Adapter (exemplo - parte 3)

```
class PlatinumContaBancaria extends AbstractContaBancariaNacional {  
  
    public function __construct(float $saldo) {  
        parent::__construct($saldo);  
        $this->setChequeEspecial(true);  
    }  
  
}  
  
class ContaBancariaPadrao extends AbstractContaBancariaNacional {  
  
    public function __construct(float $saldo) {  
        parent::__construct($saldo);  
        $this->setChequeEspecial(false);  
    }  
  
}
```



Adapter (exemplo - parte 4)

```

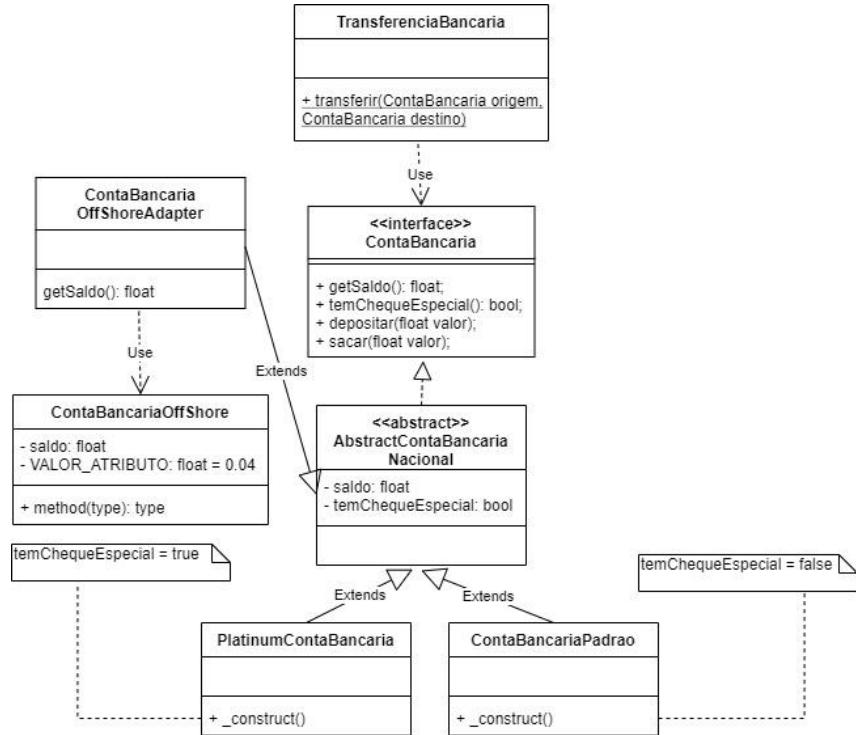
<?php
class ContaBancariaOffShore {
    private float $saldo;
    private float $valorTributo;
    public function __construct(float $saldo, float $valorTributo) {
        $this->saldo = $saldo;
        $this->valorTributo = $valorTributo;
    }
    public function getValorTributo(): float {
        return $this->valorTributo;
    }
    public function getOffshoreSaldo(): float {
        return $this->saldo;
    }
    public function depositar(float $valor) {
        $this->saldo += $valor;
    }
}

class ContaBancariaOffShoreAdapter extends AbstractContaBancariaNacional {

    private ContaBancariaOffShore $contaBancariaOffShore;

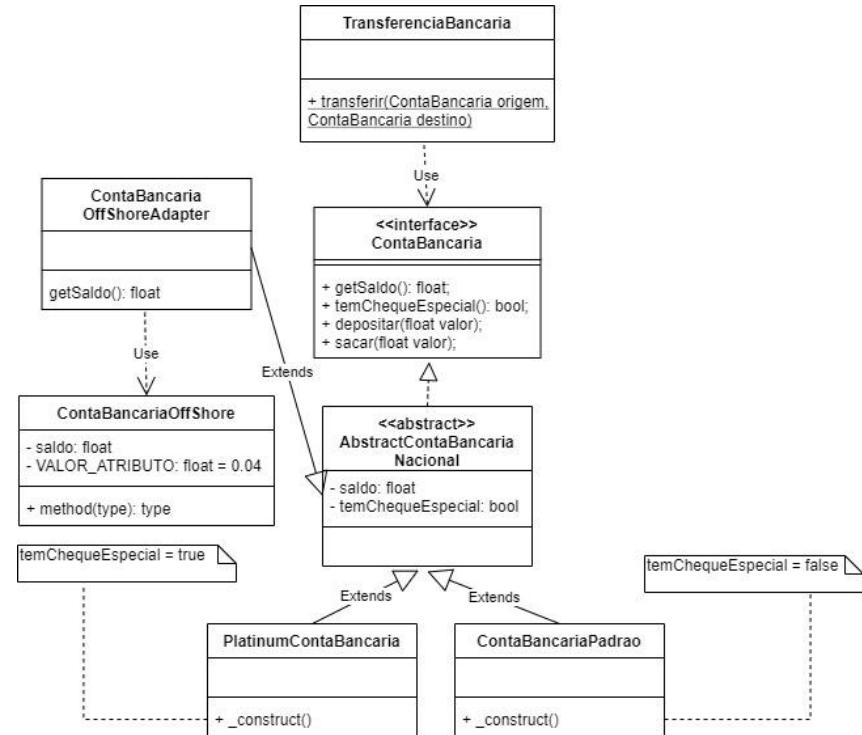
    public function __construct(ContaBancariaOffShore $contaBancariaOffShore) {
        parent::__construct($contaBancariaOffShore->getOffshoreSaldo());
        $this->contaBancariaOffShore = $contaBancariaOffShore;
    }
    public function getSaldo(): float {
        $valor_tributo = $this->contaBancariaOffShore->getValorTributo();
        $bruto = parent::getSaldo();
        $saldo_com_taxes = $bruto * $valor_tributo;
        $liquido = $bruto - $saldo_com_taxes;
        return $liquido;
    }
}

```



Adapter (exemplo - parte 5)

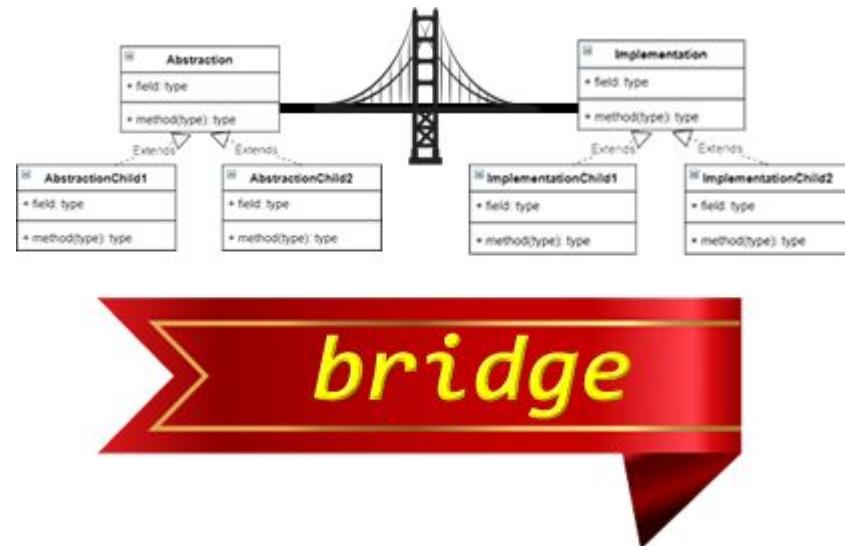
```
$origem = new ContaBancariaPadrao(2000);
$destino = new ContaBancariaOffShoreAdapter(
    new ContaBancariaOffShore(2000, 0.04));
TransferenciaBancaria::transferir(100, $origem, $destino);
var_dump($origem->getSaldo());
var_dump($destino->getSaldo());
```



Bridge

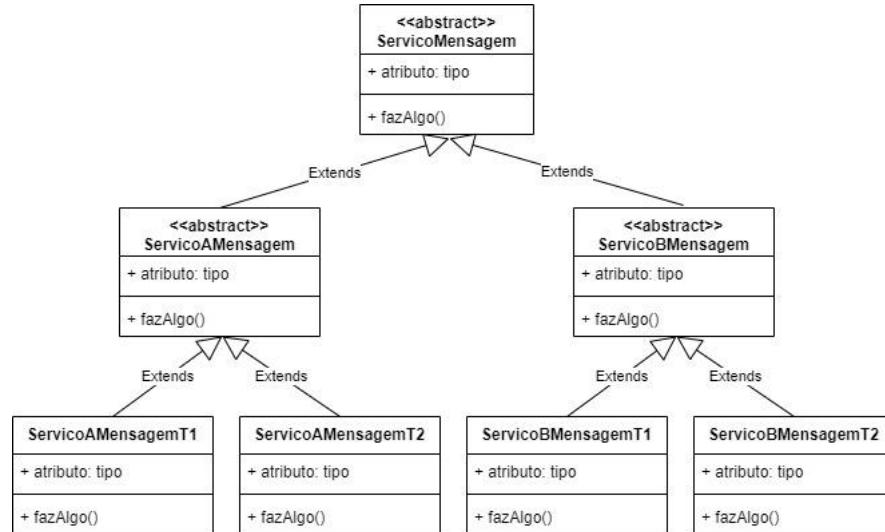
O Bridge é um pattern estrutural que permite dividir uma classe grande ou um conjunto de classes estreitamente relacionadas em duas hierarquias separadas: **abstração** e **implementação**, que podem ser desenvolvidas independentemente uma da outra (SHVETS, 2019).

O Bridge é consideravelmente útil quando a classe e o que ela faz variam com freqüência. A própria classe pode ser considerada como a abstração e o que a classe pode fazer como a implementação. O Bridge também pode ser pensado como duas camadas de abstração (WIKIPEDIA, 2020)



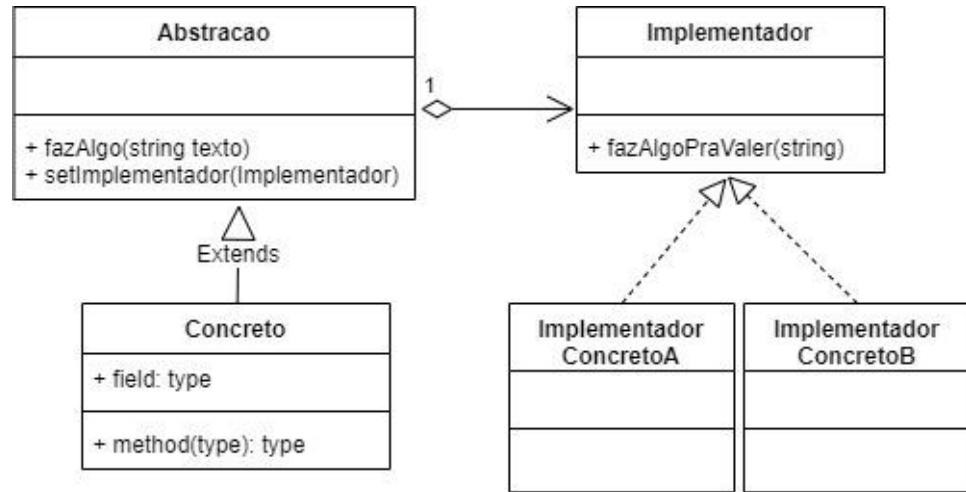
Herança e Bridge

- Usualmente, o recurso da herança é utilizado como forma de abstração para separar o código do cliente de suas implementações.
- Definimos uma interface ou uma classe abstrata e criamos hierarquias de herança, uma para cada uma das várias implementações possíveis.
- Abstrações por herança nem sempre são flexíveis. Quando usamos herança, vinculamos permanentemente a implementação à abstração. Como resultado, qualquer alteração feita em um afeta o outro.
- Uma maneira mais flexível é separar a abstração e a implementação, e é aí que entra o padrão da ponte.
- O Bridge faz isso separando a abstração e a implementação em hierarquias de classes separadas. A ponte entre as hierarquias de classe é alcançada através da agregação (Thompson, 2020);



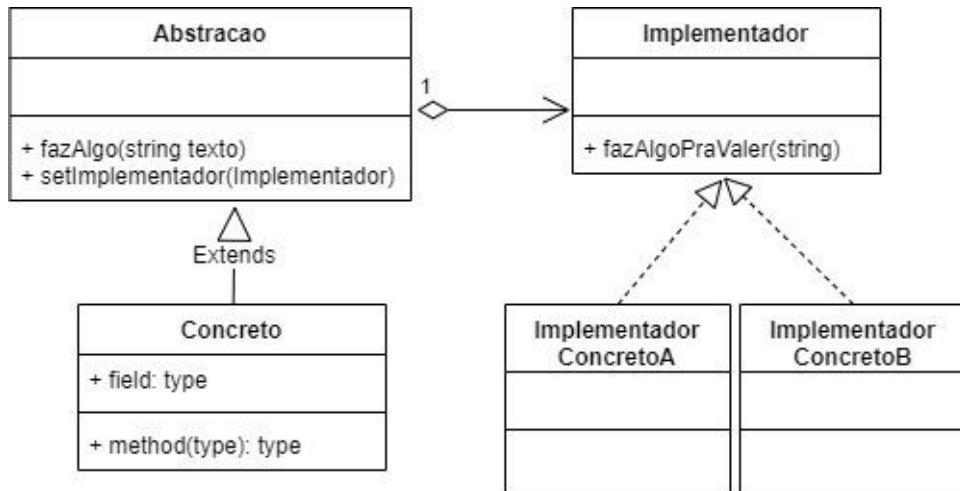
Bridge (conceitual - parte 1)

```
abstract class Abstracao {  
  
    private Implementador $implementador; //bridge  
    public function __construct(Implementador $implementador) {  
        $this->implementador = $implementador;  
    }  
    public function setImplementador(Implementador $implementador){  
        $this->implementador = $implementador;  
    }  
    public function fazAlgo(string $texto){  
        $this->implementador->fazAlgoPraValer($texto);  
    }  
}  
  
interface Implementador {  
    public function fazAlgoPraValer(string $texto);  
}  
  
class Concreto extends Abstracao {  
  
    public function fazAlgo(string $texto) {  
        parent::fazAlgo($texto);  
    }  
}  
  
class ImplementadorConcretoA implements Implementador {  
    public function fazAlgoPraValer(string $texto) {  
        echo "fiz $texto!";  
    }  
}
```



Bridge (conceitual - parte 2)

```
class ImplementadorConcretoB implements Implementador {  
  
    public function fazAlgoPraValer(string $texto) {  
        echo "fiz $texto diferente !";  
    }  
  
    $concreto = new Concreto(new ImplementadorConcretoA());  
    $concreto->fazAlgo("nada");  
    $concreto->setImplementador(new ImplementadorConcretoB());  
    $concreto->fazAlgo("algo");
```



Bridge (exemplo - parte 1)

```
<?php

abstract class Mensagem {

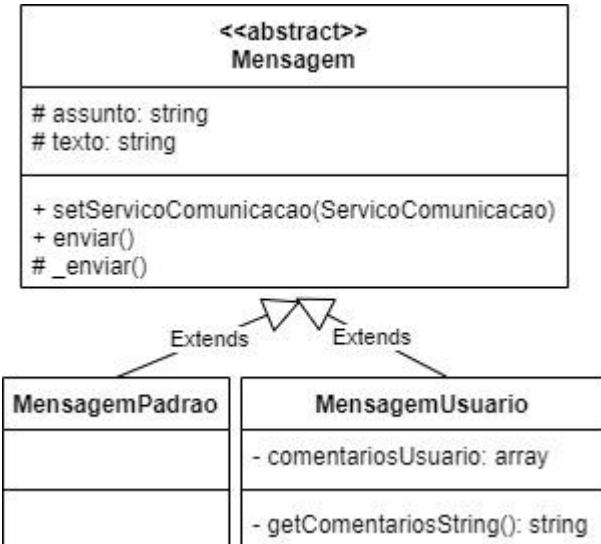
    protected ServicoComunicacao $servicoComunicacao;
    protected string $assunto;
    protected string $texto;

    function __construct(string $assunto, string $texto) {
        $this->assunto = $assunto;
        $this->texto = $texto;
    }
    function setServicoComunicacao(ServicoComunicacao $servicoComunicacao) {
        $this->servicoComunicacao = $servicoComunicacao;
    }
    public final function enviar() {
        if ($this->servicoComunicacao) {
            $this->_enviar();
        } else {
            new \Exception('Sem serviço setado');
        }
    }
    protected abstract function _enviar();
}
```

<<abstract>>
Mensagem
assunto: string
texto: string
+ setServicoComunicacao(ServicoComunicacao)
+ enviar()
_enviar()

Bridge (exemplo - parte 2)

```
class MensagemPadrao extends Mensagem {  
  
    protected function _enviar() {  
        $this->servicoComunicacao->enviarMensagem($this->assunto, $this->texto);  
    }  
  
}  
  
class MensagemUsuario extends Mensagem {  
  
    private array $comentariosUsuario;  
  
    function __construct(string $assunto, string $texto, array $comentariosUsuario) {  
        parent::__construct($assunto, $texto);  
        $this->comentariosUsuario = $comentariosUsuario;  
    }  
  
    private function getComentariosString(): string {  
        return implode("<br/>", $this->comentariosUsuario);  
    }  
  
    protected function _enviar() {  
        $fullTexto = sprintf("%s <br/>comentários:<br/> %s", $this->texto,  
        $this->getComentariosString());  
        $this->servicoComunicacao->enviarMensagem($this->assunto, $fullTexto);  
    }  
  
}
```



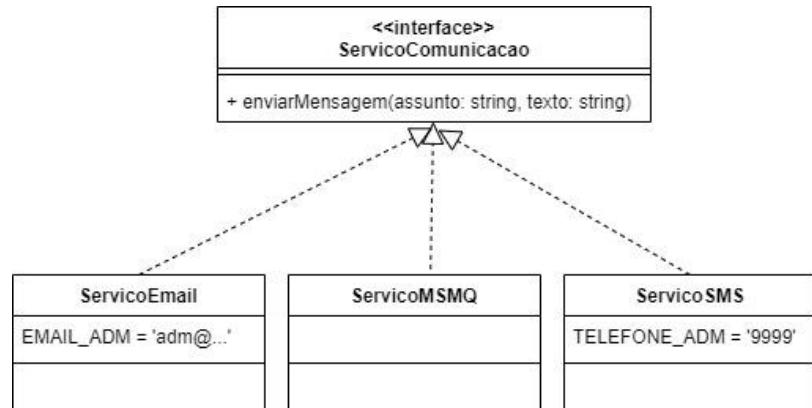
Bridge (exemplo - parte 3)

```
interface ServicoComunicacao {
    public function enviarMensagem(string $assunto, string $texto);
}

class ServicoEmail implements ServicoComunicacao {
    const EMAIL_ADMIN = 'adm@example.com';
    public function enviarMensagem(string $assunto, string $texto) {
        //mail(self::EMAIL_ADMIN, $assunto, $texto); //implementação real
        printf("Email %s %s <br/>", $assunto, $texto);
    }
}

class ServicoMSMQ implements ServicoComunicacao {
    public function enviarMensagem(string $assunto, string $texto) {
        //msgQueue = new COM("MSMQ.MSMQQueue"); //implementação real
        printf("MSMQ %s %s <br/>", $assunto, $texto);
    }
}

class ServicoSMS implements ServicoComunicacao {
    private const TELEFONE_ADMIN = '9999999999';
    public function enviarMensagem(string $assunto, string $texto) {
        //Implementação real
        //file_get_contents("http://api.clickatell.com/http/sendmsg?to=self::TELEFONE_ADMIN&msg=$texto");
        printf("SMS %s %s <br/>", $assunto, $texto);
    }
}
```



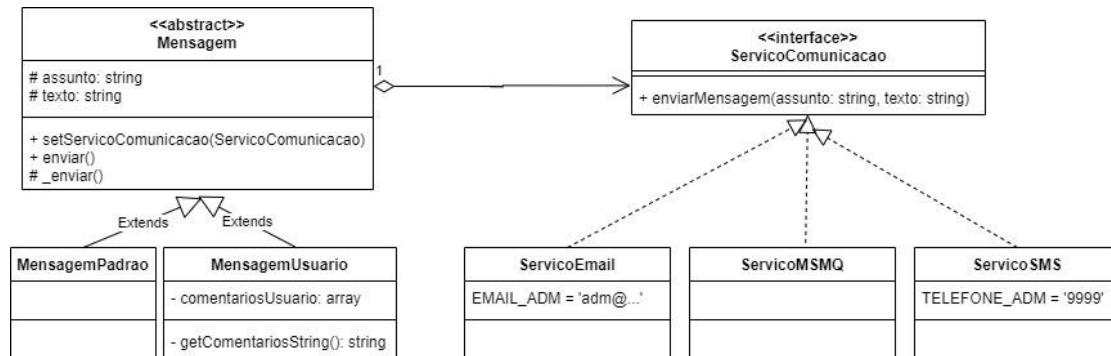
Bridge (exemplo - parte 4)

```
$email = new ServicoEmail();
$msmQ = new ServicoMSMQ();
$sms = new ServicoSMS();

$mensagem = new MensagemPadrao("Festa de
Confraternização", "Olá, todos estão convidados !");

$mensagem->setServicoComunicacao($email);
$mensagem->enviar();
$mensagem->setServicoComunicacao($msmQ);
$mensagem->enviar();
$mensagem->setServicoComunicacao($sms);
$mensagem->enviar();

$mensagemUsuario = new MensagemUsuario("Chamado
nº123", "Os seguintes assentamentos foram executados:",
["Análise","Correção","Entrega"]);
$mensagemUsuario->setServicoComunicacao($email);
$mensagemUsuario->enviar();
```



Composite

Composite é um pattern estrutural que permite compor objetos em estruturas de árvores e trabalhar com essas estruturas como se fossem objetos individuais (SHVETS, 2019).

“Componha objetos em estruturas de árvore para representar hierarquias de partes inteiras. Composite permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme” (GAMMA et al., 1994).



Composite (conceitual - parte 1)

```
<?php

interface Componente {
    public function fazerAlgo();
}

class Composite implements Componente {

    private \SplObjectStorage $componentes;

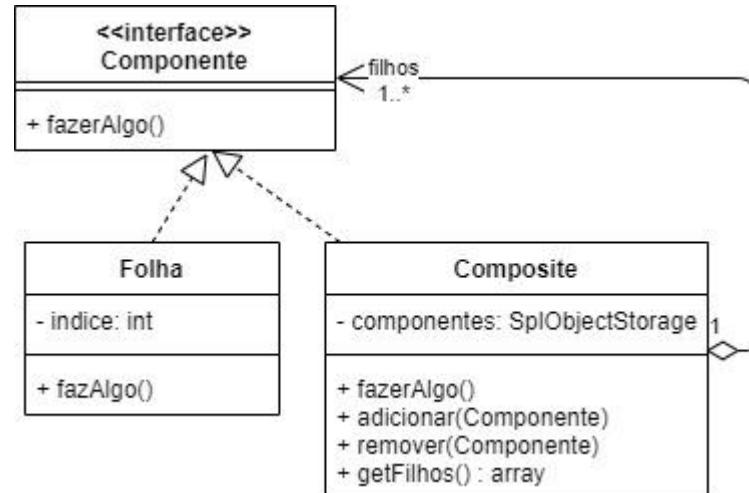
    public function __construct(){
        $this->componentes = new \SplObjectStorage();
    }

    public function fazerAlgo(){
        array_map(fn(Componente $f) => $f->fazerAlgo(), $this->getFilhos());
    }

    public function adicionar(Componente ...$componentes){
        array_map(fn($o) => $this->componentes->attach($o), $componentes);
    }

    public function remover(Componente $componente){
        $this->componentes->detach($object);
    }

    public function getFilhos() : array {
        return iterator_to_array($this->componentes);
    }
}
```



Composite (conceitual - parte 2)

```
<?php

class Folha implements Componente {

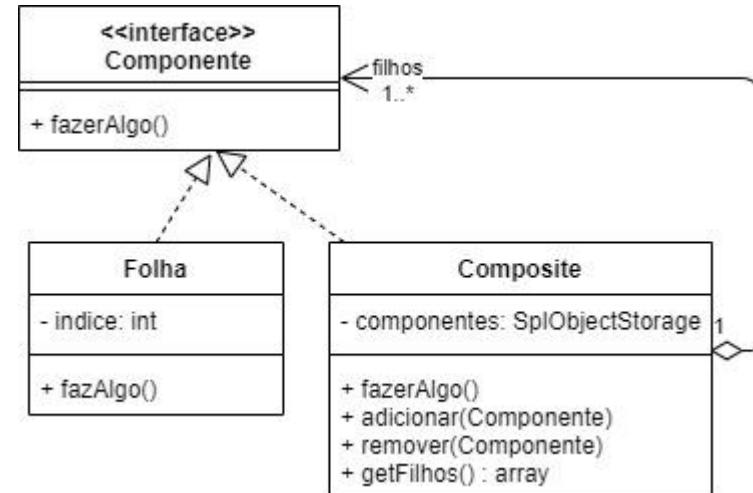
    private int $indice;

    public function __construct(int $indice) {
        $this->indice = $indice;
    }

    public function fazerAlgo() {
        echo "fazendo algo da folha {$this->indice} <br>".PHP_EOL;
    }

}

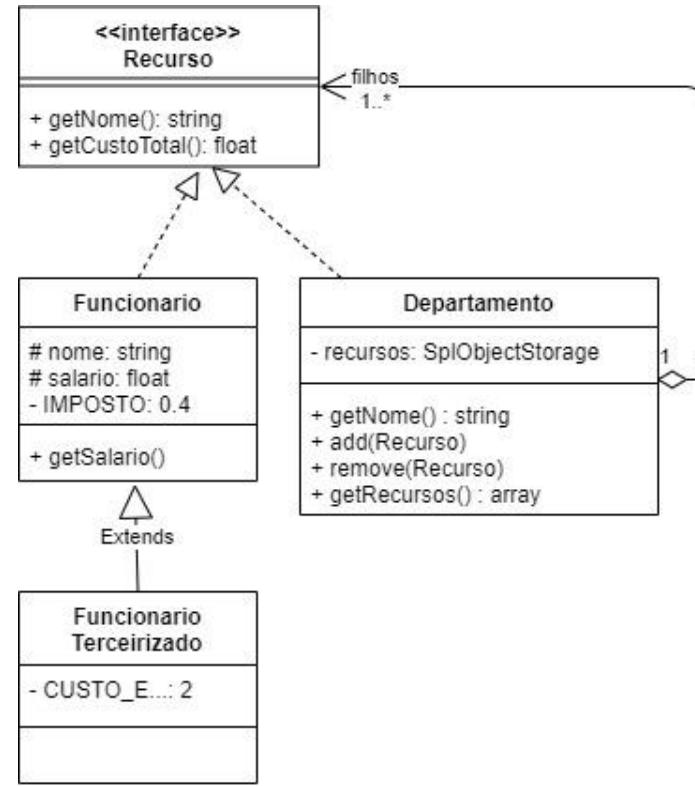
//trocar para componenteRaiz
$compositeRaiz = new Composite();
$compositeRaiz->adicionar(new Folha(1), new Folha(2), new Folha(3));
$compositeIntermediario = new Composite();
$compositeIntermediario->adicionar(new Folha(4));
$compositeRaiz->adicionar($compositeIntermediario);
$compositeRaiz->fazerAlgo();
```



Composite (exemplo - parte 1)

```
interface Recurso {
    public function getName(): string;
    public function getCustoTotal(): float;
}

class Departamento implements Recurso {
    private string $nome;
    private \SplObjectStorage $recursos;
    public function __construct(string $nome) {
        $this->recursos = new \SplObjectStorage();
        $this->nome = $nome;
    }
    public function getName(): string {
        return $this->nome;
    }
    public function adicionar(Recurso ...$recursos) {
        array_map(fn($o) => $this->recursos->attach($o), $recursos);
    }
    public function remover(Recurso $recurso): void {
        $this->recursos->detach($recurso);
    }
    public function getRecursos(): array {
        return iterator_to_array($this->recursos);
    }
    public function getCustoTotal(): float {
        return array_reduce($this->getRecursos(),
            fn($total, Recurso $r) => $total + $r->getCustoTotal(), 0);
    }
}
```



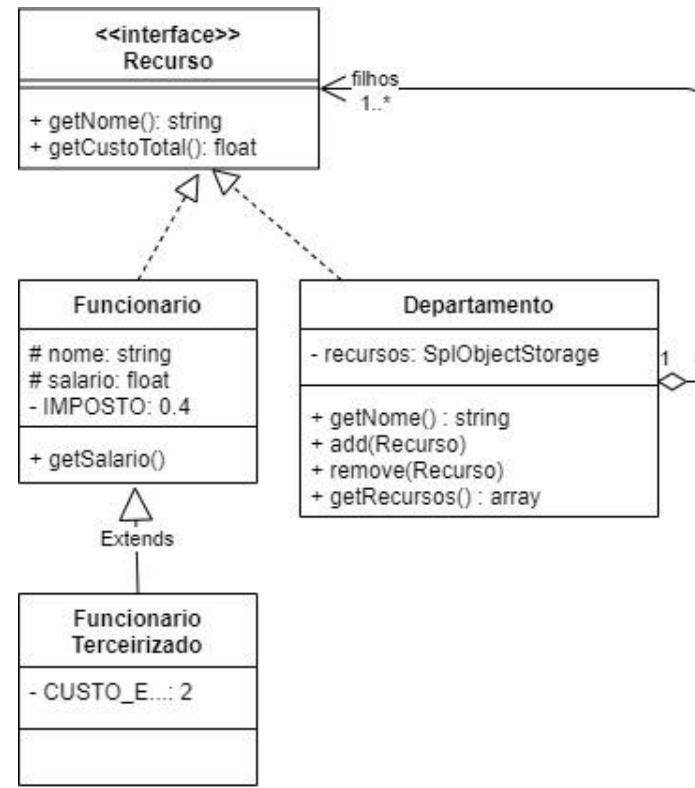
Composite (exemplo - parte 2)

```
class Funcionario implements Recurso {
    protected string $nome;
    protected float $salario;
    private const IMPOSTO = 0.4;

    function __construct(string $nome, float $salario) {
        $this->nome = $nome;
        $this->salario = $salario;
    }
    public function getNome(): string {
        return $this->nome;
    }
    public function getSalario(): float {
        return $this->salario;
    }
    public function getCustoTotal(): float {
        return $this->salario + ($this->salario * self::IMPOSTO);
    }
}

class FuncionarioTerceirizado extends Funcionario {

    const CUSTO_EMPRESA = 2;
    public function getCustoTotal(): float {
        return $this->salario * self::CUSTO_EMPRESA;
    }
}
```



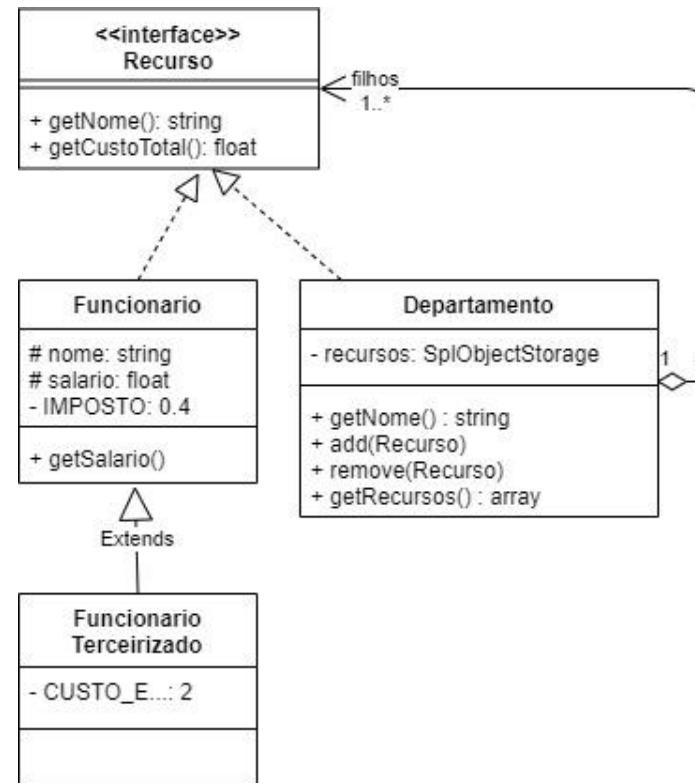
Composite (exemplo - parte 3)

```
$empresa = new Departamento('Empresa XPTO');
$gti = new Departamento('GTI');
$gti->adicionar(
    new Funcionario('José', 1200.00),
    new Funcionario('Thiago', 2000.00),
    new Funcionario('Lucas', 3000.00)
);
$suporte = new Departamento('Suporte TI');
$suporte->adicionar(new FuncionarioTerceirizado('Maria', 1200.10));
$gti->adicionar($suporte);
$empresa->adicionar($gti);

echo "A Empresa {$empresa->getNome()} tem o custo total de
{$empresa->getCustoTotal()}";
echo "<br>".PHP_EOL;
echo "O Departamento {$suporte->getNome()} tem o custo total de
{$suporte->getCustoTotal()}";

$gti->remover($suporte);

echo "<br>".PHP_EOL;
echo "A Empresa {$empresa->getNome()} agora tem o custo total de
{$empresa->getCustoTotal()}";
```



Decorator

- O padrão Decorator permite que seja adicionado novas funcionalidades a um objeto existente sem alterar sua estrutura.
- Esse padrão cria uma classe “decoradora” que envolve a classe original e fornece funcionalidades adicionais mantendo a assinatura de métodos da classe intacta.



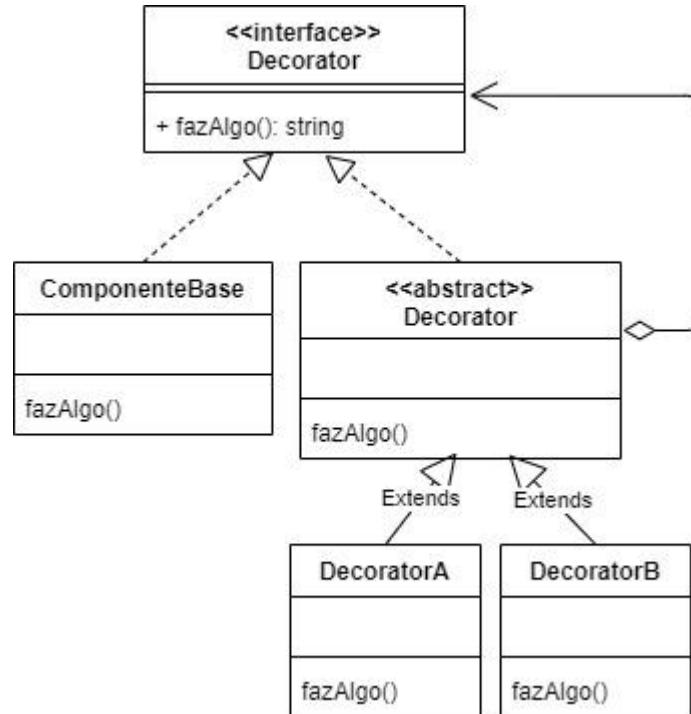
Decorator (conceitual - parte 1)

```
<?php
interface Componente {
    public function fazAlgo() : string;
}

class ComponenteBase implements Componente {
    public function fazAlgo() : string {
        return "Olá Mundo !";
    }
}

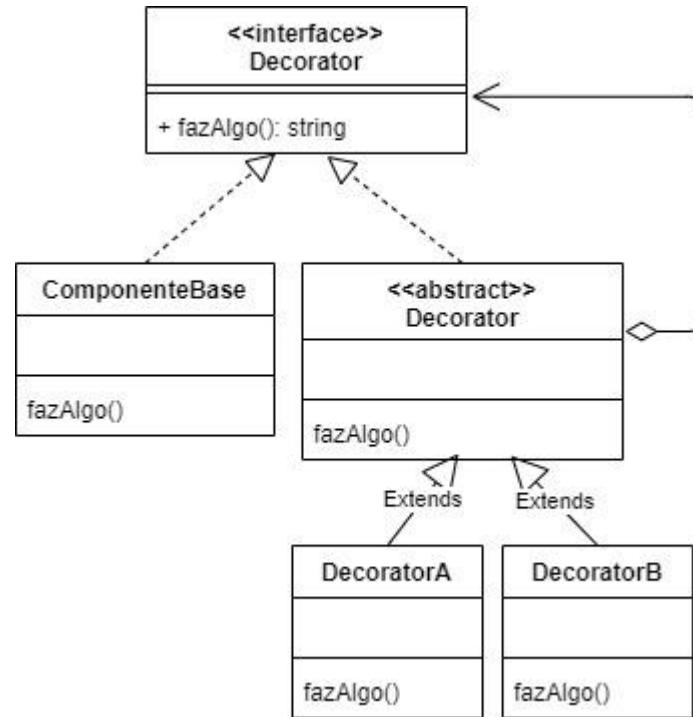
abstract class Decorator implements Componente {

    private Componente $decorado;
    public function __construct(Componente $decorado) {
        $this->decorado = $decorado;
    }
    public function fazAlgo() : string {
        return $this->decorado->fazAlgo();
    }
}
```



Decorator (conceitual - parte 2)

```
class DecoratorConcretoA extends Decorator {  
    public function fazAlgo() : string {  
        return parent::fazAlgo() . " !!! ";  
    }  
}  
  
class DecoratorConcretoB extends Decorator {  
    public function fazAlgo() : string {  
        return parent::fazAlgo() . " @@@ ";  
    }  
}  
  
$original = new ComponenteBase();  
echo $original->fazAlgo();  
echo "<br>".PHP_EOL;  
$decorado = new DecoratorConcretoA(  
    new DecoratorConcretoB(  
        new ComponenteBase()  
    )  
);  
echo $decorado->fazAlgo();
```



Decorator (exemplo - parte 1)

```
<?php
interface Sanduiche {
    public function calcularPreco(): float;
    public function getDescricao(): string;
}

class RecheioRosbife implements Sanduiche {
    public function calcularPreco(): float{
        return 7.00;
    }
    public function getDescricao(): string{
        return 'recheio de rosbife';
    }
}

class RecheioFrango implements Sanduiche {
    public function calcularPreco(): float{
        return 6.00;
    }
    public function getDescricao(): string{
        return 'recheio de frango';
    }
}
```

Decorator (exemplo - parte 2)

```
abstract class SanduicheDecorator implements Sanduiche {  
  
    protected Sanduiche $sanduiche;  
  
    public function __construct(Sanduiche $sanduiche){  
        $this->sanduiche = $sanduiche;  
    }  
}  
  
class QueijoExtra extends SanduicheDecorator {  
    private const PRECO = 4.00;  
    public function calcularPreco(): float {  
        return $this->sanduiche->calcularPreco() + self::PRECO;  
    }  
    public function getDescricao(): string {  
        return $this->sanduiche->getDescricao() . ' com queijo extra';  
    }  
}  
  
class Bacon extends SanduicheDecorator {  
    private const PRECO = 3.50;  
    public function calcularPreco(): float {  
        return $this->sanduiche->calcularPreco() + self::PRECO;  
    }  
    public function getDescricao(): string {  
        return $this->sanduiche->getDescricao() . ' e bacon';  
    }  
}
```

Decorator (exemplo - parte 3)

```
$sanduiche = new RecheioRobsife();
$sanduiche = new Bacon($sanduiche);
$sanduiche = new QueijoExtra($sanduiche);

echo $sanduiche->calcularPreco();
echo PHP_EOL;
echo $sanduiche->getDescricao();
```

Decorator (2º exemplo - parte 1)

```
<?php
interface DataSource {
    public function gerar($texto): string;
    public function recuperar($texto): string;
}

class TextoBase implements DataSource {
    public function gerar($texto): string {
        return $texto;
    }
    public function recuperar($texto): string {
        return $texto;
    }
}

abstract class Decorator implements DataSource {
    private DataSource $decorado;
    public function __construct(DataSource $decorado) {
        $this->decorado = $decorado;
    }
    public function gerar($texto): string {
        return $this->decorado->gerar($texto);
    }
    public function recuperar($texto): string {
        return $this->decorado->recuperar($texto);
    }
}
```

Decorator (2º exemplo - parte 2)

```
class CriptoDecorator extends Decorator {
    const KEY = 'vDla5JdknBqfrKOu8d7UpddnBMCH1vza'; //32
    const NONCE = 'Ra5LeH7ntW2rvkz3dmql5Stx'; //24
    public function gerar($texto): string {
        return $this->encrypt(parent::gerar($texto));
    }
    public function recuperar($texto): string {
        return parent::recuperar($this->decrypt($texto));
    }
    public function encrypt($data) {
        return sodium_crypto_secretbox($data, self::NONCE, self::KEY);
    }
    private function decrypt(string $data): string {
        return sodium_crypto_secretbox_open($data, self::NONCE, self::KEY);
    }
}
class Base64Decorator extends Decorator {
    public function gerar($texto): string {
        return $this->codificar(parent::gerar($texto));
    }
    public function recuperar($texto): string {
        return parent::recuperar($this->decodificar($texto));
    }
    private function codificar(string $stringData): string {
        return base64_encode($stringData);
    }
    private function decodificar(string $stringData): string {
        return base64_decode($stringData);
    }
}
```

Decorator (2º exemplo - parte 3)

```
class CompressaoDecorator extends Decorator {  
  
    public function gerar($texto): string {  
        return $this->comprimir(parent::gerar($texto));  
    }  
    public function recuperar($texto): string {  
        return parent::recuperar($this->descomprimir($texto));  
    }  
    private function comprimir(string $stringData): string {  
        return gzcompress($stringData);  
    }  
    private function descomprimir(string $stringData): string {  
        return gzuncompress($stringData);  
    }  
}  
  
$texto = "olá mundo !";  
$decorado = new CompressaoDecorator(  
    new Base64Decorator(  
        new CriptoDecorator(  
            new TextoBase()  
        )));  
$texto_decorado = $decorado->gerar($texto);  
echo PHP_EOL;  
echo $decorado->recuperar($texto_decorado);
```

Facade

Facade é um pattern estrutural que fornece uma interface simplificada para uma biblioteca, uma estrutura ou qualquer outro conjunto complexo de classes (SHEVTS, 2019).

De maneira análoga a uma fachada na arquitetura, uma fachada (Facade) é um objeto que serve como uma interface frontal, mascarando um código subjacente ou estrutural mais complexo (WIKIPEDIA 2020).



Facade (conceitual - parte 1)

```
namespace SubSistemaComplexo;

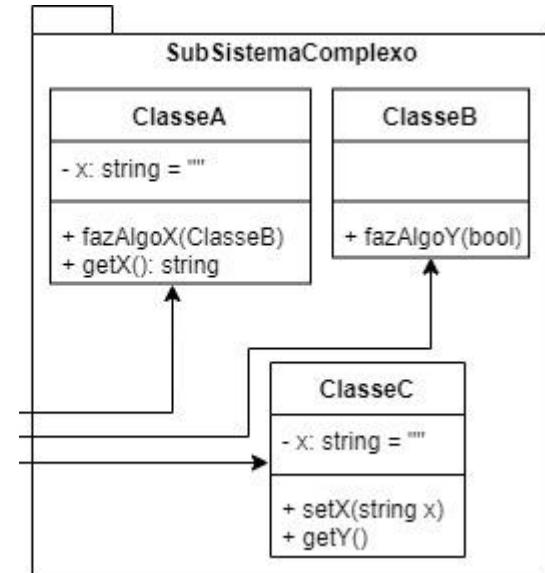
class ClasseA {

    private string $x = "";
    public function fazAlgoX(ClasseB $classeB) {
        $this->x = $classeB->fazAlgoY(true);
    }
    public function getX(): string {
        return $this->x;
    }
}

class ClasseB {

    public function fazAlgoY(bool $parametro) : string {
        return $parametro;
    }
}

class ClasseC {
    private string $x = "";
    public function setX(string $x) {
        $this->x = $x;
    }
    public function getY(): string {
        return $this->x . "!";
    }
}
```



Facade (conceitual - parte 2)

```
namespace FacadeConceitual;

use SubSistemaComplexo\ClasseA;
use SubSistemaComplexo\ClasseB;
use SubSistemaComplexo\ClasseC;

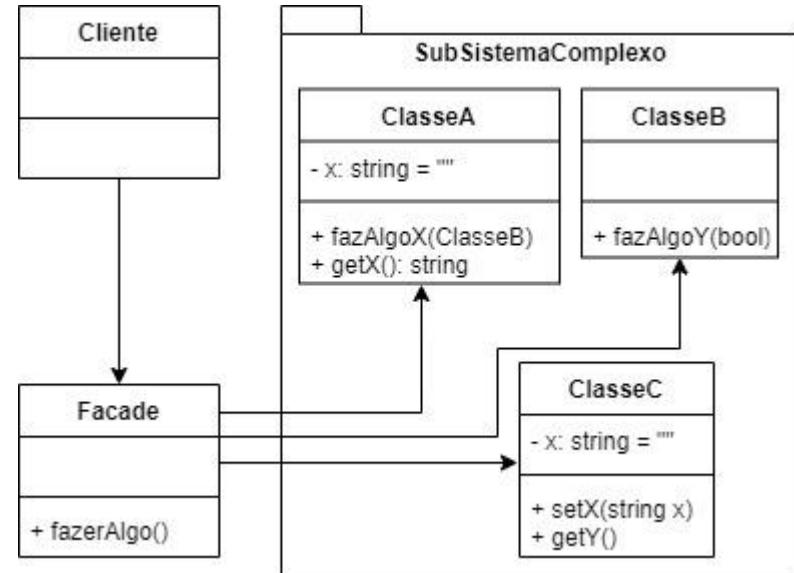
class Facade {

    private ClasseA $a;
    private ClasseB $b;
    private ClasseC $c;

    public function __construct() {
        $this->a = new ClasseA();
        $this->b = new ClasseB();
        $this->c = new ClasseC();
    }

    public function fazerAlgo(){
        $this->a->fazAlgoX($this->b);
        $this->c->setX($this->a->getX());
        echo $this->c->getY();
    }
}

$facade = new Facade();
$facade->fazerAlgo();
```

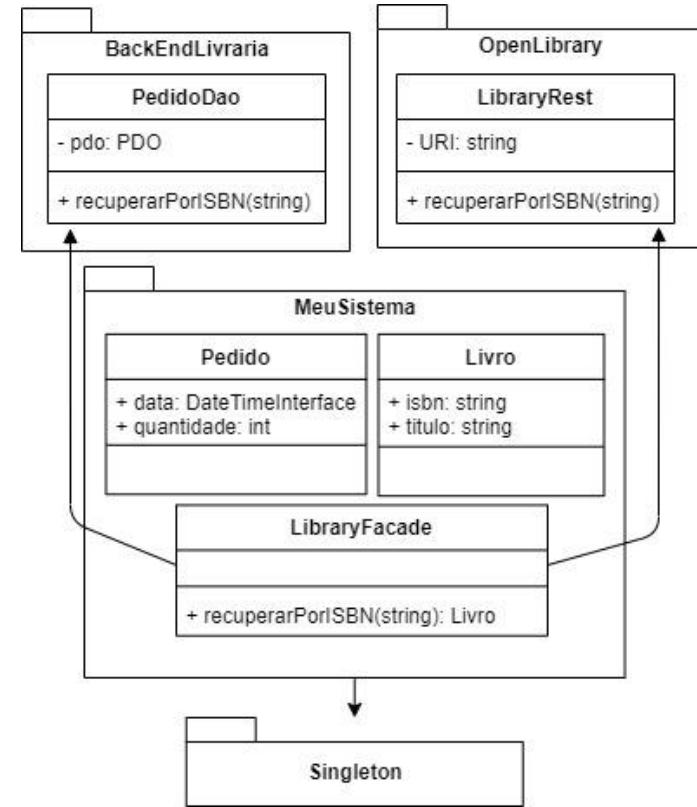


Facade (exemplo - parte 1)

```
<?php
namespace Singleton;
class Configuracao {//mesmo código usado no exemplo do singleton
}

class Conexao {//mesmo código usado no exemplo do singleton
}

namespace BackEndLivraria;
class PedidoDao {
    private \PDO $pdo;
    public function __construct(\PDO $pdo) {$this->pdo = $pdo;}
    public function recuperarPorIsbn(string $isbn) {
        $statement = $this->pdo->prepare(<<<SQL
            SELECT p.data, i.quantidade
            FROM pedido p
            INNER JOIN item_pedido i ON p.id = i.item_pedido_id
            INNER JOIN livro l ON l.id = i.livro_id
            WHERE isbn = :isbn AND p.data IS NOT NULL
            SQL);
        $statement->execute(['isbn' => $isbn]);
        return $statement->fetchAll();
    }
}
```

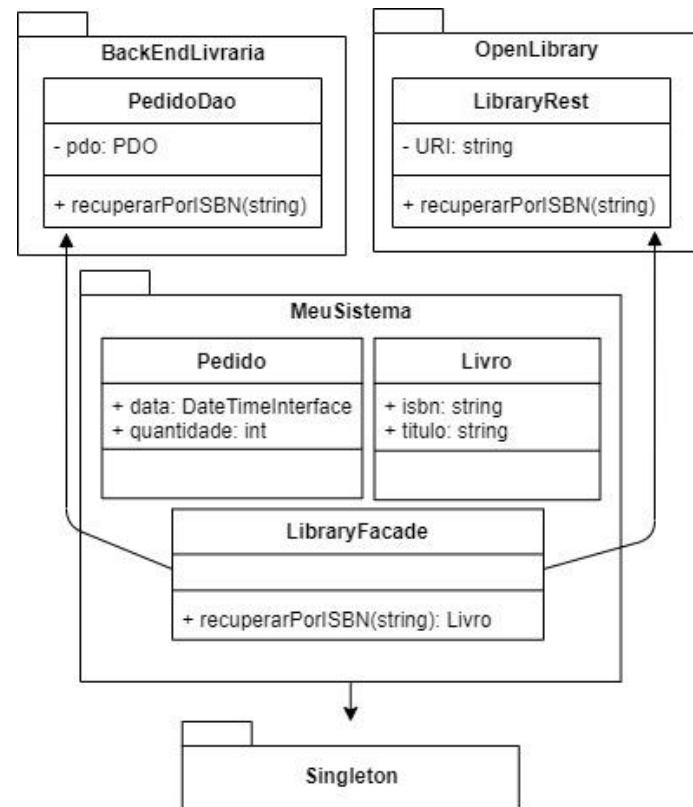


Facade (exemplo - parte 2)

```
namespace OpenLibrary;
class OpenLibraryREST {

    private const URI = 'https://openlibrary.org/api/books';

    public function recuperarPorIsbn(string $isbn){
        $uri = self::URI."?bibkeys=ISBN:$isbn&jscmd=data&format=json";
        return json_decode(file_get_contents($uri));
    }
}
```

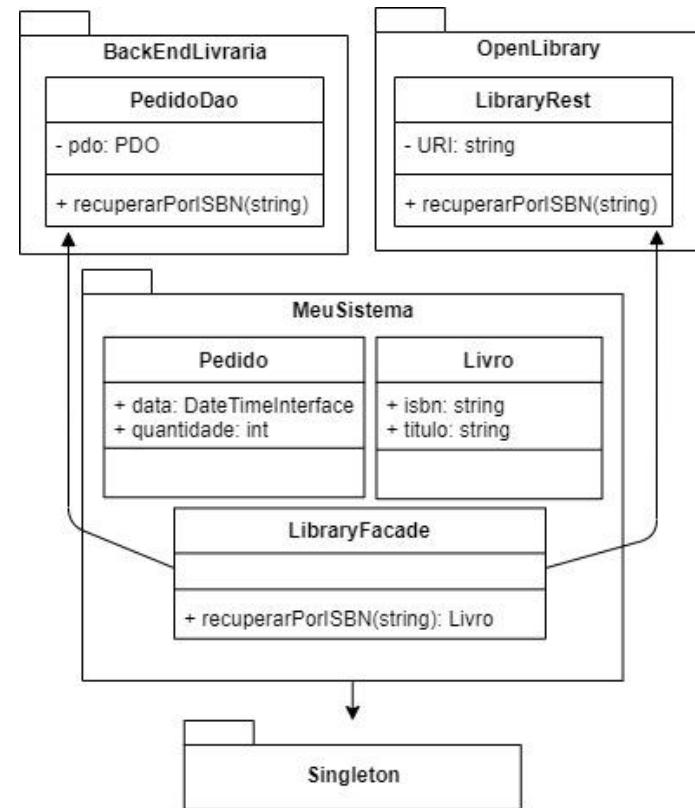


Facade (exemplo - parte 3)

```
namespace MeuSistema;
use OpenLibrary\OpenLibraryREST;
use BackEndLivraria\PedidoDao;
use Singleton\Conexao;

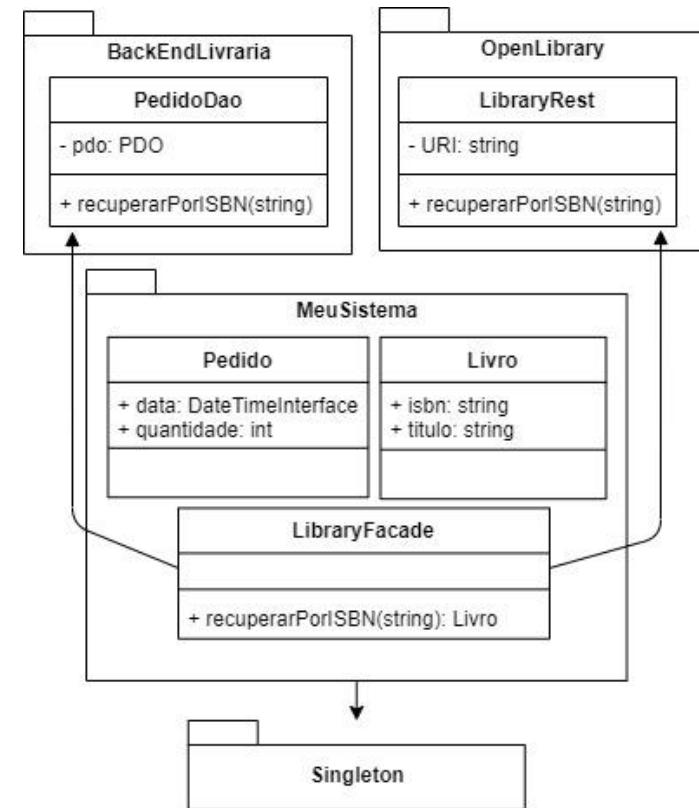
class Pedido {
    public \DateTimeInterface $data;
    public int $quantidade;
    public function __construct(\DateTimeInterface $data, int
$quantidade) {
        $this->data = $data;
        $this->quantidade = $quantidade;
    }
}

class Livro {
    public string $isbn;
    public string $titulo;
    public function __construct(string $isbn, string $titulo) {
        $this->isbn = $isbn;
        $this->nome = $titulo;
    }
}
```



Facade (exemplo - parte 4)

```
class LibraryFacade {  
  
    private OpenLibraryREST $libraryRest;  
    private PedidoDao $pedidoDao;  
  
    public function __construct() {  
        $this->libraryRest = new OpenLibraryREST();  
        $this->pedidoDao = new PedidoDao(Conexao::getInstance()->getPdo());  
    }  
  
    public function recuperarLivroPorIsbn($isbn): ?Livro {  
        $livro = null;  
  
        $resultadoRest = $this->libraryRest->recuperarPorIsbn($isbn);  
        if ($resultadoRest) {  
            $livro = new Livro($isbn, $resultadoRest->{"ISBN:$isbn"}->title);  
            $livro->pedidos = array_map(fn($v) => new Pedido(  
                \DateTime::createFromFormat('Y-m-d H:i:s', $v['data']),  
                $v['quantidade']),  
                $this->pedidoDao->recuperarPorIsbn($isbn));  
        }  
        return $livro;  
    }  
  
    $facade = new LibraryFacade();  
    var_dump($facade->recuperarLivroPorIsbn('9780201485370'));  
}
```



Flyweight

Flyweight é um pattern estrutural que permite ajustar mais objetos à quantidade disponível de RAM, compartilhando partes comuns do estado entre vários objetos, em vez de manter todos os dados em cada objeto (SHVETS, 2019).

Um flyweight é um objeto compartilhado que pode ser usado em vários contextos simultaneamente. Os flyweights não podem fazer suposições sobre o contexto em que operam. O conceito-chave aqui é a distinção entre estado intrínseco e extrínseco (GAMMA et at, 1994).



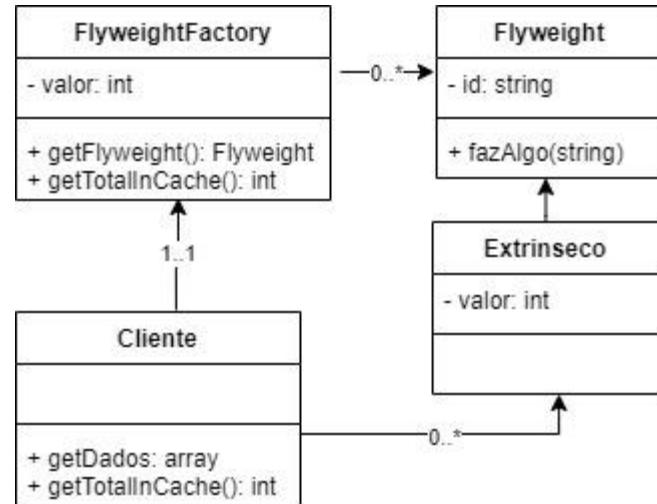
Intrínseco (invariante) x Extrínseco (variante)

Dado imutável de um objeto geralmente é chamado de estado **intrínseco**. Ele vive dentro do objeto; outros objetos podem apenas lê-lo, não alterá-lo. O restante do estado do objeto, frequentemente modificado "por fora" por outros objetos, é chamado de estado **extrínseco**. O padrão Flyweight sugere que você pare de armazenar o estado extrínseco dentro do objeto. Em vez disso, você deve passar este estado para métodos específicos que dependem dele. Somente o estado intrínseco permanece dentro do objeto, permitindo a sua reutilização em diferentes contextos (SHVETS, 2019).

O estado **intrínseco** é armazenado no *flyweight*; consiste em informações independentes do contexto do flyweight, tornando-o compartilhável. O estado **extrínseco** depende e varia de acordo com o contexto do *flyweight* e, portanto, não pode ser compartilhado. Os *objetos do cliente* são responsáveis por passar o estado extrínseco para o flyweight quando necessário (GAMMA et al, 1994).

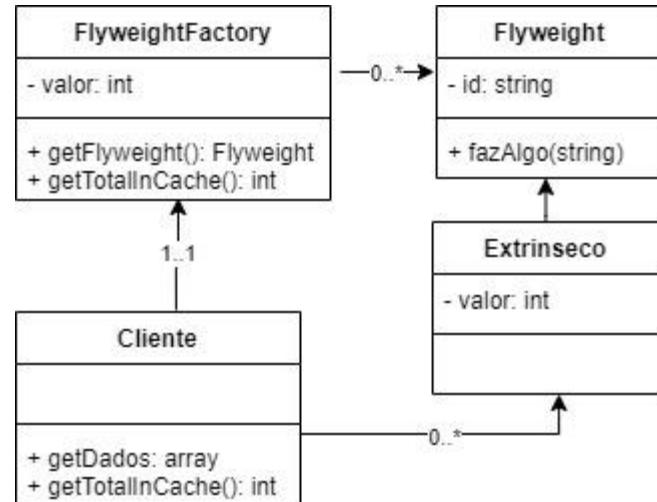
Flyweight (conceitual - parte 1)

```
class Flyweight {  
  
    private string $id = "";  
    public function __construct(string $id) {  
        var_dump("construção de $id");  
        $this->id = $id;  
    }  
    public function fazAlgo(string $extrinseco) {  
        echo "fazendo {$extrinseco} algo com o {$this->id} intrínseco";  
    }  
}  
  
class Exrinseco {  
  
    private int $valor;  
    private Flyweight $intrinseco;  
    public function __construct(Flyweight $intrinseco, int $valor) {  
        $this->intrinseco = $intrinseco;  
        $this->valor = $valor;  
    }  
}
```



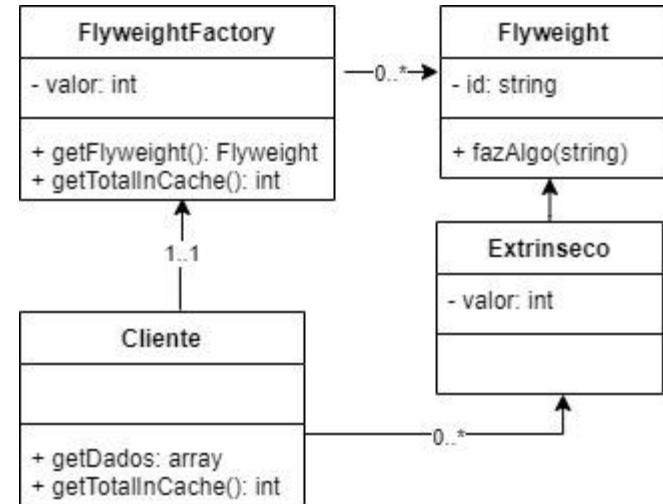
Flyweight (conceitual - parte 2)

```
class FlyweightFactory {  
  
    private array $flyweights = [];  
  
    public function getFlyweight(string $id): Flyweight {  
  
        if(!isset($this->flyweights[$id])){  
            $this->flyweights[$id] = new Flyweight($id);  
        }  
        return $this->flyweights[$id];  
  
    }  
    public function getTotalInCache(): int {  
        return count($this->flyweights);  
    }  
}
```



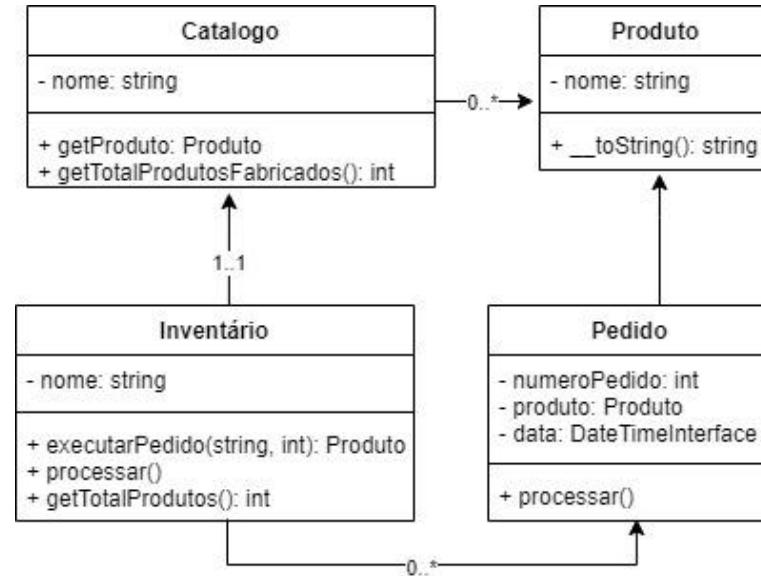
Flyweight (conceitual - parte 3)

```
class Cliente {  
    private FlyweightFactory $flyweightFactory;  
    private array $dados;  
    public function __construct() {  
        $this->flyweightFactory = new FlyweightFactory();  
    }  
    public function adicionar(string $id, int $valor){  
        $this->dados[] = new Extrinseco(  
            $this->flyweightFactory->getFlyweight($id), $valor);  
    }  
    public function getDados(): array {  
        return $this->dados;  
    }  
    public function getTotalInCache(): int {  
        return $this->flyweightFactory->getTotalInCache();  
    }  
}  
$cliente = new Cliente();  
$cliente->adicionar("A", 1);  
$cliente->adicionar("B", 2);  
$cliente->adicionar("C", 3);  
$cliente->adicionar("A", 4);  
$cliente->adicionar("C", 4);  
echo count($cliente->getDados());  
echo "<br/>";  
echo $cliente->getTotalInCache();
```



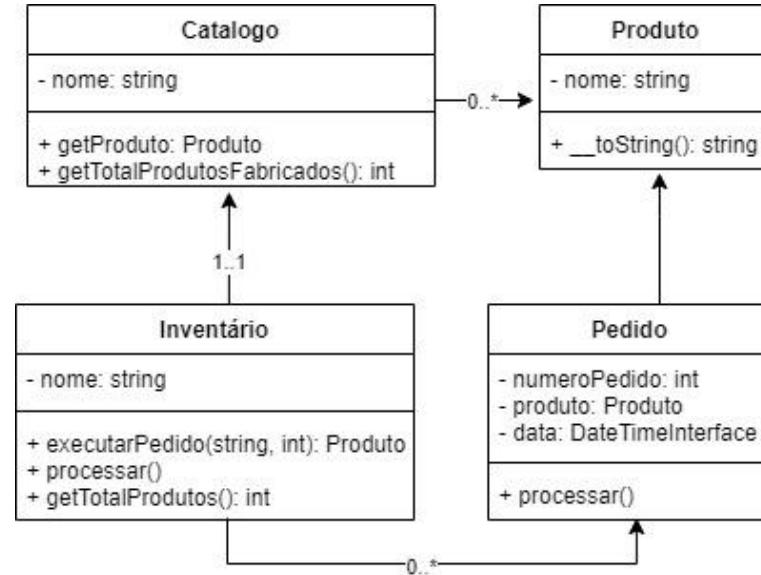
Flyweight (exemplo - parte 1)

```
class Produto { //flyweight em si (intrínseco)
    private string $nome;
    public function __construct(string $nome) {
        $this->nome = $nome;
    }
    public function __toString(): string {
        return $this->nome;
    }
}
class Pedido { //contexto (extrínseco)
    private int $numeroPedido;
    private Produto $produto;
    private \DateTimeInterface $data;
    public function __construct(int $numeroPedido, Produto $produto) {
        $this->numeroPedido = $numeroPedido;
        $this->produto = $produto;
        $this->data = new \DateTimeImmutable();
    }
    public function processar(){
        echo "Encomendando " . $this->produto . " do pedido nº " . $this->numeroPedido .
        " no dia " . $this->data->format('d/m/Y h:i:s') . "<br>" . PHP_EOL;
    }
}
```



Flyweight (exemplo - parte 2)

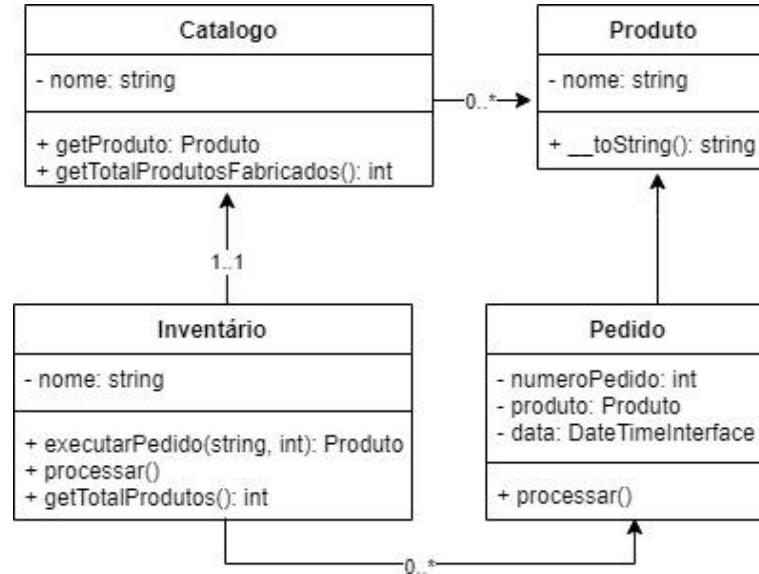
```
class Catalogo { //Fábrica Flyweight
    private array $produtos = [];
    public function getProduto(string $nomeProduto): Produto {
        if (!isset($this->produtos[$nomeProduto])) {
            $this->produtos[$nomeProduto] = new Produto($nomeProduto);
        }
        return $this->produtos[$nomeProduto];
    }
    public function getTotalProdutosFabricados(): int {
        return count($this->produtos);
    }
}
class Inventario { //Client Flyweight
    private Catalogo $catalogo;
    public array $pedidos = [];
    public function __construct() {
        $this->catalogo = new Catalogo();
    }
    function executarPedido(string $nomeProduto, int $numeroPedido) {
        $produto = $this->catalogo->getProduto($nomeProduto);
        $pedido = new Pedido($numeroPedido, $produto);
        $this->pedidos[] = $pedido;
    }
    function processar(){
        foreach ($this->pedidos as $key => $pedido) {
            $pedido->processar();
            unset($this->pedidos[$key]);
        }
    }
    function getTotalProdutos(): int {
        return $this->catalogo->getTotalProdutosFabricados();
    }
}
```



Flyweight (exemplo - parte 3)

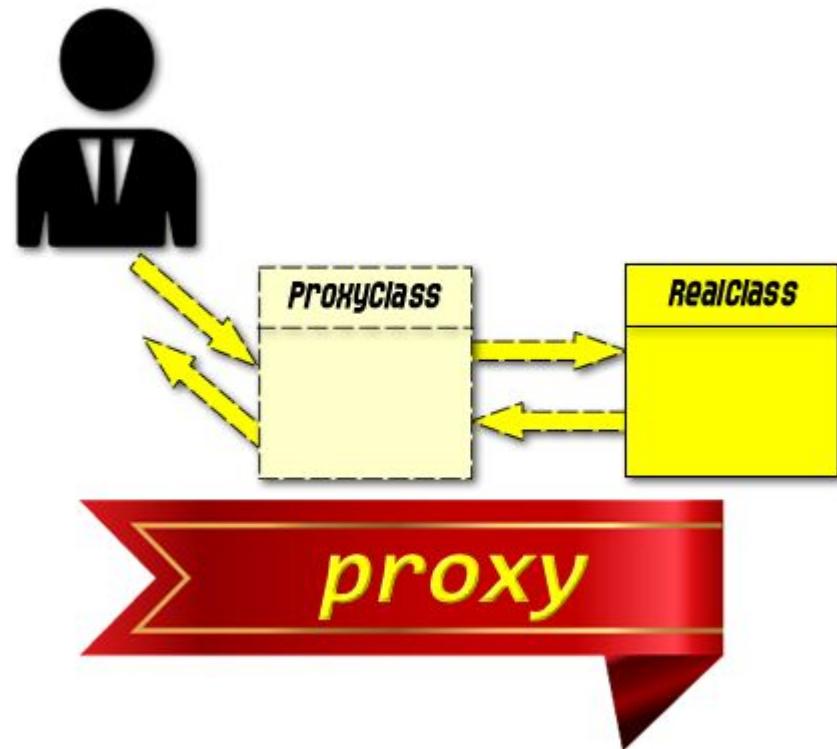
```
function formatBytes($size) {
    $base = log($size) / log(1024);
    $suffix = ["B", "KB", "MB", "GB", "TB"];
    $f_base = floor($base);
    return round(pow(1024, $base - floor($base)), 1) . $suffix[$f_base];
}

function testar(Inventory $inventory, $flyweight = true, $total = 10000) {
    $produtosNome = ["Notebook Gamer XPTO", "Fone Bluetooth YY", "Sega Dreamcast"];
    foreach (range(0, $total, count($produtosNome)) as $x) {
        foreach ($produtosNome as $key => $nome) {
            if($flyweight){
                $inventory->executarPedido($nome, $x + $key);
            }else{
                $inventory->pedidos[] = new Pedido($x + $key, new Produto($nome));
            }
        }
    }
    $inventory = new Inventory();
    testar($inventory);
    echo "<br>" . PHP_EOL;
    echo "total : " . formatBytes(memory_get_usage());
    echo "<br>" . PHP_EOL;
    $inventory->processar();
    echo $inventory->getTotalProdutos();
```



Proxy

O Proxy é pattern estrutural que visa permitir o fornecimento de um *placeholder* (substituto) para outro objeto. Um proxy controla o acesso ao objeto original, permitindo que algo seja feito antes ou depois do pedido chegar ao objeto original (SHVETS, 2019).



Tipos Principais de Proxy

Existem quatro variações principais no padrão de proxy:

- **Virtual:** cria objetos custosos sob demanda (*lazy load*);
- **Proteção:** controla o acesso ao objeto original. Os proxies de proteção são úteis quando os objetos devem ter direitos de acesso diferenciados;
- **Remoto:** fornece um representante local para um objeto localizado em outro endereço;
- **Referência Inteligente:** é uma substituição de um ponteiro simples que executa ações quando um objeto é acessado. Os usos típicos incluem:
 - contar o número de referências ao objeto real para que ele possa ser liberado automaticamente quando não há mais referências.
 - carregar um objeto persistente na memória quando ele é referenciado pela primeira vez.
 - verificar se o objeto real está bloqueado antes de ser acessado para garantir que nenhum outro objeto pode alterá-lo.

(GAMMA et al, 1994) (SHVETS, 2019).

Proxy (conceitual - parte 1)

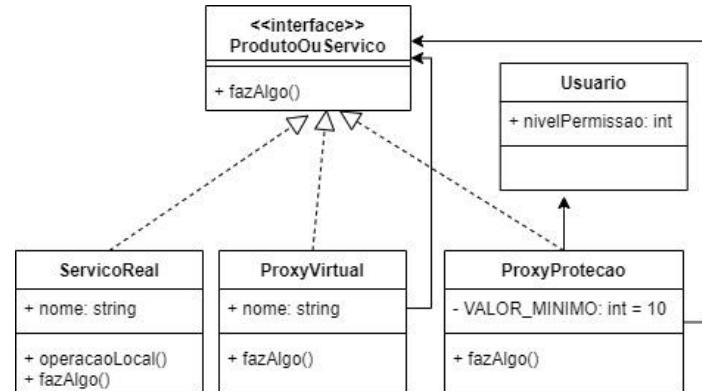
```
<?php

interface ProdutoOuServico {
    public function fazAlgo();
}

class Usuario {
    public int $nivelPermissao;
    public function __construct(int $nivelPermissao){
        $this->nivelPermissao = $nivelPermissao;
    }
}

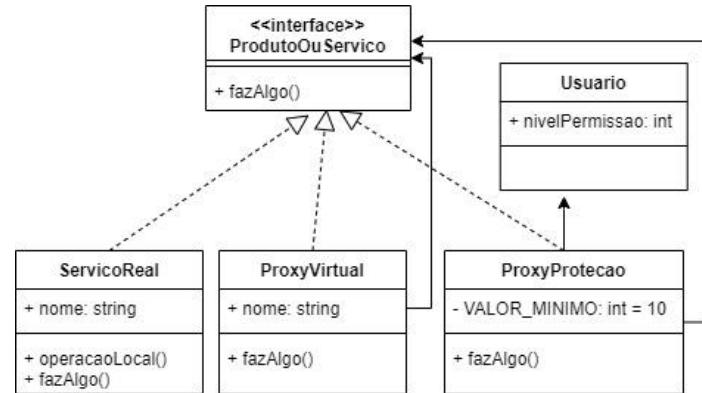
class ServicoReal implements ProdutoOuServico {

    private string $nome;
    public function __construct(string $nome) {
        $this->nome = $nome;
        $this->operacaoLocal();
    }
    private function operacaoLocal() {
        echo "carregando {$this->nome} <br/>" . \PHP_EOL;
    }
    public function fazAlgo() {
        echo "executando {$this->nome} <br/>" . \PHP_EOL;
    }
}
```



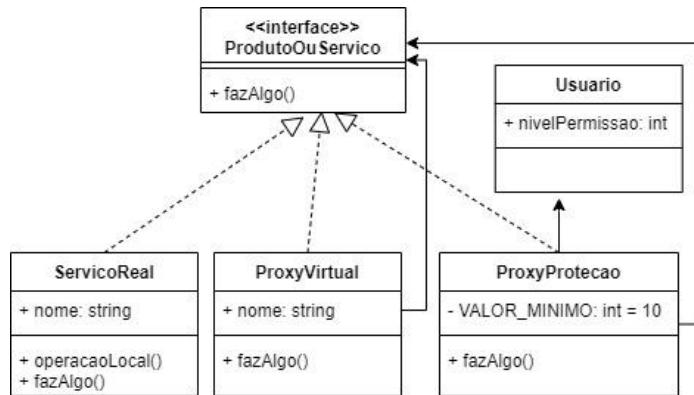
Proxy (conceitual - parte 2)

```
class ProxyVirtual implements ProdutoOuServico {
    private ?ProdutoOuServico $servico = null;
    private string $nome;
    public function __construct(string $nome) {
        $this->nome = $nome;
    }
    public function fazAlgo() { //lazy
        if ($this->servico === null) {
            $this->servico = new ServicoReal($this->nome);
        }
        $this->servico->fazAlgo();
    }
}
class ProxyProtecao implements ProdutoOuServico {
    private ProdutoOuServico $produto;
    private Usuario $produtoAssociado;
    private const VALOR_MINIMO = 10;
    public function __construct(string $nome, Usuario $produtoAssociado) {
        $this->produtoAssociado = $produtoAssociado;
        $this->produto = new ServicoReal($nome);
    }
    public function fazAlgo() {
        if($this->produtoAssociado->nivelPermissao >= self::VALOR_MINIMO){
            $this->produto->fazAlgo();
        }else{
            echo "Sem permissão";
        }
    }
}
```



Proxy (conceitual - parte 3)

```
//primeira implementação
$servico1 = new ProxyVirtual("Servico ABC 123");
$servico2 = new ProxyVirtual("Servico XPTO 9999");
$servico1->fazAlgo(); // Carregamento necessário
$servico1->fazAlgo(); // Carregamento desnecessário
$servico2->fazAlgo(); // Carregamento necessário
$servico2->fazAlgo(); // Carregamento desnecessário
$servico1->fazAlgo(); // Carregamento desnecessário
//segunda implementação
$servico3 = new ProxyProtecao("Servico Fulano 171", new Usuario(11));
$servico3->fazAlgo();
$servico3->fazAlgo();
```

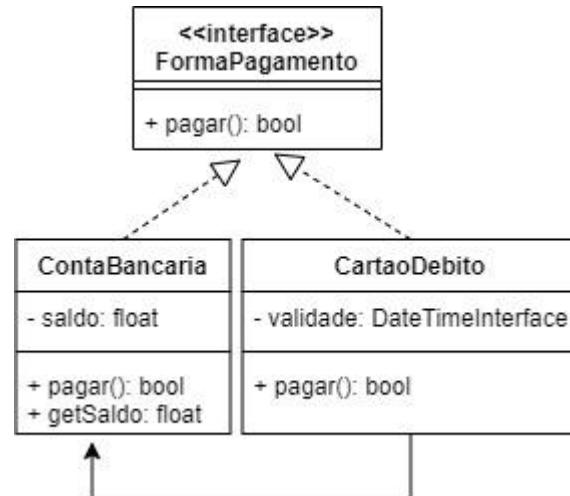


Proxy (exemplo - parte 1)

```
<?php

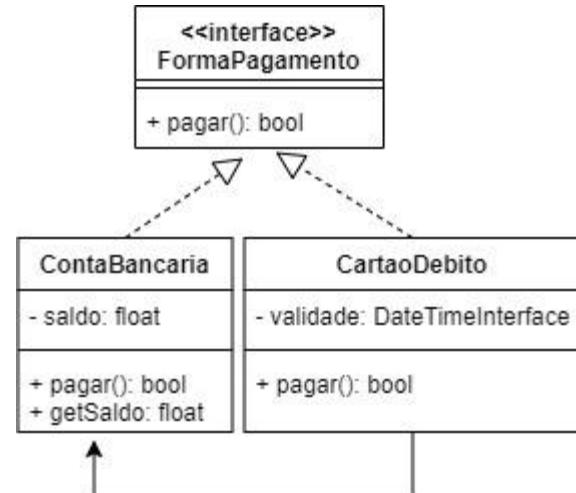
interface FormaPagamento {
    public function pagar(float $valor): bool;
}

class ContaBancaria implements FormaPagamento {
    private float $saldo;
    public function __construct(float $saldo = 0){
        $this->saldo = $saldo;
    }
    public function pagar(float $valor): bool {
        if ($this->saldo >= $valor) {
            $this->saldo -= $valor;
            return true;
        }
        return false;
    }
    public function getSaldo(): float {
        return $this->saldo;
    }
}
```



Proxy (exemplo - parte 2)

```
class CartaoDebito implements FormaPagamento {  
  
    private ContaBancaria $contaBancaria;  
    private \DateTimeInterface $validade;  
  
    public function __construct(ContaBancaria $contaBancaria, \DateTimeInterface  
$validade) {  
        $this->contaBancaria = $contaBancaria;  
        $this->validade = $validade;  
    }  
  
    public function pagar(float $valor): bool {  
        if ($this->validade >= (new \DateTime())) {  
            return $this->contaBancaria->pagar($valor);  
        } else {  
            throw new \Exception('Cartão vencido !!!');  
        }  
    }  
}  
  
$contaBancaria = new ContaBancaria(1000);  
$cartaoDebito = new CartaoDebito($contaBancaria, (new \DateTime('+1 week')));  
var_dump($cartaoDebito->pagar(100));
```



Padrões Comportamentais

Os Padrões comportamentais estão relacionados aos **algoritmos** e à atribuição de **responsabilidades entre os objetos** (GAMMA et at, 1994).

Os Padrões comportamentais descrevem **interações entre objetos** e enfocam como os objetos se **comunicam** uns com os outros. Eles podem reduzir fluxogramas complexos a meras interconexões entre objetos de várias classes. Os Padrões comportamentais também são usados para fazer com que o algoritmo utilizado em uma classe seja como um simples parâmetro ajustável em tempo de execução (GOFPATTERNS, 2021).

Padrões comportamentais são voltados aos **algoritmos** e a designação de **responsabilidades entre objetos** (SHVETS, 2019).

Chain of Responsibility

Chain of responsibility é um pattern comportamental que permite passar solicitações ao longo de uma **cadeia** de manipuladores. Ao receber uma solicitação, cada manipulador decide processar a solicitação ou passá-la para o próximo manipulador na **cadeia** (SHVETS, 2019)

Evita acoplar o remetente de uma solicitação ao destinatário, dando a mais de um objeto a chance de lidar com a solicitação. **Encadeie** os objetos de requisição e passe a solicitação ao longo da **cadeia** até que um objeto lide com isso (GAMMA et al, 1994).

Chain of responsibility é uma versão orientada a objeto do idioma if ... else if ... else if else ... endif, com o benefício de que os blocos condição-ação podem ser dinamicamente reorganizados e reconfigurados em tempo de execução (WIKIPEDIA, 2020).



Chain of Responsibility (problema)

```
$solicitacao = "A";
class Teste {
    static function fazAlgo($solicitacao){
        if($solicitacao == "A"){
            return "processando $solicitacao com A !";
        }elseif($solicitacao == "B"){
            return "processando $solicitacao com B !";
        }elseif($solicitacao == "C"){
            return "processando $solicitacao com C !";
        }
    }
    static function fazAlgo2($solicitacao){
        switch($solicitacao){
            case "A":
                return "processando $solicitacao com A !";
                break;
            case "B":
                return "processando $solicitacao com B !";
                break;
            case "C":
                return "processando $solicitacao com C !";
                break;
        }
    }
    static function fazAlgo3($solicitacao){
        return match($solicitacao){
            "A" => "processando $solicitacao com A !",
            "B" => "processando $solicitacao com B !",
            "C" => "processando $solicitacao com C !",
        };
    }
}
echo Teste::fazAlgo($solicitacao);
echo Teste::fazAlgo2($solicitacao);
echo Teste::fazAlgo3($solicitacao);
```

Chain of Responsibility (conceitual)

```
interface Manipulador { //Handler
    public function setProximoManipulador(Manipulador $proximoManipulador): Manipulador;
    public function podeManipular($solicitacao): bool;
}

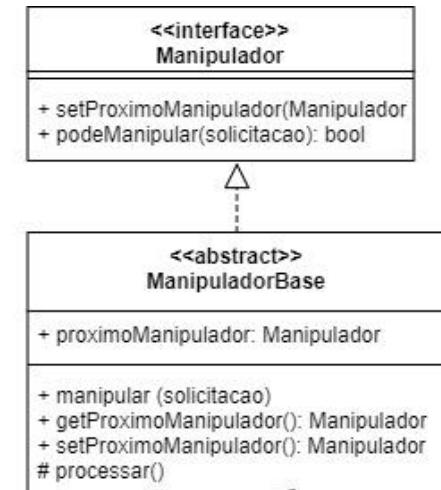
abstract class ManipuladorBase implements Manipulador {

    private ?Manipulador $proximoManipulador = null;
    public final function manipular($solicitacao) {
        if ($this->podeManipular($solicitacao)) {
            $this->processar($solicitacao);
        } else {
            if ($this->proximoManipulador != null) {
                $this->proximoManipulador->manipular($solicitacao);
            }
        }
    }

    public final function getProximoManipulador(): ?Manipulador {
        return $this->proximoManipulador;
    }

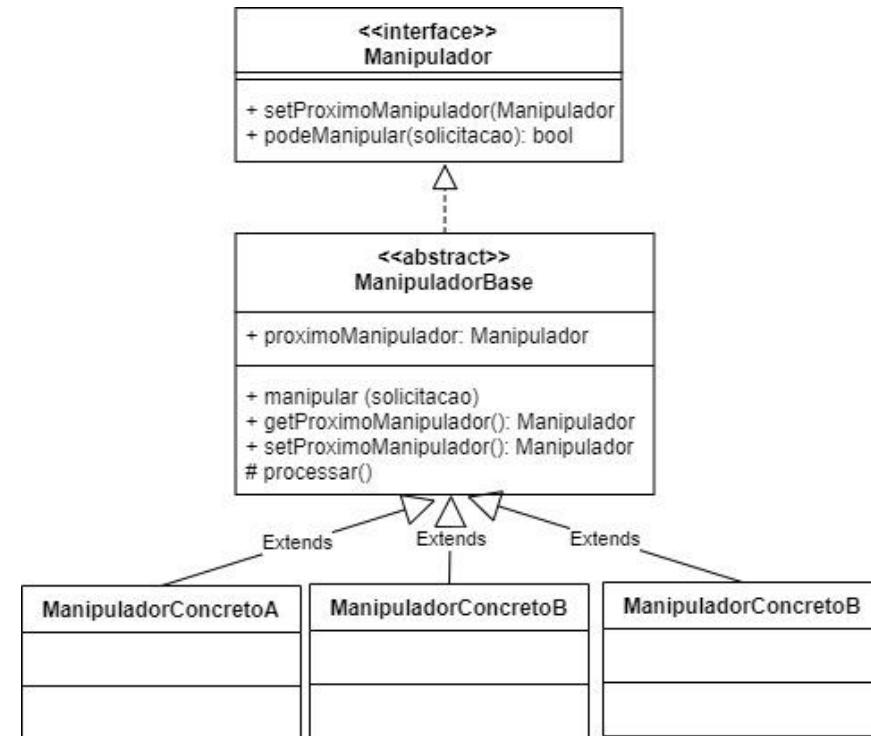
    public final function setProximoManipulador(Manipulador $proximoManipulador): Manipulador {
        $this->proximoManipulador = $proximoManipulador;
        return $proximoManipulador; //fluent interface
    }

    protected abstract function processar($solicitacao);
}
```



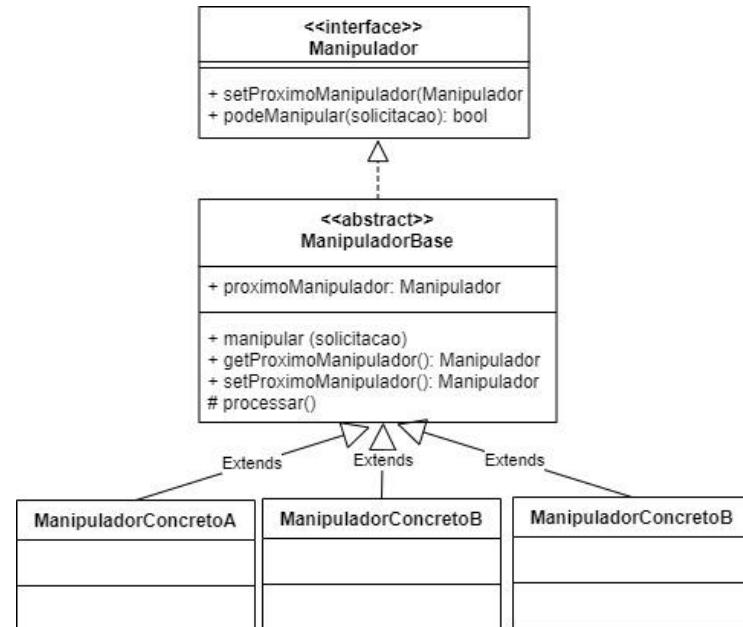
Chain of Responsibility (conceitual - parte 2)

```
class ManipuladorConcretoA extends ManipuladorBase {  
  
    public function podeManipular($solicitacao): bool {  
        if ($solicitacao == "A") {  
            return true;  
        }  
        return false;  
    }  
  
    protected function processar($solicitacao) {  
        echo "processando $solicitacao com " . self::class . "!";  
    }  
  
}  
  
class ManipuladorConcretoB extends ManipuladorBase {  
  
    public function podeManipular($solicitacao): bool {  
        if ($solicitacao == "B") {  
            return true;  
        }  
        return false;  
    }  
  
    protected function processar($solicitacao) {  
        echo "processando $solicitacao com " . self::class . "!";  
    }  
  
}
```



Chain of Responsibility (conceptual - parte 3)

```
class ManipuladorConcretoC extends ManipuladorBase {  
  
    public function podeManipular($solicitacao): bool {  
        if ($solicitacao == "C" && (new \DateTime())->format('d') < 30) {  
            return true;  
        }  
        return false;  
    }  
  
    protected function processar($solicitacao) {  
        echo "processando $solicitacao com ". self::class . "!!!";  
    }  
  
}  
  
$chain = new ManipuladorConcretoA();  
  
//montando cadeia manualmente  
$chain->setProximoManipulador(new ManipuladorConcretoB())  
    ->setProximoManipulador(new ManipuladorConcretoC());
```



Chain of Responsibility (conceitual - parte 4)

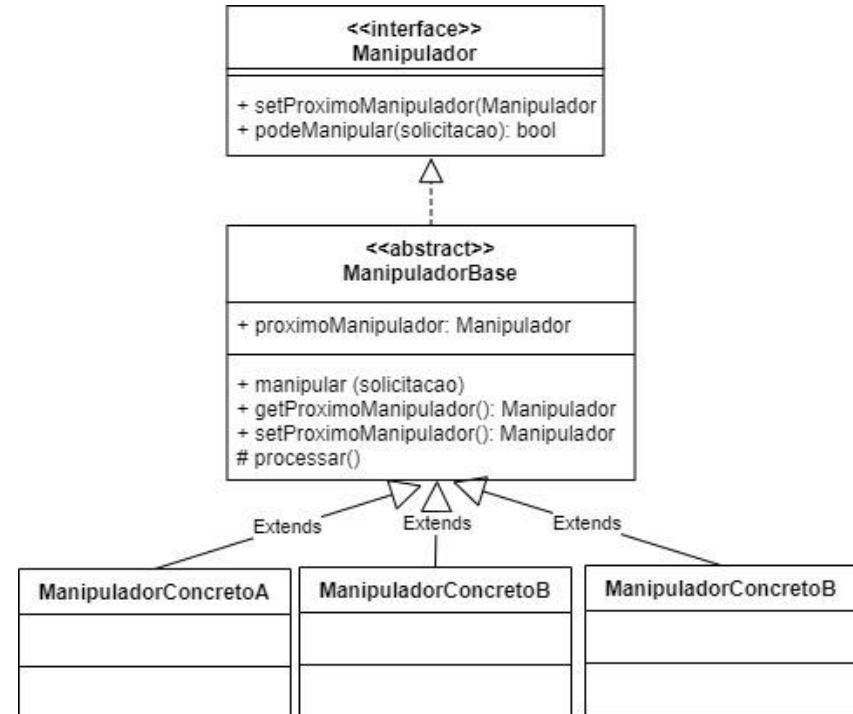
```
//montando cadeia com reflection
function getChainAuto($superClass = ManipuladorBase::class, $metodo =
'setProximoManipulador'): object {

    $manipuladores = array_filter(
        array_map(
            fn($c) => is_subclass_of($c, $superClass) ? new $c : null,
            get_declared_classes()
        ));
    }

    foreach ($manipuladores as $k => $v) {
        if ($k !== array_key_last($manipuladores)) {
            $v->{$metodo}($manipuladores[$k + 1]);
        }
    }

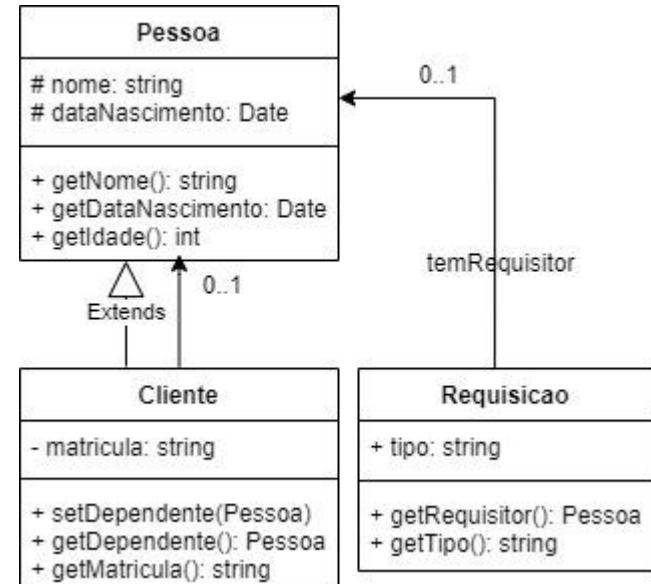
    return array_shift($manipuladores);
}
$chainAuto = getChainAuto();

//enviando requisição
echo $chain->manipular("C");
echo $chainAuto->manipular("C");
```



Chain of Responsibility (exemplo - parte 1)

```
<?php
class Pessoa {
    protected string $nome;
    protected \DateTimeInterface $dataNascimento;
    public function __construct(string $nome, \DateTimeInterface
$dataNascimento) {
        $this->nome = $nome;
        $this->dataNascimento = $dataNascimento;
    }
    public function getNome(): string {
        return $this->nome;
    }
    public function getDataNascimento(): \DateTimeInterface {
        return $this->dataNascimento;
    }
    public function getIdade(): int {
        return (new \DateTime())->diff($this->dataNascimento)->y;
    }
}
```



Chain of Responsibility (exemplo - parte 2)

```
class Cliente extends Pessoa {

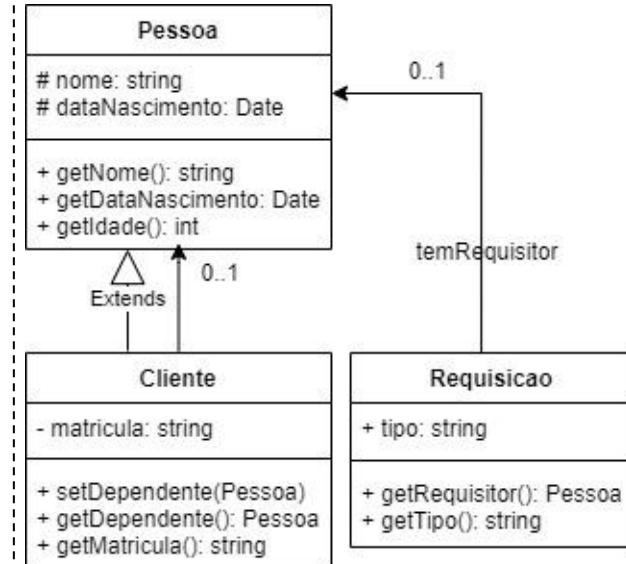
    private ?Pessoa $dependente;
    private string $matricula;

    public function __construct(string $nome, \DateTimeInterface $dataNascimento, string $matricula,
        Pessoa $dependente = null) {
        parent::__construct($nome, $dataNascimento);
        $this->matricula = $matricula;
        $this->dependente = $dependente;
    }

    public function setDependente(Pessoa $dependente): void {
        $this->dependente = $dependente;
    }

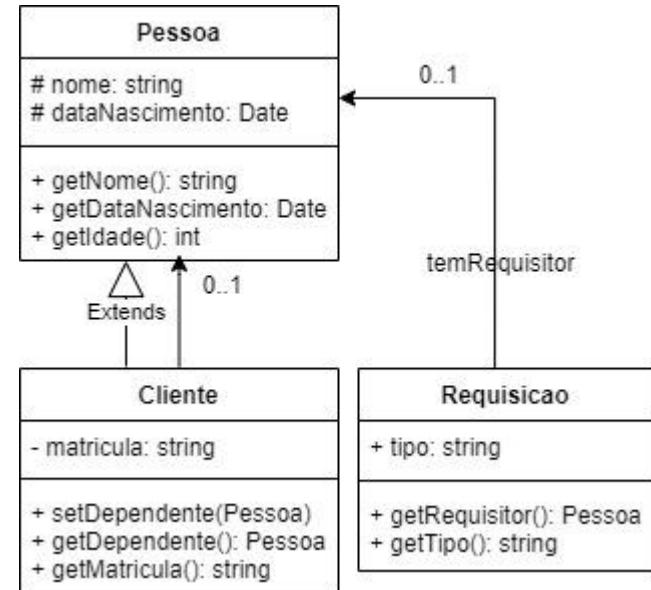
    public function getDependente(): ?Pessoa {
        return $this->dependente;
    }

    public function getMatricula(): string {
        return $this->matricula;
    }
}
```



Chain of Responsibility (exemplo - parte 3)

```
class Requisicao {  
  
    private Pessoa $requisitor;  
    private string $tipo;  
  
    public function __construct(Pessoa $requisitor, string $tipo) {  
        $this->requisitor = $requisitor;  
        $this->tipo = $tipo;  
    }  
  
    public function getRequisitor(): Pessoa {  
        return $this->requisitor;  
    }  
  
    public function getTipo(): string {  
        return $this->tipo;  
    }  
}
```



Chain of Responsibility (exemplo - parte 4)

```
interface Manipulador {  
    public function setProximoManipulador(Manipulador $proximoManipulador): Manipulador;  
    public function podeManipular(Requisicao $requisicao): bool;  
}  
  
abstract class ManipuladorBase implements Manipulador {  
  
    private ?Manipulador $proximoManipulador = null;  
  
    public final function manipular(Requisicao $requisicao) : ?string{  
        if ($this->podeManipular($requisicao)) {  
            return $this->processar($requisicao);  
        }  
        if ($this->proximoManipulador != null) {  
            return $this->proximoManipulador->manipular($requisicao);  
        }  
        return null;  
    }  
    public final function getProximoManipulador(): ?Manipulador {  
        return $this->proximoManipulador;  
    }  
    public final function setProximoManipulador(Manipulador $proximoManipulador): Manipulador {  
        $this->proximoManipulador = $proximoManipulador;  
        return $proximoManipulador; //fluent interface  
    }  
    protected abstract function processar(Requisicao $requisicao) : string;  
}
```

Chain of Responsability (exemplo - parte 5)

```
class AtendimentoNovoCliente extends ManipuladorBase {

    public function podeManipular(Requisicao $requisicao): bool {
        return get_class($requisicao->getRequisitor()) == Pessoa::class;
    }

    protected function processar(Requisicao $requisicao) : string{
        return "atendendo novo cliente solicitando: {$requisicao->getTipo()}";
    }
}

class AtendimentoClienteSemDependente extends ManipuladorBase {

    public function podeManipular(Requisicao $requisicao): bool {
        return get_class($requisicao->getRequisitor()) == Cliente::class && !($requisicao->getRequisitor()->getDependente());
    }

    protected function processar(Requisicao $requisicao) : string {
        return "atendendo cliente sem dependente solicitando: {$requisicao->getTipo()}";
    }
}

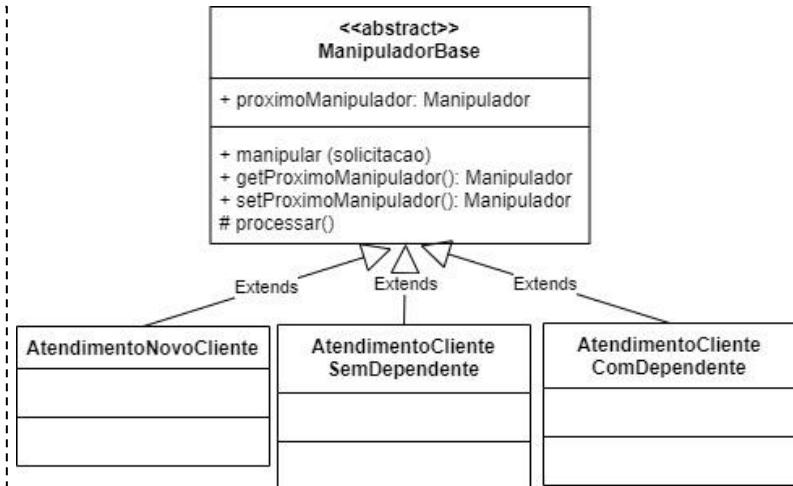
class AtendimentoClienteComDependente extends ManipuladorBase {
    public function podeManipular(Requisicao $requisicao): bool {
        return get_class($requisicao->getRequisitor()) == Cliente::class && $requisicao->getRequisitor()->getDependente();
    }

    protected function processar(Requisicao $requisicao) : string {
        return "atendendo cliente {$requisicao->getRequisitor()->getNome()} com o dependente {$requisicao->getRequisitor()->getDependente()->getNome()}";
    }
}
```

Chain of Responsibility (exemplo - parte 6)

```
<?php
$pessoa1 = new Pessoa("João", \DateTime::createFromFormat('d/m/Y', '20/01/1988'));
$cliente1 = new Cliente("Maria", \DateTime::createFromFormat('d/m/Y', '01/10/1977'), '001');
$cliente2 = new Cliente("José", \DateTime::createFromFormat('d/m/Y', '01/10/1977'), '002',
    new Pessoa("Enzo", \DateTime::createFromFormat('d/m/Y', '14/09/2018')));
$chain = new AtendimentoNovoCliente();
$chain->setProximoManipulador(new AtendimentoClienteSemDependente())
->setProximoManipulador(new AtendimentoClienteComDependente());

echo $chain->manipular(new Requisicao($pessoa1, "informações"));
echo "<br>" . PHP_EOL;
echo $chain->manipular(new Requisicao($cliente1, "dúvida"));
echo "<br>" . PHP_EOL;
echo $chain->manipular(new Requisicao($cliente2, "reclamação"));
```



Command

Command é um pattern comportamental que transforma uma solicitação em um objeto independente que contém todas as informações sobre a solicitação. Essa transformação permite parametrizar métodos com diferentes solicitações, atrasar ou enfileirar a execução de uma solicitação e oferecer suporte a operações que podem ser desfeitas (SHVETS, 2019).

Participantes:

1. **Command**: declara a interface para executar uma operação
2. **Command Concreto**: vincula a ação ao Receptor
3. **Receptor**: recebe as ações do Command
4. **Invocador**: lida com uma coleção de comandos e determina quando os comandos são executados.
5. **Cliente**: gerencia interações entre Receptor / Comando e Comando / Invocador.



Command (conceitual - parte 1)

```
<?php
interface Command {
    public function executar();
}
interface Receptor {
    public function fazAlgo();
}
class ConcreteCommand1 implements Command {
    private Receptor $receptor;
    public function __construct(Receptor $receptor) {
        $this->receptor = $receptor;
    }
    public function executar() {
        $this->receptor->fazAlgo();
    }
}
class Invocador {
    private ?Command $command;
    public function __construct(Command $command) {
        $this->command = $command;
    }
    public function setCommand(Command $command): void {
        $this->command = $command;
    }
    public function executarCommand(){
        $this->command->executar();
    }
}
```

Command (conceitual - parte 2)

```
class Receptor1 implements Receptor {  
  
    public function fazAlgo(){  
        echo "fazendo algo !";  
    }  
  
}  
  
$receptor = new Receptor1();  
$command = new ConcreteCommand1($receptor);  
$invocador = new Invocador($command);  
$invocador->executarCommand();
```

Command (exemplo - parte 1)

```
class Configuracao {//similar ao utilizado no singleton}
class Conexao {//similar ao utilizado no singleton}
//commands
abstract class Command {
    protected Receptor $receptor;
    public function __construct(Receptor $receptor) {
        $this->receptor = $receptor;
    }
    abstract function executar();
}

class CommandLogar extends Command {
    public string $texto;
    public function __construct(Receptor $receptor, $texto = "") {
        parent::__construct($receptor);
        $this->texto = $texto;
    }
    public function executar() {
        $this->receptor->logar($this->texto);
    }
}

class CommandLimpar extends Command {
    public function executar() {
        $this->receptor->limparTudo();
    }
}
```

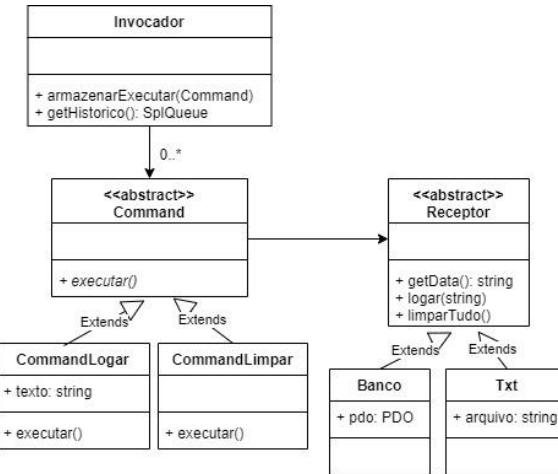
Command (exemplo - parte 2)

```
abstract class Receptor {
    protected function getData(): string {
        return (new \DateTime())->format('Y-m-d H:i:s');
    }
    abstract function logar(string $texto);
    abstract function limparTudo();
}

class Banco extends Receptor {
    private ?\PDO $pdo = null;
    public function __construct() {
        $this->pdo = Conexao::getInstance()->getPdo();
    }
    public function limparTudo() {
        $this->pdo->prepare("DELETE FROM log")->execute();
    }
    public function logar(string $texto) {
        $this->pdo
            ->prepare("INSERT INTO log (data, texto) VALUES (:data, :texto)")
            ->execute([
                'data' => $this->getData(),
                'texto' => $texto
            ]);
    }
}
```

Command (exemplo - parte 3)

```
class Txt extends Receptor {
    private string $arquivo = "log.txt";
    public function __construct() {
        touch($this->arquivo);
    }
    public function limparTudo() {
        file_put_contents($this->arquivo, "");
    }
    public function logar(string $texto) {
        file_put_contents($this->arquivo, $this->getData() . "\n" . $texto . PHP_EOL, FILE_APPEND | LOCK_EX);
    }
}
class Invocador {
    private \SplQueue $commands;
    public function __construct() {
        $this->commands = new \SplQueue();
    }
    public function armazenarExecutar(Command $command) {
        $this->commands->push($command);
        $command->executar();
    }
    public function getHistorico(): \SplQueue {
        return $this->commands;
    }
}
$receptor = new Banco(); //Txt();
$command = new CommandLogar($receptor, "testando"); //CommandLimpar
$invocador = new Invocador();
$invocador->armazenarExecutar($command);
```

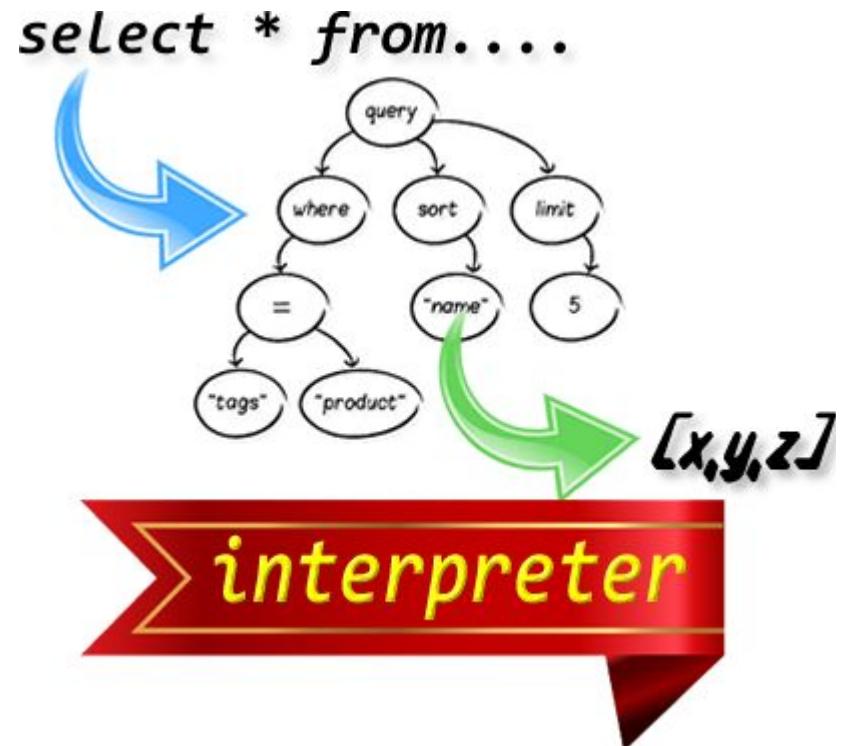


Interpreter

O Interpreter é um pattern comportamental que descreve como definir uma gramática para uma “linguagem simples”, representando sentenças desta linguagem e interpretando estas sentenças (GAMMA et al, 1994);

O Interpreter é um *design pattern* que especifica como **analisar** sentenças de uma linguagem. A ideia básica é ter uma classe para cada símbolo (**terminal ou não terminal**) em uma linguagem de computador especializada.

Permite representar uma sentença de uma linguagem por uma Árvore Binária de Expressão (*Binary Expression Tree*);



Interpreter

Os participantes de uma implementação típica do pattern Interpreter são:

AbstractExpression: define a interface do intérprete e estipula a operação de interpretação do intérprete. A interface Interpret, como o próprio nome sugere, é usada especificamente para explicar as funções a serem implementadas pelo interpretador. (Por exemplo, a interface Interpret no interpretador de adição é para completar a função de adição de dois operandos).

TerminalExpression: interpretador de terminador, usado para implementar as operações relacionadas ao terminador nas regras gramaticais, não contém mais outros interpretadores, se você usar o modo de combinação para construir uma árvore de sintaxe abstrata, é equivalente ao objeto folha no modo de combinação, você pode Existem vários intérpretes de símbolo de terminal.

NonterminalExpression: um intérprete não terminal, usado para implementar operações não relacionadas ao terminal em regras gramaticais, geralmente um intérprete corresponde a uma regra gramatical e pode conter outros intérpretes. Se você usar o modo de combinação para construir uma árvore de sintaxe abstrata, é equivalente a uma combinação de objetos combinados no padrão. Pode haver vários intérpretes não terminais.

Contexto: Contexto, geralmente contém dados ou funções comuns exigidas por cada intérprete. Este contexto desempenha um papel muito importante no modo de intérprete. Geralmente é usado para transferir dados compartilhados por todos os intérpretes, e intérpretes posteriores podem obter esses valores aqui.

Cliente: O cliente refere-se ao cliente que usa o intérprete. Normalmente, as expressões feitas de acordo com a sintaxe da linguagem são convertidas em uma árvore sintática abstrata descrita pelo objeto intérprete, e então a operação de interpretação é chamada.

Interpreter (conceitual - parte 1)

```
<?php
class Contexto {
    public string $valor = "";
    public function __construct(string $valor){
        $this->valor = $valor;
    }
}

interface Expression {
    public function interpretar(Contexto $contexto);
}

class ExpressaoTerminal implements Expression {
    public function interpretar(Contexto $contexto) {
        echo "interpretando {$contexto->valor}";
    }
}

class ExpressaoNaoTerminal implements Expression {
    public function interpretar(Contexto $contexto) {
        echo "interpretando {$contexto->valor}";
    }
}
```

Interpreter (conceitual - parte 2)

```
//*****cliente
$contexto = new Contexto("1234");

// populando 'abstract syntax tree'
$list = [];
$list[] = new ExpressaoTerminal();
$list[] = new ExpressaoNaoTerminal();
$list[] = new ExpressaoTerminal();
$list[] = new ExpressaoTerminal();

// Interpretar
foreach ($list as $key => $expressao) {
    $expressao->interpretar($contexto);
}
```

Interpreter (exemplo - parte 1)

```
interface Expressao {
    public function interpretar(array $contexto): int;
}

class ExpressaoTerminal implements Expressao {

    private string $nome;

    public function __construct(string $nome) {
        $this->nome = $nome;
    }
    public function interpretar(array $contexto): int {
        return $contexto[$this->nome] ?? throw new \DomainException(" o valor de {$this->nome} não existe
nos dados");
    }
}

abstract class OperacaoAritimetrica implements Expressao { //não-terminal

    protected Expressao $expressaoEsquerda;
    protected Expressao $expressaoDireita;
    public function __construct(Expressao $expressaoEsquerda, Expressao $expressaoDireita) {
        $this->expressaoEsquerda = $expressaoEsquerda;
        $this->expressaoDireita = $expressaoDireita;
    }
    public static abstract function getSimbolo(): string;
}
```

Interpreter (exemplo - parte 2)

```
class Adicao extends OperacaoAritimetica {

    public function interpretar(array $contexto): int {
        return $this->expressaoEsquerda->interpretar($contexto) +
            $this->expressaoDireita->interpretar($contexto);
    }

    public static function getSimbolo(): string {
        return "+";
    }
}

class Subtracao extends OperacaoAritimetica {

    public function interpretar(array $contexto): int {
        return $this->expressaoEsquerda->interpretar($contexto) -
            $this->expressaoDireita->interpretar($contexto);
    }

    public static function getSimbolo(): string {
        return "-";
    }
}
```

Interpreter (exemplo - parte 3)

```
class Calculadora { //cliente
    private Expressao $expressao;
    public function __construct(string $texto) {
        $stack = new \SplStack();
        $expressaoEsquerda = null;
        $expressaoDireita = null;
        $classesOperacoes = array_filter(get_declared_classes(),
            fn($class) => is_subclass_of($class, OperacaoAritmetica::class));

        for ($i = 0; $i < strlen($texto); $i++) {
            $operacoes = array_filter($classesOperacoes, fn($s) =>
                $s::getSimbolo() == $texto[$i]);

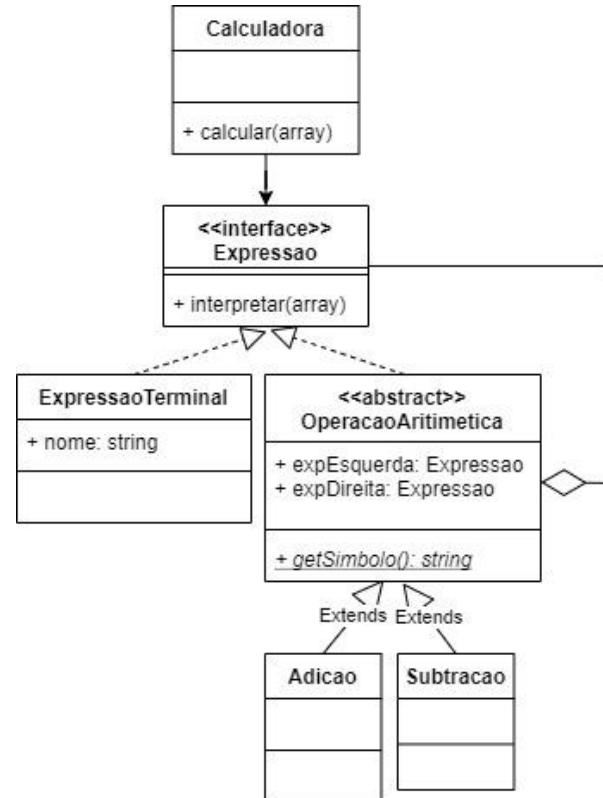
            $classeNaoTerminal = array_shift($operacoes);

            if (!$classeNaoTerminal) {
                $stack->push(new ExpressaoTerminal($texto[$i]));
            } else {
                $expressaoEsquerda = $stack->pop();
                $expressaoDireita = new ExpressaoTerminal($texto[++$i]);
                $stack->push(new $classeNaoTerminal($expressaoEsquerda, $expressaoDireita));
            }
        }
        $this->expressao = $stack->pop();
    }

    public function calcular(array $dados) {
        return $this->expressao->interpretar($dados);
    }
}
```

Interpreter (exemplo - parte 4)

```
$calculator = new Calculadora("a+b+a");
$dados = [];
$dados["a"] = 8;
$dados["b"] = 22;
echo "O resultado é {$calculator->calcular($dados)}";
```



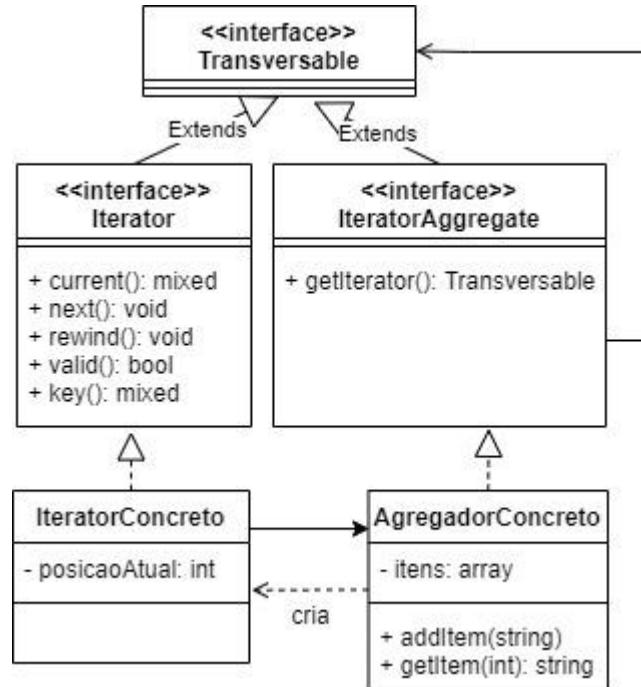
Iterator

Iterator é um pattern comportamental que permite atravessar elementos de uma coleção sem expor sua representação subjacente (SHVETS, 2019).



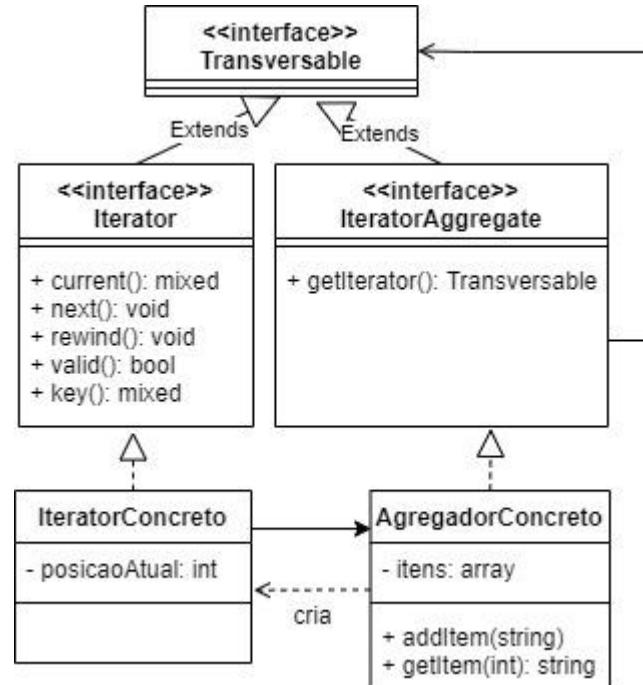
Iterator (conceitual - parte 1)

```
class IteratorConcreto implements \Iterator {  
  
    private int $posicaoAtual = 0;  
    private AgregadorConcreto $agregadorConcreto;  
  
    public function __construct(AgregadorConcreto $agregadorConcreto) {  
        $this->agregadorConcreto = $agregadorConcreto;  
    }  
    public function current() {  
        return $this->agregadorConcreto->getItem($this->posicaoAtual);  
    }  
    public function next() {  
        $this->posicaoAtual++;  
    }  
    public function rewind() {  
        $this->posicaoAtual = 0;  
    }  
    public function valid(): bool {  
        return !is_null($this->agregadorConcreto->getItem($this->posicaoAtual));  
    }  
    public function key(): int {  
        return $this->posicaoAtual;  
    }  
}
```



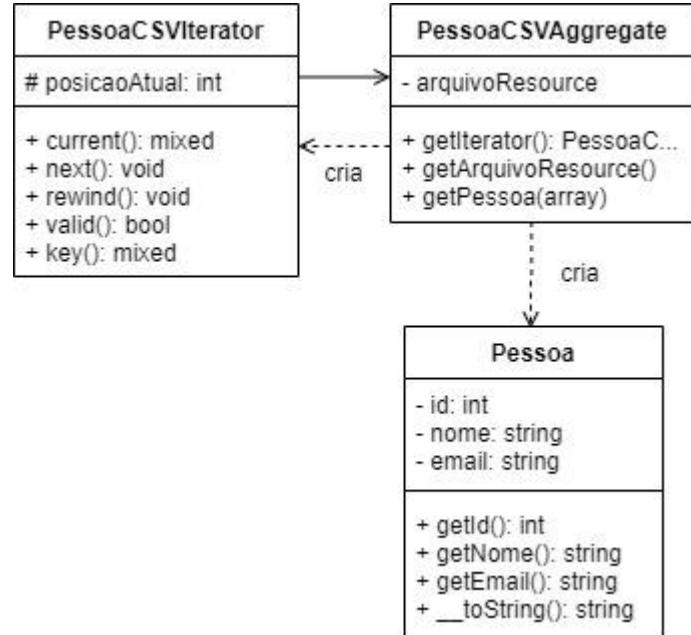
Iterator (conceitual - parte 2)

```
class AgregadorConcreto implements \IteratorAggregate {  
  
    private array $itens = [];  
    public function getIterator() {  
        return new IteratorConcreto($this);  
    }  
    public function addItem(string $string): void {  
        $this->itens[] = $string;  
    }  
    public function getItem(int $key) : ?string {  
        if (isset($this->itens[$key])) {  
            return $this->itens[$key];  
        }  
        return null;  
    }  
}  
  
//cliente  
$agregadorConcreto = new AgregadorConcreto();  
$agregadorConcreto->addItem('Brincadeira em excesso');  
$agregadorConcreto->addItem('Mosantos de Villar');  
$agregadorConcreto->addItem('Bobo da Corte');  
$agregadorConcreto->addItem('Meus benefícios');  
  
foreach ($agregadorConcreto as $texto) {  
    var_dump($texto);  
}
```



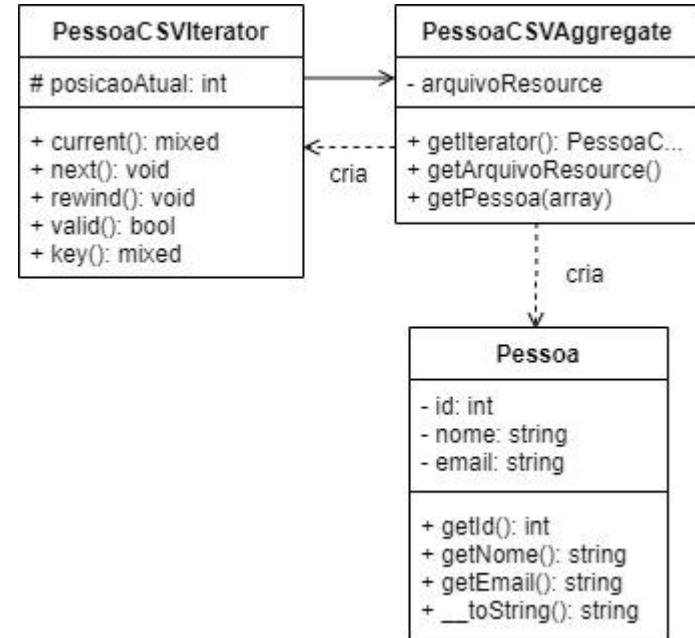
Iterator (exemplo - parte 1)

```
<?php
class Pessoa{
    private int $id;
    private string $nome;
    private string $email;
    public function __construct(int $id, string $nome, string $email){
        $this->id = $id;
        $this->nome = $nome;
        $this->email = $email;
    }
    public function getId(): int {
        return $this->id;
    }
    public function getName(): string {
        return $this->nome;
    }
    public function getEmail(): string {
        return $this->email;
    }
    public function __toString(): string {
        return sprintf('%d, %s, %s', $this->id, $this->nome, $this->email);
    }
}
```



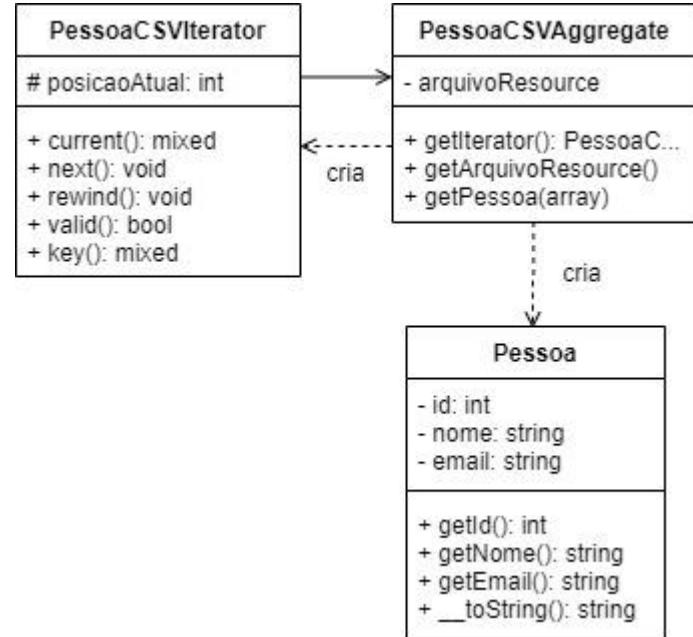
Iterator (exemplo - parte 2)

```
class PessoaCSVIterator implements \Iterator {
    private PessoaCSVAggregate $pessoaAggregate;
    protected int $posicaoAtual = -1;
    public function __construct(PessoaCSVAggregate $pessoaAggregate) {
        $this->pessoaAggregate = $pessoaAggregate;
    }
    public function rewind(): void {
        $this->posicaoAtual = -1;
        rewind($this->pessoaAggregate->getArquivoResource());
        fgets($this->pessoaAggregate->getArquivoResource()); //pular a primeira linha
    }
    public function current(): Pessoa {
        $pessoa = $this->pessoaAggregate->getPessoa(
            fgetcsv($this->pessoaAggregate->getArquivoResource())
        );
        $this->posicaoAtual++;
        return $pessoa;
    }
    public function key(): int {
        return $this->posicaoAtual;
    }
    public function next(): bool {
        return !feof($this->pessoaAggregate->getArquivoResource());
    }
    public function valid(): bool {
        if (!$this->next()) {
            fclose($this->pessoaAggregate->getArquivoResource());
            return false;
        }
        return true;
    }
}
```



Iterator (exemplo - parte 3)

```
class PessoaCSVAggregate implements \IteratorAggregate {  
  
    private $arquivoResource;  
  
    public function __construct(string $file) {  
        $this->arquivoResource = fopen($file, 'rb');  
    }  
    public function getIterator(): PessoaCsvIterator {  
        return new PessoaCsvIterator($this);  
    }  
    public function getArquivoResource():  
    {  
        return $this->arquivoResource;  
    }  
    public function getPessoa(array $array): Pessoa {  
        return new Pessoa($array[0], $array[1], $array[2]);  
    }  
}  
//cliente  
$csv = new PessoaCSVAggregate('emails.csv');  
foreach ($csv as $key => $pessoa){  
    echo "key : $key <br/>";  
    echo "$pessoa <br/>";  
}
```



Mediator

O Mediator (mediador) é um design pattern comportamental que visa reduzir dependências caóticas entre objetos. O pattern restringe as comunicações diretas entre os objetos e os força a colaborar apenas através de um objeto mediador (SHVETS, 2019).

Objetos fortemente acoplados são difíceis de implementar, alterar, testar e reutilizar porque se referem e conhecem muitos objetos diferentes (WIKIPEDIA, 2020).



Mediator (conceitual - parte 1)

```
interface Mediador {  
    public function intermediar(ComponenteAssociado $remetente, string $evento): void;  
}  
  
class MediadorConcreto implements Mediador {  
    private ComponenteAssociadoA $componenteA;  
    private ComponenteAssociadoB $componenteB;  
    public function __construct(ComponenteAssociadoA $c1, ComponenteAssociadoB $c2) {  
        $this->componenteA = $c1;  
        $this->componenteA->setMediador($this);  
        $this->componenteB = $c2;  
        $this->componenteB->setMediador($this);  
    }  
    public function intermediar(ComponenteAssociado $remetente, string $evento): void {  
        if ($evento == "A") {  
            echo "Mediador reage ao evento A e executa: <br>".PHP_EOL;  
            $remetente->setEstado("A");  
            $this->componenteB->fazAlgoZ();  
        }  
        if ($evento == "D") {  
            echo "Mediador reage ao evento D e executa:".PHP_EOL;  
            $remetente->setEstado("B");  
            $this->componenteA->fazAlgoY();  
            $this->componenteB->fazAlgoZ();  
        }  
    }  
}
```

Mediator (conceitual - parte 2)

```
//colegas
abstract class ComponenteAssociado {

    protected ?Mediador $mediador;
    protected ?string $estado;

    public function __construct(Mediador $mediador = null) {
        $this->mediador = $mediador;
    }

    public function setMediador(Mediador $mediador): void {
        $this->mediador = $mediador;
    }

    public function setEstado($estado){
        $this->estado = $estado;
    }

    public function getEstado($estado){
        return $this->estado;
    }
}
```

Mediator (conceitual - parte 3)

```
class ComponenteAssociadoA extends ComponenteAssociado {
    public function fazAlgoX(): void {
        echo "executando ".__FUNCTION__."<br>".PHP_EOL;
        $this->mediador->intermediar($this, "A");
    }
    public function fazAlgoY(): void {
        echo "executando ".__FUNCTION__."<br>".PHP_EOL;
        $this->mediador->intermediar($this, "B");
    }
}
class ComponenteAssociadoB extends ComponenteAssociado {
    public function fazAlgoZ(): void {
        echo "executando ".__FUNCTION__."<br>".PHP_EOL;
        $this->mediador->intermediar($this, "C");
    }
    public function fazAlgoXY(): void {
        echo "executando ".__FUNCTION__."<br>".PHP_EOL;
        $this->mediador->intermediar($this, "D");
    }
}
//cliente
$c1 = new ComponenteAssociadoA();
$c2 = new ComponenteAssociadoB();
$mediador = new MediadorConcreto($c1, $c2);
$c1->fazAlgoX();
echo "<br/>";
$c2->fazAlgoXY();
```

Mediator (exemplo - parte 1)

```
<?php
Trait CliColorido {

    abstract function getCor(): int;

    function printar(string $texto) {
        echo "\033[{$this->getCor()}m$texto" . PHP_EOL;
        fgets(STDIN);
    }
}

abstract class ControleTrafegoAereoMediator {

    protected array $aeronaves = [];

    abstract function existeOutraAeronaveComPrioridadeParaPousos(Aeronave $aeronave): bool;
    abstract function receberLocalizacao(Aeronave $aeronave);
    abstract function registrar(Aeronave $aeronave);
    abstract function notificarPousos(Aeronave $aeronave);
    abstract function notificarPousosConcluidos(Aeronave $aeronave);

    public final function getOutrasAeronaves(Aeronave $aeronave): array {
        return array_filter($this->aeronaves, fn($a) => $a != $aeronave);
    }
}
```

Mediator (exemplo - parte 2)

```
class ControleTrafegoAereoGaleao extends ControleTrafegoAereoMediator {
    use CliColorido;
    public function registrar(Aeronave $aeronave) {
        $this->printar(">Trafego Aéreo: Olá {$aeronave->numeroVoo}! Estou com você no radar ! Diga-me se você quer pousar ou mudar sua localização!");
        $this->aeronaves[] = $aeronave; //
    }
    public function notificarPouso(Aeronave $aeronave) {
        $this->printar(">Trafego Aéreo: Recebido {$aeronave->numeroVoo}! Vou avisar que você está pousando para os outros pilotos!");
        foreach ($this->getOutrasAeronaves($aeronave) as $outraAeronave) {
            $outraAeronave->notificarAeronavesSobrePouso($aeronave->numeroVoo);
        }
    }
    public function notificarPousoConcluido(Aeronave $aeronave) {
        $this->printar(">Trafego Aéreo: Recebido {$aeronave->numeroVoo}! Vou avisar que você já pousou os outros pilotos!");
        foreach ($this->getOutrasAeronaves($aeronave) as $outraAeronave) { //usar_map
            $outraAeronave->notificarSobrePistaDisponivel();
        }
    }
    public function existeOutraAeronaveComPrioridadeParaPouso(Aeronave $remetente): bool {
        $this->printar(">Trafego Aéreo: Eu recebi o seu pedido {$remetente->numeroVoo}, estou verificando as prioridades para pousar... ");
        foreach ($this->getOutrasAeronaves($remetente) as $outraAeronave) {
            if ($outraAeronave->altitude <= $remetente->altitude) {
                $this->printar(">Trafego Aéreo: Você precisa aguardar! O voô {$outraAeronave->numeroVoo} tem prioridade para pousar !");
                return true;
            }
        }
        return false;
    }
    public function receberLocalizacao(Aeronave $aeronave) {
        $this->printar(">Trafego Aéreo: Recebido {$aeronave->numeroVoo}! Vou notificar sua nova localização para os outros pilotos!");
        foreach ($this->getOutrasAeronaves($aeronave) as $outraAeronave) {
            $outraAeronave->verificarDistancia($aeronave->altitude);
        }
    }
    public function getCor(): int {
        return 37;
    }
}
```

Mediator (exemplo - parte 3)

```
abstract class Aeronave {
    use CliColorido;
    private ControleTrafegoAereoMediator $mediator; public int $altitude; public string $numeroVoo;
    public function __construct(ControleTrafegoAereoMediator $mediator, string $numeroVoo, int $altitude) {
        $this->mediator = $mediator;
        $this->numeroVoo = $numeroVoo;
        $this->altitude = $altitude;
        $this->mediator->registrar($this);
    }
    public final function aterrissar() {
        $this->printar(">{$this->numeroVoo}: solicito autorização para pouso.");
        if ($this->mediator->existeOutraAeronaveComPrioridadeParaPousos($this)) {
            $this->printar(">{$this->numeroVoo}: recebido! Eu irei aguardar");
            return;
        }
        $this->printar(">{$this->numeroVoo}: ok! Vou iniciar os procedimentos para pouso");
        $this->mediator->notificarPousos($this);
        $this->printar(">{$this->numeroVoo}: estou aterrissando !");
        $this->printar(">{$this->numeroVoo}: acabei de pousar ! ");
        $this->mediator->notificarPousosConcluido($this);
    }
    public function notificarAeronavesSobrePousos(string $numeroVoo) { //trocar para serNotificado....
        $this->printar(">{$this->numeroVoo}: Recebido! Favor me infomar quando o voo {$numeroVoo} pousar");
    }
    public function notificarSobrePistaDisponivel() {
        $this->printar(">{$this->numeroVoo}: Recebido! Estou próximo ao aeroporto! Eu tentarei pousar em 20 minutos!");
    }
    public function verificarDistancia(int $altitudeOutraAeronave) {
        $this->printar(">{$this->numeroVoo}: Recebido! Verificando Distância de Segurança");
        if (abs($altitudeOutraAeronave - $this->altitude) < 100) {
            $this->printar(">{$this->numeroVoo}: Estou próximo de outra aeronave, vou diminuir a velocidade");
        }
    }
    public final function setAltitude(int $altitude) {
        $this->altitude += $altitude;
        $this->printar(">{$this->numeroVoo}: Subindo mais $altitude pés para a altitude de {$this->altitude} pés!");
        $this->mediator->receberLocalizacao($this);
    }
}
```

Mediator (exemplo - parte 4)

```
//"colleagues concretos"
class Boeing747 extends Aeronave {
    public function getCor(): int {
        return 32;
    }
}

class Boeing787 extends Aeronave {
    public function getCor(): int {
        return 33;
    }
}

class Boeing777 extends Aeronave {
    public function getCor(): int {
        return 34;
    }
}

class AirbusA320 extends Aeronave {
    public function getCor(): int {
        return 31;
    }
}
```

Mediator (exemplo - parte 5)

```
echo "\033[37mIniciando...." . PHP_EOL;
echo <<<ASCII
      _=\
      \|____\_\_
-=\c`~~~~~") SEJA BEM VINDO !
  `~~~~~/~~~
    ==/ /
  ' '
ASCII.PHP_EOL;
$mediator = new ControleTrafegoAereoGaleao();
$vooA = new AirbusA320($mediator, "A37685", 2000);
$vooB = new Boeing787($mediator, "B67329", 1500);
$vooC = new Boeing747($mediator, "C234545", 1460);
$vooD = new Boeing777($mediator, "D23488", 1300);
$vooA->aterristar();
$vooB->aterristar();
$vooC->setAltitude(100);
$vooD->aterristar();
echo "\033[37m" . PHP_EOL;
```

Memento

“Memento: um objeto que você guarda para se lembrar de uma pessoa, local ou evento” (Dicionário de Cambridge).

Memento é um pattern comportamental que permite salvar e restaurar o estado anterior de um objeto sem revelar os detalhes de sua implementação (SHVETS, 2019).

Este Pattern possui três participantes:

1. **Originator:** objeto capaz produzir instantâneos de seu próprio estado, bem como restaurar seu estado de instantâneos quando necessário.
2. **Memento:** objeto de valor que como um instantâneo do estado do originador. É uma prática comum tornar a lembrança imutável e transmitir os dados apenas uma vez, por meio do construtor.
3. **Caretaker (zelador):** sabe não apenas "quando" e "por que" capturar o estado do originador, mas também quando o estado deve ser restaurado (SHVETS, 2019).



Memento (conceitual - parte 1)

```
interface Originator {
    public function salvar(): Memento;
    public function restaurar(Memento $memento): void;
    public function __toString(): string;
}

class OriginatorConcreto implements Originator {

    private array $estado = [];
    public function __construct(string $texto) {
        $this->estado[] = $texto;
    }
    public function setEstado($texto): void {
        $this->estado[] = $texto;
    }
    public function salvar(): Memento { //snapshot
        return new Memento($this->estado);
    }
    public function restaurar(Memento $memento): void {
        $this->estado = $memento->getEstado();
    }
    public function __toString(): string {
        return "printando: " . implode(" ", $this->estado) . PHP_EOL;
    }
}
```

Memento (conceitual - parte 2)

```
class Memento {
    private $estado;
    public function __construct($estado) {
        $this->estado = $estado;
    }
    public function getEstado() {
        return $this->estado;
    }
}

class Caretaker {
    private \SplStack $mementos;
    private Originator $originator;
    public function __construct(Originator $originator) {
        $this->mementos = new \SplStack();
        $this->originator = $originator;
    }
    public function persistirEstadoAtual(): void {
        $this->mementos->push($this->originator->salvar());
    }
    public function print() {
        echo $this->originator;
    }
    public function desfazer(): void {
        if (!$this->mementos->count()) {
            return;
        }
        $memento = $this->mementos->pop();
        $this->originator->restaurar($memento);
    }
}
```

Memento (conceitual - parte 3)

```
//cliente
$originator = new OriginatorConcreto("ABC123");
$caretaker = new Caretaker($originator);
$caretaker->persistirEstadoAtual();
$originator->setEstado("XYZ456");
$caretaker->persistirEstadoAtual();
$originator->setEstado("WSD999");
$caretaker->persistirEstadoAtual();
$originator->setEstado("POO111");
// $caretaker->persistirEstadoAtual();
$caretaker->print();
$caretaker->desfazer();
$caretaker->desfazer();
$caretaker->print();
```

Memento (exemplo - parte 1)

```
<?php
class Ingrediente {
    private string $nome;
    private float $valor;
    public function __construct(string $nome, float $valor) {
        $this->nome = $nome;
        $this->valor = $valor;
    }
    public function getValor(): float {
        return $this->valor;
    }
    public function getName(): string {
        return $this->nome;
    }
    public function __toString(): string {
        return $this->nome;
    }
}
class Destilado extends Ingrediente {
    private string $teorAlcoolico;
    public function __construct(string $nome, float $valor, float $teorAlcoolico) {
        parent::__construct($nome, $valor);
        $this->teorAlcoolico = $teorAlcoolico;
    }
    public function getTeorAlcoolico(): float {
        return $this->teorAlcoolico;
    }
}
```

Memento (exemplo - parte 2)

```
interface Originator {
    public function salvar(): Memento;
    public function restaurar(Memento $memento): void;
    public function __toString(): string;
}

class Coquetel implements Originator {
    private array $estado = [];
    public function __construct(Destilado $destilado) {
        $this->estado[] = $destilado;
    }
    public function addIngrediente(Ingrediente $ingrediente): void {
        $this->estado[] = $ingrediente;
    }
    public function __toString(): string {
        return "Coquetel R$ {$this->getCustoTotal()} feito de: " . implode(", ", $this->estado) . PHP_EOL;
    }
    private function getCustoTotal(): float {
        return array_reduce($this->estado,
            fn($total, Ingrediente $r) => $total + $r->getValor(), 0);
    }
    public function salvar(): Memento { //snapshot
        return new Memento($this->estado);
    }
    public function restaurar(Memento $memento): void {
        $this->estado = $memento->getEstado();
    }
}
```

Memento (exemplo - parte 3)

```
class Memento {
    private $estado;
    public function __construct($estado) {
        $this->estado = $estado;
    }
    public function getEstado() {
        return $this->estado;
    }
}
class Caretaker {
    private \SplStack $mementos;
    private Originator $coquetel;
    public function __construct(Originator $coquetel) {
        $this->mementos = new \SplStack();
        $this->originator = $coquetel;
    }
    public function persistirEstadoAtual(): void {
        $this->mementos->push($this->originator->salvar());
    }
    public function print() {
        echo $this->originator;
    }
    public function desfazer(): void {
        if (!$this->mementos->count()) {
            return;
        }
        $memento = $this->mementos->pop();
        $this->originator->restaurar($memento);
    }
}
```

Memento (exemplo - parte 4)

```
//cliente
$rum = new Destilado('rum', 5.00, 70);
$cachaca = new Destilado('cachaca', 1.00, 48);
$vodka = new Destilado('cachaca', 1.99, 40);
$acucar = new Ingrediente('açucar', 0);
$limao = new Ingrediente('limão', 1.00);
$creme1 = new Ingrediente('creme de coco', 1.00);

$coquetel = new Coquetel($cachaca);
$caretaker = new Caretaker($coquetel);
$caretaker->persistirEstadoAtual();
$coquetel->addIngrediente($acucar);
$caretaker->persistirEstadoAtual();
$coquetel->addIngrediente($limao);
$caretaker->persistirEstadoAtual();
$coquetel->addIngrediente($creme1);
//\$caretaker->persistirEstadoAtual();
$caretaker->print();
$caretaker->desfazer();
$caretaker->desfazer();
$caretaker->print();
```

Observer

Observer é um pattern comportamental que permite definir um mecanismo de subscrição para notificar vários objetos sobre quaisquer eventos que ocorram no objeto que estão observando (SHVETS, 2019).

O Observer é um pattern no qual um objeto, chamado de **sujeito** (subject), mantém uma lista de seus dependentes, chamados **observadores**, e os **notifica** automaticamente sobre qualquer alteração de estado, geralmente chamando um de seus métodos (WIKIPEDIA, 2020).



Observer (conceitual - parte 1)

```
abstract class ItemInteresse implements \SplSubject {

    protected $estado;
    protected \SplObjectStorage $observers;

    public function __construct() {
        $this->observers = new \SplObjectStorage;
    }

    public function attach(\SplObserver $observer): void {
        $this->observers->attach($observer);
    }

    public function detach(\SplObserver $observer): void {
        $this->observers->detach($observer);
    }

    public function getEstado() {
        return $this->estado;
    }

}
```

Observer (conceitual - parte 2)

```
class ItemInteresseConcreto extends ItemInteresse {

    const INCIALIZADO = "INCIALIZADO";
    const FINALIZADO = "FINALIZADO";

    public function notify(): void {
        echo "Subject: Notificando observers...<br/>" . PHP_EOL;
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function inicializar(): void {
        $this->estado = self::INCIALIZADO;
        $this->notify();
    }

    public function finalizar(): void {
        $this->estado = self::FINALIZADO;
        $this->notify();
    }
}
```

Observer (conceitual - parte 3)

```
class ObservadorConcretoA implements \SplObserver {  
    public function update(\SplSubject $subject): void {  
        echo self::class . ": reagindo ao evento ! <br/>" . PHP_EOL;  
    }  
}  
  
class ObservadorConcretoB implements \SplObserver {  
    public function update(\SplSubject $subject): void {  
        if ($subject instanceof ItemInteresseConcreto && $subject->getEstado() ==  
            ItemInteresseConcreto::INCIALIZADO) {  
            echo self::class . ": reagindo ao evento ! <br/>" . PHP_EOL;  
        }  
    }  
}  
//cliente  
$subject = new ItemInteresseConcreto();  
$o1 = new ObservadorConcretoA();  
$subject->attach($o1);  
$o2 = new ObservadorConcretoB();  
$subject->attach($o2);  
$subject->inicializar();  
$subject->finalizar();
```

Observer (exemplo - parte 1)

```
class ExceptionHandler implements \SplSubject {

    private \SplObjectStorage $observers;
    private ?Throwable $throwable = null;

    function __construct() {
        $this->observers = new \SplObjectStorage;
    }
    public function attach(\SplObserver $observer): void {
        $this->observers->attach($observer);
    }
    public function detach(\SplObserver $observer): void {
        $this->observers->detach($observer);
    }
    public function notify(): void {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }
    public function handle(Throwable $t) {
        $this->throwable = $t;
        $this->notify();
    }
    public function getMessage(): ?string {
        if ($this->throwable) {
            return $this->throwable->getMessage();
        }
        return null;
    }
    public function getType(): ?string {
        if ($this->throwable) {
            return get_class($this->throwable);
        }
        return null;
    }
}
```

Observer (exemplo - parte 2)

```
class Logger implements \SplObserver {

    const ARQUIVO_LOG = 'log.txt';

    public function __construct() {
        touch(self::ARQUIVO_LOG);
    }

    public function update(\SplSubject $subject) {
        file_put_contents(self::ARQUIVO_LOG,
            $subject->getMessage() . PHP_EOL, FILE_APPEND | LOCK_EX);
    }
}

class Mailer implements \SplObserver {

    const TIPOS_PARA_EMAIL = [\RuntimeException::class, \SoapFault::class];

    public function update(\SplSubject $subject) {

        $tipo = $subject->getType();

        if (array_filter(self::TIPOS_PARA_EMAIL, fn($t) => is_a($tipo, $t, true))) {
            echo "mandando email sobre {$subject->getType()} : {$subject->getMessage()}";
            //mail() //implementação real
        }
    }
}
```

Observer (exemplo - parte 3)

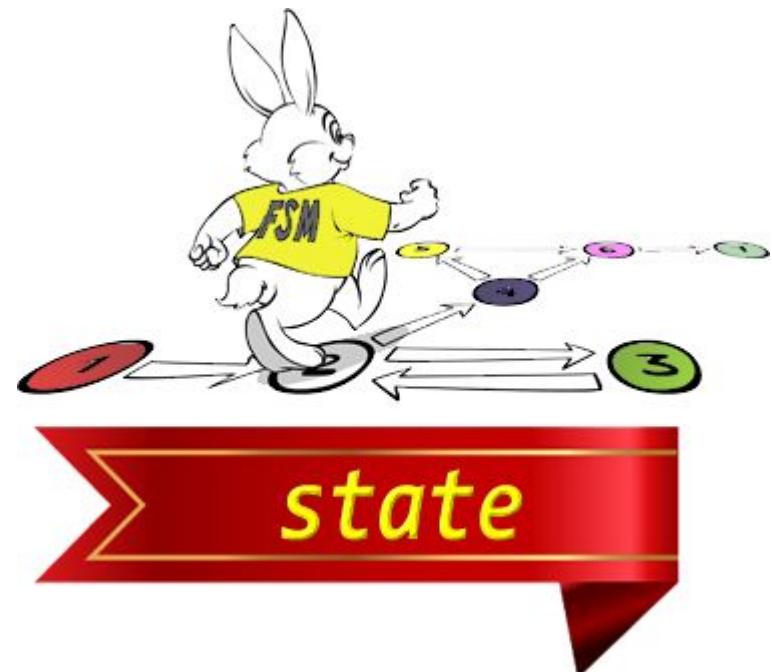
```
//cliente (montando)
$handler = new ExceptionHandler();
$handler->attach(new Logger());
$handler->attach(new Mailer());
set_exception_handler([$handler, 'handle']);

//forçando
funcaoNaoExiste();
//new \SoapClient("naoexiste.com/wsdl");
//new \PDO("mysql:errado","login","senha");
```

State

State é um pattern comportamental que permite que um objeto altere seu comportamento quando seu estado interno for alterado. Como se o objeto tivesse trocado de classe (SHEVETS, 2019).

Este pattern é próximo do conceito de máquinas de estado finita (WIKIPEDIA, 2020).



Problema

```
<?php  
  
class EntidadeQualquer {  
  
    public $estado = 'x';  
  
    function fazIsso(){  
        switch($this->estado){  
            case 'x':  
                echo 'fazendo do jeito x';  
  
                break;  
        }  
    }  
}
```

State (conceitual - parte 1)

```
interface State {
    function fazAlgoA(Contexto $contexto);
    function fazAlgoB(Contexto $contexto);
}
class EstadoConcretoA implements State {

    public function fazAlgoA(Contexto $contexto) {
        echo self::class." fazendo algo A ".PHP_EOL;
    }
    public function fazAlgoB(Contexto $contexto) {
        echo self::class." fazendo algo B e passando para o estado B ".PHP_EOL;
        $contexto->setState(new EstadoConcretoB($contexto));
    }
}

class EstadoConcretoB implements State {

    public function fazAlgoA(Contexto $contexto) {
        if($contexto->getValor() > 10){
            echo "passando para o estado de A ".PHP_EOL;
            $contexto->setState(new EstadoConcretoA($contexto));
        }else{
            echo "fazendo algoA ainda em ".self::class.PHP_EOL;
        }
    }
    public function fazAlgoB(Contexto $contexto) {
        echo "fazendo algoB de ".self::class.PHP_EOL;
    }
}
```

State (conceitual - parte 2)

```
class Contexto {
    public State $state;
    private int $valor = 0;
    public function __construct(){
        $this->state = new EstadoConcretoA();
    }
    public function setState(State $state){
        $this->state = $state;
    }
    public function fazAlgoA(){
        $this->state->fazAlgoA($this);
    }
    public function fazAlgoB(){
        $this->state->fazAlgoB($this);
    }
    public function getValor() : int {
        return $this->valor;
    }
    public function setValor(int $valor){
        $this->valor = $valor;
    }
}
$contexto = new Contexto();
$contexto->fazAlgoA();
$contexto->fazAlgoA();
$contexto->fazAlgoB();
$contexto->fazAlgoA();
$contexto->fazAlgoB();
$contexto->setValor(11);
$contexto->fazAlgoA();
$contexto->fazAlgoB();
```

State (exemplo - parte 1)

```
abstract class CaixaEletronicoState {

    protected CaixaEletronico $caixaEletronico;
    public function __construct(CaixaEletronico $caixaEletronico) {
        $this->caixaEletronico = $caixaEletronico;
    }
    abstract function sacar(float $valor);
    abstract function abastecer(float $valor);

}

class EmFuncionamento extends CaixaEletronicoState {

    public function sacar(float $valor) {
        $valorDisponivel = $this->caixaEletronico->getValorDisponivel();
        if ($valor > $valorDisponivel) {
            $valor = $valorDisponivel;
            echo("Quantia parcial !" . PHP_EOL);
        }
        echo("R$ $valor será dispensado" . PHP_EOL);
        $novoValorDisponivel = $valorDisponivel - $valor;
        $this->caixaEletronico->setValorDisponivel($novoValorDisponivel);
        if ($novoValorDisponivel == 0) {
            $this->caixaEletronico->setState(new SemDinheiro($this->caixaEletronico));
        }
    }

    public function abastecer(float $valor) {
        echo("R$ $valor foi carregado !" . PHP_EOL);
        $this->caixaEletronico->setValorDisponivel($this->caixaEletronico->getValorDisponivel() + $valor);
    }
}
```

State (exemplo - parte 2)

```
class SemDinheiro extends CaixaEletronicoState {

    public function sacar(float $valor) {
        echo("Caixa sem dinheiro" . PHP_EOL);
    }

    public function abastecer(float $valor) {
        echo("R$ $valor foi carregado !" . PHP_EOL);
        $this->caixaEletronico->setState(new EmFuncionamento($this->caixaEletronico));
        $this->caixaEletronico->setValorDisponivel(
            $this->caixaEletronico->getValorDisponivel() + $valor);
    }
}
```

State (exemplo - parte 3)

```
class CaixaEletronico {  
  
    private float $valorDisponivel = 0;  
    private CaixaEletronicoState $estadoCorrente;  
  
    public function __construct() {  
        $this->estadoCorrente = new SemDinheiro($this);  
    }  
    public function getValorDisponivel(): float {  
        return $this->valorDisponivel;  
    }  
    public function setValorDisponivel(float $valorDisponivel) {  
        $this->valorDisponivel = $valorDisponivel;  
    }  
    public function setState(CaixaEletronicoState $state) {  
        $this->estadoCorrente = $state;  
    }  
    public function getState(): CaixaEletronicoState {  
        return $this->estadoCorrente;  
    }  
    public function sacar(float $valor) {  
        $this->estadoCorrente->sacar($valor);  
    }  
    public function abastecer(float $valor) {  
        $this->estadoCorrente->abastecer($valor);  
    }  
}
```

State (exemplo - parte 4)

```
$caixaEletronico = new CaixaEletronico();
$caixaEletronico->abastecer(100);
$caixaEletronico->sacar(50);
$caixaEletronico->sacar(30);
$caixaEletronico->sacar(30);
$caixaEletronico->sacar(20);
$caixaEletronico->abastecer(50);
$caixaEletronico->sacar(50);
$caixaEletronico->sacar(50);
$caixaEletronico->sacar(50);
$caixaEletronico->sacar(50);
```

Strategy

É um pattern comportamental que permite definir uma família de algoritmos, colocar cada um deles em uma classe separada e tornar seus objetos intercambiáveis.

Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam (GAMMA et al, 1994).



Strategy (conceitual - parte 1)

```
<?php

interface Strategy {

    function executarAlgoritmo(array $entrada): array;
}

class StrategyConcretoA implements Strategy {

    public function executarAlgoritmo(array $entrada): array {
        return array_map('strtoupper', $entrada);
    }
}

class StrategyConcretoB implements Strategy {

    public function executarAlgoritmo(array $entrada): array {
        return array_map('strrev', $entrada);
    }
}
```

Strategy (conceitual - parte 2)

```
class Contexto {  
  
    private Strategy $strategy;  
    private array $dados = ['maria', 'lucas', 'thiago'];  
  
    function __construct(Strategy $strategy) {  
        $this->strategy = $strategy;  
    }  
  
    public function setStrategy(Strategy $strategy) {  
        $this->strategy = $strategy;  
    }  
  
    public function fazAlgo(): string {  
  
        $resultado = $this->strategy->executarAlgoritmo($this->dados);  
        return implode(", ", $resultado) . PHP_EOL;  
    }  
  
}  
  
$contexto = new Contexto(new StrategyConcretoA());  
echo $contexto->fazAlgo();  
$contexto->setStrategy(new StrategyConcretoB());  
echo $contexto->fazAlgo();
```

Strategy (exemplo - parte 1)

```
<?php
interface StrategyVenda {
    public function gerarValor(Produto $produto): float;
}

class StrategyVendaOrdinaria implements StrategyVenda {

    public function gerarValor(Produto $produto): float {
        return $produto->getValorCusto() + ($produto->getValorCusto() * 0.2);
    }
}

class StrategyVendaLimpezaEstoque implements StrategyVenda {

    public function gerarValor(Produto $produto): float {
        $tempoEmEstoque = $produto->getData()->diff(new \DateTime());
        if ($tempoEmEstoque->m > 2) {
            return $produto->getValorCusto() + ($produto->getValorCusto() * 0.1);
        }
        return $produto->getValorCusto() + ($produto->getValorCusto() * 0.2);
    }
}
```

Strategy (exemplo - parte 2)

```
class Produto {
    private string $nome;
    private float $valorCusto;
    private \DateTimeInterface $data;
    private float $valor;

    public function __construct(string $nome, float $valorCusto, \DateTimeInterface $data) {
        $this->nome = $nome;
        $this->valorCusto = $valorCusto;
        $this->valor = $valorCusto;
        $this->data = $data;
    }
    public function getNome(): string {
        return $this->nome;
    }
    public function getValorCusto(): float {
        return $this->valorCusto;
    }
    public function getData(): \DateTimeInterface {
        return $this->data;
    }
    public function getValor(): float {
        return $this->valor;
    }
    public function setValor(float $valor) {
        $this->valor = $valor;
    }
}
```

Strategy (exemplo - parte 3)

```
class Venda {  
  
    private array $produtos = [];  
    private \DateTimeInterface $data;  
    private StrategyVenda $strategy;  
    public function __construct(StrategyVenda $strategy) {  
        $this->strategy = $strategy;  
        $this->data = new \DateTimeImmutable();  
    }  
    public function getProdutos(): array {  
        return $this->produtos;  
    }  
    public function getData(): \DateTimeInterface {  
        return $this->data;  
    }  
    public function setStrategy(StrategyVenda $strategy) {  
        $this->strategy = $strategy;  
    }  
    public function addProduto(Produto $produto): self {  
        $produto->setValor($this->strategy->gerarValor($produto));  
        $this->produtos[] = $produto;  
        return $this;  
    }  
    public function getValorTotal(): float {  
        return array_reduce($this->getProdutos(),  
            fn($total, Produto $p) => $total + $p->getValor(), 0);  
    }  
}
```

Strategy (exemplo - parte 4)

```
//cliente
$p1 = new Produto('bala', 1.99, new \DateTime());
$p2 = new Produto('sapato', 60.88, new \DateTime('-3 months'));
$p3 = new Produto('brinquedo', 100.00, new \DateTime('-1 week'));

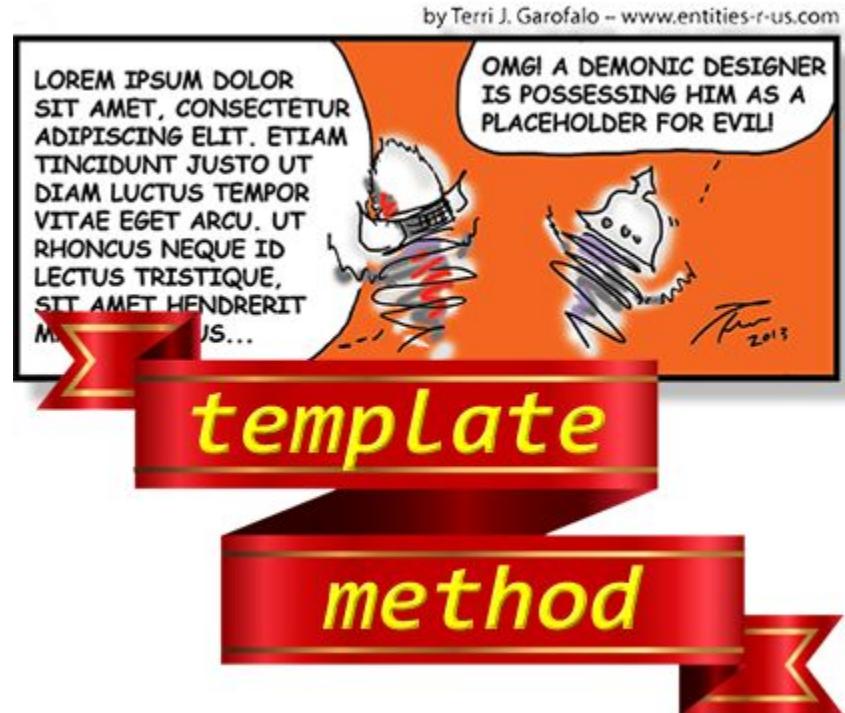
$v1 = new Venda(new StrategyVendaOrdinaria());
$v1->addProduto($p1)
    ->addProduto($p2)
    ->addProduto($p3);
echo $v1->getValorTotal();
```

Template Method

Template Method é um pattern comportamental que define o esqueleto de um algoritmo na superclasse, mas permite que as subclasses possam substituir etapas específicas do algoritmo sem alterar sua estrutura (SHVETS, 2019).

Existem três tipos de “etapas/passos” que podem ser definidos em forma de métodos no Template Method:

1. **Abstratos(as)**: precisam ser implementados pelas subclasses;
2. **Opcionais**: possuem uma implementação default, porém, podem ser sobreescritas(os) se necessário;
3. **Hooks**: similar ao opcional, porém normalmente com implementação nula; Os hooks podem ser utilizados em momentos importantes do algoritmo possibilitando uma extensão customizada nas subclasses que os(as) sobrescrevem;



Template Method (conceitual - parte 1)

```
abstract class ClasseBase {  
  
    final public function templateMethod(): void {  
        $this->hook1();  
        $this->operacaoDefault1();  
        if($this->etapaRequerida1()){  
            $this->operacaoDefault2();  
        }  
        $this->etapaRequerida2();  
        $this->hook2();  
    }  
  
    protected function operacaoDefault1(): bool {  
        return (bool) rand(0,1);  
    }  
    protected function operacaoDefault2(): void {  
        echo "fazendo __FUNCTION__ na classe ".self::class.PHP_EOL;  
    }  
    abstract protected function etapaRequerida1();  
    abstract protected function etapaRequerida2(): void;  
  
    protected function hook1(): void {}  
    protected function hook2(): void {}  
}
```

Template Method (conceitual - parte 2)

```
class ClassConcreta1 extends ClasseBase {  
  
    protected function etapaRequerida1(): void {  
        echo "fazendo __FUNCTION__ na classe ".self::class.PHP_EOL;  
    }  
    protected function etapaRequerida2(): void {  
        echo "fazendo __FUNCTION__ na classe ".self::class.PHP_EOL;  
    }  
    protected function operacaoDefault1(): bool {  
        return true;  
    }  
}  
  
class ClassConcreta2 extends ClasseBase {  
  
    protected function etapaRequerida1(): void {  
        echo "fazendo __FUNCTION__ na classe ".self::class.PHP_EOL;  
    }  
    protected function etapaRequerida2(): void {  
        echo "fazendo __FUNCTION__ na classe ".self::class.PHP_EOL;  
    }  
    protected function hook1(): void {  
        echo "fazendo __FUNCTION__ na classe ".self::class.PHP_EOL;  
    }  
}
```

Template Method (conceitual - parte 3)

```
/***cliente
echo "Utilizando a ".ClassConcreta1::class.PHP_EOL;
$o1 = new ClassConcreta1();
$o1->templateMethod();

echo "Utilizando a ".ClassConcreta2::class.PHP_EOL;
$o2 = new ClassConcreta2();
$o2->templateMethod();
```

Template Method (exemplo - 1)

```
include 'conexao.php'; //mesmo do Singleton
abstract class Paginator {
    private IPDO $pdo;
    private int $pagina;
    private int $limit;
    public function __construct(IPDO $pdo, int $pagina = 1, $limit = 5) {
        $this->pdo = $pdo;
        $this->pagina = $pagina;
        $this->limit = $limit;
    }
    public final function getPagina(): int {
        return $this->pagina;
    }
    public final function getLimit(): int {
        return $this->limit;
    }
    public final function getOffset() {
        return ($this->pagina - 1) * $this->limit;
    }
    protected abstract function getPaginateQuery(string $query, string $order = 'id'): string;
    public final function getPaginateStatement(string $query): IPDOSatement {
        $query = $this->setSQLBeforeQuery()." ".$this->getPaginateQuery($query);
        $statement = $this->pdo->prepare($query);
        $statement->bindValue(':offset', (int) $this->getOffset(), \PDO::PARAM_INT);
        $statement->bindValue(':limit', (int) $this->limit, \PDO::PARAM_INT);
        return $statement;
    }
    public function getTotal(string $query) {
        $query_total = $this->pdo->query("SELECT COUNT(*) FROM ($query) q");
        return (int) $query_total->fetchColumn();
    }
    public function setSQLBeforeQuery(): ?string {
        return null;
    }
}
```

Template Method (exemplo - parte 2)

```
class MySQLPaginator extends Paginator {

    protected function getPaginateQuery(string $query, string $order = 'id'): string {
        return "$query ORDER BY $order LIMIT :limit OFFSET :offset";
    }
}

class AnsiPaginator extends Paginator {

    protected function getPaginateQuery(string $query, string $order = 'id'): string {
        return "$query ORDER BY $order OFFSET :offset ROWS FETCH NEXT :limit
ROWS ONLY";
    }
}

class SQLServerCustomPaginator extends AnsiPaginator {

    public function setSQLBeforeQuery(): ?string {
        return "SET DATEFORMAT dmy;";
    }
}
```

Template Method (exemplo - parte 3)

```
class HtmlUtil {

    private static function montaLinha(array $row, $tag = 'td') {
        return "<tr>" . implode("", array_map(function($row) use ($tag)) {
            return "<$tag>" . $row . "</{$tag}>";
        }, $row)) . "</tr>";
    }

    public static function getTable(\PDOStatement $statement, array $header = []) {
        $statement->execute();
        $table = "<table border>";
        $table .= (!empty($header)) ? self::montaLinha($header, 'th') : '';
        while ($row = $statement->fetch()) {
            $table .= self::montaLinha($row);
        }
        return "$table </table>";
    }

    public static function getPaginationLinks(int $pagina, int $limit, int $total): string {
        $links = (($pagina - 1) > 0) ? "<a href=?pagina=" . ($pagina - 1) . ">Anterior</a>" : "Anterior";
        $links .= "&ampnbsp";
        return $links .= (($pagina * $limit < $total) ? "<a href=?pagina=" . ($pagina + 1) . ">Próximo</a>" : "Próximo";
    }
}
```

Template Method (exemplo - parte 4)

```
$pagina = (isset($_REQUEST['pagina'])) ? $_REQUEST['pagina'] : 1;
$mySql = new \MySQLPaginator(Conexao::getInstance()->getPdo(),
$pagina);
$query = "SELECT * FROM livro";
$statement = $mySql->getPaginateStatement("SELECT livro.id,
livro.titulo, livro.preco, livro.isbn FROM livro");
echo HtmlUtil::getTable($statement, ['ID','Título','Preço', 'ISBN']);
echo HtmlUtil::getPaginationLinks($pagina, $mySql->getLimit(),
$mySql->getTotal($query));
```

Visitor

Visitor é pattern comportamental que provê uma maneira de separar um algoritmo de uma estrutura de objetos na qual ele opera. Um resultado prático dessa separação é a capacidade de adicionar novas operações às estruturas de objetos existentes sem modificar as estruturas. É uma maneira de seguir o princípio aberto/fechado (WIKIPEDIA, 2020).

Em essência, o visitante permite adicionar novas funções virtuais a uma família de classes, sem modificar as classes. Em vez disso, é criada uma classe de visitante que implementa todas as especializações apropriadas da função virtual. O visitante toma a referência da instância como entrada e implementa o objetivo por meio de double dispatch (WIKIPEDIA, 2020).



Visitor (problema do despacho)

Na ciência da computação, o despacho dinâmico é o processo de selecionar qual implementação de uma operação polimórfica (método ou função) chamar no em de execução. É comumente empregado em linguagens de programação orientada a objetos (OOP).

Polimorfismo por tipo de parâmetro (ex.: Java) => <https://ideone.com/PDthf0>
Utilizando Visitor (um tipo de despacho múltiplo): <https://ideone.com/JChjkM>

Na engenharia de software, o despacho duplo é uma forma especial de despacho múltiplo e um mecanismo que despacha uma chamada de função para diferentes funções concretas, dependendo dos tipos de tempo de execução de dois objetos envolvidos na chamada. Na maioria dos sistemas orientados a objetos, a função concreta chamada de uma chamada de função no código depende do tipo dinâmico de um único objeto e, portanto, são conhecidas como chamadas de despacho único ou simplesmente chamadas de função virtual.

Visitor (conceitual - parte 1)

```
<?php
interface ComponenteOuElemento {
    public function aceitar(Visitor $visitor): void;
}

class ComponenteOuElementoA implements ComponenteOuElemento {

    public function aceitar(Visitor $visitor): void {
        $visitor->visitarComponenteOuElementoA($this);
    }

    public function fazAlgoX(): string {
        return "A";
    }
}

class ComponenteOuElementoB implements ComponenteOuElemento {

    public function aceitar(Visitor $visitor): void {
        $visitor->visitarComponenteOuElementoB($this);
    }

    public function fazAlgoY(): string {
        return "B";
    }
}
```

Visitor (conceitual - parte 2)

```
interface Visitor {  
    public function visitarComponenteOuElementoA(ComponenteOuElementoA $element): void;  
    public function visitarComponenteOuElementoB(ComponenteOuElementoB $element): void;  
}  
  
class Visitor1 implements Visitor {  
  
    public function visitarComponenteOuElementoA(ComponenteOuElementoA $element): void {  
        echo $element->fazAlgoX() . ".self::class.PHP_EOL;  
    }  
    public function visitarComponenteOuElementoB(ComponenteOuElementoB $element): void {  
        echo $element->fazAlgoY() . ".self::class.PHP_EOL;  
    }  
}  
  
class Visitor2 implements Visitor {  
  
    public function visitarComponenteOuElementoA(ComponenteOuElementoA $element): void {  
        echo $element->fazAlgoX() . ".self::class.PHP_EOL;  
    }  
    public function visitarComponenteOuElementoB(ComponenteOuElementoB $element): void {  
        echo $element->fazAlgoY() . ".self::class.PHP_EOL;  
    }  
}
```

Visitor (conceitual - parte 3)

```
//Cliente
$components = [
    new ComponenteOuElementoA(),
    new ComponenteOuElementoB(),
];
$visitor = new Visitor1();
array_map(fn(ComponenteOuElemento $c) =>
    $c->aceitar($visitor) , $components);
echo PHP_EOL;
```

Visitor (exemplo - parte 1)

```
interface Elemento {  
    public function aceitar(Visitor $visitor);  
}  
  
interface Visitor {  
    public function visitarUniversidade(Universidade $universidade): array;  
    public function visitarAluno(Aluno $aluno): string;  
}
```

Visitor (exemplo - parte 2)

```
class Aluno implements Elemento {

    private string $nome;
    private array $faltasJustificadas = [];

    public function __construct(string $nome) {
        $this->nome = $nome;
    }

    public function addFaltaJustificada(\DateTimeInterface $inicio, \DateTimeInterface $fim): void {
        $this->faltasJustificadas[] = new FaltaJustificada($inicio, $fim);
    }

    public function getName(): string {
        return $this->nome;
    }

    public function getFaltaJustificadas(): array {
        return $this->faltasJustificadas;
    }

    public function aceitar(Visitor $visitor): string {
        return $visitor->visitarAluno($this);
    }
}
```

Visitor (exemplo - parte 3)

```
class FaltaJustificada {
    private \DateTimeInterface $inicio;
    private \DateTimeInterface $fim;
    public function __construct(\DateTimeInterface $inicio, \DateTimeInterface $fim) {
        $this->inicio = $inicio;
        $this->fim = $fim;
    }
    public function getInicio(): \DateTimeInterface {
        return $this->inicio;
    }
    public function getFim(): \DateTimeInterface {
        return $this->fim;
    }
}
class Universidade implements Elemento {
    private string $nome;
    private array $alunos;
    public function __construct(string $nome, array $alunos) {
        $this->nome = $nome;
        $this->alunos = $alunos;
    }
    public function getNome(): string {
        return $this->nome;
    }
    public function getAlunos(): array {
        return $this->alunos;
    }
    public function aceitar(Visitor $visitor): array {
        return $visitor->visitarUniversidade($this);
    }
}
```

Visitor (exemplo - parte 4)

```
class RelatorioFaltas implements Visitor {

    public function visitarAluno(Aluno $aluno): string {
        $diasPerdidos = 0;

        foreach ($aluno->getFaltaJustificadas() as $faltaJustificada) {
            $diasPerdidos += $faltaJustificada->getInicio()->diff($faltaJustificada->getFim())->days + 1;
        }

        return "Aluno: {$aluno->getNome()} perdeu {$diasPerdidos} dias";
    }

    public function visitarUniversidade(Universidade $universidade): array {
        $resultado = [];
        $resultado[] = "Gerando relatório para: \"{$universidade->getNome()}\"";

        foreach ($universidade->getAlunos() as $aluno) {
            $resultado[] = $this->visitarAluno($aluno);
        }

        return $resultado;
    }
}
```

Visitor (exemplo - parte 5)

```
$aluno1 = new Aluno("João");
$aluno1->addFaltaJustificada(new \DateTime("2019-10-01"), new \DateTime("2019-10-21"));
$aluno1->addFaltaJustificada(new \DateTime("2019-11-02"), new \DateTime("2019-11-10"));
$aluno2 = new Aluno("Maria");
$aluno2->addFaltaJustificada(new \DateTime("2019-11-01"), new \DateTime("2019-11-15"));
$aluno3 = new Aluno("Carlos Picanha");

$universidade = new Universidade("Universidade do Churrasco", [$aluno1, $aluno2, $aluno3]);
$resultado = $universidade->aceitar(new RelatorioFaltas());

foreach ($resultado as $dado) {
    echo $dado . PHP_EOL;
}
```

Fluent Interface

```
<?php
class Cafe {
    private $acucar = 0;
    private $leite   = 0;

    public function addAcucar(int $colheres){
        $this->acucar += $colheres;
        return $this;
    }

    public function addLeite(int $mililitros){
        $this->leite += $mililitros;
        return $this;
    }

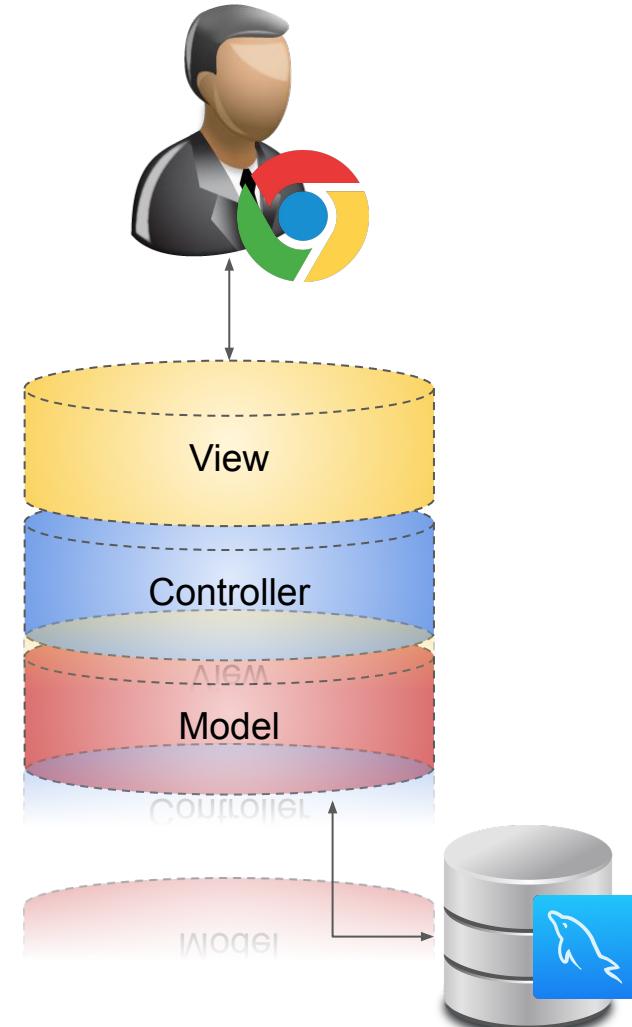
    public function preparar(){
        return "Café com {$this->acucar} colheres de
            açucar e {$this->leite} ml de leite";
    }
}
```

```
$cafe = new Cafe();
echo $cafe->addAcucar(2)
    ->addLeite(20)
    ->addAcucar(1)
    ->preparar();
```

CURL

REST

M.V.C (Model View Controller)



PSR4 - Autoloader

Apache .htaccess

Referências

<https://biratkirat.medium.com/step-23-domain-specific-language-jon-jagger-2ae8c391ce63>

https://docstore.mik.ua/orelly/webprog/pcook/ch13_05.htm

<https://springframework.guru/gang-of-four-design-patterns/bridge-pattern/>

<https://www.javatpoint.com/structural-design-patterns>

<http://pt.slideshare.net/patrick.allaert/php-data-structures-and-the-impact-of-php-7-on-them-php-days-2015>

<https://medium.com/@rafacdelnero/design-patterns-saga-18-real-world-situations-with-flyweight-6a0d2a346921rns/flyweight/php/example>

<https://docs.microsoft.com/pt-br/dotnet/standard/base-types/anchors-in-regular-ex>

Referências Bibliográficas

MARTIN, Robert Cecil. **Agile Software Development: Principles, Patterns, and Practices.** NJ, EUA: Prentice Hall, 2003. ISBN 0135974445.

GAMMA, Erich et al. **Design Patterns: Elements of Reusable Object-Oriented Software.** Westford: Addison-wesley, 1994.

MA, Burton. http://www.cse.yorku.ca/~burton/teaching/2009W/April_22.ppt&prev=search

<http://www.kathrynpieplow.pwrfaculty.org/wp-content/uploads/2010/01/Ralph-Wand-definition-design.pdf>

ISO/IEC/IEEE., **24765:2017 Systems and Software Engineering—Vocabulary.** Second Edition. 2017

DAVIS, Alan Mark. **201 Principles of software development.** 1995. ISBN 0070158401

NODET, Xavier. **What are invariants, how can they be used, and have you ever used it in your program?.** 2010. Disponível em: <https://softwareengineering.stackexchange.com/questions/32727/what-are-invariants-how-can-they-be-used-and-have-you-ever-used-it-in-your-pro>

Mili, Hafedh & El-Boussaidi, Ghizlane. (2005). **Representing and Applying Design Patterns: What Is the Problem?.** 3713. 186-200. 10.1007/11557432_14.