



Campus Manhuaçu

Introdução a programação orientada a objetos

Prof. Leonardo C. R. Soares - leonardo.soares@ifsudestemg.edu.br Instituto Federal do Sudeste de Minas Gerais 21 de novembro de 2024



Introdução

Introdução

O conceito de orientação a objetos não é novo. Desde a década de 1960 já existiam linguagens que davam suporte a esta metodologia. Nos últimos anos, esse paradigma de programação tem ganhado cada vez mais destaque no mercado sendo utilizado por linguagens como Java, JavaScript C++, PHP (desde a versão 5) etc.



Introdução

Introdução

O conceito de orientação a objetos não é novo. Desde a década de 1960 já existiam linguagens que davam suporte a esta metodologia. Nos últimos anos, esse paradigma de programação tem ganhado cada vez mais destaque no mercado sendo utilizado por linguagens como Java, JavaScript C++, PHP (desde a versão 5) etc.

Orientação a Objetos

A OO - Orientação a Objetos - é também um termo geral que inclui qualquer estilo de desenvolvimento que seja baseado no conceito de objeto - uma entidade que exibe características e comportamentos.



Classes e Objetos

Classe

Classe é uma representação abstrata de uma entidade existente no domínio do sistema. Funciona como um molde para a criação de objetos. Uma classe define todos os atributos (características) e métodos (comportamentos) que serão comuns a um grupo de objetos.

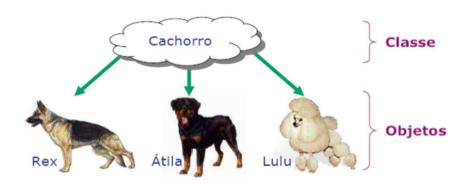


Classes e Objetos

Objeto

O objeto representa uma ocorrência específica de uma classe, ou seja, uma instância de classe.

Classes e Objetos



Atributos e métodos

Atributos

São as características que uma determinada classe possui. Os atributos são definidos pela classe, mas seus valores (em sua maioria) são definidos no objeto.

Métodos

São as ações que uma determinada classe pode executar.



Abstração

Definição

Abstração é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.

Encapsulamento

<u>Definição</u>

Encapsulamento é uma característica que nos permite ocultar detalhes da implementação que não são necessários fora de um determinado contexto



Public, Protected e Private

► Public - O atributo ou método estará visível a todas as classes:



Public, Protected e Private

- ► Public O atributo ou método estará visível a todas as classes;
- ▶ Protected O atributo ou método estará visível na classe onde foi declarado e nas subclasses dela;

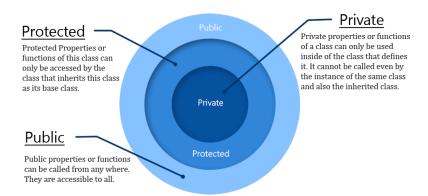




Public, Protected e Private

- ► Public O atributo ou método estará visível a todas as classes;
- ▶ Protected O atributo ou método estará visível na classe onde foi declarado e nas subclasses dela;
- ► Private O atributo ou método estará visível apenas na classe onde foi declarado.





Copyright Dreamsoft 2013 www.dreamsoftworks.blogspot.com



Herança

Definição

Herança é um recurso que permite a definição de uma hierarquia de classes, com o objetivo de reutilizar características já definidas e estendê-las em níveis maiores de detalhes ou especialização.



Polimorfismo

Definição

Polimorfismo é um recurso que permite que diversas classes descendentes implementem versões diferentes de um conjunto de ações comuns definidas na classe base



Declaração de classes

Sintaxe

Para declararmos classes em PHP utilizamos a palavra reservada class seguida do nome da classe. O corpo da classe é delimitado por chaves ({}). Por convenção, escreveremos o nome da classe com a primeira letra maiúscula.

Declaração de classes

Sintaxe

Para declararmos classes em PHP utilizamos a palavra reservada class seguida do nome da classe. O corpo da classe é delimitado por chaves ({}). Por convenção, escreveremos o nome da classe com a primeira letra maiúscula.

```
Animal{
class
```



Declaração de atributos

Sintaxe

Para declararmos atributos de classe, devemos utilizar o modificador de acesso seguido do nome do atributo. Caso o modificador de acesso seja omitido, será atribuído o modificador public.

Declaração de atributos

Sintaxe

Para declararmos atributos de classe, devemos utilizar o modificador de acesso seguido do nome do atributo. Caso o modificador de acesso seja omitido, será atribuído o modificador public.

```
Animal{
class
    public string $nome;
    public int $idade;
```

Sintaxe

Para declararmos métodos de classe, devemos utilizar o modificador de acesso seguido da palavra-chave function, do nome do método e, entre parênteses, os parâmetros que o método recebe. Caso o método não receba parâmetros, deve-se colocar parênteses vazios após o nome do método. Após os parâmetros, é possível definir o tipo de retorno do método, precedido por dois pontos.

```
Animal{
class
    public string $nome;
    public int $idade;
    public function emitirSom(): void{
        echo "...";
    public function anoNascimento(int $anoAtual): int{
        return $anoAtual - $this->idade;
```

```
class
      Animal{
    public string $nome;
    public int $idade;
    public function emitirSom(): void{
        echo "...";
    public function anoNascimento(int $anoAtual): int{
        return $anoAtual - $this->idade;
```

O operador \$this é utilizado quando a classe precisa fazer uma referência ao objeto dela própria.

```
class
      Animal{
    public string $nome;
    public int $idade;
    public function emitirSom(): void{
        echo "...";
    public function anoNascimento(int $anoAtual): int{
        return $anoAtual - $this->idade;
```

O operador \$this é utilizado quando a classe precisa fazer uma referência ao objeto dela própria.

O operador -> separa o objeto de seu atributo ou método.



Criação de objetos

Definição

Para criar um novo objeto, utilizamos a instrução new seguido do nome da classe.

Criação de objetos

Definição

Para criar um novo objeto, utilizamos a instrução new seguido do nome da classe.

```
$cao = new Animal();
$cao->nome = "Bilu";
cao - idade = 12;
$gato = new Animal();
echo "O cão " . $cao->nome . "nasceu em "
                          .$cao->anoNascimento(2022);
```







Exercício Rápido

Desenvolva uma classe que possua dois atributos numéricos (num1 e num2) e métodos que permitam efetuar as quatro operações fundamentais sobre estes números. Instâncie um objeto, dê valores aleatórios a num1 e num2 e imprima o resultado das quatro operações.



```
class Calculos {
    public float $num1;
    public float $num2;
    public function somar(): float {
        return $this->num1 + $this->num2: }
    public function subtrair(): float{
        return $this->num1 - $this->num2; }
    public function dividir(): float{
        return $this->num1 / $this->num2; }
    public function multiplicar(): float{
        return $this->num1 * $this->num2; }
```

```
require_once "Calculos.php";
$x = new Calculos();
x->num1 = rand(0,10);
x->num2 = rand(0,10);
echo "Num1: $x->num1 - Num2: $x->num2<br>";
echo $x->somar() . "<br>";
echo $x->subtrair() . "<br>":
echo $x->multiplicar() . "<br>";
echo $x->dividir() . "<br>":
```

```
require_once "Calculos.php";
$x = new Calculos();
x->num1 = rand(0,10);
x->num2 = rand(0,10);
echo "Num1: $x->num1 - Num2: $x->num2<br>";
echo $x->somar() . "<br>";
echo $x->subtrair() . "<br>":
echo $x->multiplicar() . "<br>";
echo $x->dividir() . "<br>":
```

O que acontece se o num2 for zero?

```
require_once "Calculos.php";
$x = new Calculos();
x->num1 = rand(0,10);
x->num2 = rand(0,10);
echo "Num1: $x->num1 - Num2: $x->num2<br>";
echo $x->somar() . "<br>";
echo $x->subtrair() . "<br>":
echo $x->multiplicar() . "<br>";
echo $x->dividir() . "<br>":
```

O que acontece se o num2 for zero? Para corrigir devemos lançar uma exceção e tratá-la.

```
class Calculos {
    public float $num1;
    public float $num2;
    public function somar(): float {
        return $this->num1 + $this->num2: }
    public function subtrair(): float{
        return $this->num1 - $this->num2; }
    public function dividir(): float{
        if (\frac{shis}{num} = 0)
          throw new Exception("Impossível dividir por 0.");
        return $this->num1 / $this->num2; }
    public function multiplicar(): float{
        return $this->num1 * $this->num2: }
```

```
$x = new Calculos():
x->num1 = rand(0.10):
x->num2 = rand(0.10):
echo "Num1: $x->num1 - Num2: $x->num2<br>";
echo $x->somar() . "<br>":
echo $x->subtrair() . "<br>":
echo $x->multiplicar() . "<br>";
trv{
    echo $x->dividir() . "<br>";
}catch(Exception $e){
    echo $e->getMessage();
}
```

Encapsulamento: Como fazer

O básico do encapsulamento é feito restringindo-se o acesso direto aos atributos da classe. Isto deve ser feito para garantir que as regras de negócio da classe, desconhecidas por usuários do objeto, não sejam violadas.



Encapsulamento: Como fazer

O básico do encapsulamento é feito restringindo-se o acesso direto aos atributos da classe. Isto deve ser feito para garantir que as regras de negócio da classe, desconhecidas por usuários do objeto, não sejam violadas.

Para permitir que usuários do objeto possam, quando necessário, definir ou ler os valores dos atributos, devem ser criados métodos especiais denominados método acessores.

Encapsulamento: Métodos acessores

Métodos acessores responsáveis por retornar o valor de atributos devem ser nomeados como getNomeAtributo. Em geral estes métodos não recebem parâmetros e não fazem validações.

Encapsulamento: Métodos acessores

Métodos acessores responsáveis por retornar o valor de atributos devem ser nomeados como getNomeAtributo. Em geral estes métodos não recebem parâmetros e não fazem validações.

Métodos acessores responsáveis por atribuir valores a atributos devem ser nomeados como setNomeAtributo. Em geral estes métodos recebem como parâmetro o valor a ser atribuído e, caso existam regras de negócio sobre os valores possíveis, elas são aplicadas antes da atribuição dos mesmos.

Encapsulamento: Exemplo

Considere uma classe para representar um funcionário. Esta classe tem os atributos nome e salário. Salário não pode ser negativo.

Encapsulamento: Exemplo

```
<?php
class Funcionario{
    private string $nome;
    private float $salario:
    public function getNome():string{
        return $this->nome:
    public function setNome($nome):void{
        $this->nome = $nome:
    public function getSalario():float{
        return $this->salario;
    public function setSalario($salario):void{
        if ($salario<0)
            throw new Exception("Salário não pode ser negativo.");
        $this->salario=$salario;
```



Definição

Membros de classe estáticos são atributos ou métodos que pertencem a classe e não ao objeto. Não importa a quantidade de instâncias que uma classe possua, há apenas uma cópia de cada membro estático.



Declaração e uso

Para declarar um membro como estático, utilizamos a palavra-chave static.

Declaração e uso

Para declarar um membro como estático, utilizamos a palavra-chave static.

Quando a própria classe deseja acessar um membro estático, deve-se utilizar a palavra-chave self seguido de ::

Declaração e uso

Para declarar um membro como estático, utilizamos a palavra-chave static.

Quando a própria classe deseja acessar um membro estático, deve-se utilizar a palavra-chave self seguido de ::

Quando alguém fora da classe deseja acessar um membro estático, deve utilizar o nome da classe seguido de ::



Declaração e uso

Para declarar um membro como estático, utilizamos a palavra-chave static.

Quando a própria classe deseja acessar um membro estático, deve-se utilizar a palavra-chave self seguido de ::

Quando alguém fora da classe deseja acessar um membro estático, deve utilizar o nome da classe seguido de ::

Um método estático não pode acessar uma propriedade não estática (dinâmica).

Exemplo

```
class Circulo {
    public static float $PI = 3.1415;
    public static function calculaArea(float $raio): float {
        return self::$PI*($raio*$raio);
echo "O valor de PI é ". Circulo::$PI:
echo "A área de um círculo com 10cm de raio é ".Circulo::calculaArea(10);
```

Métodos construtores

Definição

Métodos construtores definem quais ações devem ser executadas quando da criação de um objeto. Implicitamente, um método construtor retorna um objeto do próprio tipo da classe.



Definiç<u>ão</u>

Métodos construtores definem quais ações devem ser executadas quando da criação de um objeto. Implicitamente, um método construtor retorna um objeto do próprio tipo da classe.

Em PHP, o método construtor de uma classe é nomeado como construct. Assim como qualquer outro método, o construtor pode receber parâmetros. Caso isso aconteça, os parâmetros devem ser informados na criação do ojbeto.

class Funcionario{

```
private string $nome:
   private float $salario;
    public function construct(string $nome, float $sal){
        $this->setNome($nome):
        $this->setSalario($sal):
    public function getNome():string{
        return $this->nome;
    public function setNome($nome):void{
       $this->nome = $nome;
    public function getSalario():float{
        return $this->salario:
    public function setSalario($salario):void{
       if ($salario<0)
           throw new Exception("Salário não pode ser negativo.");
        $this->salario=$salario;
$p = new Funcionario("Maria",1780);
```



Métodos destrutores

Definição

Métodos destrutores são invocados quando todas as referências a um objeto particular forem removidas ou quando o objeto for explicitamente destruído.

Métodos destrutores

Definiç<u>ão</u>

Métodos destrutores são invocados quando todas as referências a um objeto particular forem removidas ou quando o objeto for explicitamente destruído.

Em PHP, o método destrutor de uma classe é nomeado como destruct. Assim como qualquer outro método, o destrutor pode receber parâmetros. Caso isso aconteça, os parâmetros devem ser informados na criação do ojbeto.



Exemplo

```
<?php
class LOG {
    public $arquivo;
    public function __construct(){
       $this->arquivo = fopen("log.txt","a");
       fwrite($this->arquivo,"Usuário logou em " . date("d/m/Y H:i:s") . "\n");
    public function __destruct(){
       fwrite($this->arquivo,"Usuário saiu em " . date("d/m/Y H:i:s") . "\n");
       fclose($this->arquivo);
$log = new Log();
?>
```

00000000000000000000



Sobrecarga

Definição

Sobrecarga (overloading) ocorre quando criamos dois ou mais métodos como o mesmo nome mas uma lista de argumentos (e possivelmente comportamentos) diferentes.



Sobrecarga

Atenção:

PHP define como sobrecarga a criação dinâmica de atributos e métodos. Não é o nosso enfoque. Estamos tratando sobrecarga sobre a ótica da programação orientada à objetos.



Herança

Definição

Herança é um recurso da POO que nos permite criar novas classes a partir de classes já existentes.

Herança

Definição

Herança é um recurso da POO que nos permite criar novas classes a partir de classes já existentes.

A classe de quem herdaremos denomina-se classe base ou super classe. Em geral, ela representa um conceito mais genérico.



Heranca

Definição

Herança é um recurso da POO que nos permite criar novas classes a partir de classes já existentes.

A classe de quem herdaremos denomina-se classe base ou super classe. Em geral, ela representa um conceito mais genérico.

A classe que estamos criando a partir da super classe, denomina-se subclasse ou classe filha. Em geral, ela representa um conceito mais específico.



Herança

Todos os atributos e métodos não privados da super classe são herdados pela subclasse. Em PHP utilizamos a palavra-chave extends para indicar um relacionamento de herança (é-um).

```
<?php
class PessoaFisica {
    protected string $nome;
    protected string $cpf;
    public function getNome(): string { return $this->nome;}
    public function setNome(string $n){ $this->nome = $n;}
    public function getCpf(): string { return $this->cpf;}
    public function setCpf(string $cpf){ $this->cpf = $cpf;}
```

```
require once "PessoaFisica.class.php";
class Funcionario extends PessoaFisica {
    private string $cargo;
    private float $salario;
    public function setCargo($c){ $this->cargo = $c;}
    public function getCargo(): string { return $this->cargo;}
    public function setSalario($s){ $this->salario = $s;}
    public function getSalario():float{ return $this->salario*0.89;}
```

```
$f = new Funcionario();
$f->setNome("Mariana");
$f->setCpf("123456");
$f->setCargo("Professora");
$f->setSalario(5000);
echo "Olá " . $f->qetNome() . " seu salário líquido é R$ ".$f->qetSalario();
?>
```



Herança

Quando a superclasse possui um método construtor, torna-se obrigação da subclasse executá-lo corretamente em seu próprio construtor.

```
class PessoaFisica {
    protected string $nome;
    protected string $cpf;
    public function __construct(string $cpf, string $nome){
        $this->setNome($nome):
        $this->setCpf($cpf);
    public function getNome(): string { return $this->nome;}
    public function setNome(string $n){ $this->nome = $n;}
    public function getCpf(): string { return $this->cpf;}
    public function setCpf(string $cpf){ $this->cpf = $cpf;}
```

```
require_once "PessoaFisica.class.php";
class Funcionario extends PessoaFisica {
   private string $cargo;
   private float $salario:
   parent::__construct($cpf,$nome);
      $this->setCargo($cargo);
      $this->setSalario($salario);
   public function setCargo($c){ $this->cargo = $c;}
   public function getCargo(): string { return $this->cargo:}
   public function setSalario($s){ $this->salario = $s;}
   public function getSalario():float{ return $this->salario*0.89;}
```

```
$f = new Funcionario("1234", "Mariana", "Professora", 5000);
echo "Olá ". $f->getCargo() ." ". $f->getNome() . " seu salário líquido é R$ ".$f->getSalario();
```



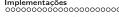
Sobrescrição

Definição

Sobrescrição (overriding) é um recurso da POO que permite às subclasses alterarem o comportamento herdado da superclasse por algo mais específico.







Exemplo

Continuando o exemplo anterior, suponha que nossa empresa tenha um tipo especial de funcionário denominado Gerente. Embora o gerente esteja sujeito aos mesmos descontos que todo funcionário, ele recebe 15% sobre o salário líquido como abono.

Exemplo

```
require_once "Funcionario.class.php";
class Gerente extends Funcionario {
    public function __construct(string $cpf, string $nome, float $salario){
        parent::__construct($cpf,$nome,"Gerente",$salario);
    public function getSalario(): float{
        $salarioLiquido = parent::getSalario();
        return $salarioLiquido*1.15;
```

Manhuacu

```
$f = new Funcionario("1234", "Mariana", "Professora", 5000);
$g = new Gerente("2333","Júlia",5000);
echo "Olá ". $f->getCargo() ." ". $f->getNome() . " seu salário líquido é R$ ".$f->getSalario();
echo "<br>";
echo "Olá ". $g->getCargo() ." ". $g->getNome() . " seu salário líquido é R$ ".$g->getSalario();
?>
```



Classe abstrata

Definição

Uma classe abstrata é uma classe que não pode ser instanciada. Ela sempre será utilizada como super classe, representando em geral, conceitos mais genéricos que devem ser especializados por suas subclasses.



Classe abstrata

Definição

Uma classe abstrata é uma classe que não pode ser instanciada. Ela sempre será utilizada como super classe, representando em geral, conceitos mais genéricos que devem ser especializados por suas subclasses.

Classes abstratas podem conter métodos abstratos, ou seja, métodos que não possuem corpo, representando comportamentos que devem ser implementados nas subclasses concretas.

Exemplo

```
abstract class Animal{
    public string $nomeCientifico;
    public abstract function emitirSom(): string;
    public abstract function dormir(): string;
    public function setNomeCientifico($n):void{
        $this->nomeCientifico = $n;
    public function getNomeCientifico():string {
        return $this->nomeCientifico;
```

abstract class Animal{

Exemplo

```
public string $nomeCientifico;
public abstract function emitirSom(): string;
public abstract function dormir(): string;
public function setNomeCientifico($n):void{
    $this->nomeCientifico = $n;
public function getNomeCientifico():string {
    return $this->nomeCientifico;
```

Se tentarmos criar um objeto desta classe (new Animal), receberemos o erro: Fatal error: Uncaught Error: Cannot instantiate abstract class Animal in ...



Subclasses concretas

Classes concretas que herdam de classes abstratas devem, obrigatoriamente, implementar os métodos abstratos que foram herdados. No exemplo apresentado, toda classe concreta que herdar de Animal (é-um) deve implementar os métodos emitirSom() e dormir().

Exemplo

```
class Gato extends Animal {
    public function emitirSom(): string {
        return "miau";
    public function dormir(): string {
        return "zzzZZzZzZz";
```



Operador Final

Definição

O operador final quando aplicado a um método impede que este método seja sobrescrito nas subclasses.

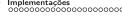
Quando aplicado a uma classe, o operador final impede que a classe seja herdada.

Exemplo

```
class Base {
    public final function sayHello(): void{
        echo "hello";
class Filha extends Base {
    public function sayHello(): void{
        echo "Olá";
```

Fatal error: Cannot override final method Base::sayHello() in...







```
final class Base {
    // 0 final aqui é indiferente
    public final function sayHello(): void{
        echo "hello";
    }
}

class Filha extends Base {
    public function sayHello(): void{
        echo "Olá";
    }
}
```

Fatal error: Class Filha may not inherit from final class (Base) in...



Interfaces

Definição

Podemos entender a interface de uma classe como sendo o conjunto de comportamentos que esta classe é capaz de executar.



Definição

Podemos entender a *interface de uma classe* como sendo o conjunto de comportamentos que esta classe é capaz de executar.

Assim, uma Interface especifica o comportamento esperado para um conjunto de classes que a implementam.

Interfaces

Definição

Podemos entender a *interface de uma classe* como sendo o conjunto de comportamentos que esta classe é capaz de executar.

Assim, uma Interface especifica o comportamento esperado para um conjunto de classes que a implementam.

Diversos autores definem interface como um contrato a ser seguido por um conjunto de classes (as classes que implementam a interface).



Interfaces

Definição

Podemos entender a *interface de uma classe* como sendo o conjunto de comportamentos que esta classe é capaz de executar.

Assim, uma Interface especifica o comportamento esperado para um conjunto de classes que a implementam.

Diversos autores definem interface como um contrato a ser seguido por um conjunto de classes (as classes que implementam a interface).

Interfaces definem o que um conjunto de classes (que implementam a interface) devem fazer e não como deve ser feito.



Definição

A criação de interfaces em PHP seguem o mesmo padrão estabelecido para a criação de classes, apenas trocamos a palavra-chave class por interface.

Todos os métodos declarados em uma interface são públicos e implicitamente abstratos. A palavra-chave abstract não deve ser colocada nos métodos (para interfaces).

Manhuacu

```
interface A{
      public function foo(): void;
2
3
```

Herança entre Interfaces

Uma interface poder herdar de outra(s) interface(s).

```
interface A{
       public function somar(float $a, float $b):
2
          float:
   interface B{
       public function subtrair(float $a, float $b)
5
          : float;
6
   interface AcoesBasicas extends A, B{
       public function dividir(float $a, float $b):
8
           float:
       public function multiplicar(int $a,int $b):
9
          float;
10
```

Relacionamento entre classes e interfaces

Classes implementam interfaces

Ao implementar uma interface, uma classe concreta é obrigada a implementar todos os métodos declarados na interface. O relacionamento é similar ao que ocorre quando uma classe concreta herda de uma classe abstrata. Entretanto, como não se trata de um relacionamento de herança, é possível que uma uma mesma classe implemente diversas interfaces.

Exemplo

```
class Calculadora implements A,B,AcoesBasicas{
       public function somar(float $a, float $b):
2
           float{
           return $a+$b;
3
       }
4
       public function subtrair(float $a, float $b)
5
           : float{
           return $a-$b;
6
       }
7
       public function dividir(float $a, float $b):
8
            float{
           return $a/$b;
9
10
       public function multiplicar(int $a, int $b):
11
            float{
           return $a*$b;
12
       }
13
14
```



Polimorfismo

Definição

Polimorfismo é a propriedade de duas ou mais classes, derivadas de uma mesma superclasse, responderem a uma mesma mensagem de formas diferentes. Ocorre quando uma subclasse redefine um método existente na superclasse, ou seja, quando temos os métodos sobrescritos (overriding).

O polimorfismo também pode ser implementado utilizando-se interfaces.

Polimorfismo

interface Animal{

```
public function emitirSom(): string;
class Cachorro implements Animal {
    public function emitirSom(): string {
        return "au au au": }
class Gato implements Animal {
    public function emitirSom(): string {
        return "miau"; }
class Lobo implements Animal {
    public function emitirSom(): string {
        return "auuuu"; }
class Ornitorrinco implements Animal {
    public function emitirSom(): string {
        return "arghhh"; }
```



Polimorfismo

```
class Zoo {
    public function movimentarAnimais(Animal $al): void{
        echo $al->emitirSom() . "<br>":
z = \text{new Zoo()}:
$animais = []:
for($i=0;$i<10;$i++){
    x = rand(0.4):
    if ($x == 0) $animais[] = new Cachorro();
    if ($x == 1) $animais[] = new Gato():
    if ($x == 2) $animais[] = new Lobo();
    if ($x == 3) $animais[] = new Ornitorrinco();
foreach($animais as $animal)
    $z->movimentarAnimais($animal);
```

► Escreva uma classe (Data) para armazenar uma data. Declare

atributos inteiros referentes ao dia, mês e ano. Encapsule a classe. Escreva um método chamado dataExtenso(), este método deverá retornar uma string contendo "dd de mmmmmm de yyyy", onde dd é o valor do dia, mmmmmm o mês por extenso e yyyy o ano. Escreva um método chamado getFullData(), este método deverá retornar uma string contendo "dd/mm/yyyy", onde dd é o valor do dia, mm o valor do mês e yyyy o valor do ano.

Exercícios

- Escreva uma subclasse de Data chamada DataUS. Sobrescreva o método dataExtenso() traduzindo o nome dos meses. Sobrescreva o método getFullData() alterando o formato de retorno para "mm-dd-yyyy".
- ► Escreva uma subclasse de Data chama DataMySQL. Sobrescreva o método getFullData() alterando o formato de retorno para "yyyy-mm-dd".
- ► Escreva uma aplicação web que permita testar as funcionalidades de todas as classes criadas.



Exercícios

- Escreva uma classe abstrata chamada Funcionario. Esta classe deverá possuir os atributos, nome, cpf e salarioBase. Encapsule a classe. Os métodos acessores não podem ser sobrescritos. Escreva um método construtor que receba os valores iniciais de todos os atributos. Escreva um método abstrato chamado salarioLiquido(): float.
- Escreva uma subclasse de Funcionario chamada Professor. A classe deverá possuir um atributo inteiro horasAulasMes. Escreva um construtor que receba valores iniciais para todos os atributos. Encapsule a classe. O método salarioLiquido deverá retornar $horasAulasMes \times salarioBase$.

Exercícios

- Escreva uma subclasse de Funcionario chamada ColaboradorInternacional. Esta classe deverá possuir um atributo real chamado cotacaoDolar. Escreva um construtor que receba o valor inicial de todos os atributos. Encapsule a classe. O método salarioLiquido deverá retornar $cotacaoDolar \times salarioBase$.
- Escreva uma subclasse de Funcionario chamada Estagiario. Esta classe deverá possuir os atributos curso e duracaoContrato. Escreva um construtor que receba o valor inicial de todos os atributos. Encapsule a classe. O método salarioLiquido deverá retornar o salarioBase menos 45%.
- Crie uma aplicação web para testar as funcionalidades implementadas.



Manhuacu

