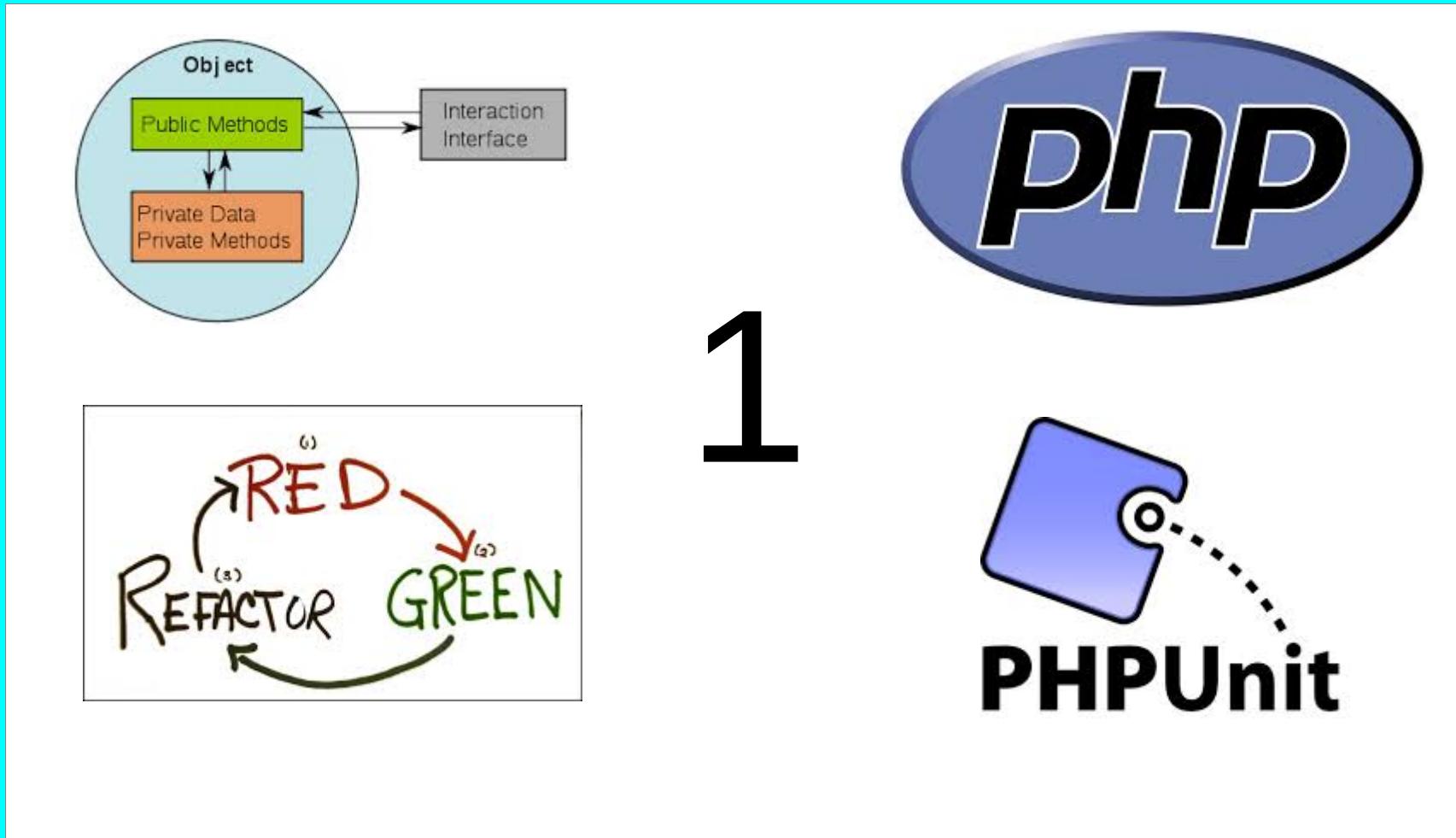


# Programação PHP orientada a objetos com testes unitários

FLÁVIO GOMES DA SILVA LISBOA



# VIDADEPROGRAMADOR

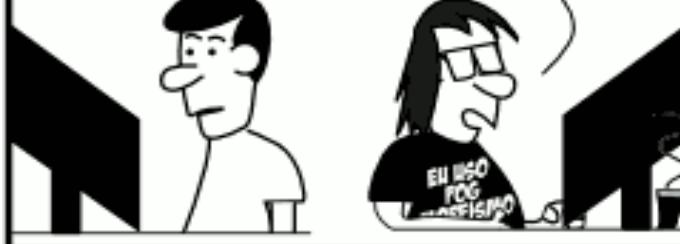
.COM.BR

/\* HISTÓRIA REAL  
ENVIADA POR  
FAGNER ALMEIDA \*/



#511

VOU COMEÇAR A USAR  
ORIENTAÇÃO A OBJETOS  
NOS PROGRAMAS EM PHP..



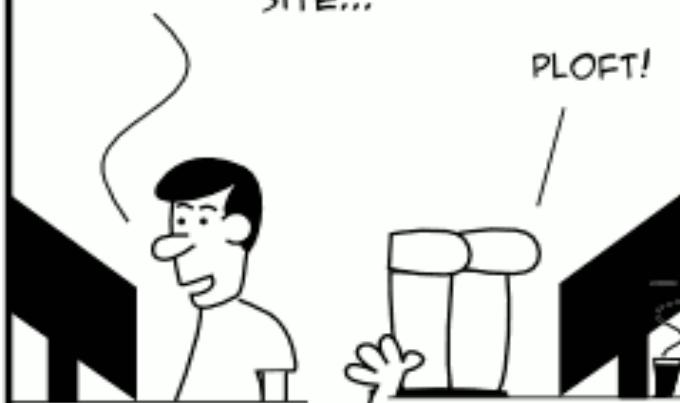
VOCE USA  
DREAMWEAVER?

NÃO, POR QUÊ?

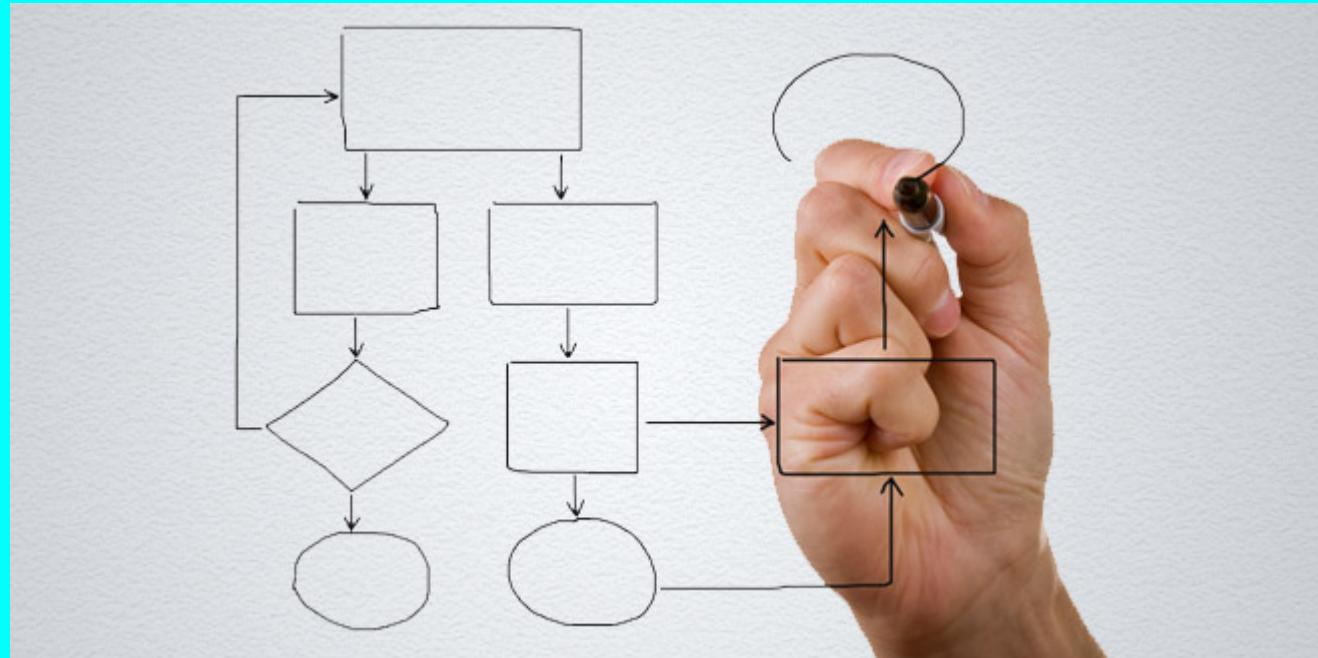


PORQUE LÁ É SIMPLES:  
VOCE SÓ PEGA OS OBJETOS  
E ARRASTA PARA O SEU  
SITE...

PLOFT!



# Planejamento



# Programação PHP orientada a objetos

## Parte 1

- Classe e objeto
- Carregamento de classes
- Atributos
- Métodos
- Visibilidade de atributos e métodos
- Constantes
- Construtor e destrutor
- Atributos e métodos de classe
- Herança
- Esterilidade
- Interfaces
- Para que classes?
- Início do trabalho de avaliação da disciplina

# Programação PHP orientada a objetos

## Parte 2

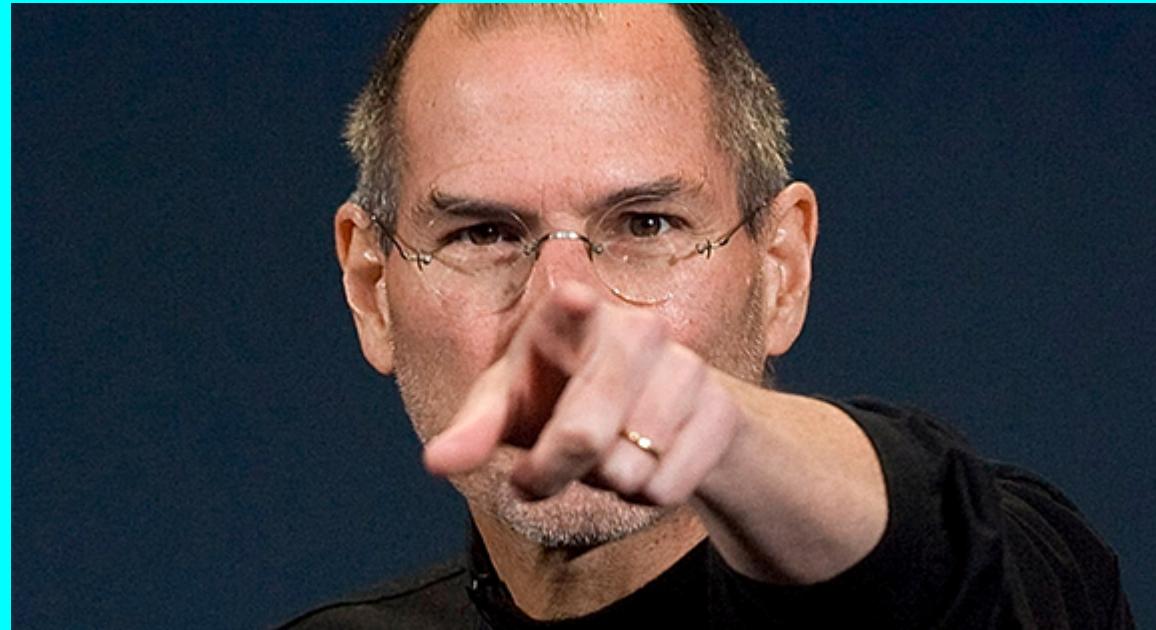
- Clonagem de objetos
- Comparação de objetos
- Indução de tipo
- Objetos e Referências
- *Late Static Bindings*
- *Serialização de objetos*
- Métodos mágicos
- Padrões PHP
- Continuação do trabalho de avaliação da disciplina

# Programação PHP orientada a objetos

## Parte 3

- *Fronteiras dos testes*
- *Traits*
- *PHP 7*
- *Spl Datastructures*
- *Spl Iterators*
- *Spl Interfaces*
- *Spl Exceptions*
- *Spl File Handling*
- *Spl Miscellaneous*
- *Spl Functions*
- *xUnit*
- *Padrões de teste*
- *Testes cheirando mal*
- *Esclarecimento de dúvidas sobre o trabalho de avaliação da disciplina*

# E os testes unitários?



# Paralelamente...

- Escreveremos testes...
- No início testes feios...
- Depois testes bonitos...



# Mas o que é um teste unitário?

*“Teste unitário é a execução de uma classe completa, rotina ou pequeno programa, escrito por um único programador ou por uma equipe de programadores. Esse teste é executado à parte do sistema mais completo”*



McConnell (2005, p. 529)

# Para que escrevemos testes?



# Para que escrevemos testes?



# O teste unitário testa

- Uma classe **completa**.
- Uma **rotina**.
- Um **pequeno** programa.



# O que o teste unitário *realmente* testa

Tipo de teste	O que é testado
De Unidade	Código
De Integração	Projeto
De Validação	Requisitos
De Sistema	“Se todos os elementos combinam adequadamente e se a função/desempenho global do sistema é alcançada”.

Fonte: Pressman (2006, p.291-292)

# Cobertura de testes



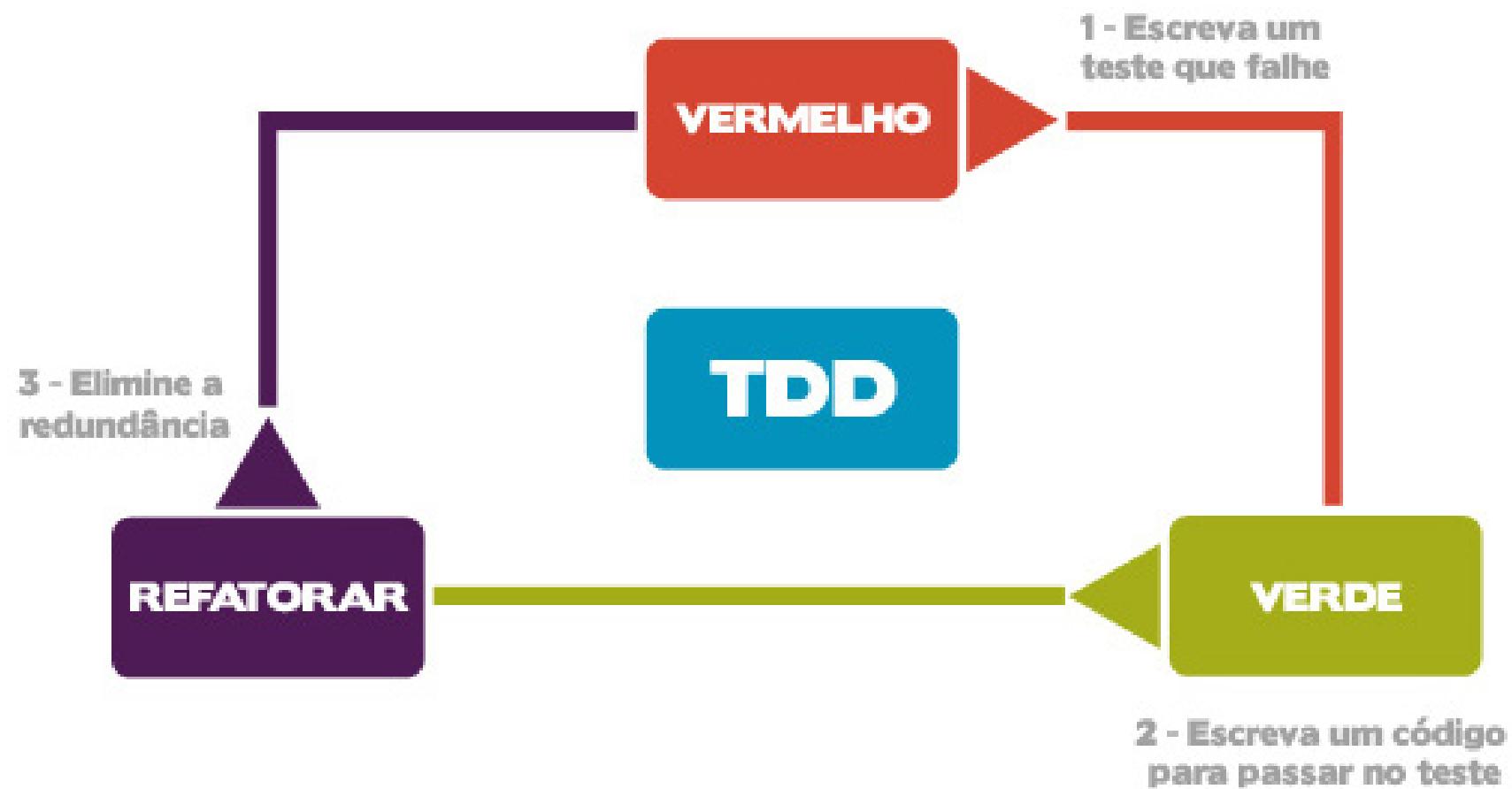


# Teoria e Prática

- O **teste unitário** diz o que você espera que o computador **faça**.
- O **código testado** diz o que o computador **deve fazer**.
- Se o computador **não faz** o que você espera, o código testado **não passará** no teste.
- Neste caso, **reescreva** o código testado, até que ele **passe** no teste.



# TDD (*Test Driven Development*)



# Em PHP o teste unitário testa

- Uma **classe completa**.
- Uma **rotina**.
- Um **pequeno** programa.



# Em PHP o teste unitário testa

- Uma **classe**.
- Uma **função**.
- Um **script**.



# Nesta disciplina iremos testar

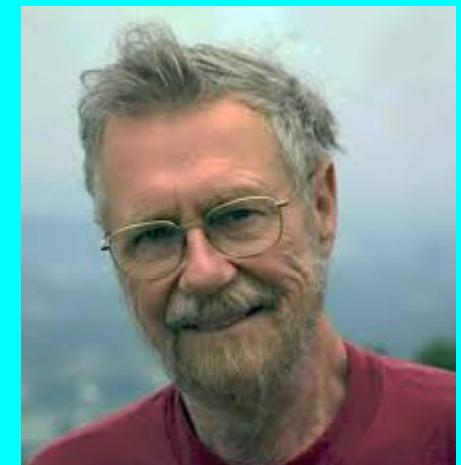
- Uma classe.
- ~~Uma função.~~
- ~~Um script.~~
- ~~Nenhuma das anteriores~~



# Complexidade

*“O programador competente por certo está ciente do tamanho **rigorosamente limitado** de seu próprio cérebro; portanto ele encara a tarefa de programação com total **humildade**”*

Dijkstra (1972 APUD McConnell, 2005, p. 491)





# Não testamos uma classe, na verdade...

- Testamos os **métodos** da classe.
- Para testar uma classe inteira, precisamos de **vários** testes unitários.

*Divide as dificuldades que tenhas de examinar em tantas partes quantas for possível, para uma melhor solução.*



René Descartes (1596-1650)

# Para testar uma classe...

Precisamos de uma classe!



# Classe e objeto

“Um **objeto** representa um 'elemento' que pode ser identificado de maneira **única**. Em um nível **apropriado** de abstração, praticamente **tudo** pode ser considerado como objeto”.

Coleman (1996, p. 16)



# Classe e objeto

“Objetos são agrupados em **conjuntos**, denominados classes. Uma **classe** é uma **abstração** que representa a ideia ou noção **geral** de um conjunto de objetos **similares**”.

Coleman (1996, p. 17)



# Fato



Programação de computadores não tem **vocabulário** próprio, embora a **computação** seja algo antigo.

*“A origem da computação (ato de calcular) é indicada pela própria origem da palavra 'calcular', que deriva da latina calculus e está relacionada com a grega chalix, ambas significando pedrinha ou seixo”*

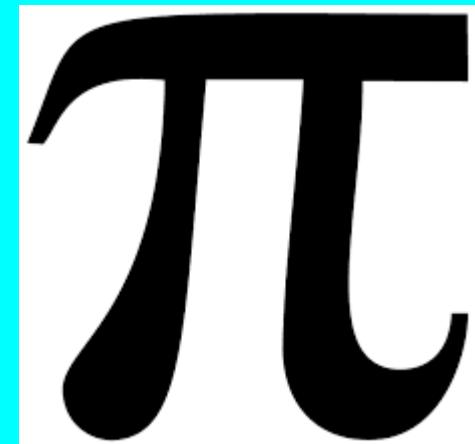
Davis (1992, p. 1)

# Exemplos de apropriação de vocabulário

- Vírus
- Cavalos de tróia
- *Worms*
- *Bugs*
- *Crashes*
- *Firewall*
- *Port*



- Variável
- Constante
- Função
- Classe
- Objeto
- Atributo
- Método



# Todo campo de conhecimento novo se apropria dos anteriores

## Marinha

- Porto
- Nave
- Embarcar



## Aeronáutica

- Aeroporto
- Aeronave
- Embarcar



# Metáforas

“Importantes desenvolvimentos frequentemente surgem a partir de **analogias**. Comparando um assunto do qual você **pouco entende** com outro assunto **semelhante**, do qual tem **maior domínio**, poderá ter ideias que resultem em uma **melhor compreensão** dessa matéria que lhe é **menos familiar**. O uso da metáfora é denominado '**modelagem**’”

McConnel (2005, p. 48)

# O valor das metáforas

USANDO A LINGUAGEM DELE...

WWW.CIBELESANTOS.COM.BR

ACORDA, ARMANDO!!! VOCÊ NÃO  
SABE QUE O AMOR É COMO UMA FLOR?  
SE NÃO FOR REGADA E BEM CUIDADA  
ELA MURCHA E MORRE!!!

HÃ?!

O AMOR É COMO O MOTOR DO CARRO  
ARMANDO! SE NÃO TROCAR O ÓLEO E  
NÃO CUIDAR DA MECÂNICA UMA HORA  
ELE NÃO FUNCIONA MAIS!!!

AH,  
ENTENDI!



POR QUE EU  
PASSARIA O  
DEDO NO  
PESCOÇO  
DELE?



# Metáforas ajudam a criar um novo vocabulário

## VOCABULÁRIO VINGADORESCO

DO MAL

### PESSOA NORMAL



TONY  
STARK



CAFÉ



ARMA



ROSQUINHA



GATO



LOKI



COISA PATÉTICA  
DOS MORTAIS



COISA PATÉTICA  
DOS MORTAIS



COISA PATÉTICA  
POSSÍVEL  
DOS MORTAIS  
ALIADO



AGENTE  
COULSON



ARMA



ARMA



ARMA



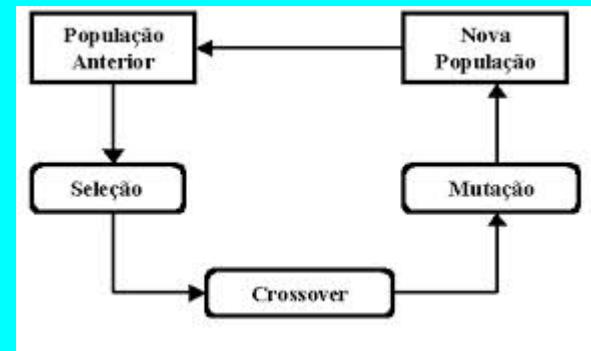
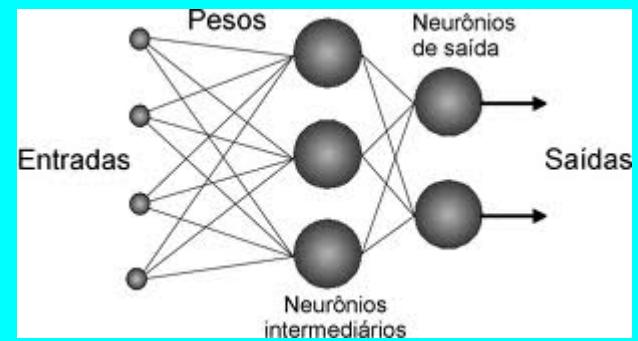
ARMA

# Classe é uma metáfora

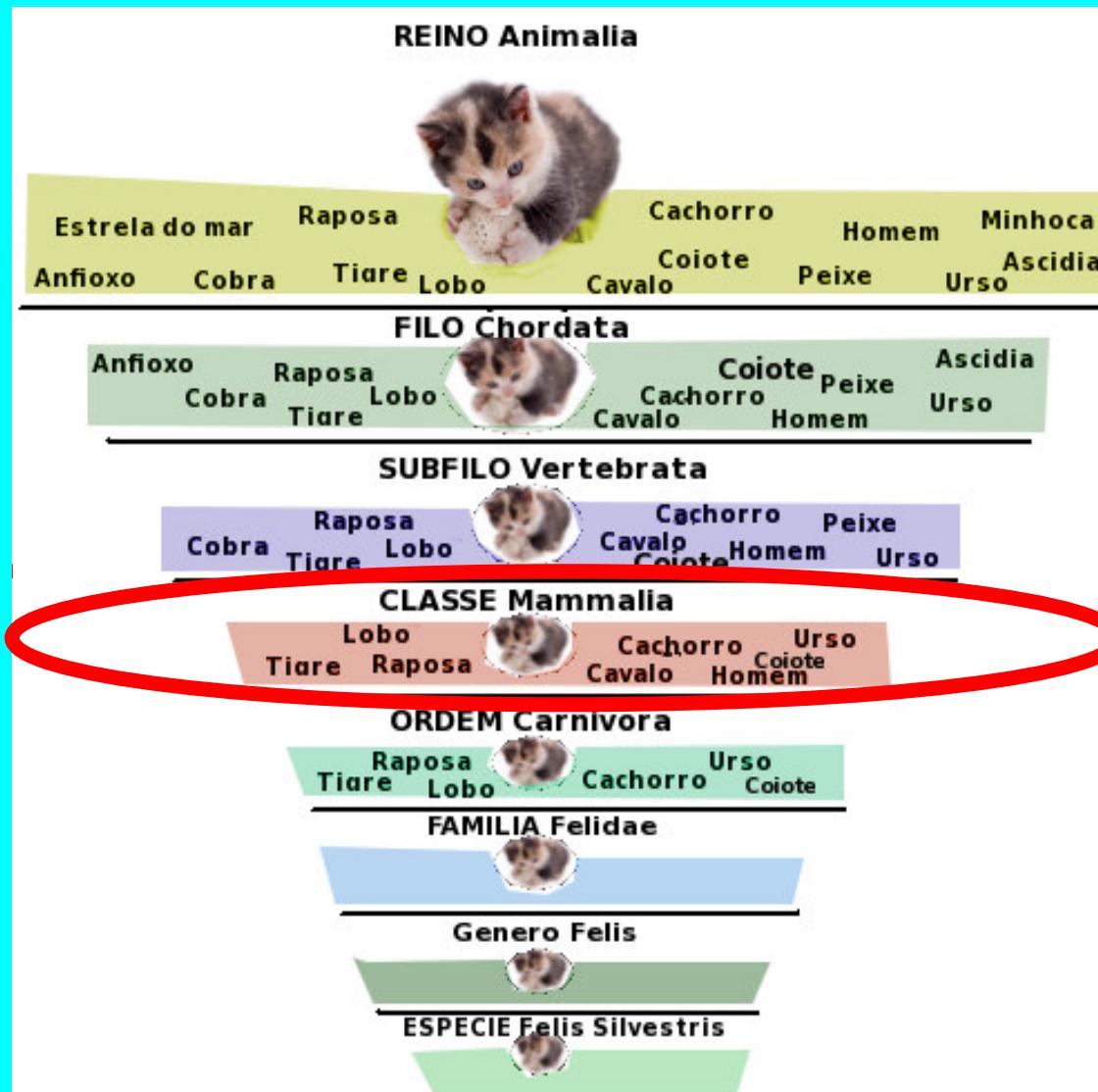
Uma das fontes preferidas de inspiração da programação de computadores é a **biologia**.

Exemplos:

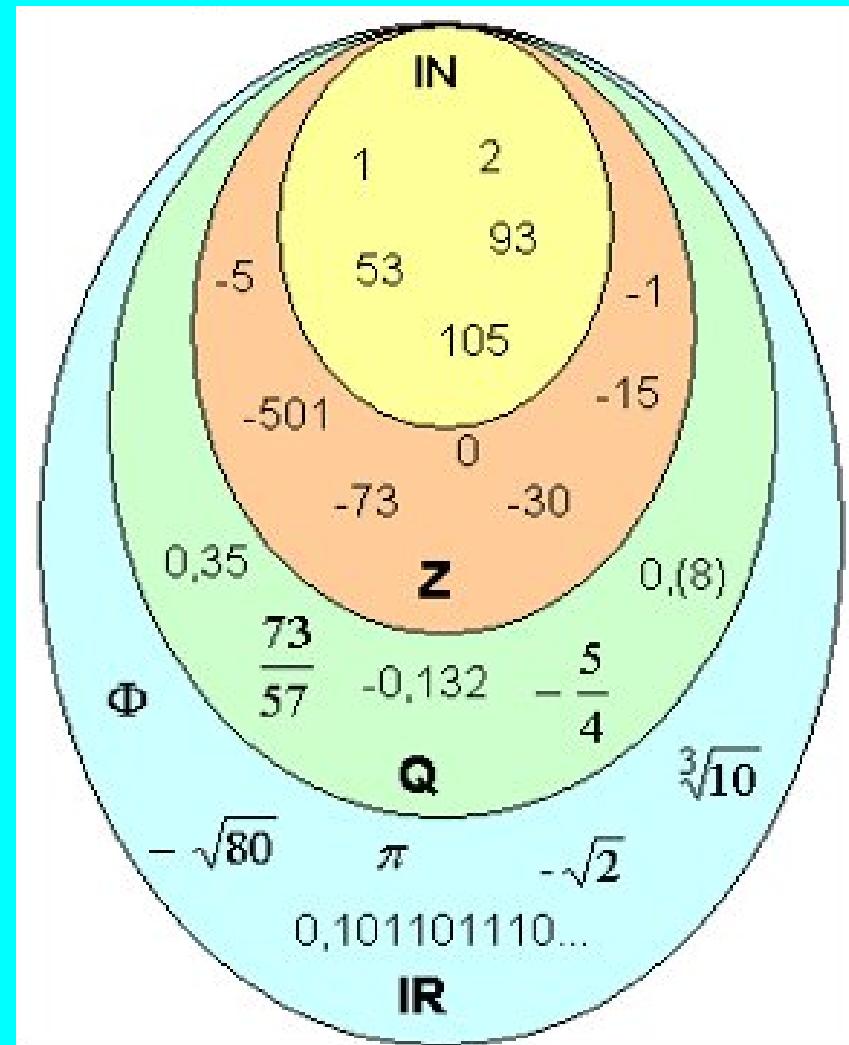
- Redes neurais
- Algoritmos genéticos



# Classe



# Classe



# Classe

```
<?php  
class Mammalia {  
}  
?  
>
```

Arquivo: Mammalia.php

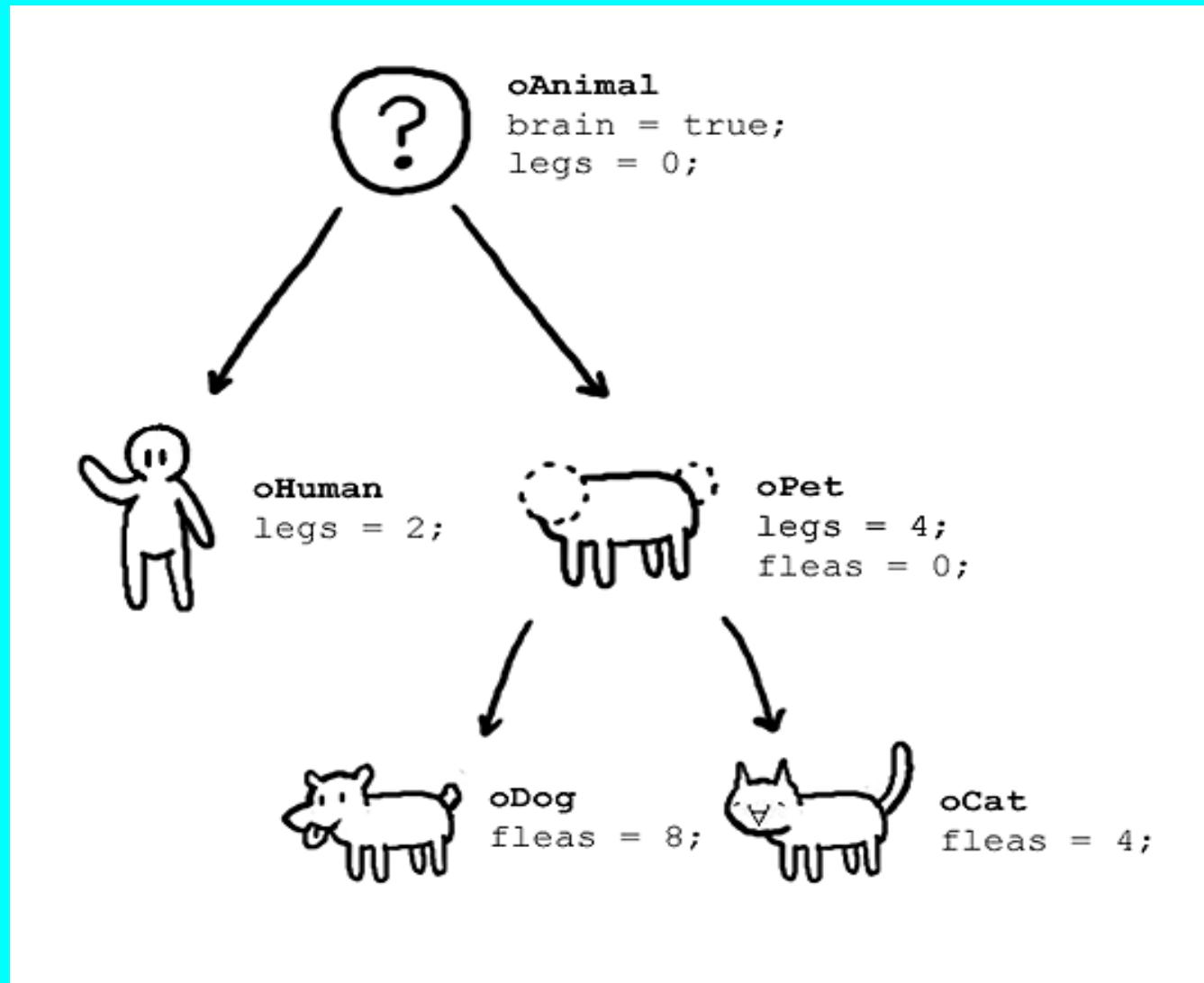
# Classe

*“Classe é uma coleção de **dados** e **rotinas** que compartilham uma responsabilidade **coesa** e **bem definida**”*

McConnell (2005, p. 154)

- **Dados:** matéria-prima
- **Rotinas:** processos que **transformam** ou **transportam** a matéria-prima

# Classe



# Relembrando: objeto

“Um **objeto** representa um 'elemento' que pode ser identificado de maneira **única**. Em um nível **apropriado** de abstração, praticamente **tudo** pode ser considerado como objeto”.

Coleman (1996, p. 16)



# Verificando...

```
<?php  
require 'Mammalia.php';  
  
$lobo = new Mammalia();  
$cachorro = new Mammalia();  
$urso = new Mammalia();  
  
var_dump($lobo);  
var_dump($cachorro);  
var_dump($urso);
```

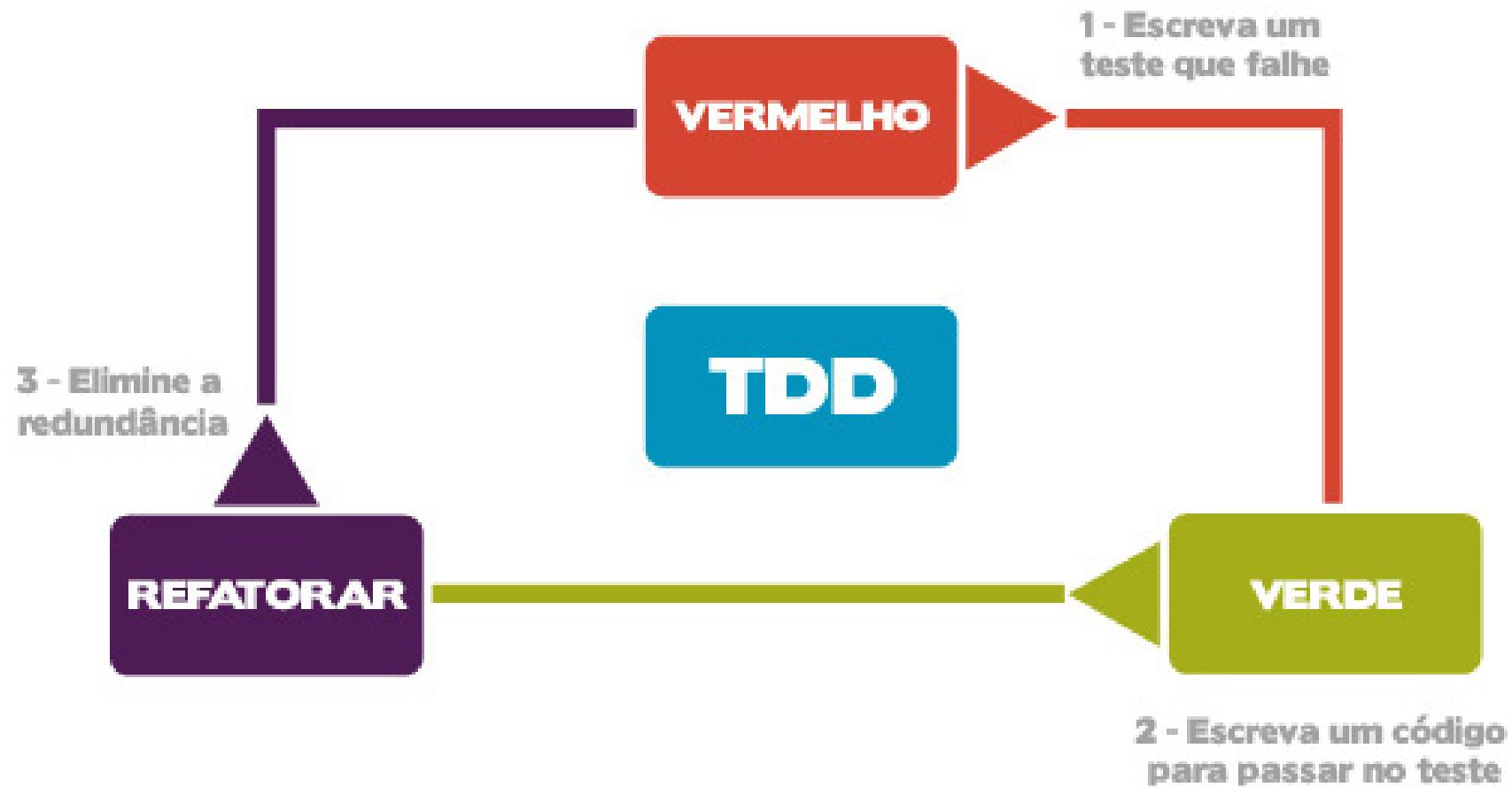
# Verificando...

```
php testeMammalia.php
object(Mammalia)#1 (0) {
}
object(Mammalia)#2 (0) {
}
object(Mammalia)#3 (0) {
}
```

ISSO É UM TESTE?  
QUE HORROR! EU  
TENHO QUE  
**INTERPRETAR** O  
RESULTADO! NÃO  
DAVA PRA  
SIMPLEMENTE DIZER  
SE ESTÁ FAZENDO  
**CERTO OU ERRADO?**



# Relembrando...



# Unicidade de objetos

A função **spl\_object\_hash()** retorna um identificador **único** para um objeto.

Em vez de imprimir o objeto, podemos comparar o retorno dessa função para uma amostra de objetos e imprimir “Passou” se todos forem diferentes, ou “Falhou” se pelo menos dois forem iguais.

# Carregamento de classes

A função **spl\_autoload\_register()** simplifica o carregamento de classes, evitando uma longa lista de requires ou includes.

Ela permite definir funções ou métodos de classes para carregar arquivos com declarações de classes a partir do *namespace* completo da classe.

# Atributos (ou Propriedades)

*“Um atributo equivale a um **elemento de dados** em um registro. Também faria sentido pensar em um atributo como uma **variável**”*



Ambler (1997, p. 124)

*“Tem-se por propriedades características **intrínsecas** à classe em questão”*



Dall'Oglio (2009, p. 90)

# Atributos

```
<?php  
class Mammalia {  
    private $cauda;  
    private $dentes;  
    private $olhos;  
    private $spelo;  
}  
  
?>
```

# Atributos

```
<?php
class Mammalia {
    /**
     *
     * @var boolean
     */
    private $cauda = FALSE;
    /**
     *
     * @var integer
     */
    private $dentes = 0;
    /**
     *
     * @var string
     */
    private $olhos = 'pretos';
    /**
     *
     * @var string
     */
    private $pelo = 'fino';
}

?>
```

# Métodos

*“Um método pode ser visto como uma **função** ou um **procedimento**. Os métodos acessam e modificam os atributos de um objeto. Alguns métodos retornam um valor (como uma função), enquanto outros não fazem isso (de maneira análoga aos procedimentos)”.*

Ambler (1997, p. 124)

*“A palavra método vem do grego, methodos, composta de meta: através de, por meio, e de hodos: via, caminho. Servir-se de um método é, antes de tudo, tentar ordenar o trajeto através do qual se possa alcançar os objetivos projetados.” (EBA, 2014)*

# Métodos

```
/**
 *
 * @param boolean $cauda
 */
public function setCauda($cauda){
$this->cauda = $cauda;
}

/**
 *
 * @return boolean
 */
public function getCauda(){
return $this->cauda;
}
```

# Vamos testar?



# Visibilidade de atributos e métodos

*“Um dos recursos mais interessantes na orientação a objetos é o **encapsulamento**, um mecanismo que provê **proteção de acesso** aos membros internos de um objeto”.*



Dall'Oglio (2009, p. 107)

# Visibilidade de atributos e métodos

```
/**  
 *  
 * @var boolean  
 */  
private $cauda = FALSE;
```

Ou

```
/**  
 *  
 * @var boolean  
 */  
protected $cauda = FALSE;
```

Ou

```
/**  
 *  
 * @var boolean  
 */  
public $cauda = FALSE;
```

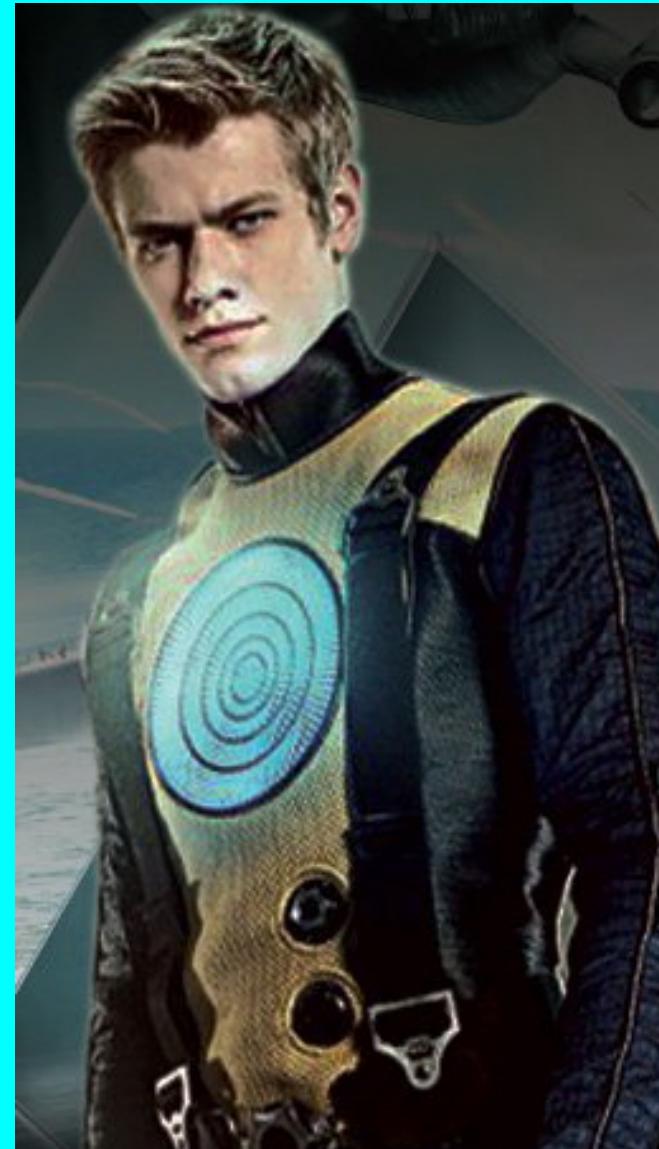
# Vamos testar?



# Constantes de classes

```
/**  
 *  
 * @var string  
 */  
const PRETOS = 'pretos';  
/**  
 *  
 * @var string  
 */  
const CASTANHOS = 'castanhos';  
/**  
 *  
 * @var string  
 */  
const VERDES = 'verdes';  
/**  
 *  
 * @var string  
 */  
const AZUIS = 'azuis';
```

# Construtor e destrutor



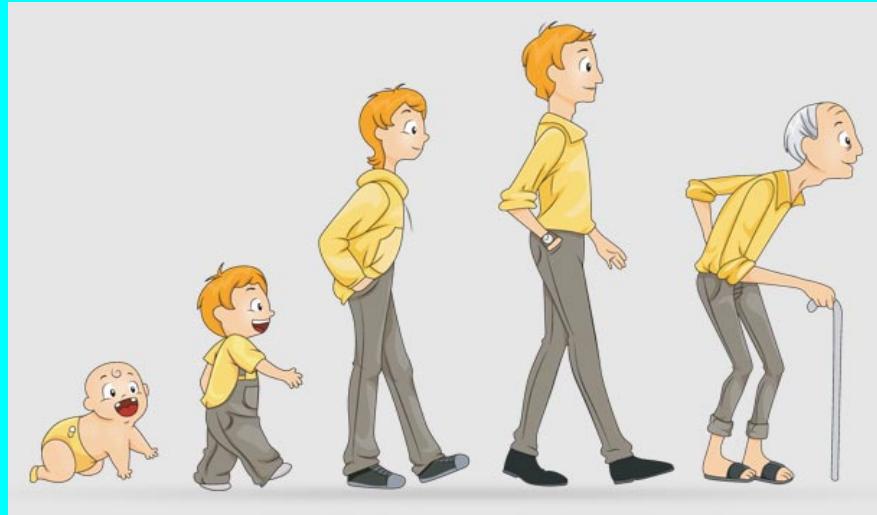
# Construtor e destrutor

```
public function __construct()  
{  
}  
  
public function __destruct()  
{  
}  
}
```

# Construtor e destrutor

*“Diz-se que o objeto é uma instância de uma classe, porque o objeto existe durante um dado instante de tempo – da sua criação até a sua destruição”.*

*Dall'Oglio (2009, p. 93)*



# Vamos testar?



# Atributos e métodos de classe

- Internamente, um objeto refere-se aos seus atributos e métodos com a palavra **\$this** e o operador →
- Uma classe declara seus atributos e métodos com a palavra **static** e refere-se aos seus atributos e métodos com a palavra **self** e o operador ::

# Vamos testar?



# Herança

*“É comum haver **similaridades** entre diferentes classes. Frequentemente, duas ou mais classes irão **compartilhar** os mesmos atributos e/ou métodos. Como nenhum de nós deseja **reescrever** várias vezes o mesmo código, seria interessante se algum mecanismo pudesse tirar proveito dessas similaridades. A herança é esse mecanismo”.*

Ambler (1997, p. 133)

# Herança

```
<?php  
abstract class Mammalia {
```

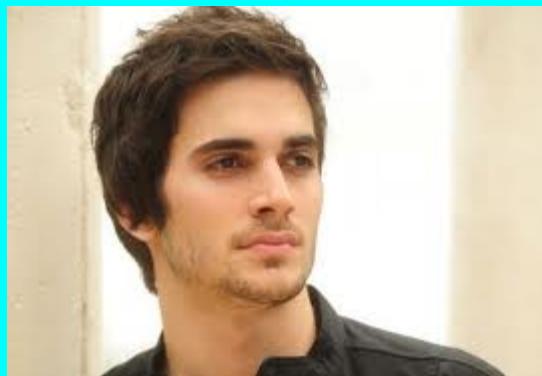
```
<?php  
class Carnivora extends \Mammalia {  
}
```

# Olha a metáfora aí, gente...

**Generalização**



**Especialização**



**Especialização**



# Vamos testar?



# Esterilidade

- A palavra **final** pode ser usada para impedir que uma classe gere herdeiras.
- Ela também pode ser usada para impedir que um método seja **sobrescrito** por uma classe herdeira.



# Vamos testar?



# Polimorfismo e Sobrecarga de Métodos

- Classes mães e herdeiras podem compartilhar métodos com mesmo nome, mas que executam operações **diferentes**. Isso é **polimorfismo**.
- Um método de uma classe herdeira pode fazer a operação da classe mãe e mais alguma coisa. Para não repetir o que está implementado na classe mãe, usamos a palavra **parent** para invocar o método, usando o operador ::

# Vamos testar?



# Interface

“Na etapa de projeto do sistema, podemos definir conjuntos de métodos que determinadas classes do nosso sistema deverão implementar **incondicionalmente**. Tais conjuntos de métodos são as *interfaces*, as quais contém a declaração de métodos de forma **prototipada**, sem qualquer implementação”.

Dall'Oglio (2009, p. 132)

# Interface

```
<?php  
interface Movimento {  
    public function andar();  
    public function correr();  
    public function parar();  
}  
  
?>
```

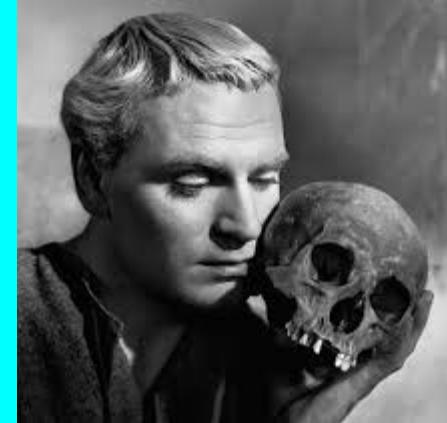
# Interface

```
<?php  
abstract class Mammalia implements Movimento{
```

# Vamos testar?



# Para que classes?



## Vantagens da Orientação a Objetos

- Reusabilidade
- Extensibilidade
- Aumento da qualidade
- Vantagens financeiras (qualidade, rapidez, economia)

Fonte: Ambler (1997, p. 6-7)



# Nem tudo são flores

## Problemas da Orientação a Objetos

- Exige maior concentração na análise e no projeto.
- Desenvolvedor e usuário precisam trabalhar em conjunto.
- Requer mudança de mentalidade e cultura.
- Os benefícios são evidenciados a longo prazo.
- Demanda treinamento.
- Não garante que o sistema será adequado.
- É apenas parte da solução.

Fonte: Ambler (1997, p. 19-21)

# Quando usar Orientação a Objetos

Principalmente, mas não somente, para:

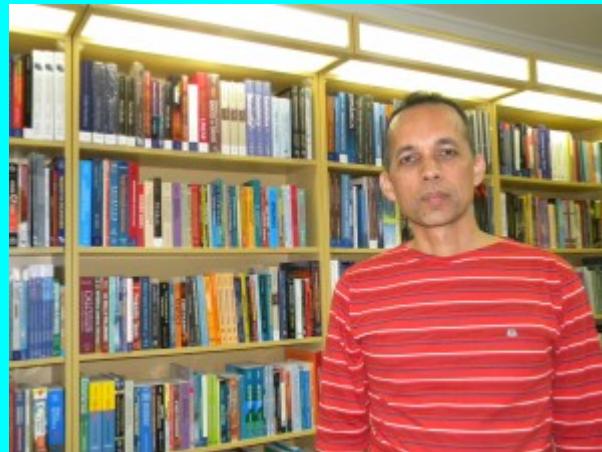
- Desenvolvimento de sistemas complexos.
- Desenvolvimento de sistemas sujeitos a mudanças.



# Requisitos Não-Funcionais

*“Os requisitos não-funcionais tem um papel de suma importância durante o desenvolvimento de um sistema, podendo ser usados como critérios de seleção na escolha de alternativas de projeto, estilo arquitetural e forma de implementação”*

Mendes (2002, p. 44)



# Requisitos Não-Funcionais

- Usabilidade
- **Manutenibilidade**
- Confiabilidade
- Desempenho
- Portabilidade
- Reusabilidade
- Segurança



Fonte: Mendes (2002, p. 45-56)

# Exercício

- O sistema de caixa de um restaurante em um shopping center de Corumbá precisa receber em **três moedas diferentes**: dólar norte-americano, real brasileiro e guarani paraguaio e retornar o troco em reais.
- Modele uma classe que receba valores de moedas diferentes, **converta-as** para real e faça operações de **soma e subtração** com as três moedas.
- Escreva um teste para essa classe.

# E agora, o momento que todos esperavam...



# Exercício de Avaliação

Crie um conjunto de classes PHP que simule o comportamento de um elevador e que valide esse comportamento com testes unitários.

O caso de uso a ser implementado é a notificação do **elevador** ao **andar** sobre qual é sua posição. A cada mudança de andar, provocada pelo pressionamento do botão correspondente ao andar, todos os andares devem ser informados da posição do elevador.

# Exercício de Avaliação

Nessa implementação, limite o uso do elevador à uma só **pessoa** de cada vez. Uma pessoa chama o elevador, ele vai até o andar, a pessoa entra, indica o andar para onde vai, o elevador a leva e a pessoa sai.

A classe de teste deve informar, para o estudo de caso implementado, onde o elevador está e onde a pessoa está. Cada mudança de andar deve atualizar essas informações.

# Exercício de Avaliação

Exemplo de simulação:

- O elevador está no 5º andar e uma pessoa que está no 3º o chama.
- O elevador desce, a pessoa entra e aperta o 8º andar.
- O elevador vai até o 8º andar e a pessoa desce.

# Exercício de Avaliação

Exemplo de saída:

Pessoa está no 3º andar. Elevador está no 5º andar.

Pessoa chama elevador. Elevador está no 5º andar.

Pessoa está no 3º andar. Elevador está no 4º andar.

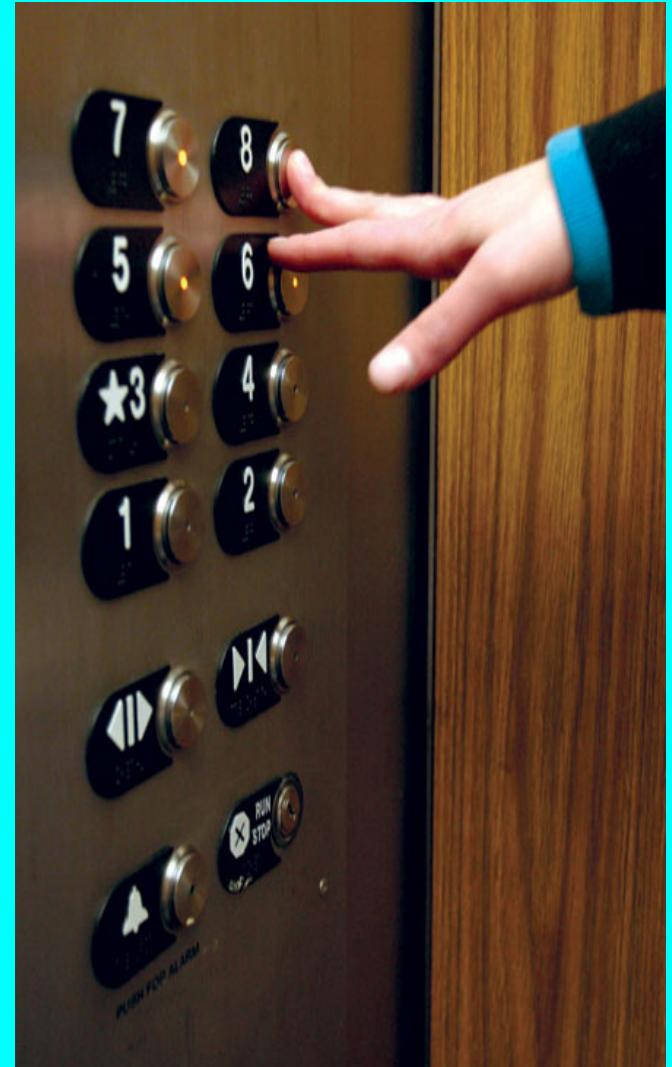
Pessoa está no 3º andar. Elevador está no 3º andar.

Pessoa entra no elevador. Elevador está no 3º andar.

Pessoa aperta 8º andar. Elevador está no 3º andar.

Pessoa está no elevador. Elevador está no 4º andar.

Pessoa aperta 8º andar. Elevador está no 3º andar.



# Exercício de Avaliação

Exemplo de saída (continuação):

Pessoa está no elevador. Elevador está no 4º andar.

Pessoa está no elevador. Elevador está no 5º andar.

Pessoa está no elevador. Elevador está no 6º andar.

Pessoa está no elevador. Elevador está no 7º andar.

Pessoa está no elevador. Elevador está no 8º andar.

Pessoa sai do elevador. Elevador está no 8º andar.

Pessoa está no 8º andar. Elevador está no 8º andar.



# Boa sorte!



Boa Sorte  
Charlie

# Bibliografia

- **Ambler, S. W.** *Análise e Projeto Orientados a Objeto*. Rio de Janeiro. Infobook, 1997.
- **Dall'Oglio, P.** *PHP: programando com orientação a objetos*. 2. ed. São Paulo. Novatec, 2009.
- **Davis, H. T.** *História da Computação*. São Paulo. Atual, 1992.
- **EBA.** *Tópicos em Artes Plásticas – Aula 7*. Disponível em <<http://www.eba.ufmg.br/graduacao/materialdidatico/apl001/aula007web.html>>. Acesso em 12/09/2014.
- **McConnell, S.** *Code Complete: Um guia prático para a construção de software*. 2.ed. Porto Alegre. Bookman, 2005.
- **Martin, R. C.** *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2009.
- **Mendes, A.** *Arquitetura de Software: desenvolvimento orientado para arquitetura*. Rio de Janeiro: Campus, 2002.
- **Pressman, R. S.** *Engenharia de Software*. 6.ed. São Paulo. McGraw-Hill, 2006.
- **Yourdon, E.** *Top Ten Software Engineering Concepts*. Disponível em <<http://www.yourdonreport.com/index.php/2008/11/13/top-ten-software-engineering-concepts-v10/>>. Acesso em 31;08;2015.