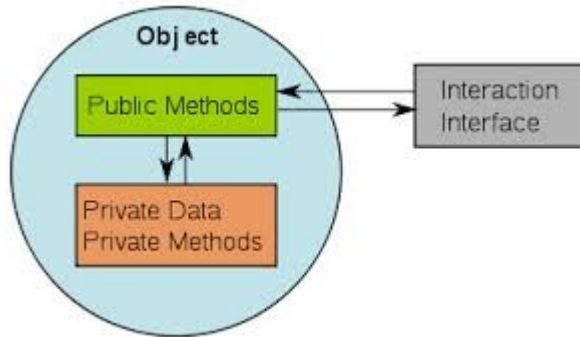
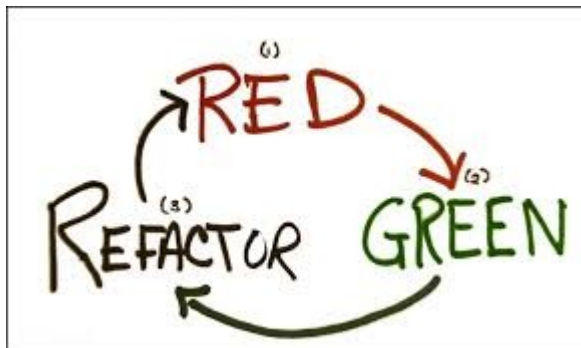


# Programação PHP orientada a objetos com testes unitários

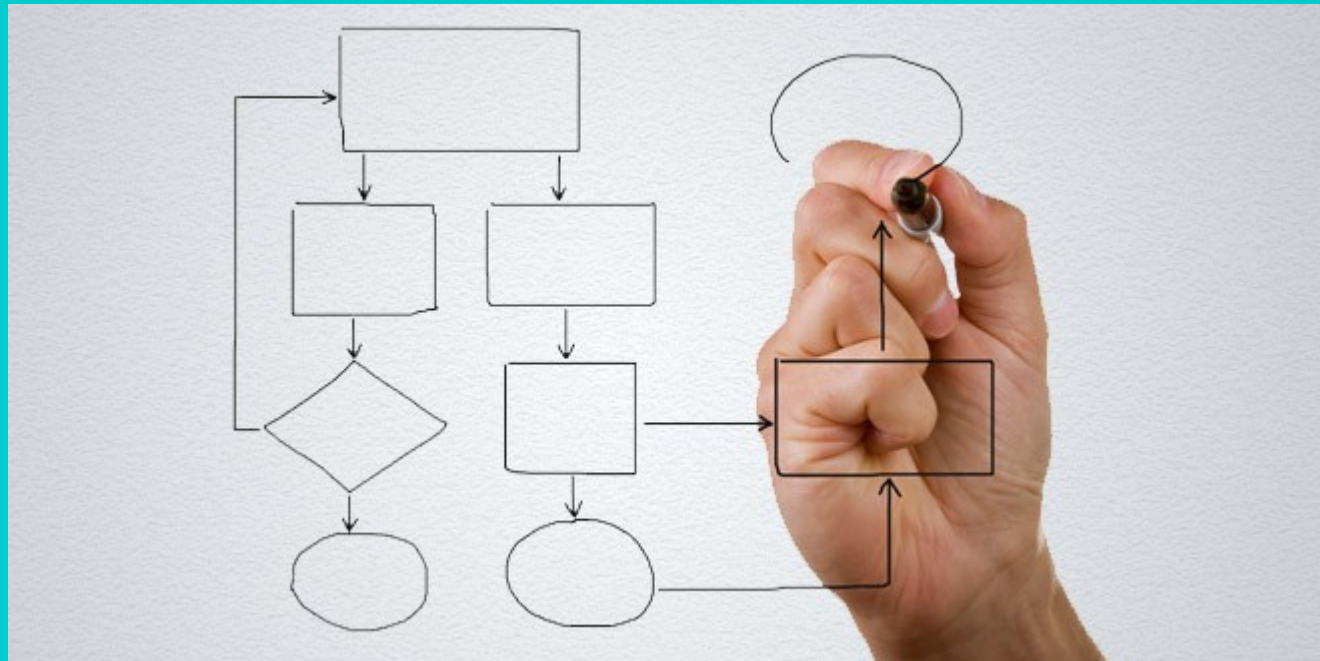
FLÁVIO GOMES DA SILVA LISBOA



2



# Planejamento



# Programação PHP orientada a objetos

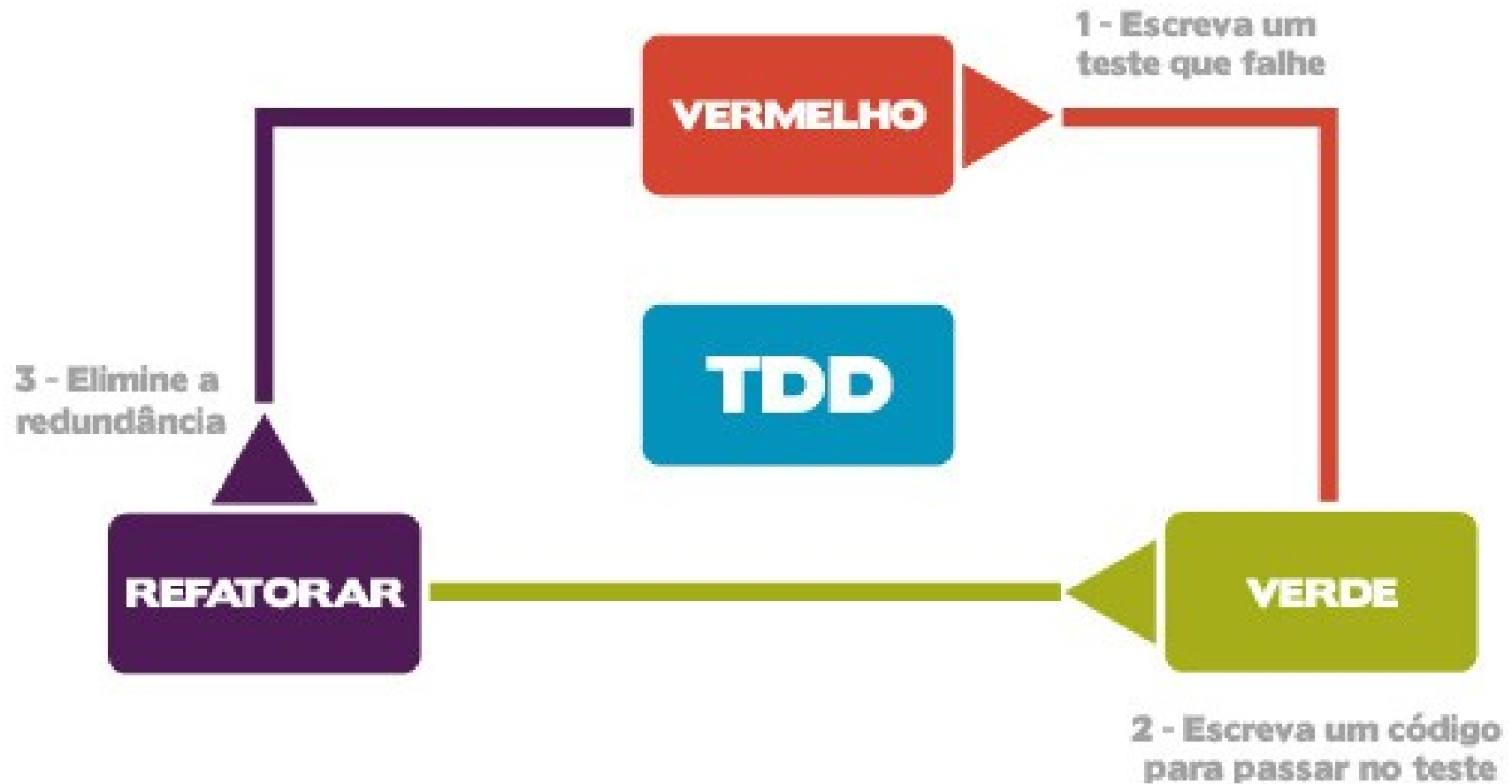
## Parte 2

- Clonagem de objetos
- Comparação de objetos
- Indução de tipo
- Objetos e Referências
- *Late Static Bindings*
- *Serialização de objetos*
- Métodos mágicos
- Padrões PHP

# E os testes unitários?



# TDD (*Test Driven Development*)



# Anteriormente, nos últimos capítulos...

- Escrevemos um *script* PHP
- Nesse *script* escrevíamos o que esperávamos que nosso código fizesse.
- Imprimíamos um texto indicando o resultado:  
**“Passou no teste”** ou **“Não passou no teste”**



[www.fgl.eti.br](http://www.fgl.eti.br)



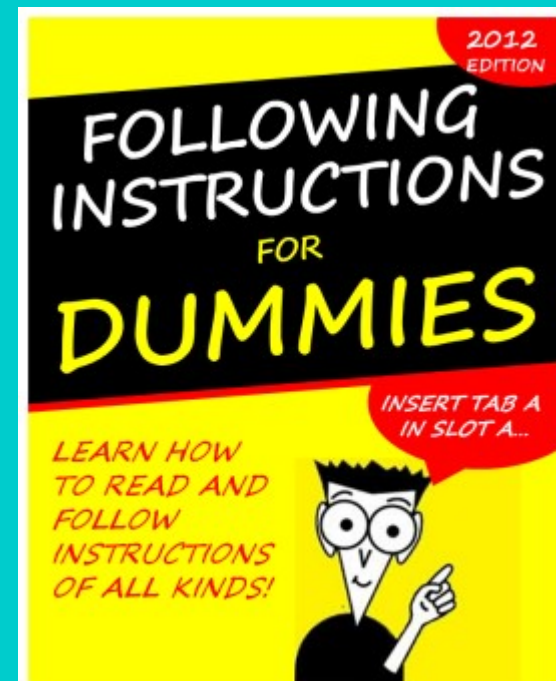
# Possibilidades de um teste

- Ter sucesso (o que era esperado ocorreu)
- Falhar (não ocorreu o que era esperado)
- Não ser concluído (erro fatal)



# Nosso algoritmo

- Execute o código
- Verifique o resultado do código
- Se for o esperado, escreva “Passou no teste”
- Senão, escreva “Não passou no teste”





# Problemas com os scripts...

- Temos de executar um por um



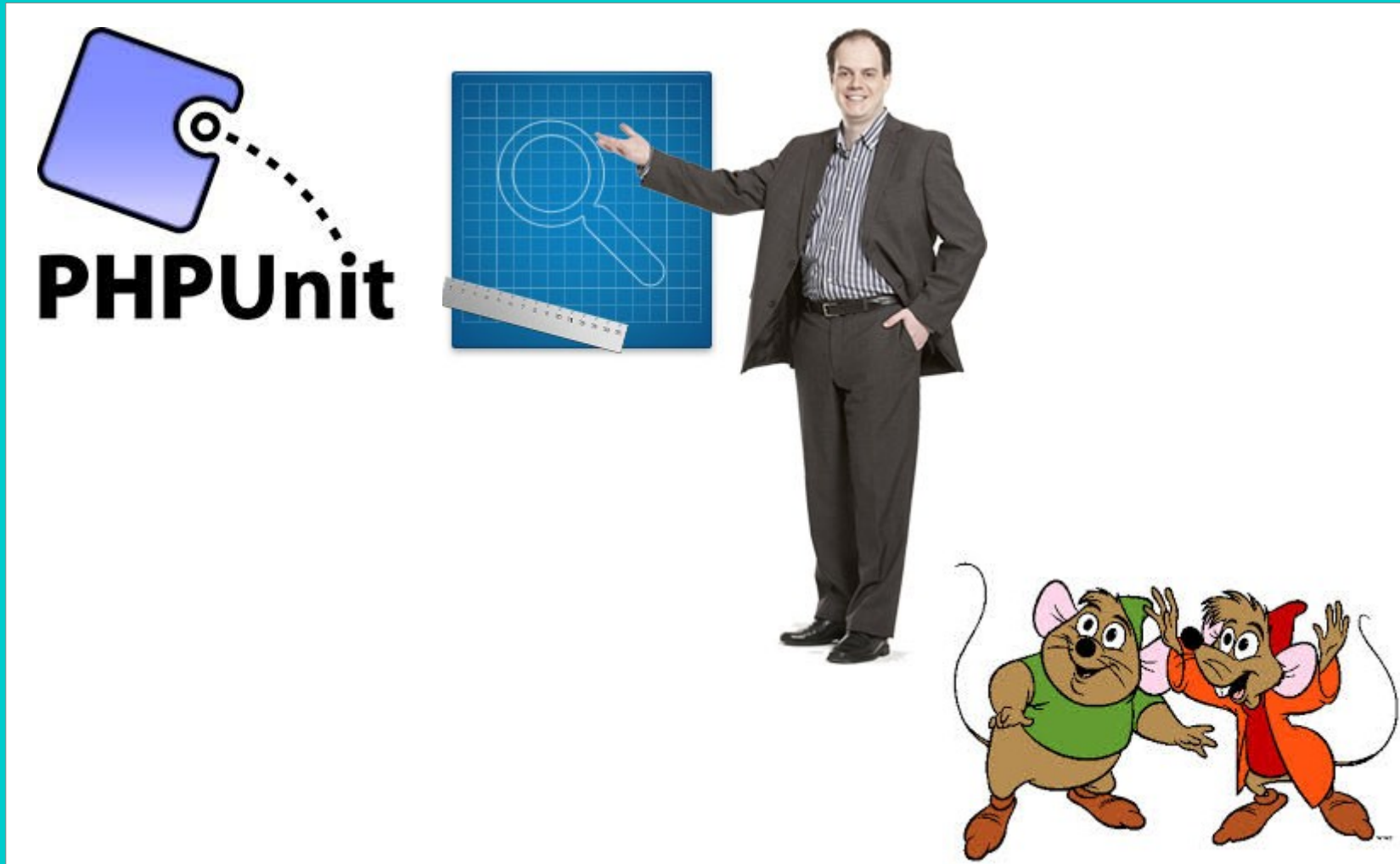


COMO SERIA  
MARAVILHOSO SE  
HOUVESSE UM JEITO  
MAIS FÁCIL DE  
ESCREVER E  
EXECUTAR TESTES...  
MAS DEVE SER UM  
SONHO...

# Seu problema acabou!



# PHPUnit chegou!



# PHPUnit

## Code

src/Money.php

```
<?php
class Money
{
    private $amount;

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

    public function getAmount()
    {
        return $this->amount;
    }

    public function negate()
    {
        return new Money(-1 * $this->amount);
    }

    // ...
}
```

## Test Code

tests/MoneyTest.php

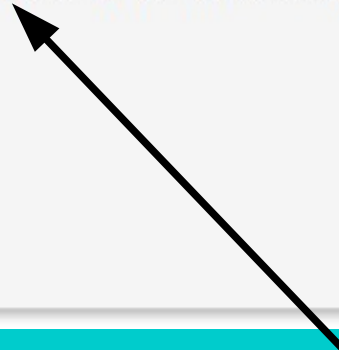
```
<?php
class MoneyTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testCanBeNegated()
    {
        // Arrange
        $a = new Money(1);

        // Act
        $b = $a->negate();

        // Assert
        $this->assertEquals(-1, $b->getAmount());
    }

    // ...
}
```



# PHPUnit

Um caso de teste PHPUnit é uma classe que estende **PHPUnit\_Framework\_TestCase**.

Cada método de uma classe *TestCase* iniciado pelo prefixo **test** é um **teste unitário**.

O método **setUp()** é executado antes de todos os testes unitários e o método **tearDown()** é executado ao final da execução dos testes. O primeiro serve para preparar o ambiente para os testes e o segundo serve para retornar ao estado inicial, antes dos testes.

# PHPUnit

Em PHPUnit não imprimimos nenhuma saída para informar o resultado. O PHPUnit imprime para nós.

```
PHPUnit X.Y.Z by Sebastian Bergmann.
```

```
Configuration read from [nome do arquivo]
```

```
.
```

```
Time: [NN] ms, Memory: [N.MM]Mb
```

```
OK ([número de testes unitários], [número de assertivas])
```

# Assertivas

*“O primeiro estágio da **tolerância a defeitos** é detectar se um defeito (um estado errôneo do sistema) ocorrerá a não ser que alguma ação seja tomada imediatamente. Para fazer isso, você precisa saber quando o valor de uma **variável de estado** é **ilegal** ou quando **relacionamentos entre variáveis de estados** não são mantidos”*

Sommerville (2009, p. 314)





# Assertivas

*“As assertivas podem ajudar a detectar erros **antecipadamente**, em especial, em sistemas **grandes**, sistemas de **alta confiabilidade** e bases de código que **mudam com muita frequência**”*

McConnell (2005, p. 238)

# Assertivas



- `assertArrayHasKey()`
- `assertClassHasAttribute()`
- `assertClassHasStaticAttribute()`
- `assertContains()`
- `assertContainsOnly()`
- `assertContainsOnlyInstancesOf()`
- `assertCount()`

# Assertivas



- `assertEmpty()`
- `assertEqualXMLStructure()`
- `assertEquals()`
- `assertFalse()`
- `assertFileEquals()`
- `assertFileExists()`
- `assertGreaterThan()`
- `assertGreaterThanOrEqual()`

# Assertivas

- `assertInstanceOf()`
- `assertInternalType()`
- `assertJsonFileEqualsJsonFile()`
- `assertJsonStringEqualsJsonFile()`
- `assertJsonStringEqualsJsonString()`
- `assertLessThan()`
- `assertLessThanOrEqual()`
- `assertNull()`



# Vamos baixar o PHPUnit?

<https://phpunit.de/>


## Download PHPUnit

All official releases of code distributed by the PHPUnit Project are signed by the release manager for the release. PGP signatures and SHA1 hashes are available for verification on [phar.phpunit.de](https://phar.phpunit.de). Please refer to the [documentation](#) for details on how to verify PHPUnit releases.

### Stable Release

PHPUnit 4.2 is the current **stable** release series of PHPUnit. It became *stable* on August 8, 2014.


You can find out what's new in PHPUnit 4.2 in the [ChangeLog](#).

 Download PHPUnit 4.2  
(latest stable release)

### Old, But Stable Release

PHPUnit 3.7 is the current **old, but stable** release series of PHPUnit. It became *stable* on September 19, 2012.

You can find out what's new in PHPUnit 3.7 in the [ChangeLog](#).

 Download PHPUnit 3.7  
(old, but stable release)

# Vamos rever os testes que fizemos?



# Executando um caso de teste PHPUnit

```
phpunit NameOfClassTest.php
```



# Executando vários casos de teste PHPUnit

Um arquivo XML pode definir uma pasta com casos de teste para o PHPUnit executar.

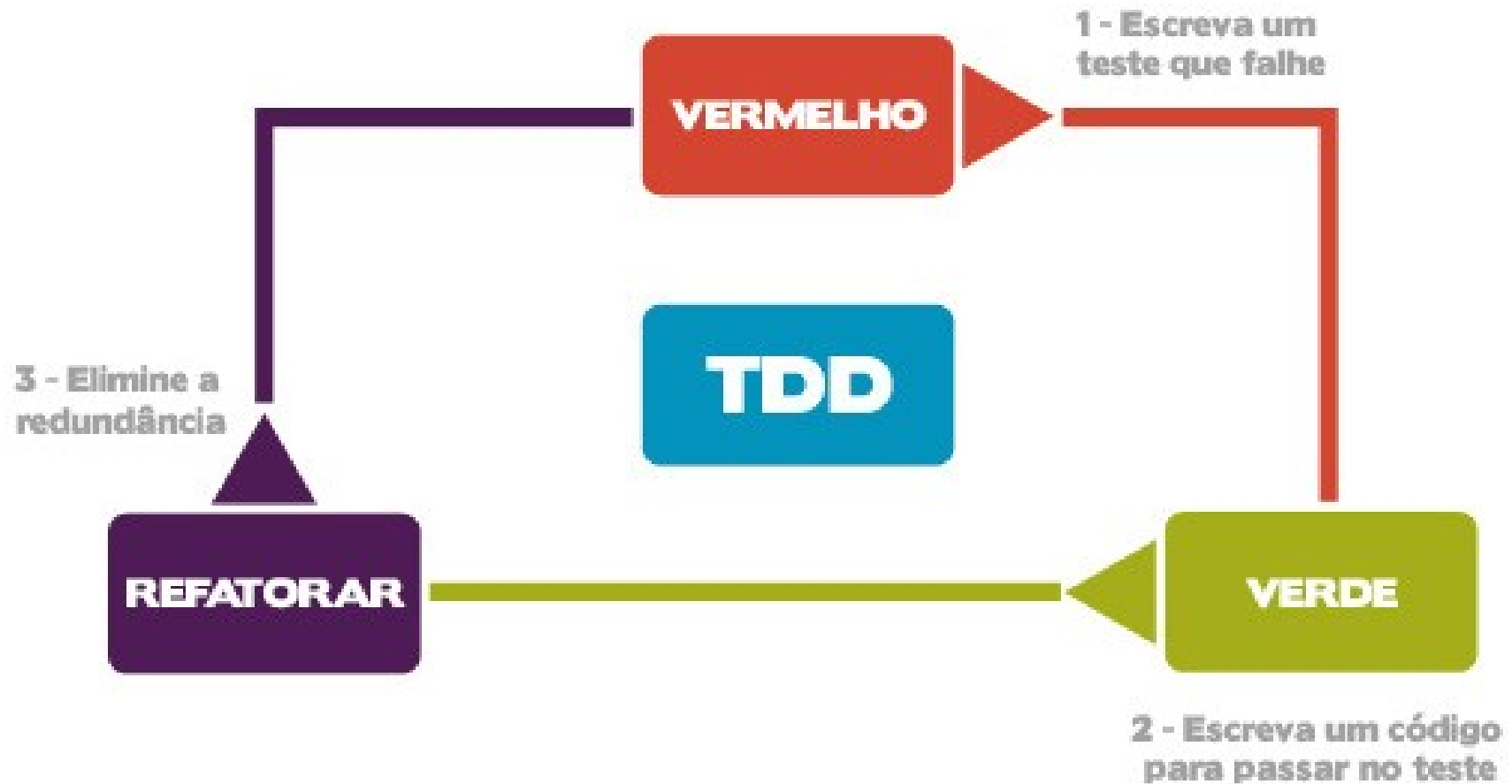
```
phpunit -c tests.xml
```



# Executando vários casos de teste PHPUnit

```
<phpunit>
  <testsuites>
    <testsuite name="webdev">
      <directory>tests</directory>
    </testsuite>
  </testsuites>
</phpunit>
```

# TDD (*Test Driven Development*)



# O Mantra do TDD

- **Vermelho** – Escrever um pequeno teste que não funcione e que talvez nem mesmo compile inicialmente.
- **Verde** – Fazer rapidamente o teste funcionar, mesmo cometendo algum pecado necessário no processo.
- **Refatorar** – Eliminar todas as duplicatas criadas apenas para que o teste funcione.

Fonte: Beck (2010, p. x)

# As 3 Leis do TDD

- **Primeira Lei:** Você não pode escrever código de produção até ter escrito um teste unitário que falhe.
- **Segunda Lei:** Você não pode escrever mais de um teste unitário do que o suficiente para falhar, e não compilar é falhar.
- **Terceira Lei:** Você não pode escrever mais código de produção do que o suficiente para passar pelo atual teste que falha.

Fonte: Martin (2009, p. 153)



# Voltando aos objetos...



# Objetos

***“Objetos são, potencialmente, componentes reusáveis porque eles são encapsulamentos independentes de estado e operações”***

Sommerville (2009, p. 209)



# Clonagem de objetos



# Clonagem de objetos

- Um clone **não** é um filho.
- Uma classe herdeira **não** é clone de uma classe abstrata, pois não tem todos os atributos da classe mãe.





# Clonagem de objetos

- O operador **new** cria um novo objeto.

```
$objeto = new Classe();
```

- O operador **clone** cria um objeto idêntico a outro.

```
$copia = clone $objeto;
```

O clone é um outro objeto, com outro identificador, mas com os mesmos valores de atributos.

# Comparação de objetos



# Comparação de objetos



- A comparação entre objetos pode ser feita por **igualdade**, quando se compara se os objetos são da mesma classe e se seus atributos tem os mesmos valores. O operador é o `==`
- A comparação entre objetos pode ser feita por **identidade**, quando se compara se os objetos exatamente o mesmo, a mesma referência em duas variáveis diferentes. O operador é o `===`



# Comparação de classes



- A comparação de classes (se a classe de um objeto é igual a uma determinada classe) é feita com o operador **instanceof**.
- A função **get\_class()** retorna a classe de um objeto.
- A constante **\_\_CLASS\_\_** retorna o nome da classe onde ela está sendo chamada.
- A função **get\_called\_class()** retorna a classe que iniciou a chamada a um método.

# Vamos testar?



# Indução de Tipo

Funções e métodos podem forçar seus parâmetros a serem objetos pela especificação de uma **classe** ou **interface** na assinatura. Caso um valor não seja uma instância da classe especificada ou que não implemente a interface especificada, ocorrerá um erro fatal, a menos que o parâmetro tenha o valor padrão **NULL**.

(**Tipo** \$parametro)

# Vamos testar?







# Objetos e referências

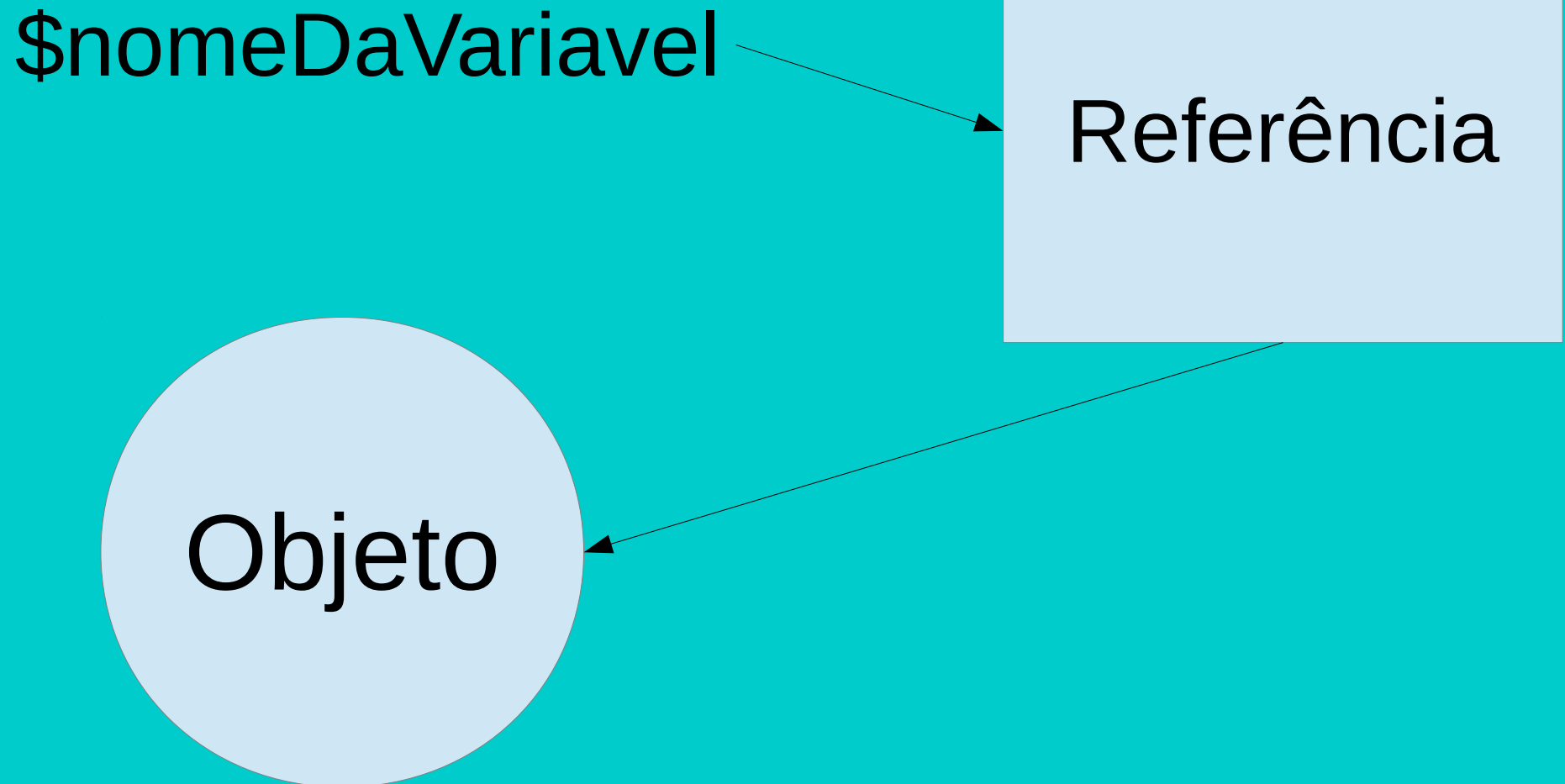


*“Uma **referência PHP** é um alias, que permite duas variáveis diferentes escreverem para o **mesmo valor**. A partir do PHP 5, uma variável objeto não contém mais o próprio objeto como valor. Ela contém um **identificador do objeto** que permite que os acessadores do objeto encontrem o objeto real. Quando um objeto é enviado por argumento, retornado ou atribuído para outra variável, as variáveis diferentes não são aliases: elas armazenam **uma cópia do identificador**, que aponta para o mesmo objeto. “*

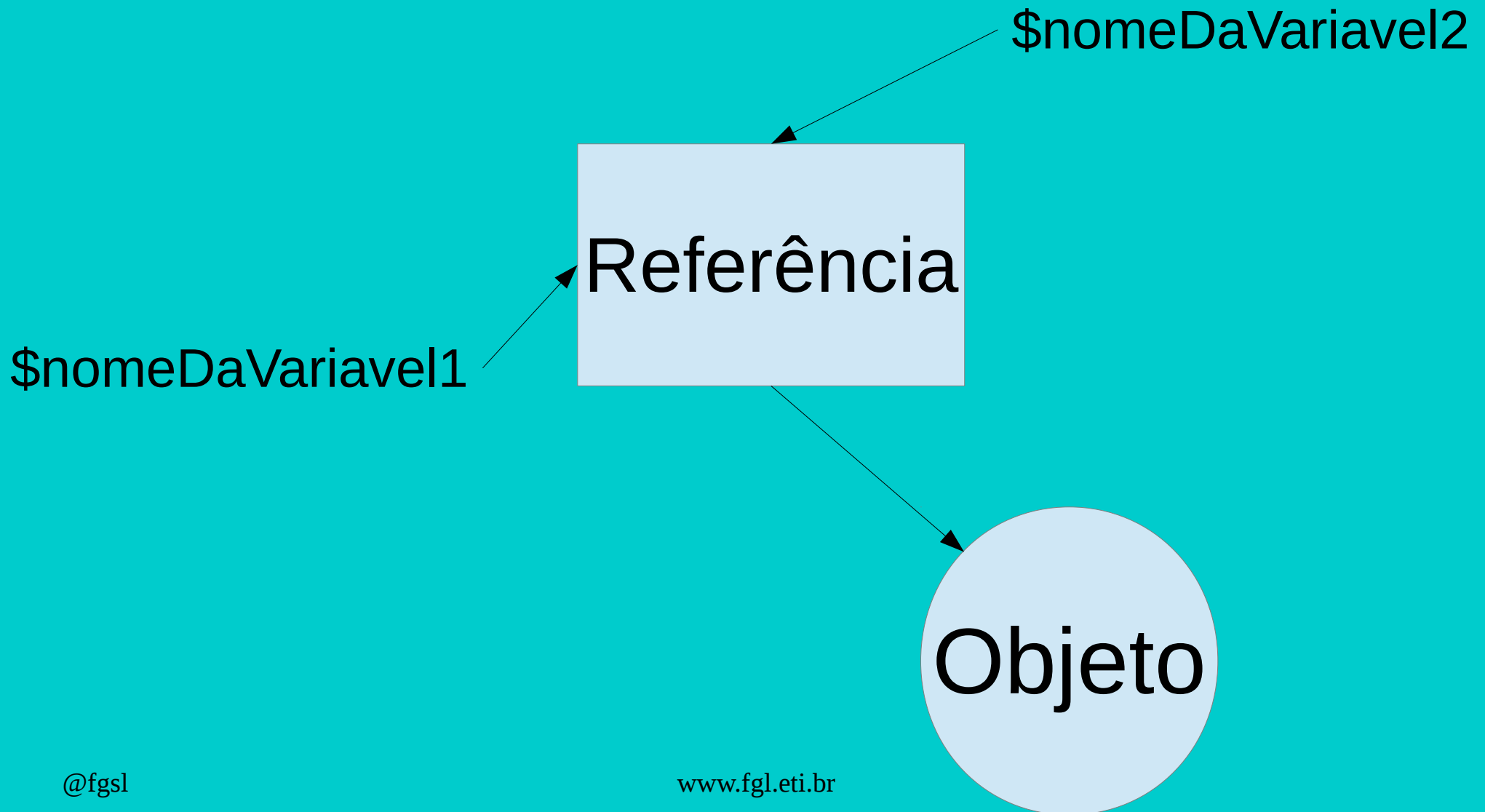
Fonte: [http://br1.php.net/manual/pt\\_BR/language.oop5.references.php](http://br1.php.net/manual/pt_BR/language.oop5.references.php)



# Objetos e referências



# Objetos e referências



# Vamos testar?



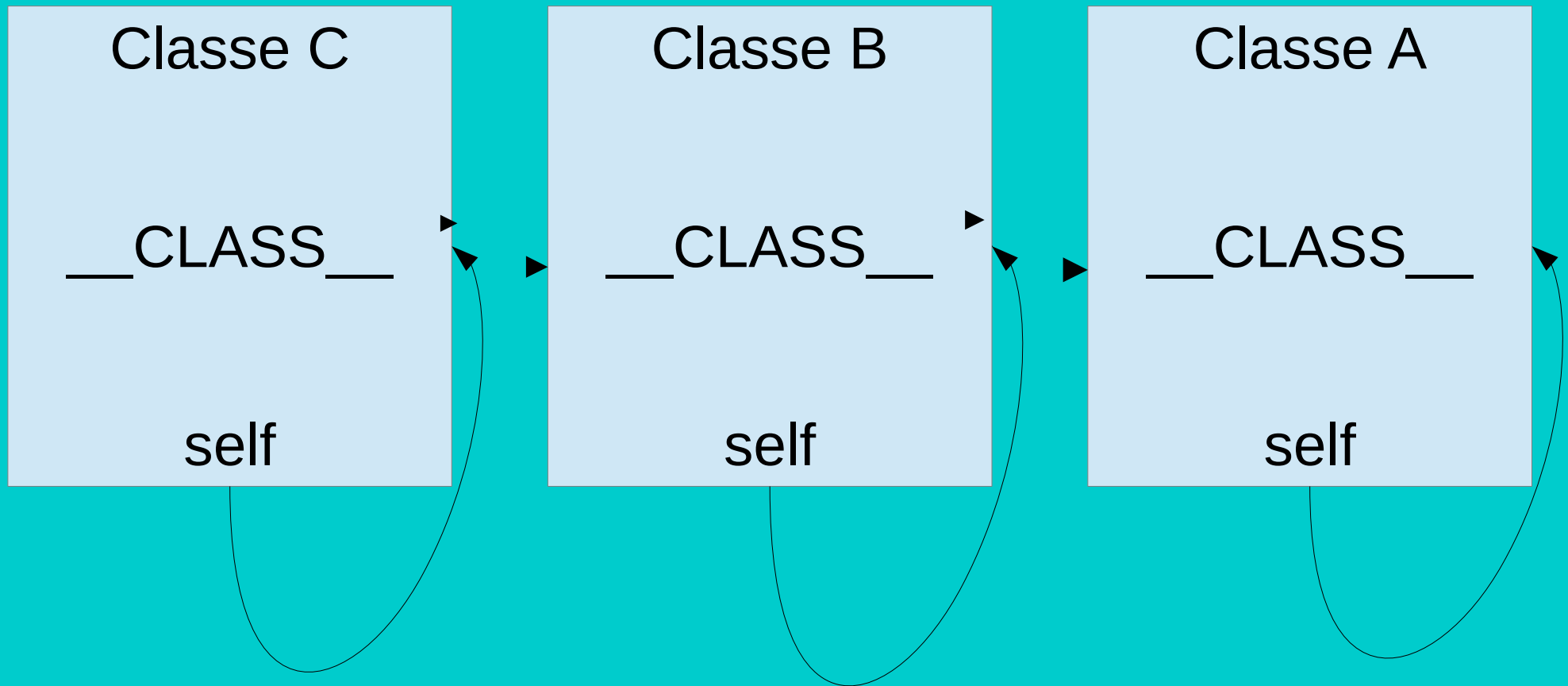
# *Late Static Bindings*

Referências estáticas para a classe atual como **self::** ou **\_\_CLASS\_\_** são resolvidas usando a classe a qual o método pertence.

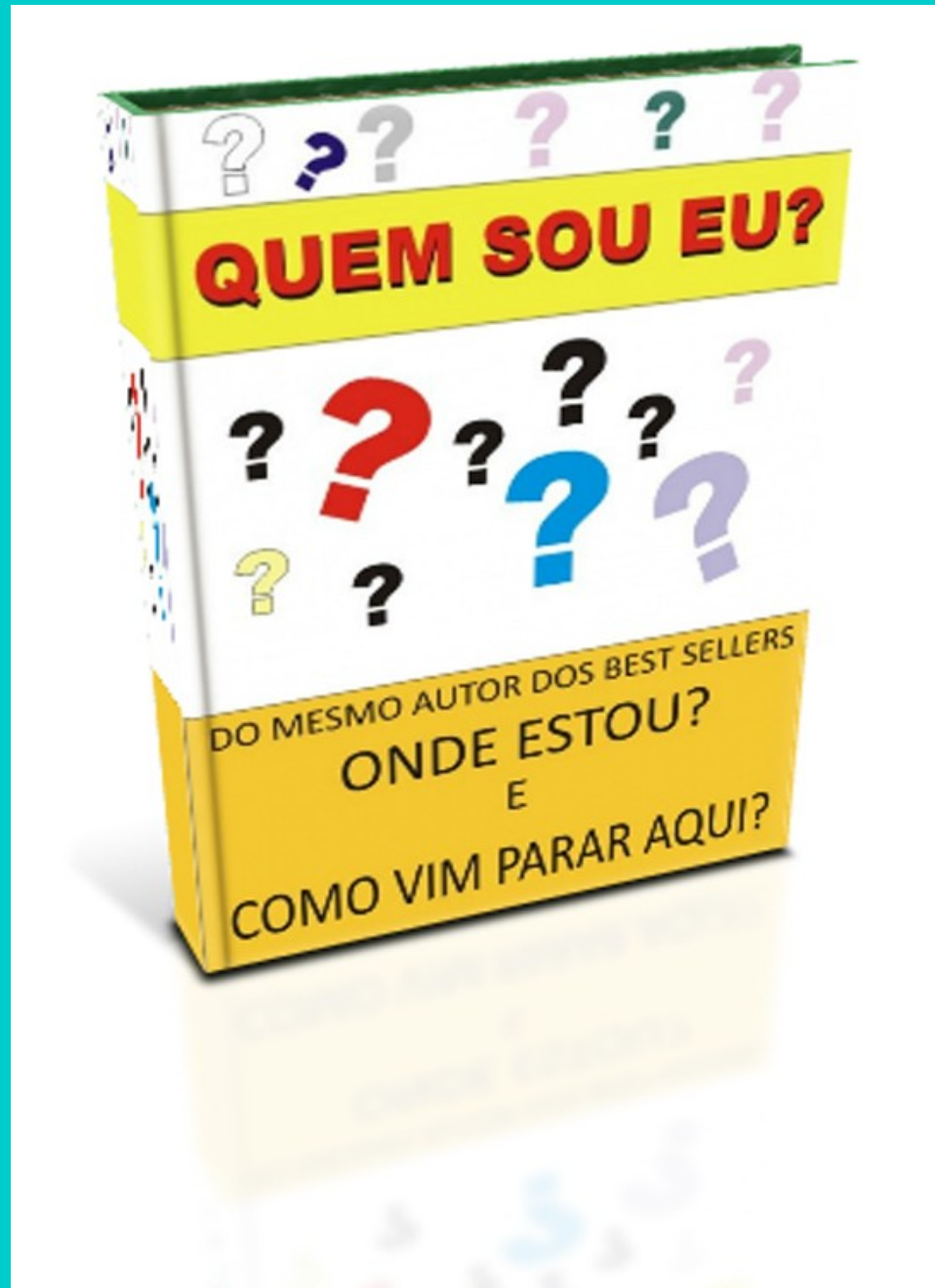
Isso significa que se uma classe usa um método herdado que contenha a palavra **\_\_CLASS\_\_**, o conteúdo dessa constante será o nome da classe mãe e não da filha.

E **self** chama os atributos e métodos da classe onde está declarado e não de onde foi chamado.

# *Late Static Bindings*



# *Late Static Bindings*



# *Late Static Bindings*

*Late static bindings* tenta resolver a limitação de **\_\_CLASS\_\_** e **self** introduzindo uma palavra-chave que referencia a classe que foi inicialmente chamada em *runtime*: **static**.



# Vamos testar?





# Serialização de objetos

**Serializar** um objeto significa transformá-lo em um texto que preserve seus dados e que permita reconstruí-lo depois.

A função **serialize()** cria uma representação textual de um valor em PHP. Para objetos, o formato do texto segue este modelo:

```
O:[id do objeto]:"[nome da classe]:  
[numero de atributos]:{[definição de  
atributos e seus valores]}
```

# Serialização de objetos

A definição de atributos contém geralmente as seguintes informações:

```
{N[se o atributo tiver valor];s:  
[comprimento do nome do  
atributo]:"[nome do atributo]";[tipo  
de atributo]s:[comprimento do valor  
do atributo se aplicável]:"[valor do  
atributo]";}
```

# Serialização de objetos

**Reverter a serialização de um objeto** significa reconstruir um objeto a partir do texto que o representa.

A função **unserialize()** cria um valor baseado em uma representação textual. Para reconstruir objetos, é necessário que a declaração da classe esteja disponível. Se ela não estiver, o PHP criará um objeto da classe **\_\_PHP\_Incomplete\_Class\_Name**, sem métodos.

Os dados serializados referem-se somente aos **valores** dos atributos. A informação sobre os métodos é da **classe**.

# Vamos testar?



# Métodos Mágicos



# Métodos Mágicos

PHP reserva todas as funções e métodos com nomes começando com `__` como **mágicos**. Por isso é recomendado que você não use funções e métodos com nomes com `__` no PHP.

Métodos mágicos não são chamados **diretamente**. Eles são executados em resposta a **eventos**.

Métodos mágicos podem ser **sobrecarregados**.



# Métodos Mágicos

Os métodos `__construct()` e `__destruct()` são exemplos de métodos mágicos.

Métodos mágicos podem **neutralizar** erros.



# Métodos Mágicos: `__set()`

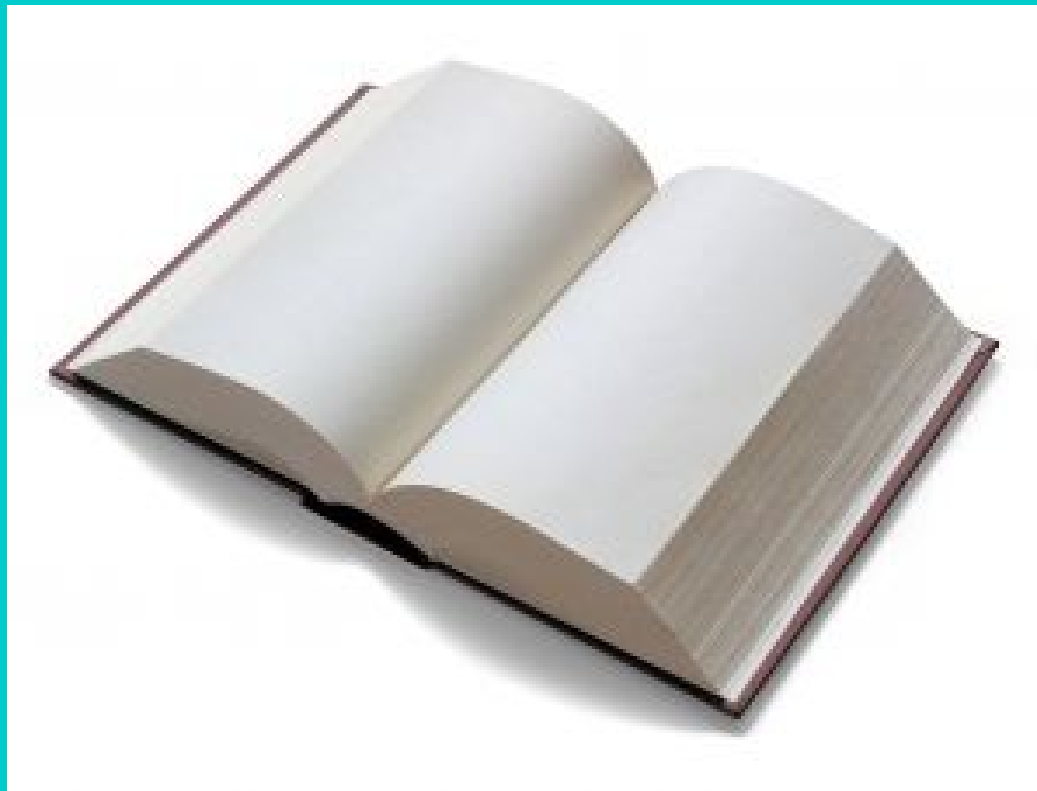
O método `__set()` é chamado quando há uma tentativa de atribuição de valor para um atributo inexistente em um objeto.





# Métodos Mágicos: `__get()`

O método `__get()` é chamado quando há uma tentativa de leitura de um atributo inexistente.

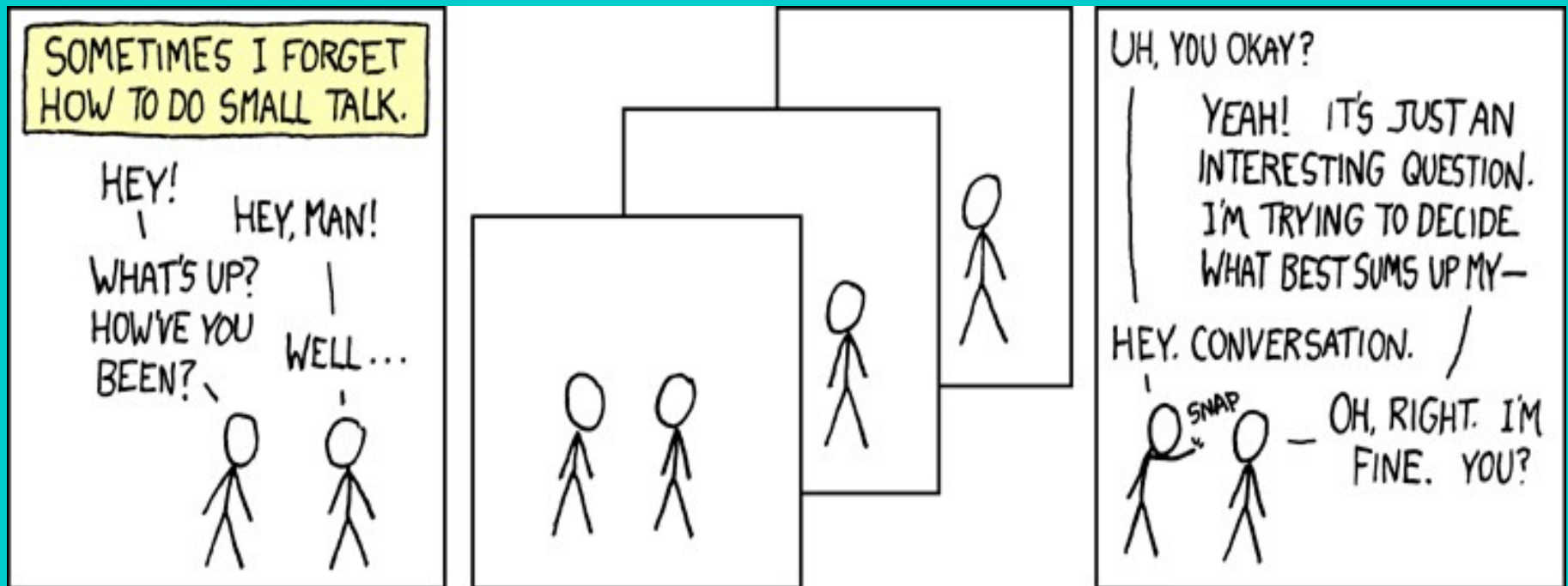


# Vamos testar?



# Métodos Mágicos: `__call()`

O método `__call()` é chamado quando é feita uma tentativa de invocar um método inexistente.



# Vamos testar?



# Métodos Mágicos: `__isset()`

O método `__isset()` é chamado quando um atributo do objeto é passado como argumento para o método `isset()`.

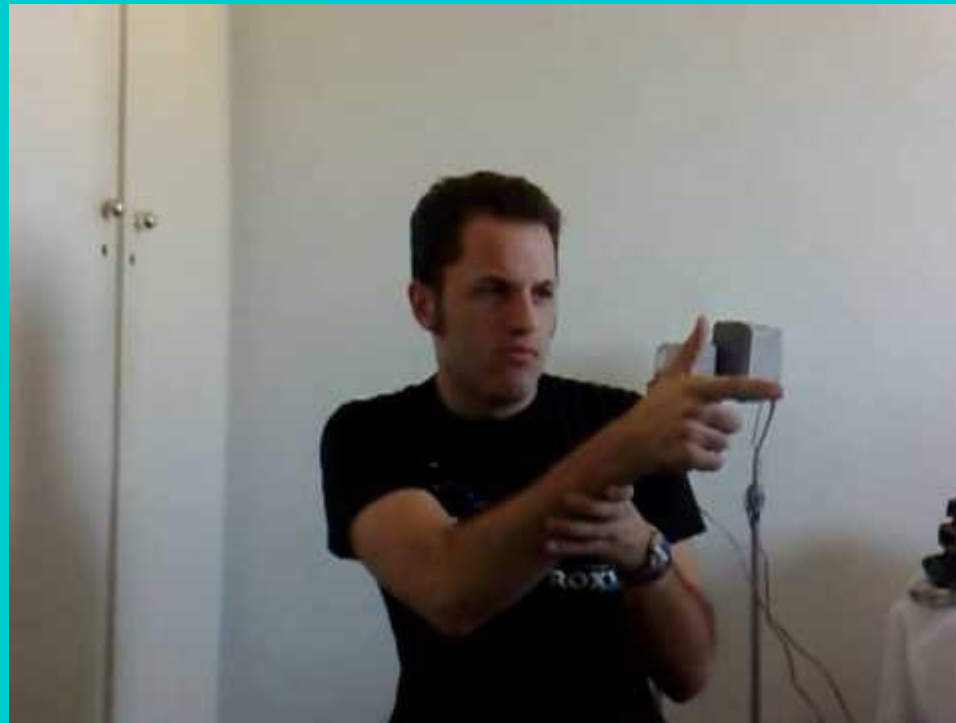


# Vamos testar?



# Métodos Mágicos: `__invoke()`

O método `__invoke()` é chamado quando uma variável contendo um objeto é usada como se fosse uma função.



# Vamos testar?





# Métodos Mágicos: `__callStatic()`

O método `__callStatic()` é chamado quando é feita uma tentativa de chamar um método estático inexistente.



# Vamos testar?



# Métodos Mágicos: `__toString()`

O método `__toString()` é chamado quando se trata um objeto como se fosse texto.

During what many archaeologists call the formative period, Amazonian societies were deeply involved in the emergence of South America's highland agrarian systems, and possibly contributed directly to the social and religious fabric constitutive of the Andean civilizational orders.

In 1500, Vicente Yáñez Pinzón was the first European to sail into the river. Pinzón called the river flow Río Santa María de la Mar Duke, later shortened to Mar Dulce (literally, sweet sea, because of its fresh-water pushing out into the ocean). For 350 years after the first European encounter of the Amazon by Pinzón, the Portuguese portion of the basin remained an untended former food gathering and planned agricultural landscape occupied by the indigenous peoples who survived the arrival of European diseases. There is ample evidence for complex large-scale, pre-Columbian social formations, including chiefdoms, in many areas of Amazonia (particularly the inter-fluvial regions) and even large towns and cities. For instance the pre-Columbian culture on the island of Marajo may have developed social stratification and supported a population of 100,000 people. The Native Americans of the Amazon rain forest may have used Terra preta to make the land suitable for the large scale agriculture needed to support large populations and complex social formations such as chiefdoms. One of Gonzalo Pizarro's lieutenants, Francisco de Orellana, during his 1541 expedition, east of Quito into the South American interior in search of El Dorado and the Country of the Cinnamon was ordered to explore the Coca River and return when the river ended. When they arrived to the confluence to the Napo River, his men menaced to mutiny if they did not continue. On 26 December 1541, he accepted to be elected chief of the new expedition and to conquest new lands in name of the king. The 49 men began to build a bigger ship for riverine navigation. During their navigation on Napo River they were threatened consistently by the Omaguas. They reached Negro River on 3 June 1542 and there I finally arrived to the Amazon River, that was so named because they were attacked by fierce female warriors like the mythological Amazons. The Icamiabas Indians dominated the area close to the Amazon River, rich in gold. When Orellana went down the river in search of gold, descends Andes (in 1541), the river was still called Grande Rio, Mar Dulce or Rio da Canela (Cinnamon), because of the great trees of cinnamon located there. The belligerent victory of the Icamiabas against the Spanish invaders was such that the fact was narrated to the king Carlos V, whom, inspired by the Greek Amazons, baptized the river as Amazon. In what is currently Brazil, Ecuador, Bolivia, Colombia, Peru, and Venezuela a number of colonial and religious settlements were established along the banks of primary rivers and tributaries for the purpose of trade, slaving and evangelization among the indigenous peoples of the vast rain forest. The total population of the Brazilian portion of the Amazon basin in 1850 was perhaps 300,000, of whom about two-thirds comprised by Europeans and slaves, the slaves amounting to about 25,000. In Brazil, the principal commercial city, Para (now Belém), had from 10,000 to 12,000 inhabitants, including slaves. The town of Manaus, now Manaus, at the mouth of the Rio Negro, had from 1,000 to 1,500 population. All the remaining villages, as far up as Tabatinga, on the Brazilian frontier of Peru, were relatively small.

# Vamos testar?



# Métodos Mágicos: `__clone()`

O método `__clone()` é chamado quando um objeto é criado com o operador **clone**.



# Vamos testar?





# Métodos Mágicos: `__sleep()`

O método `__sleep()` é chamado quando um objeto é serializado.



# Métodos Mágicos: `__wakeup()`

O método `__wakeup()` é chamado quando é feita uma tentativa (com sucesso) de reconstruir um objeto com dados serializados.





# Vamos testar?



# Padrões PHP



## Framework Interoperability Group

Um grupo de representantes de projetos PHP que falam sobre as semelhanças entre seus projetos e encontram maneiras de trabalhar juntos.

# Alguns membros do PHP-FIG

Agavi 



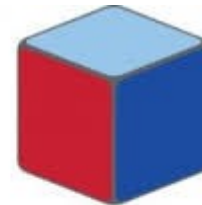
PRESTASHOP



SabreDAV



Symfony



SUGARCRM.



zend  
framework<sub>2</sub>

# PSR 0 - Autoloading Standard


Um *namespace* deve conter em seu primeiro nível o nome do fornecedor da classe e depois a estrutura organizacional das pastas.

Fornecedor\Pasta1\Pasta2\Arquivo.php

```
namespace Fornecedor\Pasta2\Pasta3;  
class Arquivo { }
```

*Underscores* devem ser convertidos em DIRECTORY\_SEPARATOR.

# PSR 1 - Basic Coding Standard

- <?php ou <?=  

- Há dois tipos de arquivos, de **declaração** ou de **ação, nunca os dois ao mesmo tempo.**
- Nomes de classes seguem o padrão **StudlyCaps.**
- Nomes de métodos seguem o padrão **camelCase.**
- Constantes são escritas em letras maiúsculas com *underscores.*

# PSR 2 - Coding Style Guide

- 4 espaços para indentar.
- Linhas devem ter 80 caracteres.
- Abertura de chaves de classes e métodos na linha seguinte.
- Abertura de chaves em estruturas de controle na mesma linha.
- Visibilidade deve ser declarada.
- Deve haver uma linha em branco após a instrução **namespace** e após o conjunto de instruções **use**.

# PSR 3 - Logger Interface

- Classes de log devem seguir a interface `LoggerInterface`
- A exceção `Psr\Log\InvalidArgumentException` deve ser lançada para nível de log desconhecido.

# PSR 4 – Improved Autoloading

O caminho físico para um arquivo contendo uma classe PHP deve ter correspondência direta com o *namespace* da classe.

Pasta1\Pasta2\Pasta3\Arquivo.php

```
namespace Pasta1\Pasta2\Pasta3;  
class Arquivo { }
```



# Exercício

Crie um **cadastro de telefones** que guarde números de telefone e nomes associados a eles. Deve ser possível incluir, alterar, excluir e pesquisar um número ou o nome.

Crie esse cadastro usando **testes**.

# Bibliografia

- **Beck, K.** *TDD Desenvolvimento Guiado por Testes*. Porto Alegre. Bookman, 2010.
- **Bergmann, S.** *PHPUnit*. Disponível em [<https://phpunit.de/getting-started.html>](https://phpunit.de/getting-started.html)
- **Martin, R. C.** *Clean Code: A Handbook of Agile Software Craftsmanship*. Boston. Pearson Education, 2009.
- **McConnell, S.** *Code Complete: Um guia prático para a construção de software*. 2.ed. Porto Alegre. Bookman, 2005.
- **Sommerville, I.** *Engenharia de Software*. 8.ed. São Paulo. Pearson Addison-Wesley, 2007.