**Avaliação Técnica - Analista/Desenvolvedor PHP**

**Boa avaliação!**

**1. (tempo: 10 min)** Considere as classes a seguir:

class Cachorro

{

function latir() { echo "auau!"; }

}

class Vaca

{

function mugir() { echo "muuuuuu!"; }

}

class Galinha

{

function cacarejar() { echo "cócóricó!"; }

}

class Fazenda

{

function emitirSom($animal)

{

// encapsulamento

if (!($animal instanceof Cachorro || $animal instanceof Vaca || $animal instanceof Galinha))

throw new InvalidArgumentException("Fazenda.class: tipo de 'animal' inválido!");

// executando ação

if ($animal instanceof Cachorro) $animal->latir();

if ($animal instanceof Vaca) $animal->mugir();

if ($animal instanceof Galinha) $animal->cacarejar();

}

}

Reescreva as classes acima aplicando o conceito de polimorfismo. É permitido criar novas classes, se julgar necessário.

**R:**

interface Animal

{

function emitirSom();

}

class Cachorro implements Animal

{

function emitirSom() { echo "auau!"; }

}

class Vaca implements Animal

{

function emitirSom() { echo "muuuuuu!"; }

}

class Galinha implements Animal

{

function emitirSom() { echo "cócóricó!"; }

}

class Fazenda

{

function emitirSom(Animal $animal)

{

$animal->emitirSom();

}

}

**2. (tempo: 15 min)** Assembly (Obs: Não usamos Assembly no dia a dia. Está sendo usado aqui apenas para avaliação de algoritmo):

Linguagem de máquina é a linguagem que os processadores entendem. É formada de comandos e operadores em números binários.

Para que um ser humano pudesse programar, foi criado o Assembly. O Assembly é a linguagem mais próxima da linguagem de máquina. Basicamente, o Assembly possui as mesmas funcionalidades que a linguagem de máquina, porém usando comandos human-readables, números decimais e hexadecimais.

Os comandos e operações disponíveis no Assembly são limitadíssimos e variam de acordo com o modelo de processador. Contudo, a diferença é pequena, de modelo para modelo.

Veja a tabela OPERAÇÕES abaixo contendo as operações disponíveis do nosso processador.

No Assembly, todas as operações são feitas tendo como alvo o acumulador (Acc).

Algumas operações podem ter um segundo *operando* envolvido. Nestas, é necessário passar ao comando/operação a posição de memória em que se encontra o *operando* a ser usado, ou mesmo o próprio valor, quando este é estático.

Quando se quer fazer uma operação com o conteúdo de uma variável (posição de memória), é necessário levar este valor para o acumulador, executar a operação e depois colocá-lo de volta na memória.

|  |  |  |
| --- | --- | --- |
| **OPERAÇÕES** | | |
| **Operação** | **Possui**  **Parâmetro** | **Descrição** |
| Ini | - | Inicia o programa |
| Term | - | Termina o programa |
| Load | Sim | Traz o conteúdo de uma posição da memória para o acumulador |
| Put | Sim | Grava o conteúdo do acumulador em uma posição da memória |
| Jnz | Sim | Pula para a linha especificada se o valor no acumulador for diferente de zero |
| Jmp | Sim | Pula para a linha especificada |
| Jip | Sim | Pula para a linha especificada se o valor no acumulador for zero ou acima |
| Inc | - | Incrementa (+1) o valor do acumulador |
| Dec | - | Decrementa (-1) o valor do acumulador |
| Sum | Sim | Soma o valor contido na posição de memória especificada ao valor do acumulador |
| Sub | Sim | Subtrai o valor contido na posição de memória especificada do valor do acumulador |
| Clr | - | Limpa o acumulador, deixando-o com valor 0. |

Nosso computador possui 7 posições de memória: A, B, C, D, E, F, G.

Não esqueça que as posições de memória disponíveis podem ter lixo ao iniciarmos o programa. Jamais execute Load para uma posição de memória que não tenha recebido (Put) um valor anteriormente.

Observe o exemplo abaixo:

**Programa que efetua multiplicações**. Os fatores da multiplicação se encontram nas posições de memória A e B, e o resultado deve estar na posição C ao final da execução do programa.

|  |  |  |  |
| --- | --- | --- | --- |
| **Linha** | **Operação** | **Parâmetro** | **Comentário** |
| 01 | Ini |  | Iniciando o programa |
| 02 | Clr |  | Zerando o acumulador |
| 03 | Put | C | Limpando a memória C |
| 04 | Load | C | Colocando o conteúdo da memória C no acumulador |
| 05 | Sum | A | Somando o valor de A ao acumulador |
| 06 | Put | C | Colocando o conteúdo do acumulador na memória C |
| 07 | Load | B | Colocando o conteúdo da memória B no acumulador |
| 08 | Dec |  | Decrementando o valor do acumulador |
| 09 | Put | B | Colocando o conteúdo do acumulador na memória B |
| 10 | Jnz | 04 | Se o valor do acumulador não for 0, volta à linha 04 |
| 11 | Term |  | Finaliza o programa |

Observe que um dos fatores foi usado como um contador de decremento.

Para o nosso programa, não importa se os valores originais dos fatores (A e B) foram alterados. Importa apenas que, ao final do programa, o resultado seja encontrado na posição de memória designada (posição C, neste caso).

**Questão:** Construa um programa que execute a divisão. O dividendo encontra-se na posição A e o divisor na B. Ao final do programa, o resultado deve estar na posição C e o resto na posição D. As posições de memória restantes podem ser usadas livremente para a execução das operações, caso necessário.

|  |  |  |
| --- | --- | --- |
| **Linha** | **Operação** | **Parâmetro** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**R:**

=/

**3. (tempo: 5 min)** Qual a diferença de comportamento entre um LEFT JOIN e um INNER JOIN?

**R:**

INNER JOIN => retorna somente os registros repetidos entre duas tabelas.

LEFT JOIN => retorna todos os registros contigos na tabela da esquerda mais os registros repetidos entre as duas tabelas.

**4. (tempo: 10 min)** Descreva, com suas palavras, o que, no framework Laravel, é o Eloquent, qual é a sua vantagem em relação aos contextos onde não há uso do Eloquent e, mostre um exemplo de código que fundamente a sua resposta.

**R:**

É um ORM nativo do Laravel para facilitar a manipulação do banco de dados seja ela simples ou complexas.

Uma das vantagens do Eloquent é a simplificação de código.

Exemplo de Model:

<?php

namespace Models;

use \Illuminate\Database\Eloquent\Model;

class Usuarios extends Model {

protected $table = ‘usuarios’;

}

?>

**5. (tempo: 5 min)** Explique com suas palavras o que é, no Laravel, o Blade, e demonstre sua utilização em um trecho de código.

**R:**

É um mecanismo que permite escrever views mais semânticas usando uma sintaxe própria e mais simples, ou seja, não se utiliza a sintaxe PHP.

<body>

<header>

<h1>Teste Life</h1>

</header>

<div class"">

@yield('flamengo')

</div>

<div class="">

@yield('vasco')

</div>

</body>

@extends('clubes')

@div('flamengo')

<h2>O melhor do Mundo</h2>

@stop

@div('vasco')

<h6>Rebaixado</h6>

@stop

**6. (tempo 5 min)** Qual(is) comando(s) Artisan você usaria para criar uma Model, um Controller e uma Migration?

**R:**

php artisan make:model NomeDoModelo -c -m -r

**7. (tempo 5 min)** Considerando um repositório GIT local, ligado à um repositório remoto. Você precisa criar uma nova branch chamada "card#1000" para realizar uma demanda e ela precisa ser baseada na branch "dev". Após realizar as alterações de código na branch "card#1000", você deve mesclá-la (merge) com a branch de "teste". Sabendo que, ao final, todas as branchs deverão estar no repositório remoto, demostre a sequencia de comandos GIT necessários para realizar esse fluxo.

**R:**

$git branch

$git branch -a

$git checkout my-branch

$git branch -d name-branch

$git branch -D name-branch

**8. (tempo: 15 min)** Implemente um crud no Laravel para uma classe Pessoa (campos nome, cpf, email, data\_nasc e nacionalidade), demonstrando o Controller, a Model e as rotas envolvidas. Para persistir os dados, utilize o padrão Mass Assignment. Não é necessário criar a view. Utilize, se julgar necessário, o FormRequest ao invés do Request. Valide os campos que julgar necessário (usando a técnica de sua preferência).

**R:**