Padrão usado: Singleton

Descrição: Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.

Onde foi usado: Este padrão foi utilizado na classe conexão, onde somente a própria classe pode chamar o construtor.

Código

```php
<?php
//design pattern Singleton
class conexao
{
    // Instância da classe
    private static $instance = null;
    private $conn;

    // Construtor privado: só a própria classe pode invocá-lo
    private function __construct()
    {
        $host = "localhost";
        $user = "root";
        $pswd = "";
        $db = "projeto";

        try {
            $this->conn = mysqli_connect($host, $user, $pswd, $db);
            $this->conn->set_charset('utf8');
        } catch (Exception $e) {
            die("Erro na conexão com MySQL! " . $e->getMessage());
        }
    }

    // método estático
    static function getInstance()
    {
        // Já existe uma instância?
        if (self::$instance == NULL)
            self::$instance = new conexao();    // Não existe, cria a instância
        return self::$instance;                 // Já existe, simplesmente retorna
    }

    // Previne o uso de __clone
    private function __clone() {}
     public function getCon(){
        return $this->conn;
    }
}
```

Padrão usado: Strategy

Descrição: O padrão Strategy serve para “definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam” (como definido no livro do GoF).

Onde foi usado: No projeto foi usado este padrão no arquivo permissões, onde varias classes implementam a interface, com o mesmo método, mas com ações diferentes.

Código:

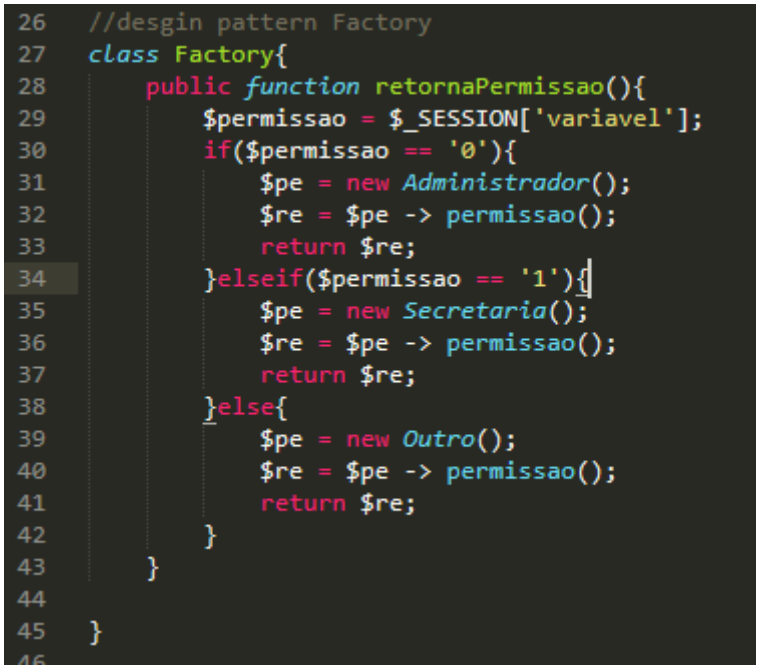
```php
<?php
session_start();
//desgin pattern Strategy
interface tipoPermissao{
    public function permissao();
}

class Administrador implements tipoPermissao{
    public function permissao(){
        $_SESSION['variavel'] = 'Administrador';
        return 'Administrador';
    }       //Sera implementado o que o este usuario tem permissao
}
class Secretaria implements tipoPermissao{
    public function permissao(){
        $_SESSION['variavel'] = 'Secretaria';
        return 'Secretaria';
    }
}
class Outro implements tipoPermissao{
    public function permissao(){
        $_SESSION['variavel'] = 'Outro';
        return 'Outro';
    }
}
```

Padrão usado: Factory

Descrição: Este método funciona como uma fabrica de instancias, onde uma classe busca varias outras classes.

Onde foi usado: No projeto foi utilizado no arquivo permissões, e ele possui uma função que busca qual classe deverá ser implementada.

Código:



```php
//desgin pattern Factory
class Factory{
    public function retornaPermissao(){
        $permissao = $_SESSION['variavel'];
        if($permissao == '0'){
            $pe = new Administrador();
            $re = $pe -> permissao();
            return $re;
        }elseif($permissao == '1'){
            $pe = new Secretaria();
            $re = $pe -> permissao();
            return $re;
        }else{
            $pe = new Outro();
            $re = $pe -> permissao();
            return $re;
        }
    }

}
```