

Caros alunos,

Este tutorial não substitui o material oficial de estudo da disciplina, com ele, temos o objetivo de proporcionar uma abordagem mais simplificada e direta dos conceitos iniciais de Programação Orientada a Objetos com a linguagem de programação PHP:

Sumário

1. Básico	2
1.1. Criar uma classe	2
1.2. Declarar seus atributos	2
1.3. Instanciar a classe.....	3
1.4. Métodos assessores	6
1.5. Métodos construtor	8
1.6. Método Construtor com Valor Padrão.....	9
2. Intermediário	11
2.1. Interface	11
2.2. Herança	13
3. PHP e Banco de dados.....	16
3.1. Conexão.....	16
3.2. Query.....	16
3.3. Inserir Registro	17
3.4. Listar Registro sem parâmetro.....	21
3.5. Listar Registro com parâmetro.....	21
3.6. Alterar Registro	22
3.7. Excluir Registro	26

Para configurarmos o nosso ambiente de trabalho, iremos criar duas pastas;

1 – Na pasta htdocs ou www ou html (dependendo do seu servidor web), crie uma pasta com o nome de revisão.

2 - Dentro desta pasta crie uma nova pasta com o nome de classes.

Depois de criado o ambiente; as pastas, lembrem-se de inicializar o servidor WEB. O editor pode ser de sua escolha. Para os testes utilizei o servidor Web XAMPP, e o editor NotePad++

1. BÁSICO

Este capítulo, tem por objetivo, introduzir os conceitos iniciais de POO, com a criação de uma classe, a declaração de atributos com suas respectivas visibilidades e a instancia de um objeto. Ainda neste capítulo, foram abordados os métodos assessores e o construtor com e sem parâmetro.

1.1. Criar uma classe

Abra um arquivo novo e salve-o com o nome de Pessoa.php na pasta classes, que está dentro da classe revisão.

Para declararmos a classe iremos utilizar a palavra class seguido do nome da classe. As boas práticas de programação dizem que o nome da classe deve iniciar com caixa alta. Confira abaixo a classe Pessoa declarada.

```
<?php
    class Pessoa{

    }
?>
```

1.2. Declarar seus atributos

Para declararmos as variáveis em orientação a objetos segue da mesma forma, contudo, em orientação a objetos as variáveis são chamadas de atributos. Os atributos possuem visibilidade, que os protegem, sendo eles:

Public: Este é o nível de acesso mais permissivo. Ele indica que o método ou atributo da classe é público, ou seja, pode ser acessado em qualquer outro ponto do código e por outras classes.

Private: Este modificador é o mais restrito. Com ele definimos que somente a própria classe em que um atributo ou método foi declarado pode acessá-lo. Ou seja, nenhuma outra parte do código, nem mesmo as classes filhas, pode acessar esse atributo ou método.

Protected: Esse modificador indica que somente a própria classe e as classes que herdam dela podem acessar o atributo ou método. Dessa forma, ao instanciar a classe os elementos protegidos (*protected*) não podem ser acessados diretamente, como ocorre com o *public*.

Vamos dar continuidade na nossa classe Pessoa, declarando os atributos nome, email e telefone com a visibilidade public. Para isso, dentro da classe Pessoa, colocaremos a visibilidade public seguido dos nomes dos atributos. Lembro que os atributos podem ser declarados na mesma linha desde que todos sejam da mesma visibilidade. Caso a visibilidade seja diferente, será necessário declarar em outra linha.

```
<?php
    class Pessoa{
        public $nome, $email, $telefone;
    }
?>
```

1.3. Instanciar a classe

Após declarar a classe, devemos instancia-la em um objeto no qual daremos o nome de funcionario, para isso iremos utilizar o operador new seguido do nome da classe, confira abaixo.

```
<?php
    class Pessoa{
        public $nome, $email, $telefone;
    }
    $funcionario = new Pessoa();
    var_dump($funcionario);
?>
```

Após executar o código acima, teremos o seguinte retorno da função `var_dump()`.

```
object(Pessoa)#1 (3) { ["nome"]=> NULL ["email"]=> NULL
["telefone"]=> NULL }
```

O objeto é da classe `Pessoa`, e possui 3 atributos (nome, email e telefone), como não povoamos o nosso objeto, o mesmo está com valor `NULL`.

As boas práticas de programação, sugerem que na classe tenhamos apenas a classe, seus atributos e métodos. Sendo assim, neste arquivo deixe apenas a classe, e vamos criar um novo arquivo que será responsável por instanciar e alimentar a nossa classe.

Crie um novo arquivo e salve-o com o nome de `controle.php` na pasta revisão. Dentro deste arquivo, devemos primeiramente incluir a classe para depois instanciar-la, segue como ficou o arquivo `controle.php`.

```
<?php
    include 'classes/Pessoa.php';
    $funcionario = new Pessoa();
    var_dump($funcionario);
?>
```

Se você testar novamente, chegará no mesmo resultado anterior. Caso apresente algum erro, repita todos os passos novamente.

Agora, vamos alimentar o nosso objeto dentro do arquivo `controle.php`. Para isso, iremos utilizar o objeto, pois como já visualizado nos retornos acima, ele possui 3 atributos (nome, email e telefone) de forma que ele aponte para o atributo da classe e depois atribuir um valor para o atributo.

```
$funcionario->nome = "Rafael Alves Florindo";
```

Como o valor atribuído ao atributo é uma string, utilizamos aspas.

Confira o código abaixo:

```
<?php
    include 'classes/Pessoa.php';
    $funcionario = new Pessoa();
    $funcionario->nome = "Rafael Alves Florindo";
    $funcionario->email = "rafael.florindo@unicesumar.edu.br";
    $funcionario->telefone = "44 3027-6360";
    var_dump($funcionario);
?>
```

Após executar o código teremos a seguinte saída em tela.

```
object(Pessoa)#1 (3) { ["nome"]=> string(21) "Rafael Alves
Florindo" ["email"]=> string(33)
"rafael.florindo@unicesumar.edu.br" ["telefone"]=> string(12)
"44 3027-6360" }
```

Bom, utilizar a função `var_dump()` para saída de tela, não é interessante, sendo assim, da mesma forma que utilizamos para povoar o nosso objeto, iremos realizar para imprimir os valores dos atributos, confira:

```
echo "Nome: " . $funcionario->nome;
```

Confira o código abaixo:

```
<?php
    include 'classes/Pessoa.php';
    $funcionario = new Pessoa();

    $funcionario->nome = "Rafael Alves Florindo";
    $funcionario->email = "rafael.florindo@unicesumar.edu.br";
    $funcionario->telefone = "44 3027-6360";

    echo "Nome: " . $funcionario->nome;
    echo "<br />E-mail: " . $funcionario->email;
    echo "<br />Telefone: " . $funcionario->telefone;
?>
```

Após executar o código `controle.php`, teremos a seguinte saída em tela.

```
Nome: Rafael Alves Florindo
E-mail: rafael.florindo@unicesumar.edu.br
Telefone: 44 3027-6360
```

A estilização das saídas de tela, fica com sua conta.

1.4. Métodos assessores

Até agora, utilizamos os atributos da classe com a visibilidade pública, por este motivo foi possível acessar os seus atributos de forma direta. Acessar os atributos desta forma, não é a mais recomendada, pois as boas práticas de programação sugerem a utilização dos métodos assessores (getters e setters) que tem a função de setar (armazenar) e resgatar (retornar) os valores dos atributos.

O método setter, serve para que possamos povoar o atributo da classe, neste método também poderá ser utilizado para realizar validações antes de setar o atributo. Por padrão este método dever ter visibilidade pública para que o mesmo possa ser acessado por outro método ou objeto externo. O método get, segue as mesmas diretrizes, contudo, não deve receber parâmetros, e deve retornar o atributo. Vejamos um exemplo:

```
<?php
.....
    public function setNome($nome){
        $this->nome = $nome;
    }
    public function getNome(){
        return $this->nome;
    }
...?>
```

Notem que o método é idêntico a uma função, a diferença é a sua visibilidade. O nome da função deve seguir a sintaxe de iniciar por set seguido do nome do atributo, ficando setNome() ... getNome(). Notem que o método setNome(), recebe por parâmetro a variável nome, setNome (\$nome), enquanto que o método getNome(), não recebe parâmetros, o mesmo retorna o atributo solicitado. O ponteiro \$this, apontará para o atributo no qual está sendo atribuído o valor da variável que recebeu, ou do atributo no qual será retornado respectivamente.

Vamos adaptar a nossa classe `Pessoa.php` e deixar os atributos: nome e e-mail como public, e o atributo telefone como private. Além de modificar a visibilidade

dos atributos, vamos implementar os métodos assessores de todos os atributos, mesmo os atributos que estão com visibilidade publicas devem ser implementados.

```
<?php
class Pessoa{
    public $nome, $email;
    private $telefone;

    public function setNome($nome){
        $this->nome = $nome;
    }
    public function getNome(){
        return $this->nome;
    }
    public function setEmail($email){
        $this->email = $email;
    }
    public function getEmail(){
        return $this->email;
    }
    public function setTelefone($telefone){
        $this->telefone = $telefone;
    }
    public function getTelefone(){
        return $this->telefone;
    }
}
?>
```

A ordem da implementação dos métodos dentro da classe não influencia. Agora, com a classe implementada, devemos ir ao nosso arquivo `controle.php`, e realizar as modificações necessárias, pois se tentar rodar o arquivo, dará erro (**Fatal error: Uncaught Error: Cannot access private property Pessoa::\$telefone**), devido ao atributo `telefone` ser privado, e você não pode chama-lo mais. Isto acontece, pois para ter acesso a ele deve passar pelo método público `setTelefone()` e `getTelefone()`.

```
<?php
include 'classes/Pessoa.php';
$funcionario = new Pessoa();
$funcionario->setNome("Rafael Alves Florindo");
$funcionario->setEmail("rafael.florindo@unicesumar.edu.br");
$funcionario->setTelefone("44 3027-6360");
//var_dump($funcionario);
```

```
echo "Nome: " . $funcionario->getNome();  
echo "<br />E-mail: " . $funcionario->getEmail();  
echo "<br />Telefone: " . $funcionario->getTelefone();  
?>
```

Notem que ao invés do objeto apontar direto o atributo da classe, `$funcionario->nome = "Rafael Alves Florindo"`, estamos acessando ele através do método, `$funcionario->setNome("Rafael Alves Florindo")`. Neste caso o objeto está apontando para o método `setNome()` e passando por parâmetro uma string `("Rafael Alves Florindo")`.

O mesmo ocorre com a impressão, deixamos de acessar diretamente o atributo da classe, `echo "Nome: " . $funcionario->nome`, estamos acessando ele através do método `echo "Nome: " . $funcionario->getNome()`. Neste caso, não passamos o parâmetro.

O método `var_dump()` poderá lhe mostrar em detalhes a visibilidade do atributo, caso seja público será omitido a visibilidade.

```
object(Pessoa)#1 (3) { ["nome"]=> string(21) "Rafael Alves  
Florindo" ["email"]=> string(33)  
"rafael.florindo@unicesumar.edu.br"  
["telefone":"Pessoa":private]=> string(12) "44 3027-6360" }
```

1.5. Métodos construtor

Podemos ainda, implementar o recuso de método construtor. Este método será executado mesmo se não implementado. Para implementar este método basta colocar dois underline seguido da palavra `construct`.

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

O objetivo deste método, é inicializar a classe com alguns valores.

1.6. Método Construtor com Valor Padrão

Para exemplificarmos, vamos duplicar o conteúdo da classe Pessoa para a classe Aluno, para isso, duplique a classe `Pessoa.php` com o nome de `Aluno.php` dentro da pasta `classes`. Feito isso, logo após a declaração dos atributos, implemente o método construtor com os parâmetros (nome, email e telefone) setando estes valores nos atributos da classe. Os métodos setters, iremos deixar no código, pois podemos setar outros valores se necessário. Confira abaixo:

```
<?php
class Aluno{
    public $nome, $email;
    private $telefone;

    public function __construct($nome, $email, $telefone){
        $this->nome = $nome;
        $this->email = $email;
        $this->telefone = $telefone;
    }
    public function setNome($nome){
        $this->nome = $nome;
    }
    public function getNome(){
        return $this->nome;
    }
    public function setEmail($email){
        $this->email = $email;
    }
    public function getEmail(){
        return $this->email;
    }
    public function setTelefone($telefone){
        $this->telefone = $telefone;
    }
    public function getTelefone(){
        return $this->telefone;
    }
}
?>
```

Agora, iremos instanciar a nossa classe em um novo arquivo na pasta `revisão` com o nome de `controleAluno.php`, com objeto chamado `$aluno`. Para que seja possível o construtor receber os parâmetros, criamos 3 variáveis, e atribuímos a cada

uma delas, uma string, depois passamos as variáveis para o método construtor por parâmetro de valor. Confira como ficou o nosso arquivo abaixo:

```
<?php
    include 'classes/Aluno.php';
    $nome = "Ricardo Alves Florindo";
    $email= "xxxx@gmail.com";
    $telefone = "44 3027-6360";
    $aluno = new Aluno($nome, $email, $telefone);
    var_dump($aluno);
?>
```

A saída será:

```
object(Aluno)#1 (3) { ["nome"]=> string(22) "Ricardo Alves Florindo" ["email"]=> string(26) "ricardo.florindo@gmail.com" ["telefone":"Aluno":private]=> string(12) "44 3027-6360" }
```

Notem, que como deixamos os métodos assessores podemos chama-los novamente, para que seja possível trocar os valores contidos nos atributos da classe.

Confira abaixo:

```
<?php
    include 'classes/Aluno.php';
    $nome = "Ricardo Alves Florindo";
    $email= "ricardo.florindo@gmail.com";
    $telefone = "44 3027-6360";
    $aluno = new Aluno($nome, $email, $telefone);
    var_dump($aluno);

    $aluno->setNome("Jose da Silva");
    $aluno->setEmail("yyyyy@unicesumar.edu.br");
    $aluno->setTelefone("0800-xxxxxxxxxx");
    var_dump($aluno);
?>
```

Ao executar novamente, poderá perceber que os valores do objeto foram trocados, ou seja substituído.

```
object(Aluno)#1 (3) {
    ["nome"]=>
    string(22) "Ricardo Alves Florindo"
    ["email"]=>
    string(26) "ricardo.florindo@gmail.com"
    ["telefone":"Aluno":private]=>
```

```
        string(12) "44 3027-6360"
    }
    object(Aluno)#1 (3) {
        ["nome"]=>
        string(13) "Jose da Silva"
        ["email"]=>
        string(23) "yyyyy@unicesumar.edu.br"
        ["telefone":"Aluno":private]=>
        string(15) "0800-xxxxxxxxxx"
    }
```

2. INTERMEDIÁRIO

Este capítulo, tem por objetivo, avançar um pouco o conteúdo de orientação a objetos. Neste iremos trabalhar e aplicar os conceitos de interface (mais conhecido como contrato), herança, classe abstrata e final. Lembro que caso venha surgir dúvidas, verificar o material oficial da sua disciplina.

2.1. Interface

A interface funciona como um contrato entre os analistas e os desenvolvedores, desta forma ao desenvolver uma interface, você obriga o desenvolvedor a implementar os métodos declarados.

Para continuarmos crie um novo arquivo dentro da pasta classes e salve com o nome de `interfacePessoa.php`. Neste arquivo foi declarado uma interface com o nome de `pessoa`, e declarado dois métodos, `cadastrar` e `imprimir`. No método `cadastrar` ele recebe três parâmetros, enquanto o `imprimir` não recebe parâmetro.

```
<?php
    interface Pessoa{
        public function cadastrar($nome, $email, $fone);
        public function imprimir();
    }
?>
```

Agora com a interface criada, vamos criar uma classe chamada de `funcionario.php` dentro da pasta classes, que implementará a

interfacePessoa.php. Nesta classe, precisamos primeiramente incluir a interface, e ao declarar a classe, colocar a palavra *implements* seguido do nome da interface.

```
<?php

    class Funcionario implements Pessoa{}

?>
```

Notem que na interface abaixo, temos dois métodos cadastrar e imprimir, sendo assim precisamos implementá-los na classe, caso não implemente, ou a assinatura do método é diferente, dará erro na tela **"Class Funcionario contains 2 abstract methods and must therefore be declared abstract or implement the remaining methods (Pessoa::cadastrar, Pessoa::imprimir)"**.

Abaixo, criamos uma classe `Funcionario` (`funcionario.php`) dentro da pasta classes, que implementa a `interfacePessoa`, e nesta foi declarado 3 atributos e seus respectivos métodos assessores. Os métodos da interface podem ser implementados em qualquer local dentro da classe, confira o código abaixo.

```
<?php
    include 'interfacePessoa.php';

    class Funcionario implements Pessoa{
        public $nome, $email;
        private $telefone;

        public function setNome($nome){
            $this->nome = $nome;
        }
        public function getNome(){
            return $this->nome;
        }
        public function setEmail($email){
            $this->email = $email;
        }
        public function getEmail(){
            return $this->email;
        }
        public function setTelefone($telefone){
            $this->telefone = $telefone;
        }
    }
```

```
public function getTelefone(){
    return $this->telefone;
}
public function cadastrar($nome, $email, $fone){
    $this->nome = $nome;
    $this->email = $email;
    $this->telefone = $telefone;
}
public function imprimir(){
    return array("nome"=>$this->getNome(),
"email"=>$this->getEmail(), "telefone"=>$this->getTelefone());
}
}
?>
```

Como você pode ter percebido, ao implementar uma interface, você começa a criar padrões de programação dentro de sua equipe. A interface não pode ser instanciada, apenas implementada, desta forma, vamos criar um código `controleFuncionario.php` dentro da pasta `revisão`, no qual instanciaremos a classe `funcionário`, segue o código abaixo (o código dispensa explicação, uma vez, que os conceitos já foram trabalhados).

```
<?php
include 'classes/Funcionario.php';
$nome = "Ricardo Alves Florindo";
$email= "ricardo.florindo@gmail.com";
$telefone = "44 3027-6360";
$func = new Funcionario();
$func -> cadastrar($nome, $email, $telefone);
$func -> imprimir();
var_dump($func);
?>
```

2.2. Herança

Quando trabalhamos com herança, dizemos que estamos herdando algo de alguém, ou seja, estamos herdando todas as características de uma outra classe. Para exemplificar, vamos criar uma classe chamada de `Pessoa` e nesta vamos criar os atributos `nome`, `email` e `telefone`, bem como os métodos `assessores`, e dois métodos `cadastrar` e `imprimir`, sendo que o `cadastrar` recebe por parâmetro os valores

e seta os atributos enquanto o método imprimir realiza a impressão dos dados da tela. Confira abaixo.

```
<?php
class Pessoa{
    public $nome, $email;
    private $telefone;

    public function setNome($nome){
        $this->nome = $nome;
    }
    public function getNome(){
        return $this->nome;
    }
    public function setEmail($email){
        $this->email = $email;
    }
    public function getEmail(){
        return $this->email;
    }
    public function setTelefone($telefone){
        $this->telefone = $telefone;
    }
    public function getTelefone(){
        return $this->telefone;
    }
    public function cadastrar($nome, $email, $telefone){
        $this->setNome($nome);
        $this->setEmail($email);
        $this->setTelefone($telefone);
    }
    public function imprimir(){
        echo "<br />Nome = " . $this->getNome();
        echo "<br />Email = " . $this->getEmail();
        echo "<br />Telefone = " . $this->getTelefone();
    }
}
?>
```

Agora, vamos criar uma classe chamada de Cliente e nesta vamos criar o atributo dataNascimento, bem como os métodos assessores, e dois métodos cadastrarCliente e imprimirCliente, sendo que o cadastrar recebe por parâmetro os valores passa para o método cadastrar da classe pai e seta e seta o atributo dataNascimento. Enquanto o método imprimirCliente chama o método da classe pai

que imprime por lá os dados herdados e por aqui realiza a impressão da dataNascimento. Confira abaixo.

```
<?php
    include 'Pessoa.php';

    class Cliente extends Pessoa{
        public $dataAniversario;
        public function setDataAniversario($dataAniversario){
            $this->dataAniversario = $dataAniversario;
        }
        public function getDataAniversario(){
            return $this->dataAniversario;
        }

        public function cadastrarCliente($nome, $email,
$telefone, $dataAniversario){
            parent::cadastrar($nome, $email, $telefone);
            $this->setDataAniversario($dataAniversario);
        }
        public function imprimirCliente(){
            parent::imprimir();
            echo "<br />Data Aniversario = " . $this-
>getDataAniversario();
        }
    }
?>
```

Agora, que já implementamos a nossa herança, vamos instanciar a classe Cliente, chamar o método cadastrarCliente e passar por parametros os seus valores, e depois chamar o método imprimirCliente. Confira abaixo:

```
<?php
    include 'classes/Cliente.php';
    $nome = "Rafael Alves Florindo";
    $email= "rafael.florindo@gmail.com";
    $dataAniversario = "2018-05-05";
    $telefone = "44 3027-6360";
    $clie = new Cliente();
    $clie -> cadastrarCliente($nome, $email, $telefone,
$dataAniversario);
    $clie -> imprimirCliente();
?>
```

3. PHP E BANCO DE DADOS

Iremos trabalhar com a API do Banco Mysql orientada a objeto, ou seja, utilizando a classe Mysqli().

3.1. Conexão

Para que a conexão com o seu banco de dados funcione, será necessário utilizar uma classe que já está pronta no mysqli, contudo ela espera receber alguns parâmetros em seu construtor, sendo eles: local, usuário, senha e a base de dados, vejamos o exemplo abaixo:

```
<?php
    $local = "localhost";
    $user = "root";
    $password = "";
    $database = "revisao";

    $conectar = new mysqli($local, $user, "", $database);

    if ($conectar->connect_errno){
        echo "Houve um erro na tentativa de conexão com MySQL:
            (" . $conectar->connect_errno . ") " . $conectar-
>connect_error;
    }
?>
```

3.2. Query

O método query é responsável por executar as consultas ao banco de dados, sendo assim. Para que o método funcione, devemos incluir a conexão e utilizar o objeto criado que chamará o método query passando para ele por parâmetro uma variável que contém uma string ou a própria string envolvida por "".

```
include 'conexao.php';
$insertir = $conectar-> query($sql);
```


3.3. Inserir Registro

Para continuar, vamos criar uma base de dados com o nome de revisão, uma tabela de produtos e um formulário, conforme imagens abaixo:

Figura: Tabela


#	Nome	Tipo
1	id 	int(11)
2	nome	varchar(50)
3	descricao	varchar(255)
4	quantidade	int(11)
5	precoCompra	decimal(10,2)
6	precoVenda	decimal(10,2)
7	datacadastro	datetime
8	dataAtualizacao	timestamp
9	ativo	tinyint(4)

Figura: Formulário

Nome

Descrição

Quantidade

Preço de compra

Preço de venda

Após este conteúdo, devemos criar um arquivo em php que recebe estes dados via comando `filter_input(INPUT_POST, "")`, que realiza um filtro na variável que está sendo recebida. Incluiremos a classe e instanciaremos o objeto. Feito isso, devemos chamar o método de cadastro e passar para ele os parâmetros a serem cadastrados `$cadastrarUsuario -> cadastrar($nome, $descricao,`

\$quantidade, \$precoCompra, \$precoVenda). Após os dados serem cadastrados ou não, é realizada uma validação a fim de imprimir na tela de foi realizada com sucesso ou apresentou erro na execução. Este arquivo vamos chamar de controleProduto.php.

```
<?php
    include 'classes/Produto.php';

    $cadastrarProduto = new Produto();

    $nome = filter_input(INPUT_POST, "nome");
    $descricao = filter_input(INPUT_POST, "descricao");
    $quantidade = filter_input(INPUT_POST, "quantidade");
    $precoCompra = filter_input(INPUT_POST, "precoCompra");
    $precoVenda = filter_input(INPUT_POST, "precoVenda");
    $cadastrarUsuario -> cadastrar($nome, $descricao,
    $quantidade, $precoCompra, $precoVenda);

    if ($cadastrarProduto){
        echo "Gravado com sucesso!!!";
    }else{
        echo "Erro ao gravar as informações";
    }
?>
```

Depois vamos para a classe produto.php, nela declaramos os atributos e seus métodos assessores.

```
<?php
class Produto{
    private $id, $nome, $descricao, $quantidade,
    $precoCompra, $precoVenda, $datacadastro,
    $dataAtualizacao, $ativo;

    public function setNome($nome){
        $this->nome = $nome;
    }
    public function setdescricao($descricao){
        $this->descricao = $descricao;
    }
    public function setQuantidade($quantidade){
        $this->quantidade = $quantidade;
    }
    public function setPrecoCompra($precoCompra){
        $this->precoCompra = $precoCompra;
    }
}
```

```
}
public function setPrecoVenda($precoVenda) {
    $this->precoVenda = $precoVenda;
}
public function setDataCadastro($dataCadastro) {
    $this->dataCadastro = $dataCadastro;
}
public function setDataAtualizacao($dataAtualizacao) {
    $this->dataAtualizacao = $dataAtualizacao;
}
public function setAtivo($ativo) {
    $this->ativo = $ativo;
}

public function getNome() {
    return $this->nome;
}
public function getDescricao() {
    return $this->descricao;
}
public function getQuantidade() {
    return $this->quantidade;
}
public function getPrecoCompra() {
    return $this->precoCompra;
}
public function getPrecoVenda() {
    return $this->precoVenda;
}
public function getDataCadastro() {
    return $this->dataCadastro;
}
public function getDataAtualizacao() {
    return $this->dataAtualizacao;
}
public function getAtivo() {
    return $this->ativo;
}
//...
?>
```

O método de cadastrar recebe os dados, seta os valores nos atributos `$this->setNome($nome)`, e posteriormente monta a query de inserção resgatando os valores dos atributos `$this->getNome()` e executa a consulta no banco de dados.

```
<?php
//..continuando a classe produto
public function cadastrar($nome, $descricao, $quantidade,
$precoCompra, $precoVenda){
    $this->setNome($nome);
    $this->setdescricao($descricao);
    $this->setQuantidade($quantidade);
    $this->setPrecoCompra($precoCompra);
    $this->setPrecoVenda($precoVenda);
    $this->setDataCadastro(date("Y-m-d h:m:s"));
    $this->setAtivo(1);

    $sql = "insert into produto
        (nome, descricao, quantidade, precoCompra,
precoVenda, datacadastro, ativo)
        values
        (
            '{$this->getNome()}',
            '{$this->getDescricao()}',
            '{$this->getQuantidade()}',
            '{$this->getPrecoCompra()}',
            '{$this->getPrecoVenda()}',
            '{$this->getDataCadastro()}',
            '{$this->getativo()}'
        )";

    include 'conexao.php';
    $insserir = $conectar-> query($sql);

    $registroAfetados = $conectar->affected_rows;
    if ($registroAfetados == 1){
        return 1;
    }else{
        return 0;
    }

}
}
?>
```

No método cadastrar, após executar a query, ele realiza uma verificação de registros afetados `$registroAfetados = $conectar->affected_rows;` com a consulta ao banco, se o número de linhas afetadas for igual a 1, retorna verdadeiro para quem a chamou, neste caso o arquivo cadastroProduto.php, que irá realizar uma nova verificação e imprime na tela se foi gravado com sucesso.

3.4. Listar Registro sem parâmetro

Um recurso muito utilizado é a seleção dos registros de uma tabela do banco de dados. Abaixo apresento um código que inclui a classe produto, instancia ela, e chama o método `listarProduto()`.

```
<?php
    include 'classes/Produto.php';
    $selecionarProduto = new Produto();
    $retorno = $selecionarProduto->listarProduto();
    var_dump($retorno);
?>
```

Após ser chamado o método na classe, é montado a query de consulta `$sql = "select * from produto";`, depois incluída a conexão e realizado a query `$listar = $conectar-> query($sql);` com o banco de dados. Depois é criado um array dinâmico `$dados = array();` que armazenará os registros/tuplas/linhas da tabela. Após o retorno do banco, temos que associar este retorno no nosso array, neste caso utilizamos a estrutura de repetição WHILE `while($linha = $listar->fetch_assoc()){ $dados[] = $linha; }`, que irá repetir até que a última linha seja associada no vetor.

```
<?php
//..continuando a classe produto
public function listarProduto(){
    $sql = "select * from produto";
    include 'conexao.php';
    $listar = $conectar-> query($sql);

    $dados = array();
    while($linha = $listar->fetch_assoc()){
        $dados[] = $linha;
    }
    return $dados;
}
```

3.5. Listar Registro com parâmetro

Um outro recurso muito utilizado na seleção dos registros de uma tabela do banco de dados é a passagem de parâmetros. Abaixo apresento um código que inclui

e instancia a classe produto. A mesma recebe por parâmetro \$idProduto e chama o método pesquisaProduto(), passando este parâmetro, pesquisaProduto(\$idProduto).

```
<?php
    include 'classes/Produto.php';

    $selecionarProduto = new Produto();
    $idProduto = filter_input(INPUT_GET, "idProduto")); {
    $retornoPesquisa = $selecionarProduto-
>pesquisaProduto($idProduto);
    var_dump($retornoPesquisa);
?>
```

Na classe, o método recebe o valor, seta no atributo pelo método \$this->setIdProduto(\$idProduto), posteriormente monta a query de consulta ao banco e executa. Como neste caso retornaremos apenas um registro, não foi necessário utilizar um laço de repetição para associar ele a um array.

```
<?php
//..continuando a classe produto

public function pesquisaProduto($idProduto){
    $this->setIdProduto($idProduto);
    $sql = "select * from produto where id = '{$this-
>getIdProduto()}'";
    include 'conexao.php';
    $listar = $conectar->query($sql);
    $registros = $listar->num_rows;

    if($registros == 1){
        $dados_usuario = $listar->fetch_assoc();
        return $dados_usuario;
    }else{
        return $dados_usuario;
    }
}
```

3.6. Alterar Registro

Este processo, é o mais complexo de todo, devido ter a necessidade de uma listagem de registros, e um formulário com os valores preenchidos pelo retorno do

banco de dados. Para este processo crie um arquivo e salve-o com o nome de `formAlterar.php`. Neste logo no início, iremos incluir e instanciar a classe `Produto`, feito isso iremos chamar o método `pesquisaProduto()`, passando por parâmetro o `idProduto` recebido, no qual retornará os dados do produto selecionado na lista que está no final do código. Como o retorno será um vetor, se o mesmo não retornar nada, criamos um array com as suas posições nulas. Se tiver valor neste vetor, o mesmo será impresso no elemento “value” correspondente no formulário.

Notem que no final do formulário foi necessário o uso de um campo oculto, que tem por objetivo passar um campo de forma oculta, `<input type="hidden" name="idProduto" value = "<?php echo $idProduto; ?>">`, neste estamos passando o `idProduto`. Vejamos o código do formulário seguido da continuação da classe.

```
<?php
    include 'classes/Produto.php';

    $selecionarProduto = new Produto();
    if($idProduto = filter_input(INPUT_GET, "idProduto")) {
        $retornoPesquisa = $selecionarProduto-
>pesquisaProduto($idProduto);
    }else{
        $retornoPesquisa =
        array(
            "nome" => " ",
            "descricao"=>" ",
            "quantidade"=>" ",
            "precoCompra"=>" ",
            "precoVenda"=>" "
        );
    }
?>
```

```
<form action="controleAlterarProduto.php" method="post">

    <label>Nome</label><br />
    <input type="text" name="nome" value="<?php echo
$retornoPesquisa["nome"];?>"><br />
    <label>Descrição</label><br /><input type="text"
name="descricao" value="<?php echo
$retornoPesquisa["descricao"];?>"><br />
```

```

        <label>Quantidade</label><br /><input type="text"
name="quantidade" value="<?php echo
$retornoPesquisa["quantidade"];?>"><br />
        <label>Preço de compra</label><br /><input type="text"
name="precoCompra" value="<?php echo
$retornoPesquisa["precoCompra"];?>"><br />
        <label>Preço de venda</label><br /><input type="text"
name="precoVenda" value="<?php echo
$retornoPesquisa["precoVenda"];?>"><br />
        <input type="hidden" name="idProduto" value = "<?php
echo $idProduto; ?>">
        <input type="submit" value="ALTERAR">
</form>

<table border="1">
<?php

        $selecionarProduto = new Produto();

        $retorno = $selecionarProduto->listarProduto();

        foreach($retorno as $linha){
            ?>
            <tr>
                <td><?php echo $linha["id"]; ?></td>
                <td><?php echo $linha["nome"]; ?></td>
                <td><?php echo $linha["descricao"]; ?></td>
                <td><?php echo $linha["quantidade"]; ?></td>
                <td><?php echo $linha["precoCompra"]; ?></td>
                <td><?php echo $linha["precoVenda"]; ?></td>
                <td><a href="formAlterar.php?idProduto=<?php echo
$linha["id"]; ?>">Editar</a></td>
                <td><a
href="controleExcluirProduto.php?idProduto=<?php echo
$linha["id"]; ?>">Excluir</a></td>
            </tr>
            <?php
        }
    ?>
</table>

<?php
//..continuando a classe produto
public function alterar($idProduto, $nome, $descricao,
$quantidade, $precoCompra, $precoVenda){
    $this->setIdProduto($idProduto);
    $this->setNome($nome);
    $this->setdescricao($descricao);
    $this->setQuantidade($quantidade);

```



```

$this->setPrecoCompra($precoCompra);
$this->setPrecoVenda($precoVenda);

echo $sql = "update produto set
nome = '{$this->getNome()}',
descricao = '{$this->getDescricao()}',
quantidade = '{$this->getQuantidade()}',
precoCompra='{$this->getPrecoCompra()}',
precoVenda='{$this->getPrecoVenda()}' where id =
'{$this->getIdProduto()}'
";
include 'conexao.php';
$deletar = $conectar->query($sql);
$registroAfetados = $conectar->affected_rows;
if ($registroAfetados == 1){
    return 1;
}else{
    return 0;
}
}
?>

```

Notem que no final do código do formulário, temos uma tabela que lista todos os registros da tabela, isso é possível devido ao objeto `$selecionarProduto`, que está chamando o método `listarProduto()`. Como este retorno será um array de registros, colocamos ele para imprimir num `foreach`, onde em cada linha será apresentada um registro, e em cada coluna, será apresentada os campos da tabela do banco de dados. Notem que na última coluna, temos dois links, que nos permitirão editar o registro e excluir o registro. Ao selecionar o editar, ele traz para o próprio arquivo passando o identificador do registro como parâmetro.

Após este arquivo pronto, como o de cadastrar, precisamos receber os dados e passar este para o método que irá alterar os dados da tabela.

```

<?php
include 'classes/Produto.php';

$alterarProduto = new Produto();

$idProduto = filter_input(INPUT_POST, "idProduto");
$nome = filter_input(INPUT_POST, "nome");
$descricao = filter_input(INPUT_POST, "descricao");
$quantidade = filter_input(INPUT_POST, "quantidade");
$precoCompra = filter_input(INPUT_POST, "precoCompra");
$precoVenda = filter_input(INPUT_POST, "precoVenda");

```

```
$alterarProduto -> alterar($idProduto, $nome, $descricao,  
$quantidade, $precoCompra, $precoVenda);  
  
if ($alterarProduto){  
    echo "Alterado com sucesso!!!";  
}else{  
    echo "Erro ao alterar as informações";  
}  
?  
>
```

3.7.Excluir Registro

Dando continuidade do arquivo da classe produto, temos o método excluir, que recebe por parâmetro o idProduto, `excluir($idProduto)`. Segue o método abaixo

```
<?php  
//..continuando a classe produto  
public function excluir($idProduto){  
    $this->setIdProduto($idProduto);  
    $sql = "delete from produto where id = '{$this->  
>getIdProduto()}'";  
  
    include 'conexao.php';  
    $deletar = $conectar-> query($sql);  
    $registroAfetados = $conectar->affected_rows;  
    if ($registroAfetados == 1){  
        return 1;  
    }else{  
        return 0;  
    }  
}  
?  
>
```

Chegamos ao final do tutorial de revisão sobre programação orientada a objetos com PHP e myqli. Espero que tenham internalizados os pontos aqui trabalhados de forma prática.