## Introdução à Orientação à Objetos

## Entendendo Herança

Herança em classe para entender melhor vamos ver na prática no exemplo da classe Post, no arquivo a seguir:

**Arquivo index.php**

```
<?php
	class Post{
		private int $id;
		private int $likes = 0;
		//encapsulamento das 2 variáveis
		public function setId($i){
			$this->id = $i;
		}
		public function getId(){
			return $this->id;
		}
		//função set/get do $likes
		public function setLikes($n){
			$this->likes = $n;
		}
		public function getLikes(){
			return $this->likes;
		}
	}
?>
```

Como podem ver nesse exemplo até aqui mostra uma classe de postagem que seta o id, seta a quantidade de likes e acaba nisso. Agora vamos supor que estamos fazendo o facebook e temos um post que é uma foto, temos posts que são vídeos outros que são textos e diversas outras coisas diferentes. Então vamos usar a classe Post acima como a classe geral, como por exemplo, a explicação geral, um ser humano. E depois classificamos esse ser humano por exemplo, de acordo com a nacionalidade, tipo americano, brasileiro, chinês e assim por diante, assim dizendo para melhor compreensão do que vai ser feito a seguir no arquivo index.php. Então voltando para exemplos reais, digitais vamos criar uma nova classe chamada **Foto**, e essa nova classe vai ***estender (extends)*** a classe Post. Que isso quer dizer? Significa que dentro da classe Foto, vamos usar tudo o que tem dentro da classe Post. Ou seja, vamos herdar as coisas da classe Post, daí a palavra herança. Fazendo isso, dentro da classe Foto vamos ter acesso ao setId, getId, setLikes, getLikes. Com isso a classe Foto herda todas as características de Post. Mas a classe Foto pode ter características adicionais, veja o código do index.php com essa aplicação de herança:

**Arquivo index.php**

```
<?php
	class Post {
		private int $id;
		private int $likes = 0;
		//encapsulamento das 2 variáveis
		public function setId($i){
			$this->id = $i;
		}
		public function getId(){
			return $this->id;
		}
		//função set/get do $likes
		public function setLikes($n){
			$this->likes = $n;
		}
		public function getLikes(){
			return $this->likes;
		}
```

```
    }

    class Foto extends Post {
//Ex. na linha a seguir da característica adicional que só a classe foto tem private $url:
        private $url;

        public function __construct($id) {
                $this->setId($id);
        }

        public function setUrl($u){
                $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }

//Usando a classe Foto
$foto = new Foto(20);
$foto->setLikes(12);
$foto->setUrl('abc');

echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();

?>
```

Ao executar o código mostra o número de ids, quantidade de likes e a url:


Como podem ver no exemplo anterior, a classe Foto só tem dois métodos setUrl e getUrl, mas está usando também os métodos da classe Post.

Com uso de herança Deixamos assim o código mais organizado, quando usamos características gerais e depois objetos com características específicas, ou até mesmo reaproveitamento de códigos, por exemplo, podemos deixar a classe Post em um arquivo separado da classe Foto, ou classes, se tivéssemos mais no arquivo index.php herdando herança.

## Propriedade Private e Protected

Vamos agora entender a propriedade privada e protegida. Vamos começar usando a propriedade protected, veja para entender, um exemplo prático a seguir, onde está escrito em vermelho:

**Arquivo index.php**

```
<?php
        class Post {
//Transformemos o id em uma propriedade protected
                protected int $id;
                private int $likes = 0;
                //encapsulamento das 2 variáveis
                public function setId($i){
                        $this->id = $i;
                }
                public function getId(){
```

```
                return $this->id;
        }
        //função set/get do $likes
        public function setLikes($n){
                $this->likes = $n;
        }
        public function getLikes(){
                return $this->likes;
        }
 }

 class Foto extends Post {

//Ex. na linha a seguir da característica adicional que só a classe Foto tem private $url:
    private $url;

    public function __construct($id) {
        $this->setId($id);
    }

    public function setUrl($u){
        $this->url = $u;
    }

    public function getUrl() {
       return $this->url;
    }
```

```
    }
//Usando a classe Foto
$foto = new Foto(20);
$foto->setLikes(12);
$foto->setUrl('abc');

echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();
?>
```

Observem que executando esse arquivo não vai mudar em nada o funcionamento:


Foto: #20 - 12 likes - abc

Agora na classe Foto temos o setId($id) que faz parte da classe Post. Agora vamos alterar esse setId no código, observem onde está escrito em vermelho:

**Arquivo index.php**

```
<?php
	class Post {
		protected int $id;
		private int $likes = 0;
		//encapsulamento das 2 variáveis
		public function setId($i){
			$this->id = $i;
		}
		public function getId(){
			return $this->id;
		}
```

```
        //função set/get do $likes
        public function setLikes($n){
            $this->likes = $n;
        }
        public function getLikes(){
            return $this->likes;
        }
    }

    class Foto extends Post {
        //Ex. na linha a seguir da característica adicional que só a classe foto tem private $url:
        private $url;

        public function __construct($id) {
            $this->id = 600;
        }

        public function setUrl($u){
            $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }

//Usando a classe Foto
```

```
$foto = new Foto(20);

$foto->setLikes(12);

$foto->setUrl('abc');


echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();


?>
```

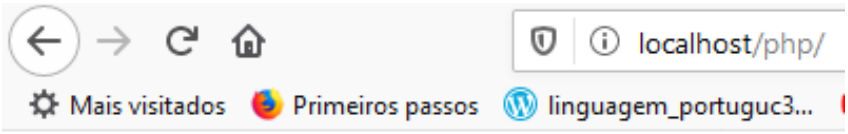


Foto: #600 - 12 likes - abc

Alteramos o id que é da classe Post. Mas observem que não podemos fazer isso fora da classe, observem o estanciamento onde está escrito em vermelho no código:

**Arquivo index.php**

```
<?php
	class Post {
		protected int $id;
		private int $likes = 0;
		//encapsulamento das 2 variáveis
		public function setId($i){
			$this->id = $i;
		}
		public function getId(){
			return $this->id;
		}
		//função set/get do $likes
		public function setLikes($n){
```

```
                $this->likes = $n;
            }
            public function getLikes(){
                return $this->likes;
            }
    }

    class Foto extends Post {
        //Ex. na linha a seguir da característica adicional que só a classe foto tem private $url:
        private $url;

        public function __construct($id) {
            $this->id = 600;
        }

        public function setUrl($u){
            $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }

//Usando a classe Foto
$foto = new Foto(20);
$foto->setLikes(12);
```

```
$foto->setUrl('abc');
$foto->id = 700;


echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();
?>
```

Assim vai ocorrer um erro. Mas se fizermos essa alteração dentro da classe a qual herdamos conseguimos normalmente, por que ela passa a ser uma propriedade da classe Foto.

Agora se colocarmos essa propriedade como **private** não conseguimos fazer essas mesmas alterações. Ou seja, está com uma proteção que não permite fazer alterações diretamente de fora da classe mãe, no caso classe Post. Ela está privada, só pode ter alterações pela classe que a criou. Observem alteração no código onde está escrito em vermelho:

**Arquivo index.php**

```
<?php
	class Post {
		private int $id;
		private int $likes = 0;
		//encapsulamento das 2 variáveis
		public function setId($i){
			$this->id = $i;
		}
		public function getId(){
			return $this->id;
		}
		//função set/get do $likes
		public function setLikes($n){
			$this->likes = $n;
		}
		public function getLikes(){
			return $this->likes;
```

```
                }
    }

    class Foto extends Post {

        private $url;

        public function __construct($id) {
                $this->id = 600;
        }

        public function setUrl($u){
                $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }

//Usando a classe Foto
$foto = new Foto(20);
$foto->setLikes(12);
$foto->setUrl('abc');
echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();
?>
```

Ao executar o código acima observem que dá um erro, ou seja, só aceita alterações das propriedades na classe que as criou, na classe mãe (Post). Aí vocês perguntarão, mas por que setId($id); funciona ? Por que o setId() é público.

```
public function setId($i){
        $this->id = $i;
    }
```

Como a classe Post serve de referência para outras classes, então podemos colocar as funções dentro dessa classe como protected.

Quando herdamos ou quando sobrescrevemos um método específico? Ou seja, podemos colocar a função como protected e recriar essa função onde quisermos. Veja a seguir no código:

**Arquivo index.php**

```
<?php
    class Post {
        private int $id;
        private int $likes = 0;
        //encapsulamento das 2 variáveis
        public function setId($i){
            $this->id = $i;
        }
        public function getId(){
            return $this->id;
        }
        //função set/get do $likes
        protected function setLikes($n){
            $this->likes = $n;
        }
        protected function getLikes(){
            return $this->likes;
```

```
                }
    }

    class Foto extends Post {

        private $url;

        public function __construct($id) {
            $this->setId($id);
                }
            /A função a seguir foi sobrescrita, obs que ela é da classe Post
                public function setLikes($n){
                    echo 'Chamou !';
                }

        public function setUrl($u){
            $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }

//Usando a classe Foto
$foto = new Foto(20);
$foto->setLikes(12);
```

```
$foto->setUrl('abc');


echo "Foto: #".$foto->getId();
//." - ".$foto->getLikes()." likes - ".$foto->getUrl();


?>
```

Observem que mesmo que tenha puxado, estendido de Post em **Foto**, mesmo assim podemos recriar métodos e propriedades que temos herdados, e vai substituir, ou seja prevalece da classe onde recriamos, nesse exemplo na classe Foto onde está no código escrito em vermelho. Caso não tenha nessa classe essa função ( public function setLikes($n) ), então prevalece a da classe mãe que é de onde herdamos, da classe Post.

**Arquivo index.php**

```
<?php
	class Post {
		private int $id;
		private int $likes = 0;
		//encapsulamento das 2 variáveis
		protected function setId($i){
			$this->id = $i;
		}
		public function getId(){
			return $this->id;
		}
		//função set/get do $likes
		public function setLikes($n){
			$this->likes = $n;
		}
		public function getLikes(){
			return $this->likes;
```

```
                }
    }

    class Foto extends Post {

        private $url;

        public function __construct($id) {
                $this->setId($id);
                }

                public function setLikes($n){
                        echo 'Chamou !';
                }

        public function setUrl($u){
                $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }

//Usando a classe Foto
$foto = new Foto(20);
$foto->setLikes(12);
```

```
$foto->setUrl('abc');


echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();


?>
```

Observem que os set, deveriam ser protected e por isso não podem ser usados fora da classe como na linha **$foto->setLikes(12).** Vamos mudar os set para protected e vamos colocar esse set dentro do construtor, observem a seguir:

**Arquivo index.php**

```
<?php
        class Post {
                private int $id;
                private int $likes = 0;
                //encapsulamento das 2 variáveis
                protected function setId($i){
                        $this->id = $i;
                }
                public function getId(){
                        return $this->id;
                }
                //função set/get do $likes
                protected function setLikes($n){
                        $this->likes = $n;
                }
                public function getLikes(){
                        return $this->likes;
                }
    }
```

```
    class Foto extends Post {

        private $url;

        public function __construct($id) {
                        $this->setId($id);
                        $this->setLikes(12);
                }

                public function setLikes($n){
                        echo 'Chamou !';
                }

        public function setUrl($u){
              $this->url = $u;
        }

        public function getUrl() {
            return $this->url;
        }
    }
//Usando a classe Foto
$foto = new Foto(20);
$foto->setUrl('abc');

echo "Foto: #".$foto->getId()." - ".$foto->getLikes()." likes - ".$foto->getUrl();
```

?>

Isso porque o **protected function setLikes($n)** está protegido significando que ele só é utilizável na própria classe ou então em quem herdar ele. Sendo protegido não pode ser usado fora, porque fora da classe só **métodos** e **propriedades públicas**.