Object Oriented Programming in PHP

Berkenalan Dengan Pemrogarman Berorientasi Objek

Pemrograman berorientasi objek (OOP) adalah metode yang mengatur kode ke dalam objek-objek yang memiliki atribut dan metode. Ini mencerminkan dunia nyata dalam bentuk kode. Berikut ringkasan konsep dasar OOP:

1. **Kelas (Class)**: Cetak biru untuk membuat objek yang mendefinisikan atribut dan metode.
2. **Objek (Object)**: Instansiasi dari kelas, yaitu variabel yang menggunakan atribut dan metode kelas.
3. **Atribut (Attribute)**: Variabel yang dimiliki objek, menggambarkan karakteristiknya.
4. **Metode (Method)**: Fungsi yang dijalankan oleh objek untuk melakukan tugas tertentu.
5. **Enkapsulasi**: Pembungkusan atribut dan metode dalam kelas, memungkinkan pengaturan aksesibilitas.

6. **Pewarisan (Inheritance)**: Kelas baru dapat mewarisi atribut dan metode dari kelas lain.
7. **Polimorfisme**: Kemampuan objek untuk merespons metode yang sama dengan cara berbeda tergantung kelasnya.
8. **Abstraksi (Abstraction)**: Fokus hanya pada informasi penting dari objek, mengabaikan detail yang tidak relevan.
9. **Instantiasi (Instantiation)**: Proses pembuatan objek dari kelas.
10. **Kelas Abstrak (Abstract Class)**: Kelas yang tidak bisa diinstansiasi tetapi digunakan sebagai dasar kelas turunan.
11. **Antarmuka (Interface)**: Kontrak yang menentukan metode yang harus diimplementasikan oleh kelas yang menggunakannya.

Ringkasan ini menyederhanakan konsep OOP, membantu pemahaman lebih cepat.

Pemrogarman Prosedular VS Pemrograman Berorientasi Objek

Berikut adalah perbandingan antara Pemrograman Prosedural (PP) dan Pemrograman Berorientasi Objek (PBO), serta situasi di mana PBO mungkin lebih cocok:

1. **Kompleksitas Proyek**:
    - **PP**: Cocok untuk proyek yang lebih sederhana dengan alur linear.
    - **PBO**: Lebih sesuai untuk proyek kompleks dengan banyak entitas dan interaksi antar bagian.

2. **Reusabilitas Kode**:
    - **PP**: Kode lebih spesifik untuk satu proyek atau fungsi.
    - **PBO**: Kelas dan objek yang dibuat dapat digunakan kembali di berbagai proyek.

3. **Pengembangan Kolaboratif**:
    - **PP**: Lebih sulit untuk dikerjakan oleh banyak pengembang karena keterkaitan antara berbagai bagian kode.
    - **PBO**: Mendukung pengembangan oleh tim, karena kelas dapat dikembangkan secara independen.

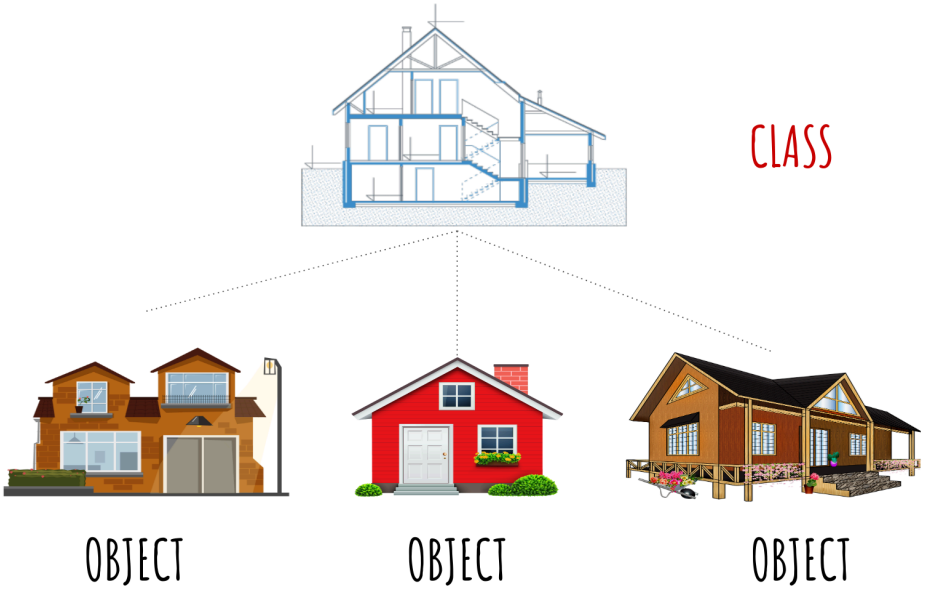
4. **Skalabilitas**:
    - **PP**: Kurang fleksibel dalam penambahan fitur baru tanpa memengaruhi keseluruhan kode.
    - **PBO**: Desain modular memudahkan penambahan fungsionalitas tanpa mengganggu kode yang ada.
5. **Pewarisan dan Polimorfisme**:
    - **PP**: Tidak memiliki fitur pewarisan atau polimorfisme.
    - **PBO**: Memungkinkan pewarisan dan polimorfisme, yang sangat berguna untuk mengorganisasi hierarki kelas dan metode.
6. **Abstraksi**:
    - **PP**: Detail implementasi lebih eksplisit, tanpa banyak abstraksi.
    - **PBO**: Menyediakan abstraksi, menyembunyikan detail yang tidak perlu dan membuat kode lebih bersih.

Kesimpulan:

- **PBO** lebih cocok untuk proyek besar, kompleks, atau kolaboratif, serta untuk kode yang diinginkan modular dan dapat digunakan ulang.
- **PP** tetap relevan untuk proyek kecil atau skrip sederhana, dan lebih natural untuk bahasa pemrograman seperti C yang mengutamakan pendekatan prosedural.

Basic OOP

Berkenalan dengan Class dan Objek



Kelas dan objek dalam PHP adalah dasar dari Pemrograman Berorientasi Objek (OOP), di mana **kelas** adalah cetak biru untuk membuat **objek**. Berikut penjelasan singkatnya:

Kelas (Class)

- Kelas adalah struktur yang mendefinisikan atribut dan metode.
- Kelas digunakan untuk merepresentasikan entitas atau konsep dari dunia nyata dalam bentuk kode.
- Atribut adalah variabel yang dimiliki oleh kelas, sedangkan metode adalah fungsi yang dimiliki kelas.

Contoh Kelas dalam PHP:

```php
class Mobil {
    public $merek;
    public $model;

    public function info() {
        echo "Mobil ini adalah {$this->merek} {$this->model}.";
    }
}
```

Objek (Object)

- Objek adalah instansiasi atau wujud nyata dari kelas.
- Objek mewarisi semua atribut dan metode yang didefinisikan dalam kelas.

Contoh Pembuatan Objek dari Kelas:

```php
$mobil1 = new Mobil();
$mobil2 = new Mobil();

$mobil1->merek = "Toyota";
$mobil1->model = "Camry";

$mobil2->merek = "Honda";
$mobil2->model = "Civic";
```

Mengakses Atribut dan Memanggil Metode

- Atribut dan metode dari sebuah objek dapat diakses menggunakan tanda panah (->).

Contoh Penggunaan Objek:

```php
echo $mobil1->merek;  // Output: Toyota
echo $mobil2->model;  // Output: Civic

$mobil1->info();  // Output: Mobil ini adalah Toyota Camry.
$mobil2->info();  // Output: Mobil ini adalah Honda Civic.
```

Dengan memahami konsep ini, Anda dapat memanfaatkan OOP dalam PHP untuk membuat kode yang lebih terstruktur dan mudah di-maintain.

Berkenalan Dengan Property dan Method

Property dan method adalah dua komponen penting dalam kelas (class) dalam pemrograman berorientasi objek (OOP) dengan PHP. Ini adalah cara untuk mendefinisikan data (property) dan perilaku (method) dari objek yang akan dibuat dari kelas tersebut. Mari kita lihat lebih rinci:

Property (Atribut) dalam PHP:

- Property adalah variabel yang terkait dengan suatu kelas. Ini mendefinisikan karakteristik atau atribut dari objek yang akan dibuat dari kelas tersebut
- Property dapat memiliki visibilitas yang berbeda, seperti public, private, atau protected. Ini memengaruhi aksesibilitas property dari luar kelas.

Method dalam PHP:

- Method adalah fungsi yang terkait dengan suatu kelas. Ini mendefinisikan perilaku atau operasi yang dapat dilakukan pada objek yang dibuat dari kelas tersebut.
- Seperti property, method juga dapat memiliki visibilitas yang berbeda, seperti public, private, atau protected. Contoh penggunaan property dan method dalam kelas Mobil:

```php
class Mobil {
    // Property dengan visibilitas public
    public $merek;
    public $model;

    // method
    public function pesanProduk() {
        return "Produk dipesan....";
    }
}
```

Cara Mengakses Properti dan Metode di PHP

Dalam **Pemrograman Berorientasi Objek (OOP)** di PHP, **properti** dan **metode** dalam suatu kelas diakses melalui objek yang dibuat dari kelas tersebut. Berikut ini adalah cara mengaksesnya secara jelas:

1. Mengakses Properti

- **Properti** adalah variabel yang ada dalam suatu kelas dan digunakan untuk menyimpan data atau nilai.
- Untuk mengakses properti suatu kelas, gunakan tanda **panah (–>)** pada objek yang telah dibuat.

Contoh:

```php
class Produk {
    public $merek = "Samsung";  // Properti kelas
    public $harga = 4000000;     // Properti kelas
}

// Membuat objek dari kelas Produk
$televisi = new Produk();

// Mengakses properti dari objek
echo $televisi->merek;  // Output: Samsung
echo $televisi->harga;  // Output: 4000000
```

2. Mengakses Metode

- **Metode** adalah fungsi yang ada dalam suatu kelas, dan digunakan untuk menjalankan aksi tertentu.
- Untuk memanggil metode dari suatu objek, juga gunakan tanda **panah (–>)**, diikuti dengan nama metode dan tanda kurung `()`.

Contoh:

```
class Produk {
    public $merek = "Samsung";  // Properti kelas
    public $harga = 4000000;     // Properti kelas

    // Metode kelas
    public function pesanProduk() {
        return "Produk telah dipesan!";
    }
}

// Membuat objek dari kelas Produk
$televisi = new Produk();

// Memanggil metode dari objek
echo $televisi->pesanProduk();  // Output: Produk telah dipesan!
```

3. Mengakses Properti Private dengan Getter/Setter

- Jika sebuah properti menggunakan visibilitas **private**, properti tersebut tidak bisa diakses langsung dari luar kelas.
- Untuk mengaksesnya, Anda harus menggunakan **getter** (untuk mendapatkan nilai) dan **setter** (untuk mengatur nilai).

Contoh:

```
class Produk {
    private $harga;

    // Setter untuk mengatur nilai harga
    public function setHarga($harga) {
        $this->harga = $harga;
    }

    // Getter untuk mendapatkan nilai harga
    public function getHarga() {
        return $this->harga;
    }
```

```php
}

// Membuat objek dari kelas Produk
$televisi = new Produk();

// Mengatur dan mengakses properti private dengan setter dan getter
$televisi->setHarga(5000000);
echo $televisi->getHarga();  // Output: 5000000
```

Kesimpulan:

1. **Properti** diakses langsung menggunakan tanda panah (`->`).
2. **Metode** dipanggil dengan tanda panah diikuti oleh tanda kurung `()`.
3. Jika properti bersifat **private**, akses dilakukan melalui **getter** dan **setter**.

Dengan memahami cara mengakses properti dan metode, Anda dapat bekerja lebih efektif dengan objek dan kelas dalam PHP.

Berikut adalah contoh cara mengisi nilai ke dalam **properti** dalam kelas di PHP:

4. Contoh: Mengisi Nilai ke Properti dengan Cara Langsung

Anda dapat mengisi atau mengubah nilai properti dari objek yang dibuat dengan cara langsung setelah objek dibuat.

```php
<?php
class Produk {
    public $merek;
    public $harga;

    // Metode untuk menampilkan informasi produk
    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Membuat objek dari kelas Produk
$televisi = new Produk();

// Mengisi nilai ke dalam properti
$televisi->merek = "LG";       // Mengisi properti 'merek'
$televisi->harga = 5500000;    // Mengisi properti 'harga'

// Menampilkan informasi produk
echo $televisi->infoProduk();  // Output: Merek: LG, Harga: Rp. 5500000
```

5. Contoh: Mengisi Nilai ke Properti Menggunakan Metode Setter

Jika properti memiliki visibilitas **private**, Anda harus menggunakan metode **setter** untuk mengisi nilainya.

```php
<?php
class Produk {
    private $merek;
    private $harga;

    // Setter untuk mengisi nilai ke properti
    public function setMerek($merek) {
        $this->merek = $merek;
    }

    public function setHarga($harga) {
        $this->harga = $harga;
    }

    // Metode untuk menampilkan informasi produk
    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Membuat objek dari kelas Produk
$televisi = new Produk();

// Mengisi nilai ke dalam properti menggunakan metode setter
$televisi->setMerek("Sony");
$televisi->setHarga(6000000);

// Menampilkan informasi produk
echo $televisi->infoProduk();  // Output: Merek: Sony, Harga: Rp. 6000000
```

Kesimpulan:

1. Anda bisa langsung mengisi nilai ke properti jika visibilitasnya **public**.
2. Jika properti bersifat **private**, gunakan metode **setter** untuk mengisi nilainya.

Dengan dua metode ini, Anda bisa fleksibel dalam mengisi nilai properti sesuai kebutuhan dan visibilitas properti di kelas.

Pseudo-variable `$this` dalam PHP OOP

Dalam pemrograman berorientasi objek (OOP), **pseudo-variable** `$this` adalah salah satu konsep penting yang digunakan untuk merujuk ke **instance objek saat ini** di dalam suatu kelas. Ketika Anda bekerja dengan objek, `$this` memungkinkan Anda untuk mengakses properti dan metode milik objek tersebut dari dalam kelas yang sama.

Penjelasan:

1. **Mengacu pada Objek Saat Ini:**
   - `$this` selalu mengacu pada objek spesifik yang memanggil metode di dalam suatu kelas. Setiap objek yang dibuat dari kelas memiliki salinan propertinya sendiri, dan `$this` membantu Anda mengakses propertinya.
2. **Akses Properti dan Metode:**
   - Menggunakan `$this`, Anda dapat mengakses properti (variabel yang dideklarasikan dalam kelas) dan metode (fungsi dalam kelas) dari **objek tertentu** yang sedang memanggil metode tersebut.
3. **Membedakan Properti Kelas dan Variabel Lokal:**
   - Sering kali, nama parameter dalam metode sama dengan nama properti dalam kelas. `$this` digunakan untuk membedakan antara variabel lokal (parameter) dan properti objek itu sendiri. Misalnya, `$this->merek` menunjuk pada properti **`merek`** dari objek saat ini, sementara `merek` (tanpa `$this`) adalah parameter metode.

Contoh Penggunaan `$this`

```
<?php
class Produk {
    public $merek;
    public $harga;

    // Metode untuk mengisi nilai properti dengan $this
    public function setMerek($merek) {
        $this->merek = $merek;  // Menggunakan $this untuk mengakses
properti 'merek'
    }

    public function setHarga($harga) {
        $this->harga = $harga;  // Menggunakan $this untuk mengakses
properti 'harga'
    }

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Membuat objek dari kelas Produk
$televisi = new Produk();
$televisi->setMerek("Samsung");
```

```php
$televisi->setHarga(5000000);

echo $televisi->infoProduk();  // Output: Merek: Samsung, Harga: Rp. 5000000
?>
```

Penjelasan:

- Pada metode `setMerek`, `$this->merek` merujuk pada properti `merek` dari objek $televisi**, yang merupakan instance dari kelas **Produk**. **`merek` adalah parameter lokal yang diinput saat metode dipanggil.
- Ketika Anda menggunakan `$this->merek`, Anda sebenarnya mengakses properti `merek` dari objek yang memanggil metode tersebut.

Mengapa `$this` Penting?

1. **Mengakses Properti dan Metode dari Dalam Kelas:**
    - `$this` sangat penting untuk bekerja dengan propertinya sendiri dan memastikan bahwa objek dapat memodifikasi data di dalam dirinya sendiri. Ini memungkinkan OOP memanipulasi data spesifik dari objek saat ini, bukan variabel acak.
2. **Membantu Membedakan Lingkup Variabel:**
    - Saat ada nama variabel atau parameter yang sama dalam metode, menggunakan `$this` sangat penting untuk memastikan bahwa Anda merujuk pada **properti dari objek**, bukan variabel lokal atau parameter dalam metode.

Contoh Masalah Tanpa `$this`:

```php
<?php
class Produk {
    public $merek;

    public function setMerek($merek) {
        $merek = $merek;  // Ini hanya mengubah variabel lokal, bukan
properti kelas
    }

    public function getMerek() {
        return $this->merek;  // Akan mengembalikan null atau nilai default
    }
}

$televisi = new Produk();
```

```
$televisi->setMerek("LG");

echo $televisi->getMerek();  // Output: null
?>
```

Penjelasan:

- Di sini, metode `setMerek` gagal mengubah properti `merek` dari objek *televisi* ∗ ∗*karenatidakmenggunakan* ∗ ∗‘**this** `**. Sebaliknya, kode hanya bekerja dengan variabel lokal yang namanya kebetulan sama dengan nama properti. Hasilnya, properti **merek`** tetap tidak berubah, dan output adalah **null**.

Kapan Menggunakan `$this`?

- Anda harus menggunakan `$this` setiap kali Anda ingin mengakses atau memodifikasi **properti atau metode** dari objek yang ada **di dalam kelas**. Ini adalah cara utama dalam PHP OOP untuk berinteraksi dengan objek.

Kesimpulan:

1. `$this` adalah pseudo-variable yang **mengacu pada objek saat ini** di dalam metode sebuah kelas.
2. `$this` digunakan untuk mengakses **properti dan metode** dari objek yang sedang bekerja.
3. Tanpa `$this`, PHP akan menganggap variabel sebagai variabel lokal atau parameter, dan properti objek tidak akan diakses atau diubah.
4. `$this` membantu menjaga **lingkup variabel** agar tidak bercampur dengan variabel lokal atau parameter metode.

Dengan memahami pseudo-variable `$this`, Anda dapat dengan lebih mudah bekerja dengan OOP dalam PHP, khususnya dalam konteks mengelola data dari objek yang Anda buat.

Argument dan Parameter

Dalam pemrograman berorientasi objek (OOP) di PHP, **metode** dalam sebuah kelas dapat memiliki **parameter** dan menerima **argument** seperti fungsi pada umumnya. Parameter dalam metode kelas memungkinkan Anda untuk mengirimkan data ke objek

saat memanggil metode tertentu, dan argument adalah nilai yang dikirimkan ke parameter tersebut.

Pengertian Parameter dan Argument dalam Kelas

- **Parameter** adalah variabel yang digunakan dalam deklarasi metode untuk menerima nilai saat metode dipanggil.
- **Argument** adalah nilai aktual yang Anda kirimkan saat memanggil metode tersebut.

Contoh Sederhana Kelas dengan Argument dan Parameter

Misalkan kita memiliki kelas **Produk** yang berisi metode untuk menetapkan merek dan harga produk. Metode-metode ini menerima **argument** yang menggantikan **parameter** ketika dipanggil.

```php
<?php
class Produk {
    public $merek;
    public $harga;

    // Metode dengan parameter untuk menerima argument
    public function setMerek($merek) {
        $this->merek = $merek;
    }

    public function setHarga($harga) {
        $this->harga = $harga;
    }

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Membuat objek dari kelas Produk
$televisi = new Produk();

// Mengirimkan argument ke parameter metode
$televisi->setMerek("Samsung");   // "Samsung" menjadi nilai untuk parameter $merek
$televisi->setHarga(5000000);     // 5000000 menjadi nilai untuk parameter $harga

// Menampilkan informasi produk
echo $televisi->infoProduk();     // Output: Merek: Samsung, Harga: Rp. 5000000
?>
```

Penjelasan:

1. `$merek` dan `$harga` adalah **parameter** dalam metode **setMerek()** dan **setHarga()**.
2. Saat Anda memanggil metode **setMerek("Samsung")**, argument **"Samsung"** dikirim ke parameter **$merek**.
3. Demikian juga, **5000000** dikirim sebagai argument ke parameter **$harga** saat metode **setHarga(5000000)** dipanggil.

Parameter dengan Nilai Default

Sama seperti fungsi biasa, metode dalam kelas juga bisa memiliki **nilai default** pada parameternya. Jika nilai argument tidak diberikan saat metode dipanggil, maka nilai default akan digunakan.

Contoh:

```php
<?php
class Produk {
    public $merek;
    public $harga;

    public function setMerek($merek = "Tidak Diketahui") {
        $this->merek = $merek;
    }

    public function setHarga($harga = 1000000) {
        $this->harga = $harga;
    }

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

$televisi = new Produk();
$televisi->setMerek();  // Menggunakan nilai default "Tidak Diketahui"
$televisi->setHarga();  // Menggunakan nilai default 1000000

echo $televisi->infoProduk();  // Output: Merek: Tidak Diketahui, Harga: Rp. 1000000
?>
```

Parameter dengan Tipe Data (Type Hinting)

PHP mendukung **type hinting** untuk parameter dalam metode kelas. Ini memungkinkan Anda untuk memastikan bahwa argument yang dikirimkan sesuai dengan tipe data yang diharapkan.

Contoh Tipe Data pada Parameter:

```php
<?php
class Produk {
    public $harga;

    // Parameter $harga harus bertipe int
    public function setHarga(int $harga) {
        $this->harga = $harga;
    }

    public function infoHarga() {
        return "Harga: Rp. {$this->harga}";
    }
}

$televisi = new Produk();
$televisi->setHarga(5000000);  // Argument yang dikirim harus bertipe
integer

echo $televisi->infoHarga();   // Output: Harga: Rp. 5000000
?>
```

Jika Anda mencoba mengirimkan argument yang tidak sesuai tipe, PHP akan menghasilkan **error**.

Pass by Reference dalam Metode Kelas

Dalam PHP, secara default argument dikirim ke metode sebagai **salinan** (pass by value), sehingga perubahan yang dilakukan pada parameter tidak memengaruhi nilai asli dari argument yang dikirim. Namun, Anda bisa membuat argument dikirimkan sebagai **referensi** dengan menambahkan simbol **&** di depan parameter. Dengan demikian, perubahan pada parameter akan mempengaruhi nilai asli.

Contoh Pass by Reference:

```php
<?php
class Produk {
    public $harga;

    // Menggunakan reference untuk parameter $harga
    public function diskonHarga(&$harga) {
        $harga -= 500000;  // Mengurangi harga sebesar 500000
    }
}

$hargaTelevisi = 5000000;
```

```php
$televisi = new Produk();
$televisi->diskonHarga($hargaTelevisi);

echo $hargaTelevisi;  // Output: 4500000 (nilai asli berubah karena pass by reference)
?>
```

Variadic Method (Parameter dengan Jumlah Argument Dinamis)

PHP memungkinkan Anda membuat metode yang menerima **argument dengan jumlah yang tidak terbatas** menggunakan **spread operator (`...`)** di depan parameter.

Contoh Variadic Method:

```php
<?php
class Produk {
    public function setHarga(...$harga) {
        return array_sum($harga);
    }
}

$televisi = new Produk();

// Mengirimkan beberapa argument sekaligus
$totalHarga = $televisi->setHarga(1000000, 2000000, 3000000);

echo $totalHarga;  // Output: 6000000
?>
```

Dengan menggunakan `...$harga`, metode **setHarga()** bisa menerima **jumlah argument yang tidak terbatas**.

Kesimpulan:

1. **Argument dan Parameter** dalam metode kelas digunakan untuk mengirimkan nilai dari luar ke dalam metode, seperti pada fungsi biasa.
2. Parameter dapat memiliki **nilai default**, sehingga argument tidak harus selalu diberikan saat metode dipanggil.
3. PHP mendukung **type hinting** pada parameter untuk memastikan tipe data yang sesuai.
4. Anda dapat menggunakan **pass by reference** untuk mengubah nilai asli dari argument yang dikirim ke metode.

5. Metode dalam kelas juga bisa dibuat untuk menerima **jumlah argument yang tidak terbatas** menggunakan variadic method.

Dengan memahami cara kerja argument dan parameter dalam metode kelas, Anda dapat membuat metode yang lebih fleksibel dan modular untuk mengelola data di dalam objek.

*Constructor dan Destructor dalam PHP**

1. Constructor

Constructor adalah metode khusus dalam pemrograman berorientasi objek (OOP) yang secara otomatis dipanggil ketika sebuah objek dibuat dari sebuah kelas. Fungsi utama constructor adalah untuk menginisialisasi properti atau melakukan tugas lain yang dibutuhkan saat objek pertama kali diinstansiasi.

Di PHP, constructor didefinisikan dengan menggunakan metode khusus bernama `__construct()`. Ini adalah metode bawaan yang dipanggil secara otomatis saat kita membuat objek baru dari sebuah kelas.

Sintaksis Constructor di PHP:

```php
<?php
class MyClass {
    // Properti
    public $nama;
    public $umur;

    // Constructor
    public function __construct($nama, $umur) {
        $this->nama = $nama;
        $this->umur = $umur;
    }

    // Metode untuk menampilkan informasi
    public function info() {
        echo "Nama: " . $this->nama . ", Umur: " . $this->umur;
    }
}

// Membuat objek dari kelas MyClass
$objek1 = new MyClass("Arif", 30);

// Memanggil metode info
$objek1->info();
?>
```

Penjelasan:

- `__construct()` dipanggil ketika kita membuat objek baru menggunakan `new`.
- Constructor menerima parameter `$nama` dan `$umur`, dan menginisialisasi properti kelas `$this->nama` dan `$this->umur`.
- Ketika objek `$objek1` dibuat, constructor otomatis menjalankan proses inisialisasi dengan nilai yang diberikan.

2. Destructor

Destructor adalah metode khusus yang dipanggil secara otomatis ketika sebuah objek dihapus dari memori atau tidak lagi dibutuhkan. Destructor digunakan untuk membersihkan sumber daya, seperti menutup koneksi database, menulis ke file log, atau tugas-tugas lain yang diperlukan sebelum objek dihapus dari memori.

Di PHP, destructor didefinisikan dengan metode `__destruct()`.

Sintaksis Destructor di PHP:

```php
<?php
class MyClass {
    // Properti
    public $nama;

    // Constructor
    public function __construct($nama) {
        $this->nama = $nama;
        echo "Objek dengan nama " . $this->nama . " telah dibuat.\n";
    }

    // Destructor
    public function __destruct() {
        echo "Objek dengan nama " . $this->nama . " telah dihapus.\n";
    }
}

// Membuat objek dari kelas MyClass
$objek1 = new MyClass("Arif");

// Destructor dipanggil saat script selesai, atau bisa dipanggil secara
eksplisit
unset($objek1);
?>
```

Penjelasan:

- `__destruct()` akan dipanggil ketika objek `$objek1` tidak lagi digunakan, baik secara otomatis pada akhir skrip atau saat objek dihapus secara manual menggunakan `unset()`.
- Pada contoh di atas, destructor mengeluarkan pesan ketika objek dihapus dari memori.

Perbedaan Constructor dan Destructor:

- **Constructor (`__construct()`)**: Dipanggil **saat objek dibuat** dan digunakan untuk menginisialisasi nilai properti atau melakukan tugas lain yang dibutuhkan selama pembuatan objek.
- **Destructor (`__destruct()`)**: Dipanggil **saat objek dihapus dari memori** dan digunakan untuk membersihkan sumber daya atau melakukan tugas-tugas yang harus dilakukan sebelum objek dihancurkan.

Kesimpulan:

- **Constructor**: Untuk inisialisasi objek.
- **Destructor**: Untuk membersihkan atau melakukan tindakan saat objek akan dihapus dari memori.

Ini adalah dasar penggunaan **Constructor** dan **Destructor** dalam PHP yang penting untuk pemrograman berorientasi objek.

Inheritance dalam PHP (Pewarisan)

Inheritance atau **pewarisan** adalah salah satu konsep utama dalam Pemrograman Berorientasi Objek (OOP) di mana sebuah kelas baru dapat mewarisi properti dan metode dari kelas yang sudah ada. Konsep ini memungkinkan **penggunaan kembali kode** yang sudah ada, sehingga kode menjadi lebih modular dan lebih mudah untuk dikembangkan.

Konsep Inheritance:

1. **Parent Class (Kelas Induk)**: Kelas yang mewariskan properti dan metode kepada kelas lain.
2. **Child Class (Kelas Turunan)**: Kelas yang mewarisi properti dan metode dari parent class. Child class dapat menggunakan, menambah, atau bahkan menimpa (override) metode dan properti dari parent class.

Membuat Inheritance dalam PHP

Untuk membuat inheritance di PHP, gunakan kata kunci `extends` agar child class mewarisi dari parent class.

Contoh:

```php
<?php
// Kelas Induk
class Produk {
    public $merek;
    public $harga;

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Kelas Turunan (mewarisi dari kelas Produk)
class Televisi extends Produk {
    public $ukuranLayar;

    // Menambahkan metode baru khusus untuk kelas Televisi
    public function infoUkuran() {
        return "Ukuran layar: {$this->ukuranLayar} inch";
    }
}

// Membuat objek dari kelas turunan
$tv = new Televisi();
$tv->merek = "Samsung";
$tv->harga = 5000000;
$tv->ukuranLayar = 55;

echo $tv->infoProduk();      // Output: Merek: Samsung, Harga: Rp. 5000000
echo $tv->infoUkuran();      // Output: Ukuran layar: 55 inch
?>
```

Penjelasan:

1. `class Produk` adalah **parent class** (kelas induk) yang berisi properti `$merek` dan `$harga`, serta metode `infoProduk()`.
2. `class Televisi` adalah **child class** yang menggunakan keyword `extends` untuk mewarisi properti dan metode dari kelas `Produk`.
3. Di dalam **child class** `Televisi`, kita juga dapat menambahkan properti atau metode baru seperti `$ukuranLayar` dan `infoUkuran()`.

4. Ketika objek `$tv` dari kelas **Televisi** dibuat, ia mewarisi properti `$merek` dan `$harga` dari kelas **Produk**, serta dapat menggunakan metode `infoProduk()`.

Overriding (Menimpa Metode atau Properti Parent Class)

Dalam inheritance, child class bisa **menimpa** (override) metode yang ada di parent class dengan mendefinisikan metode dengan nama yang sama di child class. Ini digunakan ketika Anda ingin mengubah atau menyesuaikan perilaku metode dari parent class.

Contoh Overriding:

```php
<?php
// Kelas Induk
class Produk {
    public $merek;
    public $harga;

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Kelas Turunan
class Televisi extends Produk {
    public $ukuranLayar;

    // Menimpa metode infoProduk() dari kelas induk
    public function infoProduk() {
        return "Televisi: Merek: {$this->merek}, Harga: Rp. {$this->harga},
Ukuran: {$this->ukuranLayar} inch";
    }
}

// Membuat objek dari kelas Televisi
$tv = new Televisi();
$tv->merek = "LG";
$tv->harga = 6000000;
$tv->ukuranLayar = 65;

echo $tv->infoProduk();  // Output: Televisi: Merek: LG, Harga: Rp. 6000000,
Ukuran: 65 inch
?>
```

Penjelasan Overriding:

- Di sini, metode `infoProduk()` di kelas **Televisi** menimpa (override) metode yang sama dari kelas **Produk**. Sekarang, ketika metode `infoProduk()` dipanggil dari

objek `$tv`, ia menggunakan metode yang didefinisikan di **Televisi**, bukan yang ada di **Produk**.

Akses ke Metode Parent Class dengan `parent::`

Jika Anda ingin memanggil metode dari parent class yang telah di-override, Anda bisa menggunakan kata kunci `parent::`.

Contoh:

```php
<?php
// Kelas Induk
class Produk {
    public $merek;
    public $harga;

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Kelas Turunan
class Televisi extends Produk {
    public $ukuranLayar;

    // Override metode infoProduk(), tapi masih memanggil versi dari parent
class
    public function infoProduk() {
        // Memanggil metode infoProduk() dari kelas induk
        $infoProdukInduk = parent::infoProduk();
        return "{$infoProdukInduk}, Ukuran: {$this->ukuranLayar} inch";
    }
}

$tv = new Televisi();
$tv->merek = "LG";
$tv->harga = 6000000;
$tv->ukuranLayar = 65;

echo $tv->infoProduk();  // Output: Merek: LG, Harga: Rp. 6000000, Ukuran:
65 inch
?>
```

Property Overriding dalam PHP

Property Overriding adalah konsep dalam Pemrograman Berorientasi Objek (OOP) di mana sebuah **child class** menimpa atau menggantikan properti yang didefinisikan dalam

parent class dengan properti baru yang memiliki nama yang sama. Dalam PHP, meskipun Anda dapat melakukan **overriding pada properti**, cara terbaik untuk melakukannya adalah melalui metode (getter dan setter) karena properti di PHP tidak sepenuhnya memiliki mekanisme overriding sebaik metode.

Contoh Property Overriding:

```php
<?php
// Kelas Induk (Parent Class)
class Produk {
    public $merek = "Samsung";
    public $harga = 4000000;

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Kelas Turunan (Child Class) yang menimpa properti merek dan harga
class Televisi extends Produk {
    public $merek = "LG";  // Menimpa properti merek di parent class
    public $harga = 5000000;  // Menimpa properti harga di parent class
}

// Membuat objek dari kelas turunan
$tv = new Televisi();

echo $tv->infoProduk();
// Output: Merek: LG, Harga: Rp. 5000000
?>
```

Penjelasan:

1. **Kelas** `Produk` adalah **parent class** yang memiliki properti `$merek` dan `$harga`, serta metode `infoProduk()` untuk menampilkan informasi produk.
2. **Kelas** `Televisi` adalah **child class** yang **menimpa properti** `$merek` dan `$harga` dengan nilai baru, meskipun properti dengan nama yang sama sudah didefinisikan di parent class.
3. Ketika objek `$tv` dipanggil dan menggunakan metode `infoProduk()`, nilai properti yang digunakan adalah yang telah ditimpa di child class, yaitu **"LG"** untuk merek dan **5000000** untuk harga.

Menggunakan Metode Getter dan Setter untuk Property Overriding

Dalam praktik yang baik, seringkali property overriding dilakukan melalui metode **getter** dan **setter**, karena hal ini memberikan kontrol lebih baik terhadap validasi dan logika bisnis yang mungkin ingin diterapkan pada properti.

Contoh Menggunakan Getter dan Setter:

```php
<?php
// Kelas Induk
class Produk {
    protected $merek = "Samsung";
    protected $harga = 4000000;

    // Getter untuk properti merek
    public function getMerek() {
        return $this->merek;
    }

    // Setter untuk properti merek
    public function setMerek($merek) {
        $this->merek = $merek;
    }

    // Getter untuk properti harga
    public function getHarga() {
        return $this->harga;
    }

    // Setter untuk properti harga
    public function setHarga($harga) {
        $this->harga = $harga;
    }
}

// Kelas Turunan
class Televisi extends Produk {

    // Override setter untuk merek
    public function setMerek($merek) {
        $this->merek = "Televisi: " . $merek;
    }
}

$tv = new Televisi();
$tv->setMerek("LG");
$tv->setHarga(5000000);

echo $tv->getMerek();  // Output: Televisi: LG
echo $tv->getHarga();  // Output: 5000000
?>
```

Penjelasan:

1. Di parent class `Produk`, properti `$merek` dan `$harga` didefinisikan sebagai **protected**, sehingga hanya bisa diakses melalui getter dan setter.
2. Di child class `Televisi`, kita menimpa (override) metode `setMerek()` untuk menambahkan prefiks "Televisi: " pada nama merek setiap kali setter dipanggil.
3. Metode getter pada child class tetap menggunakan metode dari parent class tanpa perubahan.

Catatan Penting:

- **Overriding properti** tidak memberikan fleksibilitas sebanyak **overriding metode**. Oleh karena itu, metode getter dan setter sering digunakan untuk memastikan kontrol lebih baik terhadap modifikasi nilai properti.
- `parent::` tidak dapat digunakan secara langsung pada properti, hanya bisa digunakan untuk memanggil metode dari parent class yang telah di-override.

Kesimpulan:

- **Property Overriding** memungkinkan **child class** menimpa properti dari **parent class**.
- Properti yang ditimpa di **child class** akan menggantikan nilai yang ada di **parent class**.
- Praktik yang baik dalam OOP adalah menggunakan metode **getter** dan **setter** untuk mengelola properti, memberikan kontrol lebih besar terhadap modifikasi dan akses nilai properti.

Constructor Overriding dalam PHP

Constructor Overriding adalah konsep dalam Pemrograman Berorientasi Objek (OOP) yang memungkinkan **kelas turunan (child class)** menimpa (override) **konstruktor** dari **kelas induk (parent class)**. Dalam PHP, constructor adalah metode khusus dengan nama `__construct()`, yang dipanggil secara otomatis ketika objek dari sebuah kelas dibuat.

Ketika kita melakukan **overriding constructor**, kita mengganti implementasi constructor di kelas induk dengan constructor baru di kelas turunan. Namun, jika kita tetap ingin menggunakan constructor dari kelas induk, kita bisa memanggilnya menggunakan fungsi `parent::__construct()` di dalam constructor dari kelas turunan.

Contoh Constructor Overriding:

```php
<?php
// Kelas Induk (Parent Class)
class Produk {
    public $merek;
    public $harga;

    // Constructor di kelas induk
    public function __construct($merek, $harga) {
        $this->merek = $merek;
        $this->harga = $harga;
    }

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Kelas Turunan (Child Class) menimpa constructor
class Televisi extends Produk {
    public $ukuranLayar;

    // Constructor di kelas turunan
    public function __construct($merek, $harga, $ukuranLayar) {
        // Memanggil constructor dari parent class
        parent::__construct($merek, $harga);
        $this->ukuranLayar = $ukuranLayar;
    }

    public function infoTelevisi() {
        return parent::infoProduk() . ", Ukuran Layar: {$this->ukuranLayar}
inch";
    }
}

// Membuat objek dari kelas turunan
$tv = new Televisi("LG", 5000000, 55);

echo $tv->infoTelevisi();
// Output: Merek: LG, Harga: Rp. 5000000, Ukuran Layar: 55 inch
?>
```

Penjelasan:

1. **Kelas `Produk` (parent class)** memiliki constructor yang menerima dua parameter, `$merek` dan `$harga`. Constructor ini digunakan untuk menginisialisasi properti tersebut saat objek dibuat.
2. **Kelas `Televisi` (child class)** menimpa constructor dari parent class dan menambahkan parameter tambahan, `$ukuranLayar`. Di dalam constructor kelas turunan, kita menggunakan `parent::__construct($merek, $harga)` untuk

memanggil constructor dari parent class dan menginisialisasi properti yang ada di parent class.

3. **Method `infoTelevisi()`** digunakan untuk menampilkan informasi lengkap, termasuk properti baru `$ukuranLayar` yang ada di kelas turunan.

Menggunakan Constructor Parent Secara Opsional

Anda juga dapat menggunakan constructor dari parent class hanya jika diperlukan, atau bahkan tidak memanggilnya sama sekali jika semua properti diinisialisasi dalam constructor dari child class.

Contoh Tanpa Memanggil Constructor Parent:

```php
<?php
// Kelas Induk
class Produk {
    public $merek;
    public $harga;

    public function infoProduk() {
        return "Merek: {$this->merek}, Harga: Rp. {$this->harga}";
    }
}

// Kelas Turunan dengan constructor baru (tanpa memanggil
parent::__construct)
class Televisi extends Produk {
    public $ukuranLayar;

    public function __construct($ukuranLayar) {
        // Hanya inisialisasi properti di kelas turunan
        $this->merek = "Samsung"; // Properti dari parent class
        $this->harga = 7000000;   // Properti dari parent class
        $this->ukuranLayar = $ukuranLayar; // Properti dari child class
    }

    public function infoTelevisi() {
        return parent::infoProduk() . ", Ukuran Layar: {$this->ukuranLayar}
inch";
    }
}

$tv = new Televisi(50);
echo $tv->infoTelevisi();
// Output: Merek: Samsung, Harga: Rp. 7000000, Ukuran Layar: 50 inch
?>
```

Penjelasan:

- Di contoh ini, **constructor dari parent class** tidak dipanggil, tetapi properti dari parent class tetap diinisialisasi langsung di constructor dari child class.
- `$merek` dan `$harga` diinisialisasi dalam constructor child class `Televisi`, sementara `$ukuranLayar` adalah properti yang khusus untuk kelas turunan tersebut.

Kesimpulan:

- **Constructor Overriding** memungkinkan Anda untuk memberikan implementasi constructor yang berbeda di kelas turunan sambil tetap bisa memanggil constructor dari kelas induk jika dibutuhkan.
- Penggunaan `parent::__construct()` memungkinkan Anda untuk tetap memanfaatkan logika inisialisasi yang ada di kelas induk.
- Overriding constructor sangat berguna ketika kelas turunan membutuhkan parameter tambahan atau memerlukan logika inisialisasi yang berbeda dari kelas induk.

Destructor Overriding dalam PHP

Destructor adalah metode khusus dalam pemrograman berorientasi objek yang digunakan untuk melakukan tindakan pembersihan atau mengakhiri suatu objek ketika objek tersebut dihancurkan atau dieksekusi. Dalam PHP, **destructor** ditentukan dengan nama metode `__destruct()`, yang dipanggil secara otomatis saat objek tidak lagi digunakan atau dihapus dari memori.

Destructor Overriding adalah konsep ketika **kelas turunan (child class)** mendefinisikan ulang destructor yang ada di **kelas induk (parent class)**. Sama seperti constructor, jika Anda ingin menggunakan destructor dari parent class, Anda bisa memanggilnya dengan `parent::__destruct()` dalam destructor child class.

Cara Kerja Destructor dalam PHP:

- Destructor tidak menerima argumen.
- PHP memanggil destructor ketika skrip selesai atau ketika semua referensi ke suatu objek dihapus.
- Destructor biasanya digunakan untuk membersihkan sumber daya seperti file, koneksi basis data, atau menghapus sesi.

Contoh Destructor Overriding:

```php
<?php
// Kelas Induk (Parent Class)
class Produk {
    public $merek;
    public $harga;

    // Constructor
    public function __construct($merek, $harga) {
        $this->merek = $merek;
        $this->harga = $harga;
        echo "Produk {$this->merek} telah dibuat.<br>";
    }

    // Destructor di kelas induk
    public function __destruct() {
        echo "Objek produk {$this->merek} dihapus dari memori.<br>";
    }
}

// Kelas Turunan (Child Class) menimpa destructor
class Televisi extends Produk {
    public $ukuranLayar;

    // Constructor
    public function __construct($merek, $harga, $ukuranLayar) {
        parent::__construct($merek, $harga);  // Memanggil constructor
parent
        $this->ukuranLayar = $ukuranLayar;
    }

    // Destructor di kelas turunan
    public function __destruct() {
        echo "Objek Televisi {$this->merek} dengan layar {$this-
>ukuranLayar} inch dihapus dari memori.<br>";
        parent::__destruct();  // Memanggil destructor parent
    }
}

// Membuat objek dari kelas turunan
$tv = new Televisi("LG", 5000000, 55);

?>
```

Penjelasan:

1. **Kelas `Produk` (parent class)** memiliki constructor yang mencetak pesan ketika objek dibuat dan destructor yang mencetak pesan ketika objek dihapus dari memori.
2. **Kelas `Televisi` (child class)** menimpa destructor untuk menambahkan informasi tentang ukuran layar saat objek dihapus dari memori.

3. Di dalam destructor `Televisi`, setelah mencetak pesan khusus, kita memanggil `parent::__destruct()` untuk memastikan destructor dari parent class juga dieksekusi.
4. Setelah skrip selesai dijalankan, PHP akan memanggil destructor untuk menghapus objek dari memori.

Output:

```
Produk LG telah dibuat.
Objek Televisi LG dengan layar 55 inch dihapus dari memori.
Objek produk LG dihapus dari memori.
```

Kapan Menggunakan Destructor?

- **Membersihkan sumber daya eksternal:** Jika objek membuka file, membuat koneksi ke database, atau menggunakan sesi, destructor dapat digunakan untuk memastikan bahwa semua sumber daya tersebut ditutup atau diakhiri dengan benar sebelum objek dihancurkan.
- **Logging atau jejak aktivitas:** Anda dapat menggunakan destructor untuk mencatat log bahwa objek telah dihancurkan atau operasinya telah selesai.

Tanpa Destructor di Child Class:

Jika Anda tidak menimpa destructor di child class, PHP akan otomatis menggunakan destructor dari parent class saja. Anda tidak perlu mendefinisikan destructor di setiap class kecuali ada kebutuhan khusus.

```php
<?php
class Produk {
    public function __construct() {
        echo "Produk dibuat.<br>";
    }

    public function __destruct() {
        echo "Produk dihapus dari memori.<br>";
    }
}

class Televisi extends Produk {
    public function __construct() {
        parent::__construct();
    }
}
```

```php
$tv = new Televisi();
// Output: Produk dibuat.
// Produk dihapus dari memori.
?>
```

Kesimpulan:

- **Destructor Overriding** memungkinkan Anda untuk menimpa destructor di kelas turunan jika Anda memerlukan logika pembersihan tambahan.
- Destructor parent class bisa dipanggil secara manual di dalam destructor child class dengan `parent::__destruct()` jika Anda masih ingin mempertahankan logika pembersihan dari parent class.
- Destructor berguna untuk membersihkan sumber daya, terutama pada program yang menggunakan banyak interaksi dengan file atau koneksi jaringan.

Dengan demikian, **destructor overriding** memberikan fleksibilitas dalam pengelolaan memori dan pembersihan sumber daya di aplikasi PHP.

Final Keyword dalam PHP

`final` adalah sebuah keyword di PHP yang digunakan untuk menandai kelas, metode, atau atribut yang tidak dapat diubah lebih lanjut oleh kelas turunan. Ketika Anda menggunakan `final`, Anda memastikan bahwa kelas atau metode tersebut tidak bisa di-overriding atau diwarisi.

Penggunaan Final di PHP:

1. **Final Class**
    - Kelas yang ditandai dengan `final` tidak dapat diwarisi. Anda tidak dapat membuat kelas turunan dari kelas `final`.

    Contoh:

    ```php
    <?php
    final class Kendaraan {
        public function info() {
            return "Informasi kendaraan.";
        }
    }
    ```

```
// Kelas berikut ini akan menghasilkan kesalahan jika di-uncomment
// class Mobil extends Kendaraan { } // Error: Cannot inherit from
final class Kendaraan
?>
```

2. **Final Method**

   - Metode yang ditandai dengan `final` dalam sebuah kelas tidak dapat di-overriding oleh kelas turunan. Ini berguna untuk memastikan bahwa implementasi metode tersebut tidak diubah.

   Contoh:

```php
<?php
class Kendaraan {
    final public function info() {
        return "Informasi kendaraan.";
    }
}

class Mobil extends Kendaraan {
    // Kode berikut ini akan menghasilkan kesalahan jika di-uncomment
    // public function info() { return "Mobil."; } // Error: Cannot
override final method Kendaraan::info()
}

$kendaraan = new Kendaraan();
echo $kendaraan->info();  // Output: Informasi kendaraan.
?>
```

3. **Final Property**

   - PHP tidak mendukung `final` untuk property (atribut) secara langsung. Namun, Anda dapat menggunakan metode `final` untuk mengontrol akses atau perilaku property. Penggunaan `final` pada metode dapat membantu menjaga integritas data.

Kapan Menggunakan Final:

- **Keamanan dan Stabilitas**: Menggunakan `final` pada kelas atau metode dapat membantu menjaga integritas kode, memastikan bahwa kelas penting tidak dapat dimodifikasi dan perilakunya tetap konsisten.
- **Desain API**: Dalam pembuatan API atau library, menandai kelas atau metode sebagai `final` menjadi bagian dari kontrak yang memastikan pengguna tidak

mengubah perilaku yang sudah ditentukan.

- **Pengoptimalan Kinerja**: PHP dapat melakukan optimasi tertentu pada metode `final`, yang mungkin meningkatkan kinerja aplikasi dalam beberapa situasi.

Contoh Penggunaan Final dalam Kode:

```php
<?php
final class Matematika {
    public function tambah($a, $b) {
        return $a + $b;
    }

    final public function kali($a, $b) {
        return $a * $b;
    }
}

// Kelas berikut ini akan menghasilkan kesalahan
// class Operasi extends Matematika { } // Error: Cannot inherit from final class Matematika

$matematika = new Matematika();
echo $matematika->tambah(3, 4);  // Output: 7
echo "<br>";
echo $matematika->kali(3, 4);    // Output: 12
?>
```

Kesimpulan:

- **Final** dalam PHP adalah alat yang kuat untuk mengendalikan pewarisan dan perilaku kelas serta metode. Menggunakan `final public function` membantu menjaga stabilitas kode dan mencegah modifikasi yang tidak diinginkan, terutama dalam aplikasi yang kompleks. Dengan memahami cara menggunakan `final`, Anda dapat merancang sistem yang lebih aman dan dapat dipelihara.

Visibility atau Access Modifier dalam PHP

Visibility (modifikasi akses) dalam PHP menentukan tingkat akses terhadap properti dan metode dalam sebuah kelas. PHP memiliki tiga jenis access modifier: `public`, `protected`, dan `private`. Masing-masing modifier ini memiliki fungsi dan penggunaan yang berbeda dalam konteks pemrograman berorientasi objek (OOP).

1. Public

- **Definisi**: Properti atau metode yang ditandai dengan modifier `public` dapat diakses dari mana saja. Ini berarti bahwa properti atau metode tersebut dapat diakses dari dalam kelas, dari kelas turunan, dan juga dari luar kelas.
- **Penggunaan**: Gunakan `public` ketika Anda ingin agar properti atau metode dapat diakses secara luas.

Contoh:

```php
<?php
class Kendaraan {
    public $merek;

    public function info() {
        return "Ini adalah kendaraan.";
    }
}

$kendaraan = new Kendaraan();
$kendaraan->merek = "Toyota"; // Akses dari luar kelas
echo $kendaraan->info();       // Output: Ini adalah kendaraan.
?>
```

2. Protected

- **Definisi**: Properti atau metode yang ditandai dengan modifier `protected` hanya dapat diakses dari dalam kelas itu sendiri dan dari kelas yang mewarisinya (kelas turunan). Ini tidak dapat diakses dari luar kelas.
- **Penggunaan**: Gunakan `protected` ketika Anda ingin membatasi akses ke properti atau metode, tetapi masih ingin memungkinkan kelas turunan untuk mengaksesnya.

Contoh:

```php
<?php
class Kendaraan {
    protected $merek;

    protected function info() {
        return "Ini adalah kendaraan.";
    }
}

class Mobil extends Kendaraan {
    public function setMerek($merek) {
        $this->merek = $merek; // Akses dari kelas turunan
```

```
    }

    public function getInfo() {
        return $this->info(); // Akses dari kelas turunan
    }
}

$mobil = new Mobil();
$mobil->setMerek("Honda");
echo $mobil->getInfo(); // Output: Ini adalah kendaraan.
?>
```

3. Private

- **Definisi**: Properti atau metode yang ditandai dengan modifier `private` hanya dapat diakses dari dalam kelas itu sendiri. Ini tidak dapat diakses dari kelas turunan maupun dari luar kelas.
- **Penggunaan**: Gunakan `private` ketika Anda ingin melindungi properti atau metode dari akses luar dan dari kelas turunan, sehingga hanya dapat diakses oleh metode dalam kelas yang sama.

Contoh:

```
<?php
class Kendaraan {
    private $merek;

    public function setMerek($merek) {
        $this->merek = $merek; // Akses dari dalam kelas
    }

    private function info() {
        return "Ini adalah kendaraan: " . $this->merek;
    }

    public function getInfo() {
        return $this->info(); // Akses dari dalam kelas
    }
}

$kendaraan = new Kendaraan();
$kendaraan->setMerek("Toyota");
echo $kendaraan->getInfo(); // Output: Ini adalah kendaraan: Toyota
// echo $kendaraan->info(); // Error: Cannot access private method
Kendaraan::info()
?>
```

Perbandingan Access Modifier:

Modifier	Akses dari Dalam Kelas	Akses dari Kelas Turunan	Akses dari Luar Kelas
public	Ya	Ya	Ya
protected	Ya	Ya	Tidak
private	Ya	Tidak	Tidak

Kesimpulan:

- **Public**: Digunakan untuk properti dan metode yang perlu diakses secara luas.
- **Protected**: Membatasi akses hanya untuk kelas itu sendiri dan kelas turunan, menjaga informasi dari akses luar.
- **Private**: Melindungi properti dan metode sepenuhnya dari akses luar dan dari kelas turunan, memastikan kontrol penuh atas data dalam kelas.

Dengan memahami dan menggunakan access modifier dengan benar, Anda dapat merancang kelas yang lebih baik dan lebih aman, menjaga integritas data dalam aplikasi Anda.

Getter dan Setter dalam PHP

Getter dan setter adalah metode yang digunakan untuk mengakses dan memodifikasi nilai dari properti dalam sebuah kelas, terutama ketika properti tersebut bersifat `private` atau `protected`. Penggunaan getter dan setter adalah praktik yang baik dalam pemrograman berorientasi objek (OOP) karena dapat meningkatkan enkapsulasi, menjaga integritas data, dan memberikan kontrol lebih besar terhadap bagaimana properti diakses dan diubah.

1. Getter

Getter adalah metode yang digunakan untuk mengambil atau mendapatkan nilai dari properti. Biasanya, metode ini dinamai dengan awalan `get` diikuti oleh nama properti yang ingin diakses.

Contoh:

```
<?php
class Kendaraan {
    private $merek;
```

```
    // Getter
    public function getMerek() {
        return $this->merek;
    }

    // Setter
    public function setMerek($merek) {
        $this->merek = $merek;
    }
}

$kendaraan = new Kendaraan();
$kendaraan->setMerek("Toyota");
echo $kendaraan->getMerek(); // Output: Toyota
?>
```

2. Setter

Setter adalah metode yang digunakan untuk mengatur atau memodifikasi nilai dari properti. Biasanya, metode ini dinamai dengan awalan `set` diikuti oleh nama properti yang ingin diubah.

Contoh:

```
<?php
class Kendaraan {
    private $merek;

    // Getter
    public function getMerek() {
        return $this->merek;
    }

    // Setter
    public function setMerek($merek) {
        $this->merek = $merek;
    }
}

$kendaraan = new Kendaraan();
$kendaraan->setMerek("Honda");
echo $kendaraan->getMerek(); // Output: Honda
?>
```

Manfaat Menggunakan Getter dan Setter:

1. **Enkapsulasi**: Dengan menyembunyikan properti di balik getter dan setter, Anda dapat melindungi data dan memastikan bahwa hanya metode yang ditentukan yang dapat mengakses atau mengubah nilai properti.

2. **Validasi Data**: Anda dapat menambahkan logika validasi dalam metode setter untuk memastikan bahwa nilai yang diberikan sesuai dengan kriteria tertentu.

   Contoh Validasi dalam Setter:

```php
<?php
class Kendaraan {
    private $merek;

    public function getMerek() {
        return $this->merek;
    }

    public function setMerek($merek) {
        if (is_string($merek) && !empty($merek)) {
            $this->merek = $merek;
        } else {
            throw new Exception("Merek harus berupa string yang tidak kosong.");
        }
    }
}

$kendaraan = new Kendaraan();
try {
    $kendaraan->setMerek("Toyota");
    echo $kendaraan->getMerek(); // Output: Toyota

    // $kendaraan->setMerek(""); // Akan menghasilkan Exception
} catch (Exception $e) {
    echo $e->getMessage(); // Output: Merek harus berupa string yang tidak kosong.
}
?>
```

3. **Kontrol Akses**: Anda dapat mengubah tingkat akses properti (misalnya, mengubah dari `private` ke `protected`) tanpa mengubah cara pengguna mengakses atau mengubah data.

Kesimpulan:

Getter dan setter adalah alat penting dalam pemrograman berorientasi objek yang membantu menjaga enkapsulasi dan integritas data. Dengan menggunakan metode ini, Anda dapat memberikan akses yang lebih terkontrol ke properti kelas, serta

menambahkan logika untuk memvalidasi data yang diterima, sehingga menciptakan kode yang lebih bersih dan aman.

Penggunaan Setter, Getter, dan Constructor dalam PHP

Dalam pemrograman berorientasi objek (OOP) di PHP, setter, getter, dan constructor adalah komponen penting yang memungkinkan pengelolaan data dalam kelas secara efektif. Berikut adalah penjelasan rinci tentang masing-masing komponen, lengkap dengan contoh penggunaannya.

1. Constructor

Constructor adalah metode khusus yang secara otomatis dipanggil saat objek dari kelas dibuat. Constructor digunakan untuk menginisialisasi properti objek dengan nilai awal. Dalam PHP, constructor dideklarasikan dengan nama metode `__construct`.

Contoh Penggunaan Constructor:

```php
<?php
class Mobil {
    private $merek;
    private $model;

    // Constructor
    public function __construct($merek, $model) {
        $this->merek = $merek;
        $this->model = $model;
    }

    // Getter untuk merek
    public function getMerek() {
        return $this->merek;
    }

    // Getter untuk model
    public function getModel() {
        return $this->model;
    }
}

// Membuat objek Mobil dengan constructor
$mobil = new Mobil("Toyota", "Camry");
echo "Merek: " . $mobil->getMerek() . "\n"; // Output: Merek: Toyota
echo "Model: " . $mobil->getModel() . "\n"; // Output: Model: Camry
?>
```

2. Getter

Getter adalah metode yang digunakan untuk mengambil atau mendapatkan nilai dari properti objek. Dengan menggunakan getter, Anda dapat mengakses nilai properti tanpa langsung mengakses properti itu sendiri. Getter biasanya diberi awalan `get` diikuti dengan nama properti.

Contoh Penggunaan Getter:

```php
<?php
class Mobil {
    private $merek;

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

    // Getter untuk merek
    public function getMerek() {
        return $this->merek;
    }
}

// Membuat objek Mobil
$mobil = new Mobil("Honda");
echo "Merek: " . $mobil->getMerek() . "\n"; // Output: Merek: Honda
?>
```

3. Setter

Setter adalah metode yang digunakan untuk mengatur atau memodifikasi nilai dari properti objek. Anda dapat menambahkan logika validasi dalam setter untuk memastikan bahwa nilai yang diberikan sesuai dengan kriteria tertentu. Setter biasanya diberi awalan `set` diikuti dengan nama properti.

Contoh Penggunaan Setter:

```php
<?php
class Mobil {
    private $merek;

    public function __construct($merek) {
        $this->setMerek($merek);
    }
```

```php
    // Getter untuk merek
    public function getMerek() {
        return $this->merek;
    }

    // Setter untuk merek
    public function setMerek($merek) {
        if (is_string($merek) && !empty($merek)) {
            $this->merek = $merek;
        } else {
            throw new Exception("Merek harus berupa string yang tidak
kosong.");
        }
    }
}

// Membuat objek Mobil
$mobil = new Mobil("Suzuki");
echo "Merek: " . $mobil->getMerek() . "\n"; // Output: Merek: Suzuki

// Mengubah merek menggunakan setter
try {
    $mobil->setMerek("Nissan");
    echo "Merek setelah diubah: " . $mobil->getMerek() . "\n"; // Output:
Merek setelah diubah: Nissan

    // Mengubah merek dengan nilai tidak valid
    // $mobil->setMerek(""); // Akan menghasilkan Exception
} catch (Exception $e) {
    echo "Error: " . $e->getMessage(); // Output: Merek harus berupa string
yang tidak kosong.
}
?>
```

Rangkuman Penggunaan:

1. **Constructor**: Digunakan untuk menginisialisasi properti saat objek dibuat. Memastikan objek sudah dalam keadaan siap digunakan.
2. **Getter**: Memberikan cara untuk mengakses nilai properti secara aman tanpa mengubah nilai tersebut. Memudahkan dalam pembacaan data dari objek.
3. **Setter**: Digunakan untuk memodifikasi nilai properti dengan validasi, menjaga integritas data. Memastikan bahwa nilai yang dimasukkan sesuai dengan kriteria yang diinginkan.

Kesimpulan

Menggunakan setter, getter, dan constructor dalam PHP memungkinkan Anda untuk menjaga encapsulation, validasi data, dan inisialisasi yang mudah saat bekerja dengan

objek. Dengan mengikuti praktik ini, Anda dapat meningkatkan keterbacaan, pemeliharaan, dan keamanan kode Anda dalam aplikasi berbasis OOP.