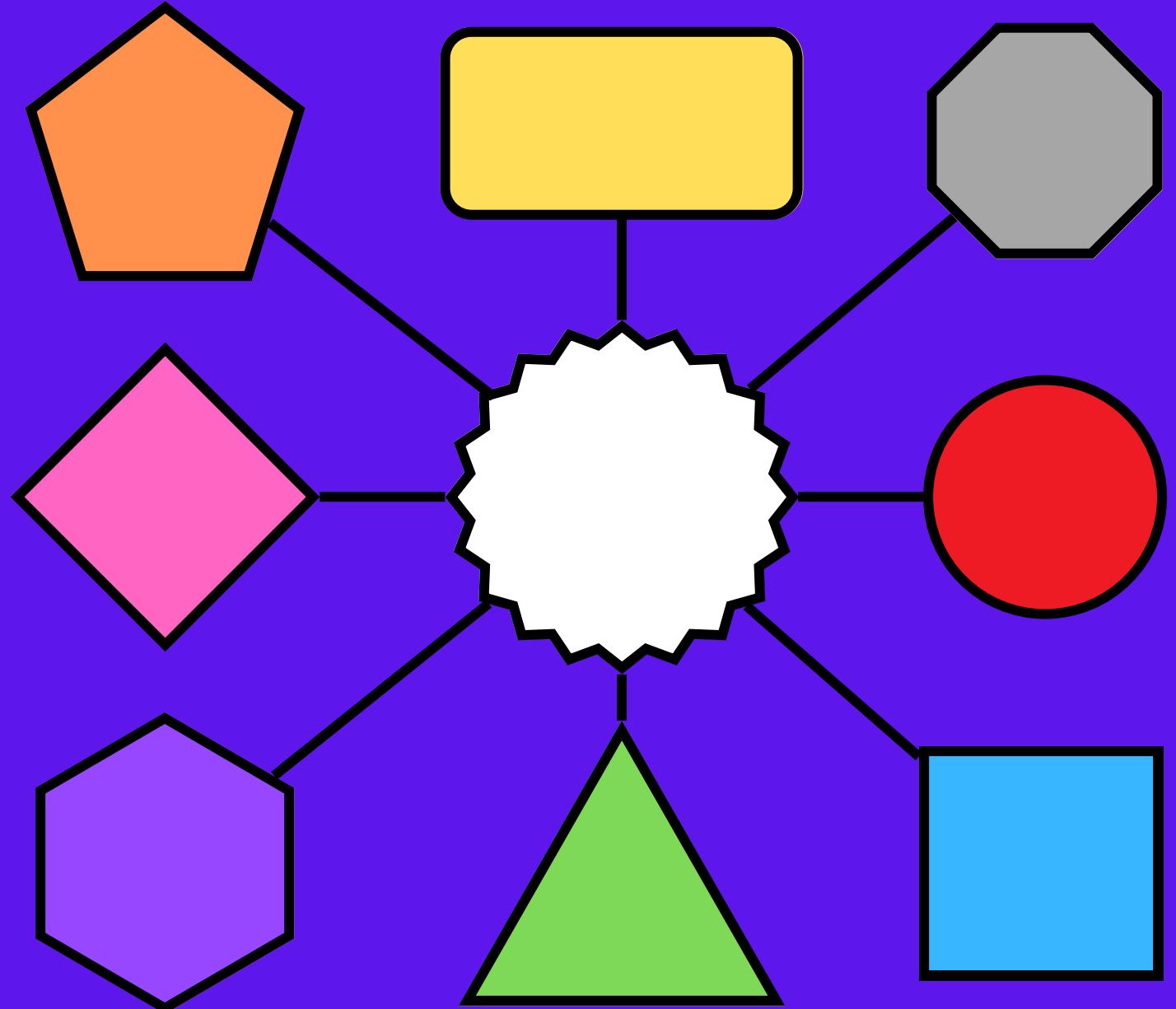


Jefri Maruli

Back-End Development

Materi 3



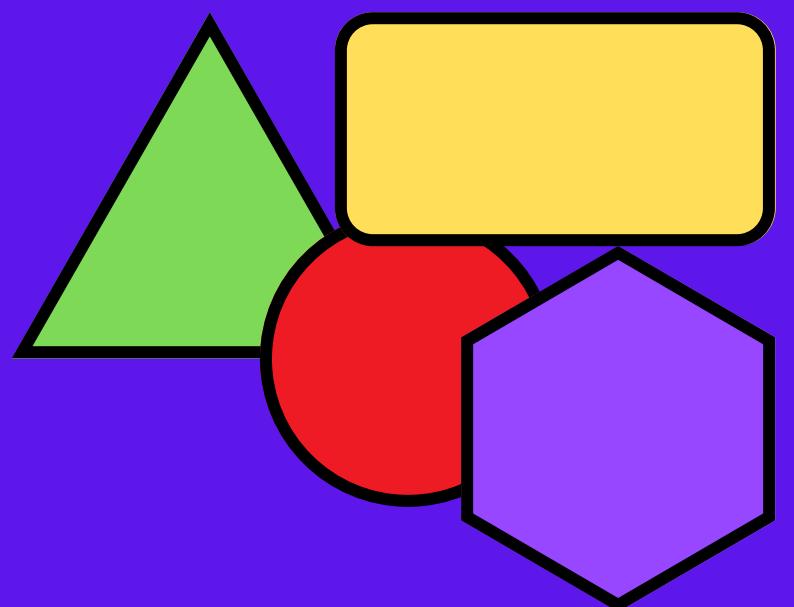
Agenda Belajar

- **Object Oriented Programming (OOP)**
 - **Class & Object**
 - **Properties**
 - **Method / Function**
 - **Static Method & Static Properties**
 - **Constructor & Destructor**
 - **Inheritance (Pewarisan)**
 - **Access Modifier / Encapsulation**
 - **Overriding & Overloading**
 - **Chaining Method**
 - **Abstract Class**
 - **Interface**
 - **Polymorphism (Coming soon)**
 - **Traits (Coming soon)**
 - **Namespace (Coming soon)**



Object Oriented Programming

Object Oriented Programming (OOP) merupakan sebuah paradigma atau metode pemrograman yang orientasinya terletak pada suatu objek yang tujuannya untuk memudahkan developer dalam pembuatan aplikasi.

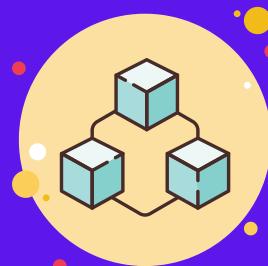


Alasan Menggunakan OOP



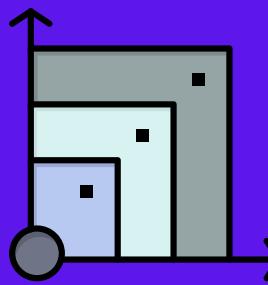
Reusable

Membuat baris kode menjadi lebih rapih dan terstruktur atau kita tidak perlu membuat baris kode yang sifatnya sama ketika membuat file baris kode yang lain.



Parallel Development

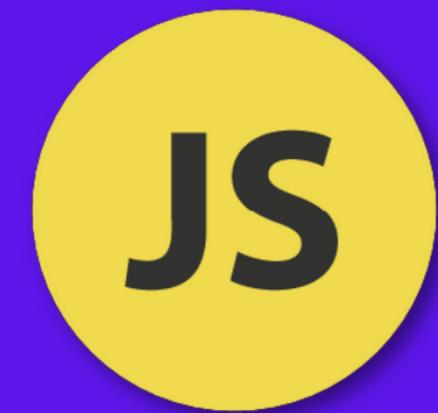
Memudahkan programmer ketika bekerja sama dengan tim saat membangun sebuah komponen secara individual, kemudian komponen yang telah dibangun dari setiap programmer dapat digabung menjadi satu yang tentu saja lebih menghemat waktu.



Scalability

Dengan konsep OOP akan lebih mudah untuk membangun aplikasi dalam skala besar atau rumit dan lebih mudah saat melakukan perbaikan (maintenance) karena sistem penulisan baris kode lebih terstruktur.

Bahasa Pemrograman Bisa Menggunakan Paradigma OOP



JavaScript



Python



Java



PHP



Ruby

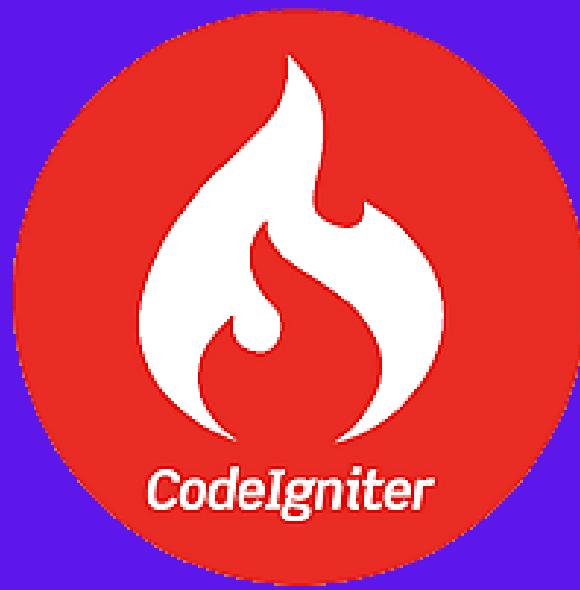


C#

Framework PHP Menggunakan Paradigma OOP



Laravel



CodeIgniter



Yii



Cake PHP

Perbandingan Paradigma Pemrograman

● ● ● Procedural Programming (PP)

```
1 <?php
2
3 $numberA = 1;
4 $numberB = 1;
5
6 $addition = $numberA + $numberB;
7
8 echo $addition;
9
10 $numberA = 3;
11 $numberB = 4;
12
13 $substract = $numberA - $numberB;
14
15 echo $substract;
16
17 $numberA = 4;
18 $numberB = 2;
19
20 $multiply = $numberA * $numberB;
21
22 echo $multiply;
```

● ● ● Functional Programming (FP)

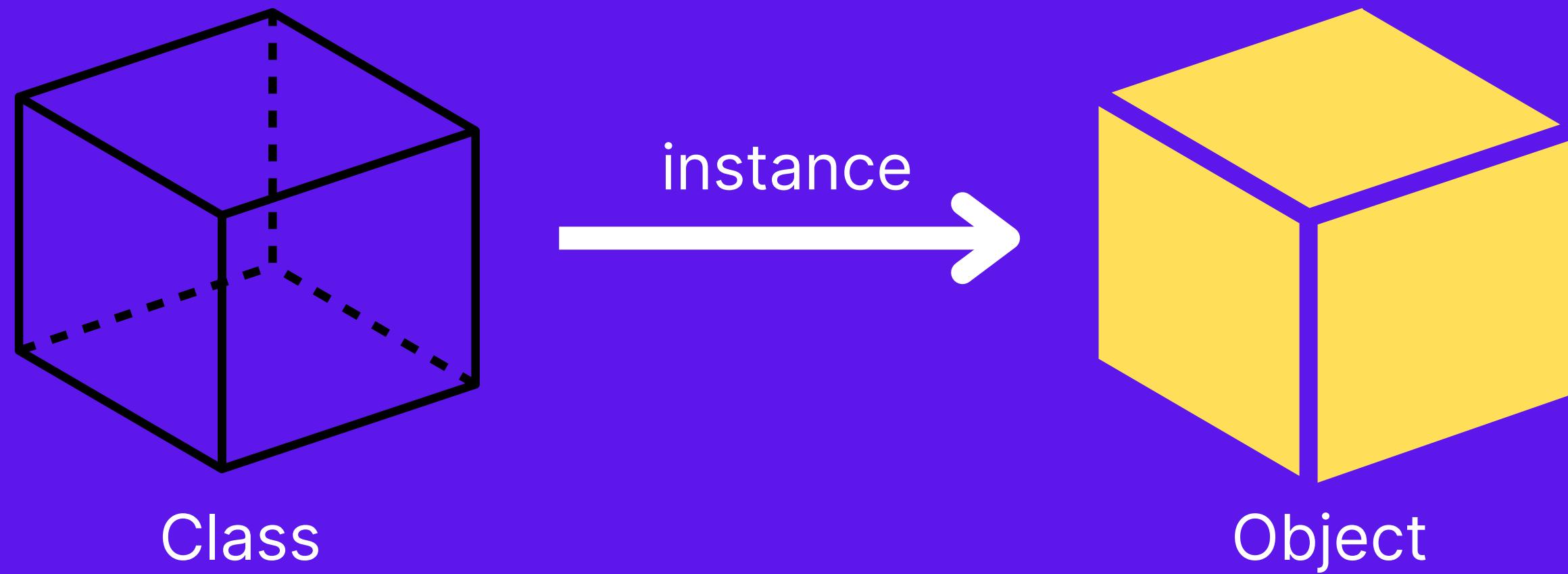
```
1 <?php
2
3 function addition($numberA, $numberB)
4 {
5     return $numberA + $numberB;
6 }
7
8 function subtract($numberA, $numberB)
9 {
10    return $numberA - $numberB;
11 }
12
13 function multiply($numberA, $numberB)
14 {
15    return $numberA * $numberB;
16 }
17
18 echo addition(1, 1) . PHP_EOL;
19 echo subtract(3, 4) . PHP_EOL;
20 echo multiply(4, 2) . PHP_EOL;
```

● ● ● Object Oriented Programming (OOP)

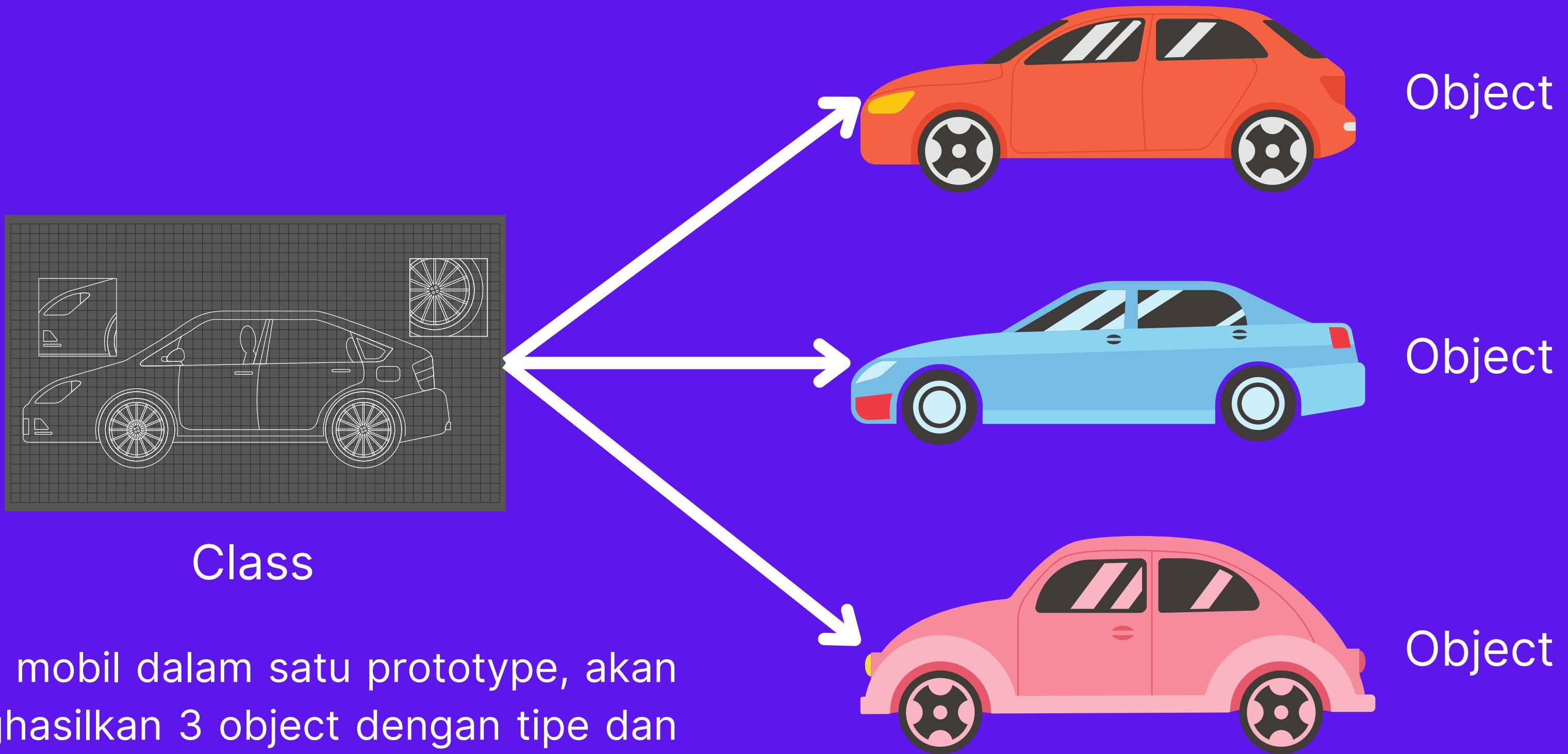
```
1 <?php
2
3 class Aritmetic
4 {
5     public function __construct($a, $b)
6     {
7         $this->a = $a;
8         $this->b = $b;
9     }
10
11    public function addition()
12    {
13        return $this->a + $this->b . PHP_EOL;
14    }
15
16    public function substraction()
17    {
18        return $this->a - $this->b . PHP_EOL;
19    }
20
21    public function multiplication()
22    {
23        return $this->a * $this->b . PHP_EOL;
24    }
25 }
26
27 $aritmetic = new Aritmetic(10, 5);
28 echo $aritmetic->addition();
29 echo $aritmetic->substraction();
30 echo $aritmetic->multiplication();
```

Class dan Object

Class adalah sebuah blueprint atau kerangka dari sebuah object dan object adalah sebuah instance atau perwujudan dari sebuah class.



Analogi Class dan Object



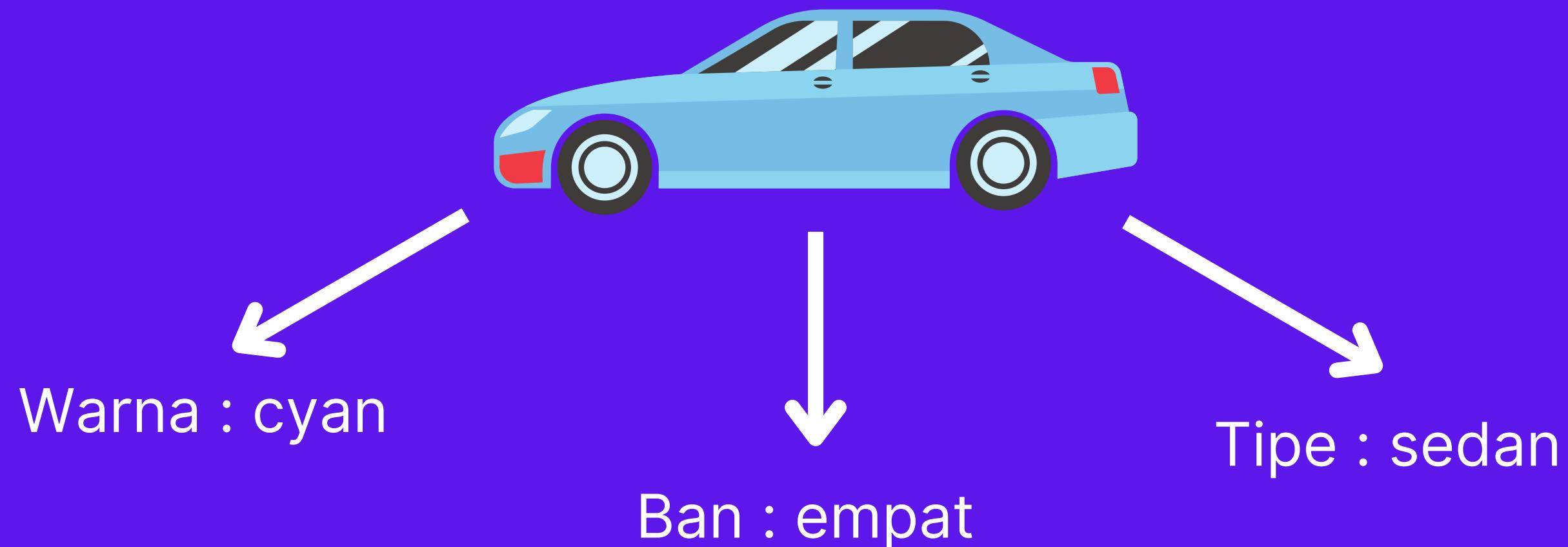
Implementasi Class dan Object

```
● ● ●  
1 <?php  
2  
3 class Car  
4 {  
5  
6 }  
7  
8 $car = new Car();
```

Untuk membuat class **Car** dengan kata kunci class dan ditambah dengan nama kelasnya dengan huruf kapital di awalanya, kemudian untuk membuat object (instance) dengan kata kunci "new" dan ditambah dengan nama classnya.

Properties / Attributes

Properties atau disebut dengan atribut atau field merupakan sebuah identitas yang menentukan spesifik dari sebuah objek yang ditampung dalam sebuah variable atau konstanta.



Contoh Properties / Attributes



```
1 <?php  
2  
3 class Car  
4 {  
5     var $color;  
6     var $type;  
7     var $tires;  
8 }  
9  
10 $car = new Car();  
11 var_dump($car);
```

```
object(Car)#1 (3) { ["color"]=> NULL ["type"]=> NULL ["tires"]=> NULL }
```

```
<?php  
  
class Car  
{  
    var $color;  
    var $color;  
}
```

Membuat property pada object bisa menggunakan kata kunci var dan ditambah dengan nama property yang akan dibuat, maka tipe data yang dihasilkan saat melakukan debugging adalah tipe data object. Pendefinisian property tidak bisa duplikasi atau tidak boleh sama.

Implementasi Properties



```
1 <?php  
2  
3 class Car  
4 {  
5     var $color;  
6     var $type;  
7     var $tires;  
8 }  
9  
10 $car = new Car();  
11 $car->color = 'cyan';  
12 $car->type = 'sedan';  
13 $car->tires = 4;  
14  
15 echo "Warna mobil: {$car->color}<br>";  
16 echo "Tipe mobil: {$car->type}<br>";  
17 echo "Jumlah ban: {$car->tires}<br>";
```

Untuk menampilkan output dari sebuah property menggunakan tanda panah "->" setelah object dibuat atau instance dari sebuah class

Default Properties Value



```
1 <?php  
2  
3 class Car  
4 {  
5     var $color = 'cyan';  
6     var $type = 'sedan';  
7     var $tires = 4;  
8 }  
9  
10 $car = new Car();  
11  
12 echo "Warna mobil: {$car→color}";  
13 echo "Tipe mobil: {$car→type}";  
14 echo "Jumlah ban: {$car→tires}";
```

Properties bisa menggunakan default properties values yang fungsinya sama seperti variable.

Type Declaration Properties / Typed Properties



```
1 <?php  
2  
3 class Car  
4 {  
5     var string $name = 'Mobil';  
6     var int $tires = 4;  
7     var string $type = 'sedan';  
8 }  
9  
10 $car = new Car();  
11 var_dump($car);
```

Secara default PHP sudah menentukan tipe data berdasarkan value yang sudah diinput (Dynamic Typing) misal: jika masukkan nilai berupa string, maka tipe data dari sebuah property akan menghasilkan bertipe string. Properties type declaration berfungsi untuk menentukan tipe data secara eksplisit (Static Typing), artinya tipe data yang sudah ditentukan tidak bisa diubah oleh tipe valuenya.

Jenis Type Declaration Properties

- Scalar Types
 - **int**
 - **string**
 - **float**
 - **boolean**
- Compound Types
 - **array**
 - **iterable**
 - **object**
- Nullable Types
 - **? (null)**
- Other Types
 - **class**
 - **interface**
 - **self**
 - **parent**



Implementasi Properties Type Declaration



```
1 <?php
2
3 class Car
4 {
5     var string $color = 'cyan';
6     var string $type = 'sedan';
7     var int $tires = 4;
8     var bool $isRunning = false;
9 }
10
11 $car = new Car();
12 echo "Mobil berwarna {$car→color} <br>";
13 echo "Mobil berjenis {$car→type} <br>";
14 echo "Mobil memiliki {$car→tires} ban <br>";
15 if ($car→isRunning) {
16     echo "Mobil sedang berjalan <br>";
17 } else {
18     echo "Mobil sedang berhenti <br>";
19 }
```

Mobil berwarna cyan
Mobil berjenis sedan
Mobil memiliki 4 ban
Mobil sedang berhenti

Kesalahan Type Declaration Properties



```
1 <?php  
2  
3 class Car  
4 {  
5     var int $tires = 'empat';  
6 }
```

Jika mengubah value dari property yang sifatnya statis atau yang sudah ditentukan, maka akan terjadi error "Default value for property of type it can only be int in". Artinya nilai integer saja yang boleh di terima oleh property tersebut.

Fatal error: Default value for property of type int can only be int in

Alternatif Type Declaration Properties

```
1 <?php
2
3 class Car
4 {
5     var ?string $owner = null;
6     var array $doors;
7     var iterable $wheels;
8 }
9
10 $car = new Car();
11
12 $car→doors = ['depan-kiri', 'depan-kanan', 'belakang-kiri', 'belakang-kanan'];
13 $car→wheels = ['depan-kiri', 'depan-kanan', 'belakang-kiri', 'belakang-kanan'];
14
15 echo "Pemilik mobil: {$car→owner}<br>";
16
17 echo "Pintu mobil: <br>";
18 foreach ($car→doors as $door) {
19     echo "- $door <br>";
20 }
21
22 echo "Ban mobil: <br>";
23 foreach ($car→wheels as $wheel) {
24     echo "- $wheel <br>";
25 }
```

Pemilik mobil:

Pintu mobil:

- depan-kiri

- depan-kanan

- belakang-kiri

- belakang-kanan

Ban mobil:

- depan-kiri

- depan-kanan

- belakang-kiri

- belakang-kanan

Nullable Type Properties



```
1 <?php
2
3 class Car
4 {
5     var ?string $owner = null;
6 }
7
8 $car = new Car();
9 $car→owner = 'John';
10
11 echo "Owner: $car→owner";
```

Nullable type (?) properties berfungsi untuk memberikan nilai yang bersifat null atau kosong jika tidak ada data yang dimasukkan ke sebuah property.

Nullable Type Properties



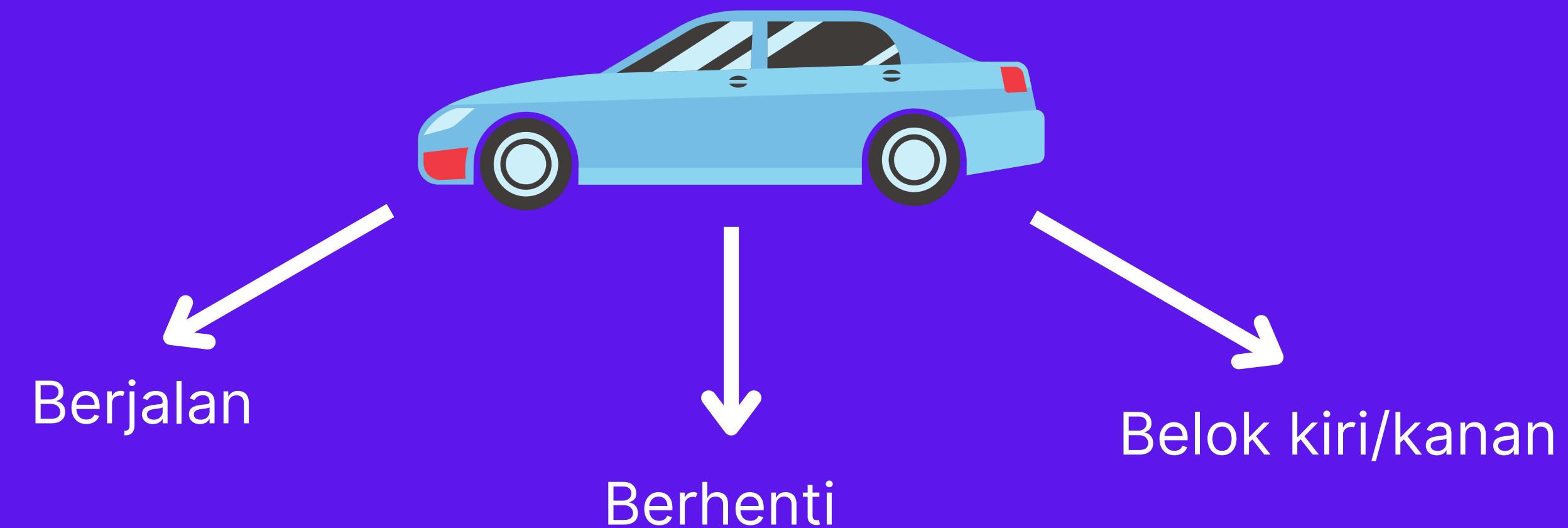
```
1 <?php
2
3 class Car
4 {
5     var string $owner = null;
6 }
7
8 $car = new Car();
9 $car→owner = 'John';
10
11 echo "Owner: $car→owner";
```

Jika tidak menggunakan nullable types maka akan menyebabkan error, karena property tidak bisa menampung nilai null tanpa menggunakan nullable types, meskipun nilainya sudah diisi setelah instance pada sebuah object.

Fatal error: Default value for property of type string may not be null. Use the nullable type ?string to allow null default value

Method / Function

Method atau yang dikenal sebagai function adalah sebuah perilaku atau aksi dari sebuah class dan object.



Contoh Method / Function

```
● ● ●  
1 <?php  
2  
3 class Car  
4 {  
5     function move()  
6     {  
7         return 'Mobil bergerak';  
8     }  
9  
10    function stop()  
11    {  
12        return 'Mobil berhenti';  
13    }  
14  
15    function turnLeft()  
16    {  
17        return 'Mobil belok kiri';  
18    }  
19  
20    function turnRight()  
21    {  
22        return 'Mobil belok kanan';  
23    }  
24 }
```

Membuat method menggunakan keyword function kemudian dilanjutkan dengan nama methodnya.

Implementasi Method / Function

```
1 <?php
2
3 class Car
4 {
5     function move()
6     {
7         return 'Mobil bergerak';
8     }
9
10    function stop()
11    {
12        return 'Mobil berhenti';
13    }
14
15    function turnLeft()
16    {
17        return 'Mobil belok kiri';
18    }
19
20    function turnRight()
21    {
22        return 'Mobil belok kanan';
23    }
24 }
25
26 $car = new car();
27 echo $car->move();
28 echo $car->stop();
29 echo $car->turnLeft();
30 echo $car->turnRight();
```

Untuk menampilkan output dari method hampir sama dengan cara menampilkan output dari property, perbedaanya adalah method bisa menyisipkan parameter berupa nilai yang akan di proses oleh method itu sendiri.

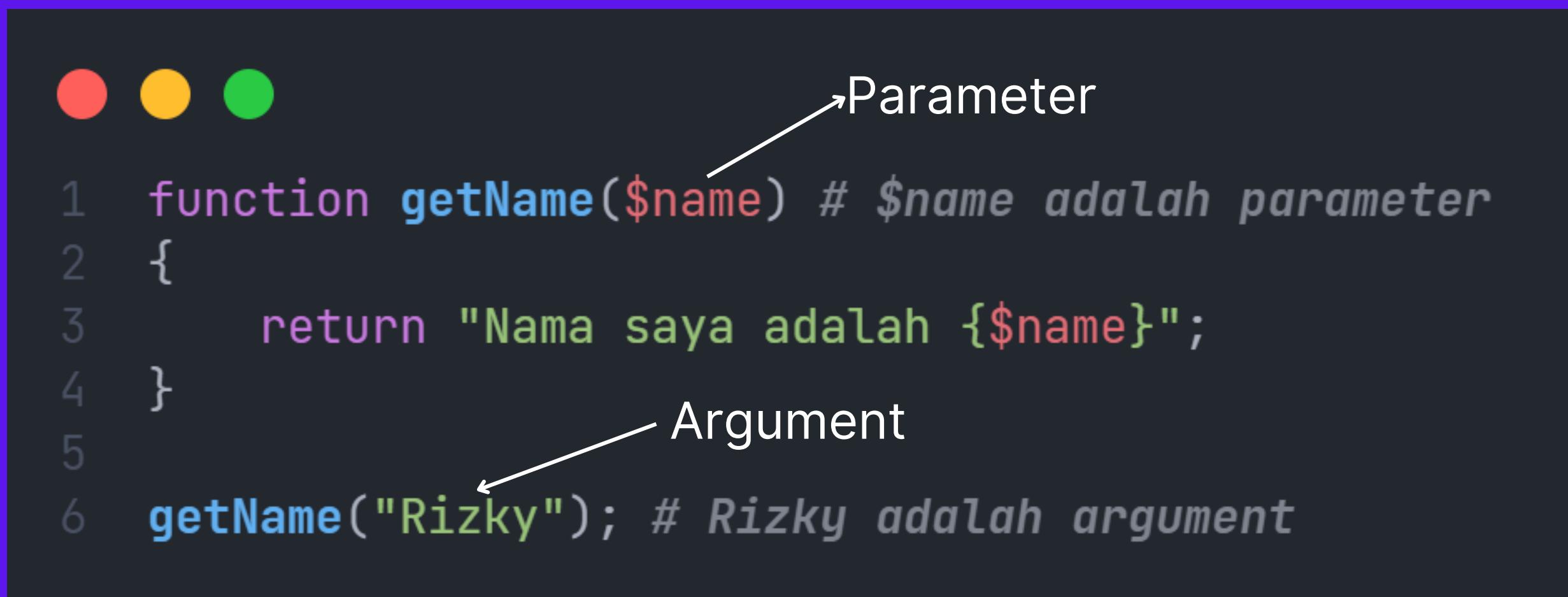
Implementasi Dynamic Object Oriented Programming

```
1 <?php
2
3 class Car
4 {
5     var $tires = 4;
6     var $color = "cyan";
7     var $type = "sedan";
8
9     function move()
10    {
11        return "Mobil bergerak";
12    }
13
14    function stop()
15    {
16        return "Mobil berhenti";
17    }
18
19    function turnLeft()
20    {
21        return "Mobil belok kiri";
22    }
23
24    function turnRight()
25    {
26        return "Mobil belok kanan";
27    }
28 }
29
30 $car = new Car();
31 echo "Mobil berwarna {$car->color}, memiliki {$car->tires}
32 ban dan bertipe {$car->type} <br>";
33 echo "Mobil bergerak dengan cara " . $car->move() . "<br>";
34 echo "Mobil berhenti dengan cara " . $car->stop() . "<br>";
35 echo "Mobil belok kiri dengan cara " . $car->turnLeft() . "<br>";
36 echo "Mobil belok kanan dengan cara " . $car->turnRight() . "<br>";
```

Secara umum standar penulisan sederhana pada Dynamic Object Oriented Programming terdiri dari beberapa komponen yaitu pendefinisian sebuah class, instance object dari sebuah class, pendefinisan dari prototype dan method.

Function Parameters & Function Arguments

Function parameter sebutan untuk nilai inputan fungsi pada saat fungsi itu di definisikan dan function arguments adalah sebutan untuk nilai inputan fungsi pada saat fungsi itu dipanggil.



```
● ● ●
1 function getName($name) # $name adalah parameter
2 {
3     return "Nama saya adalah {$name}";
4 }
5
6 getName("Rizky"); # Rizky adalah argument
```

The image shows a terminal window with three colored circles at the top left. The terminal displays the following PHP code:

```
function getName($name) # $name adalah parameter
{
    return "Nama saya adalah {$name}";
}
getName("Rizky"); # Rizky adalah argument
```

Annotations with arrows point to specific parts of the code:

- An arrow points from the text "Parameter" to the parameter declaration `$name` in the first line of the function definition.
- An arrow points from the text "Argument" to the argument value `"Rizky"` in the final line of the code.

Implementasi Function Parameters & Arguments

```
● ● ●  
1  <?php  
2  
3  class Person  
4  {  
5      function getName($name) // $name adalah parameter  
6      {  
7          return "Nama saya adalah {$name}";  
8      }  
9  }  
10  
11 $person = new Person();  
12 echo $person→getName("Rizky"); // Rizky adalah argument
```

Function parameter dan argument juga bisa digunakan dalam konsep Object Oriented Programming.

Multi Parameter & Argument



```
1  <?php
2
3  class Authentication
4  {
5      function login($username, $password)
6      {
7          if ($username == "admin" && $password == "admin") {
8              return true;
9          } else {
10              return false;
11          }
12      }
13  }
14
15 $auth = new Authentication();
16 if ($auth->login("admin", "admin")) {
17     echo "Login berhasil";
18 } else {
19     echo "Login gagal";
20 }
```

Implementasi argumen dan parameter bisa melebihi dari satu, dan urutan atau posisi parameternya dimulai dari sebelah paling kiri kemudian ditambahkan koma "," sebagai urutan parameter lanjutannya.

Login berhasil

Kesalahan Parameter & Argument



```
1 <?php
2
3 class Person
4 {
5     function getName($firstName, $lastName)
6     {
7         return "Nama saya adalah {$firstName} {$lastName}";
8     }
9 }
10
11 $person = new Person();
12 echo $person→getName("Rizky");
```

Jika jumlah parameter dan jumlah argument tidak sama, maka akan terjadi error "Too few arguments" artinya argument yang di masukkan lebih sedikit jumlahnya daripada jumlah parameter tersebut.

Fatal error: Uncaught ArgumentCountError: Too few arguments to function Person::getName(), 1 passed

Default Parameter Value



```
1 <?php  
2  
3 class Person  
4 {  
5     function getName($firstName, $lastName = "Kurniawan")  
6     {  
7         return "Nama saya adalah {$firstName} {$lastName}";  
8     }  
9 }  
10  
11 $person = new Person();  
12 echo $person→getName("Rizky");
```

Nama saya adalah Rizky Kurniawan

Cara mengatasi jika ada error seperti "Too few arguments" selain menyamakan parameter dan argument adalah dengan menambahkan default parameter dan value yang ingin di tambahkan

Implementasi Default Parameter Value

```
1 <?php
2
3 class Person
4 {
5     function getName($firstName, $lastName = "Kurniawan")
6     {
7         return "Nama saya adalah {$firstName} {$lastName}";
8     }
9 }
10
11 $person = new Person();
12 echo $person->getName("Rizky", "Rahman");
```

Nama saya adalah Rizky Rahman

Jika argument tidak diisi maka output yang akan dipanggil adalah value dari default parameter yang ditentukan, dan sebaliknya jika mengisi argument sesuai urutan maka hasil nilainya adalah output dari argument yang telah diisi.

```
1 <?php
2
3 class Person
4 {
5     function getName($firstName, $lastName = "Kurniawan")
6     {
7         return "Nama saya adalah {$firstName} {$lastName}";
8     }
9 }
10
11 $person = new Person();
12 echo $person->getName("Rizky");
```

Nama saya adalah Rizky Kurniawan

Type Declaration Parameter

```
● ● ●  
1 <?php  
2  
3 class Person  
4 {  
5     function getName(string $firstName)  
6     {  
7         return "Nama saya adalah {$firstName}";  
8     }  
9 }  
10  
11 $person = new Person();  
12 echo $person→getName("John");
```

Nama saya adalah John

Type declaration parameter memiliki fungsi yang sama dengan type declaration property.

Kesalahan Type Declaration Parameter



```
1 <?php
2
3 class Person
4 {
5     function getName(string $firstName)
6     {
7         return "Nama saya adalah {$firstName}";
8     }
9 }
10
11 $person = new Person();
12 echo $person->getName(['name' => 'Rizky']);
```

Jika argument yang di masukkan tidak sesuai dengan tipe parameter maka akan terjadi error seperti "Uncaught TypeError: Argument 1 passed to Person::getName() must be of the type string, array given". Artinya tipe data yang ditampung di parameter yang di perbolehkan hanya string sedangkan argument yang di input adalah tipe data array.

Fatal error: Uncaught TypeError: Argument 1 passed to Person::getName() must be of the type string, array given,

Alternatif Type Declaration Parameter



```
1 <?php
2
3 class Person
4 {
5     function getName(string $name)
6     {
7         return $name;
8     }
9
10    function getAge(int $age)
11    {
12        return $age;
13    }
14
15    function getWeight(float $weight)
16    {
17        return $weight;
18    }
19
20    function getMarried(bool $married)
21    {
22        return $married;
23    }
24
25    function getChildren(array $children)
26    {
27        return $children;
28    }
29
30    function getCar(object $car)
31    {
32        $car->brand = 'BMW';
33        $car->model = 'X5';
34        return $car;
35    }
36 }
```

Alternatif dari type declaration parameter memiliki beberapa tipe data seperti di gambar

Alternatif Type Declaration Parameter

```
● ● ●
1 <?php
2
3 class Person
4 {
5     function getName(string $name)
6     {
7         return $name;
8     }
9
10    function getAge(int $age)
11    {
12        return $age;
13    }
14
15    function getWeight(float $weight)
16    {
17        return $weight;
18    }
19
20    function getMarried(bool $married)
21    {
22        return $married;
23    }
24
25    function getChildren(array $children)
26    {
27        return $children;
28    }
29
30    function getCar(object $car)
31    {
32        $car->brand = 'BMW';
33        $car->model = 'X5';
34        return $car;
35    }
36 }
37
38 class Car
39 {
40     var $brand;
41     var $model;
42 }
43
44 $person = new Person();
45 // Output
46
47 echo "<pre>";
48 var_dump($person->getName('John')); // John
49 var_dump($person->getAge(25)); // 25
50 var_dump($person->getWeight(75.5)); // 75.5
51 var_dump($person->getMarried(true)); // 1
52 var_dump($person->getChildren(['John', 'Mary'])) . PHP_EOL; // Array
53 var_dump($person->getCar(new Car())); // Car Object
54 echo "</pre>";
```

```
string(4) "John"
int(25)
float(75.5)
bool(true)
array(2) {
    [0]=>
    string(4) "John"
    [1]=>
    string(4) "Mary"
}
object(Car)#2 (2) {
    ["brand"]=>
    string(3) "BMW"
    ["model"]=>
    string(2) "X5"
}
```

Implementasikan type declaration parameter hingga menghasilkan output seperti ini

Return Type Declaration Method



```
1 <?php
2
3 class Person
4 {
5     function getName(): string
6     {
7         return 'John Doe';
8     }
9 }
```

Setelah type declaration properties dan type declaration parameter, return type pada method juga mempunyai type declaration yang fungsinya juga menentukan tipe data dari nilai yang akan dikembalikan.

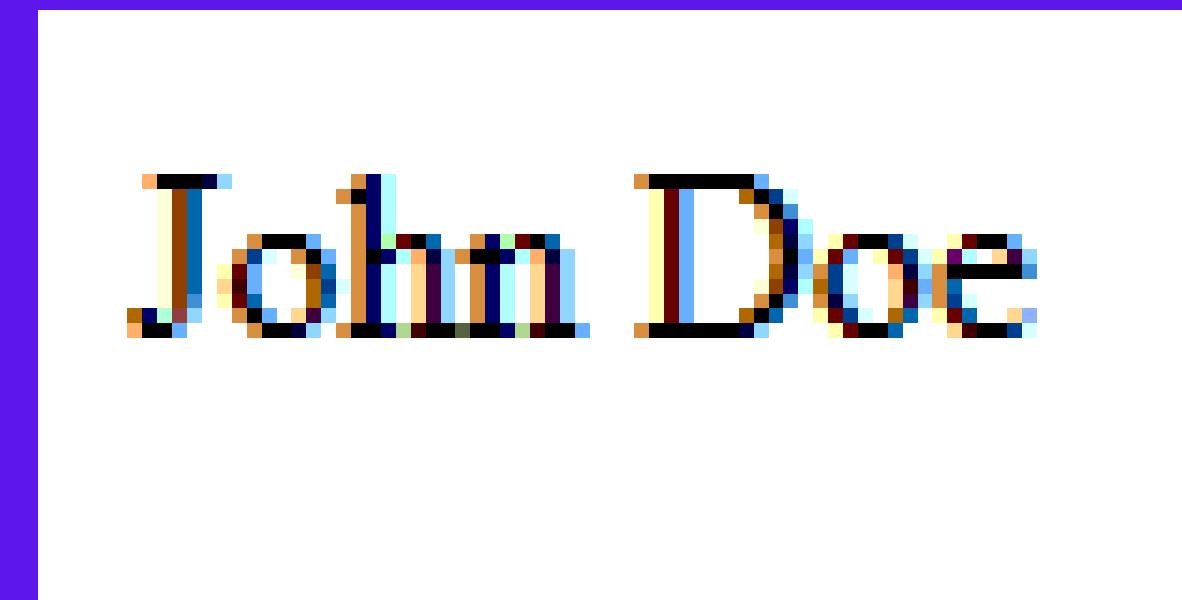
Jenis Return Type Declaration Method

- Scalar Types
 - **int**
 - **string**
 - **float**
 - **boolean**
- Compound Types
 - **array**
- Other Types
 - **void**



Implementasi Return Type Declaration Method

```
1 <?php  
2  
3 class Person  
4 {  
5     function getName(): string  
6     {  
7         return 'John Doe';  
8     }  
9 }  
10  
11 $person = new Person();  
12 echo $person->getName();  
13
```



Kesalahan Return Type Declaration Method

```
1 <?php
2
3 class Person
4 {
5     public function getName(): string
6     {
7         return [
8             'name' => 'John Doe'
9         ];
10    }
11 }
12
13 $person = new Person();
14 echo $person→getName();
```

"Uncaught TypeError: Return value of Person::getName() must be of the type string, array returned in" karena tipe data return valuenya array, sedangkan return type yang diminta adalah string.

Fatal error: Uncaught TypeError: Return value of Person::getName() must be of the type string, array returned in '

Kesalahan Return Type Declaration Method

```
1 <?php
2
3 class Person
4 {
5     public function getName(): string
6     {
7         return [
8             'name' => 'John Doe'
9         ];
10    }
11 }
12
13 $person = new Person();
14 echo $person→getName();
```

"Uncaught TypeError: Return value of Person::getName() must be of the type string, array returned in" karena tipe data return valuenya array, sedangkan return type yang diminta adalah string.

Fatal error: Uncaught TypeError: Return value of Person::getName() must be of the type string, array returned in '

Kesalahan Return Type Declaration Method



```
1 <?php
2
3 class Person
4 {
5     function getName(): string
6     {
7         return ['name' => 'John Doe'];
8     }
9 }
10
11 $person = new Person();
12 echo $person->getName();
```

Jika argument yang di masukkan tidak sesuai dengan tipe parameter maka akan terjadi error seperti "Uncaught TypeError: Argument 1 passed to Person::getName() must be of the type string, array given". Artinya tipe data yang ditampung di parameter yang di perbolehkan hanya string sedangkan argument yang di input adalah tipe data array.

Fatal error: Uncaught TypeError: Return value of Person::getName() must be of the type string, array returned in C:\xampp\php\phpStudy\WWW\index.php:12

Return Type Declaration Method Dengan Parameter

```
1 <?php
2
3 class Person
4 {
5     function getFirstName(string $firstName): string #static parameter
6     {
7         return $firstName;
8     }
9
10    function getLastName($lastName): string #dynamic parameter
11    {
12        return $lastName;
13    }
14 }
15
16 $person = new Person();
17 echo $person->getFirstName('John');
18 echo $person->getLastName('Doe');
```



Return type declaration bisa menggunakan parameter bersifat statis atau type declaration parameter dan juga bisa menggunakan parameter yang bersifat dinamis.

Alternatif Return Type Declaration Method

```
●●●  
1 <?php  
2  
3 class Person  
4 {  
5     function getName(string $firstName): string #static parameter  
6     {  
7         return $firstName;  
8     }  
9  
10    function getAge(int $age): int  
11    {  
12        return $age;  
13    }  
14  
15    function getWeight(float $weight): float  
16    {  
17        return $weight;  
18    }  
19  
20    function getMarried(bool $married): bool  
21    {  
22        return $married;  
23    }  
24  
25    function getChildren(array $children): array  
26    {  
27        return $children;  
28    }  
29  
30    function getCar(object $car): object  
31    {  
32        $car->brand = 'BMW';  
33        $car->model = 'X5';  
34        return $car;  
35    }  
36  
37    function sayHello(): void  
38    {  
39        echo "Hello";  
40    }  
41 }  
42  
43 class Car  
44 {  
45     var $brand;  
46     var $model;  
47 }  
48  
49 $person = new Person();  
50  
51 // Output  
52 echo "<pre>";  
53 var_dump($person->getName('John')); // John  
54 var_dump($person->getAge(25)); // 25  
55 var_dump($person->getWeight(75.5)); // 75.5  
56 var_dump($person->getMarried(true)); // 1  
57 var_dump($person->getChildren(['John', 'Mary'])); // Array  
58 var_dump($person->getCar(new Car())); // Car Object  
59 $person->sayHello(); // Hello  
60 echo "</pre>";  
61
```

```
string(4) "John"  
int(25)  
float(75.5)  
bool(true)  
array(2) {  
    [0]=>  
    string(4) "John"  
    [1]=>  
    string(4) "Mary"  
}  
object(Car)#2 (2) {  
    ["brand"]=>  
    string(3) "BMW"  
    ["model"]=>  
    string(2) "X5"  
}  
Hello
```

Implementasikan type declaration method hingga menghasilkan output seperti ini

Return Type Void Method



```
1 <?php  
2  
3 class Person  
4 {  
5     // return void type  
6     function getFirstName(): void  
7     {  
8         echo 'John';  
9     }  
10    // return string type  
11    function getLastNames(): string  
12    {  
13        return 'John';  
14    }  
15 }
```

Void merupakan sebuah sub-program yang tidak mengembalikan nilai, void hanya melakukan sesuatu. **Void = Do something**, sedangkan method tanpa kata “void” berarti harus mengembalikan nilai yang disebut dengan return, **return = answer something**.

Implementasi Return Type Void Method

```
1 <?php
2
3 class Person
4 {
5     // return void type
6     function getFirstName(): void
7     {
8         if ($this->getLastName() = 'Doe') {
9             echo 'John';
10        } else {
11            echo 'Jane';
12        }
13    }
14
15    // return string type
16    function getLastName(): string
17    {
18        return 'Doe';
19    }
20}
21
22 $person = new Person();
23 $person->getFirstName();
```



Implementasi return type void bisa menggunakan call function menggunakan this keyword.

This Keyword

```
● ● ●  
1 <?php  
2  
3 class Person  
4 {  
5     var string $name;  
6     var int $age;  
7  
8     function getPerson(): string  
9     {  
10         return "Name: {$this->name} and Age: {$this->age}";  
11     }  
12  
13     function setPerson(string $name, int $age): void  
14     {  
15         $this->name = $name;  
16         $this->age = $age;  
17     }  
18  
19     function call() : void  
20     {  
21         $this->setPerson('John', 25);  
22         echo $this->getPerson();  
23     }  
24 }  
25  
26 $person = new Person();  
27 $person->call();
```

Keyword `$this` adalah sebuah variabel yang digunakan untuk mengakses object seperti property atau method dalam lingkup class itu sendiri.

Analogi This Keyword



Keyword this bisa disebut kata ganti untuk menunjukkan sebuah benda (object) atau dengan kata "ini" orang lebih mudah mengingat daripada menyebut nama object tersebut.

Implementasi This Keyword

```
1 <?php
2
3 class Person
4 {
5     var string $name;
6     var int $age;
7
8     function getPerson(): string
9     {
10         return "Nama: {$this->name} dan Umur: {$this->age}";
11     }
12
13     function setPerson(string $name, int $age): void
14     {
15         $this->name = $name;
16         $this->age = $age;
17     }
18
19     function call() : void
20     {
21         $this->setPerson('John', 25);
22         echo $this->getPerson();
23     }
24 }
25
26 $person = new Person();
27 $person->call();
```

Nama: John dan Umur: 25

Class Constant



```
1 <?php  
2  
3 define('BUILD', 1.0);  
4  
5 class Box  
6 {  
7     const VERSION = 1.0;  
8 }
```

Class constant adalah konstanta yang berada di dalam class dimana konstanta memiliki sifat nilai yang sudah didefinisikan tidak bisa diubah.

Implementasi Class Constant

```
● ● ●  
1 <?php  
2  
3 define('APP_VERSION', '1.0.0');  
4  
5 class Database  
6 {  
7     const DBHOST = 'localhost';  
8     const DBUSER = 'root';  
9     const DBPASS = '';  
10    const DBNAME = 'oop';  
11 }  
12  
13 echo APP_VERSION . "<br>";  
14 echo Database::DBHOST . "<br>";  
15 echo Database::DBUSER . "<br>";  
16 echo Database::DBPASS . "<br>";  
17 echo Database::DBNAME . "<br>";
```

1.0.0
localhost
root
oop

Untuk pemanggilan class constant yaitu menggunakan nama classnya, kemudian menggunakan simbol ":" dan nama constantnya, karena sifat pemanggilan disebut sebagai metode static.

Perbedaan Constant dan Property

```
1 <?php  
2  
3 class Database  
4 {  
5     const DBHOST = 'localhost';  
6     const DBUSER = 'root';  
7     const DBPASS = '';  
8     const DBNAME = 'oop';  
9 }  
10  
11 echo Database::DBHOST . "<br>";  
12 echo Database::DBUSER . "<br>";  
13 echo Database::DBPASS . "<br>";  
14 echo Database::DBNAME . "<br>";
```

```
1 <?php  
2  
3 class Database  
4 {  
5     var string $host = 'localhost';  
6     var string $user = 'root';  
7     var string $password = '';  
8     var string $dbname = "oop";  
9 }  
10  
11 $db = new Database();  
12 echo $db->host . "<br>";  
13 echo $db->user . "<br>";  
14 echo $db->password . "<br>";  
15 echo $db->dbname . "<br>";
```

Perbedannya ketika menggunakan constant tidak perlu membuat object seperti property, karena cara mengakses property di global scope perlu instance object terlebih dahulu.

Self Keyword

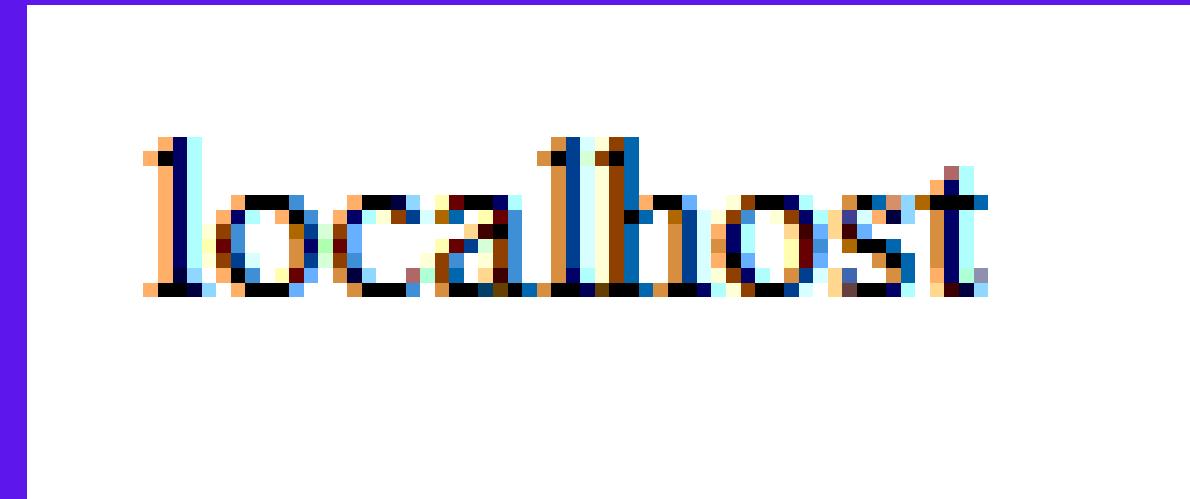


```
1 <?php
2
3 class Database
4 {
5     const DBHOST = 'localhost';
6     const DBUSER = 'root';
7     const DBPASS = '';
8     const DBNAME = 'oop';
9
10    function getHost(): void
11    {
12        echo self::DBHOST;
13    }
14 }
```

Self keyword hampir sama dengan this keyword yang fungsinya untuk memanggil object dalam lingkup classnya sendiri, yang membedakanya adalah self keyword digunakan untuk memanggil constant, property dan method yang bersifat static.

Implementasi Keyword

```
1 <?php  
2  
3 class Database  
4 {  
5     const DBHOST = 'localhost';  
6     const DBUSER = 'root';  
7     const DBPASS = '';  
8     const DBNAME = 'oop';  
9  
10    function getHost(): void  
11    {  
12        echo self::DBHOST;  
13    }  
14 }  
15  
16 $db = new Database();  
17 $db->getHost();
```



Self Keyword Dengan Property

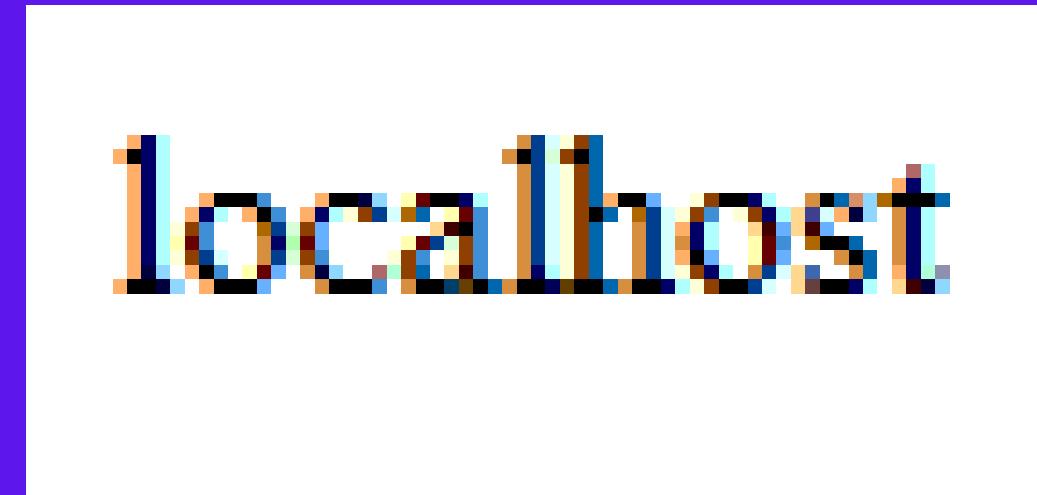
```
1 <?php
2
3 class Database
4 {
5     static string $host;
6     static string $username = 'root';
7     static string $dbpass = '';
8     static string $dbname = 'oop';
9
10    function getHost(): string
11    {
12        return self::$host;
13    }
14
15    function setHost(string $host): void
16    {
17        self::$host = $host;
18    }
19
20    function callSetHost(): void
21    {
22        $this->setHost('localhost');
23        echo $this->getHost();
24    }
25}
26
27 $db = new Database();
28 $db->callSetHost();
```



Self keyword tidak hanya untuk constant saja, tetapi self keyword juga bisa digunakan untuk properties dengan menambahkan keyword static.

Self Keyword Dengan Method

```
1 <?php
2
3 class Database
4 {
5     static string $host;
6     static string $username = 'root';
7     static string $dbpass = '';
8     static string $dbname = 'oop';
9
10    static function getHost(): string
11    {
12        return self::$host;
13    }
14
15    static function setHost(string $host): void
16    {
17        self::$host = $host;
18    }
19
20    static function callSetHost(): void
21    {
22        self::setHost('localhost');
23        echo self::getHost();
24    }
25 }
26
27 Database::callSetHost();
```



Self keyword juga bisa digunakan untuk method dengan menambahkan static pada method tersebut.

Kesalahan Self Keyword



```
1 <?php
2
3 class Database
4 {
5     static string $host;
6     static string $username = 'root';
7     static string $dbpass = '';
8     static string $dbname = 'oop';
9
10    static function getHost(): string
11    {
12        return self::$host;
13    }
14
15    static function setHost(string $host): void
16    {
17        self::$host = $host;
18    }
19
20    static function callSetHost(): void
21    {
22        $this->setHost('localhost');
23        echo $this->getHost();
24    }
25 }
26
27 Database::callSetHost();
```

: Uncaught Error: Using \$this when not in object context in

Jika menggunakan this ketika menggunakan static method akan menyebabkan error "Uncaught Error: Using \$this when not in object context in" keyword \$this hanya berlaku untuk object saja tidak untuk statis.

Kapan Menemukan Baris Kode Static Method dan Property?

```
1 <?php
2
3 namespace App\Http\Controllers\API;
4
5 use App\Models\Role;
6 use App\Models\User;
7 use App\Models\Member;
8 use App\Models\DoorHistory;
9 use Illuminate\Http\Request;
10 use App\Http\Controllers\Controller;
11 use App\Http\Resources\DoorHistoryResource;
12 use Illuminate\Support\Facades\Validator;
13 use Yajra\DataTables\DataTables;
14 use Illuminate\Support\Facades\DB;
15
16 class DoorHistoryController extends Controller
17 {
18     public function index(Request $request)
19     {
20         if ($auth()>user()>role_id != 1) {
21             return response()->json([
22                 'message' => 'forbidden access',
23                 'statusCode' => 403,
24                 'role' => 'member',
25             ], 403);
26         }
27
28         $data = DoorHistory::get();
29
30         return DoorHistoryResource::collection($data);
31     }
32 }
```



Ketika menggunakan framework laravel, akan menemukan banyak baris kode yang bersifat static terutama di method dan property.

Constructor

```
● ● ●  
1 <?php  
2  
3 class MotorCyle  
4 {  
5     var string $brand;  
6     var string $model;  
7     var string $color;  
8  
9     function __construct(string $brand, string $model, string $color)  
10    {  
11        $this->brand = $brand;  
12        $this->model = $model;  
13        $this->color = $color;  
14    }  
15 }
```

Constructor adalah method khusus yang dijalankan pertama kali secara otomatis pada saat proses instance (pembuatan object) pada sebuah object.

Implementasi Constructor



```
1 <?php
2
3 class MotorCyle
4 {
5     var string $brand;
6     var string $model;
7     var string $color;
8
9     function __construct(string $brand, string $model, string $color)
10    {
11        $this->brand = $brand;
12        $this->model = $model;
13        $this->color = $color;
14    }
15
16    function getAllProperties(): string
17    {
18        return "Brand: $this->brand, Model: $this->model, Color: $this->color";
19    }
20}
21
22 $myMotorCyle = new MotorCyle('Honda', 'CBR 150R', 'Red');
23 echo $myMotorCyle->getAllProperties();
```

Brand: Honda, Model: CBR 150R, Color: Red

Untuk menggunakan constructor dengan menulis nama function `__construct()`, kemudian cukup memasukan argument ke dalam parameter saat instance pada sebuah object.

Implementasi Constructor



```
1 <?php
2
3 class MotorCyle
4 {
5     var string $brand;
6     var string $model;
7     var string $color;
8
9     function __construct(string $brand, string $model, string $color)
10    {
11        $this->brand = $brand;
12        $this->model = $model;
13        $this->color = $color;
14    }
15
16    function getAllProperties(): string
17    {
18        return "Brand: $this->brand, Model: $this->model, Color: $this->color";
19    }
20}
21
22 $myMotorCyle = new MotorCyle('Honda', 'CBR 150R', 'Red');
23 echo $myMotorCyle->getAllProperties();
```

Brand: Honda, Model: CBR 150R, Color: Red

Untuk menggunakan constructor dengan menulis nama function `__construct()`, kemudian cukup memasukan argument ke dalam parameter saat instance pada sebuah object.

Perbandingan Menggunakan Constructor & Tidak

```
● ● ●  
1  <?php  
2  
3  class MotorCyle  
4  {  
5      var string $brand;  
6      var string $model;  
7      var string $color;  
8  
9      function __construct(string $brand, string $model, string $color)  
10     {  
11         $this->brand = $brand;  
12         $this->model = $model;  
13         $this->color = $color;  
14     }  
15  
16     function getAllProperties(): string  
17     {  
18         return "Brand: $this->brand, Model: $this->model, Color: $this->color";  
19     }  
20 }  
21  
22 $myMotorCyle = new MotorCyle('Honda', 'CBR 150R', 'Red');  
23 echo $myMotorCyle->getAllProperties();
```

```
● ● ●  
1  <?php  
2  
3  class MotorCyle  
4  {  
5      var string $brand;  
6      var string $model;  
7      var string $color;  
8  
9      function getBrand(string $brand): string  
10     {  
11         return $this->brand = $brand;  
12     }  
13  
14     function getModel(string $model): string  
15     {  
16         return $this->model = $model;  
17     }  
18  
19     function getColor(string $color): string  
20     {  
21         return $this->color = $color;  
22     }  
23  
24     function getAllProperties(): string  
25     {  
26         return "Brand: $this->brand, Model: $this->model, Color: $this->color";  
27     }  
28 }  
29  
30 $myMotorCyle = new MotorCyle();  
31 $myMotorCyle->getBrand('Honda');  
32 $myMotorCyle->getModel('CBR 150R');  
33 $myMotorCyle->getColor('Red');  
34 echo $myMotorCyle->getAllProperties();
```

Bisa di lihat perbandinganya menggunakan constructor dan tidak, sehingga tidak perlu menyebutkan function yang sifatnya repetitif, dan menyebabkan baris kode menjadi bloated (bengkak).

Alternatif Constructor

```
1 <?php
2
3 class MotorCyle
4 {
5     var string $brand;
6     var string $model;
7     var string $color;
8
9     public function __construct(string $brand, string $model, string $color)
10    {
11        $this->brand = $brand;
12        $this->model = $model;
13        $this->color = $color;
14        $this->getInfo();
15    }
16
17    public function getInfo(): void
18    {
19        echo "Brand: {$this->brand} <br>";
20        echo "Model: {$this->model} <br>";
21        echo "Color: {$this->color} <br>";
22    }
23 }
24
25 $myMotorCyle = new MotorCyle('Honda', 'CBR 150R', 'Red');
```

```
Brand: Honda
Model: CBR 150R
Color: Red
```

Constructor juga bisa memanggil method lainnya yang akan dijalankan, sehingga tidak perlu menyebutkan method tersebut saat instance pada sebuah object.

Destructor

```
1 <?php  
2  
3 class MotorCyle  
4 {  
5     function __destruct()  
6     {  
7         echo "Objek MotorCyle telah dihapus";  
8     }  
9 }
```

Jika constructor adalah method khusus yang dijalankan pertama kali secara otomatis pada saat proses instance sebuah object, maka destructor sebaliknya ialah method khusus yang akan dijalankan di paling akhir secara otomatis, ketika object dihapus dari memory atau objectnya sudah tidak digunakan lagi, biasanya digunakan untuk menutup koneksi database.

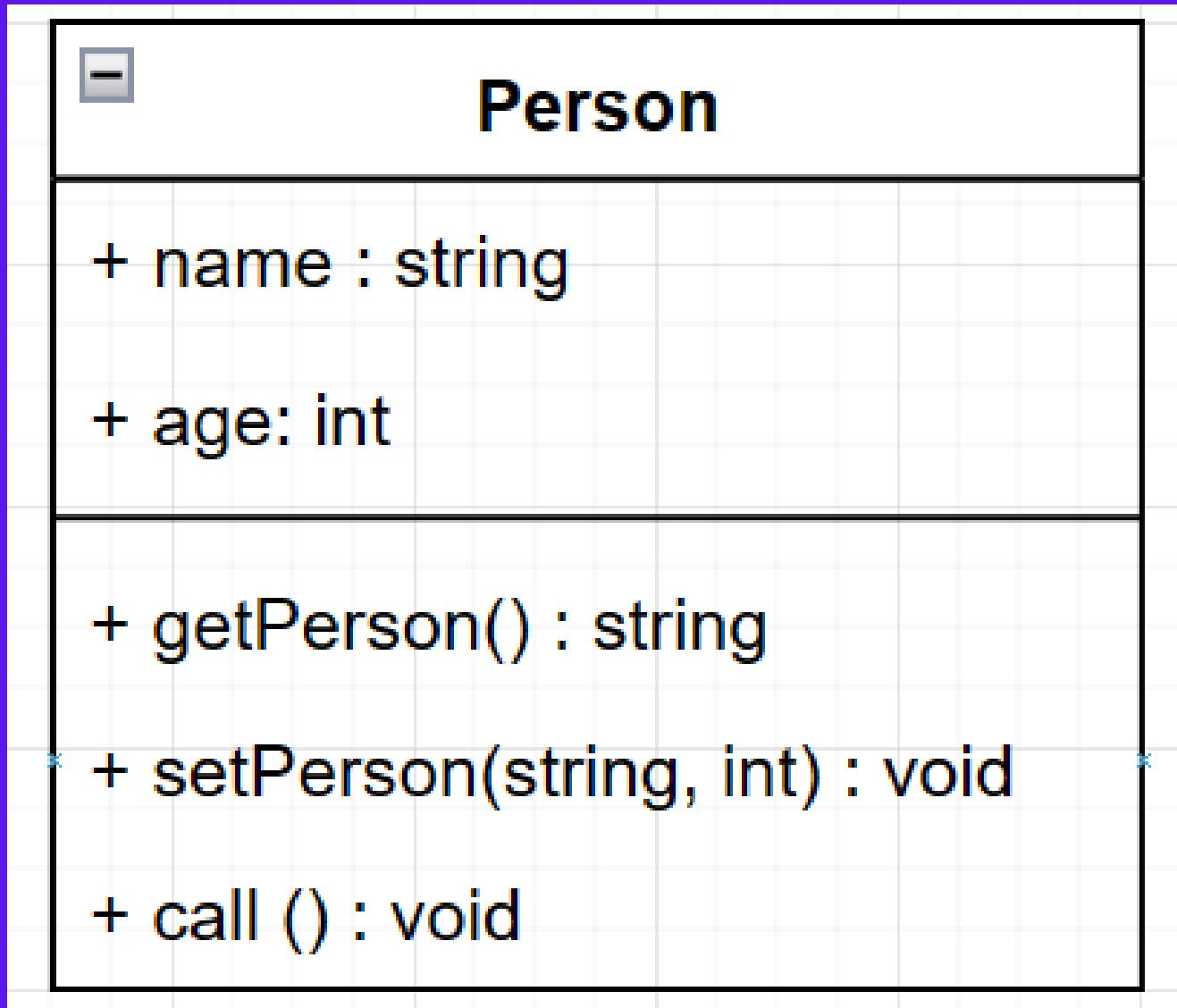
Implementasi Destructor

```
● ● ●
1 <?php
2
3 class MotorCyle
4 {
5     var string $brand;
6     var string $model;
7     var string $color;
8
9
10    function __construct(string $brand, string $model, string $color)
11    {
12        $this->brand = $brand;
13        $this->model = $model;
14        $this->color = $color;
15    }
16
17    function __destruct()
18    {
19        echo "Object $this->brand telah dihapus <br>";
20    }
21 }
22
23 $honda = new MotorCyle('Honda', 'CBR 150R', 'Red');
24 $yamaha = new MotorCyle('Yamaha', 'NMAX', 'Black');
```

Object Yamaha telah dihapus
Object Honda telah dihapus

Karena destructor akan menjalankan urutan yang paling terakhir maka yang akan muncul pertama adalah object "Yamaha" meskipun object "Honda" berada di posisi pertama saat pembuatan object (instance)

Static Object Oriented Programming



```
● ● ●
1 <?php
2
3 class Person
4 {
5     var string $name;
6     var int $age;
7
8     function getPerson(): string
9     {
10        return "Name: {$this->name} and Age: {$this->age}";
11    }
12
13    function setPerson(string $name, int $age): void
14    {
15        $this->name = $name;
16        $this->age = $age;
17    }
18
19    function call() : void
20    {
21        $this->setPerson('John', 25);
22        echo $this->getPerson();
23    }
24 }
25
26 $person = new Person();
27 $person->call();
```

Ketika sudah mengimplementasikan dynamic object oriented programming, maka dengan adanya static OOP akan lebih sesuai dengan perancangan class diagram pada UML (Unified Modeling Language).

Inheritance / Pewarisan



```
1 <?php  
2  
3 class Father  
4 {  
5 }  
6  
7 class Son extends Father  
8 {  
9 }
```

Inheritance atau pewarisan adalah konsep pemrograman dimana sebuah class dapat menurunkan property dan method yang dimilikinya kepada class lain yang akan diturunkan.



Jenis Inheritance

- Parent Class / Super Class
- Child Class / Sub Class



Parent Class / Super Class

```
● ● ●  
1 <?php  
2  
3 class Father  
4 {  
5     var string $name;  
6     var string $address;  
7  
8     function __construct(string $name, string $address)  
9     {  
10         $this->name = $name;  
11         $this->address = $address;  
12         $this->getProperties();  
13     }  
14  
15     function getProperties(): void  
16     {  
17         echo "Name: " . $this->name . ", Address: " . $this->address;  
18     }  
19 }  
20  
21 $father = new Father("John", "USA");
```

Parent class atau super class yang sering di implementasi pada slide sebelumnya, jika tidak ada keyword extend artinya class tersebut adalah sebuah class parent.

Child Class / Sub Class

```
● ● ●  
1  <?php  
2  
3  class Son extends Father  
4  {  
5      var int $age;  
6  
7      function __construct(string $name, string $address, int $age)  
8      {  
9          parent::__construct($name, $address);  
10         $this->age = $age;  
11         $this->getProperties();  
12     }  
13  
14     function getProperties(): void  
15     {  
16         echo "Name: " . $this->name . ", Address: " . $this->address . ", Age: " . $this->age;  
17     }  
18 }  
19  
20 $son = new Son("Dennis", "USA", 15);
```

Child class akan mendapatkan property dan method dari class turunannya atau dari class parent yang di turunkan, dan biasanya child class menggunakan keyword "extend".

Implementasi Inheritance

```
● ● ●  
1 <?php  
2  
3 class Vehicle  
4 {  
5     var $wheels;  
6  
7     function __construct(string $wheels)  
8     {  
9         $this->wheels = $wheels;  
10        $this->move();  
11        $this->getWheels();  
12    }  
13  
14    function move(): void  
15    {  
16        echo "Kendaraan Berjalan <br>";  
17    }  
18  
19    function getWheels(): void  
20    {  
21        echo "Jumlah Roda: " . $this->wheels . "<br>";  
22    }  
23 }  
24  
25 $vehicle = new Vehicle(4);  
26  
27 class Car extends Vehicle  
28 {  
29 }  
30  
31 $car = new Car(4);
```

Kendaraan Berjalan
Jumlah Roda: 4
Kendaraan Berjalan
Jumlah Roda: 4

Class **Vehicle** akan mewarisi property **wheels**, method **move()** dan method **getWheels()** kepada class child **Car**

Alternatif Inheritance

```
● ● ●  
1 class Car extends Vehicle  
2 {  
3     var int $wheels = 4;  
4  
5     function stop(): void  
6     {  
7         echo "Kendaraan Berhenti <br>";  
8     }  
9 }  
10  
11 $car = new Car(4);  
12 $car->stop();  
13
```

Kendaraan Berjalan

Jumlah Roda: 4

Kendaraan Berjalan

Jumlah Roda: 4

Kendaraan Berhenti

Selain mendapatkan warisan property dan method dari parent class, child juga bisa menentukan method atau propertinya sendiri seperti class pada umumnya.

Inheritance Pada Constructor

```
1 $vehicle = new Vehicle(4);
2
3 class Car extends Vehicle
4 {
5     var string $color;
6
7     function __construct(string $wheels, string $color)
8     {
9         $this->color = $color;
10        $this->stop();
11    }
12
13    function stop(): void
14    {
15        echo "Kendaraan Berhenti <br>";
16    }
17 }
18
19 $car = new Car(4, 'Red');
```

Kendaraan Berjalan
Jumlah Roda: 4
Kendaraan Berhenti

Secara default ketika child class menggunakan constructor dari parent class, tetapi child juga ingin mempunyai constructor dimana , child juga ingin mengeksekusi method yang ada di dalam child class itu sendiri.

Permasalahan Inheritance Pada Constructor



```
1 class Vehicle
2 {
3     var int $wheels;
4
5     function __construct(string $wheels)
6     {
7         $this->wheels = $wheels;
8         $this->move();
9         $this->getWheels();
10    }
```



```
1 $vehicle = new Vehicle(4);
2
3 class Car extends Vehicle
4 {
5     var string $color;
6
7     function __construct(string $wheels, string $color)
8     {
9         $this->color = $color;
10        $this->stop();
11    }
12
13    function stop(): void
14    {
15        echo "Kendaraan Berhenti <br>";
16    }
17 }
18
19 $car = new Car(4, 'Red');
```

Setelah membuat constructor pada child class, child class tidak akan lagi menjalankan constructor dari parent class, bagaimana cara mengatasinya?

Parent Class Constructor

```
● ● ●  
1 class Car extends Vehicle  
2 {  
3     var string $color;  
4  
5     function __construct(string $wheels, string $color)  
6     {  
7         $this->color = $color;  
8         parent::__construct($wheels);  
9         $this->stop();  
10    }  
11  
12    function stop(): void  
13    {  
14        echo "Kendaraan Berhenti <br>";  
15    }  
16 }  
17  
18 $car = new Car(4, 'Red');  
19
```

Kendaraan Berjalan
Jumlah Roda: 4
Kendaraan Berjalan
Jumlah Roda: 4
Kendaraan Berhenti

Solusi dari permasalahan diatas ketika ingin menjalankan constructor pada keduanya yaitu menggunakan parent class constructor.

Inheritance Class Berbeda File

```
● ● ●  
1 <?php  
2  
3 class Vehicle  
4 {  
5     var int $wheels;  
6  
7     function __construct(string $wheels)  
8     {  
9         $this->wheels = $wheels;  
10        $this->move();  
11        $this->getWheels();  
12    }  
13  
14    function move(): void  
15    {  
16        echo "Kendaraan Berjalan <br>";  
17    }  
18  
19    function getWheels(): void  
20    {  
21        echo "Jumlah Roda: " . $this->wheels . "<br>";  
22    }  
23 }  
24  
25 $vehicle = new Vehicle(4);
```

```
● ● ●  
1 <?php  
2  
3 require_once 'Vehicle.php';  
4  
5 class Car extends Vehicle  
6 {  
7     var string $color;  
8  
9     function __construct(string $wheels, string $color)  
10    {  
11        $this->color = $color;  
12        parent::__construct($wheels);  
13        $this->stop();  
14    }  
15  
16    function stop(): void  
17    {  
18        echo "Kendaraan Berhenti <br>";  
19    }  
20 }  
21  
22 $car = new Car(4, 'Red');
```

Konsep inheritance juga bisa dilakukan berada di file yang berbeda dengan cara menggunakan keyword "require_once" saat memanggil file lain, buatlah class parent dengan nama Vehicle.php dan child class dengan nama Car.php

Kapan Menemukan Baris Kode Inheritance?



```
1 <?php
2
3 defined('BASEPATH') or exit('No direct script access allowed');
4
5 class Home extends CI_Controller
6 {
7
8     public function __construct()
9     {
10         parent::__construct();
11         $this->load->model('HomeModel');
12     }
13
14     public function index()
15     {
16
17         $total = $this->HomeModel->countData('pendaftaran');
18         $data = [
19             'total' => $total,
20             'title' => 'Bootcamp Mardira 2022'
21         ];
22         $this->template->load('template/default', 'Home/index', $data);
23     }
}
```



```
1
2 class MemberController extends Controller
3 {
4
5     public function index(Request $request)
6     {
7
8         if (auth()->user()->role_id != 1) {
9             return response()->json([
10                 'message' => 'forbidden access',
11                 'statusCode' => 403,
12                 'role' => 'member',
13             ], 403);
14     }
15
16     $data = Member::get();
17
18     return MemberResource::collection($data);
19 }
```

Setelah mempelajari inheritance pada OOP, framework seperti CI dan Laravel juga menggunakan konsep inheritance pada OOP.

Access Modifier / Encapsulation / Visibility

```
1 <?php
2
3 class Person
4 {
5     public string $name;
6     public string $address;
7     public string $phone;
8
9     public function __construct(string $name, string $address, string $phone)
10    {
11        $this->name = $name;
12        $this->address = $address;
13        $this->phone = $phone;
14    }
15
16    protected function sayHello(string $name): void
17    {
18        echo "Halo $name, nama saya $this->name <br>";
19    }
20
21    private function sayGoodbye(string $name): void
22    {
23        echo "Selamat tinggal $name, sampai bertemu nanti <br>";
24    }
25 }
```

Access Modifier adalah kemampuan mengakses property, method dan constant dapat di akses darimana saja.

Access Level

```
3 class Person  
4 {  
5     var string $name; # public  
6 }
```

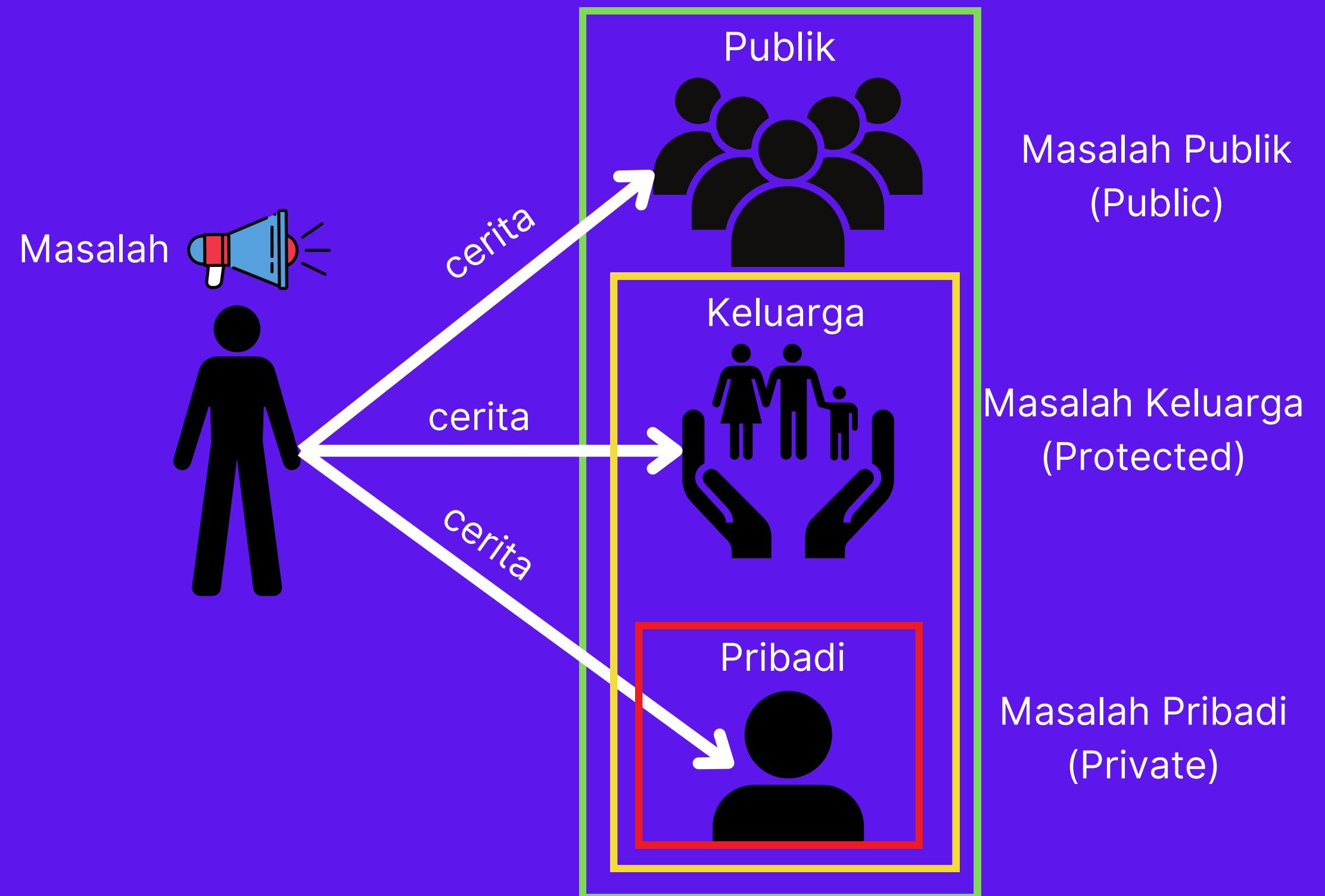


```
1 function setName(): void #public  
2 {  
3 }
```

Modifier	Class (Parent)	Child/Sub Class	Global / World
public	Ya	Ya	Ya
protected	Ya	Ya	Tidak
private	Ya	Tidak	Tidak

Secara default dengan menyebutkan keyword var pada properties, dan function untuk method pada umumnya adalah sifatnya public.

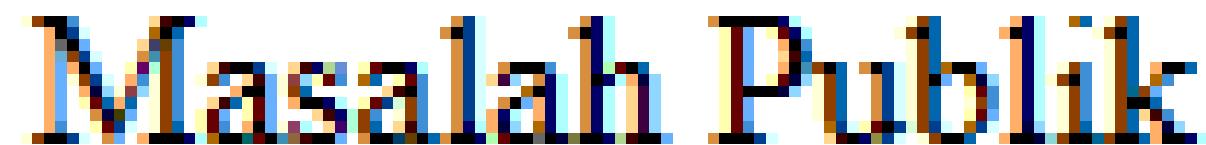
Analogi Access Modifier



ketika ada orang lagi mengalami masalah pada hidupnya, entah itu masalah pribadi, masalah keluarga, maupun masalah publik yang sering terjadi di kehidupan sehari-hari. Ketika masalah publik semua orang bisa tau tentang masalahnya, tetapi ketika ada masalah keluarga, hanya keluarga dan pribadi yang tau, dan masalah pribadi juga terkadang tidak bisa di ceritakan kesiapapun maupun ke keluarga ataupun di publikasi.

Public Access Modifier

```
1 <?php  
2  
3 class Person  
4 {  
5     public string $name;  
6  
7     public function publicIssue(): void  
8     {  
9         echo "Masalah Publik <br>";  
10    }  
11 }  
12  
13 $person = new Person();  
14 $person→publicIssue();
```



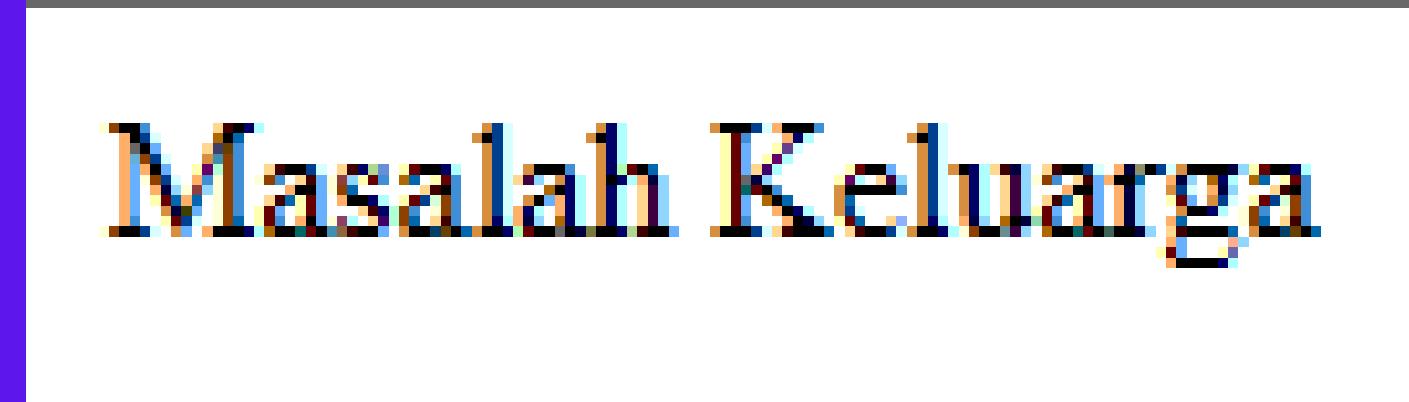
Public Access Modifier merupakan access modifier yang bisa diakses dimana saja, baik di dalam class, pada class turunan hingga di luar class sekalipun

Modifier	Class (Parent)	Child/Sub Class	Global / World
public	Ya	Ya	Ya

Protected Access Modifier



```
1 <?php
2
3 class Father
4 {
5     protected string $address;
6
7     public function __construct()
8     {
9         $this->familyIssue();
10    }
11
12     protected function familyIssue(): void
13    {
14        echo "Masalah Keluarga <br>";
15    }
16 }
17
18 class Son extends Father
19 {
20     protected string $name;
21
22     public function __construct()
23     {
24         parent::__construct();
25     }
26 }
27
28 $son = new Son();
```



Protected access modifier adalah access modifier yang hanya bisa di akses dari dalam class dan class turunan saja.

Modifier	Class (Parent)	Child/Sub Class	Global / World
protected	Ya	Ya	Tidak

Kesalahan Protected Access Modifier



```
1 <?php
2
3 class Father
4 {
5     protected string $address;
6
7     protected function familyIssue(): void
8     {
9         echo "Masalah Keluarga <br>";
10    }
11 }
12
13 $father = new Father();
14 $father→familyIssue();
```

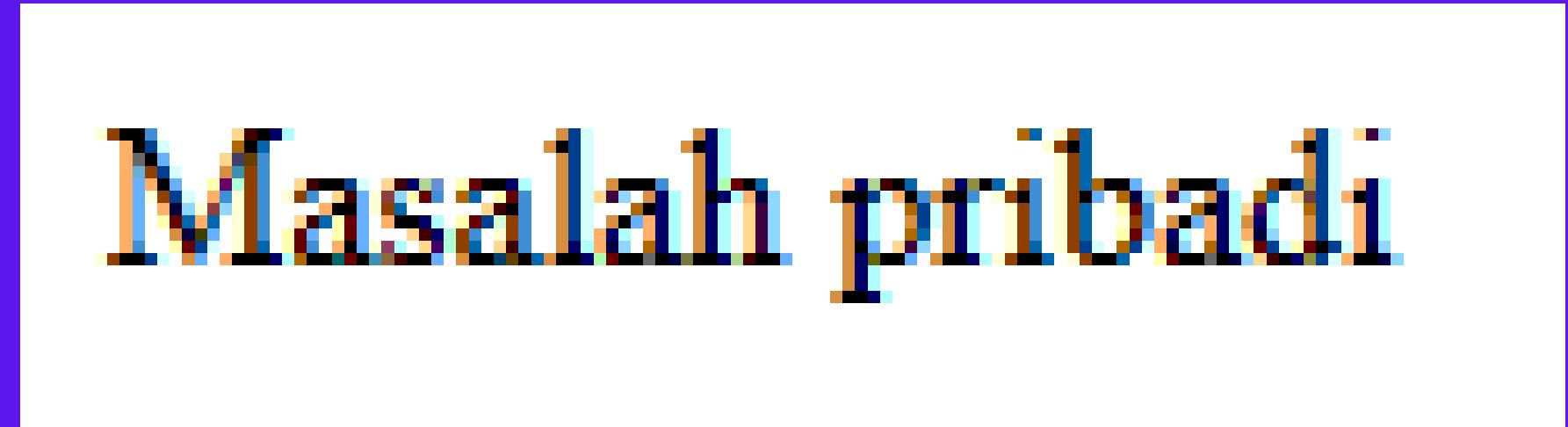
Jika global scope ingin mengakses method yang sifatnya protected, maka akan menyebabkan error.

Fatal error: Uncaught Error: Call to protected method Father::familyIssue() from context

Private Access Modifier



```
1 <?php
2
3 class Person
4 {
5     private int $money = 0;
6
7     function call(): void
8     {
9         if ($this->personalIssue()) {
10             echo "Masalah pribadi";
11         } else {
12             echo "Tidak ada masalah pribadi";
13         }
14     }
15
16     private function personalIssue(): bool
17     {
18         if ($this->money < 100) {
19             return true;
20         } else {
21             return false;
22         }
23     }
24 }
25
26 $person = new Person();
27 $person->call();
```



Private access modifier yaitu access modifier yang hanya bisa diakses didalam class saja.

Modifier	Class (Parent)	Child/Sub Class	Global / World
private	Ya	Tidak	Tidak

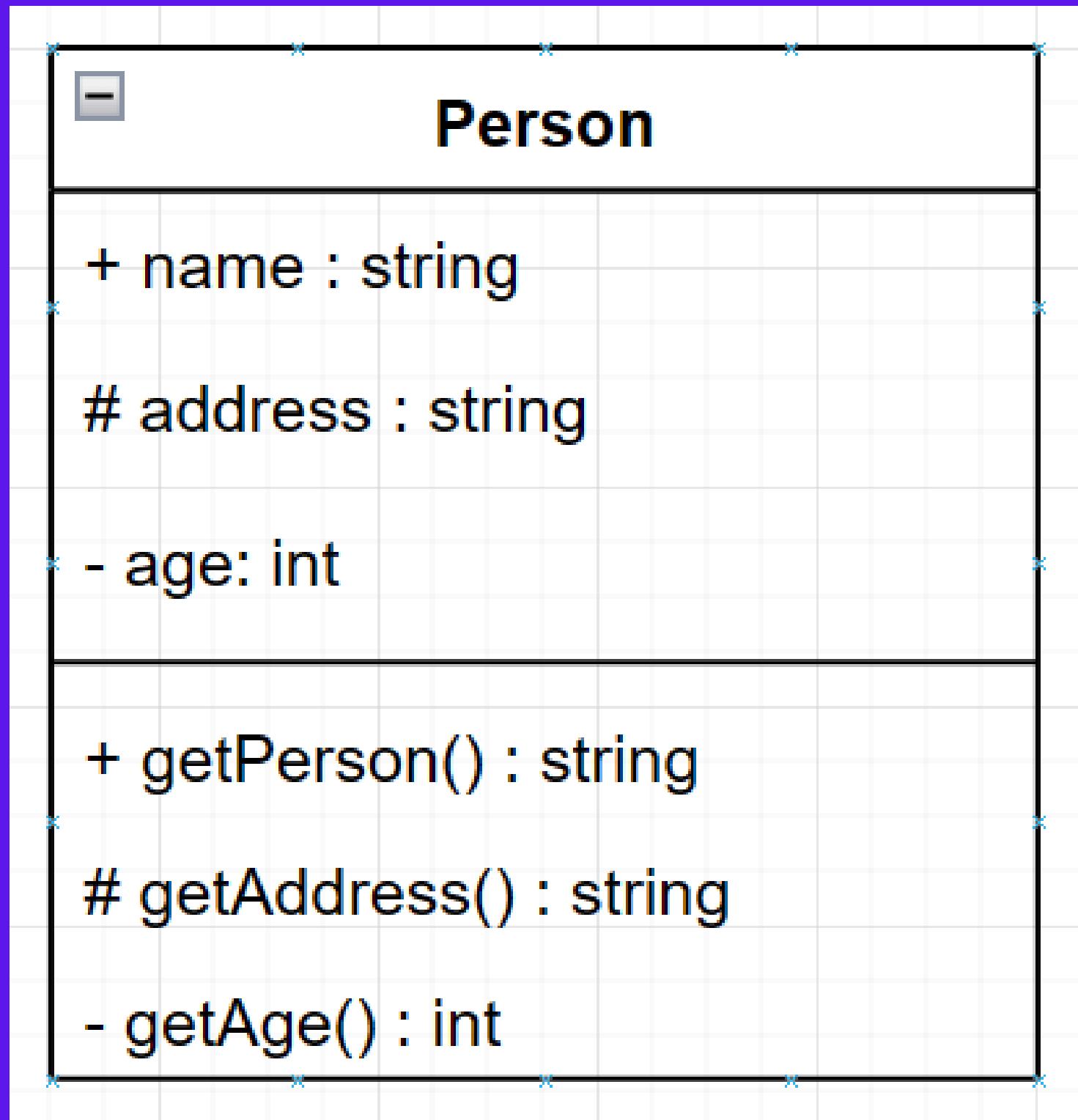
Kesalahan Private Access Modifier

```
1 <?php
2
3 class Person
4 {
5     private int $money = 0;
6
7     function call(): void
8     {
9         if ($this->personalIssue()) {
10             echo "Masalah pribadi";
11         } else {
12             echo "Tidak ada masalah pribadi";
13         }
14     }
15
16     private function personalIssue(): bool
17     {
18         if ($this->money < 100) {
19             return true;
20         } else {
21             return false;
22         }
23     }
24 }
25
26 class Employee extends Person
27 {
28     function call(): void
29     {
30         if ($this->personalIssue()) {
31             echo "Masalah pribadi";
32         } else {
33             echo "Tidak ada masalah pribadi";
34         }
35     }
36 }
37
38 $person = new Person();
39 $person->personalIssue();
```

Jika global scope ingin mengakses method yang sifatnya protected, maka akan menyebabkan error.

```
Uncaught Error: Call to private method Person::personalIssue() from context 'Employee' in /var/www/html/latihan/latihan.php:39
```

Access Modifier Pada Class Diagram



A screenshot of a code editor showing a PHP code snippet. The code defines a class "Person" with attributes \$name (public), \$address (protected), and \$age (private). It contains three methods: getPerson(), getAddress(), and getAge(). The code uses color-coded syntax highlighting for PHP keywords and variables.

```
<?php
class Person {
    public string $name;
    protected string $address;
    private int $age;

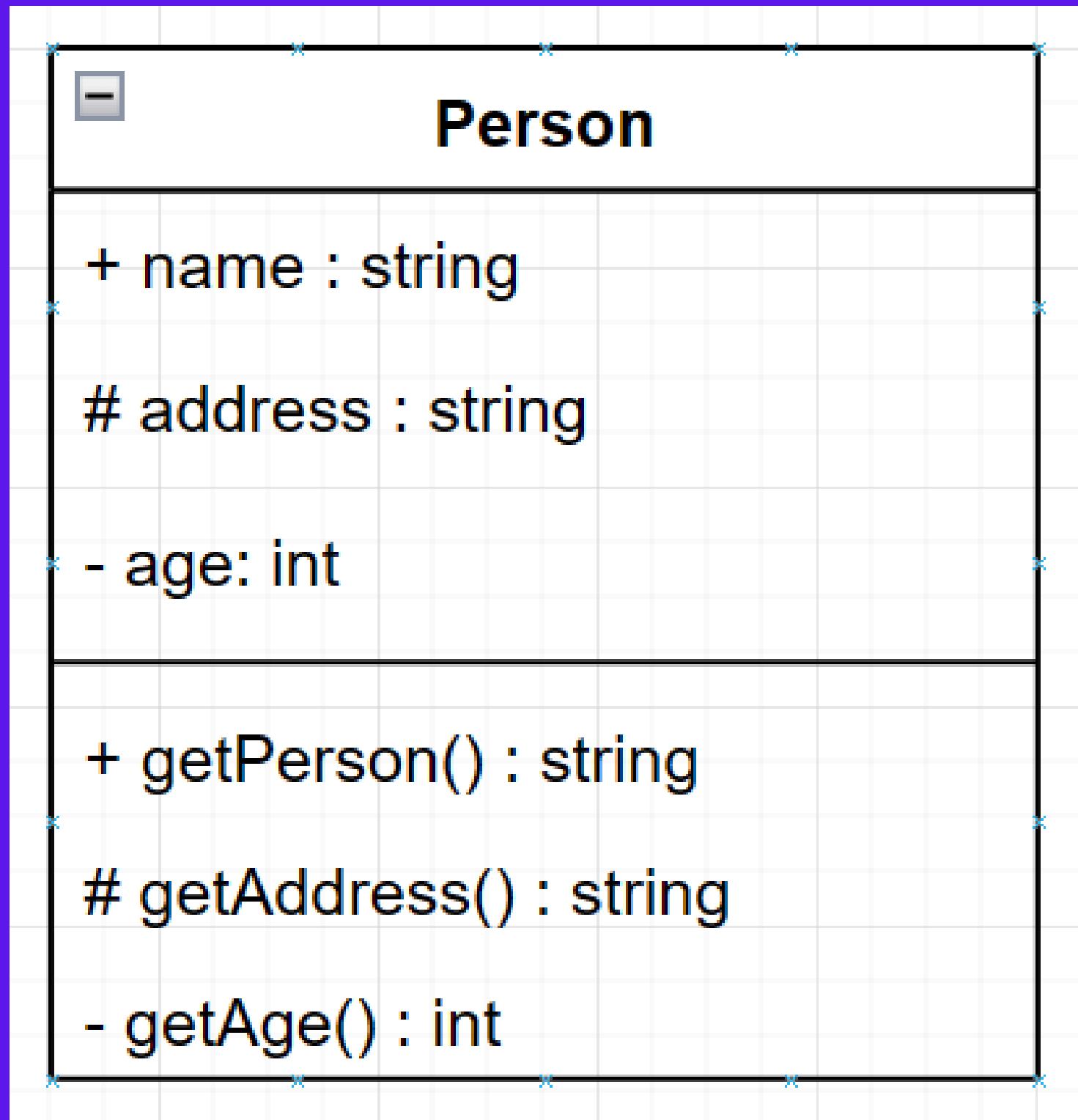
    public function getPerson(string $name): string
    {
        return $name;
    }

    protected function getAddress(string $address): string
    {
        return $address;
    }

    private function getAge(int $age): int
    {
        return $age;
    }
}
```

Untuk class diagram simbol + (public), # (protected), dan - (private).

Access Modifier Pada Class Diagram



```
1 <?php
2
3 class Person
4 {
5     public string $name;
6     protected string $address;
7     private int $age;
8
9     public function getPerson(string $name): string
10    {
11        return $name;
12    }
13
14    protected function getAddress(string $address): string
15    {
16        return $address;
17    }
18
19    private function getAge(int $age): int
20    {
21        return $age;
22    }
23 }
```

Untuk class diagram simbol + (public), # (protected), dan - (private).

Overriding dan Overloading



```
1 <?php  
2  
3 class Animal  
4 {  
5     public function eat(): void  
6     {  
7         echo "Hewan ini makan";  
8     }  
9 }  
10  
11 class Cat extends Animal  
12 {  
13     public function eat(): void  
14     {  
15         echo "Kucing ini makan";  
16     }  
17 }
```

Konsep Overriding dan Overloading sebenarnya adalah dua hal yang tidak bisa dipisahkan karena overriding adalah pendefinisian ulang sebuah method dan attribute pada child class, dalam pendefinisian ulang, atribut yang terdapat dalam method bisa tambah atau diubah itu yang disebut dengan overloading

Overriding dan Overloading

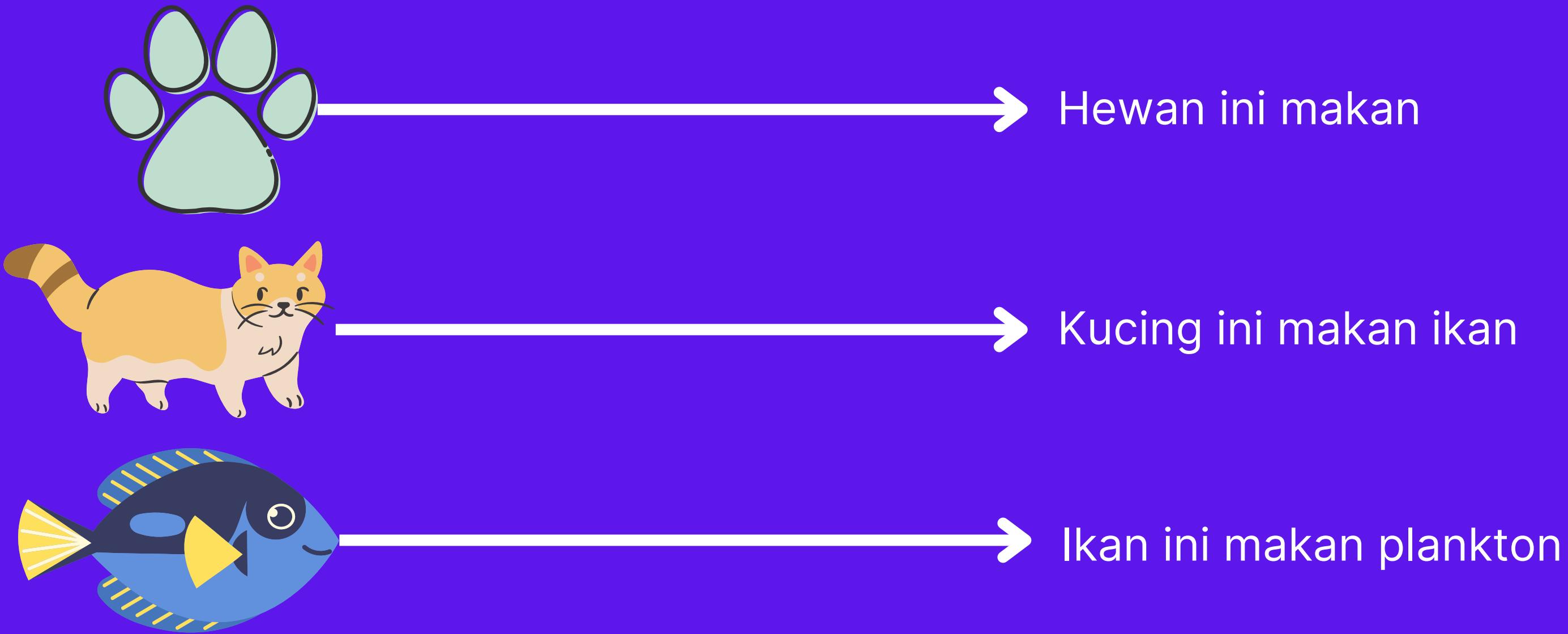


```
1 <?php  
2  
3 class Animal  
4 {  
5     public function eat(): void  
6     {  
7         echo "Hewan ini makan";  
8     }  
9 }  
10  
11 class Cat extends Animal  
12 {  
13     public function eat(): void  
14     {  
15         echo "Kucing ini makan ikan";  
16     }  
17 }
```

Memiliki method yang sama yaitu eat(), sebutannya adalah **overriding**

Karena outputnya berbeda dengan class parent maka itu disebut dengan **overloading**

Analogi Overriding dan Overloading



Setiap hewan jenis makannya sendiri maka konsep overriding dibutuhkan sebagai pembeda output

Implementasi Tanpa Overriding & Menggunakan Overriding

```
● ● ●  
1 <?php  
2  
3 class Animal  
4 {  
5     public function eat(): void  
6     {  
7         echo "Hewan ini makan <br>";  
8     }  
9 }  
10  
11 class Cat extends Animal  
12 {  
13 }  
14  
15 class Fish extends Animal  
16 {  
17 }  
18  
19 $animal = new Animal();  
20 $cat = new Cat();  
21 $fish = new Fish();  
22  
23 $animal->eat(); // Hewan ini makan  
24 $cat->eat(); // Hewan ini makan  
25 $fish->eat(); // Hewan ini makan
```

Hewan ini makan
Hewan ini makan
Hewan ini makan

```
● ● ●  
1 <?php  
2  
3 class Animal  
4 {  
5     public function eat(): void  
6     {  
7         echo "Hewan ini makan <br>";  
8     }  
9 }  
10  
11 class Cat extends Animal  
12 {  
13     public function eat(): void  
14     {  
15         echo "Kucing ini makan ikan<br>";  
16     }  
17 }  
18  
19 class Fish extends Animal  
20 {  
21     public function eat(): void  
22     {  
23         echo "Ikan ini makan plankton<br>";  
24     }  
25 }  
26  
27  
28 $animal = new Animal();  
29 $cat = new Cat();  
30 $fish = new Fish();  
31  
32 $animal->eat(); // Hewan ini makan  
33 $cat->eat(); // Kucing ini makan ikan  
34 $fish->eat(); // Ikan ini makan plankton
```

Hewan ini makan
Kucing ini makan ikan
Ikan ini makan plankton

Setiap hewan jenis makannya sendiri maka konsep overriding dibutuhkan sebagai pembeda output

Method Chaining

Method Chaining merupakan salah satu konsep OOP untuk mengikat / merantai sebuah method yang dapat memanggil lebih dari satu method dalam satu kali operasi dengan mengolah object yang sama. Manfaat menggunakan Chaining adalah membuat kode lebih mudah dibaca dan lebih pendek.



```
1  
2 $chain = new Chain();  
3 $chain->chainOne()->chainTwo()->chainThree();
```

Perbandingan Method Chaining & Tanpa Method Chaining



```
1 $chain = new Chain();
2 $chain->chainOne()->chainTwo()->chainThree();
```

Method Chaining

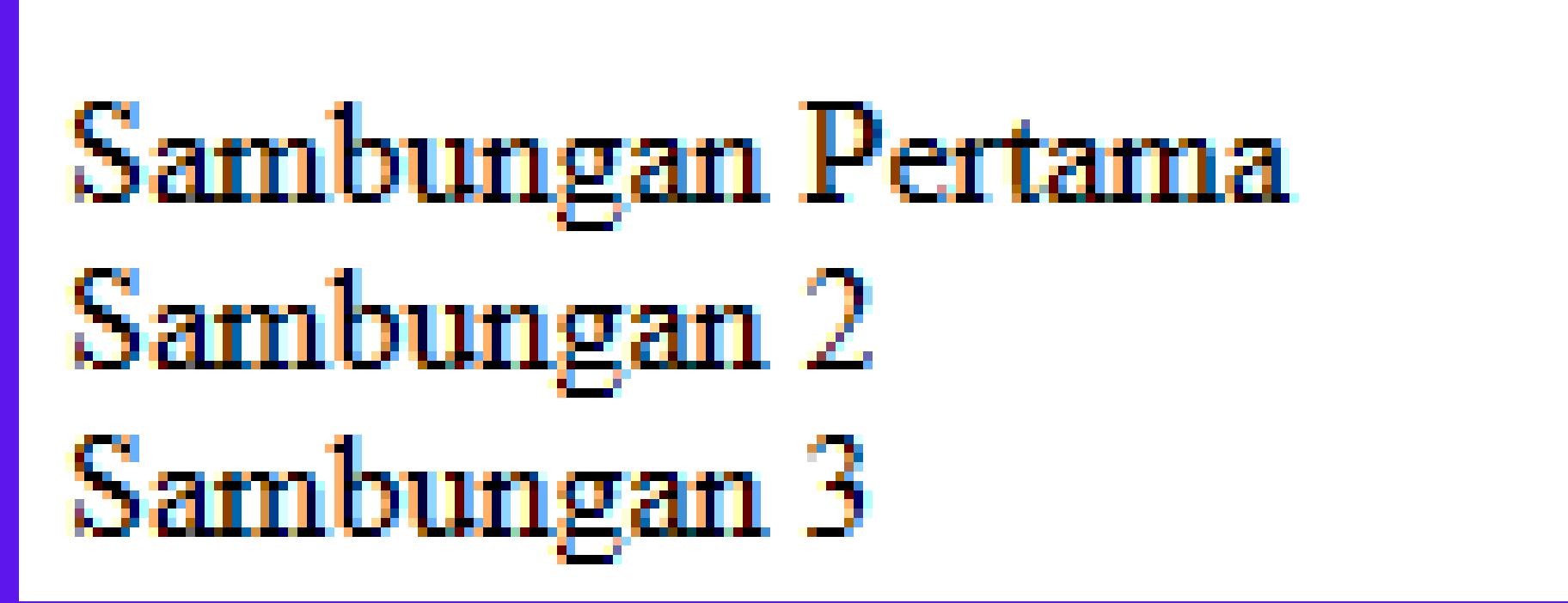
Tanpa Method Chaining



```
1 $chain = new Chain();
2 $chain->chainOne();
3 $chain->chainTwo();
4 $chain->chainThree();
```

Implementasi Method Chaining

```
● ● ●  
1 <?php  
2  
3 class Chain  
4 {  
5     public function chainOne(): Chain  
6     {  
7         echo "Sambungan Pertama <br>";  
8         return $this;  
9     }  
10  
11    public function chainTwo(): Chain  
12    {  
13        echo "Sambungan 2<br>";  
14        return $this;  
15    }  
16  
17    public function chainThree(): Chain  
18    {  
19        echo "Sambungan 3<br>";  
20        return $this;  
21    }  
22 }  
23  
24 $chain = new Chain();  
25 $chain->chainOne()->chainTwo()->chainThree();  
26
```



Sambungan Pertama
Sambungan 2
Sambungan 3

Agara Method bisa di panggil secara berantai (Chained), setiap method harus mengembalikan (return) sebuah output dengan variable \$this yang dimaksud disini adalah berfungsi untuk mengembalikan object untuk dioperasikan lagi pada method selanjutnya.

Jenis Penulisan Chaining Method



```
1  
2 $chain = new Chain();  
3 $chain->chainOne()->chainTwo()->chainThree();
```

Cara pertama

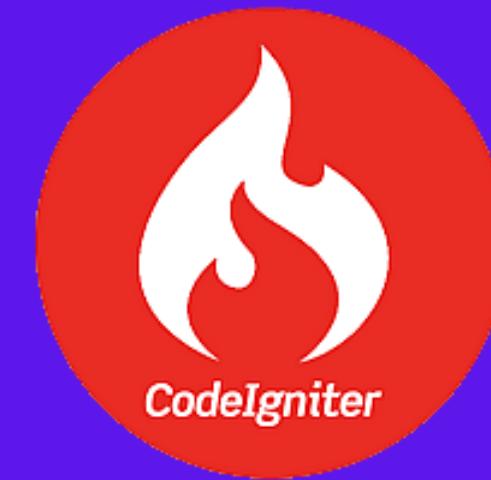


Cara kedua

```
1 $chain = new Chain();  
2 $chain->chainOne()  
3           ->chainTwo()  
4           ->chainThree();
```

Kapan Menemukan Baris Kode Chaining Method?

```
● ● ●  
1 <?php  
2  
3  
4 defined('BASEPATH') or exit('No direct script access allowed');  
5  
6 class Home extends CI_Controller  
7 {  
8  
9     public function __construct()  
10    {  
11        parent::__construct();  
12        $this->load->model('HomeModel');  
13    }  
14  
15    public function index()  
16    {  
17        $total = $this->HomeModel->countData('pendaftaran');  
18        $data = [  
19            'total' => $total,  
20            'title' => 'Bootcamp Mardira 2022'  
21        ];  
22        $this->template->load('template/default', 'Home/index', $data);  
23    }  
}
```



Pada framework CodeIgniter 3 , bisa menemukan baris kode tentang chaining method

Abstract Class



```
1 <?php  
2  
3 abstract class Shape  
4 {  
5  
6 }
```

Abstract Class adalah sebuah class yang tidak bisa diinstance (tidak bisa dibuat menjadi objek) dan berperan sebagai 'kerangka dasar' bagi class turunannya.

Aturan Membuat Abstract Class

- Menggunakan keyword "abstract" sebelum keyword "class"
- Abstract tidak bisa dibuat sebagai object secara langsung (instance), hanya bisa diturunkan ke class turunannya.
- Abstract class digunakan di dalam inheritance (pewarisan class) untuk ‘memaksakan’ implementasi method yang sama bagi seluruh class yang diturunkan dari abstract class

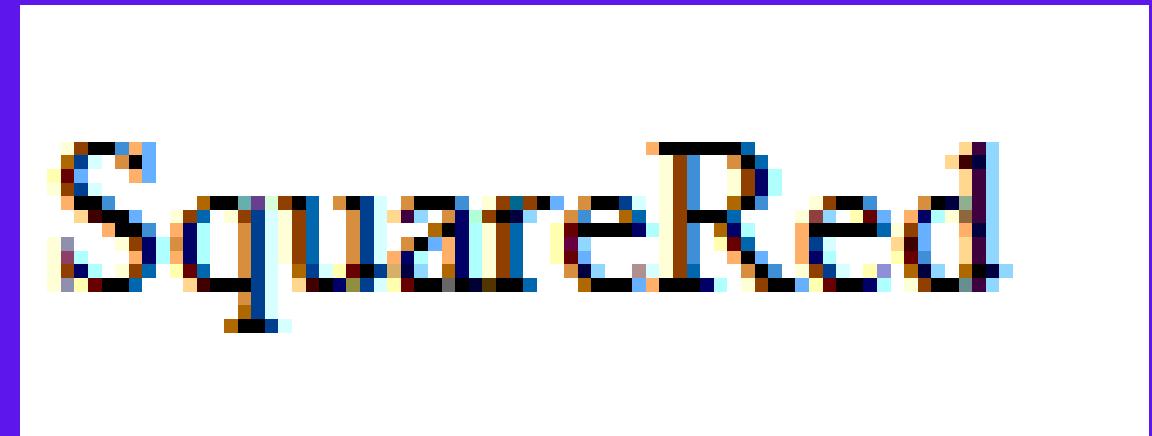


```
1 <?php  
2  
3 abstract class Shape  
4 {  
5  
6 }
```

Implementasi Abstract Class



```
1 <?php  
2  
3 abstract class Shape  
4 {  
5     public string $name;  
6     public string $color;  
7 }  
8  
9 class Square extends Shape  
10 {  
11     public int $length = 4;  
12 }  
13  
14 $square = new Square();  
15 echo $square->name = "Square";  
16 echo $square->color = "Red";
```



Abstract Method / Function



```
1 <?php
2
3 abstract class Shape
4 {
5     public string $name;
6     public string $color;
7
8     abstract public function area(): float;
9 }
```

Abstract Method adalah sebuah method dasar dari sebuah abstract class yang harus diimplementasikan ulang di dalam sub class (child class). Abstract method ditulis tanpa isi dari method tersebut.

Aturan Membuat Abstract Function

- Menggunakan keyword "abstract" sebelum keyword "function"
- Abstract function wajib di override oleh class turunannya
- Abstract function tidak boleh membuat block / compound function tersebut.
- Abstract function tidak boleh memiliki access modifier private

```
● ● ●  
1 <?php  
2  
3 abstract class Shape  
4 {  
5     public string $name;  
6     public string $color;  
7  
8     abstract public function area(): float;  
9 }
```

Implementasi Abstract Function

```
● ● ●  
1 <?php  
2  
3 abstract class Shape  
4 {  
5     public string $name;  
6     public string $color;  
7  
8     abstract public function area(): float;  
9 }  
10  
11 class Square extends Shape  
12 {  
13     public float $length = 0;  
14  
15     public function area(): float  
16     {  
17         return pow($this->length, 2);  
18     }  
19 }  
20  
21 $square = new Square();  
22 $square->length = 4;  
23 echo $square->area();
```



Interface



```
1 <?php  
2  
3 interface Vehicle  
4 {  
5     public function setTire($tire);  
6     public function setEngine($engine);  
7 }  
8 }
```

Interface adalah sebuah kontrak atau perjanjian implementasi method artinya setiap class yang terlibat kontrak dengan interface tersebut harus mengimplementasikan ulang seluruh method yang ada di dalam interface.

Interface



```
1 <?php
2
3 interface Vehicle
4 {
5     public function setTire($tire);
6
7     public function setEngine($engine);
8 }
```

Interface adalah sebuah kontrak atau perjanjian implementasi method artinya setiap class yang terlibat kontrak dengan interface tersebut harus mengimplementasikan ulang seluruh method yang ada di dalam interface.

Perbedaan Interface dan Abstract



```
1 <?php
2
3 interface Vehicle
4 {
5     public function setTire($tire);
6
7     public function setEngine($engine);
8 }
```



```
1 <?php
2
3 abstract class Shape
4 {
5     public string $name;
6     public string $color;
7
8     abstract public function area(): float;
9 }
```

Meskipun memiliki persamaan dalam kontrak untuk mewajibkan mengimplementasikan ulang setiap object kepada class turunannya, abstract class sama layaknya dengan class memiliki property dan method, sedangkan interface hanya memiliki method saja

Implementasi Interface

```
● ● ●  
1 <?php  
2  
3 interface Vehicle  
4 {  
5     public function setTire($tire): string;  
6  
7     public function setEngine($engine): string;  
8 }  
9  
10 class Car implements Vehicle  
11 {  
12     public function setTire($tire): string  
13     {  
14         return "Mobil mempunyai {$tire} ban <br>";  
15     }  
16  
17     public function setEngine($engine): string  
18     {  
19         return "Mobil mempunyai mesin {$engine} <br>";  
20     }  
21 }  
22  
23 $car = new Car();  
24 echo $car->setTire(4);  
25 echo $car->setEngine('1.5L');
```

Mobil mempunyai 4 ban
Mobil mempunyai mesin 1.5L

Untuk mewarisi semua method dari interface kepada class turunan menggunakan keyword **implements**

Implementasi Multi Interface

```
● ● ●  
1 <?php  
2  
3 interface HasEngine  
4 {  
5     public function setEngine($engine): string;  
6 }  
7  
8 interface HasTire  
9 {  
10    public function setTire($tire): string;  
11 }  
12  
13 class Car implements HasEngine, HasTire  
14 {  
15     public function setTire($tire): string  
16     {  
17         return "Mobil mempunyai {$tire} ban <br>";  
18     }  
19  
20     public function setEngine($engine): string  
21     {  
22         return "Mobil mempunyai mesin {$engine} <br>";  
23     }  
24 }  
25  
26 $car = new Car();  
27 echo $car->setTire(4);  
28 echo $car->setEngine('1.5L');
```

Mobil mempunyai 4 ban
Mobil mempunyai mesin 1.5L

Implementasi multi interface bisa menggunakan 2 interface sekaligus dengan menyebutkan interfacenya satu persatu

Inheritance Interface

```
1 <?php
2
3 interface HasEngine
4 {
5     public function setEngine($engine): string;
6 }
7
8 interface HasTire
9 {
10    public function setTire($tire): string;
11 }
12
13 interface Vehicle extends HasEngine, HasTire
14 {
15 }
16
17 class Car implements Vehicle
18 {
19     public function setTire($tire): string
20     {
21         return "Mobil mempunyai {$tire} ban <br>";
22     }
23
24     public function setEngine($engine): string
25     {
26         return "Mobil mempunyai mesin {$engine} <br>";
27     }
28 }
29
30 $car = new Car();
31 echo $car->setTire(4);
32 echo $car->setEngine('1.5L');
```

Mobil mempunyai 4 ban
Mobil mempunyai mesin 1.5L

Interface juga bisa diwariskan juga / extends kepada interface lainnya, kemudian class turunan atau class yang di kontrak sama interface akan menjalankan interface yang sudah di wariskan oleh interface sebelumnya. Secara tidak langsung class car menjalankan 3 interface

Inheritance Interface & Inheritance Class

```
1 <?php
2
3 interface HasEngine
4 {
5     public function setEngine($engine): string;
6 }
7
8 interface HasTire
9 {
10    public function setTire($tire): string;
11 }
12
13 interface HasMethod extends HasEngine, HasTire
14 {
15 }
16
17 abstract class Vehicle
18 {
19     abstract public function setType($type): string;
20 }
21
22 class Car extends Vehicle implements HasMethod
23 {
24     public function setTire($tire): string
25     {
26         return "Mobil mempunyai {$tire} ban <br>";
27     }
28
29     public function setEngine($engine): string
30     {
31         return "Mobil mempunyai mesin {$engine} <br>";
32     }
33
34     public function setType($type): string
35     {
36         return "Mobil ini adalah {$type} <br>";
37     }
38 }
39
40 $car = new Car();
41 echo $car->setTire(4);
42 echo $car->setEngine('1.5L');
43 echo $car->setType('Sedan');
```

Mobil mempunyai 4 ban
Mobil mempunyai mesin 1.5L
Mobil ini adalah Sedan

Inheritance interface bisa dengan inheritance class pada abstract class dengan menggunakan extend dan juga menggunakan implements

Terimakasih