

Pewarisan

Pewarisan adalah mekanisme pemberian sifat maupun ciri khusus dari induk (*parent*) kepada keturunannya (*child*). Bila kita hubungkan dengan konsep pada OOP maka pewarisan adalah mekanisme memberikan *property* maupun *method* dari *parent class* kepada *child class*.

Property dan *method* yang diwariskan adalah *property*, *method* dan konstanta yang mempunyai visibilitas *protected* dan *public* sebagaimana yang telah dibawah pada bab sebelumnya.

- **Penerapan Pewarisan**

Untuk membuat sebuah *class* menjadi turunan dari *class* lainnya, kita menggunakan *keyword* *extends* sebagai berikut:

```
?php
class Hewan
{
    private $jenis;

    public function setJenis($jenis)
    {
        $this->jenis = $jenis;
    }
    public function getJenis()
    {
        return $this->jenis;
    }
}
class Kambing extends Hewan
{
}
class Harimau extends Hewan
{
}
class Singa extends Hewan
{
}

$kambing = new Kambing();
$kambing->setJenis('Herbivora');
$harimau = new Harimau();
$harimau->setJenis('Karnivora');
$singa = new singa();
```

```

$singa->setJenis('Karnivora');
echo    $kambing->getJenis();
echo    PHP_EOL;
echo    $harimau->getJenis();
echo    PHP_EOL;
echo    $singa->getJenis();
echo    PHP_EOL;

```

Pada contoh diatas, secara otomatis *class* Kambing, Harimau dan Singa memiliki semua *method* yang ada pada *class* Hewan sehingga bila program diatas dijalankan maka *output*-nya adalah sebagai berikut:

```

php Hewan.php
Output:
Herbivora
Karnivora
Karnivora

```

Pada bahasa pemrograman PHP tidak dikenal *multiple inheritance* sehingga kita setiap *child class* hanya boleh memiliki 1 *parent class*. Perhatikan contoh berikut:

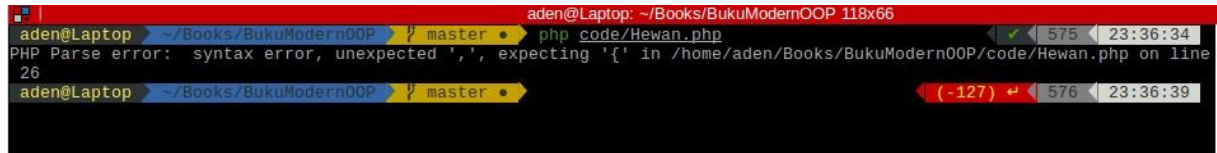
```

<?php
class    Hewan
{
    private $jenis;
    public function    setJenis($jenis)
    {
        $this->jenis    =    $jenis;
    }
    public function    getJenis()
    {
        return    $this->jenis;
    }
}
class    Mamalia
{
    public function    menyusui()
    {
        echo    'mik    susu    aaahhh';
    }
}
class    Kambing extends Hewan,    Mamalia
{
}

```

```
$kambing = new Kambing();
```

Bila program diatas dijalankan maka yang terjadi adalah *error syntax* karena setelah pada baris, Mamalia seharusnya adalah tanda kurung kurawan buka { seperti gambarberikut:



Namun kita dapat menggunakan *nested inheritance* untuk mengakali hal tersebut seperti tampak pada contoh dibawah ini:

```
<?php
class Hewan
{
    private $jenis;
    public function setJenis($jenis)
    {
        $this->jenis = $jenis;
    }
    public function getJenis()
    {
        return $this->jenis;
    }
}
class Mamalia extends Hewan
{
    public function menyusui()
    {
        echo 'mik susu aaahhh';
    }
}
class Kambing extends Mamalia
{
}

$kambing = new Kambing();
$kambing->menyusui();
echo PHP_EOL;
```

- ***Overloading dan Overriding***

- ***Keyword parent***

Pada pembahasan sebelumnya kita telah memahami tentang 2 *keyword* yang merujuk pada *object* yang diinstansiasi yaitu *\$this* dan *self*.

Pada pembahasan kali ini, saya akan menambahkan satu lagi *keyword* yang merujuk pada *object* yaitu *parent*. Sama seperti *\$this* dan *self* yang akan bersifat dinamis dan akan digantikan oleh *real object* ketika diinstansiasi, *keyword parent* pun demikian. Hanya saja bila *keyword \$this* dan *self* itu merujuk pada *class* dimana ia didefinisikan, maka *keyword parent* itu merujuk kepada *parent class*.

Keyword parent hanya dapat digunakan pada *child class* dan hanya digunakan untuk memanggil *method* yang ada pada *parent class*. *Keyword* ini erat kaitannya dengan pembahasan berikutnya Yaitu *overloading* dan *overriding*.

- ***Apa itu Overloading dan Overriding***

Berbeda dengan konsep OOP pada Java yang memungkinkan kita mendefinisikan ulang sebuah *method* dengan nama yang sama asalkan parameternya berbeda. Di PHP kita tidak diperbolehkan untuk melakukan hal tersebut sehingga konsep *overloading* dan *overriding* dalam OOP PHP tidak dapat dipisahkan.

Selain itu karena kita tidak dapat mendefinisikan ulang sebuah *method* dengan nama yang sama, maka konsep *overloading* dan *overriding* dalam OOP PHP menjadi bergantung dengan konsep Pewarisan yang telah kita bahas sebelumnya. *Overriding* secara mudah dipahami sebagai pendefinisian ulang sebuah *method* milik *parent class* oleh *child class*.

Biasanya pendefinisian ulang tersebut diperlukan jika kita ingin menambahkan *logic* atau memodifikasi *logic* yang ada pada *method* tersebut. Penambahan atau modifikasi *logic* pada *method* di *child class* tersebut itulah yang disebut *overloading*.

- ***Penerapan Overloading dan Overriding***

Untuk lebih memahami konsep *overloading* dan *overriding* pada PHP, perhatikan contoh berikut:

```

<?php
class PostRepository
{
    public function getLatestPost()
    {
        $posts = [
            [
                'id' => 1,
                'title' => 'Judul Pertama',
                'content' => 'Contoh Content Pertama',
            ],
            [
                'id' => 2,
                'title' => 'Judul Kedua',
                'content' => 'Contoh Content Kedua',
            ],
            [
                'id' => 3,
                'title' => 'Judul Ketiga',
                'content' => 'Contoh Content Ketiga',
            ],
        ];
        return $posts;
    }
}

class SuffledPostRepository extends PostRepository
{
    public function getLatestPost()
    {
        $posts = parent::getLatestPost();
        shuffle($posts);
        return $posts;
    }
}

$postRepository = new PostRepository();
print_r($postRepository->getLatestPost());
$suffledPostRepository = new SuffledPostRepository();
print_r($suffledPostRepository->getLatestPost());

```

Pada contoh diatas kita mempunyai dua *class* yaitu *PostRepository* dan *SuffledPostRepository*, dimana *class* *SuffledPostRepository* adalah turunan dari *class* *PostRepository*. Pada *class* *SuffledPostRepository* kita meng-*override method* *getLatestPost()* dan menambahkan *logic* yaitu mengacak *index* dari *\$posts*. Penambahan *logic* inilah yang disebut dengan *overloading*.

```
$posts = parent::getLatestPost();
```

Pada baris *code* diatas, kita memanggil *method* `getLatestPost()` milik *class* `PostRepository` sebagaimana telah saya jelaskan pada pembahasan penggunaan *keyword* `parent`. *Keyword* `parent` kita gunakan karena paham dan tahu dengan pasti bahwa *logic* pada *parent class* sudah sesuai dengan apa yang kita inginkan sehingga kita tidak perlu menulis ulang semua *code* tersebut.

Kita hanya perlu sedikit perubahan untuk membuatnya sesuai dengan kebutuhan.

Agar semakin memahami tentang *override* dan *overload*, perhatikan contoh dibawah ini:

```
<?php
class Connection
{
    public function connect($database, $username, $password, $host =
    'localhost')
    {
        throw new RuntimeException('Anda harus mengimplementasikan met
        hod connect() sesuai dengan database driver yang Anda gunakan.');
```

```
    }
}

class MySqlConnection extends Connection
{
    public function connect($database, $username, $password, $host
    = 'localhost')
    {
        /**
         * Ceritanya ini logic koneksi ke database MySQL
         *
         * Anda tidak bisa menggunakan _keyword_ `parent` untuk
         memanggil _method_ `connect()` milik _parent class_
         * karena akan mengakibatkan error.
         */
    }
}

class PostgreSQLConnection extends Connection
{
    public function connect($database, $username, $password,
    $host = 'localhost')
    {
        /**
         * Ceritanya ini logic koneksi ke database PostgreSQL
```

```

        *
        *  Anda tidak bisa menggunakan _keyword_ `parent` un
tuk memanggil _method_ `connect()` milik _parent class_
        *  karena akan mengakibatkan error.
        **/
    }
}

```

Berbeda dengan contoh sebelumnya dimana kita dapat menggunakan *keyword* *parent* untuk mengambil *logic* pada *parent class*, pada contoh diatas, *parent class* memaksa kita untuk membuat *logic* sendiri sesuai dengan *database driver* yang kita gunakan. Ini terjadi karena saya men-trigger *exception* menggunakan *keyword* *throw* (pembahasan tentang *exception* dan *keyword* *throw* akan dibahas pada bab terpisah) sehingga akan memicu *error* Ketika *method* *connect()* pada *class* *Connection* dipanggil.

Bagaimana sudah dapat dipahami bagaimana *override* dan *overload* bekerja? Untuk memaksa *child class* mengimplementasikan *logic*-nya sendiri seperti cara diatas sebenarnya tidak benar karena pada OOP ada namanya konsep *abstract* yang akan dibahas setelah pembahasan *override* dan *overload* ini.