Interface

1. Apa itu *Interface*

Apa yang Anda bayangkan jika mendengar kata *interface*? Mungkin yang terbayang adalah sebuah halaman web yang lengkap dengan biasa Anda sebut dengan istilah *interface* atau *user interface*. Tapi dalam pemrograman berbasis objek, konsep *interface* yang dimaksud sangat berbeda jauh dengan *interface* yang Anda bayangkan tersebut.

Dalam pemrograman berbasis objek, *interface* adalah sebuah *class* yang semua *method*-nya adalah *abstract method*. Karena semua *method*-nya adalah *abstract method* maka *interface* pun harus diimplementasikan oleh *child class* seperti halnya pada *abstract class*. Hanya saja bila kita sebelumnya menggunakan *keyword* extends untuk mengimplementasikan sebuah *abstract class*, maka pada *interface* kita menggunakan *keyword* implements untuk mengimplementasikan sebuah *interface*.

2. Contoh Penggunaan *Interface*

Untuk lebih memahami bagaimana sebuah *interface* bekerja, coba perhatikan contoh berikut:

```php
<?php
interface HewanInterface
{
    public function getJenis();
}
class Kambing implements HewanInterface
{
    public function getJenis()
    {
        return  'Herbivora';
    }
}
class Harimau implements HewanInterface
{
    public  function    getJenis()
    {
        return  'Karnivora';
    }
}
class Singa implements HewanInterface
{
    public  function    getJenis()
```

```
    {
        return  'Karnivora';
    }
}
```

Pada contoh diatas, kita memiliki *interface* HewanInterface dan mempunyai 1 *abstract method* yaitu getJenis() serta memiliki 3 implementasi yaitu Kambing, Harimau dan Singa. Setiap *class* yang mengimplementasikan *interface* HewanInterface harus membuat implementasi dari *method* getJenis() seperti tampak pada contoh diatas. Selain itu juga kita dapat menggabungkan antara *interface* dan *abstract class* sebagaimana tampak pada contoh dibawah ini:

```
<?php
interface HewanInterface
{
    public function getJenis();
}
abstract class Hewan
{
    private $jenis;
    public  function    setJenis($jenis)
    {
        $this->jenis    =   $jenis;
    }
    public  function    getJenis()
    {
        return  $this->jenis;
    }
}
class Kambing extends Hewan implements HewanInterface
{
}
class Harimau   extends Hewan implements HewanInterface
{
}
class   Singa   extends Hewan   implements  HewanInterface
{
}
```

```
$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;
```

Dengan membuat *code* seperti diatas, *child class* kita tidak perlu mengimplementasikan *abstract method* milik *interface* HewanInterface karena sudah diimplementasikan oleh *abstract class* Hewan Dengan cara tersebut, maka *code* kita semakin rapi, mudah dibaca, dan *reusable*. Bila *code* diatas dijalankan, maka *output*-nya adalah sebagai berikut:

```
php  Hewan.php
Output:
Herbivora

Karnivora
Karnivora
```

Mungkin sampai disini Anda belum terlalu paham kenapa kita harus membuat *interface* sementara kita bisa menggunakan *abstract class* karena keduanya terlihat sama saja. Bahkan disatu sisi, *abstract class* jauh lebih menjanjikan karena kita masih bisa membuat *logic*, sementara pada *interface* karena semuanya adalah *abstract method* secara otomatis kita tidak diperkenankan membuat *logic* pada *interface*.

Pada prateknya, *interface* dan *abstract class* memiliki fungsi yang berbeda. *Interface* berfungsi untuk memastikan *child class* memiliki fitur-fitur yang telah ditetapkan dalam *interface*, sementara *abstract class* berguna untuk memberikan fitur-fitur dasar pada *child class* dimana fitur-fitur tersebut dapat digunakan secara bersama-sama oleh *child class*.

Seperti contoh diatas, dimana *interface* HewanInterface memiliki fitur getJenis(), sementara *abstract class* Hewan memberikan fitur setJenis() dan getJenis() untuk dapat dipakai secara bersama-sama oleh *child class* Singa, Harimau, dan Kambing.

Tidak seperti pada pewarisan, pada *interface* kita dapat menggunakan *multiple interface* sekaligus seperti pada contoh berikut:

```
<?php
interface HewanInterface
{
   public   function    getJenis();
```

```
}
interface   MamaliaInterface
{
   public   function    menyusui();
}
abstract    class   Hewan
{
    private $jenis;
    public  function    setJenis($jenis)
    {
         $this->jenis   =   $jenis;
    }
    public  function    getJenis()
    {
        return  $this->jenis;
    }
}
class Kambing extends Hewan implements HewanInterface, MamaliaInterface
{
    public  function    menyusui()
    {
        echo    'nyusu  kambing';
    }
}
class   Harimau extends Hewan   implements  HewanInterface, MamaliaInterfa
ce
{
    public  function    menyusui()
    {
        echo    'nyusu  maung';
    }
}
class   Singa   extends Hewan   implements  HewanInterface, MamaliaInterfa
ce
{
    public  function    menyusui()
    {
        echo    'nyusu  singa';
    }
}
```

Di era milenial seperti sekarang ini penggunaan *interface* sangat masif. Banyak *framework* dan *library* yang kalau kita mau membaca *source code*-nya maka akan mudah sekali bagi kita untuk menemukan *interface*.

Penggunaan *interface* tidak lain karena fitur yang dimiliki *interface* itu sendiri yaitu sebagai hirarki tertinggi pada *parameter casting* (akan dibahas pada bab tersendiri) dimana setiap *object* yang mengimplementasikan sebuah *interface* akan valid jika dimasukkan kedalam *method* yang menggunakan *interface* tersebut sebagai *type hinting* atau *parameter casting*.

Pada paradigma pemrograman modern, ada istilah "*interface as contract*" yang maksudnya adalah *interface* digunakan pada *parameter casting* sebagai pengikat bahwa *object* yang akan dimasukkan kedalam *method* pasti memiliki fitur-fitur atau *method method* yang didefinisikan pada *interface* tersebut Sehingga dengan menggunakan *interface* tersebut sebagai *parameter casting* pada *method* maka didalam *method* tersebut kita bisa dengan percaya diri untuk menggunakan *method-method* yang ada pada *interface* tanpa takut terjadi *error* undefined method

.