Penulisan Abstract Class

Cara penulisan *abstract class* sama seperti class biasa, yakni dengan keyword **class** kemudian diikuti dengan nama class. Di dalam abstract class juga bisa diisi property dan method sebagaimana biasanya. Yang membedakan adalah tambahan keyword **abstract** sebelum nama class:

```
1   abstract class Produk {
2     //... isi class di sini
3   }
```

Dalam contoh ini saya membuat class Produk sebagai **abstract class**. Salah satu fitur dari abstract class adalah tidak bisa di instansiasi:

01.abstract_class_instansiasi.php

```
1   <?php
2   abstract class Produk {
3   }
4
5   $produk01 = new Produk();
```

Hasil kode program:

// Fatal error: Uncaught Error: Cannot instantiate abstract class Produk

Saya mencoba melakukan instansiasi class Produk ke dalam variabel $produk01. Hasilnya tampil pesan error "*Cannot instantiate abstract class Produk*". Kembali ke pengertiannya, abstract class memang dirancang sebagai "template" untuk class turunan. Nantinya yang akan kita pakai adalah class turunan, bukan langsung dari abstract class:

02.abstract_class_extends.php

```
1   <?php
2   abstract class Produk {
3   }
4
5   class Televisi extends Produk{
6   }
7
8   $produk01 = new Televisi();
```

Di sini saya menurunkan class abstract Produk ke dalam class Televisi, kemudian membuat object dari class Televisi.

Abstract Method

Abstract method adalah sebuah method dengan tambahan keyword **abstract**. *Abstract method* hanya bisa ditulis di dalam *abstract class:*

```
1   <?php
2   abstract class Produk {
3     abstract public function cekHarga();
4   }
```

Dalam kode ini saya membuat sebuah abstract method cekHarga() di dalam abstract class Produk. Abstract method cukup ditulis *signature*-nya saja, yakni nama method serta parameter (jika ada). Kita tidak bisa menulis implementasi dari abstract method:

03.abstract_method_implement.php

```php
<?php
abstract class Produk {
  abstract public function cekHarga(){
    return 3000000;
  }
}
```

Hasil kode program:

Fatal error: Abstract function Produk::cekHarga() cannot contain body

Error di atas tampil karena dalam abstract method cekHarga() saya membuat implementasi berupa { return 3000000; }. Kenapa? Karena di dalam konsep abstract class, yang harus membuat implementasi adalah class turunan, bukan di dalam abstract class itu sendiri. Malah setiap class turunan **wajib** untuk membuat implementasi dari abstract method:

04.abstract_method_extends.php

```php
<?php
abstract class Produk {
  abstract public function cekHarga();
}

class Televisi extends Produk{
}

$produk01 = new Televisi();
```

Di sini saya membuat class Televisi sebagai turunan dari abstract class Produk. Perhatikan bahwa di dalam abstract class Produk terdapat abstract method cekHarga(). Bagaimana hasilnya?

Fatal error: Class Televisi contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (Produk::cekHarga)

Error ini terjadi karena class Televisi **diharuskan** untuk membuat implementasi dari method cekHarga(). Agar tidak error, saya harus membuat ulang method cekHarga() di dalam class Televisi:

```php
<?php
abstract class Produk {
  abstract public function cekHarga();
}

class Televisi extends Produk{
  public function cekHarga(){
    return 3000000;
  }
}

$produk01 = new Televisi();
echo $produk01->cekHarga();  // 3000000
```

Hasil kode program:

```
3000000
```

Inilah inti dari konsep *abstract class*, yakni memaksa class turunan untuk membuat implementasi dari abstract method.
Jika saya membuat class MesinCuci sebagai turunan dari class Produk, maka di dalam class MesinCuci juga harus ada implementasi dari method cekHarga():

06.abstract_method_implement_ok_2.php

```php
<?php
abstract class Produk {
  abstract public function cekHarga();
}

class Televisi extends Produk{
  public function cekHarga(){
    return 3000000;
  }
}

class MesinCuci extends Produk{
  public function cekHarga(){
    return 1500000;
  }
}

$produk01 = new Televisi();
echo $produk01->cekHarga();

echo "<br>";

$produk01 = new MesinCuci();
echo $produk01->cekHarga();
```

Hasil kode program:

```
3000000
1500000
```

Yang terpenting dari abstract class adalah, setiap class turunan harus membuat implementasi dari seluruh abstract method. Mengenai isi dari method tersebut bisa saja berbeda-beda dan diserahkan kepada class turunan. Sebuah abstract class juga bisa berisi property dan method biasa:

07.abstract_class_property_method.php

```php
<?php
abstract class Produk {
  private $stok = 200;
  abstract public function cekHarga();

  public function cekStok(){
    return $this->stok;
  }
}

class Televisi extends Produk{
  public function cekHarga(){
    return 3000000;
  }
}


$produk01 = new Televisi();
echo $produk01->cekHarga();  // 3000000

echo "<br>";

echo $produk01->cekStok();  // 15000
```

Hasil kode program:

```
3000000
200
```

Kali ini saya mengisi class Produk dengan property $stok serta method cekStok(). Keduanya adalah property dan method "biasa" yang tetap bisa diakses dari class turunan. Method cekStok() sendiri hanya berisi kode untuk menampilkan isi dari property $stok.

Implemetasi abstract method di class turunan juga harus sesuai dengan *signature method* tersebut, yakni nama method serta parameter. Jika dalam abstract method terdapat parameter, maka implementasi di class turunan juga harus menyertakan parameter:

08.abstract_method_signature.php

```php
<?php
abstract class Produk {
  abstract public function cekHarga($jumlah);
}

class Televisi extends Produk{
  public function cekHarga($jumlah){
    return 3000000 * $jumlah;
  }
}

$produk01 = new Televisi();
echo $produk01->cekHarga(2);    // 6000000
```

Kali ini abstract method cekHarga() memiliki parameter $jumlah (baris 3). Oleh karena itu di dalam class Televisi saya juga harus membuat implementasi method cekHarga() dengan parameter $jumlah. Jika *signature*-nya tidak sesuai, akan tampil pesan error:

09.abstract_method_signature_error.php

```php
<?php
abstract class Produk {
  abstract public function cekHarga($jumlah);
}

class Televisi extends Produk{
  public function cekHarga(){
    return 3000000;
  }
}

$produk01 = new Televisi();
```

Hasil kode program:

Fatal error: Declaration of Televisi::cekHarga() must be compatible with Produk::cekHarga($jumlah)

Error ini terjadi karena dalam implementasi method cekHarga() di class Televisi saya tidak menyertakan parameter $jumlah. Dengan kata lain, PHP menganggap method cekHarga($jumlah) berbeda dengan method cekHarga(). Untuk visibility, method turunan harus memiliki visibility yang sama atau lebih luas. Misalnya di *abstract method* menggunakan public, maka di method turunan juga harus di set sebagai public, tidak bisa protected atau private:

```php
<?php
abstract class Produk {
  abstract public function cekHarga();
}

class Televisi extends Produk{
  protected function cekHarga(){
    return 3000000;
  }
}

$produk01 = new Televisi();
// Fatal error: Access level to Televisi::cekHarga() must be public (as in class Produk)
```

Hasil kode program:

Fatal error: Access level to Televisi::cekHarga() must be public (as in class Produk)

Error di atas terjadi karena saya membuat visibility method cekHarga() di class Televisi sebagai protected, padahal di class Produk sudah di set sebagai public. Namun jika visibility ini "diperluas" di class turunan, itu tidak masalah:

11.abstract_method_visibility_ok.php

```php
<?php
abstract class Produk {
  abstract protected function cekHarga();
}

class Televisi extends Produk{
  public function cekHarga(){
    return 3000000;
  }
}

$produk01 = new Televisi();
echo $produk01->cekHarga();     // 3000000
```

Dalam class Produk, abstract method cekHarga() saya set sebagai protected. Pada class Televisi, method cekHarga() ini kemudian di implementasikan ulang sebagai public. Ini diperbolehkan karena visibility-nya diperluas dari protected ke public. Namun abstract method tidak bisa di set sebagai private, karena itu akan menyalahi tujuan dari abstract class yang harus ditimpa di dalam class turunan:

12.abstract_method_private_error.php

```php
<?php
abstract class Produk {
  abstract private function cekHarga();
}
```

Hasil kode program:
Fatal error: Abstract function Produk::cekHarga() cannot be declared private

Untuk design program yang kompleks, kita juga bisa menurunkan abstract class ke abstract class lain, seperti contoh berikut:

13.abstract_class_turunan.php

```php
<?php
abstract class Produk {
  abstract public function cekHarga();
}

abstract class Televisi extends Produk{
  abstract public function cekTipe();
}

class TelevisiLED extends Televisi{
  public function cekHarga(){
    return 3000000;
  }
  public function cekTipe(){
    return "TV LED";
  }
}

$produk01 = new TelevisiLED();
echo $produk01->cekHarga();   // 3000000
echo "<br>";
echo $produk01->cekTipe();    // TV LED
```

Di sini saya membuat 2 buah abstract class, yakni class Produk dan class Televisi. Di dalam class Produk terdapat abstract method cekHarga(), serta di dalam class Televisi terdapat abstract method cekTipe(). Class Televisi sendiri merupakan turunan dari class Produk. Di baris 10 saya membuat class TelevisiLED yang diturunkan dari class Televisi. Akibatnya, class TelevisiLED harus meng-implementasikan method cekHarga() dan cekTipe() karena kedua method ini ada di parent class dan grand parent class yang merupakan abstract class. Yang cukup unik, kita juga bisa mengisi abstract class dengan static property dan static method, kemudian memanggilnya:

```php
<?php
abstract class Produk {
  public static $totalProduk = 100;

  public static function cekProduk(){
    return "Total Produk ada ".self::$totalProduk;
  }
}

echo Produk::$totalProduk;    // 100
echo Produk::cekProduk();     // Total Produk ada 100
```

Cara ini tidak banyak dipakai namun memperlihatkan bahwa ini bisa dilakukan ke dalam abstract class.

Polymorfism

Jika anda pernah mempelajari teori pemrograman berorientasi objek, besar kemungkinan sudah paham atau setidaknya pernah mendengar 3 istilah yang menjadi konsep dasar OOP, yakni **encapsulation**, **inheritance** dan **polymorphism**. Secara tidak langsung sebenarnya kita telah

mempraktekkan *encapsulation* dan *inheritance*. **Encapsulation** adalah mekanisme untuk "menyatukan" kode program menjadi satu kesatuan yang utuh, serta menyembunyikan kode program internal agar tidak bisa diakses dari luar (karena memang tidak perlu). Proses penyatuan kode program ini diperoleh dengan penerapan **class** dan **object**. Sebuah class atau object berisi kode program yang saling berhubungan. Sedangkan proses menyembunyikan kode program diperoleh dari penerapan *visibility* atau *access modifier*, yakni: *private*, *public* dan *protected*.

Inheritance adalah penurunan class yang bertujuan agar kode program bisa dipakai ulang tanpa harus menulisnya kembali (*code reuse*). Jika sebuah property atau method sudah didefinisikan di parent class, secara otomatis semua class turunan akan memiliki property dan method tersebut. Di dalam PHP kita membuat turunan dengan keyword extends. Dibandingkan 2 konsep di atas, **polymorphism** mungkin sedikit rumit. Secara bahasa, *polymorphism* berasal dari dua kata latin yakni *poly* dan *morph*. **Poly** berarti banyak dan **morph** berarti bentuk. Polimorfisme berarti banyak bentuk.

Polymorphism adalah konsep pemrograman object dimana sebuah method bisa memiliki nama yang sama, tapi penerapannya berbeda-beda tergantung object dari method tersebut. Konsep ini sebenarnya bisa dibuat menggunakan method *overriding*, yakni menimpa method parent class dengan membuat nama method yang sama, namun abstract class yang baru saja kita pelajari "memaksa" terjadinya *polymorphism*.

Sebagai contoh, perhatikan kode program berikut:

```
1 <?php
2 abstract class Produk {
3   abstract public function cekHarga();
4   abstract public function cekMerek();
5   abstract public function cekStok();
6   abstract public function beli();
7 }
8
9 class Televisi extends Produk{
10   // isi class Televisi
11 }
12
13 class MesinCuci extends Produk{
14   // isi class MesinCuci
15 }
16
17 class LemariEs extends Produk{
18   // isi class LemariEs
19 }
```

Dengan pendefinisian seperti ini, class Televisi, MesinCuci dan LemariEs bisa dipastikan akan memiliki method cekHarga(), cekMerek(), cekStok() dan beli(). Namun seperti apa hasil dari setiap method, itu bisa berbeda-beda tergantung implementasi dari setiap class. Sebagai contoh, saya akan buat implementasi dari method cekMerek():

```php
<?php
abstract class Produk {
  abstract public function cekMerek();
}

class Televisi extends Produk{
  public function cekMerek(){
    return "Polytron";
  }
}

class MesinCuci extends Produk{
  public function cekMerek(){
    return "Electrolux";
  }
}

class LemariEs extends Produk{
  public function cekMerek(){
    return "Sharp";
  }
}

$produk01 = new Televisi();
$produk02 = new MesinCuci();
$produk03 = new LemariEs();

echo $produk01->cekMerek(). "<br>";    // Polytron
echo $produk02->cekMerek(). "<br>";    // Electrolux
echo $produk03->cekMerek(). "<br>";    // Sharp
```

Inilah implementasi dari *polymorphism*, dimana setiap class memiliki method yang seragam, yakni cekMerek(). Dengan membuat class Produk sebagai abstract class, semua class turunan dari class Produk akan dipaksa memiliki method cekMerek(). Lebih jauh lagi, nantinya kita bisa membuat sebuah function atau method "generik" yang bisa memproses semua class turunan dari class Produk. Berikut contohnya:

16.polymorfism_function.php

```php
<?php
abstract class Produk {
  abstract public function cekMerek();
}

class Televisi extends Produk{
  public function cekMerek(){
    return "Polytron";
  }
}

class MesinCuci extends Produk{
  public function cekMerek(){
    return "Electrolux";
  }
}

class LemariEs extends Produk{
  public function cekMerek(){
    return "Sharp";
  }
}

$produk01 = new Televisi();
$produk02 = new MesinCuci();
$produk03 = new LemariEs();

function tampilkanMerek($objectProduk){
   return $objectProduk->cekMerek(). "<br>";
}

echo tampilkanMerek($produk01);    // Polytron
echo tampilkanMerek($produk02);    // Electrolux
echo tampilkanMerek($produk03);    // Sharp
```

Kode program di baris 1 – 26 sama seperti contoh sebelumnya, dimana saya mendefinisikan class Produk sebagai abstract class lalu membuat class turunan berupa class Televisi, MesinCuci dan LemariEs. Di baris 28 – 30, saya membuat fungsi tampilkanMerek() dengan 1 parameter bernama $objectProduk. Sesuai namanya, parameter ini nantinya akan berisi object turunan class Produk. Fungsi tampilkanMerek() ini mengembalikan nilai hasil $objectProduk->cekMerek(). Mari kita bahas apa yang terjadi ketika perintah echo tampilkanMerek($produk01) dijalankan. Di baris 24 saya men-inisialisasi variabel $produk01 sebagai object dari class Televisi. Variabel $produk01 ini kemudian menjadi inputan argument untuk fungsi tampilkanMerek() di baris 32. Dalam fungsi tampilkanMerek(), variabel $produk01 akan dikirim ke dalam parameter $objectProduk. Lalu ditampilkanlah hasil dari $objectProduk->cekMerek(). Hal yang saya juga dilakukan untuk object $produk02 dan $produk03 di baris 33 dan 34. Di sini kita membuat fungsi tampilkanMerek() sebagai sebuah class "generik" yang bisa memproses semua object turunan class Produk. Perintah $objectProduk->cekMerek() akan selalu bisa berjalan selama argument yang diisi adalah object dari turunan class Produk. Kenapa? Karena di dalam class Produk terdapat abstract method cekMerek(), sehingga semua class turunannya harus memiliki method cekMerek(). Inilah hubungan dari abstract class dengan konsep *polymorphism* di dalam pemrograman berorientasi objek.