BAB 8

MODEL OBJEK PHP

Bagaimana sesungguhnya model objek pada PHP? Fitur model objek baru dari PHP 5 merupakan jawaban tepat untuk menjelaskannya. Sebenarnya kemampuan sebagai pemrograman yang mendukung orientasi objek bukanlah satu-satunya fitur baru dari PHP 5. Akan tetapi, dukungan orientasi objek ini merupakan salah satu fitur yang cukup menonjol sekali. Terbukti dengan adanya perbaikan fitur berbasis objek dalam versi ini dan ditambahkan pula model objek baru yang tidak ditemukan di versi sebelumnya.

Tidak dapat dipungkiri memang, pada kenyataannya untuk saat ini fitur canggih tersebut belum digunakan secara meluas. Hal ini bisa jadi karena belum banyak web server yang menggunakannya dan masih setia dengan PHP 4, selain itu juga karena performansinya masih memberikan sedikit keraguan bagi beberapa kalangan.

Terlepas dari adanya kekurangan pada pemrograman berorientasi objek dalam PHP 5, pembahasan di sini dimaksudkan untuk menjelaskan bagaimana membuat aplikasi web berorientasi objek. Bagaimanapun juga, pendekatan pemrograman dengan berorientasi objek mampu menyajikan kode program lebih sederhana dan *reusability*, sehingga memudahkan Anda mengembangkan aplikasi web yang kompleks.

8.1 Prosedural dan Orientasi Objek

Kebanyakan pemrogram lebih familiar dengan pemrograman terstruktur atau prosedural. Salah satu faktor penyebabnya adalah usianya yang sudah cukup matang, sehingga juga sudah meluas digunakan oleh berbagai kalangan. Di mana sekitar tahun 1950 diperkenalkan bahasa pemrograman Fortran dengan tipe prosedural.

Ada pun pemrograman berorientasi objek meskipun sebenarnya juga sudah cukup lama dikenal namun penggunaannya tidak seluas pemrograman prosedural. Kurang lebih sekitar tahun 1970 muncul *SmallTalk* yang memperkenalkan orientasi objek. Seiring dengan semakin meningkatnya kebutuhan, orientasi objek dipandang sebagai solusi yang cukup menarik dalam mengembangkan aplikasi.

Berkaitan dengan pendekatan yang digunakan dalam pemrograman, PHP memungkinkan Anda untuk membuat program menggunakan pendekatan secara prosedural atau berorientasi objek. Secara normal, kebanyakan pemrogram lebih sering menggunakan pendekatan prosedural, apalagi sebelumnya dukungan objek pada PHP masih belum selengkap saat ini. Oleh karena itu, apabila Anda baru mengenal PHP, maka pemrograman prosedural mungkin akan lebih akrab bagi Anda. Akan tetapi, apabila Anda ingin mengembangkan aplikasi menggunakan PHP untuk implementasi lebih baik, disarankan mengenal orientasi objek dari PHP.

Untuk mengetahui lebih jelas mengenai keduanya, kita akan membahas satu per satu, karena bagaimana pun juga baik prosedural maupun orientasi objek sangat diperlukan dalam mengembangkan aplikasi.

8.1.1 Pemrograman Prosedural

Kalangan yang fanatik terhadap prosedural umumnya tidak menyarankan penggunaan pendekatan abstrak. Contoh ekstrem dari kalangan ini adalah melakukan langkah penolakan objek, dan tidak menerima abstraksi tentunya. Mereka cenderung melihat bagaimana menghasilkan sesuatu yang cepat dan tidak memperhatikan jika orang lain dapat membaca kode programnya.

Bahkan tidak jarang yang menganggap bahwa pemrograman adalah kompetisi kecepatan pada aktivitas tim. Dalam pengembangan PHP, kalangan seperti inilah yang memungkinkan pembuatan modul PECL serta memiliki kontribusi terhadap efisiensi kode program.

Listing program berikut menunjukkan contoh penulisan program oleh kalangan prosedural.

```
<?php
/* procedural.php */

print "Hello, world.";

?>
```

8.1.2 Pemrograman Berorientasi Objek

Lain halnya dengan kalangan yang fanatik terhadap objek, umumnya mereka tidak begitu memperhatikan faktor performansi pada pendekatan yang dilakukan. Bahkan terlihat kalangan ini sangat menikmati konsep desain abstrak, karena orang-orang seperti ini biasanya berkarir di bidang manajemen proyek atau dokumentasi.

Contoh penulisan listing program di kalangan orientasi objek dapat digambarkan seperti berikut.

```
<?php
/* objek.php */

class HelloWorld {
  function myPrint() {
    print "Hello, world.";
  }
}
$myHelloWorld = new HelloWorld();
$myHelloWorld->myPrint();

?>
```

Dalam lingkungan PHP, kalangan orientasi objek lebih cenderung membuat modul PEAR dan memiliki kontribusi besar terhadap kemudahan pengelolaan kode program.

8.2 Konsep Dasar OOP

Orientasi objek telah terbukti kelayakannya selama bertahun-tahun dan terbukti pula sebagai pemrograman yang cukup tangguh. OOP merupakan paradigma pemrograman yang cukup dominan saat ini, karena mampu memberikan solusi kaidah pemrograman modern. Meskipun demikian, bukan berarti bahwa pemrograman prosedural sudah tidak layak lagi.

OOP diciptakan karena dirasakan masih adanya keterbatasan pada bahasa pemrograman tradisional. Konsep dari OOP sendiri adalah, semua pemecahan masalah dibagi ke dalam objek. Dalam OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek. Proses perancangan atau desain dalam suatu pemrograman merupakan proses yang tidak terpisah dari proses yang mendahului, yaitu analisis dan proses yang mengikutinya.

Pembahasan mengenai orientasi objek tidak akan terlepas dari konsep objek seperti *inheritance* atau penurunan, *encapsulation* atau pembungkusan, dan *polymorphism* atau kebanyakrupaan. Konsep-konsep ini merupakan fundamental dalam orientasi objek yang perlu sekali dipahami serta digunakan dengan baik, dan menghindari penggunaannya yang tidak tepat.

8.2.1 Class dan Objek

Dalam lingkungan program berorientasi objek, pemrogram mendefinisikan class secara statik. Pada saat *run-time*, class akan diinstantiasi menjadi objek. Ada pun objek yang merupakan instantiasi dari suatu class selalu dapat diacu melalui *current* objek, apa pun nama instant-nya.

Dapat didefinisikan bahwa class merupakan struktur data dari suatu objek, lebih jelasnya adalah sebuah bentuk dasar atau *blueprint* yang mendefinisikan variabel *method* umum pada semua objek dari beberapa macam. Objek sendiri adalah kumpulan variabel dan fungsi yang dihasilkan dari template khusus atau disebut *class*.

Kiranya cukup penting untuk membedakan antara class dengan objek. Di mana objek adalah elemen pada saat run-time yang akan diciptakan, dimanipulasi, dan dihancurkan ketika eksekusi. Ada pun class merupakan definisi statik dari himpunan objek yang mungkin diciptakan sebagai instantiasi dari class. Sederhananya adalah kumpulan objek yang mempunyai atribut sama. Dengan demikian, pada saat run-time maka yang kita miliki adalah objek.

Paling tidak suatu class memiliki struktur sebagai berikut:

```
class NamaClass {

}
```

Agar dapat digunakan, maka class memerlukan atribut dan operasi, di mana dibuat dengan cara mendeklarasikan variabel di dalam class menggunakan keyword var.

```
class NamaClass {
    var $atribut1;
    var $atribut2;

    function operasi() {

    }
}
```

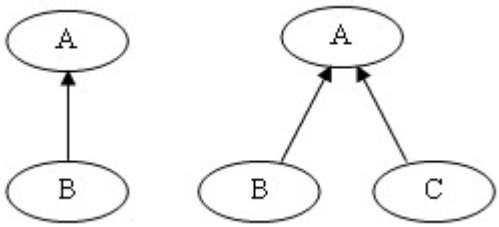
Dalam bahasa pemrograman lain seperti Java, file program harus disimpan sama dengan nama class. Lain halnya dengan PHP, Anda dapat memberikan nama yang tidak harus sama.

8.2.2 Inheritance

Untuk menggambarkan *inheritance* atau pewarisan di dalam pemrograman, pada saat Anda menggunakan kembali atau mengganti method dari class yang sudah ada, serta ketika menambahkan *field instant* dan method baru, maka pada saat itulah Anda bekerja dengan *inheritance*. Konsep ini merupakan konsep yang fundamental dalam orientasi objek dan harus digunakan dengan baik.

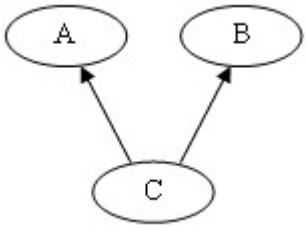
Ada beberapa macam jenis *inheritance* yang dikenal dalam pemrograman berorientasi objek, di antaranya adalah *single inheritance* dan *multiple inheritance*.

Dalam *single inheritance*, sebuah class turunan merupakan turunan dari sebuah class induk, perhatikan ilustrasi pada Gambar 8.1. Terlihat bahwa class B mewarisi class A, bentuk lain menjelaskan bahwa class B serta class C adalah turunan dari class A.



Gambar 8.1 Single inheritance

Ada pun pada *multiple inheritance*, sebuah class turunan mewarisi lebih dari satu class induk (*join*). Hal ini dapat menimbulkan beberapa persoalan jika ternyata ada fitur di class-class induk yang ternyata konflik, misalnya konflik nama atau body.



Gambar 8.2 Multiple inheritance

Pada hubungan *inheritance*, sebuah class turunan mewarisi kelas leluhur. Oleh karena mewarisi, maka semua atribut dan method class dari induk

akan dibawa, secara intrinsik menjadi bagian dari class anak. Dalam beberapa keadaan, membawa secara intrinsik semua atribut dan method tidak dikehendaki, sehingga pemroses bahasa menyediakan sarana untuk:

- Menambah fitur baru
- Mengubah atau mengganti fitur yang diwarisi
- Menghapus fitur yang diwarisi dan
- Menentukan fitur yang masih belum terdefinisi

Pada kenyataannya, hal ini menimbulkan persoalan yang tidak sederhana, karena penghapusan fitur dapat menimbulkan beberapa konsekuensi berbahaya sehingga sedikit sekali yang menyarankan penghapusan fitur. Listing program berikut akan menunjukkan bagaimana implementasi pewarisan.

```
<?php
/* inheritance.php */

class Bapak {
  var $nama ="Bapak";

  function Bapak($n) {
      $this->nama = $n;
  }

  function Hallo() {
      echo "Halo, saya $this->nama <br>";
  }
}

class Anak extends Bapak {

}

$test = new Anak("Anak dari Bapak");
$test->Hallo();

?>
```

Hasil tampilan dari listing program di atas adalah “Halo, saya Anak dari Bapak” dan bukannya “Halo, saya Bapak”. Mengapa demikian? Memang di dalam class Bapak didefinisikan variabel nama dengan nilai Bapak, selanjutnya kita membuat objek dari class Anak yang merupakan

turunan dari class Bapak. Lihat bahwa instantiasi sekaligus mengisikan parameter baru "Anak dari Bapak", sehingga ketika dipanggil maka mengisikan `$this->nama` dengan parameter tersebut.

Ada pun di dalam implementasi pemrograman, kebanyakan pemrogram merasakan beberapa manfaat dari *inheritance* atau pewarisan, di antaranya:

- Subclass mampu menyediakan perilaku khusus dari elemen dasar yang disediakan oleh superclass.
- Pemrogram dapat mengimplementasikan superclass untuk memanggil class abstrak yang menyatakan perilaku umum.

Catatan:

Class turunan selalu bergantung pada base class tunggal, dan sampai saat ini PHP tidak mendukung adanya multiple inheritance.

8.2.3 Encapsulation

Konsep fundamental berikutnya di dalam pemrograman berorientasi objek adalah encpsulation atau pembungkusan. Orientasi objek mendukung karakteristik enkapsulasi menggunakan konsep class, dan setelah terbentuk maka class akan bertindak sebagai entitas yang terenkapsulasi (terbungkus). Tentu saja enkapsulasi objek ini memiliki maksud tersendiri, terutama dalam impelementisnya ketika mengembangkan perangkat lunak berbasis objek.

Sering pula dikatakan bahwa enkapsulasi merupakan teknik penyembunyian informasi ke dalam struktur. Tujuan enkapsulasi adalah untuk menyembunyikan properti dan method dari suatu objek, dan hanya menampilkan properti serta method yang dibutuhkan. Ada pun properti atau method yang ditampilkan dari suatu objek dikenal dengan istilah interface. Pembahasan mengenai interface akan dijelaskan dalam subbab tersendiri.

Beberapa keuntungan yang didapatkan pemrogram melalui penggunaan enkapsulasi adalah:

- Modularitas

  Kode program untuk objek dapat ditulis serta dikelola secara independen untuk kode program pada objek lainnya.

- Informasi yang tersimpan dan tersembunyi

  Objek memiliki *interface* public yang dapat digunakan oleh objek lain untuk berkomunikasi. Selain itu, objek juga memiliki informasi private yang dapat diubah setiap saat dengan tanpa mempengaruhi objek lain yang bergantung padanya.

8.2.4 Polymorphism

Istilah *polymorphism* berasal dari kata poly yang berarti banyak (*many*) dan morphos yaitu bentuk (*form*). Istilah ini ternyata tidak hanya digunakan dalam bahasa pemrograman, akan tetapi bidang-bidang seperti biologi dan kimia juga sering memakainya. Terlepas dari adanya bidang lain yang menggunakan, tentu saja di sini kita akan membahas dari sisi pemrograman.

Dalam pemrograman berorientasi objek, *polymorphism* merupakan hasil alamiah dari hubungan *is-a* dan suatu mekanisme dari *message passing*, *inheritance*, serta konsep *substitutability*. Sebagaimana diketahui, salah satu kemampuan menonjol dari pendekatan OOP bahwa peralatan dapat dikombinasikan dalam bermacam cara, dan menghasilkan teknik berbagi kode dan penggunaan ulang. Oleh karena itu, di dalam pemrograman berorientasi objek, polymorphism diartikan merupakan suatu konsep yang menyatakan bahwa sesuatu yang sama dapat memiliki berbagai bentuk serta perilaku berbeda.

Berkaitan dengan eksepsi *overloading*, polymorphism membuat kemungkinan melalui adanya variabel yang disebut variabel *polymorphism*. Begitu pula di dalam *overloading*, suatu method dikatakan *polymorphism* apabila memiliki banyak bentuk. Perlu diperhatikan bahwa di dalam PHP, hanya fungsi yang merupakan member dari class yang dapat menggunakan konsep *polymorphism* ini.

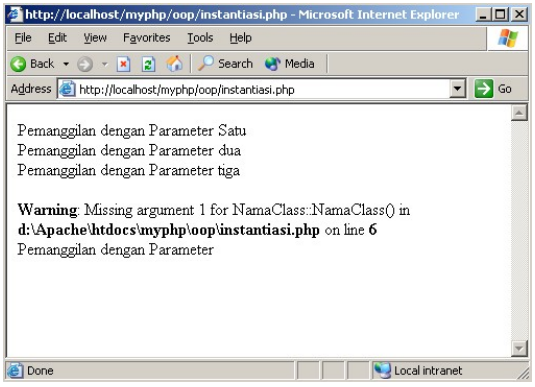
8.2.5 Instantiation

Begitu Anda selesai mendeklarasikan class, maka Anda akan melangkah ke pembuatan objek. Hal ini juga Anda alami ketika membuat *instance* atau *instantiating* suatu class.

Dalam pembuatan objek dengan kata kunci (*keyword*) *new*, Anda perlu menetapkan apa class objek yang akan di-*instance* dan menyediakan parameter pendukung.

```
<?php
/* instantiasi.php */

class NamaClass {

    function NamaClass ($param) {
        echo "Pemanggilan dengan Parameter $param <br>";
    }
}

$a = new NamaClass ("Satu");
$b = new NamaClass ("dua");
$b = new NamaClass ("tiga");
$d = new NamaClass();

?>
```

Pada listing program di atas, oleh karena konstruktor dipanggil terpisah maka dibuat objek tersendiri. Perhatikan tampilan hasil eksekusi program, meskipun objek `$d` berhasil dijalankan, namun terdapat pesan yang menyatakan bahwa diperlukan argumen pada pemanggilan parameter.



Gambar 8.3 Instantiasi objek

8.2.6 Overriding

Sebagaimana kita ketahui, *subclass* mendeklarasikan atribut dan operasi baru. Terkadang juga diperlukan untuk mendeklarasikan ulang atribut dan operasi yang sama. Dalam melakukan hal ini, kita memberikan nilai atribut yang berbeda pada *subclass* untuk objek yang sama pada *superclass*.

Alternatif lain adalah memberikan operasi pada *subclass* yang fungsinya berbeda dengan operasi yang sama pada *superclass*. Pada saat Anda melakukan hal seperti inilah maka dinamakan dengan *overriding*. Perhatikan contoh berikut:

```
<?php
/* overriding.php */

class A {
    var $attr = "Nilai A";

    function operasi () {
        echo "Something <br>";
        echo "Nilai = $this->attr <br>";
    }
}

class B extends A {
    var $attr = "Nilai B";

    function operasi () {
        echo "<p> Something else<br>";
        echo "Nilai = $this->attr <br>";
    }
}

$a = new A();
$a->operasi();

$b = new B();
$b->operasi();

?>
```

8.3 PHP dan OOP

Sejauh mana orientasi objek diterapkan dalam PHP? Dan mengapa perlu menggunakan OOP pada aplikasi web? Jawaban pertanyaan pertama mungkin belum memusakan jika Anda tidak mempraktekkan langsung pada PHP 5 dan membandingkan dengan versi sebelumnya. Jawaban

pertanyaan kedua bisa kondisional, artinya menyesuaikan dengan kebutuhan yang diperlukan. Akan tetapi, kita mencoba menatap ke depan, di mana aplikasi web semakin berkembang dan cukup kompleks. Oleh karena itu, orientasi objek sangat membantu ketika kita ingin mengembangkan aplikasi yang kompleks dan dengan kode program sederhana.

Sebelum melangkah lebih jauh dengan OOP, akan lebih baik jika kita mengetahui terlebih dahulu bagaimana orientasi objek pada PHP. Perlu diingat bahwa dukungan objek ini sudah dimasukkan sejak lama, artinya sebelum versi PHP 5 juga sudah mendukung orientasi objek. Meskipun demikian, ada beberapa kelemahan dan kekurangan yang cukup berarti.

Berbicara masalah kelemahan orientasi objek pada PHP versi lama, akan lebih mudah kalau kita ilustrasikan langsung pada program seperti berikut ini.

```
<?php
/* objek_baru.php */

class Pemrogram {
    var $nama;

    function getName() {
        return $this->nama;
    }
    function setName($nama) {
        $this->nama = $nama;
    }
    function Pemrogram($nama) {
        $this->setName($nama);
    }
}

function changeName($pemrogram, $nama) {
    $pemrogram->setName($nama);
}

$prog = new Pemrogram("Zend");
print $prog->getname();

print "<p> Objek setelah diganti <br>";
changeName($prog, "Andi");
print $prog->getName();

?>
```

Sebelumnya perhatikan maksud dari listing program di atas, di mana instantiasi objek sekaligus mengisikan parameter *Zend*. Ingat bahwa

objek dapat diakses dan diubah, sehingga kita melakukan pengubahan nilai parameter Andi. Ada pun hasil akhir yang ditampilkan setelah pengubahan adalah Andi, karena pengiriman objek ke fungsi `changeName() by-value`. Hal ini disebabkan isi dari `$prog` di-copy ke fungsi `changeName()` dan mengubah nilainya.

Pada kenyataannya tidak demikian di dalam PHP 4, meskipun objek sudah diubah, akan tetapi yang ditampilkan tetap saja *Zend*. Hal ini disebabkan pengiriman nilai parameter yang digunakan adalah pengiriman pertama, sehingga pengiriman melalui pengubahan akan diabaikan.

Kasus di atas merupakan salah satu fitur perbaikan dari PHP 5, di mana dikembangkan untuk memiliki perilaku seperti layaknya pemrograman berorientasi objek. Dalam orientasi objek, variabel sesungguhnya memiliki *handle* atau pointer ke objek, oleh sebab itu ketika terjadi pengkopian maka tidak akan membuat duplikat objek. Ada pun model objek dari PHP sebelumnya, tidak hanya bermasalah seperti contoh kasus di atas, namun juga menimbulkan masalah dasar dalam mengimplementasikan beberapa fasilitas. Anda akan melihat lebih jauh mengenai model objek baru ini pada pembahasan selanjutnya.

8.4 Model Objek Baru pada PHP 5

Model objek baru pada PHP 5 adalah salah satu fitur yang merupakan langkah untuk menyempurnakan dukungan orientasi objek pada PHP. Oleh karena itu, model-model dari objek lama yang sedikit bermasalah kini diperbaiki untuk menyesuaikan dengan kaidah bahasa pemrograman berorientasi objek.

Mengingat pembahasan kali ini banyak mengimplementasikan fitur baru pada PHP 5 maka disarankan Anda sudah menggunakannya. Akan lebih baik lagi jika Anda menggunakan versi terbaru, yang jelas sudah menutup kekurangan-kekurangan yang mungkin ada.

8.4.1 Objek

Sebelumnya telah dijelaskan mengenai definisi dari objek, selanjutnya di sini akan dibahas bagaimana membuat objek pada pemrograman PHP. Lebih penting lagi adalah cara mengakses serta mengubah objek, jika memang dimungkinkan. Untuk membuat objek, Anda pertama kali perlu mendesain template yang dapat diinstantiasi. Template tersebut kita kenal dengan nama class, perhatikan contoh class sederhana berikut:

```
class MyObjek {

}
```

Selanjutnya langkah pembuatan objek dilakukan menggunakan statemen new seperti contoh berikut.

```
$objek1 = new MyObjek();
$objek2 = new MyObjek();
```

Pemeriksaan terhadap objek dapat Anda lakukan menggunakan fungsi gettype() dengan parameter nama objek yang akan diperiksa. Fungsi ini bertujuan untuk mengambil tipe dari suatu variabel.

```
<?php
/* myobjek.php */

class MyObjek {

}

$objek1 = new MyObjek();
$objek2 = new MyObjek();

print "\$objek1 adalah ".gettype($objek1)." <br>";
print "\$objek2 adalah ".gettype($objek2)." <br>";

?>
```

Objek memiliki akses ke variabel khusus yang disebut properti dan dapat dideklarasikan di mana saja asalkan di dalam *body class*. Ada pun properti ini dapat berupa nilai, *array*, atau objek lainnya.

```
<?php
/* myobjek2.php */
```

```
class MyObjek {
    var $nama = "A";

}

$objek1 = new MyObjek();
$objek2 = new MyObjek();

$objek1->nama = "B";

print "$objek1->nama <br>";
print "$objek2->nama <br>";

?>
```

Perhatikan bahwa operator selektor -> memungkinkan kita untuk mengakses atau mengubah objek. Terlihat setelah pembuatan objek nilai variabel diubah menjadi B, sehingga objek1 bernilai B.

8.4.2 Access Modifier

Dalam model objek baru ini, Anda dapat menggunakan access modifier umum untuk mengontrol akses ke method maupun property. Access modifier ini digunakan untuk menentukan tipe akses dari suatu objek.

Pada pengaksesan anggota class, hak akses dunia luar terhadap anggota (data dan fungsi) diatur melalui tiga kunci wilayah, yaitu private, public, dan protected.

- Public

  Access modifier public dapat diakses oleh fungsi di luar class atau fungsi bukan anggota class tersebut. Cara pengaksesan dilakukan dengan menggunakan selektor (. atau ->).

- Private

  Jika wilayah member dideklarasikan sebagai private, maka ia hanya dapat diakses oleh fungsi anggota class tersebut.

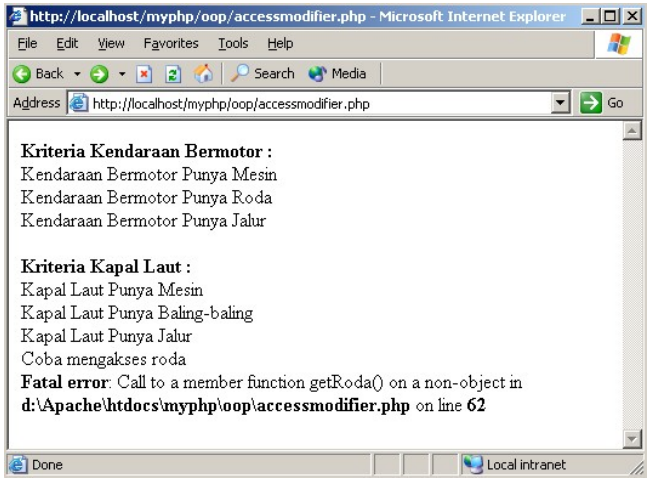
- Protected

  Protected hanya dapat diakses oleh fungsi anggota class dan fungsi-fungsi class turunan.

Setiap fungsi anggota class selalu dapat mengakses data dan fungsi anggota class tersebut di manapun data tersebut dideklarasikan, baik *private*, *public*, atau *protected*. Lain halnya dengan fungsi bukan anggota class, di mana ia hanya diperbolehkan untuk mengakses anggota yang berada di bagian public saja.

```
<?php
/* accessmodifier.php */

class KendBermotor {
  public $mesin;
  private $roda;
  protected $jalur;

    function __construct() {
    $this->mesin = "Kendaraan Bermotor Punya Mesin <br>";
    $this->roda = "Kendaraan Bermotor Punya Roda <br>";
    $this->jalur = "Kendaraan Bermotor Punya Jalur <br>";
    }

    function getMesin() {
    return $this->mesin;
    }

    function getJalur() {
    return $this->jalur;
    }

    function getRoda() {
    return $this->roda;
    }

}

// Class turunan dari KendBermotor
class KapalLaut extends KendBermotor {
  private $baling2;

    function __construct() {
    // Dapat dijalankan
    $this->mesin = "Kapal Laut Punya Mesin <br>";
    $this->jalur = "Kapal Laut Punya Jalur <br>";
    $this->baling2 = "Kapal Laut Punya Baling-baling <br>";

    // Tidak dapat diakses
    $this->roda = "Kapal Laut Punya Roda";
    }

    function getBaling2() {
    return $this->baling2;
    }
}

$KB = new KendBermotor();
echo "<p><b> Kriteria Kendaraan Bermotor : </b><br>";
echo $KB->getMesin();
echo $KB->getRoda();
```

```
echo $KB->getJalur();

$KL = new KapalLaut();
echo "<p><b> Kriteria Kapal Laut : </b><br>";
// Dapat diakses
echo $KL->getMesin();
echo $KL->getBaling2();
echo $KL->getJalur();

// Tidak dapat diakses
echo "Coba mengakses roda";
echo $PB->getRoda();

?>
```



Gambar 8.4 Mengontrol akses dengan access modifier

Seperti kita ketahui, variabel roda memiliki akses private sehingga ketika kita mencoba mengaksesnya dari class lain, yaitu KapalLaut maka tidak diperkenankan. Lain halnya dengan variabel mesin dan jalur yang bersifat public dan protected sehingga memungkinkan untuk dapat diakses oleh class turunan.

8.4.3 Interface

Interface merupakan kumpulan dari definisi method, selain itu interface juga dapat berisi deklarasi konstanta. Mengacu pada konsep penurunan,

suatu *interface* dapat diturunkan seperti halnya sebuah class. Ada pun class sebagaimana diketahui dapat diturunkan dari sebuah class saja, namun demikian bisa juga diimplementasikan sebagai interface jika memang memungkinkan.

```
interface Bangun {
    function display1();
}

interface Bangun2 extends Bangun {
    function display2();
}

/* implementasi class sebagai interface */
class SegiTiga implements Bangun2 {
    function display1() {
        print "Segi Tiga Siku-Siku <br>";
    }

    function display2() {
        print "Segi Tiga Sama Kaki";
    }

}
```

Hal penting yang perlu diperhatikan pada interface adalah bahwa Anda tidak dapat melakukan instantiasi *interface* Bangun sebagaimana dilakukan pada class. Oleh karena itu, jika Anda melakukan instantiasi maka akan menimbulkan kesalahan, misalnya seperti berikut:

```
/* Interface tidak dapat diinstantiasi */
$intrfc1 = new Bangun();
print $intrfc1->display1();
print $intrfc1->display2();
```

Alternatif lain untuk melakukan instantiasi adalah menggunakan class SegiTiga yang sebelumnya telah diimplementasikan sebagai interface.

```
<?php
/* interface.php */

interface Bangun {
    function display1();
}

interface Bangun2 extends Bangun {
    function display2();
}
```

```
/* implementasi class sebagai interface */
class SegiTiga implements Bangun2 {
    function display1() {
        print "Segi Tiga Siku-Siku <br>";
    }

    function display2() {
        print "Segi Tiga Sama Kaki";
    }

}

/* Interface tidak dapat diinstantiasi
$intrfc1 = new Bangun();
print $intrfc1->display1();
print $intrfc1->display2();
*/

$intrfc2 = new SegiTiga();
print $intrfc2->display1();
print $intrfc2->display2();

?>
```

Catatan:

Ketika class mengimplementasikan interface, maka semua method yang ada harus direalisasikan.

8.4.4 Constructor dan Destructor

Konstruktor (*constructor*) merupakan fungsi anggota yang mempunyai nama sama sesuai dengan nama class. Kegunaannya untuk meng-alokasikan ruang bagi sebuah objek, memberikan nilai awal, serta membentuk tugas-tugas umum lainnya. Mungkin sekali suatu class tidak memiliki konstruktor atau bahkan memiliki lebih dari satu konstruktor di dalamnya.

Pada umumnya konstruktor dapat dibedakan menjadi dua jenis, yaitu *default constructor* dan *user-defined constructor*.

- Default constructor

  Merupakan konstruktor yang menginisialisasi objek dengan nilai-nilai default yang ditentukan oleh perancang class. Di dalam deklarasi class, konstruktor ini tidak memiliki parameter formal.

- User-defined constructor

  Merupakan konstruktor yang menginisialisasi objek dengan nilai yang diberikan oleh pemakai class pada saat objek diciptakan. Dalam deklarasi class, konstruktor ini memiliki satu atau lebih parameter formal.

Untuk menjelaskan bagaimana penggunaan suatu konstruktor di dalam class, perhatikan listing program berikut.

```
<?php
/* konstruktor.php */

class Utama {

  function Utama() {
      echo "Konstruktor Utama <br>";
  }

  function Utama2() {
      echo "Bukan Konstruktor Utama <br>";
  }
}

$utm1 = new Utama();
$utm1->Utama2();
?>
```

Model objek baru dari PHP memungkinkan Anda untuk membuat konstruktor dari dalam, kemudian memanggilnya dari luar.

```
class NamaClass {

    // konstruktor dari dalam
    function __construct() {

        print "Konstruktor di dalam<br>";
    }
}

$test = new NamaClass();
```

Kembali pada listing konstruktor.php, konstruktor dari dalam ini sama halnya dengan konstruktor Utama.

```
  function __construct() {
      echo "Konstruktor Utama <br>";
  }
```

Destruktor (*destructor*) secara khusus memiliki fungsi untuk memanfaatkan kembali memori yang disimpan untuk objek dan sudah tidak digunakan lagi. Beberapa bahasa pemrograman secara eksplisit memiliki method destruktor untuk membersihkan kode yang mungkin diperlukan. Namun demikian, ada juga yang secara otomatis akan menghapus kode, sehingga tidak diperlukan lagi adanya destruktor, misalnya bahasa Java.

Secara umum destruktor diletakkan pada bagian public, dan dengan sendirinya akan dijalankan ketika objek akan berakhir (hilang). Hal ini pun dapat Anda lakukan dalam model objek baru pada kode program Anda.

```
<?php

class NamaClass {

    // destruktor
    function __destruct() {

        print "Menghapus objek";
    }
}

?>
```

8.4.5 Instance

Setiap Anda membuat instance pada suatu class, maka sistem akan membuat sebuah copy-an dari tiap-tiap variabel instance untuk instance tersebut. Pada PHP 4, pemeriksaan instance ini dilakukan menggunakan fungsi is_a().

```
<?php
/* is_a.php */

class Kendaraan {
  var $roda = 'Roda Kendaraan';
}

// Membuat objek baru
$K = new Kendaraan();

// Memeriksa instance
if (is_a($K, 'Kendaraan')) {
  echo " \$K adalah Kendaraan\n";
} else {
  echo " \$K Bukan Kendaraan\n";
}

?>
```

Untuk maksud yang sama, pemeriksaan *instance* ini pada versi PHP 5 dapat dilakukan menggunakan operator `instanceof`.

```
<?php
/* instance.php */

class Kendaraan {
  var $roda = 'Roda Kendaraan';
}

// Membuat objek baru
$K = new Kendaraan();

// Memeriksa instance
if ($K instanceof Kendaraan) {
  echo "\$K adalah Kendaraan\n";
} else {
  echo "\$K Bukan Kendaraan\n";
}

?>
```

8.4.6 Reference dan Objek Cloning

Reference ke suatu variabel adalah nama alias terhadap variabel tersebut, hal ini berbeda sekali dengan *pointer*. Dengan demikian, jika sudah digunakan untuk mengacu pada suatu objek atau variabel, maka *reference* tidak dapat di-*reset* untuk mengacu ke objek atau variabel lainnya. Fasilitas ini dapat dimanfaatkan untuk memberikan alias terhadap suatu variabel yang mempunyai nama panjang, misalnya karena berada dalam struktur yang berlapis.

Contoh berikut akan menunjukkan bagaimana cara sederhana me-*refer* variabel sekaligus memeriksa hasil *reference*.

```
<?php
/* reference.php */

class MyReference {

}

$a = new MyReference;
$b = $a;

if ($a === $b) {
    echo "$a dan $b me-refer pada objek yang sama";
} else {
    echo "$a dan $b TIDAK me-refer pada objek yang sama";
}
```

```
$c = new MyReference;
$d = new MyReference;

if ($c === $d) {
    echo "<br> $c dan $d me-refer pada objek yang sama";
} else {
    echo "<br> $c dan $d TIDAK me-refer pada objek yang sama";
}

?>
```

Perhatikan bahwa objek `$b` me-*refer* atau mengacu pada objek `$a`, sehingga `$b` sama dengan `$a` dan dalam satu objek yang sama. Berikutnya kita membuat objek baru bernama `$c` dan `$d`. Meskipun terlihat bahwa kedua objek adalah sama, akan tetapi sebenarnya tidak mengacu pada objek satu. Bisa jadi objek `$c` dan `$d` ditempatkan pada lokasi berbeda, meskipun isinya sama akan tetapi bukan merupakan objek yang sama.

Fitur baru pada PHP 5 juga memungkinkan Anda untuk melakukan *cloning* terhadap objek. Ini sama halnya ketika kita melakukan *reference* objek, sehingga objek yang di-*clone* akan mengacu pada objek yang sama.

```
<?php
/* clone.php */

class MyClone {
    function __clone() {
    echo "Objek di cloning <br>";
    }
}

$a = new MyClone;
$b = clone $a;

if ($a == $b) {
    echo "$a dan $b me-refer pada objek yang sama";
} else {
    echo "$a dan $b TIDAK me-refer pada objek yang sama";
}

?>
```

8.4.7 Static Member dan Method

Dalam model objek baru kali ini, Anda dapat memasukkan *static member* (*property*) kemudian mengaksesnya melalui class ketika mendefinisikan sebuah class.

```
<?php
/* static_member.php */

class TestStatik {

    // member static
    static $static = 1;

    public function inc() {
        return self::$static++;
    }
}

$a = new TestStatik;
$b = new TestStatik;

echo '$a->inc()     = ' . $a->inc()        . "\n";
echo '$b->inc()     = ' . $b->inc()        . "\n";

?>
```

Method static sebenarnya termasuk suatu class, hanya saja di sini tidak melakukan operasi pada class. Hal ini dapat pula diartikan bahwa Anda juga dapat menggunakan method static meskipun tanpa membuat class. Method static tidak didefinisikan dengan *keyword* `$this`, sebagaimana pada member static.

```
<?php
/* static_method.php */

class Test {

    public static function staticMethod() {
        echo "Pemanggilan Test::staticMethod..";
    }

}

Test::staticMethod();

?>
```

8.4.8 Exception

Secara ideal, semua kesalahan program dapat dideteksi pada saat start atau sebelum eksekusi dilakukan. Akan tetapi, untuk kesalahan yang terjadi pada saat *runtime* tidak mungkin dihindari. Kesalahaan pada saat ekseksi akan menimbulkan kegagalan program, yang jika tidak ditangani maka menyebabkan program *abort*.

Banyak sekali kejadian di luar dugaan yang bisa saja terjadi pada program-program yang Anda buat, misalnya *disk error*, file *read-only*, koneksi gagal, dan sebagainya. Pada umumnya, *exception* merupakan pesan kesalahan yang dapat berakibat fatal bagi program, selain itu juga dapat berisi situasi tak terduga lainnya. Nah, dengan melakukan manajemen *exception,* Anda akan dapat memperbaiki kesalahan yang ada pada program.

Dalam kaitannya dengan orientasi objek, PHP 5 menyediakan mekanisme penanganan kesalahan atau *exception handling*. Paradigma yang juga lebih dikenal dengan `try/throw/catch` ini memungkinkan kita untuk melempar objek dari class Exception.

```
class SQLException extends Exception {
  public $problem;
    function __construct($problem) {
        $this->problem = $problem;
    }
}

try {
    ...
    throw new SQLException("Couldn&#8217;t connect to
database");
    ...
} catch (SQLException $e) {
    print "Caught an SQLException with problem $obj->problem";
} catch (Exception $e) {
    print "Caught unrecognized exception";
}
```