

Pertemuan 4 PBO

Pemrograman Berbasis Object

Pengertian Pemrograman Berbasis Object

"Pemrograman berbasis objek adalah sebuah tata cara pembuatan program (programming paradigm) dengan menggunakan konsep "objek", dimana objek ini bisa memiliki data (dikenal dengan istilah atribut) dan kode program dalam bentuk prosedur (dikenal dengan istilah method)."

Secara sederhana, **pemrograman berbasis object** adalah sebuah cara penulisan kode program dengan menggunakan **object** untuk memecahkan masalah. Dalam teori pemrograman, terdapat 3 prinsip dasar yang melandasi pemrograman berbasis object, yakni *encapsulation*, *inheritance* dan *polymorphism*.

Kenapa Harus Pemrograman Berbasis Object?

Untuk bisa menjawab pertanyaan ini, kita harus cari pembandingnya. Selain pemrograman berbasis object, terdapat cara atau "paradigma" lain untuk membuat kode program, yakni **pemrograman prosedural**.

Pemrograman prosedural (*procedural programming*), atau kadang disebut juga sebagai pemrograman fungsional (*functional programming*), adalah cara penulisan kode program yang dipecah ke dalam function-function. Secara umum, pemrograman prosedural lebih sederhana jika dibandingkan dengan pemrograman berbasis object. Dalam pemrograman prosedural, kode program ditulis secara berurutan dari baris paling atas sampai baris paling bawah. Jika kode atau masalah yang dihadapi cukup panjang, kita bisa pecah menjadi beberapa function yang kemudian di satukan kembali di dalam kode program utama.

Tidak ada yang salah dari pemrograman prosedural. Selain praktis, cara pembuatan kode program seperti ini juga sederhana dan mudah dipelajari. Namun seiring kompleksitas aplikasi yang dibuat, pada titik tertentu pemrograman prosedural memiliki beberapa keterbatasan.

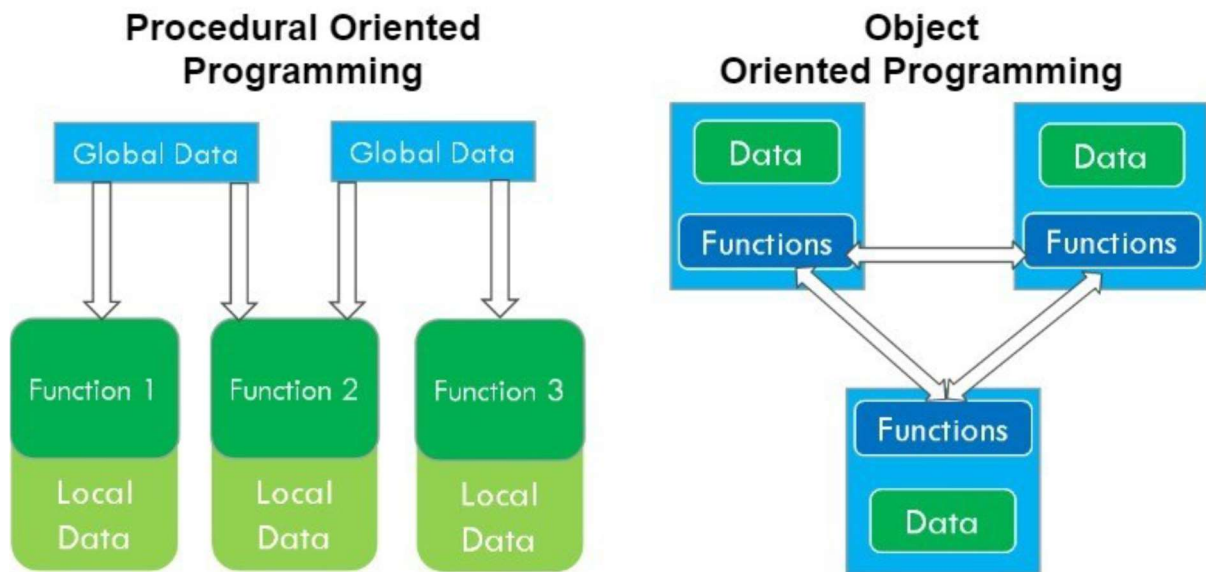
Pertama, tidak ada mekanisme pengelompokkan function. Semua function bisa diakses dari mana saja dan memiliki nama bebas. Jika kode program ini dibuat oleh tim, sangat mungkin seorang anggota tim membuat nama function yang sama persis dengan programmer lain.

Kedua, pemrograman prosedural berfokus ke alur program secara linear (berurutan). Ini membuatnya susah jika suatu saat terdapat perubahan kode program. Satu perubahan di awal akan berdampak ke bagian yang lain.

Ketiga, kode program "terlalu melekat" dengan masalah yang dipecahkan saat ini, sehingga tidak bisa dipakai untuk masalah lain. Sebagai contoh, jika kita memiliki form login, form register dan form edit, agak susah membuat sebuah mekanisme agar ketiganya bisa diproses dengan kode program yang sama.

Masalah inilah yang bisa diatasi dengan lebih mudah jika menggunakan pemrograman berbasis object atau OOP.

Di dalam OOP, kita bisa memakai **perumpamaan objek di dunia nyata**, seperti object *user*, object *produk*, atau object *gambar*, dsb. Setiap object ini punya "dunianya" masing-masing sehingga saling terpisah satu sama lain. Dengan demikian, perubahan di satu bagian tidak akan berdampak langsung ke bagian lain.



Kekurangan OOP

Pertama, kita perlu "perencanaan" sebelum membuat kode program. Perencanaan yang dimaksud adalah object apa saja yang nantinya harus dibuat, lalu bagaimana hubungan satu object dengan object lain. Dalam teori programming, perancangan konsep OOP ini biasanya menggunakan diagram **UML** (Unified Modeling Language) atau **Class Diagram**. Perencanaan inilah yang membuat OOP menjadi *modular*, mudah dikembangkan dan dikelola. Di awal kita sudah harus memikirkan seperti apa alur kode program secara keseluruhan, kemudian bagaimana mengantisipasi jika terdapat perubahan atau butuh penambahan fitur baru.

Kelemahan **kedua** adalah perubahan pola pikir (mindset) agar bisa membuat kode program yang benar-benar menerapkan prinsip pemrograman object, dan ini tidak mudah. Dalam banyak kasus, kita lebih sering menggabungkan konsep pemrograman prosedural dengan pemrograman object. Pemrograman object biasa dipakai untuk mengakses library atau object bawaan PHP (yang berisi fungsi-fungsi tertentu), namun program utama tetap dibuat dalam bentuk prosedural.

Kelemahan **ketiga** dari OOP adalah, untuk website sederhana, kode program yang diperlukan akan lebih panjang daripada pemrograman prosedural. Namun ini sebenarnya sebagai kompensasi dari keunggulan yang ditawarkan oleh OOP.

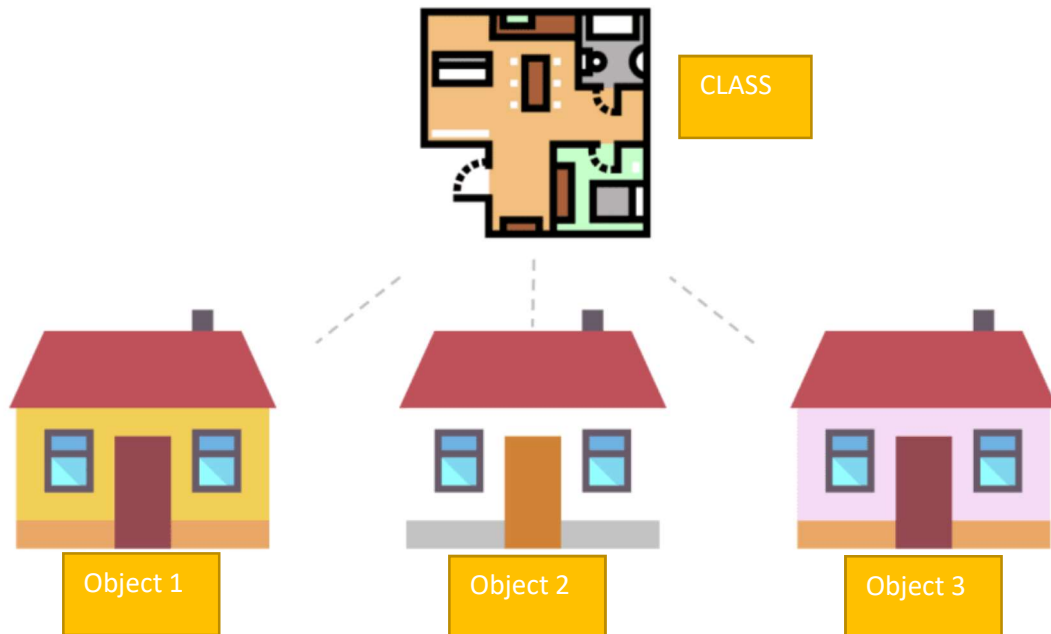
Basic OOP PHP

Class dan Object

Class dan **object** merupakan fondasi paling dasar dari *object oriented programming*, keduanya serupa tapi tak sama. Class adalah *blueprint* atau "cetakan" untuk object. Bisa disebut juga bahwa object adalah implementasi konkret dari sebuah class.

Sebagai analogi, ibaratnya kita ingin membuat **object rumah**. **Class** adalah **gambar desain**

rumah yang dirancang oleh arsitek. Melalui gambar design ini, nantinya bisa dibuat tidak hanya 1 rumah, tapi bisa banyak rumah (seperti yang ada di kompleks perumahan). Di sini, **object rumah adalah implementasi konkret dari class gambar rumah.**



Pemrograman berbasis object mencoba menjadikan object dunia nyata ke dalam konsep pemrograman. Misalnya jika nanti kita membuat kode untuk pemrosesan user (login, register, logout, dst), maka akan ada class **User**. Jika kita ingin membuat website jual beli, nanti akan ada class **Produk**, **Invoice**, dst.

Untuk membuat class di dalam PHP, caranya adalah dengan menulis **keyword class** kemudian diikuti dengan nama class. Isi class nantinya ada di dalam tanda kurung kurawal. Berikut contoh pembuatan class **Produk** di dalam PHP, simpan dengan nama **class.php** pada folder baru **pertemuan_4**:

```
<?php
class Produk {
}
```

Dengan program diatas kita sudah berhasil membuat sebuah class basic dengan nama **Produk**,

Catatan:

Kebiasaan banyak programmer PHP (dan juga di sebagian besar bahasa pemrograman lain), adalah menggunakan huruf besar pada karakter pertama nama class. Daripada membuat class produk, lebih baik menggunakan class Produk. Jika nama class tersebut lebih dari 2 kata, gunakan penulisan seperti class ProdukTelevisi atau class UserAdmin.

Dari penjelasan sebelumnya disebutkan bahwa class hanya **kerangka kerja atau prototype**. Kita tidak akan mengakses class secara langsung, tapi harus melalui object. Object adalah **bentuk konkret dari class**

membuat sebuah object.

Untuk membuat object dari suatu class, gunakan keyword **new** seperti contoh berikut, simpan dengan nama **object.php**:

```
<?php
class Produk {

}

$televisi = new Produk();      //object 1
$buku = new Produk();          //object 2
$smartphone = new Produk();    //object 3
$speaker = new Produk();       //object 4 dan seterusnya
```

Program diatas artinya kita telah membuat **4 buah object dari class Produk**. Lihat strukturnya, object berada diluar class atau diluar tanda kurung kurawal. Jumlah object yang dibuat bisa tak terbatas sesuai kebutuhan.

Catatan:

Proses pembuatan object ini dikenal dengan istilah *instansiasi object* atau *object instantiation*. !!!

Bila program diatas dipanggil melalui browser maka tidak akan menghasilkan sesuatu apapun meskipun tidak error, karena memang belum ada method yang menjalankan fungsi apapun pada class maupun object.

Ok agar ada sedikit gambaran tentang object yang telah kita buat, kita akan menambahkan fungsi **var_dump()** untuk melihat detail dari variable yang kita buat. Lengkapi program sebelumnya atau boleh buat baru.

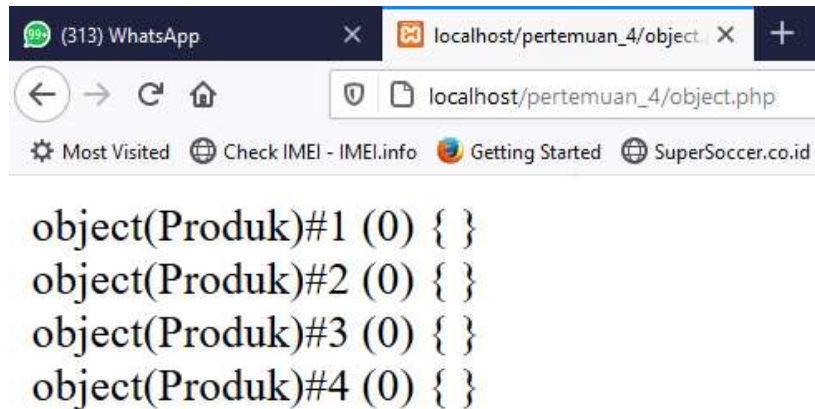
```
<?php
class Produk {

}

$televisi = new Produk();
$buku = new Produk();
$smartphone = new Produk();
$speaker = new Produk();

var_dump($televisi);    // object(Produk)#1 (0) { }
echo "<br>";
var_dump($buku);        // object(Produk)#2 (0) { }
echo "<br>";
var_dump($smartphone);  // object(Produk)#3 (0) { }
echo "<br>";
var_dump($speaker);     // object(Produk)#4 (0) { }
```

apabila program diatas kembali dipanggil melalui browser maka akan ada output yang ditampilkan.



Catatan:

Hasil perintah `var_dump($televisi)` bisa dibaca bahwa `$televisi` adalah **object** dari **class** `Produk`. Angka #1 menandakan ini adalah object `Produk` pertama yang ada di dalam file PHP. Angka (0) menandakan jumlah *property* yang ada di dalam object (akan kita bahas sesaat lagi).

PAHAM???

Property dan Method

Property dan **method** tidak lain adalah sebutan untuk *variabel* dan *function* yang berada di dalam class. Cara penulisannya pun sama seperti variabel dan function, tapi dengan tambahan *access modifier* di awal penulisan seperti contoh berikut, simpan dengan nama `property_method.php`:

```
<?php
class Produk { //class dengan 3 property dan 1 fungsi
    public $id = "001"; //property class harus mempunyai access modifier
    public $merek = "Samsung";
    public $harga = 4000000;

    public function pesanProduk(){
        return "Produk dipesan...";
    }
}
```

Program diatas hanya melengkapi class yang telah kita buat sebelumnya, dengan menambahkan property pada class, ada 3 property yang ditambahkan yaitu `$id`, `$merek`,

\$harga, dengan access modifier atau visibility **public** (akan dijelaskan selanjutnya). Property yang dibuat juga telah diisi langsung dengan sebuah contoh nilai.

Selain menambahkan property kita juga menambahkan sebuah method/function di dalam class dengan nama **pesanProduk()**, dimana fungsi sederhana ini akan mengembalikan nilai (**return**) berupa teks Produk dipesan...

Catatan:

Cara penulisan sebuah fungsi diawali dengan nama fungsi diikuti tanda kurung buka dan kurung tutup contoh: **pesanProduk()**.

Setelah itu untuk menuliskan statement dari fungsi diawali dan diakhiri dengan tanda kurung kurawal, contoh

```
public function pesanProduk(){  
    return "Produk dipesan...";  
}
```

Ini adalah bentuk paling sederhana dari sebuah fungsi, untuk bentuk lain akan kita pelajari selanjutnya.

Paham???

Cara Mengakses Property dan Method

Bagaimana cara mengakses property dan method? **Ingat bahwa dalam pemrograman berbasis object, yang akan kita akses adalah object, bukan class.** Artinya apa??? Agar kita bisa mengakses property dan method dari sebuah class maka kita harus membuat object dari class tersebut (sudah kita lakukan sebelumnya cara membuat sebuah object). Kita akan membuat sebuah contoh object untuk melengkapi latihan kita.

```
<?php  
class Produk {  
    public $id = "001";  
    public $merek = "Samsung";  
    public $harga = 4000000;  
  
    public function pesanProduk(){  
        return "Produk dipesan...";  
    }  
}  
  
$televisi = new Produk(); //buat object dari class produk  
  
echo $televisi->id; // 001 (object televisi mengakses atribut id dari class)  
echo "<br>";  
echo $televisi->merek; // Samsung  
echo "<br>";  
echo $televisi->harga; // 4000000  
echo "<br>";
```

```
echo $televisi->pesanProduk();  
// Produk dipesan... (object televisi mengakses fungsi dari class)
```

program diatas menjelaskan, setelah kita memiliki sebuah class lengkap dengan attribute dan method maka untuk dapat mengakses attribute dan method tersebut kita harus membuat sebuah object pada contoh di atas adalah televisi.

Karena televisi merupakan object dari class produk maka secara otomatis televisi mewarisi seluruh property dan method yang dimiliki oleh class Produk.

Sehingga pada saat kita ingin menampilkan isi dari property dari televisi caranya adalah

```
echo $televisi->id;
```

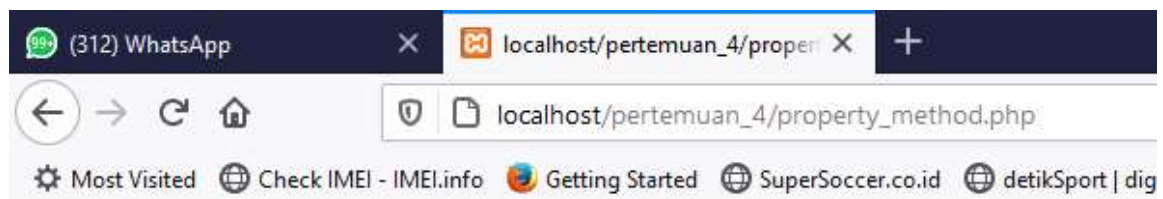
artinya object televisi mengakses property id. Cara yang sama untuk mengakses property yang lain. Fungsi echo agar isi dari attribute dapat tampil pada browser.

Sedangkan untuk mengakses fungsi dilakukan cara yang sama yaitu

```
echo $televisi->pesanProduk();
```

artinya object televisi mengakses fungsi pesanProduk(), lihat perbedaan saat mengakses property. Saat mengakses fungsi ciri cirinya adalah ada tanda kurung buka dan kurung tutup.

Apabila kita akses melalui browser akan memberikan hasil sebagai berikut:



001
Samsung
4000000
Produk dipesan...

SILAKAN DIPAHAMI DULU...!!!

MEMBUAT 2 OBJECT DARI CLASS

Kali ini kita akan mencoba membuat 2 buah object berbeda dari class Produk, kita akan buat object televisi dan mesinCuci.

Berikut kode programnya, simpan dengan nama **object_lanjut.php**

```
<?php
class Produk {
    public $id = "000"; // nilai awal bebas
    public $merek = ""; //boleh berupa string kosong
    public $harga = 0; //boleh berupa angka 0 untuk bilangan

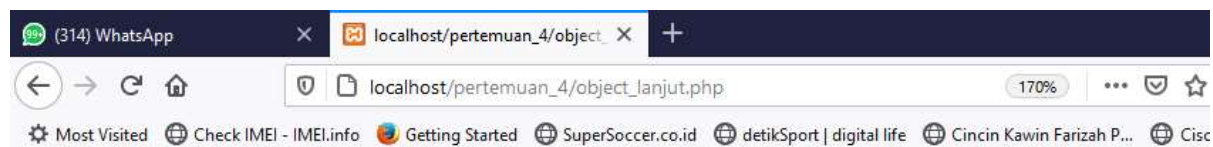
    public function pesanProduk(){
        return "Produk dipesan...";
    }
}

$televisi = new Produk(); //object pertama
$televisi->id = "001";
$televisi->merek = "LG";
$televisi->harga = 1500000;

$mesinCuci = new Produk(); //object kedua
$mesinCuci->id = "002";
$mesinCuci->merek = "Samsung";
$mesinCuci->harga = 1500000;

print_r ($televisi); // print_r() fungsi untuk mencetak detail object
echo "<br>";
print_r ($mesinCuci);
echo "<br>";
```

perbedaan dengan program sebelumnya adalah pada program ini kita memberikan nilai dari masing-masing property setelah kita membuat object nya, sedikit berbeda dengan program sebelumnya, silakan dibandingkan. **Nilai awal property pada saat definisi class akan ditimpa dengan nilai yang baru.**



```
Produk Object ( [id] => 001 [merek] => 4000000 [harga] => 1500000 )
Produk Object ( [id] => 002 [merek] => LG [harga] => 1500000 )
```

APAKAH KALIAN PAHAM DENGAN OUTPUT PROGRAM DIATAS???

Yup baris pertama menampilkan detail dari object Televisi dan baris kedua menampilkan detail dari object mesinCuci.

Fungsi print_r() dapat menampilkan detail property beserta nilainya dari sebuah object

Catatan:

Pahami secara detail dan pelan-pelan apa yang sudah kalian lakukan diatas.

Bagaimana cara membuat kelas, bagaimana cara mendefinisikan property class dan method class, bagaimana mengisi sebuah nilai pada property.

Ingat pahami jangan asal program jalan lalu ditinggalkan!!!

Pseudo-variable \$this

Kali ini kita akan membahas sebuah konsep yang sering membuat bingung tapi sangat sering digunakan dalam konsep OOP, yakni tentang **pseudo-variable \$this**. Sebelum ke sana, silahkan pelajari sebentar kode program berikut ini, simpan dengan nama **this.php**:

```
<?php
class Produk {
    public $jenis = "";
    public $merek = "";

    public function pesanProdukTelevisi(){
        return "Televisi dipesan...";
    }

    public function pesanProdukMesinCuci(){
        return "Mesin cuci dipesan...";
    }
}

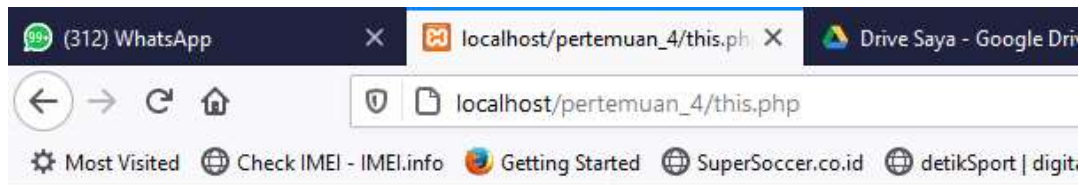
$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";

$produk02 = new Produk();
$produk02->jenis = "Mesin cuci";
$produk02->merek = "LG";

echo $produk01->pesanProdukTelevisi();    // Televisi dipesan...
echo "<br>";
echo $produk02->pesanProdukMesinCuci();    // Mesin cuci dipesan...
```

langsung pada perbedaan dengan program kita sebelumnya, class produk ini memiliki 2 fungsi masing-masing untuk **pesanProdukTelevisi()** dan **pesanProdukMesinCuci()** sesuai dengan produk yang rencananya akan dibuat yaitu **produk01** dan **produk02** sampai sini masih aman ya.

Ok sekarang coba akses program diatas melalui browser!!



Televisi dipesan...
Mesin cuci dipesan...

Apakah ada yang salah dengan program diatas??? Tentu tidak, program dapat berjalan dengan baik, menampilkan output sesuai yang diinginkan..

LALU MASALAHNYA DIMANA???

Masalahnya adalah dengan struktur class seperti diatas maka class Produk **menjadi tidak fleksible**, maksudnya????

Bayangkan kalian akan membuat 100 produk dan masing2 produk memiliki fungsi pesanProduk() sesuai dengan 2 fungsi sebelumnya, artinya dalam class produk kita perlu membuat 100 fungsi pesanProduk juga sesuai dengan produk yang dibuat. Dan ini sangat merepotkan serta tidak fleksible, bukan seperti itu cara kerja OOP.

Solusinya???

Ada cara yang lebih baik, dari pada membuat 1 method untuk setiap tipe produk, kita bisa rancang method "generik" bernama pesanProduk() dengan konsep sebagai berikut simpan dengan nama **this_1.php**:

```
<?php
class Produk {
    public $jenis = "";
    public $merek = "";

    public function pesanProduk(){
        return $jenis." dipesan...";
    }
}

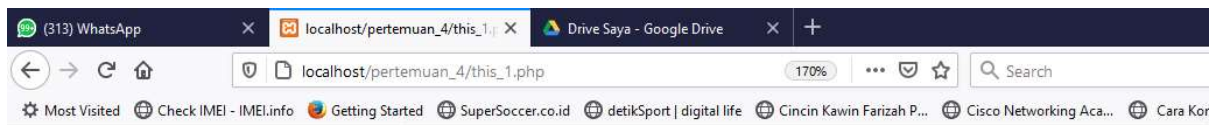
$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";

echo $produk01->pesanProduk();
```

program diatas mirip dengan sebelumnya hanya saja kita memodifikasi bagian fungsi dengan membuat sebuah fungsi generik **pesanProduk()** yang nantinya dapat digunakan oleh seluruh produk yang dibuat.

Pada contoh diatas kita membuat 1 buah produk baru produk01.

Ok sekarang kalian coba jalankan programnya dan lihat hasilnya



Notice: Undefined variable: jenis in C:\xampp\htdocs\pertemuan_4\this_1.php on line 7 dipesan...

Yup ternyata ada error yang tidak diharapkan, harapan kita adalah program diatas bisa menampilkan teks “**Televisi dipesan...**” tetapi ternyata variable jenis yang harusnya berisi televisi tidak dikenali...

Pesan error di atas mengatakan bahwa PHP tidak menemukan variabel \$jenis di baris 7. Loh, bukannya di baris 3 kita sudah mendefinisikan property \$jenis? yang kemudian diisi kembali di baris 12? kenapa PHP tidak bisa membacanya?

Untuk bisa memahami apa yang terjadi, kita harus kembali kepada definisi dari object, yakni implementasi konkret sebuah class. Selama pemrosesan kode PHP, yang seharusnya kita akses adalah object, bukan class. Class adalah tempat untuk pembuatan definisi saja (sebagai *blueprint*).

PENTING!!!

Ketika saya menulis perintah `return $jenis." dipesan..."` di dalam method `pesanProduk()`, artinya saya sedang mencoba mengakses **property milik class**, padahal seharusnya yang diakses itu adalah **property milik object**. Di sinilah kita butuh sebuah variabel bantu sebagai penanda bahwa yang ingin diakses adalah property milik object, **bukan milik class**. Variabel bantu tersebut adalah **\$this**.

Agar method `pesanProduk()` bisa diproses sebagaimana mestinya, saya harus tulis ulang menjadi: `return $this->jenis." dipesan..."`. Berikut perubahan dari kode program sebelumnya:

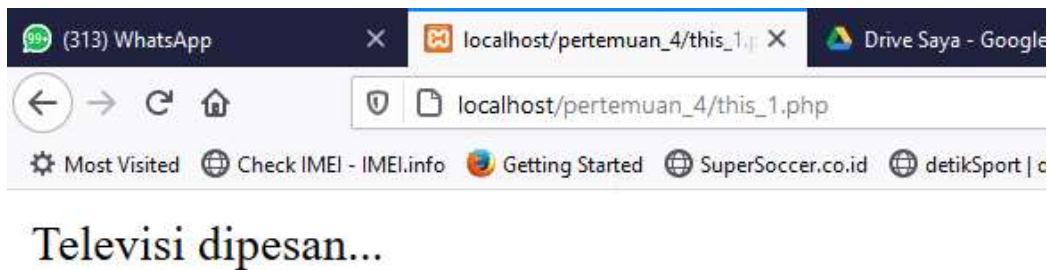
```
<?php
class Produk {
    public $jenis = "";
    public $merek = "";

    public function pesanProduk(){
        return $this->jenis." dipesan..."; //perubahan di sini!!!
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";

echo $produk01->pesanProduk();
```

coba jalankan ulang program kalian dan pahami...



Pesan akan muncul sesuai dengan yang diharapkan.. sampai sini jelas???
Pahami lagi ya, baca pelan-pelan

Buat latihan kedua dengan memodifikasi sedikit baris program sebelumnya, simpan dengan nama **this_2.php**

```
<?php
class Produk {
    public $jenis;
    public $merek;

    public function pesanProduk(){
        return $this->jenis." ".$this->merek." dipesan...";
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";

$produk02 = new Produk();
$produk02->jenis = "Mesin cuci";
$produk02->merek = "LG";

echo $produk01->pesanProduk();    // Televisi Samsung dipesan...
echo "<br>";
echo $produk02->pesanProduk();    // Mesin cuci LG dipesan...
```

pada contoh ini terlihat jelas perbedaan dengan latihan **this.php sebelumnya**, silakan dibandingkan. Fungsi pada class hanya ada 1 fungsi generic yaitu pesanProduk() yang dapat digunakan berulang kali pada object yang dibentuk..

Pertemuan 5 PBO

Pada pertemuan kali ini kita masih akan melanjutkan pemahaman materi sebelumnya yang telah kita pelajari.

Kita akan membahas lebih jauh pemahaman konsep THIS dalam OOP.

Kita mulai dengan contoh berikut untuk mengingat kembali materi sebelumnya, simpan dengan nama **this_3.php** dalam direktori baru **pertemuan_5**

```
<?php
class Produk {
    public $jenis;
    public $merek;

    public function pesanProduk(){
        return $this->jenis." ".$this->merek." dipesan...";
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";

$produk02 = new Produk();
$produk02->jenis = "Mesin cuci";
$produk02->merek = "LG";

echo $produk01->pesanProduk();    // Televisi Samsung dipesan...
echo "<br>";
echo $produk02->pesanProduk();    // Mesin cuci LG dipesan...
```

program di atas masih sama dengan sebelumnya dengan sedikit modifikasi,

1. Saya mau menunjukkan bahwa pada saat deklarasi property sebuah class tidak harus memberikan nilai awal, lihat baris 3 dan 4 pada program.
2. Penulisan \$this->sesuatu berada pada saat pendefinisian class, bukan pada object

Silakan coba program diatas dan jalankan.

Latihan buatlah class Mahasiswa dan class Dosen, property bebas minimal 4 atribut silakan berkreasi, buat juga fungsi pada masing-masing class untuk menampilkan data dari dosen dan mahasiswa. Object yang harus dibuat untuk masing masing class mahasiswa dan dosen adalah 3 object. Simpan dengan nama mahasiswa.php dan dosen.php

Latihan

Untuk menguji materi kita sampai saat ini, saya membuat sebuah soal sederhana: Buatlah sebuah class dengan nama Produk. Class Produk ini memiliki 3 buah property: \$jenis, \$merek, dan \$stok, serta 2 buah method: pesanProduk() dan cekStok(). Property \$jenis dan \$merek dipakai untuk menampung data string seperti contoh kita sebelumnya. Sedangkan property \$stok dipakai untuk menampung angka integer yang menunjukkan sisa stok produk, misalnya 100.

Pada saat method pesanProduk() dipanggil, ini akan mengurangi stok 1 buah. Jika sebelumnya stok ada 100, maka setelah pemanggilan method pesanProduk(), stok menjadi 99. Jika method ini dipanggil lagi, maka jumlah stok kembali berkurang 1 menjadi 98, dst.

Silahkan anda luangkan waktu sebentar untuk memikirkan seperti apa kode untuk pembuatan class Produk, lalu coba rancang kode programnya. Kita perlu menggunakan variabel \$this untuk mengakses property \$stok di dalam class, untuk digunakan dalam fungsi. Pada latihan in kita sudah mulai menggukan fungsi untuk perhitungan aritmatika, tidak hanya menampilkan nilai strings seperti latihan sebelumnya.

Berikut solusi programnya, simpan dalam file **cek_stok.php** (ingat pahami program berikut sampai tau alurnya).

```
<?php
class Produk {
    public $jenis;
    public $merek;
    public $stok;

    public function pesanProduk(){
        $this->stok = $this->stok - 1;
    }

    public function cekStok(){
        return "Sisa stok: ". $this->stok ."<br>";
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";
$produk01->stok = 54;

echo $produk01->cekStok();    // Sisa stok: 54

$produk01->pesanProduk();
$produk01->pesanProduk();
$produk01->pesanProduk();

echo $produk01->cekStok();    // Sisa stok: 51
```

silakan dipanggil program diatas melalui browser dan lihat hasilnya

saya akan bahas sedikit mengenai program di atas, langsung ke bagian fungsi **pesanProduk()**.

```
public function pesanProduk(){  
    $this->stok = $this->stok - 1;  
}
```

Fungsi pesanProduk() akan melakukan tugas mengurangi jumlah stok terakhir sebanyak 1 apabila fungsi tersebut dipanggil, jadi apabila kita panggil fungsi pesan produk sebanyak 3 kali maka stok otomatis akan berkurang 3. Begini cara panggilnya

```
$produk01->pesanProduk();
```

Artinya object \$produk01 memanggil fungsi pesanProduk()

Dalam fungsi pesan produk pseudo variable \$this digunakan karena akan mengakses property stok dari object yang terbentuk dari class produk, tanpa pseudo variable \$this maka property tidak akan bisa dikenali. **Ingat materi sebelumnya**

fungsi cekStok() berguna untuk menampilkan jumlah stok terakhir dari barang, apabila fungsi pesan produk belum pernah dipanggil maka nilai stok yang ditampilkan dari fungsi cekStok() adalah nilai awal stok yang diberikan padasaat deklarasi variable.

Pada contoh diatas dilakukan pemanggilan fungsi pesanProduk() sebanyak 3 kali artinya akan terjadi pengurangan stok sebanyak 3. **Paham ya.**

Argument Method

OK, kita menuju next level. Kali ini kita akan membahas mengenai argument method.

Method di dalam sebuah class atau object tidak berbeda dengan function, oleh karena itu seluruh fitur-fitur function juga bisa kita terapkan. Salah satunya mengirim *argument*.

Argument adalah sebutan untuk nilai input yang diberikan pada saat pemanggilan function. Sebagai contoh, PHP memiliki function bawaan sqrt() untuk mencari nilai akar kuadrat. Function sqrt() butuh 1 argumen berupa angka yang akan dicari nilai akar kuadratnya. Untuk mencari akar kuadrat dari 49, perintahnya adalah sqrt(49), angka **49** di sini merupakan sebuah **argument**. Pahami ya penjelasan sederhananya??

Latihan pertama kita akan melanjutkan program sebelumnya dengan menambahkan fungsi baru, sebelumnya kita sudah punya fungsi pesanProduk() dan cekStok(), nah kali ini saya ingin membuat method borongProduk(). Jika pada pesanProduk() hanya bisa memesan 1 buah produk saja, pada method borongProduk() ini bisa membeli banyak produk sekaligus sesuai yang kita inginkan. **Jumlah produk yang dibeli nantinya akan menjadi sebuah argument, sehingga kita akan menambah sebuah parameter pada fungsi borongProduk()**. Berikut kode programnya, simpan dengan nama **argument_1.php**:

Lho kok ada istilah argument dan parameter??? Apa bedanya??? Next akan saya bahas.


```

<?php
class Produk {
    public $jenis;
    public $merek;
    public $stok;

    public function pesanProduk(){
        $this->stok = $this->stok - 1;
    }

    public function borongProduk($jumlah){
        $this->stok = $this->stok - $jumlah;
    }

    public function cekStok(){
        return "Sisa stok: ". $this->stok . "<br>";
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";
$produk01->stok = 54;

echo $produk01->cekStok();    // Sisa stok: 54

$produk01->borongProduk(10);
echo $produk01->cekStok();    // Sisa stok: 44

$produk01->borongProduk(25);
echo $produk01->cekStok();    // Sisa stok: 19

```

secara umum program diatas sama dengan program sebelumnya hanya saja ditambahkan fungsi borongProduk().

Kita focus dulu di sini

```

public function borongProduk($jumlah){
    $this->stok = $this->stok - $jumlah;
}

```

Diatas kita lihat dalam fungsi ada sebuah **parameter \$jumlah** fungsi parameter ini adalah untuk menampung nilai yang dikirim dari object saat fungsi dipanggil, nilai yang diterima akan digunakan untuk perhitungan didalam fungsi. Ketika method borongProduk(\$jumlah) dipanggil, kurangkan nilai \$this->stok dengan parameter \$jumlah, lalu simpan kembali ke dalam \$this->stok. gimana caranya?? Berikut ini cara pemanggilannya.

```

$produk01->borongProduk(10);

```

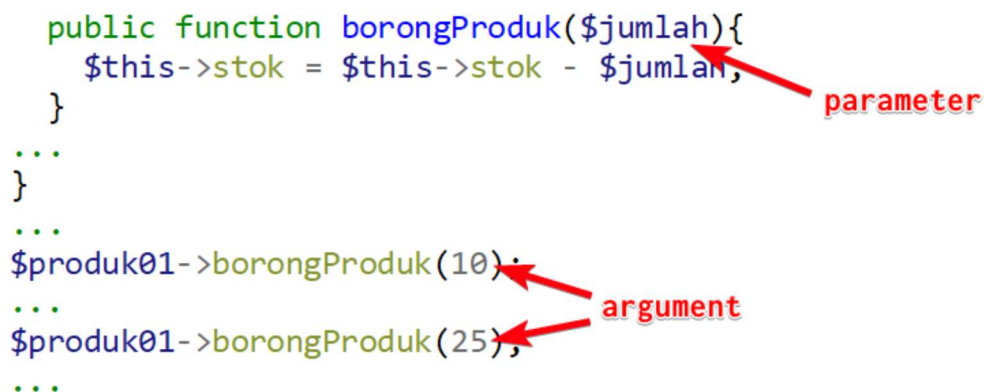
Program diatas artinya object \$produk01 memanggil fungsi borongProduk() dengan mengirimkan **argumen** 10 untuk nantinya ditangkap oleh argument \$jumlah, **gampang ya???**

Argument vs Parameter

Istilah **argument** dan **parameter** juga serupa tapi tak sama. Keduanya sama-sama dipakai untuk menyebut nilai yang diinput ke dalam function atau method. **Argument** adalah sebutan untuk inputan pada saat *pemanggilan method*, sedangkan **parameter** adalah sebutan untuk inputan pada saat *pendefinisian method*.

Dalam contoh di atas, variabel \$jumlah adalah sebuah *parameter*, sedangkan angka 10 dan 25 pada saat pemanggilan method adalah *argument*. Kedua istilah ini sering di pertukarkan, malah di dokumentasi resmi PHP (PHP Manual) istilah *argument* dipakai untuk menyebut keduanya.

```
public function borongProduk($jumlah){
    $this->stok = $this->stok - $jumlah;
}
...
}
...
$produk01->borongProduk(10);
...
$produk01->borongProduk(25);
...
```



The diagram illustrates the difference between parameters and arguments in the provided PHP code. A red arrow labeled "parameter" points to the variable `$jumlah` in the function definition `public function borongProduk($jumlah){`. Another red arrow labeled "argument" points to the value `10` in the function call `$produk01->borongProduk(10);`. A second red arrow also labeled "argument" points to the value `25` in the function call `$produk01->borongProduk(25);`.

Default Parameter

Kita perlu mengenal juga default parameter, fungsinya adalah ketika method dipanggil tanpa argument, nilai default-lah yang akan dipakai.

Lengkapi program sebelumnya dan sipan dengan nama **default_parameter.php**

```
<?php
class Produk {
    public $jenis;
    public $merek;
    public $stok;

    public function borongProduk($jumlah = 10){
        $this->stok = $this->stok - $jumlah;
    }

    public function cekStok(){
        return "Sisa stok: ". $this->stok . "<br>";
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
```

```
$produk01->merek = "Samsung";
$produk01->stok = 54;

$produk01->borongProduk();
echo $produk01->cekStok(); // Sisa stok: 44

$produk01->borongProduk(20);
echo $produk01->cekStok(); // Sisa stok: 44
```

perubahan yang terjadi adalah pada baris berikut dengan menambahkan nilai default

```
public function borongProduk($jumlah = 10){
    $this->stok = $this->stok - $jumlah;
}
```

Kita berikan nilai default pada parameter jumlah yaitu 10. Sehingga bila terjadi pemanggilan seperti dibawah tidak akan menimbulkan error.

```
$produk01->borongProduk();
```

Baris di atas kita memanggil method borongProduk() tanpa mengirimkan nilai argument, tapi karena kita sudah memberikan nilai default program dapat berjalan dengan benar.

Catatan:

Apabila sebuah method memiliki parameter maka pada saat pemanggilan method tersebut juga harus mengirimkan argument yang sesuai, kecuali kita sudah menetapkan nilai default. Apabila tidak maka akan muncul error, silakan dicoba memunculkan errornya.

Ok kita menuju next level dengan contoh soal yang sedikit lebih rumit.

Latihan. Simpan dengan nama argument_2.php

saya ingin membuat method tambahstok() ke dalam class Produk. Sesuai dengan namanya, method ini dipakai untuk menambah nilai \$stok. Method tambahstok() nantinya bisa menerima 1 argumen, yakni jumlah stok yang akan ditambah.

Jika method ini dipanggil tanpa argument, maka dianggap stok bertambah 1 lusin (12 buah). Karena ukuran gudang yang terbatas, \$stok dibatasi maksimal 100 barang. Jika method tambahstok() dipanggil dan ternyata itu akan melewati 100 stok, batalkan penambahan dan tampilkan pesan kesalahan.

Silakan pahami soalnya, apakah sudah ada gambaran solusi???

Yup pada soal tersebut secara umum mirip dengan contoh sebelumnya, tapi kita akan memerlukan sebuah logika if untuk mengecek apakah stok melewati batas yang ditentukan atau tidak. Berikut solusi programnya.

Saya akan bahas bagian-bagian penting dari program karena secara umum sama seperti sebelumnya.

```

<?php
class Produk {
    public $jenis;
    public $merek;
    public $stok;

    public function tambahStok($jumlah = 12){

        $totalStok = $this->stok + $jumlah;

        if ($totalStok <=100){
            $this->stok = $totalStok;
            $pesan = "Stok berhasil ditambah <br>";
            $pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";
        }
        else {
            $pesan = "Maaf, stok sudah penuh. Penambahan stok dibatalkan <br>";
            $pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";
        }
        return $pesan;
    }
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";
$produk01->stok = 54;

echo $produk01->tambahStok();
echo "<br>";
echo $produk01->tambahStok(20);
echo "<br>";
echo $produk01->tambahStok(15);

```

Hasil kode program:

```

Stok berhasil ditambah
Jumlah stok saat ini: 66
Stok berhasil ditambah
Jumlah stok saat ini: 86
Maaf, stok sudah penuh. Penambahan stok dibatalkan
Jumlah stok saat ini: 86

```

pembahasan:

```
public function tambahStok($jumlah = 12)
```

Kita menambahkan default parameter \$jumlah=12 karena pada soal ada keterangan bila method dipanggil tanpa mengirim argument maka jumlah akan diisi nilai 1 lusin (12).

```
$totalStok = $this->stok + $jumlah;
```

Variable \$totalStok untuk menyimpan sementara hasil penambahan stok dari stok sebelumnya, ingat kenapa menggunakan \$this->stok karena fungsi tambahStok() mengakses property milik object.

```
if ($totalStok <=100){  
    $this->stok = $totalStok;  
    $pesan = "Stok berhasil ditambah <br>";  
    $pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";  
}  
else {  
    $pesan = "Maaf, stok sudah penuh. Penambahan stok dibatalkan <br>";  
    $pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";  
}  
return $pesan;
```

ini bagian pentingnya, logika if digunakan untuk melihat apakah nilai \$totalStok <= 100 atau tidak, karena bila stok sudah lebih dari 100 proses penambahan akan dibatalkan, dan bila \$totalStok masih kurang dari 100 maka proses penambahan sukses dan akan ditampilkan pesan.

Kita sudah mempelajari logika if sebelumnya, klo masih bingung silakan buka lagi materi sebelumnya.

```
$pesan = "Stok berhasil ditambah <br>";  
$pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";
```

Ini bagian tidak terlalu penting tapi perlu dimengerti. Perhatikan tanda titik dibelakang variable \$pesan kedua, itu adalah fungsi concat untuk menggabungkan string.

```
return $pesan;
```

ini adalah bagian output dari sebuah method, karena tanpa return sebuah method tidak akan mengembalikan sebuah nilai. \$pesan adalah nilai yang dikembalikan ke object yang memanggil fungsi.

Ok kira-kira seperti itu penjelasannya silakan jalankan programnya, apabila ada yang perlu ditanyakan silakan disampaikan di grup.

Lalu bagaimana bila parameter fungsi lebih dari satu??? Ya tinggal disesuaikan saja dengan kebutuhan kasusnya.

Contoh method **luasPersegiPanjang(\$panjang, \$lebar)**, disini ada 2 buah parameter yang dibutuhkan, karena memang rumus luas persegi panjang adalah panjang x lebar

Latihan

Buat class **PersegiPanjang**, **Lingkaran** dan **segitiga** simpan dalam 3 file berbeda. Masing masing class memiliki fungsi untuk menghitung Luas dan Keliling. Lalu buat satu object dari masing-masing class untuk mengimplementasikan fungsi nya.

Constructor dan Destructor

Constructor adalah method khusus yang otomatis dijalankan ketika sebuah object dibuat (**yakni pada saat proses *inisialisasi* dengan perintah new**). Sedangkan **destructor** adalah method khusus yang otomatis dijalankan pada saat object di hapus.

Dengan *constructor*, kita bisa membuat "persiapan" untuk sebuah object, seperti mengisi nilai awal atau membuka koneksi ke database. Sebaliknya, *destructor* bisa dipakai untuk proses "pembersihan" setelah object dihapus seperti menutup koneksi ke database, tujuannya agar ruang memory bisa kembali kosong.

Dibandingkan *constructor*, **destructor relatif jarang dipakai** karena PHP sudah memiliki sistem "*garbage collection*" internal, yang secara otomatis akan menghapus semua object dan membersihkan memory tanpa perlu perintah tambahan.

Untuk membuat constructor di PHP, rancang sebuah method dengan nama `__construct()`, sedangkan untuk destructor, buat method dengan nama `__destruct()`.

Ok agar ada gambaran kita langsung membuat contoh bagaimana constructor bekerja. Buat file dengan nama **constructor_1.php**

```
<?php
class Produk {

    public function __construct(){ // underscore nya 2 ya
        echo "Constructor dijalankan... <br>";
    }

}

$produk01 = new Produk();
$produk02 = new Produk();
```

Hasil kode program:

```
Constructor dijalankan...
Constructor dijalankan...
```

Dalam kode di atas saya merancang class **Produk** yang berisi 1 buah method, yakni `__construct()`. Di dalamnya terdapat satu perintah berupa `echo "Constructor dijalankan...
"`.

Hasilnya, begitu class **Produk** ini diinisialisasi ke object `$produk01` dan `$produk02`, isi dari method `__construct()` otomatis dijalankan sebanyak 2 kali.

Artinya dengan adanya constructor setiap kali kita membuat object tanpa memanggil fungsi constructor otomatis fungsi tersebut akan langsung dijalankan. Ini sering digunakan pada fungsi-fungsi khusus misal koneksi database sehingga kita tidak perlu melakukan pemanggilan fungsi untuk koneksi ke database setiap kali sistem dijalankan.

Sebelumnya kita tau bahwa method class harus dipanggil secara manual, khusus untuk constructor akan dijalankan otomatis saat object dibuat.

Pahami!!!

Ok kita menuju next level untuk memahami penggunaan constructor.

Untuk implementasi yang lebih "real", constructor sering dipakai dalam proses *instansiasi* object, yakni untuk memberikan nilai awal ke dalam object. Proses ini sebenarnya sudah kita lakukan namun masih manual seperti contoh berikut, simpan dengan nama

constructor_2.php:

```
<?php
class Produk {
    public $jenis;
    public $merek;
    public $stok;
}

$produk01 = new Produk();
$produk01->jenis = "Televisi";
$produk01->merek = "Samsung";
$produk01->stok = 20;

$produk02 = new Produk();
$produk02->jenis = "Mesin cuci";
$produk02->merek = "LG";
$produk02->stok = 10;

print_r ($produk01);
echo "<br>";
print_r ($produk02);
```

program diatas pernah kita buat sebelumnya, intinya adalah membuat class dan membuat object dari class tersebut, fungsi print_r() digunakan untuk melihat detail dari object yang dibuat.

Hasil kode program:

```
Produk Object ( [jenis] => Televisi [merek] => Samsung [stok] => 20 )
Produk Object ( [jenis] => Mesin cuci [merek] => LG [stok] => 10 )
```

Dari hasil diatas kita bisa lihat bahwa ada 2 object yang dibuat dengan property masing masing object sesuai dengan yang di inputkan,

Panulisan [jenis]=>Televisi pada hasil print_r() adalah assosiatif array akan kita bahas selanjutnya.

Apakah cara diatas salah??? Tentu tidak, tapi setelah kita mengenal constructor ada cara yang lebih simple untuk inisiasi sebuah object, dan tentunya akan lebih menyingkat penulisan kode program

Kita lihat perubahannya dengan memanfaatkan constructor, simpan dengan nama **constructor_3.php**

```
<?php
class Produk {
    public $jenis;
    public $merek;
    public $stok;

    public function __construct($a, $b, $c){
        $this->jenis = $a;
        $this->merek = $b;
        $this->stok = $c;
    }
}

$produk01 = new Produk("Televisi","Samsung",20);
$produk02 = new Produk("Mesin cuci","LG", 10);

print_r ($produk01);
echo "<br>";
print_r ($produk02);
```

hasil dari program ini akan sama dengan program sebelumnya, tapi apakah kalian melihat perbedaan struktur programnya???? Yup, kita akan bahas perbagian.

```
public function __construct($a, $b, $c){
    $this->jenis = $a;
    $this->merek = $b;
    $this->stok = $c;
}
```

Baris diatas adalah contoh dari implementasi constructor, dimana kita membuat fungsi construct dengan **3 parameter** \$a, \$b, \$c. kenapa 3 parameter??? Ya sesuai dengan property class yang nantinya akan digunakan untuk membuat object.

kemudian diinput ke dalam property \$jenis, \$merek dan \$stok. Karena property ini adalah "kepunyaan" object, maka kita harus menggunakan *pseudo variabel* \$this->\$jenis, \$this->\$merek, dan \$this->\$stok.

Fungsi construct akan menerima argument yang dikirimkan oleh object, untuk diberikan pada masing-masing propertinya.

Lalu bagian baris ini juga yang paling terlihat perbedaannya.

```
$produk01 = new Produk("Televisi","Samsung",20);  
$produk02 = new Produk("Mesin cuci","LG", 10);
```

Jika sebelumnya saat kita membuat sebuah object hanya dengan perintah new Produk() tanpa mengirim argument, kali ini kita membuat object dengan sekaligus mengirimkan argument untuk mengisi nilai dari masing-masing property. Paham ya. Tidak ada lagi proses inisiasi nilai pada property object setelah object dibuat. **Lebih simple ya???**

Silakan amati dan bandingkan dengan struktur program sebelumnya yang tanpa menggunakan constructor sampai benar benar paham. Jangan skip apabila belum paham lebih baik ditanyakan.

Sekarang perhatikan cara penulisan parameter pada fungsi construct

```
__construct($a, $b, $c)
```

Kenapa \$a,\$b,\$c ??? kenapa bukan seperti nama property nya, \$jenis, \$merek, \$stok ???

Ini sering kali ditanyakan dan membuat bingung bila tidak paham. Nama parameter pada fungsi construct **TIDAK HARUS SAMA** dengan nama property dari class, yang penting jumlah nya sama.

TETAPI biasanya para programmer menuliskan nama parameter yang sama dengan nama property dari object yang akan dibentuk, **jadi boleh menggunakan bentuk seperti dibawah??**

```
__construct($jenis, $merek, $stok)
```

Boleh, malah disarankan seperti ini. Ok paham ya???

Dan berikut adalah cara pengiriman nilai argumennya, sesuaikan dengan tipe data apa yang dikirim, bila string diapit dengan tanda kutip dua, bila angka maka tidak dikutip.

```
$produk01 = new Produk("Televisi","Samsung",20);
```

Pertanyaan selanjutnya, klo misal argument yang dikirim jumlah tidak sesuai bagaimana??? Seperti contoh dibawah

```
$produk01 = new Produk("Televisi","Samsung");
```

Argument stok tidak dikirim saat pembuatan object, hal ini akan menimbulkan **ERROR**. **Kenapa??** Karena fungsi construct yang kita buat memiliki 3 buah parameter.

Apakah ada solusi???? Tentu ada

Seperti dibawah

```
__construct($jenis="televisi", $merek="samsung", $stok = 10)
```

Yup dengan memberi nilai default pada parameter fungsi. Dengan bentuk diatas apabila kita membuat object dengan tanpa mengirim variable maka otomatis nilai property akan diisi dengan nilai default yang sudah ditentukan... **PAHAM BANGET YA???**

LATIHAN LAGI ☺

Ubahlah latihan sebelumnya dengan mengimplementasikan fungsi construct. Simpan dengan nama file yang berbeda juga dengan sebelumnya.

Buat class PersegiPanjang, Lingkaran dan segitiga simpan dalam 3 file berbeda. Masing masing class memiliki fungsi untuk menghitung Luas dan Keliling. Lalu buat satu object dari masing-masing class untuk mengimplementasikan fungsi nya.

Ok sampai disini dulu materi pertemuan 5, selamat menikmati. Apabila ada yang kurang jelas bisa didiskusikan kemudian.