

Andre Pratama



OOP PHP Uncover

Panduan Belajar Pemrograman Object PHP

DuniaIlkom

OOP PHP Uncover

Panduan Belajar Pemrograman Object PHP

Andre Pratama

Buku ini ditulis dan diterbitkan secara mandiri oleh DuniaIlkom (www.duniailkom.com)

09 April 2022

~ Update Log ~

- ✓ Release OOP PHP Uncover 1.0 - Januari 2019
- ✓ Update OOP PHP Uncover 2.0 - Januari 2021
- ✓ Mini update OOP PHP Uncover 2.1 - Januari 2022
- ✓ Mini update OOP PHP Uncover 2.2 - April 2022

Cover Photo by [rawpixel.com](#) on [Pexels](#)

© 2022 DuniaIlkom

Daftar Isi

Ucapan Terima kasih.....	11
Tentang Penulis.....	12
Lisensi.....	13
Kata Pengantar.....	16
Asumsi / Pengetahuan Dasar.....	17
Contoh Kode Program.....	18
1. Berkenalan dengan Object Oriented Programming.....	19
1.1. Pengertian Pemrograman Berorientasi Objek.....	19
1.2. Kenapa Harus Pemrograman Berorientasi Objek?.....	20
1.3. Kekurangan OOP.....	21
1.4. Jadi, Pilih OOP atau Prosedural?.....	23
1.5. Pemrograman Object di PHP.....	23
2. Basic OOP PHP.....	25
2.1. Text Editor, XAMPP dan Web Browser.....	25
2.2. Class dan Object.....	26
2.3. Property dan Method.....	29
Cara Mengakses Property dan Method.....	30
2.4. Pseudo-variable \$this.....	32
2.5. Argument Method.....	38
2.6. Constructor dan Destructor.....	44
Deprecated Constructor.....	49
2.7. Inheritance.....	50
Method Overriding.....	55
Property Overriding.....	57
Constructor Overriding.....	59
Destructor Overriding.....	67
Final Keyword.....	69
2.8. Visibility.....	70
Visibility Public.....	71

Visibility Private.....	73
Visibility Protected.....	74
2.9. Getter dan Setter.....	76
Setter, Getter dan Constructor.....	81
2.10. Static Property dan Static Method.....	86
Static Property.....	87
Static Method.....	88
Static Increment.....	91
Helper Class.....	93
2.11. Class Constant.....	95
2.12. Constructor Property Promotion.....	97
3. Advanced OOP PHP.....	101
3.1. Abstract Class.....	101
Penulisan Abstract Class.....	101
Abstract Method.....	102
Polymorfism.....	109
3.2. Interface.....	112
Interface Inheritance.....	117
Interface Constant.....	118
3.3. Trait.....	119
Urutan Prioritas Method.....	122
Conflict Resolution.....	124
Trait di dalam Trait.....	126
Trait Property.....	127
Trait Static Property dan Static Method.....	128
Trait Abstract Method.....	128
Trait Access Modifier.....	129
Trait Name Collision.....	130
3.4. Magic Constant.....	130
3.5. Magic Method.....	133
Magic Method __toString().....	133
Magic Method __get().....	134
Magic Method __set().....	140
Magic Method __call().....	145
Magic Method __callStatic().....	148
Magic Method __isset().....	149

Magic Method __unset()	153
3.6. Perbandingan Object	156
3.7. Object Clone	161
Magic Method __clone()	162
Shallow Copy dan Deep Copy	163
3.8. Method Chaining	170
3.9. Type Hinting	174
Array Type Hinting	174
Object Type Hinting	176
Scalar Type Hinting	184
3.10. Typed Properties	186
Typed Properties untuk Object	188
Null Typed Properties	189
Union Typed Properties	190
3.11. Nullsafe Operator	190
3.12. Late Static Binding	193
3.13. Anonymous Class	197
3.14. stdClass	200
4. Class dan Object Function	203
4.1. Function is_object()	204
4.2. Function is_a()	204
4.3. Function is_subclass_of()	206
4.4. Function get_class_*(*)	207
4.5. Function get_parent_class()	209
4.6. Function get_declared_*	209
4.7. Function *_exists()	211
Fungsi class_exists()	212
Fungsi interface_exists()	213
Fungsi trait_exists()	214
Fungsi property_exists()	214
Fungsi method_exists()	215
4.8. Function class_alias()	216
4.9. Function get_called_class()	217
5. Namespace	218
5.1. Pengertian Namespace	218

5.2. Cara Penulisan Namespace.....	220
5.3. Cara Penulisan Sub-namespace.....	223
5.4. Namespace Import dan Alias.....	224
5.5. Group use Declarations.....	226
5.6. Global dan Relative Namespace.....	227
5.7. Unqualified, Qualified dan Fully Qualified Namespace.....	231
5.8. Multiple Namespace.....	231
5.9. Namespace untuk Function dan Konstanta.....	233
5.10. Namespace Magic Constant.....	235
6. Autoloading.....	240
6.1. Pengertian Autoloading.....	240
6.2. Cara Penggunaan Autoloading.....	241
Mengenal Function __autoload().....	250
6.3. Multiple Autoloading.....	251
6.4. Autoloading dengan Namespace.....	254
7. Exception.....	261
7.1. Pengertian Exception.....	261
7.2. Struktur try - catch.....	265
7.3. Custom Exception.....	275
7.4. Function set_exception_handler().....	279
7.5. Exception Trace.....	281
7.6. Struktur try - catch - finally.....	287
8. DateTime Object.....	290
8.1. DateTime Class.....	290
8.2. DateTime::format() Method.....	293
8.3. DateTimeZone Class.....	295
8.4. DateTime::add() Method dan DateInterval Class.....	297
8.5. DateTime::modify().....	300
8.6. DateTime::diff().....	303
8.7. DateTimeImmutable Class.....	306
9. Mysqli Object.....	309
9.1. Mysqli Class.....	310
Penanganan Error mysqli Object.....	313
Menutup mysqli Object.....	317

Membuat Database Dengan mysqli Object.....	318
Memilih Database Dengan mysqli Object.....	324
Membuat Tabel Dengan mysqli Object.....	325
Mengisi Tabel Dengan mysqli Object.....	328
9.2. Mysqli_result Class.....	332
Menampilkan Tabel Dengan mysqli_result Object.....	335
Mengambil Data Tabel Dari mysqli_result Object.....	341
Menghapus Data Tabel.....	349
Multi Query.....	351
Validasi Data.....	353
9.3. Mysqli_stmt Class.....	359
9.4. Mysqli Transaction.....	373
10. PDO.....	377
10.1. Pengertian PDO.....	377
10.2. Mengaktifkan PDO Extension.....	378
10.3. Membuat PDO Object.....	380
10.4. Error handling PDO Object.....	382
10.5. Menjalankan Query dengan method PDO::exec().....	384
10.6. Error Handling Query.....	386
10.7. Pengaturan PDO dengan method PDO::setAttribute().....	390
10.8. Menjalankan Query dengan method PDO::query().....	392
10.9. Menampilkan hasil Query dengan PDOStatement::fetch().....	394
10.10. Menampilkan hasil Query dengan PDOStatement::fetchAll().....	399
10.11. Prepared Statement dengan PDO.....	407
10.12. Prepared Statement dengan Named Parameters.....	410
10.13. Multiple Execution Prepared Statement.....	411
10.14. Transaction Query dengan PDO.....	413
10.15. Proses Bind Manual untuk Prepared Statement.....	415
11. Case Study: Database Query Builder.....	422
11.1. MySQL Query Builder.....	423
11.2. Class DB.....	425
11.3. Class DB: Constructor.....	426
11.4. Class DB: Singleton Pattern.....	429
11.5. Class DB: Method runQuery.....	434
11.6. Class DB: Method getQuery.....	440

11.7. Class DB: Method get.....	441
11.8. Class DB: Method select.....	444
11.9. Class DB: Method orderBy.....	447
11.10. Class DB: Method get (condition).....	449
11.11. Class DB: Method getWhere.....	450
11.12. Class DB: Method getWhereOnce.....	452
11.13. Class DB: Method getLike.....	455
11.14. Class DB: Method check.....	457
11.15. Class DB: Method insert.....	459
11.16. Class DB: Method count.....	467
11.17. Class DB: Method update.....	469
11.18. Class DB: Method delete.....	477
12. Case Study: Validate Class.....	481
12.1. Mempersiapkan Form Tambah Barang.....	481
12.2. Class Input: Method get.....	484
12.3. Function filter_var().....	487
String Sanitizing.....	488
Int Sanitizing.....	488
Float Sanitizing.....	489
Email Sanitizing.....	490
URL Sanitizing.....	490
12.4. Class Input: Method runSanitize.....	491
12.5. Function validate().....	493
Menampilkan Pesan Error.....	494
Function check_required.....	496
Function check_min_char.....	497
Function check_numeric.....	499
Function Validate.....	500
Validate: required.....	505
Validate: min_char.....	506
Break Loop.....	508
Validate: max_char.....	509
Validate: numeric.....	511
Validate: min_value dan max_value.....	512
Validate + Sanitize.....	513
Repopulate.....	516

12.6. Validate Class.....	520
12.7. Validate Class: Method getError.....	524
12.8. Mempersiapkan Form Tambah User.....	526
12.9. Validasi Form Tambah User.....	528
Validate: matches.....	528
Validate: email.....	530
Validate: url.....	532
Validate: regexp.....	534
12.10. Validasi Form ke Database.....	536
12.11. Proses Input ke Database.....	538
12.12. Validasi Checkbox.....	540
Re-populate Checkbox.....	545
Validasi dengan Regular Expression.....	546
12.13. Validasi Radio.....	549
12.14. Validasi Select.....	551
Re-populate Select.....	553
Method Input::generateOption.....	554
13. Case Study: Ilkoom Stock Manager.....	558
13.1. Ilkoom Stock Manager.....	558
13.2. Ilkoom Stock Manager: CRUD barang.....	559
Template: header.php dan footer.php.....	559
Autoloading: init.php.....	562
Menampilkan Tabel Barang: tampil_barang.php.....	563
Menambah Data Barang: tambah_barang.php.....	567
Class Barang: Method validasi(), insert() dan getItem().....	571
Mengubah Data Barang: edit_barang.php.....	575
Class Barang: Method generate() dan update().....	579
Menghapus Data Barang: hapus_barang.php.....	581
Class Barang: Method delete().....	584
13.3. Ilkoom Stock Manager: Autentikasi User.....	585
Password Hashing.....	586
Generate Tabel user.....	593
Registrasi User: register_user.php.....	594
Class User: Method validasiInsert(), insert() dan getItem().....	598
Pemberitahuan Register: register_berhasil.php.....	602
Login User: login.php.....	603

Class User: Method validasiLogin() dan login().....	606
Proteksi Halaman.....	608
Logout User: logout.php.....	614
Class User: Method logout().....	614
Menampilkan Data User: profile.php.....	614
Class User: Method logout().....	619
Pemberitahuan Ubah Password: ubah_password_berhasil.php.....	622
13.4. Ilkoom Stock Manager: Done!.....	623
Halaman Welcome: index.php.....	624
Proses Generate Database: db_generate_tabel_barang_dan_user.php.....	627
Penutup OOP PHP Uncover.....	632
Daftar Pustaka.....	635

Ucapan Terima kasih

Dalam kesempatan ini saya ingin mengucapkan terima kasih kepada Allah SWT karena dengan karuniaNya saya masih diberi kesempatan dan kesehatan untuk bisa menulis buku kedelapan DuniaIlkom: **OOP PHP Uncover**.

Selanjutnya kepada keluarga yang terus memberi motivasi dan dukungan tiada henti untuk terus mengembangkan DuniaIlkom.

Terakhir kepada rekan-rekan pembaca dan pengunjung setia DuniaIlkom. Terutama bagi yang telah memberikan donasi untuk membeli buku saya sebelumnya. Karena dari *feedback* dan dukungan rekan-rekan lah saya bisa lanjut menulis buku OOP PHP ini. Terima kasih :)

Padang Panjang, 2022

Penulis

Andre Pratama

www.duniaIlkom.com

Tentang Penulis



Andre Pratama

Andre memiliki background S1 Ilmu Komputer dari Universitas Sumatera Utara. Karena kecintaan akan dunia programming, mulai merintis web DuniaIlkom sejak tahun 2012.

Harapannya, tutorial di web serta buku terbitan DuniaIlkom bisa menjadi salah satu media belajar programming terbaik di Indonesia.

Andre berdomisili di kota Padang Panjang, Sumatera Barat. Jika ada pertanyaan, saran, kritik yang membangun bisa menghubungi duniailkom@gmail.com atau WA ke 083180285808.

Lisensi

Terima kasih untuk tidak memperbanyak / mengedarkan / mencopy eBook ini

Menulis sebuah buku hingga ratusan halaman butuh waktu yang tidak sebentar. Belum lagi saya harus berjuang mempelajari referensi yang kebanyakan dalam bahasa inggris. Ini saya lakukan agar pembaca bisa mendapatkan materi yang detail, update, dan berkualitas.

Saya menyadari kekurangan sebuah ebook adalah mudah dicopy-paste dan disebarluaskan. Tapi dengan eBook, harga buku bisa ditekan. Selain tidak perlu mencetak, eBook DuniaIlkom ini bisa di dapat dengan mudah dan murah, termasuk bagi teman-teman di daerah yang ongkos kirimnya lumayan mahal (jika berbentuk buku fisik).

Atas dasar itulah saya mohon kerjasamanya dari rekan-rekan semua untuk **tidak memperbanyak, menggandakan, atau mengupload ulang buku ini di forum, situs maupun media lain dalam bentuk apapun.**

Saya juga berharap rekan-rekan tidak memposting materi apapun yang ada di dalam buku ini. Jika ingin sebagai bahan artikel untuk postingan blog/situs, silahkan ambil materi yang ada di website duniaIlkom (jangan yang dari buku).

Apabila rekan-rekan memperoleh buku ini **bukan** dari DuniaIlkom, saya mohon bantuan donasinya untuk membeli versi asli. Donasi pembelian buku ini adalah sumber mata pencarian saya untuk menafkahsi keluarga. Lisensi atau hak guna buku ini hanya untuk 1 orang, yakni yang telah membeli langsung ke duniaIlkom@gmail.com.

Dengan kualitas yang ditawarkan, harga buku ini cukup terjangkau. Buku ini saya buat dengan waktu yang tidak sebentar, hingga berbulan-bulan, kadang sampai tengah malam. Bantuan donasi dari rekan-rekan yang membeli buku secara resmi sangat saya hargai, selain mendapat ilmu yang berkah, ini juga bisa menjadi penyemangat saya untuk terus berkarya dan menghadirkan ebook-ebook programming berkualitas lainnya.

Untuk yang membeli dari DuniaIlkom, saya ucapan banyak terimakasih :)

Anda diperbolehkan untuk:

- ✓ Mencetak eBook ini untuk keperluan pribadi dan dibaca sendiri.
- ✓ Mencopy eBook ini ke laptop/smartphone/laptop milik sendiri.
- ✓ Membuat ringkasan buku untuk digunakan sebagai bahan ajar (bukan keseluruhan isi buku).

Anda tidak dibolehkan untuk:

- ✗ Mencetak eBook ini untuk dibaca oleh orang lain, walaupun gratis.
- ✗ Mencopy eBook ini untuk dijual ulang, maupun dibagikan kepada orang lain dengan gratis.
- ✗ Membeli buku ini untuk dibaca bersama-sama (lisensi buku ini hanya untuk 1 orang).
- ✗ Mengambil sebagian atau seluruh isi buku untuk di publish ke blog, situs, artikel, dan media lain dalam bentuk apapun.
- ✗ Membagikan eBook ini kepada murid/siswa/mahasiswa (jika digunakan untuk bahan pengajaran).

Setiap pelanggaran dari lisensi ini akan dituntut sesuai undang-undang yang berlaku di Republik Indonesia, terutama **Pasal 12 UU No. 19 Tahun 2002** tentang **Hak Cipta**.

Penjelasan lebih lanjut bisa ke: [Apakah Mengunduh E-book Termasuk Perbuatan Illegal?](#)
Khusus untuk pembaca muslim bisa ke: [Hukum Memakai Barang Bajakan](#). Mari kita jaga agar ilmu yang di dapat berkah dan bermanfaat, bukan dari sumber yang haram.

REPUBLIK INDONESIA
KEMENTERIAN HUKUM DAN HAK ASASI MANUSIA

SURAT PENCATATAN CIPTAAN

Dalam rangka pelindungan ciptaan di bidang ilmu pengetahuan, seni dan sastra berdasarkan Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta, dengan ini menerangkan:

Nomor dan tanggal permohonan :

Pencipta

Nama : Andre Pratama

Alamat :

Kewarganegaraan :

Indonesia

Pemegang Hak Cipta

Nama : Andre Pratama

Alamat :

Kewarganegaraan :

Indonesia

Jenis Ciptaan :

e-Book

Judul Ciptaan :

OOP PHP Uncover

Tanggal dan tempat diumumkan untuk pertama kali di wilayah Indonesia atau di luar wilayah Indonesia

: 24 Januari 2019, di Padang Panjang

Jangka waktu pelindungan

: Berlaku selama hidup Pencipta dan terus berlangsung selama 70 (tujuh puluh) tahun setelah Pencipta meninggal dunia, terhitung mulai tanggal 1 Januari tahun berikutnya.

Nomor pencatatan :

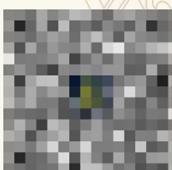
adalah benar berdasarkan keterangan yang diberikan oleh Pemohon.

Surat Pencatatan Hak Cipta atau produk Hak terkait ini sesuai dengan Pasal 72 Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta.

a.n. MENTERI HUKUM DAN HAK ASASI MANUSIA
DIREKTUR JENDERAL KEKAYAAN INTELEKTUAL



Dr. Freddy Harris, S.H., LL.M., ACCS.
NIP. 196611181994031001



Kata Pengantar

Object oriented programming, disingkat sebagai **OOP** merupakan sebuah "jargon" atau istilah yang tidak asing di dunia bahasa pemrograman. Istilah ini hadir sejak tahun 1970an dengan bahasa [Simula](#) dan [Smalltalk](#), yang kemudian booming dengan bahasa JAVA di tahun 1995.

Salah satu keunggulan dari OOP adalah mudah dipakai untuk memecahkan masalah yang kompleks serta menyediakan kerangka kerja yang fleksibel untuk pengembangan. Ini pula yang menjadi alasan banyak bahasa pemrograman lain untuk mengimplementasikan konsep OOP, termasuk bahasa PHP.

Pada awalnya, PHP adalah bahasa *pemrograman procedural*, dimana kita membuat kode program secara berurutan dari atas ke bawah dan bisa dipecah menjadi function. Sejak PHP 4, fitur OOP mulai hadir dan disempurnakan di PHP 5, PHP 7 dan PHP 8.

Kompleksitas web modern yang semakin rumit, menjadi pendorong implementasi OOP di PHP. Selain itu tersedia juga berbagai framework PHP seperti Code Igniter, Laravel, Symfony, atau Zend yang semuanya berbasis kepada pemrograman object. Jika butuh kode "bantu" seperti library, mayoritas juga berbentuk object.

Oleh karena itu, pengetahuan cara kerja OOP di PHP menjadi mutlak harus dikuasai. Apalagi jika anda berniat untuk berkarir sebagai programmer web back-end, dimana mayoritas lowongan kerja mensyaratkan harus paham salah satu framework PHP. Namun karena framework ini dibuat dengan konsep pemrograman object, maka untuk menguasainya harus memahami terlebih dahulu tentang OOP PHP.

Dalam buku **OOP PHP Uncover** ini kita akan membahas banyak hal tentang konsep dasar OOP, mulai dari pengertian *class*, cara pembuatan *object*, *property*, *method*, pewarisan (*inheritance*), *encapsulation*, *polimorfisme*, *abstract class*, *interface* dan *trait*.

Setelah itu akan dijelaskan juga tentang fitur yang berhubungan dengan OOP seperti *namespace*, *autoloading* dan *exception*. Kemudian kita akan membahas object bawaan PHP seperti *DateTime object*, *Mysqli object* serta *PDO* yang akan meningkatkan skill PHP anda ke level intermediate.

Di akhir buku terdapat studi kasus sebagai sarana latihan dan implementasi dari semua materi yang telah dipelajari.

Akhir kata, semoga buku **OOP PHP Uncover** ini bisa bermanfaat dan menjadi panduan terbaik untuk menguasai object oriented programming di PHP. Sampai jumpa di bab terakhir :)

Asumsi / Pengetahuan Dasar

Buku **OOP PHP Uncover** tidak saya tuju untuk pemula. Buku ini lebih ke level *intermediate* dimana anda harus memiliki pengetahuan seputar HTML dan PHP dasar terlebih dahulu.

Di buku ini saya berasumsi anda sudah paham tentang konsep pemrograman dasar PHP seperti variabel, tipe data, perulangan, if else dan function. Atau bisa juga disebut bahwa buku ini adalah lanjutan dari buku **PHP Uncover** DuniaIlkom.

Pemahaman tentang MySQL juga diperlukan terutama untuk bab tentang Mysqli object dan PDO. Setidaknya anda bisa memahami cara menggunakan query seperti SELECT, INSERT, UPDATE dan DELETE.

Khusus untuk studi kasus di bab terakhir yang membahas **Ilkoom Stock Manager**, saya menggunakan Bootstrap untuk membuat design tampilan. Materi tentang Bootstrap ini memang tidak berhubungan langsung dengan kode PHP, hanya sekedar mempercantik tampilan aplikasi kita.

Contoh Kode Program

Seluruh kode program yang ada dalam buku **OOP PHP Uncover** ini bisa di download dari folder sharing Google Drive yang di kirim pada saat pembelian: **belajar_oop_php.zip**. Isinya berupa file PHP yang disusun sesuai dengan bab tempat kode tersebut di bahas. Untuk menjalankan kode tersebut, ditempatkan ke dalam folder **htdocs** XAMPP.

Penomoran baris (*line numbering*) pada contoh kode program dalam buku ini berguna untuk memudahkan pembahasan. Jika anda ingin men-copy kode ini langsung dari eBook **pdf**, gunakan kombinasi tombol **ALT + tahan tombol mouse ketika proses seleksi** agar *line numbering* tidak ikut di-copy paste.

Alternatif yang lebih saya sarankan adalah dengan mengetik ulang seluruh kode program. Tujuannya agar anda lebih cepat paham sekaligus bisa menghafal posisi dari setiap kode.

Jika setelah diketik ternyata tidak jalan, besar kemungkinan ada penulisan yang salah. Solusinya, samakan kode yang anda tulis dengan file yang tersedia di **belajar_oop_php.zip**.

1. Berkenalan dengan Object Oriented Programming

Mengawali pembahasan di buku OOP PHP Uncover ini kita akan mulai dengan membahas pengertian pemrograman berorientasi objek, atau dalam bahasa inggris disebut sebagai *object oriented programming* (disingkat sebagai **OOP**).

1.1. Pengertian Pemrograman Berorientasi Objek

Secara sederhana, **pemrograman berorientasi objek** adalah sebuah cara penulisan kode program dengan menggunakan **object** untuk memecahkan masalah. Object ini bisa dianggap sebagai bagian dari kode program utama yang bisa berfungsi secara mandiri.

Jika didefinisikan seperti ini, object berfungsi layaknya sebuah **function**. Seperti yang mungkin sudah anda pahami, function (atau fungsi) juga bagian dari kode program utama dan dipakai untuk memecahkan masalah tertentu.

Tapi object jauh lebih powerful daripada function. Sebuah object nantinya bisa diturunkan kepada object lain, bisa di-setting hak aksesnya, memiliki *property* dan *method*, serta berbagai fitur lain.

Dalam teori pemrograman, terdapat 3 prinsip dasar yang melandasi pemrograman berorientasi objek, yakni *encapsulation*, *inheritance* dan *polymorphism*. Saya belum akan membahas pengertian dari istilah ini, tapi ketiganya dipakai untuk memecahkan masalah yang terlalu kompleks jika menggunakan function.

Agar lebih formal, berikut pengertian *pemrograman berorientasi objek* yang saya kutip dari [wikipedia](#):

"Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods"

Terjemahan bebas:

"Pemrograman berorientasi 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*).

Jika ini adalah pertama kali anda belajar pemrograman berorientasi objek, pengertian di atas akan membuat bingung. Namun tidak masalah, untuk saat ini cukup dipahami bahwa pemrograman berorientasi objek adalah sebuah cara pembuatan kode program yang menggunakan versi "upgrade" dari function, yakni **object**.

1.2. Kenapa Harus Pemrograman Berorientasi Objek?

Untuk bisa menjawab pertanyaan ini, kita harus cari pembandingnya. Selain pemrograman berorientasi objek, 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 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 pengelompokan 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.

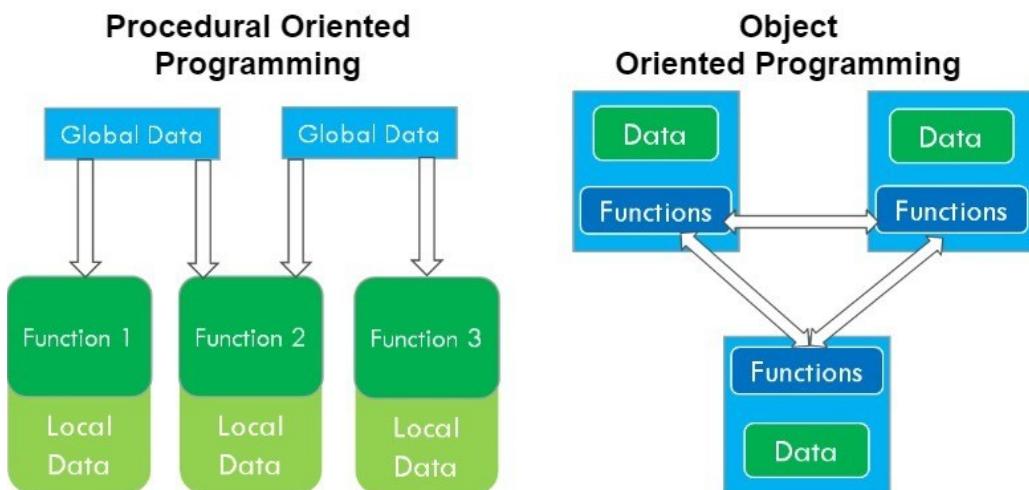
Masalah inilah yang bisa diatasi dengan lebih mudah jika menggunakan pemrograman berorientasi objek atau OOP.

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.

Konsep modular ini juga menjadi nilai tambah untuk tim. Setiap programmer bisa diberi tugas dengan pemisahan yang jelas. Bisa saja terdapat function yang sama di dalam object (dikenal dengan sebutan method), namun karena objectnya sudah berbeda, ini tidak menjadi masalah.

Jika dibuat dengan baik, kode program yang menggunakan OOP akan lebih mudah dipelajari dan dikembangkan.



Visualisasi perbedaan pemrograman prosedural dengan berorientasi object

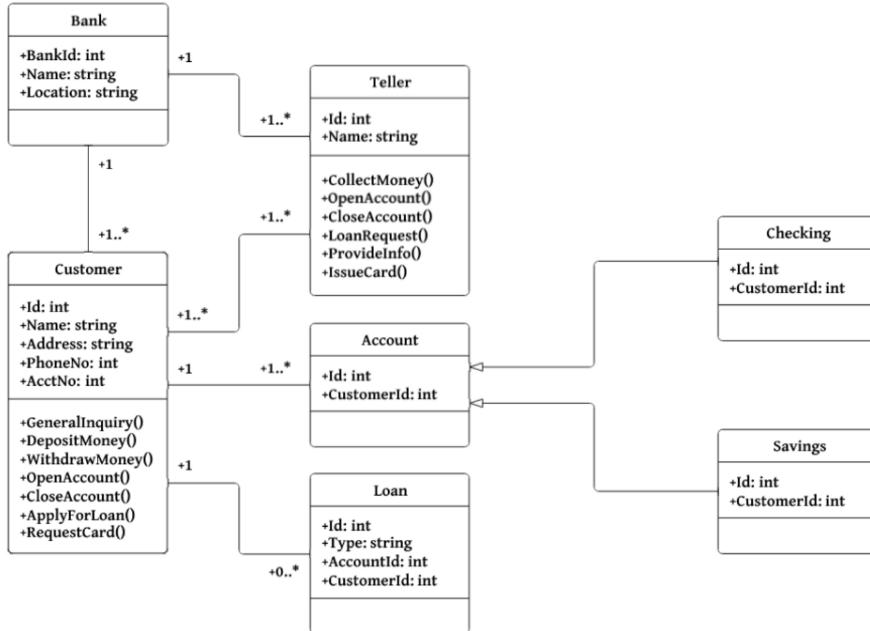
1.3. Kekurangan OOP

Dibalik semua keunggulannya, pemrograman berorientasi objek juga memiliki kelemahan.

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.

Ini sedikit berbeda dengan pemrograman prosedural, dimana kita bisa langsung mulai menulis kode program dan mencari solusi ketika bertemu sebuah masalah nanti.



Gambar: Contoh tampilan class diagram (sumber: medium.com)

Kelemahan **kedua** adalah perubahan pola pikir (mindset) agar bisa membuat kode yang benar-benar menerapkan prinsip pemrograman object, dan ini tidak mudah.

Dalam banyak kasus, kita lebih sering menggabung konsep pemrograman prosedural dengan pemrograman object. Pemrograman object dipakai untuk mengakses library atau object bawaan PHP (yang berisi fungsi-fungsi tertentu), namun program utama tetap dibuat dalam bentuk prosedural.

Sebagai contoh, di akhir buku **PHP Uncover** terdapat sebuah studi kasus aplikasi CRUD sederhana. Di sana saya menggunakan **mysqli extension** sebagai "jembatan" untuk mengakses database MySQL. Mysqli extension terdiri dari kumpulan function bawaan PHP yang menjadi ciri khas pemrograman prosedural.

Namun PHP juga menyediakan cara lain untuk mengakses database MySQL, yakni menggunakan mysqli extension versi object atau memakai **PDO** (PHP Data Objects). Jika saya mengkonversi seluruh function mysqli menjadi PDO, artinya sudah memakai object di dalam kode program.

Tapi kode program itu belum menerapkan prinsip OOP. Program utama tetap diproses secara berurutan dari atas ke bawah. Yang berubah hanyalah satu atau dua function dari mysqli menjadi PDO.

Jika ingin menerapkan prinsip OOP yang baku, sisa kode lain juga harus di rombak ulang misalnya menggunakan arsitektur **MVC** (Model-View-Controller). Secara sederhana, MVC adalah pemisahan kode antara "programming", "database", dan "design tampilan".

Di dalam MVC, seluruh kode yang berhubungan dengan database harus dikonversi menjadi

object, terpisah dengan kode program untuk memproses form, serta juga terpisah dengan kode HTML + CSS untuk membuat tampilan.

Inilah yang dimaksud bahwa tidak mudah menerapkan pemrograman yang "benar-benar OOP". Untuk itu perlu pengalaman dan skill khusus untuk menetapkan apa saja yang nantinya bisa di konversi menjadi sebuah object dan apa saja yang harus dipecah.

Sebagai jalan pintas, tersedia framework PHP seperti Code Igniter atau Laravel yang "memaksa" kita memisahkan bagian-bagian program ini. Saya sebut memaksa karena secara bawaan **Code Igniter** dan **Laravel** (serta berbagai framework lain) sudah menggunakan konsep seperti MVC sebagai penerapan dari OOP.

Meskipun tidak ideal, saya sering temukan programmer pemula yang langsung belajar Code Igniter tanpa paham apa itu pemrograman berorientasi objek yang sesungguhnya.

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.

1.4. Jadi, Pilih OOP atau Prosedural?

Menurut saya, kedua cara penulisan program ini punya peruntukannya masing-masing. Tidak serta merta OOP akan lebih baik dari pada pemrograman prosedural. Jika masalah yang akan dibuat sederhana dan kemungkinan besar tidak akan ada revisi (seperti untuk tugas dari dosen kampus), akan lebih praktis ditulis dalam pemrograman prosedural saja. Membuat kode program dengan prinsip OOP butuh persiapan dan pengetahuan tambahan.

Dan dalam banyak kasus, kita akan lebih sering menggunakan pemrograman prosedural "rasa" object, yakni kode program utama tetap dibuat prosedural, namun untuk beberapa bagian menggunakan library atau object tambahan.

Bagi pendukung OOP murni, pencampuran antara OOP dengan prosedural ini dianggap kurang tepat, tapi selama hasil akhirnya sesuai dengan keinginan menurut saya masih bisa dimaklumi. Kecuali anda memang sudah punya skill dan kemampuan, serta project yang dikerjakan butuh fleksibilitas yang tinggi, maka pemrograman berorientasi objek adalah pilihan yang terbaik.

1.5. Pemrograman Object di PHP

PHP termasuk ke dalam bahasa pemrograman "multi paradigma", yakni mendukung pemrograman prosedural dan juga pemrograman object.

Kita bisa menulis sebuah aplikasi web lengkap dengan PHP secara prosedural. Dan inilah yang dipakai sejak PHP 1 sampai PHP 3. Pemrograman object di PHP baru diperkenalkan di versi 4

(walaupun belum sempurna), dan menjadi fokus pengembangan di PHP 5 dan PHP 7.

Tuntutan perlunya memahami pemrograman object di PHP datang karena aplikasi web yang dibuat semakin kompleks. Kemudian, untuk bisa mempelajari framework PHP seperti **Code Igniter** dan **Laravel**, juga butuh skill pemrograman object di PHP.

Jika anda ingin berkariere sebagai programmer PHP (*back-end programmer*), paham tentang konsep pemrograman object menjadi sebuah keharusan, terlebih rata-rata lowongan kerja PHP mensyaratkan paham framework PHP yang secara tidak langsung berarti harus menguasai OOP PHP.

Selain PHP, bahasa pemrograman lain yang juga bisa ditulis secara prosedural dan object (*multi paradigm*) adalah **Python**, **C++** dan **JavaScript**. Sedangkan bahasa pemrograman yang secara khusus hanya bisa ditulis dalam bentuk pemrograman berorientasi objek adalah **Java** dan **C#**.

Konsep pemrograman object di mayoritas bahasa pemrograman juga akan sama. Jika anda sudah paham pemrograman object di Java, tidak akan kesulitan untuk memahami pemrograman object di PHP, meskipun ada beberapa perbedaan dari penerapan konsep pemrograman object antara PHP dengan bahasa Java.

Dalam bab ini kita telah membahas sekilas tentang apa itu pemrograman berorientasi objek, melihat kelemahan dan kelebihannya, serta perlunya memahami object oriented programming ini.

Berikutnya, kita akan langsung "ngoding" dan mempelajari cara pembuatan object di PHP.

2. Basic OOP PHP

Pembahasan mengenai materi dasar pemrograman object di PHP akan saya bagi ke dalam 2 bagian besar, yakni **Basic OOP PHP** yang akan kita bahas dalam bab ini, serta **Advanced OOP PHP** di bab berikutnya.

Sesuai dengan namanya, Basic OOP PHP akan pemrograman object dasar yang akan sering kita temui, diantaranya tentang pembuatan class, hak akses, konsep pewarisan (*inheritance*), serta constructor dan destructor.

Di advanced PHP nanti akan dibahas tentang konsep pemrograman object yang relatif lebih rumit, seperti *abstraction*, *interface*, *trait*, dan *object cloning*.

2.1. Text Editor, XAMPP dan Web Browser

Tidak ada hal khusus yang diperlukan untuk bisa mengikuti materi dalam buku ini. Karena ini adalah buku PHP lanjutan, saya berkesimpulan anda sudah memiliki text editor favorit. Dalam buku ini saya akan menggunakan **VS Code**, dan tidak masalah jika anda menggunakan teks editor lain seperti Notepad++, Sublime Text, Atom, atau Komodo Edit. Selama bisa membuat file teks dengan akhiran .php, itu tidak masalah.

Sebagai web server, saya menggunakan **Apache** bawaan **XAMPP 8.1.1**, yang artinya menggunakan **PHP 8.1**. Meskipun sebagian besar materi dalam buku ini tetap bisa berjalan di PHP 5.6, saya sarankan untuk memakai PHP 7.0 ke atas.

Saya juga berasumsi anda sudah paham cara menjalankan kode PHP, yakni dengan menempatkannya di dalam folder xampp/htdocs/, kemudian menjalankan **Apache Web Server** dari **XAMPP Control Panel**.

Dalam buku ini saya membuat folder belajar_oop_php sebagai tempat dari seluruh kode program, yang kemudian di bagi berdasarkan nama bab. Sebagai contoh, seluruh file kode PHP pada pembahasan bab ini ada di xampp/htdocs/belajar_oop_php/bab_02/. Anda tidak harus mengikuti struktur seperti ini, selama file berada di folder htdocs, itu tidak masalah.

Mengenai web browser, juga tidak ada batasan. Kode program PHP tidak memiliki ketergantungan dengan web browser sehingga seharusnya akan tetap tampil di web browser Internet Explorer 6 sekalipun.

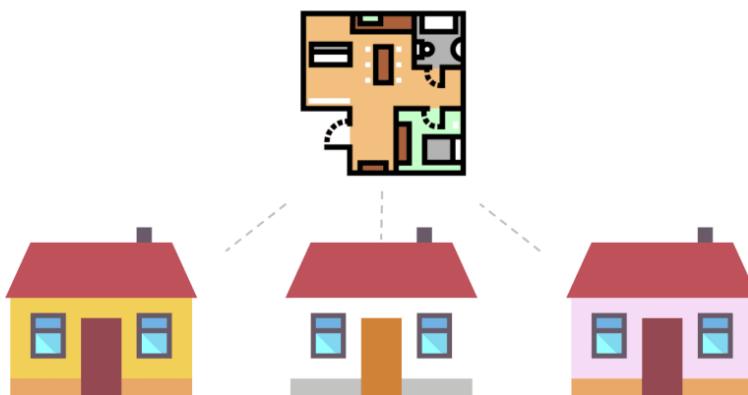
Akan tetapi karena nantinya saya akan menggunakan beberapa kode CSS, akan lebih baik memakai web browser modern seperti Google Chrome atau Mozilla Firefox. Sepanjang buku ini saya akan lebih banyak menggunakan Mozilla Firefox versi 62.

Baik, setelah semua perangkat tersedia, kita akan masuk ke kode program.

2.2. 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*.



Ilustrasi class design gambar rumah menjadi 3 object rumah (sumber: dev.to/charanrajgolla)

Perumpamaan rumah ini tampak agak aneh karena nyaris tidak berhubungan dengan programming. Akan tetap inilah inti dari OOP. Pemrograman berorientasi objek 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.

Namun tidak selamanya class dan object ini merujuk ke object asli. Object juga bisa berbentuk virtual seperti class **FileUpload**, atau class **FormValidation**.

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:

01.basic_class.php

```
1 <?php
2 class Produk {
3
4 }
```

Nama dari class boleh bebas selama mengikuti aturan *identifier* PHP, yakni tidak boleh diawali dengan angka, tidak boleh mengandung spasi dan tidak boleh berisi karakter "aneh" seperti %, # atau !. Aturan ini sebenarnya sama seperti penamaan *variabel* dan *function*.

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 *camel case*, seperti class ProdukTelevisi atau class UserAdmin.

Ini hanya tips dan kebiasaan saja. Jika pun kita ingin menulis nama class produk atau class produk_televisi, itu pun tetap bisa di proses oleh PHP.

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.

Untuk membuat object dari suatu class, gunakan keyword new seperti contoh berikut:

02.basic_object.php

```
1 <?php
2 class Produk {
3
4 }
5
6 $televisi = new Produk();
```

Baris 2 – 4 berisi kode program untuk membuat class Produk. Saat ini class Produk tidak berisi kode apapun, tapi itu tetap dianggap valid oleh PHP.

Di baris 6, saya membuat sebuah variabel bernama \$televisi. Variabel ini kemudian diisi dengan hasil operator new yang diikuti dengan nama Product(). Proses ini dikenal dengan istilah *instansiasi object* atau *object instantiation*.

Instansiasi object adalah proses pembuatan object dari sebuah class. Dalam contoh di atas, saya ingin membuat object dari class Produk yang kemudian disimpan dalam variabel \$televisi. Hasilnya, variabel \$televisi adalah sebuah **object dari class Produk**. Ini bisa kita periksa menggunakan function var_dump():

03.basic_object_var_dump.php

```
1 <?php
2 class Produk {
```

```

3
4 }
5
6 $televisi = new Produk();
7
8 var_dump($televisi); // object(Produk)#1 (0) { }

```

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).

Dari satu class `Produk` ini, kita bisa membuat beberapa object seperti contoh berikut:

04.basic_objects.php

```

1 <?php
2 class Produk {
3
4 }
5
6 $televisi = new Produk();
7 $mesinCuci = new Produk();
8 $speaker = new Produk();
9
10 var_dump($televisi); // object(Produk)#1 (0) { }
11 echo "<br>";
12 var_dump($mesinCuci); // object(Produk)#2 (0) { }
13 echo "<br>";
14 var_dump($speaker); // object(Produk)#3 (0) { }

```

Dalam kode program ini saya membuat 3 buah object dari class `Produk`, yakni `$televisi`, `$mesinCuci` dan `$speaker`.

Berikutnya, mari kita isi Class ini dengan **property** dan **method**.

Jika anda perhatikan, di setiap contoh kode program saya hanya menggunakan tanda pembuka "`<?php`", tapi tidak menutupnya di bagian akhir, yakni tidak terdapat tag penutup "`?>`". Penulisan seperti ini boleh dan disarankan untuk file `*.php` yang semuanya terdiri dari kode PHP (tidak ada kode HTML).

Alasannya adalah, untuk menghindari masalah karena sering tidak sengaja terdapat tambahan spasi setelah tag penutup "`?>`". Ini bisa membuat error ketika file di *include* ke dalam halaman lain (misalnya ketika dipakai di function yang melibatkan HTTP Header).

Oleh karena itu disarankan untuk tidak menulis tag penutup "`?>`" di baris terakhir. PHP secara otomatis akan menutupnya ketika tidak ada lagi kode yang akan diproses.

2.3. 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:

05.property_and_method.php

```

1 <?php
2 class Produk {
3     public $sku = "001";
4     public $merek = "Samsung";
5     public $harga = 4000000;
6
7     public function pesanProduk(){
8         return "Produk dipesan...";
9     }
10 }
11
12 $televisi = new Produk();

```

Di baris 3 – 5 saya membuat 3 buah property dengan nama `$sku`, `$merek` dan `$harga`. Pada awal penulisan ketiganya, terdapat keyword `public` yang disebut sebagai *access modifier* atau *visibility*.

Access modifier berfungsi untuk mengatur batasan hak akses dari sebuah property atau method. Access modifier `public` artinya semua property dan method bisa diakses dari mana saja, termasuk dari luar class.

Kita akan mempelajari *access modifier* dengan lebih detail pada bahasan tersendiri, untuk saat ini anggap saja semua property dan method harus memiliki awalan `public`, jika tidak PHP akan mengeluarkan pesan error.

Untuk ketiga property: `$sku`, `$merek` dan `$harga` masing-masing saya input dengan string `"001"`, `"Samsung"` dan integer `4000000`. Cara pemberian nilai ini sama seperti variabel biasa, yakni menggunakan operator *assignment* tanda sama dengan `" = "`.

Di baris 7 terdapat sebuah function bernama `pesanProduk()`. Karena function ini berada di dalam class, namanya berubah menjadi **method**. Sama seperti property, method juga diawali dengan *access modifier* yang saya set sebagai `public`. Isi dari method `pesanProduk()` berupa perintah `return "Produk dipesan..."`.

Selanjutnya di baris 12, class `Produk` saya inisialisasi ke dalam variabel `$televisi`.

Sebagai info tambahan, **sku** adalah kependekan dari stock keeping unit, yang tidak lain adalah nomor identitas untuk sebuah produk. SKU untuk produk, sama artinya seperti NIM (nomor induk mahasiswa) untuk mahasiswa, atau NIK (nomor induk kependudukan)

bagi warna negara Indonesia.

Sku merupakan istilah yang cukup umum dipakai dalam aplikasi manajemen stok. Tentu saja anda bisa menggunakan nama lain.

Cara Mengakses Property dan Method

Bagaimana cara mengakses property dan method? Ingat bahwa dalam pemrograman berorientasi objek, yang akan kita akses adalah object, bukan class.

Artinya untuk mengakses property dan method class Produk, di lakukan dari object yang dalam contoh di atas adalah variabel \$televisi:

06.property_and_method_access.php

```

1 <?php
2 class Produk {
3     public $sku = "001";
4     public $merek = "Samsung";
5     public $harga = 4000000;
6
7     public function pesanProduk(){
8         return "Produk dipesan...";
9     }
10 }
11
12 $televisi = new Produk();
13 echo $televisi->sku;           // 001
14 echo "<br>";
15 echo $televisi->merek;        // Samsung
16 echo "<br>";
17 echo $televisi->harga;         // 4000000
18 echo "<br>";
19 echo $televisi->pesanProduk(); // Produk dipesan...

```

Untuk mengakses property dan method yang ada di dalam sebuah object, kita menggunakan operator tanda panah " -> ", yakni gabungan dari tanda minus " - " dan tanda lebih besar " > ".

Setelah membuat object \$televisi di baris 12, saya mengakses isi property \$sku dengan perintah \$televisi->sku. Perhatikan bahwa kita tidak menulis tanda dollar untuk \$sku. Penulisan yang benar adalah \$televisi->sku, bukan \$televisi->\$sku.

Begitu juga dengan cara mengakses property lain, yakni \$televisi->merek dan \$televisi->harga. Perintah echo sendiri di pakai untuk menampilkan nilai dari setiap property.

Di baris 19, saya mengakses method pesanProduk(). Caranya sangat mirip seperti pengaksesan property, namun kali ini dengan tambahan tanda kurung di akhir, yakni \$televisi->pesanProduk().

Kemudian, bagaimana cara mengisi nilai ke dalam property? Berikut contohnya:

07.property_and_method_access_outside.php

```

1 <?php
2 class Produk {
3     public $sku = "001";
4     public $merek = "Samsung";
5     public $harga = 4000000;
6
7     public function pesanProduk(){
8         return "Produk dipesan...";
9     }
10}
11
12 $mesinCuci = new Produk();
13 $mesinCuci->sku = "002";
14 $mesinCuci->merek = "LG";
15 $mesinCuci->harga = 1500000;
16
17 echo $mesinCuci->sku;           // 002
18 echo "<br>";
19 echo $mesinCuci->merek;        // LG
20 echo "<br>";
21 echo $mesinCuci->harga;        // 1500000
22 echo "<br>";
23 echo $mesinCuci->pesanProduk(); // Produk dipesan...

```

Kali ini saya membuat object `$mesinCuci` yang berasal dari class `Product` di baris 12.

Di baris 13, 14 dan 15 saya mengakses ketiga property dari object `$mesinCuci` dan mengubah nilainya. Cara pemberian nilai ini kurang lebih sama seperti pemberian nilai ke dalam variabel PHP biasa, hanya saja sekarang menggunakan tanda panah "`->`".

Di baris 17 – 23 saya menampilkan kembali isi property yang sudah diubah serta mengakses method `pesanProduk()`. Terlihat bahwa isi ketiga property sudah ter-update dengan nilai baru.

Sebuah class bisa diinisialisasi untuk banyak object, seperti contoh berikut:

08.property_and_method_3_object.php

```

1 <?php
2 class Produk {
3     public $sku = "000";
4     public $merek = "";
5     public $harga = 0;
6
7     public function pesanProduk(){
8         return "Produk dipesan...";
9     }
10}
11
12 $televisi = new Produk();

```

```

13 $televisi->sku = "001";
14 $televisi->merek = "4000000";
15 $televisi->harga = 1500000;
16
17 $mesinCuci = new Produk();
18 $mesinCuci->sku = "002";
19 $mesinCuci->merek = "LG";
20 $mesinCuci->harga = 1500000;
21
22 $speaker = new Produk();
23 $speaker->sku = "003";
24 $speaker->merek = "Edifier ";
25 $speaker->harga = 950000;
26
27 print_r ($televisi);
28 echo "<br>";
29 print_r ($mesinCuci);
30 echo "<br>";
31 print_r ($speaker);

```

The screenshot shows a browser window with the URL `localhost/belajar_oop_php/bab_02/08.property_and_method_3`. The page content displays the output of the `print_r()` function for three objects:

```

Produk Object ( [sku] => 001 [merek] => 4000000 [harga] => 1500000 )
Produk Object ( [sku] => 002 [merek] => LG [harga] => 1500000 )
Produk Object ( [sku] => 003 [merek] => Edifier [harga] => 950000 )

```

Hasil perintah `print_r` untuk 3 object dari class `Product`

Terdapat sedikit perubahan untuk class `Produk`. Kali ini saya memberi nilai awal "000", string kosong "" dan angka 0 ke dalam property `$sku`, `$merek` dan `$harga`. Nilai "dummy" ini nantinya akan ditimpak pada saat pembuatan object.

Di baris 12 – 15, saya membuat object `$televisi` dari class `Produk()`, kemudian mengisi ulang ketiga property, hal yang sama juga dilakukan untuk object `$televisi` di baris 17 – 20, serta object `$speaker` di baris 22 – 25.

Terakhir, saya menggunakan function `print_r()` untuk menampilkan isi ketiga object. Function `print_r()` ini bisa dianggap sebagai versi sederhana dari `var_dump()`, yakni untuk menampilkan isi detail sebuah variabel (termasuk object). Function `print_r()` dan `var_dump()` sangat pas dipakai untuk proses pencarian kesalahan (*debugging*).

Hasilnya, ketiga object sudah berisi nilai yang berbeda meskipun semuanya berasal dari class yang sama, yakni `Produk`.

2.4. Pseudo-variable `$this`

Kali ini kita akan membahas sebuah konsep yang sering membuat bingung, yakni tentang pseudo-variable `$this`. Sebelum ke sana, silahkan pelajari sebentar kode program berikut ini:

09.this_problem.php

```

1 <?php
2 class Produk {
3     public $jenis = "";
4     public $merek = "";
5
6     public function pesanProdukTelevisi(){
7         return "Televisi dipesan...";
8     }
9
10    public function pesanProdukMesinCuci(){
11        return "Mesin cuci dipesan...";
12    }
13 }
14
15 $produk01 = new Produk();
16 $produk01->jenis = "Televisi";
17 $produk01->merek = "Samsung";
18
19 $produk02 = new Produk();
20 $produk02->jenis = "Mesin cuci";
21 $produk02->merek = "LG";
22
23 echo $produk01->pesanProdukTelevisi(); // Televisi dipesan...
24 echo "<br>";
25 echo $produk02->pesanProdukMesinCuci(); // Mesin cuci dipesan...

```

Saya mengubah kembali struktur class `Produk`. Sekarang class `Produk` memiliki property `jenis` dan `merek`, kemudian terdapat method `pesanProdukTelevisi()` dan `pesanProdukMesinCuci()`. Kedua method ini mengembalikan nilai string "Televisi dipesan..." atau "Mesin cuci dipesan...".

Setelah itu saya membuat 2 buah object dengan nama `$produk01` dan `$produk02`, serta mengisi property `jenis` dan `merek` ke dalam masing-masing object (baris 15 – 21).

Selanjutnya saya memesan `$produk01` dengan cara memanggil method `$produk01->pesanProdukTelevisi()` di baris 23, serta memesan `$produk01` dengan menjalankan method `$produk02->pesanProdukMesinCuci()` di baris 24.

Sekilas tidak ada yang salah dari kode di atas, tapi class `Produk` menjadi kurang fleksibel.

Bagaimana jika saya ingin membuat `$object03` dengan jenis speaker dan ingin memesan speaker ini? Maka saya harus modifikasi ulang class `Produk` untuk menambahkan method `pesanProdukSpeaker()`. Dan bagaimana jika nanti ada 100 produk? Kita terpaksa menulis satu per satu method untuk setiap `Produk`.

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:

10.this_problem_2.php

```

1  <?php
2  class Produk {
3      public $jenis = "";
4      public $merek = "";
5
6      public function pesanProduk(){
7          return $jenis." dipesan...";
8      }
9  }
10
11 $produk01 = new Produk();
12 $produk01->jenis = "Televisi";
13 $produk01->merek = "Samsung";
14
15 echo $produk01->pesanProduk();

```

Perhatikan isi dari method pesanProduk() di baris 7. Saya bermaksud menyambung isi property \$jenis milik class Produk, dengan string " dipesan".

Nantinya, property \$jenis ini akan berisi string "Televisi" di baris 12, sehingga saya mengharapkan hasil pemanggilan method \$produk01->pesanProduk() di baris 15 akan menjadi "Televisi dipesan...".

Namun ketika dijalankan hasilnya adalah sebagai berikut:

```
Warning: Undefined variable $jenis ... on line 7
```

Pesan error ini 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).

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:

11.this_solution.php

```
1 <?php
2 class Produk {
3     public $jenis = "";
4     public $merek = "";
5
6     public function pesanProduk(){
7         return $this->jenis." dipesan...";
8     }
9 }
10
11 $produk01 = new Produk();
12 $produk01->jenis = "Televisi";
13 $produk01->merek = "Samsung";
14
15 echo $produk01->pesanProduk(); // Televisi dipesan...
```

Sekarang, pada saat method \$produk01->pesanProduk() dijalankan, hasilnya adalah "Televisi dipesan..."

Lebih jauh lagi, saya bisa modifikasi sebagai berikut:

12.this_solution_2.php

```
1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5
6     public function pesanProduk(){
7         return $this->jenis." ".$this->merek." dipesan...";
8     }
9 }
10
11 $produk01 = new Produk();
12 $produk01->jenis = "Televisi";
13 $produk01->merek = "Samsung";
14
15 $produk02 = new Produk();
16 $produk02->jenis = "Mesin cuci";
17 $produk02->merek = "LG";
18
19 echo $produk01->pesanProduk(); // Televisi Samsung dipesan...
20 echo "<br>";
21 echo $produk02->pesanProduk(); // Mesin cuci LG dipesan...
```

Saya tidak lagi memberikan nilai awal untuk property \$jenis dan \$merek di baris 3 – 4. Ini tidak masalah karena nilainya nanti akan diisi setelah object dibuat.

Method pesanProduk() juga sedikit berubah, sekarang saya memadukan nilai dari \$this->jenis, dengan \$this->merek. Tanda titik di antara penulisan tersebut adalah operator penyambungan string (*string concatenation*).

Di baris 11 – 17 saya membuat dua buah object, yakni \$produk01 dan \$produk02, kemudian mengisi property \$jenis dan \$merek untuk masing-masing object.

Di baris 19 terdapat pemanggilan method pesanProduk() dari object \$produk01. Karena property \$jenis sudah diisi string "Televisi" dan property \$merek diisi string "Samsung", maka inilah yang menjadi nilai dari \$this->jenis dan \$this->merek.

Di baris 20 juga terdapat pemanggilan method pesanProduk(), tapi kali ini dari object \$produk02. Sekarang, nilai dari \$this->jenis dan \$this->merek akan menyesuaikan diri dengan object \$produk02 yang sebelumnya di set sebagai "Mesin cuci" dan "LG". Inilah maksud bahwa variabel \$this, merujuk ke object, bukan class.

Konsep tentang *pseudo-variable* \$this ini memang cukup membingungkan namun sangat penting. Kembali, perlu dipahami bahwa \$this adalah **sebuah variabel khusus yang merujuk kepada object pada saat kita menulis sesuatu di dalam class**.

Exercise

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.

Method cekStok() bisa dipakai untuk menampilkan sisa stok yang ada. Berikut potongan kode program saat pembuatan object:

```

1 <?php
2
3 // Definisi class Produk di sini
4
5 $produk01 = new Produk();
6 $produk01->jenis = "Televisi";
7 $produk01->merek = "Samsung";
8 $produk01->stok = 54;
9
10 echo $produk01->cekStok(); // Sisa stok: 54
11
12 $produk01->pesanProduk();
13 $produk01->pesanProduk();
14 $produk01->pesanProduk();

```

```

15
16 echo $produk01->cekStok(); // Sisa stok: 51

```

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.

Baik, berikut kode program yang saya rancang untuk soal di atas:

13.this_exercise.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function pesanProduk(){
8         $this->stok = $this->stok - 1;
9     }
10
11    public function cekStok(){
12        return "Sisa stok: ". $this->stok . "<br>";
13    }
14 }
15
16 $produk01 = new Produk();
17 $produk01->jenis = "Televisi";
18 $produk01->merek = "Samsung";
19 $produk01->stok = 54;
20
21 echo $produk01->cekStok(); // Sisa stok: 54
22
23 $produk01->pesanProduk();
24 $produk01->pesanProduk();
25 $produk01->pesanProduk();
26
27 echo $produk01->cekStok(); // Sisa stok: 51

```

Fokus utama dari pembuatan class Produk ada di method `pesanProduk()` di baris 7 – 9. Ketika method ini dipanggil, jalankan perintah `$this->stok = $this->stok - 1`. Artinya, ambil property `$stok` yang ada di object, lalu kurangi 1 angka dan simpan kembali ke dalam property `$stok`.

Pada saat object `$produk01` dibuat, saya memberikan nilai stok sebanyak 54 (baris 19), setelah itu terdapat 3 kali pemanggilan method `pesanProduk()` yang setiap pemanggilan akan mengurangi stok produk sebanyak 1 buah.

Untuk class yang cukup panjang seperti ini, akan lebih mudah jika dibaca bagian setelah

inisialisasi object terlebih dahulu (baris 16, 17, dst), kemudian setiap kali terdapat pemanggilan property atau method, baru pindah mempelajari class-nya.

2.5. 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.

Kita juga bisa menggunakan metode yang sama untuk method.

Melanjutkan latihan tentang method `pesanProduk()`, saya ingin membuat method `borongProduk()`. Jika pada `pesanProduk()` hanya bisa memesan 1 buah produk saja, pada method `borongProduk()` ini bisa membeli banyak produk sekaligus. Jumlah produk yang dibeli nantinya akan menjadi sebuah argument. Berikut kode programnya:

14.method_argument.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function pesanProduk(){
8         $this->stok = $this->stok - 1;
9     }
10
11    public function borongProduk($jumlah){
12        $this->stok = $this->stok - $jumlah;
13    }
14
15    public function cekStok(){
16        return "Sisa stok: ". $this->stok . "<br>";
17    }
18 }
19
20 $produk01 = new Produk();
21 $produk01->jenis = "Televisi";
22 $produk01->merek = "Samsung";
23 $produk01->stok = 54;
24
25 echo $produk01->cekStok(); // Sisa stok: 54

```

```

26
27 $produk01->borongProduk(10);
28 echo $produk01->cekStok(); // Sisa stok: 44
29
30 $produk01->borongProduk(25);
31 echo $produk01->cekStok(); // Sisa stok: 19

```

Perhatikan cara pendefinisian method `borongProduk()` di baris 11 – 13. Di sini saya membuat sebuah parameter `$jumlah`. Parameter `$jumlah` nantinya akan menjadi variabel penampung nilai input argument pada saat pemanggilan method `borongProduk()`.

Ketika method `borongProduk($jumlah)` dipanggil, kurangkan nilai `$this->stok` dengan parameter `$jumlah`, lalu simpan kembali ke dalam `$this->stok`.

Untuk parameter `$jumlah`, kita tidak perlu memakai *pseudo variable* `$this`, karena `$jumlah` ini bukanlah property dari class `Produk`, tapi hanya sebuah variabel yang ada di dalam method `borongProduk()` saja.

Setelah pembuatan object `$produk01` di baris 20, saya mengisi nilai property `$stok` di baris 23, yakni dengan perintah `$produk01->stok = 54`. Artinya stok `produk01` sebanyak 54 buah.

Kemudian terdapat pemanggilan method `$produk01->borongProduk(10)` di baris 27. Pemanggilan ini akan mengirim nilai **10** ke dalam parameter `$jumlah`, yang kemudian akan mengurangi nilai property `$stok` sebanyak 10 buah, hasilnya `$stok` tinggal 44.

Di baris 31, terdapat pemanggilan kedua dari method `borongProduk()`, kali ini dengan argument 25. Nilai ini juga akan dikirim ke parameter `$jumlah` yang akan mengurangi stok sebanyak 25 buah. Hasilnya, property `$stok` tinggal 19.

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),
...

```

Perbedaan argument dan parameter

Dalam buku ini saya akan menggunakan istilah *parameter* dan *argument* secara bergantian (sesuai dengan pengertiannya).

Kita juga bisa menggunakan fitur lanjutan function PHP seperti *default parameter* (atau di PHP Manual disebut sebagai *default argument*). Tujuannya, ketika method dipanggil tanpa argument, nilai default-lah yang akan dipakai.

Berikut modifikasi method `borongProduk()` dengan tambahan *default parameter*:

15.method_default_argument.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function borongProduk($jumlah = 10){
8         $this->stok = $this->stok - $jumlah;
9     }
10
11    public function cekStok(){
12        return "Sisa stok: ". $this->stok .<br>;
13    }
14 }
15
16 $produk01 = new Produk();
17 $produk01->jenis = "Televisi";
18 $produk01->merek = "Samsung";
19 $produk01->stok = 54;
20
21 $produk01->borongProduk();
22 echo $produk01->cekStok(); // Sisa stok: 44
23
24 $produk01->borongProduk(20);
25 echo $produk01->cekStok(); // Sisa stok: 24

```

Pada pendefinisian method `borongProduk()` di baris 7, saya menulisnya sebagai `borongProduk($jumlah = 10)`. Ini artinya jika method `borongProduk()` dipanggil tanpa argument, variabel `$jumlah` akan berisi nilai 10. Namun jika method `borongProduk()` dipanggil

dengan argument, nilai argument-lah yang akan dipakai.

Di baris 21, saya memanggil method `$produk01->borongProduk()`, karena ini tanpa argumen, maka stok dikurangi 10 sesuai dengan nilai default. Di baris 24 terdapat pemanggilan `$produk01->borongProduk(20)`, sekarang stok akan dikurangi 20 sesuai dengan nilai argument.

Exercise

Latihan ini adalah tentang pembuatan method dengan argument, tapi saya ingin sedikit lebih kompleks.

Kasusnya seperti ini, 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.

Berikut cara pemanggilan method `tambahstok()` ini:

```

1 <?php
2
3 // Definisi class Produk di sini
4
5 $produk01 = new Produk();
6 $produk01->jenis = "Televisi";
7 $produk01->merek = "Samsung";
8 $produk01->stok = 54;
9
10 echo $produk01->tambahstok();
11 echo "<br>";
12 echo $produk01->tambahstok(20);
13 echo "<br>";
14 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

```

Pertama saya mengisi object \$produk01 dengan nilai stok awal adalah 54 (baris 8).

Kemudian di baris 10 saya menjalankan \$produk01->tambahstok(), hasilnya stok bertambah sebanyak 12 buah menjadi 66. Perhatikan hasil yang tampil, selain menambah stok saat ini, method tambahstok() juga mengembalikan string, yakni:

```
Stok berhasil ditambah
Jumlah stok saat ini: 66
```

Di baris 12, saya kembali menjalankan \$produk01->tambahstok(20), sehingga stok akan bertambah lagi menjadi 86.

Terakhir di baris 14 saya memanggil \$produk01->tambahstok(15). Namun karena penambahan ini mengakibatkan total stok menjadi 101 (hasil dari 86 + 15), maka proses penambahan tidak bisa dilakukan. Hasil yang tampil adalah:

```
Maaf, stok sudah penuh. Penambahan stok dibatalkan
Jumlah stok saat ini: 86
```

Silahkan anda coba rancang seperti apa kira-kira cara pembuatan method tambahstok(). Yang jelas, di dalam method kita butuh pengecekan kondisi berapa total akhir \$stok setelah penambahan. Jika kurang dari 100, update \$stok, tetapi jika hasilnya lebih besar dari 100, batalkan proses penambahan. Di sini kita akan butuh sebuah kondisi **if else**.

Baik, berikut kode yang saya pakai untuk membuat class Produk dengan method tambahstok():

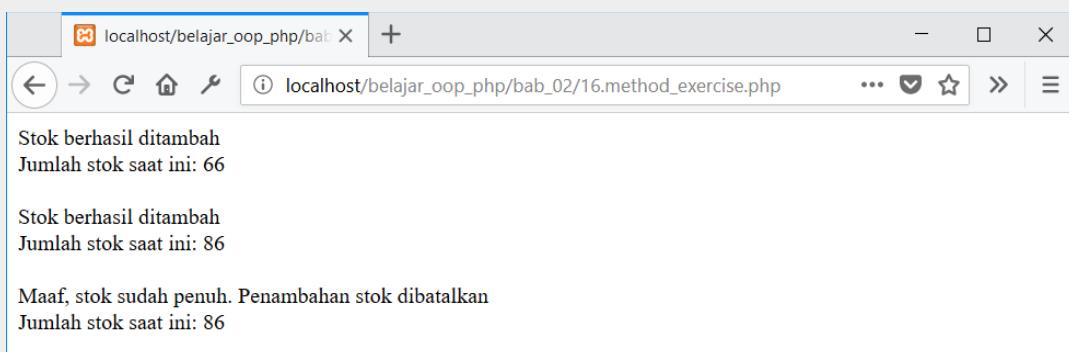
16.method_exercise.php

```
1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function tambahStok($jumlah = 12){
8
9         $totalStok = $this->stok + $jumlah;
10
11        if ($totalStok <=100){
12            $this->stok = $totalStok;
13            $pesan = "Stok berhasil ditambah <br>";
14            $pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";
15        }
16        else {
17            $pesan = "Maaf, stok sudah penuh. Penambahan stok dibatalkan <br>";
18            $pesan .= "Jumlah stok saat ini: ".$this->stok."<br>";
19        }
20        return $pesan;
21    }
22}
```

```

21 }
22 }
23
24 $produk01 = new Produk();
25 $produk01->jenis = "Televisi";
26 $produk01->merek = "Samsung";
27 $produk01->stok = 54;
28
29 echo $produk01->tambahStok();
30 echo "<br>";
31 echo $produk01->tambahStok(20);
32 echo "<br>";
33 echo $produk01->tambahStok(15);

```



Fokus utama kita ada di pendefinisian method `tambahstok()` di baris 7 – 21. Di sini saya menggunakan *default parameter* `$jumlah` dengan nilai 12. Artinya jika method `tambahstok()` dipanggil tanpa argument, parameter `$jumlah` akan berisi nilai 12.

Di baris 9, saya membuat sebuah variabel bantu yakni `$totalStok`. Isinya berupa hasil dari penambahan property `$stok` dengan parameter `$jumlah`.

Karena ada syarat total stok tidak boleh lebih dari 100, maka perlu sebuah kondisi **if else**. Jika ternyata hasil `$totalstok` kurang dari 100, update nilai property `$stok`, tapi jika hasil `$totalstok` lebih dari 100, maka jangan lakukan update.

Apabila ternyata `$totalstok` kurang dari 100, blok kode program di baris 12 – 14 akan diproses. Di baris 12 saya mengupdate property `$stok` dengan perintah `$this->stok = $totalstok`. Hasilnya, jumlah `$stok` akan bertambah sesuai nilai yang ada di dalam parameter `$jumlah`.

Kemudian saya ingin menampilkan pesan bahwa penambahan stok sukses di proses (baris 13 dan 14). Pesan ini di simpan ke dalam variabel `$pesan`. Isinya berupa string "stok berhasil ditambah
", yang kemudian disambung di baris kedua dengan pesan "Jumlah stok saat ini: ".\$this->stok."
". Untuk menyambung string `$pesan` ini digunakan operator gabungan assignment, yakni tanda titik sama dengan " .= ".

Selanjutnya bagian **else** hanya akan dijalankan jika kondisi `if ($totalstok <=100)` bernilai **false**, yang artinya jika total stok melebihi 100. Apabila ini yang terjadi, jangan

lakukan update, tapi cukup isi variabel \$pesan dengan keterangan bahwa stok sudah penuh.

Terakhir di baris 20 saya mengembalikan nilai variabel \$pesan dengan perintah `return $pesan`. Karena method `tambahStok()` mengembalikan nilai string, maka bisa langsung dijalankan dengan perintah `echo`.

2.6. 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 dihapus.

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, rancang sebuah method dengan nama `__construct()`, sedangkan untuk destructor, buat method dengan nama `__destruct()`.

Di dalam PHP, sebuah method yang diawali dengan tanda garis bawah 2 kali biasanya adalah method khusus (dikenal dengan istilah *magic method*). Selain `__construct()` dan `__destruct()`, nantinya kita akan lihat *magic method* lain bawaan PHP, diantaranya `__get()`, `__set()`, dan `__toString()`.

Karena alasan ini, tidak disarankan membuat method dengan awalan tanda garis bawah 2 kali, karena bisa bentrok dengan *magic method* bawaan PHP.

Langsung saja kita lihat cara membuat constructor di dalam PHP:

17. constructor_basic.php

```
1 <?php
2 class Produk {
3
4     public function __construct(){
5         echo "Constructor dijalankan... <br>";
6     }
7
8 }
```

```
9  
10 $produk01 = new Produk();  
11 $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.

Bagaimana dengan *destructor*? Cara penulisannya kurang lebih sama:

18.destructor_basic.php

```
1 <?php  
2 class Produk {  
3  
4     public function __construct(){  
5         echo "Constructor dijalankan... <br>";  
6     }  
7  
8     public function __destruct(){  
9         echo "Destructor dijalankan... <br>";  
10    }  
11  
12 }  
13  
14 $produk01 = new Produk();  
15 $produk02 = new Produk();
```

Hasil kode program:

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

Sekarang class Produk memiliki 2 buah method, yakni `__construct()` dan `__destruct()`. Keduanya otomatis dijalankan tanpa perlu memanggil method ini secara manual.

Secara default, sebuah object akan dihapus ketika halaman yang berisi kode PHP selesai diproses (setelah icon *loading* di web browser berhenti). Pada saat itulah method `__destruct()` di jalankan.

Namun kita juga bisa menghapus object secara manual, caranya gunakan function `unset()`, atau ganti isi variabel penampung object dengan nilai `null` seperti contoh berikut:

19.destructor_basic_2.php

```
1 <?php
2 class Produk {
3
4     public function __construct(){
5         echo "Constructor dijalankan... <br>";
6     }
7
8     public function __destruct(){
9         echo "Destructor dijalankan... <br>";
10    }
11
12 }
13
14 $produk01 = new Produk();
15 $produk01 = null;
16
17 echo "Program selesai <br>";
```

Hasil kode program:

```
Constructor dijalankan...
Destructor dijalankan...
Program selesai
```

Struktur dari class `Produk` di atas masih sama seperti sebelumnya, namun kali ini saya menghapus manual object `$produk01` dengan cara mengisi variabel ini dengan nilai `null` (baris 15). Hasilnya, destructor akan di jalankan terlebih dahulu sebelum perintah `echo` di baris 17.

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:

20.constructor_problem.php

```
1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6 }
7
8 $produk01 = new Produk();
9 $produk01->jenis = "Televisi";
10 $produk01->merek = "Samsung";
11 $produk01->stok = 20;
12
13 $produk02 = new Produk();
14 $produk02->jenis = "Mesin cuci";
15 $produk02->merek = "LG";
16 $produk02->stok = 10;
17
```

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

Hasil kode program:

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

Class Produk memiliki 3 property, yakni \$jenis, \$merek dan \$stok. Setelah object \$produk01 dan \$produk02 selesai dibuat, saya mengisi ketiga property ini dengan cara menginput langsung ke dalam property tersebut (baris 9 – 11 dan 14 – 16).

Cara di atas memang bisa dipakai, namun ada yang lebih baik. Dari pada menginput manual setiap property, kita bisa memanfaatkan constructor. Semua nilai-nilai ini bisa dikirim sebagai argumen ke dalam constructor. Yup, karena constructor tidak lain adalah sebuah method (atau function), maka bisa menerima argument seperti pembahasan tentang method sebelumnya.

Berikut modifikasi class Produk dengan penambahan constructor:

21.constructor_solution.php

```
1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($a, $b, $c){
8         $this->jenis = $a;
9         $this->merek = $b;
10        $this->stok = $c;
11    }
12 }
13
14 $produk01 = new Produk("Televisi","Samsung",20);
15 $produk02 = new Produk("Mesin cuci","LG", 10);
16
17 print_r ($produk01);
18 echo "<br>";
19 print_r ($produk02);
```

Hasil kode program:

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

Silahkan anda pelajari sejenak kode program di atas.

Di dalam class Produk, saya merancang constructor agar bisa menerima 3 buah argument. Setiap argument akan ditampung oleh 3 parameter: \$a, \$b dan \$c. Ketiga parameter ini

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`.

Pada saat pembuatan object `$produk01`, perintahnya menjadi `new Produk("Televisi", "Samsung", 20)`. Argument "Televisi" akan dikirim ke dalam parameter `$a`, argument "Samsung" ke parameter `$b`, dan argument 20 ke dalam parameter `$c`. Yang kemudian dikirim ulang ke setiap property.

Hasilnya, semua property untuk `$produk01` sudah langsung terisi pada saat object dibuat. Hal yang sama juga berlaku untuk `$produk02` namun dengan nilai argument yang berbeda.

Nama parameter sengaja saya buat dengan variabel `$a`, `$b` dan `$c` agar bisa dibedakan dengan nama property. Umumnya, nama parameter di tulis dengan nama yang sama seperti nama property, seperti contoh berikut:

22.constructor_solution_2.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok = 10){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 $produk01 = new Produk("Televisi", "Samsung", 20);
15 $produk02 = new Produk("Mesin cuci", "LG");
16
17 print_r ($produk01);
18 echo "<br>";
19 print_r ($produk02);

```

Hasil kode program:

```

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

```

Di sini saya mengganti nama parameter pada definisi constructor dari `$a`, `$b` dan `$c` ke `$jenis`, `$merek`, `$stok`. Khusus untuk parameter `$stok` memiliki tambahan berupa nilai default 10.

Jika ditulis seperti ini, harus dipahami bahwa variabel `$jenis` yang menjadi parameter di dalam constructor, tidak sama dengan property `$jenis` yang ada di dalam class (meskipun keduanya memiliki nama yang sama). Variabel `$jenis` yang ada di dalam parameter hanya dikenal di dalam constructor saja.

Jika anda ragu dengan penjelasan ini, boleh bandingkan dengan kode sebelumnya yang menggunakan parameter \$a, \$b dan \$c.

Depreciated Constructor

Bagian ini hanya untuk pengetahuan umum karena sudah berstatus *deprecated* (usang) dan sudah dihapus di PHP 8. Info ini mungkin berguna jika anda mendapati kode program PHP lama.

Dulunya di PHP 4 ke bawah, cara penulisan constructor adalah dengan membuat nama method yang sama dengan nama class. Misalnya method dengan nama `Produk()` akan berfungsi sebagai constructor di dalam class `Produk` seperti contoh berikut:

23.constructor_old.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function Produk($jenis, $merek, $stok = 10){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 $produk01 = new Produk("Televisi","Samsung",20);
15 $produk02 = new Produk("Mesin cuci","LG");
16
17 print_r ($produk01);
18 echo "<br>";
19 print_r ($produk02);

```

Hasil kode program (PHP 7):

```

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

```

Dalam PHP 8, kode di atas akan menampilkan hasil berikut:

```

Produk Object ( [jenis] => [merek] => [stok] => )
Produk Object ( [jenis] => [merek] => [stok] => )

```

Yang artinya sudah tidak bisa lagi dipakai di PHP 8.

2.7. Inheritance

Inheritance atau **penurunan** atau **pewarisan** adalah salah satu konsep utama dalam pemrograman berorientasi objek. Dengan *inheritance*, kita bisa "menurunkan" isi dari sebuah class ke dalam class lain. Isi class yang dimaksud adalah property dan method.

Konsep *inheritance* memungkinkan kita untuk membuat hierarki class atau struktur class yang berhubungan satu sama lain. Kita akan membahasnya secara bertahap, pertama perhatikan kode program di bawah ini:

24.tanpa_inheritance.php

```

1 <?php
2 class Produk {
3     public $merek = "Sharp";
4     public $stok = 50;
5
6     public function cekStok(){
7         return "Sisa stok: ".$this->stok;
8     }
9 }
10
11 $produk01 = new Produk();
12 echo $produk01->merek;           // Sharp
13 echo "<br>";
14 echo $produk01->cekStok();       // Sisa stok: 50

```

Tidak ada yang baru dalam kode program di atas. Saya membuat class **Produk** dengan property **\$merek** dan **\$stok** serta langsung mengisinya dengan nilai awal "Sharp" dan 50. Kemudian terdapat method **cekStok()** yang mengembalikan string berupa nilai stok saat ini.

Di baris 11 saya menginstansiasi class **Produk** ke dalam object **\$produk01**, diikuti dengan mengakses isi property **\$produk01->merek** di baris 12 serta method **\$produk01->cekStok()** di baris 14.

Kode di atas berjalan sebagaimana mestinya. Namun class **Produk** sejak awal saya rancang bukan sebagai class akhir. Class **Produk** hanya sebagai class "induk" yang nantinya diturunkan ke dalam class lain, seperti class **Televisi**, class **MesinCuci**, dll.

Untuk menurunkan sebuah class ke dalam class lain, gunakan keyword **extends** seperti format berikut:

```

class Televisi extends Produk {
    // ... isi class di sini
    // ... isi class di sini
}

```

Kode di atas artinya saya ingin menurunkan atau mewariskan class **Produk** ke dalam class **Televisi**. Class **Produk** berperan sebagai class *induk*, atau **parent class**. Sedangkan class

Televisi bertindak sebagai *class anak*, atau **child class**. Child class nantinya akan memiliki semua property dan method yang ada di dalam *parent class*.

Berikut kode program lengkap penurunan class Produk ke dalam class Televisi:

25.inheritance_basic_1.php

```

1 <?php
2 class Produk {
3     public $merek = "Sharp";
4     public $stok = 50;
5
6     public function cekStok(){
7         return "Sisa stok: ".$this->stok;
8     }
9 }
10
11 class Televisi extends Produk {
12 }
13
14 $produk01 = new Televisi();
15 echo $produk01->merek;           // Sharp
16 echo "<br>";
17 echo $produk01->cekStok();       // Sisa stok: 50

```

Isi class Produk masih sama seperti sebelumnya. Di baris 11 saya membuat class Televisi sebagai turunan dari class Produk. Isi dari class Televisi sendiri sengaja dikosongkan.

Selanjutnya, di baris 14 terdapat proses *instansiasi* class Televisi, sekarang variabel \$produk01 berisi object dari class Televisi, bukan lagi class Produk. Di baris 15 dan 17 saya mencoba mengakses property \$merek dan method cekStok() dari \$produk01.

Jika anda perhatikan, property \$merek dan method cekStok() adalah kepunyaan dari class Produk, bukan class Televisi. Namun karena class Televisi adalah turunan dari class Produk, property dan method milik Produk tetap bisa diakses dari dalam class Televisi.

Class Televisi sendiri juga bisa kita isi sebagaimana layaknya class biasa:

26.inheritance_basic_2.php

```

1 <?php
2 class Produk {
3     public $merek = "Sharp";
4     public $stok = 50;
5
6     public function cekStok(){
7         return "Sisa stok: ".$this->stok;
8     }
9 }
10
11 class Televisi extends Produk {
12     public $jenis = "Televisi";
13

```

```

14  public function cekStokTelevisi(){
15      return $this->jenis." ".$this->merek.", ".$this->cekStok();
16  }
17 }
18
19 $produk01 = new Televisi();
20 echo $produk01->jenis;           // Televisi
21 echo "<br>";
22 echo $produk01->cekStokTelevisi(); // Televisi Sharp, Sisa stok: 50

```

Sekarang di dalam class `Televisi` terdapat property `$jenis` dan method `cekStokTelevisi()`.

Di dalam method `cekStokTelevisi()`, saya mengakses property `$this->jenis` yang memang kepunyaan dari class `Televisi`, kemudian juga mengakses property `$this->merek` dan method `$this->cekStok()` kepunyaan class `Produk`. Ini bisa dilakukan karena kembali, semua property dan method dari *parent class* bisa diakses dari *child class*.

Lebih jauh lagi, saya bisa membuat class `TelevisiLCD` sebagai turunan dari class `Televisi`.

Artinya, class `TelevisiLCD` adalah cucu atau *grandchild* dari class `Produk`:

27.inheritance_basic_3.php

```

1  <?php
2  class Produk {
3      public $merek = "Sharp";
4      public $stok = 50;
5
6      public function cekStok(){
7          return "Sisa stok: ".$this->stok;
8      }
9  }
10
11 class Televisi extends Produk {
12     public $jenis = "Televisi";
13
14     public function cekStokTelevisi(){
15         return $this->jenis." ".$this->merek.", ".$this->cekStok();
16     }
17 }
18
19 class TelevisiLCD extends Televisi {
20     public $tipe = "LCD";
21
22     public function cekStokTelevisiLCD(){
23         return $this->tipe." ".$this->cekStokTelevisi();
24     }
25 }
26
27 $produk01 = new TelevisiLCD();
28 echo $produk01->merek;           // Sharp
29 echo "<br>";
30 echo $produk01->jenis;           // Televisi
31 echo "<br>";

```

```

32 echo $produk01->tipe;           // LCD
33 echo "<br>";
34 echo $produk01->cekStokTelevisiLCD(); // LCD Televisi Sharp, Sisa stok: 50

```

Perubahan dari kode sebelumnya ada di baris 19 – 25, dimana saya membuat class `TelevisiLCD` yang diturunkan dari class `Televisi`. Di dalam class `TelevisiLCD` sendiri terdapat property `$type` serta method `cekStokTelevisiLCD()`.

Di dalam method `cekStokTelevisiLCD()`, saya menyambung isi string `$this->tipe` dengan hasil pemanggilan `$this->cekStokTelevisi()`. Method `cekStokTelevisi()` berasal dari class `Televisi`, yakni *parent class* dari `TelevisiLCD`.

Aturan bahwa *child class* bisa mengakses *parent class* tetap berlaku hingga *grandparent class* (dan class turunannya jika ada).

Di baris 27 saya membuat object `$produk01` dari class `TelevisiLCD`. Kemudian di baris 28, saya mengakses property `$merek`. Property `$merek` ini adalah property kepunyaan class `Produk`, yakni *grandparent* dari class `TelevisiLCD`.

Di baris 29, giliran property `$jenis` milik class `Televisi` yang diakses, diikuti dengan mengakses property `$tipe` milik class `TelevisiLCD` di baris 32. Dan terakhir saya mengakses method `cekStokTelevisiLCD()` yang juga berhasil dijalankan.

Dalam PHP, tidak ada batasan sebanyak banyak turunan sebuah class. Bisa jadi untuk aplikasi yang kompleks, butuh penurunan class hingga 5 level ke bawah dan 5 level ke atas.

Sebuah class juga bisa diturunkan ke dalam class yang berbeda-beda, seperti contoh berikut:

28.inheritance_basic_4.php

```

1 <?php
2 class Produk {
3     public $sku;
4     public $stok;
5 }
6
7 class Televisi extends Produk {
8     public $ukuranLayar;
9 }
10
11 class MesinCuci extends Produk {
12     public $kapasitas;
13 }
14
15 class Speaker extends Produk {
16     public $konfigurasi;
17 }
18
19 $produk01 = new Televisi();
20 $produk02 = new MesinCuci();
21 $produk03 = new Speaker();

```

Saya menyederhanakan class `Produk` yang kali ini hanya berisi property `$sku` dan `$stok`. Di baris 7, class `Produk` diturunkan ke dalam class `Televisi`. Di baris 11, class `Produk` di turunkan ke dalam class `MesinCuci`. Dan di baris 15 class `Produk` juga diturunkan ke dalam class `Speaker`.

Dalam hal ini terlihat bahwa kita membuat sebuah struktur hirarki class. Class `Televisi`, `MesinCuci` dan `Speaker` adalah turunan dari class `Produk`. Nantinya class `Televisi` juga bisa diturunkan ke dalam class `TelevisiLCD`, `TelevisiLED`, `TelevisiPlasma`, dst. Begitu juga dengan class `MesinCuci` yang bisa diturunkan ke dalam class `MesinCuciFrontLoad`, `MesinCuciTopLoad`, dst.

Kenapa membuat struktur seperti ini? Karena setiap produk adalah unik dan memiliki property yang berbeda-beda. Sebuah mesin cuci memiliki property `$kapasitas`, yang nantinya bisa diisi misalnya dengan 7 Liter atau 10 Liter. Class `Speaker` akan butuh property `$konfigurasi`, apakah speaker itu 2.1, 5.1, atau 7.1 sesuai dengan jumlah satelit dan subwoofer dari sebuah produk speaker.

Namun setiap produk punya property yang sama, seperti `$sku` dan `$stok`. Daripada menulis ulang property `$sku` dan `$stok` di setiap class, lebih baik kita menempatkannya ke dalam class `Produk`. Ketika diturunkan, otomatis setiap class juga akan memiliki `$sku` dan `$stok`. Ini akan mengurangi terjadinya duplikasi data. Pendefinisian property `$sku` dan `$stok` cukup di 1 tempat saja.

Selain itu, kita memiliki sebuah konsep yang jelas. Setiap produk **pasti** memiliki property `$sku` dan `$stok`, karena semuanya diturunkan dari class `Produk`. Hal ini cukup penting karena nantinya kita bisa membuat method atau function yang memproses berbagai class turunan `Produk`.

Konsep yang sama juga berlaku untuk method. Method seperti `cekStok()` dan `tambahStok()` perlu dimiliki oleh setiap class untuk memeriksa dan menambah stok. Cukup dengan menulis method ini di class `Produk`, otomatis semua class juga memiliki method ini. Sebuah konsep yang sangat efisien.

Untuk memeriksa apakah sebuah class merupakan turunan dari class lain, kita bisa menggunakan function `is_a()`. Function ini membutuhkan 2 buah argument. Argument pertama berupa object yang diperiksa, dan argument kedua adalah string nama class. Function `is_a()` mengembalikan nilai boolean `true` atau `false` tergantung apakah object tersebut benar sebuah turunan atau tidak.

Berikut contoh penggunaan dari function `is_a()`:

29.inheritance_is_a.php

```
1 <?php
2 class Produk {
3     public $sku;
4     public $stok;
```

```

5   }
6
7   class Televisi extends Produk {
8     public $ukuranLayar;
9   }
10
11  class MesinCuci extends Produk {
12    public $kapasitas;
13  }
14
15  class Speaker extends Produk {
16    public $konfigurasi;
17  }
18
19 $produk01 = new Televisi();
20 $produk02 = new MesinCuci();
21 $produk03 = new Speaker();
22
23 var_dump( is_a($produk01, 'Produk') );           // bool(true)
24 var_dump( is_a($produk01, 'Televisi') );          // bool(true)
25 var_dump( is_a($produk02, 'Produk') );           // bool(true)
26 var_dump( is_a($produk02, 'Televisi') );          // bool(false)
27 var_dump( is_a($produk03, 'Speaker') );          // bool(true)

```

Kode di baris 1 – 21 sama seperti sebelumnya, dimana saya mendefinisikan class `Produk` yang kemudian diturunkan ke dalam class `Televisi`, `MesinCuci` dan `Speaker`. Di baris 19 – 21 setiap class di instansiasi ke dalam object `$produk01`, `$produk02` dan `$produk03`.

Pada baris 23 – 27, saya menampilkan hasil dari beberapa pemanggilan function `is_a()`. Karena fungsi ini mengembalikan nilai boolean, maka perlu menggunakan `var_dump()` agar hasilnya lebih jelas.

Perintah `is_a($produk01, 'Produk')` menghasilkan nilai **true** karena `$produk01` adalah turunan dari class `Produk`. Dan karena `$produk01` juga merupakan class `Televisi` maka hasil dari `is_a($produk01, 'Televisi')` juga **true**. Hal yang sama juga berlaku untuk object lain.

Di baris 26 saya menjalankan `is_a($produk02, 'Televisi')`, hasilnya **false** karena `$produk02` bukan bagian dari class `Televisi`, melainkan class `MesinCuci`.

Fungsi `is_a()` ini sering di pakai untuk program yang memiliki banyak object. Misalnya kita mendapati `$produk39`. Jika hasil dari perintah `is_a($produk39, 'Televisi')` adalah **true**, maka kita bisa mengakses property `$sku`, `$stok` dan `$ukuranLayar`.

Method Overriding

Inheritance menawarkan fleksibilitas namun juga masalah tersendiri. Silahkan anda pelajari sejenak kode berikut dan tebak seperti apa hasilnya:

30.inheritance_method_overriding_1.php

```
1 <?php
```

```

2 class Produk {
3     public function hello(){
4         return "Ini dari Produk";
5     }
6 }
7
8 class Televisi extends Produk {
9     public function hello(){
10        return "Ini dari Televisi";
11    }
12 }
13
14 $produk01 = new Televisi();
15 echo $produk01->hello();

```

Di dalam class `Produk` saya membuat method bernama `hello()`. Method ini hanya berisi perintah `return "Ini dari Produk"` (baris 3 - 5). Kemudian class `Produk` saya turunkan ke dalam class `Televisi`. Di dalam class `Televisi` juga terdapat method `hello()` yang kali ini mengembalikan string `"Ini dari Televisi"` (baris 9 - 11).

Pertanyaannya, ketika variabel `$produk01` yang merupakan object dari class `Televisi` mengakses method `hello()`, method manakah yang dipakai? apakah `hello()` kepunyaan class `Televisi`, atau `hello()` dari `Produk`? Atau malah error karena ada 2 nama method yang sama?

Berikut hasilnya:

```
Ini dari Televisi
```

Yang akan dijalankan adalah method `hello()` dari class `Televisi`. Di sini kita melakukan sebuah sesuatu yang dinamakan **method overriding**, yakni menimpa method *parent class* dengan cara menulis nama method yang sama di *child class*.

Masalahnya, sekarang method `hello()` kepunyaan class `Produk` tidak bisa diakses lagi. Jika di dalam class `Televisi` saya menulis `$this->hello()`, yang akan berjalan tetap method punya class `Televisi`.

Untuk mengatasi hal ini, PHP memiliki konstanta khusus: **parent**.

Konstanta `parent` mirip seperti `$this`, tapi yang diakses adalah **parent class**. Selain itu, operator yang dipakai tidak lagi tanda panah, tetapi titik dua, dua kali, yakni tanda `" :: "`. Sehingga untuk mengakses method `hello()` milik class `Produk` dari dalam class `Televisi`, perintahnya adalah `parent::hello()`.

Tanda titik dua, dua kali ini dikenal dengan istilah **scope resolution operator**. Nantinya kita akan lihat ada penggunaan lain dari tanda ini.

Baik, berikut contoh kode program untuk mengakses method `hello()` milik class `Produk`:

31.inheritance_method_overriding_2.php

```

1  <?php
2  class Produk {
3      public function hello(){
4          return "Ini dari Produk";
5      }
6  }
7
8  class Televisi extends Produk {
9      public function hello(){
10         return "Ini dari Televisi";
11     }
12
13     public function helloProduk(){
14         return parent::hello();
15     }
16 }
17
18 $produk01 = new Televisi();
19 echo $produk01->helloProduk();      // Ini dari Produk

```

Perubahan dari kode sebelumnya ada di dalam class `Televisi`, yakni baris 13 – 15. Saya membuat method `helloProduk()` yang berfungsi sebagai "jembatan" untuk bisa mengakses method `hello()` dari class `Produk`. Caranya, jalankan perintah `parent::hello()`.

Sekarang, object `$produk01` bisa mengakses method `hello()` milik class `Televisi`, dan juga bisa mengakses method `hello()` milik class `Produk` melalui method `helloProduk()`.

Mungkin terlintas di pikiran anda bahwa, daripada repot-repot seperti ini, akan lebih baik kita tidak menggunakan nama method yang sama di *child class* dan *parent class*. Memang benar, dan itu adalah cara paling ideal. Jika dari awal kita merancang method `helloProduk()` di class `Produk` dan `helloTelevisi()` di class `Televisi`, tidak akan ada masalah dengan "method yang tertimpa" seperti kasus di atas.

Namun ada situasi dimana nama class ini mau tidak mau harus sama, yakni untuk **constructor**. Kita akan lihat contoh kasusnya sesaat lagi, sebelum ke sana saya akan bahas sedikit tentang **property overriding**.

Property Overriding

Melihat namanya, saya yakin anda sudah bisa menebak bahwa kali ini kita akan membahas kasus ketika property di *parent class* dan *child class* menggunakan nama yang sama. Betul, dan berikut contoh kode program untuk *property overriding*:

32.inheritance_property_overriding_1.php

```

1  <?php
2  class Produk {
3      public $merek = "Sony";
4  }

```

```

5
6 class Televisi extends Produk {
7     public $merek = "Panasonic";
8 }
9
10 $produk01 = new Televisi();
11 echo $produk01->merek;      // Panasonic

```

Di sini saya membuat class `Produk` dengan 1 property, yakni `$merek`. Kemudian class `Televisi` yang merupakan turunan dari class `Produk` juga memiliki nama property yang sama, yakni `$merek`.

Ketika di akses dari object `$produk01` yang diinisialisasi dari class `Televisi`, property `$merek` merujuk ke class `Televisi`, bukan class `Produk`. Hasil ini kurang lebih sama seperti kasus di *method overriding*, di mana method *child class* akan menimpa method *parent class*.

Mari coba akses property `$merek` milik class `Produk` menggunakan konstanta `parent` seperti di method sebelumnya:

33.inheritance_property_overriding_2.php

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4 }
5
6 class Televisi extends Produk {
7     public $merek = "Panasonic";
8     public $merekProduk = parent::$merek;
9 }
10
11 $produk01 = new Televisi();
12 echo $produk01->merekProduk;

```

Kali ini saya membuat sebuah property `$merekProduk` di dalam class `Televisi` (baris 8). Property `$merekProduk` ini diisi dengan hasil dari `parent::$merek`, dimana saya mencoba mengakses property `$merek` milik class `Produk`.

Variabel `$object01` kemudian di instansiasi sebagai object dari class `Televisi`. Di baris 12 saya menampilkan isi property `$produk01->merekProduk`. Seperti apa hasilnya?

Fatal error: Uncaught Error: Undefined constant parent::\$merek

Hasilnya berupa pesan error. Kenapa? Karena di dalam PHP ternyata kita tidak bisa mengakses property *parent class* yang sudah tertimpa. Perintah `parent::$merek` di dalam class `Televisi` akan merujuk ke sebuah **konstanta** `merek`, bukan property `merek` dari class `Produk`. Mengenai konstanta sendiri akan kita bahas dalam bagian tersendiri.

Dengan demikian, satu-satunya cara untuk menghindari masalah property yang tertimpa adalah dengan menulis nama property yang berbeda antara *parent class* dengan *child class*.

Constructor Overriding

Kita sudah membahas tentang *method overriding* dan *property overriding*. **Constructor overriding** sebenarnya adalah kasus khusus dari *method overriding*, dimana constructor *parent class* bisa tertutupi oleh constructor dari *child class*.

Tapi sebelum sampai ke sana, kita akan lihat terlebih dahulu seperti apa efek **inheritance** ke dalam constructor. Sekedar pengingat, constructor adalah method khusus yang secara otomatis berjalan pada saat instansiasi object. Berikut contohnya:

34.inheritance_constructor_1.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 $produk01 = new Produk("Televisi","Samsung",20);
15
16 echo "<pre>";
17 print_r ($produk01);
18 echo "</pre>";

```

Hasil kode program:

```

Produk Object
(
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)

```

Kode program ini sangat mirip seperti contoh yang kita pakai pada saat mempelajari constructor. Perbedaannya hanya di baris akhir dimana saya memakai tambahan tag `<pre>` agar tampilan `print_r()` lebih nyaman dilihat. Sekarang, isi seluruh property `$produk01` ditampilkan secara vertikal ke bawah.

Dalam kode tersebut saya merancang class `Produk` dengan sebuah constructor. Constructor pada baris 7 – 11 dipakai untuk mengisi property class `Produk` yakni `$jenis`, `$merek` dan `$stok`. Jika anda kurang paham maksud kode ini, boleh kembali sejenak ke pembahasan tentang constructor.

Baik, bagaimana jika class `Produk` saya turunkan ke dalam class `Televisi`? Apakah constructor

milik Produk juga bisa diakses dari \$produk01? Mari kita coba:

35.inheritance_constructor_2.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 class Televisi extends Produk {
15 }
16
17 $produk01 = new Televisi("Televisi", "Samsung", 20);
18
19 echo "<pre>";
20 print_r ($produk01);
21 echo "</pre>";

```

Hasil kode program:

```

Televisi Object
(
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)

```

Di baris 14, saya membuat class `Televisi` sebagai turunan dari class `Produk`. Class `Televisi` sendiri tidak berisi kode apapun.

Di baris 17, variabel `$produk01` saya instansiasi sebagai object `Televisi`, bukan lagi object dari class `Produk`. Ternyata constructor class `Produk` tetap bisa diakses, meskipun ini adalah object dari class `Televisi` (bisa dilihat di baris 1 hasil `print_r()`, yakni `Televisi Object`).

Hal di atas tidak mengejutkan, karena *child class* memang akan mewarisi semua property dan method dari *parent class*. Namun dengan catatan selama *child class* tidak menimpa property atau method *parent class*.

Sekarang bagaimana jika ternyata dalam class `Televisi` juga terdapat constructor?

36.inheritance_constructor_overriding_1.php

```

1 <?php
2 class Produk {
3     public $jenis;

```

```

4  public $merek;
5  public $stok;
6
7  public function __construct($jenis, $merek, $stok){
8      $this->jenis = $jenis;
9      $this->merek = $merek;
10     $this->stok = $stok;
11 }
12 }
13
14 class Televisi extends Produk {
15     public function __construct(){
16     }
17 }
18
19 $produk01 = new Televisi("Televisi", "Samsung", 20);
20
21 echo "<pre>";
22 print_r ($produk01);
23 echo "</pre>";

```

Hasil kode program:

```

Televisi Object
(
    [jenis] =>
    [merek] =>
    [stok] =>
)

```

Saya menambah method `__construct()` ke dalam class `Televisi` di baris 15. Hasilnya, terjadi **constructor overriding**, dimana constructor child class akan mengambil alih constructor parent class. Meskipun pada saat proses instansiasi `$produk01` tetap ada argument, tapi ini tidak sampai ke constructor class `Produk`.

Terdapat 2 cara untuk mengatasi masalah ini. Pertama, kita bisa membuat ulang constructor di dalam class `Televisi`, seperti contoh berikut:

37.inheritance_constructor_overriding_2.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 class Televisi extends Produk {

```

```

15  public function __construct($jenis, $merek, $stok){
16      $this->jenis = $jenis;
17      $this->merek = $merek;
18      $this->stok = $stok;
19  }
20 }
21
22 $produk01 = new Televisi("Televisi", "Samsung", 20);
23
24 echo "<pre>";
25 print_r ($produk01);
26 echo "</pre>";

```

Hasil kode program:

```

Televisi Object
(
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)

```

Sekarang hasil kode program sudah sesuai sebagaimana mestinya, dimana semua property sudah kembali terisi.

Akan tetapi ini tidak efisien, karena terdapat duplikasi kode program. Isi constructor class `Televisi` sama persis dengan isi constructor di class `Produk`. Akan lebih baik jika kita memanggil constructor `Produk` dari dalam class `Televisi`, dan inilah pilihan kedua untuk mengatasi constructor *overriding*:

38.inheritance_constructor_overriding_3.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 class Televisi extends Produk {
15     public function __construct($jenis, $merek, $stok){
16         parent::__construct($jenis, $merek, $stok);
17     }
18 }
19
20 $produk01 = new Televisi("Televisi", "Samsung", 20);
21
22 echo "<pre>";

```

```
23 print_r ($produk01);
24 echo "</pre>";
```

Hasil kode program:

```
Televisi Object
(
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)
```

Fokus utama ada di cara pendefinisian constructor class `Televisi` (baris 14 – 17). Saya tetap harus mendefinisikan method `__construct()` dan menyediakan semua parameter yang diperlukan. Parameter ini nantinya akan menjadi argument untuk memanggil constructor *parent class* dengan perintah `parent::__construct()`.

Agar lebih jelas, berikut alur pengiriman argument yang terjadi:

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

public function __construct($jenis, $merek, $stok){
    $this->jenis = $jenis; ⑤
    $this->merek = $merek;
    $this->stok = $stok;
}

class Televisi extends Produk { ②
    public function __construct($jenis, $merek, $stok){
        parent::__construct($jenis, $merek, $stok); ③
    }
}

$produk01 = new Televisi("Televisi", "Samsung", 20); ①
```

Sebagai contoh, kita akan lihat perjalanan string "`Televisi`" sejak diinput sebagai argument hingga ke property `$jenis` di class `Produk`:

1. Argument "`Televisi`" diinput pada saat proses instansiasi.
2. Argument "`Televisi`" masuk ke dalam parameter `$jenis` di constructor class `Televisi`.
3. Parameter `$jenis` menjadi inputan (*argument*) untuk constructor parent class dari `Televisi`, yakni constructor untuk class `Produk`.
4. Argument `$jenis` diterima dan diinput ke dalam parameter `$jenis` di constructor class `Produk`.
5. Parameter `$jenis` menjadi nilai inputan untuk property `$jenis`.

6. Property \$jenis sudah berisi nilai "Televisi".

Huff.. sebuah perjalanan panjang untuk sekedar mengisi sebuah property class. Tapi konsep ini cukup penting untuk dipahami karena pemanggilan constructor parent class akan sering dilakukan.

Mari masuk ke contoh yang sedikit lebih rumit (dan lebih nyata). Seperti yang sudah kita pelajari, **inheritance** dipakai untuk membuat struktur hierarki class. Sebuah class "umum" akan diturunkan ke dalam class yang lebih "khusus". Class Produk kita turunkan ke dalam class Televisi karena nantinya di dalam class Televisi ini nantinya ditambah dengan property dan method yang hanya khusus ada di dalam class Televisi saja.

Konsep ini sudah pernah kita buat, dimana class Televisi akan memiliki property \$ukuranLayar yang bisa diisi dengan nilai seperti 21, 32, atau 40 (dalam satuan inci). Property ini spesifik untuk class Televisi saja.

Dengan demikian, class Televisi akan memiliki 4 buah property, yakni \$jenis, \$merk, \$stock yang didapat dari turunan class Produk serta property \$ukuranLayar. Saya ingin semua property ini bisa diinput para saat proses instansiasi seperti contoh berikut:

```
$produk01 = new Televisi("Televisi", "Samsung", 20, 32);
```

Perhatikan sekarang ada 4 buah argument, dimana argument ke-4, yakni angka 32 adalah inputan untuk property \$ukuranLayar di class Televisi.

Bagaimana cara merancang constructor-nya? Pertama, kita bisa menggunakan cara biasa, yakni membuat semua proses pemindahan dari argument ke dalam property di dalam constructor class Televisi:

39.inheritance_constructor_overriding_4.php

```

1  <?php
2  class Produk {
3      public $jenis;
4      public $merek;
5      public $stok;
6
7      public function __construct($jenis, $merek, $stok){
8          $this->jenis = $jenis;
9          $this->merek = $merek;
10         $this->stok = $stok;
11     }
12 }
13
14 class Televisi extends Produk {
15     public $ukuranLayar;
16
17     public function __construct($jenis, $merek, $stok, $ukuranLayar){
18         $this->jenis = $jenis;
19         $this->merek = $merek;
```

```

20     $this->stok = $stok;
21     $this->ukuranLayar = $ukuranLayar;
22 }
23 }
24
25 $produk01 = new Televisi("Televisi","Samsung",20,32);
26
27 echo "<pre>";
28 print_r ($produk01);
29 echo "</pre>";

```

Hasil kode program:

```

Televisi Object
(
    [ukuranLayar] => 32
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)

```

Di baris 17 – 22 saya membuat constructor class `Televisi` yang menampung semua argument dan menginputnya ke dalam property. Cara ini bisa berjalan dan hasilnya sudah sesuai.

Tapi kembali, di sini terjadi duplikasi data karena di class `Produk` juga ada constructor. Cara yang paling baik adalah memanggil constructor class `Produk` untuk proses inputan property `$jenis`, `$merek` dan `$stock` (karena ketiganya adalah kepunyaan class `Produk`), kemudian memproses property `$ukuranLayar` di constructor class `Televisi`.

Berikut kode programnya:

40.inheritance_constructor_overriding_5.php

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 class Televisi extends Produk {
15     public $ukuranLayar;
16
17     public function __construct($jenis, $merek, $stok, $ukuranLayar){
18         parent::__construct($jenis, $merek, $stok);
19         $this->ukuranLayar = $ukuranLayar;
20     }

```

```

21 }
22
23 $produk01 = new Televisi("Televisi", "Samsung", 20, 32);
24
25 echo "<pre>";
26 print_r ($produk01);
27 echo "</pre>";

```

Hasil kode program:

```

Televisi Object
(
    [ukuranLayar] => 32
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)

```

Perhatikan isi constructor class Televisi di baris 18 dan 19. Di sini saya memanggil constructor class Produk untuk "menangani" proses pemindahan argument `$jenis`, `$merek` dan `$stock`, sedangkan untuk argument `$lebarLayar`, tetap di proses di dalam class Televisi.

Cara ini nantinya bisa kita pakai untuk class-class lain. Tujuan akhirnya adalah untuk menghindari duplikasi data.

Sampai di sini anda mungkin sudah mulai bertanya, kenapa kita harus repot-repot sampai ke constructor overriding hanya untuk menginput nilai ke dalam property? Akan lebih cepat jika ditulis seperti ini saja:

41.tanpa_inheritance.php

```

1 <?php
2 class Televisi {
3     public $jenis;
4     public $merek;
5     public $stok;
6     public $ukuranLayar;
7
8     public function __construct($jenis, $merek, $stok, $ukuranLayar){
9         $this->jenis = $jenis;
10        $this->merek = $merek;
11        $this->stok = $stok;
12        $this->ukuranLayar = $ukuranLayar;
13    }
14 }
15
16 $produk01 = new Televisi("Televisi", "Samsung", 20, 32);
17
18 echo "<pre>";
19 print_r ($produk01);
20 echo "</pre>";

```

Hasil kode program:

```
Televisi Object
(
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
    [ukuranLayar] => 32
)
```

Betul, kode di atas juga akan menampilkan hasil yang sama. Namun pemrograman berorientasi object yang baik, lebih "melihat ke depan". Saya membuat terlebih dahulu class Produk sebagai induk dari semua class karena memprediksi ke depan saya akan butuh puluhan hingga ratusan produk. Saat ini yang diperlukan memang cuma baru produk Televisi saja, tapi dengan membuat class Produk, saya sudah mempersiapkan kode program tersebut untuk menangani hal yang besar suatu saat nanti.

Inilah salah satu perbedaan konsep berfikir dari *object oriented programming* dibandingkan *procedural programming*. Agar bisa memanfaatkan paradigma OOP, kita harus mau bersusah-sudah di awal perancangan kode program. Harapannya, jika proyek ini suatu saat menjadi besar, kode program juga sudah bisa mengikuti.

Destructor Overriding

Dibandingkan constructor, *destructor overriding* ini tidak terlalu sering kita pakai. Namun agar pembahasan kita lebih lengkap, saya akan membahasnya sedikit.

Secara umum, tidak ada perbedaan mendasar dari cara pemanggilan *destructor overriding* dengan *constructor overriding*. Berikut contoh penggunaannya:

42.inheritance_destructor_overriding.php

```
1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12
13    public function __destruct(){
14        unset($this->jenis, $this->merek, $this->stok);
15        echo "Property class Produk sudah dihapus... <br>";
16    }
17
18 }
```

```

19 class Televisi extends Produk {
20     public $ukuranLayar;
21
22
23     public function __construct($jenis, $merek, $stok, $ukuranLayar){
24         parent::__construct($jenis, $merek, $stok);
25         $this->ukuranLayar = $ukuranLayar;
26     }
27
28     public function __destruct(){
29         unset($this->ukuranLayar);
30         echo "Property class Televisi sudah dihapus... <br>";
31         parent::__destruct();
32     }
33 }
34
35 $produk01 = new Televisi("Televisi", "Samsung", 20, 32);
36
37 echo "<pre>";
38 print_r ($produk01);
39 echo "</pre>";

```

Hasil kode program:

```

Televisi Object
(
    [ukuranLayar] => 32
    [jenis] => Televisi
    [merek] => Samsung
    [stok] => 20
)
Property class Televisi sudah dihapus...
Property class Produk sudah dihapus...

```

Saya memodifikasi sedikit kode program sebelumnya, kali ini dengan tambahan destructor di class `Produk` (baris 13 – 16) dan destructor di class `Televisi` (baris 28 – 32).

Isi kedua destructor sama-sama menghapus property menggunakan function `unset()`. Destructor di class `Produk` akan meng-unset property `$jenis`, `$merek` dan `$stok`, kemudian saya memberikan perintah `echo` sebagai penanda bahwa destructor sudah berjalan.

Untuk destructor di class `Televisi`, yang di unset hanya 1 property saja, yakni `$lebarLayar`. Namun karena destructor di class `Televisi` ini akan menutup atau men-override destructor di class `Produk`, maka saya harus jalankan dengan perintah `parent::__destruct()` agar destructor di class `Produk` juga dijalankan.

Hasilnya, akan tampil teks hasil `echo` sebagai bukti bahwa constructor kedua class sudah dijalankan secara otomatis pada saat object `#produk01` dihapus oleh PHP. Kembali karena PHP sudah memiliki fitur '*'garbage collection'*', penghapusan property seperti ini sebenarnya tidak terlalu diperlukan.

Final Keyword

PHP menyediakan keyword **final** yang bisa dipakai untuk "memutus" overriding dan penurunan. Sebuah method yang di set sebagai *final*, tidak bisa di override oleh method lain di child class. Sedangkan class yang di set sebagai *final*, tidak bisa lagi diturunkan kepada class lain.

Mari kita lihat praktik **final method** terlebih dahulu:

42a.final_method.php

```

1 <?php
2 class Produk {
3     final public function hello(){
4         return "Ini dari Produk";
5     }
6 }
7
8 class Televisi extends Produk {
9     public function hello(){
10        return "Ini dari Televisi";
11    }
12 }
```

Hasil kode program:

```
// Fatal error: Cannot override final method Produk::hello()
```

Di dalam class *Produk* saya menulis keyword **final** sebelum method *hello()* di baris 3. Akibatnya, ketika class *Televisi* mencoba mendefinisikan ulang method *hello()* akan tampil pesan error: *Cannot override final method*.

Meskipun di set sebagai *final*, method *hello()* sendiri tetap bisa diakses sebagaimana biasanya:

42b.final_method_akses.php

```

1 <?php
2 class Produk {
3     final public function hello(){
4         return "Ini dari Produk";
5     }
6 }
7
8 $produk01 = new Produk();
9 echo $produk01->hello(); // Ini dari Produk
```

Bagaimana dengan **final class**? Berikut contoh praktiknya:

42c.final_class.php

```
1 <?php
```

```
2 final class Produk {
3     public function hello(){
4         return "Ini dari Produk";
5     }
6 }
7
8 class Televisi extends Produk {
9 }
```

Hasil kode program:

```
// Fatal error: Class Televisi may not inherit from final class (Produk)
```

Saya menambahkan keyword **final** sebelum penulisan class **Produk**. Akibatnya, proses penurunan di baris 8 tidak bisa dilakukan. PHP mengeluarkan pesan error bahwa class **Televisi** tidak bisa diturunkan dari final class.

Keyword **final** ini tidak akan terlalu sering di pakai, namun dalam situasi dimana kita tidak ingin sebuah method di definisikan ulang, atau class tidak boleh diturunkan lagi, keyword ini bisa menjadi solusi.

Tapi tunggu dulu, bagaimana dengan **final property**? Berikut percobaannya:

42d.final_property.php

```
1 <?php
2 class Produk {
3     final public $hehe = 12;
4 }
```

Hasil kode program:

```
Fatal error: Cannot declare property Produk::$hehe final, the final modifier is allowed only for methods and classes
```

Ternyata PHP tidak memperbolehkan hal ini, keyword **final** hanya bisa dipakai untuk method dan class saja. Jadi bagaimana jika kita tidak ingin sebuah property di definisikan ulang di dalam class turunan?

Solusinya adalah memakai **class constant**, yakni konstanta class. Lebih jauh tentang **class constant** akan kita bahas pada bagian tersendiri.

2.8. Visibility

Visibility atau disebut juga sebagai **access modifier** adalah sebuah cara untuk membatasi akses ke property dan method class. Tujuannya, supaya kode internal di dalam class tidak bisa diakses atau "diganggu" oleh kode program yang ada di luar class.

Kasus seperti ini sering diterapkan pada project yang dibuat oleh tim (terdiri dari banyak

programmer). Misalnya perusahaan Lapakpedia sedang mengembangkan aplikasi e-commerce, programmer A mendapat tugas untuk membuat logika pemrosesan produk, sedangkan programmer B bertugas untuk menangani pemrosesan dari dan ke database.

Programmer A memberitahu programmer B bahwa dalam kode program yang dia buat terdapat class Produk. Class Produk ini memiliki method `cekStok()` dan `setStock($jumlah)` yang dirancang sebagai sarana input dan output data.

Di dalam class Produk sebenarnya juga terdapat method lain seperti `hitungTotalStok()`, namun method ini hanya dipakai dalam internal class Produk saja. Kode program di luar class tidak diizinkan untuk mengakses method `hitungTotalStok()`. Pembatasan seperti inilah yang bisa dilakukan memakai *visibility* atau *access modifier*.

PHP (dan mayoritas bahasa pemrograman object lain), memiliki 3 level visibility: **public**, **private** dan **protected**:

- ◆ Visibility **public** adalah level yang paling terbuka. Jika sebuah property atau method di set sebagai **public**, maka seluruh kode program di dalam dan luar class bisa mengaksesnya.
- ◆ Visibility **private** adalah level yang paling tertutup. Jika sebuah property atau method di set sebagai **private**, yang bisa mengaksesnya hanya kode program di dalam class itu saja. Kode program di luar class tidak bisa mengakses property dan method ini (akan menghasilkan error).
- ◆ Visibility **protected** ada di posisi "tengah". Jika sebuah property atau method di set sebagai **protected**, kode program di luar class tidak bisa mengaksesnya, kecuali dari turunan class tersebut.

Kita akan bahas satu per satu metode pembatasan ini.

Visibility Public

Jika sebuah property atau method di set sebagai **public**, maka seluruh kode program di dalam dan luar class bisa mengaksesnya. Contoh kode program yang kita buat sejak awal buku ini semuanya sudah menggunakan visibility **public**. Berikut contohnya:

43.visibility_public.php

```

1 <?php
2 class Produk {
3     public $merek;
4
5     public function hello(){
6         return "Ini adalah Produk";
7     }
8 }
9
10 $produk01 = new Produk();

```

```

11 $produk01->merek = "Asus";
12
13 echo $produk01->merek;           // Asus
14 echo "<br>";
15 echo $produk01->hello();         // Ini adalah Produk

```

Dalam kode program di atas saya membuat class `Produk` dengan property `$merek` dan method `hello()`. Keduanya di set sebagai `public` sehingga bisa diakses dari luar class seperti di baris 13 dan 15.

PHP mewajibkan penulisan visibility ini untuk setiap property. Jika tidak, akan menghasilkan error:

```

1 <?php
2 class Produk {
3     $merek;
4     ...
5 }

```

Hasil kode program:

```
Parse error: syntax error, unexpected '$merek' (T_VARIABLE), expecting function (T_FUNCTION) or const (T_CONST)
```

Di baris 3 saya tidak menulis visibility untuk property `$merek`, akibatnya tampil pesan error.

Untuk menjaga kompatibilitas, yakni menyesuaikan diri dengan kode program lama, PHP 8 masih mendukung penulisan keyword `var` sebagai alternatif visibility `public`. Berikut contohnya:

```

1 <?php
2 class Produk {
3     var $merek;
4     ...
5 }

```

Keyword `var` dipakai pada era PHP 4 dan memiliki efek yang sama seperti `public`, yakni bisa diakses dari mana saja.

Di versi awal PHP 5, penulisan `var` untuk membuat visibility sudah berstatus *deprecated* (besar kemungkinan akan dihapus dari PHP) serta menampilkan error level `E_STRICT`. Namun sejak PHP 5.3 ke atas, penggunaan keyword `var` sudah di perbolehkan kembali (mengalami *de-deprecated*). Akan tetapi tetap disarankan menggunakan keyword `public` daripada `var` karena lebih umum.

Sedangkan untuk method, visibility tidak harus ditulis dan tidak akan menghasilkan error seperti contoh berikut:

```

1 <?php
2 class Produk {

```

```

3     ...
4     function hello(){
5         return "Ini adalah Produk";
6     }
7 }
```

Dalam kode di atas saya langsung menulis function hello, tanpa menambahkan public di awal. Jika ditulis tanpa visibility seperti ini, method hello() dianggap sebagai public. Namun cara yang disarankan tetap menulis visibility agar kode program menjadi lebih jelas.

Visibility Private

Jika sebuah property atau method di set sebagai private, maka property dan method tersebut tidak bisa diakses dari luar class (akan menghasilkan pesan error). Berikut contohnya:

44.visibility_private_error.php

```

1 <?php
2 class Produk {
3     private $merek;
4
5     private function hello(){
6         return "Ini adalah Produk";
7     }
8 }
9
10 $produk01 = new Produk();
11
12 // Fatal error: Uncaught Error: Cannot access private property Produk::$merek
13 $produk01->merek = "Asus";
14
15 // Fatal error: Uncaught Error: Cannot access private property Produk::$merek
16 echo $produk01->merek;
17
18 // Fatal error: Uncaught Error: Call to private method Produk::hello()
19 echo $produk01->hello();
```

Kali ini property \$merek dan method hello() saya set sebagai private (baris 3 dan 5), sehingga keduanya tidak akan bisa diakses dari luar class Produk. Kode program di baris 13, 16 dan 19 semuanya akan menghasilkan error.

Pembatasan akses private seperti ini sangat cocok untuk property dan method internal yang hanya dipakai di dalam class itu saja.

Yang sering dilakukan adalah, men-set semua property sebagai private kemudian membuat method untuk mengakses nilai dari property tersebut. Berikut contohnya:

45.visibility_private.php

```

1 <?php
2 class Produk {
3     private $merek;
```

```

4
5     public function setMerek($merek){
6         $this->merek = $merek;
7     }
8
9     public function getMerek(){
10        return $this->merek;
11    }
12 }
13
14 $produk01 = new Produk();
15 $produk01->setMerek("Asus");
16
17 echo $produk01->getMerek();           // Asus

```

Saya membatasi hak akses ke property `$merek` dengan cara men-setnya sebagai `private` (baris 3). Akibatnya, property ini tidak akan bisa diakses dari luar class.

Namun, saya bisa membuat function `setMerek()` dan `getMerek()` untuk mengakses property `$merek` secara tidak langsung. Jika saya ingin mengubah isi property `$merek`, caranya adalah dengan memanggil method `setMerek("nama_merek")`, kemudian jika ingin menampilkan nilainya, bisa melalui method `getMerek()`.

Setelah membuat object `$produk01` di baris 14, saya men-set nilai property `$merek` dengan perintah `$produk01->setMerek("Asus")`, kemudian menampilkannya dengan mengakses `$produk01->getMerek()`.

Teknik seperti ini dikenal dengan sebutan **getter** dan **setter** yang akan kita bahas dengan lebih detail sesaat lagi.

Visibility Protected

Cara kerja `protected` mirip seperti `private`, yakni tidak bisa diakses dari luar class. Namun bedanya, `protected` tetap bisa diakses dari class turunan (*child class*).

Sebagai contoh pertama, mari kita lihat efek `protected` tanpa penurunan terlebih dahulu:

46.visibility_protected.php

```

1 <?php
2 class Produk {
3     protected $merek;
4
5     protected function hello(){
6         return "Ini adalah Produk";
7     }
8 }
9
10 $produk01 = new Produk();
11
12 // Fatal error: Uncaught Error: Cannot access protected property Produk::$merek
13 $produk01->merek = "Asus";

```

```

14
15 // Fatal error: Uncaught Error: Cannot access protected property Produk::$merek
16 echo $produk01->merek;
17
18 // Fatal error: Uncaught Error: Call to protected method Produk::hello()
19 echo $produk01->hello();

```

Saya men-set property `$merek` dan method `hello()` sebagai `protected`. Hasilnya, tampil pesan error seperti pada contoh pembatasan akses `private`. Ini membuktikan bahwa `protected` tidak bisa diakses dari luar class, kecuali untuk *child class* seperti contoh berikut:

47.visibility_inheritance_protected.php

```

1 <?php
2 class Produk {
3     protected $merek = "Asus";
4
5     protected function hello(){
6         return "Ini adalah Produk";
7     }
8 }
9
10 class Laptop extends Produk{
11     public function helloLaptop(){
12         return $this->hello()." Laptop ".$this->merek;
13     }
14 }
15
16 $produk01 = new Laptop();
17 echo $produk01->helloLaptop();      // Ini adalah Produk Laptop Asus

```

Di sini saya menurunkan class `Produk` ke dalam class `Laptop` (baris 10 - 14). Dalam class `Laptop` terdapat method `helloLaptop()` yang isinya akan mengakses method `hello()` dan property `$merek`.

Meskipun di dalam method `helloLaptop()` saya menggunakan *pseudo-variable* `$this`, yang diakses adalah method dan property dari class `Produk`, karena di dalam class `Laptop` tidak ada method `hello()` dan property `$merek`.

Hasilnya, method `helloLaptop()` tetap bisa mengakses method `hello()` dan property `$merek` milik parent class `Produk`, karena kedua property ini di set sebagai `protected`.

Sebagai percobaan, bagaimana jika method `hello()` dan property `$merek` di class `Produk` saya set sebagai `private`? Berikut hasilnya:

48.visibility_inheritance_private.php

```

1 <?php
2 class Produk {
3     private $merek = "Asus";
4
5     private function hello(){

```

```

6     return "Ini adalah Produk";
7 }
8 }
9
10 class Laptop extends Produk{
11     public function helloLaptop(){
12         return $this->hello()." Laptop ".$this->merek;
13     }
14 }
15
16 $produk01 = new Laptop();
17 echo $produk01->helloLaptop();
18 // Fatal error:
19 // Uncaught Error: Call to private method Produk::hello() from context 'Laptop'

```

Sekarang terjadi error karena method `helloLaptop()` tetap tidak bisa mengakses property yang di set sebagai `private` di `parent class Produk`.

Visibility atau *access modifier* yang kita bahas di sini adalah penerapan dari konsep pemrograman object yang dikenal sebagai **encapsulation** (atau diterjemahkan sebagai *enkapsulasi*) yakni sebuah metode untuk menyembunyikan data dari "dunia luar".

Data yang dimaksud lebih ke *property class*. Sebaiknya, kita tidak mengizinkan kode apapun di luar class untuk mengakses property secara langsung. Dalam beberapa bahasa pemrograman seperti **Smalltalk** dan **Ruby**, secara bawaan pengaksesan property [hanya bisa dilakukan dari method](#). Namun di bahasa pemrograman seperti PHP, Java, C++ dan Python, pengaturan hak akses property menjadi tanggung jawab programmer yang membuatnya.

Dengan menyembunyikan property (dan juga method), kita membatasi interaksi ke dalam class yang akan membuat kode lebih sederhana. Pengguna dari class `Produk` cukup mengakses method `getMerek()` untuk mendapatkan nama merek, ia tidak perlu paham bagaimana proses yang terjadi di dalam class `Produk` ini.

Dalam kehidupan sehari-hari, kita tidak perlu paham bagaimana listrik sampai ke rumah dari generator PLN. Method yang disediakan oleh PLN hanya saklar on dan off di meteran listrik. Jika saklar ini off, maka hubungan listrik dari rumah ke gardu PLN akan terputus. Kita pun juga dibatasi dan tidak boleh mengutak-atik "daleman" dari meteran listrik. Analogi seperti inilah yang dipakai di pemrograman berorientasi objek.

Masih berhubungan dengan konsep **encapsulation**, berikutnya kita akan membahas tentang `getter` dan `setter`.

2.9. Getter dan Setter

Getter dan **setter** adalah sebutan untuk method yang mengisi (set) dan mengambil nilai (get) dari sebuah property.

Seperi yang sudah kita singgung sebelumnya, sedapat mungkin property harus selalu di buat

`private` agar tidak bisa diakses dari luar class. Untuk bisa memahami mengapa harus seperti itu, kita akan bahas dengan sedikit contoh kasus.

Sebelumnya, berikut contoh kode program dengan property yang di-set ke `public`:

49.tanpa_getter_dan_setter.php

```

1 <?php
2 class Produk {
3     public $merek;
4     public $stok;
5 }
6
7 $produk01 = new Produk();
8 $produk01->merek = "Acer";
9 $produk01->stok = 10;
10
11 echo $produk01->merek;      // Acer
12 echo "<br>";
13 echo $produk01->stok;      // 10

```

Di sini saya membuat class `Produk` yang isinya hanya 2 buah property, yakni `$merek` dan `$stok`. Class `Produk` selanjutnya di instansiasi ke dalam object `$produk01`.

Di baris 8, property `$merek` diisi string "Acer", dan di baris 9 property `$stok` diisi angka 10. Kemudian kedua property ini ditampilkan menggunakan perintah `echo`. Hal ini bisa kita lakukan karena property `$merek` dan `$stok` di set sebagai `public`. Kode program di atas berjalan sebagaimana mestinya.

Sekarang saya akan ubah hak akses property `$merek` dan `$stok` menjadi `private`. Efeknya, kita tidak bisa lagi mengakses kedua property menggunakan kode sebelumnya (error). Jalan satunya adalah membuat method khusus untuk mengisi dan menampilkan property ini:

50.getter_and_setter.php

```

1 <?php
2 class Produk {
3     private $merek;
4     private $stok;
5
6     public function setMerek($merek){
7         $this->merek = $merek;
8     }
9
10    public function setStok($stok){
11        $this->stok = $stok;
12    }
13
14    public function getMerek(){
15        return $this->merek;
16    }
17

```

```

18  public function getStok(){
19      return $this->stok;
20  }
21 }
22
23 $produk01 = new Produk();
24 $produk01->setMerek("Acer");
25 $produk01->setStok(10);
26
27 echo $produk01->getMerek();      // Acer
28 echo "<br>";
29 echo $produk01->getStok();      // 10

```

Kode program menjadi lebih panjang karena saya butuh 4 method untuk mengakses property \$merek dan \$stok, yakni 2 property untuk mengisi nilai dan 2 property untuk mengakses nilai.

Untuk mengisi nilai, tersedia method `setMerek()` dan `setStok()`. Kedua method ini butuh 1 argument berupa nilai yang akan diinput ke dalam setiap property. Sebenarnya nama method juga tidak harus memiliki awalan "set", bisa saja ditulis sebagai `inputMerek()` dan `inputStok()`.

Hanya saja sudah kebiasaan umum banyak programmer bahwa method yang dipakai untuk proses input property menggunakan awalan "set" yang kemudian diikuti dengan nama property-nya.

Kode program untuk method `setMerek()` dan `setStok()` juga sangat sederhana, yakni ambil nilai argument lalu input ke dalam `$this->merek` dan `$this->stok`.

Untuk mengakses nilai, saya membuat method `getMerek()` dan `getStok()`. Kembali, nama ini juga hanya kebiasaan saja, anda boleh memakai nama lain seperti `ambilMerek()` dan `ambilStok()`. Isi kedua method ini hanya perintah `return $this->merek` dan `return $this->stok`.

Setelah proses instansiasi di baris 23, saya men-set nilai property `$merek` dengan perintah `$produk01->setMerek("Acer")`, serta men-set nilai property `$stok` dengan perintah `$produk01->setStok(10)`. Hasilnya, property `$merek` akan berisi "Acer" dan property `$stok` akan berisi angka 10.

Pertanyaannya, kenapa harus repot-repot seperti ini? Bukankah kalau di-set sebagai public kodennya akan lebih singkat?

Yup, betul. Malah kalau method set dan get-nya seperti di atas, tidak ada peningkatan apa-apa dibandingkan dengan tanpa **set** dan **get**. Kode program juga di eksekusi sedikit lebih cepat jika diakses langsung (tanpa set dan get). **Setter** dan **getter** baru berguna jika kita melalukan "sesuatu" di dalam method tersebut.

Agar bisa memahami situasinya, bayangkan anda ditugaskan untuk membuat class `Produk`, namun yang akan memakai class ini adalah anggota tim (bukan anda sendiri). Jika situasinya seperti itu, kita harus "memproteksi" class `Produk` agar hasil yang ditampilkan tetap sesuai.

Bayangkan jika ada yang memberikan nilai seperti ini:

```
$produk01->setStok('Sepuluh');
```

Property `$stok` di rancang untuk menampung nilai stok barang. Nantinya di dalam class `Produk` bisa saja terdapat method `tambahStok()` dan `kurangiStok()` yang dipakai untuk memproses stok. Akan tetapi sekarang yang diinput ke dalam property `$stok` adalah string '`Sepuluh`', ini akan menjadi masalah karena nilai yang kita harapkan adalah integer, tidak bisa yang lain.

Bagaimana cara mengatasinya? Kita butuh sebuah **proses validasi**. Jika ada yang ingin mengubah isi property stok, harus diperiksa terlebih dahulu apakah itu integer (angka bulat) atau tipe data yang lain. Di dalam method `setStok()`, saya bisa tulis sebagai berikut:

51.setter_validation.php

```

1  <?php
2  class Produk {
3      private $stok = 0;
4
5      public function setStok($stok){
6          if (is_int($stok)) {
7              $this->stok = $stok;
8          }
9          else {
10              echo "Error: stok harus angka bulat <br>";
11          }
12      }
13
14      public function getStok(){
15          return $this->stok;
16      }
17  }
18
19 $produk01 = new Produk();
20 echo $produk01->getStok();           // 0
21 echo "<br>";
22
23 $produk01->setStok(10.5);          // Error: stok harus angka bulat
24 echo $produk01->getStok();          // 0
25 echo "<br>";
26
27 $produk01->setStok("Satu");        // Error: stok harus angka bulat
28 echo $produk01->getStok();          // 0
29 echo "<br>";
30
31 $produk01->setStok(10);
32 echo $produk01->getStok();          // 10

```

Di dalam method `setStok()` saya membuat sebuah proses validasi sederhana. Isi parameter `$stok` akan diperiksa terlebih dahulu dalam kondisi if. Jika isinya berupa angka bulat, fungsi

`is_int($stok)` akan menghasilkan nilai **true**, sehingga proses input bisa dilakukan. Namun jika ternyata hasilnya **false** (yang artinya isi parameter `$stok` bukan berisi angka integer), tampilkan pesan kesalahan `Error: stok harus angka bulat.`

Saya mencoba menginput nilai 10.5 di baris 23 (tipe data `float`) dan "Satu" di baris 27 (tipe data `string`). Hasilnya akan tampil pesan error.

Inilah alasan kita membuat property `$stok` menjadi `private`, karena hanya di method seperti `setStok()` validasi seperti ini bisa dilakukan. Hal yang sama juga bisa diterapkan ke property `$merek` yang seharusnya hanya bisa menerima tipe data `string` saja.

Bagaimana dengan **getter**? **Getter** adalah method yang dipakai untuk menampilkan nilai property. Dibandingkan setter, getter tidak sepenting setter karena kita hanya menampilkan nilai saja. Namun karena semua property sudah di set sebagai `private`, mau tidak mau harus dibuat sebuah method untuk menampilkan nilai property ini.

Selain itu ada kondisi dimana getter akan berguna daripada di akses langsung dari property. Sebagai ilustrasi, jika property `$merek` di set sebagai `public`, untuk menampilkan isinya, saya bisa menggunakan kode berikut:

```
echo $produk01->merek;      // Acer
```

Bayangkan kita sudah menggunakan cara ini di puluhan tempat sepanjang kode program (puluhan instansiasi object `Produk`). Kemudian tiba-tiba ada keputusan bahwa semua merek harus tampil dalam huruf besar. Cara yang mungkin dipakai adalah mencari satu-satu kode di atas, lalu tambahkan function `strtoupper()`, cara yang cukup merepotkan.

Namun apabila ketika menampilkan property `$merek` kita sudah memakai method `getMerek()` seperti berikut:

```
echo $produk01->getMerek();      // Acer
```

Maka tinggal ubah dari dalam class `Produk` saja:

```
1 public function getMerek(){
2     return strtoupper($this->merek);
3 }
```

Daripada menggunakan `return $this->merek`, saya bisa memproses isi property `$merek` ke dalam function `strtoupper()` terlebih dahulu. Hasilnya, semua merek akan tampil dalam huruf besar.

Inilah manfaat menggunakan getter daripada mengakses property secara langsung. Dengan getter, kita bisa memproses nilai akhir yang akan ditampilkan.

Berikut kode program lengkap yang memproses setter dan getter untuk property `$merek`:

52.getter_processing.php

```

1  <?php
2  class Produk {
3      private $merek = "";
4
5      public function setMerek($merek){
6          if (is_string($merek)) {
7              $this->merek = $merek;
8          }
9          else {
10              echo "Error: merek harus berbentuk string <br>";
11          }
12      }
13
14      public function getMerek(){
15          return strtoupper($this->merek);
16      }
17  }
18
19 $produk01 = new Produk();
20 $produk01->setMerek(9);           // Error: merek harus berbentuk string
21
22 $produk01->setMerek("Acer");
23 echo $produk01->getMerek();      // ACER

```

Dalam kode saya membuat proses validasi untuk method `setMerek()`. Caranya kurang lebih sama seperti method `setStok()`, hanya saja sekarang kondisi yang dipakai adalah `if (is_string($merek))`. Kondisi ini menghasilkan nilai `true` hanya jika parameter `$merek` berisi tipe data string.

Untuk method `getMerek()`, ditambah function `strtoupper($this->merek)` agar string di konversi menjadi huruf besar. Jika property `$merek` berisi string "Acer", hasil dari `strtoupper()` akan menjadi "ACER".

Setelah proses instansiasi di baris 20, saya coba men-set `$merek` dengan angka 9, hasilnya akan tampil pesan error karena yang diisi bukan tipe data string. Pada saat diisi dengan string "Acer" di baris 22, barulah nilai ini akan di set oleh method `setMerek()`.

Setter, Getter dan Constructor

Dalam contoh-contoh sebelumnya saya men-set property class `Produk` menggunakan method `setMerek()` dan `setStok()`, padahal ada cara yang lebih baik yakni menggunakan **constructor**.

Meskipun menggunakan constructor, tetap saja validasi menjadi poin penting untuk menghindari hal yang tidak diinginkan. Selain itu kita juga bisa "melewatkannya" method setter melalui constructor seperti contoh berikut:

53.setter_getter_constructor.php

```

1  <?php

```

```

2  class Produk {
3      private $merek = "";
4      private $stok = 0;
5
6      private function setMerek($merek){
7          if (is_string($merek)) {
8              $this->merek = $merek;
9          }
10         else {
11             die("Error: merek harus berbentuk string <br>");
12         }
13     }
14
15     private function setStok($stok){
16         if (is_int($stok)) {
17             $this->stok = $stok;
18         }
19         else {
20             die("Error: stok harus angka bulat <br>");
21         }
22     }
23
24     public function __construct($merek, $stok){
25         $this->setMerek($merek);
26         $this->setStok($stok);
27     }
28
29     public function getMerek(){
30         return strtoupper($this->merek);
31     }
32
33     public function getStok(){
34         return $this->stok;
35     }
36 }
37
38 $produk01 = new Produk("Acer",10);
39 echo "Stok produk ".$produk01->getMerek()." : ".$produk01->getStok();

```

Hasil kode program:

Stok produk ACER: 10

Di sini saya menggabungkan validasi property \$merek dan \$stok ke dalam class Produk. Selain itu ada sedikit perubahan.

Pertama, method setMerek() dan setStok() saya set sebagai private, yang artinya tidak bisa diakses dari luar class. Kedua method ini nantinya akan diakses secara internal dari dalam constructor.

Kedua, pesan error jika tipe data tidak sesuai sekarang menggunakan function die(), bukan lagi echo. Alasannya adalah supaya program langsung berhenti jika tipe data tidak sesuai.

Sebenarnya terdapat alternatif yang lebih baik daripada die(), yakni menggunakan **exception** dan blok **try-catch**. Materi ini akan kita pelajari pada bahasan terpisah.

Method untuk constructor berada di baris 24 – 27. Di sini saya melewatkkan argument \$merek dan \$stok ke dalam method setMerek() dan setStok(). Proses input nilai akan ditangani oleh masing-masing method ini.

Untuk method getMerek() dan getStok() tidak ada yang baru, dimana saya menggunakan function strtoupper() untuk property \$merek dan tidak langsung mengembalikan nilai untuk property \$stok.

Karena sudah memiliki constructor, proses input nilai bisa dilakukan langsung pada saat instansiasi di baris 38. Di baris 39 saya mengakses method getMerek() dan getStok() untuk memastikan hasilnya sudah sesuai.

Sekarang mari kita test dengan tipe data yang salah:

54.setter_getter_constructor_error.php

```
1 <?php
2 class Produk {
3     ...
4     ...
5 }
6
7 $produk01 = new Produk("Acer", '5');
8 echo "Stok produk ".$produk01->getMerek()." : ".$produk01->getStok();
```

Hasil kode program:

Error: stok harus angka bulat

Dalam percobaan ini saya memberi string '5' untuk nilai awal \$stok, hasilnya akan tampil pesan error karena tipe data stok harus angka bulat (integer).

Berikut percobaan untuk property \$merek:

55.setter_getter_constructor_error_2.php

```
1 <?php
2 class Produk {
3     ...
4     ...
5 }
6
7 $produk01 = new Produk(0, '5');
8 echo "Stok produk ".$produk01->getMerek()." : ".$produk01->getStok();
```

Hasil kode program:

Error: merek harus berbentuk string

Saya memberikan angka 0 sebagai argument pertama constructor, hasilnya tampil pesan error karena property \$merek hanya bisa menerima tipe data string saja.

Exercise

Kali ini saya menantang anda untuk memodifikasi class Produk dengan skenario berikut:

Class Produk akan memiliki 2 property, yakni \$sku dan \$stok yang di set sebagai private. Buatlah method setter dan getter untuk kedua property ini dengan aturan sebagai berikut:

- ◆ Property \$sku hanya bisa diisi kode barang dalam format "AAA000", dimana 3 karakter awal berupa alphabet dalam huruf besar, kemudian diikuti oleh 3 karakter angka 0 – 9. Jika format yang diinput tidak sesuai, tampilkan pesan error.
- ◆ Property \$stok hanya bisa menerima angka integer dan harus positif. Jika diinput integer negatif atau 0, tampilkan pesan error.
- ◆ Kedua setter di set sebagai private dan dijalankan dari constructor.
- ◆ Getter untuk \$sku dan \$stok tidak perlu pemrosesan, cukup kembalikan nilai saja.

Berikut contoh instansiasi class Produk dan hasil yang diharapkan:

```

1 <?php
2 class Produk {
3     ...
4     ...
5 }
6
7 $produk01 = new Produk('ACR014',9);
8 echo "Stok produk ".$produk01->getSku()." : ".$produk01->getStok()." buah";
9 // Stok produk ACR014: 9 buah
10
11 echo "<br>";
12
13 $produk02 = new Produk('LNV023',100);
14 echo "Stok produk ".$produk02->getSku()." : ".$produk02->getStok()." buah";
15 // Stok produk LNV023: 100 buah
16
17 echo "<br>";
18
19 $produk03 = new Produk('2NV050',67);
20 echo "Stok produk ".$produk03->getSku()." : ".$produk03->getStok()." buah";
21 // Error: sku harus 6 digit (3 huruf dan 3 angka), seperti AAA001
22
23 echo "<br>";
24
25 $produk04 = new Produk('HP002',10);
26 echo "Stok produk ".$produk04->getSku()." : ".$produk04->getStok()." buah";
27 // Error: sku harus 6 digit (3 huruf dan 3 angka), seperti AAA001

```

```

28
29 echo "<br>";
30
31 $produk05 = new Produk('DEL099', -5);
32 echo "Stok produk ".$produk05->getSku()." : ".$produk05->getStok()." buah";
33 // Error: stok harus angka bulat positif

```

Silahkan anda pelajari hasil pemanggilan constructor class Produk dan rancang seperti apa kira-kira kode programnya.

Tips: Untuk mengecek format \$sku, saya menggunakan **regular expression**, yakni fungsi preg_match(). Fungsi ini akan mengembalikan nilai **true** jika pola sesuai, dan **false** jika pola tidak sesuai. Anda mungkin butuh membuka kembali buku **PHP Uncover** di bab tentang form processing untuk mengingat kembali cara penggunaan fungsi preg_match().

Baik, berikut kode program yang saya pakai untuk membuat class Produk:

```

1 <?php
2 class Produk {
3     private $sku = "";
4     private $stok = 0;
5
6     private function setSku($sku){
7         if (preg_match("/^A-Z{3}[0-9]{3}$/", $sku)) {
8             $this->sku = $sku;
9         }
10        else {
11            die("Error: sku harus 6 digit (3 huruf dan 3 angka), seperti AAA001");
12        }
13    }
14
15    private function setStok($stok){
16        if (is_int($stok) && ($stok>0)) {
17            $this->stok = $stok;
18        }
19        else {
20            die("Error: stok harus angka bulat positif <br>");
21        }
22    }
23
24    public function __construct($sku, $stok){
25        $this->setSku($sku);
26        $this->setStok($stok);
27    }
28
29    public function getSku(){
30        return $this->sku;
31    }
32
33    public function getStok(){

```

```

34     return $this->stok;
35 }
36 }
```

Fokus utama kita ada di baris 7 dan 16, yakni cara memeriksa apakah parameter `$sku` dan `$stok` sesuai dengan syarat atau tidak.

Fungsi `preg_match("/^[A-Z]{3}[0-9]{3}$/, $sku)` artinya saya ingin memeriksa apakah isi parameter `$sku` cocok dengan pola regular expression `"/^[A-Z]{3}[0-9]{3}$/"`. Pola ini bisa dibaca: "Awali dengan huruf A-Z sebanyak tiga digit, lalu akhiri dengan digit 0-9 sebanyak tiga digit".

Sedangkan untuk `$stok`, saya mengombinasikan 2 buah kondisi, yakni `if (is_int($stok) && ($stok>0))`. Hasilnya hanya akan `true` jika parameter `$stok` berisi tipe data integer dan harus lebih besar dari nol.

Method setter dan getter yang kita pelajari di sini bisa dikembangkan lebih lanjut tidak hanya untuk mengecek tipe data saja.

Misalkan untuk property `$sku`, bisa saja method setter akan mengakses ke database untuk memeriksa apakah sudah ada produk dengan nomor SKU yang sama. Jika ada, tampilkan pesan error dan minta user menginput nomor SKU yang lain.

Intinya, sedapat mungkin jangan buat property sebagai `public`, tapi set menjadi `private` agar kita bisa "melindungi" nilainya. Kecuali untuk kasus-kasus tertentu seperti pengajaran, tugas kampus, atau kita yakin tidak akan ada orang lain yang menggunakan class ini.

Dalam beberapa kesempatan saya masih men-set property sebagai `public` semata-mata agar contoh kode program kita tidak terlalu panjang. Karena jika di set ke `private`, harus dibuat method setter dan getter untuk mengakses property tersebut.

2.10. Static Property dan Static Method

Di awal pembahasan buku ini saya menjelaskan bahwa seluruh property dan method hanya bisa diakses dari objek. Namun *static property* dan *static method* merupakan pengecualianya.

Static property dan **static method** adalah property dan method yang **melekat kepada class**, bukan kepada objek. Konsep static ini memang sepertinya 'agak keluar' dari logika pemrograman berorientasi objek, karena seharusnya semua property dan method diakses dari object, bukan langsung dari class.

Namun ada beberapa situasi khusus dimana *static property* dan *static method* lebih cocok dipakai daripada property dan method biasa. Sebelum sampai ke sana, kita akan lihat terlebih dahulu seperti apa cara pemakaian *static property* dan *static method* ini.

Static Property

Static property adalah property milik class, sehingga untuk mengaksesnya kita tidak perlu membuat instansiasi object, tapi langsung dipanggil dengan menulis nama classnya.

Berikut contoh kode program untuk membuat dan mengakses static property:

57.static_property_basic.php

```
1 <?php
2 class Produk {
3     public static $totalProduk = 100;
4 }
5
6 echo Produk::$totalProduk; // 100
```

Untuk membuat static property, tambahkan keyword **static** setelah penulisan *access modifier*. Di baris 3 saya menulis `public static $totalProduk`, artinya `$totalProduk` adalah sebuah **property static** dengan hak akses **public**.

Cara mengisi static property tidak berbeda dengan property biasa. Dalam contoh di atas, property `$totalProduk` saya isi dengan angka 100.

Karena static property tidak "melekat" kepada object, maka kita tidak perlu melakukan proses instansiasi. Cara mengaksesnya adalah dengan menulis nama class, titik dua (dua kali), dan tulis nama property-nya. Dengan demikian untuk mengakses static property `$totalProduk` milik class `Produk`, perintahnya: `Produk::$totalProduk`. Perhatikan untuk nama property diawali dengan tanda \$.

Bagaimana jika kita tetap buat object dari class `Produk` dan mengaksesnya seperti property biasa? Mari kita coba:

58.static_property_basic_object.php

```
1 <?php
2 class Produk {
3     public static $totalProduk = 100;
4 }
5
6 $produk01 = new Produk();
7 echo $produk01->totalProduk;
```

Di baris 6 saya membuat object `$produk01` dari class `Produk` (proses instansiasi). Kemudian property `$totalProduk` di akses menggunakan cara biasa, yakni `$produk01->totalProduk`. Bagaimana hasilnya?

Notice: Accessing static property `Produk::$totalProduk` as non static

Tampil pesan error yang menyebutkan bahwa kita tidak bisa mengakses property `Produk::$totalProduk` menggunakan cara biasa (non static). Karena di sini property `$totalProduk`

bukanlah milik object `$produk01`, tapi milik class `Produk`.

Percobaan berikutnya, bagaimana jika class `Produk` kita turunkan? Apakah property `$totalProduk` masih bisa diakses? Berikut hasilnya:

59.static_property_extends.php

```

1 <?php
2 class Produk {
3     public static $totalProduk = 100;
4 }
5
6 class Blender extends Produk {
7 }
8
9 echo Blender::$totalProduk; // 100

```

Saya menurunkan class `Produk` ke dalam class `Blender`. Class `Blender` sendiri tidak berisi kode apapun. Kemudian mengakses property `$totalProduk` melalui class `Blender` (baris 9), hasilnya tetap bisa diproses. Hal ini memperlihatkan bahwa static property juga akan diturunkan dari *parent class* ke *child class*.

Static Method

Sama seperti static property, **static method** adalah method yang dimiliki oleh class, bukan object. Cara penulisan dan pengaksesannya juga sama seperti static property:

60.static_method_basic.php

```

1 <?php
2 class Produk {
3     public static $totalProduk = 100;
4
5     public static function cekProduk(){
6         return "Total Produk ada 100";
7     }
8 }
9
10 echo Produk::cekProduk(); // Total Produk ada 100

```

Di dalam class `Produk` saya membuat `public static function cekProduk()`. Maksudnya ini adalah method static yang memiliki hak akses *public*. Isi dari method `cekProduk()` hanya 1 baris, yakni mengembalikan string `"Total Produk ada 100"`.

Untuk mengakses static method, caranya juga sama seperti static property, yakni tulis nama class, diikuti dengan titik dua (dua kali), kemudian nama method. Sehingga untuk mengakses mehtod static `cekProduk()` milik class `Produk`, penulisannya adalah `Produk::cekProduk()`.

Method `cekProduk()` sebenarnya saya rancang sebagai **getter** untuk property `$totalProduk`. Jadi seharusnya di dalam method ini saya mengakses isi property `$totalProduk`, bukan

langsung menulis jumlah produk 100 ke dalam string.

Tapi bagaimana caranya? Kita tidak bisa memakai `$this->totalProduk` karena `$totalProduk` adalah static property, *pseudo-variable* `$this` merujuk ke object, bukan class. Untuk keperluan ini PHP menyediakan keyword **self**, berikut contohnya:

61.static_method_self.php

```

1 <?php
2 class Produk {
3     public static $totalProduk = 100;
4
5     public static function cekProduk(){
6         return "Total Produk ada ".self::$totalProduk;
7     }
8 }
9
10 echo Produk::cekProduk(); // Total Produk ada 100

```

Perhatikan perintah di baris 6, saya menggabungkan string "Total Produk ada " dengan `self::$totalProduk`. Dalam PHP, `self` adalah keyword khusus yang merujuk ke **class saat ini**.

Salah satu konsep pemrograman object yang sering membuat bingung adalah: apa beda `$this` dengan `self`? Ini hanya bisa dijawab jika anda sudah benar-benar paham tentang OOP. Di dalam PHP, `$this` merujuk ke **object** hasil instansiasi, sedangkan `self` merujuk ke **class**. Keduanya sering dipakai di dalam method.

Alternatif cara lain adalah dengan menulis nama class di dalam method:

62.static_method_name.php

```

1 <?php
2 class Produk {
3     private static $totalProduk = 100;
4
5     public static function cekProduk(){
6         return "Total Produk ada ".Produk::$totalProduk;
7     }
8 }
9
10 echo Produk::cekProduk(); // Total Produk ada 100

```

Di sini saya mengganti `self::` dengan `Produk::` (baris 6), pemanggilan seperti ini diperbolehkan di dalam PHP.

Sama seperti static property, static method juga diturunkan ke child class:

63.static_method_extends.php

```

1 <?php
2 class Produk {
3     private static $totalProduk = 100;

```

```

4
5     public static function cekProduk(){
6         return "Total Produk ada ".self::$totalProduk;
7     }
8 }
9
10 class Blender extends Produk {
11 }
12
13 echo Blender::cekProduk(); // Total Produk ada 100

```

Saya membuat class `Blender` sebagai child class dari `Produk`. Hasilnya, kita tetap bisa mengakses static method `cekProduk()` dari class `Blender`.

Percobaan selanjutnya, bagaimana cara mengakses static method dan static property milik parent class dari dalam method child class? Apakah memakai keyword `parent`, `self`, atau nama class-nya langsung? Berikut prakteknya:

64.static_method_extends_self_parent.php

```

1 <?php
2 class Produk {
3     private static $totalProduk = 100;
4
5     public static function cekProduk(){
6         return "Total Produk ada ".self::$totalProduk;
7     }
8 }
9
10 class Blender extends Produk {
11     public function cekBlender(){
12         return self::cekProduk(). ', termasuk 3 jenis Blender <br>';
13     }
14 }
15
16 class HairDryer extends Produk {
17     public function cekHairDryer(){
18         return parent::cekProduk(). ', termasuk 5 jenis Hair Dryer <br>';
19     }
20 }
21
22 class Mixer extends Produk {
23     public function cekMixer(){
24         return Produk::cekProduk(). ', termasuk 2 jenis Mixer <br>';
25     }
26 }
27
28 $produk01 = new Blender();
29 echo $produk01->cekBlender();
30
31 $produk02 = new HairDryer();
32 echo $produk02->cekHairDryer();
33

```

```
34 $produk03 = new Mixer();
35 echo $produk03->cekMixer();
```

Hasil kode program:

```
Total Produk ada 100, termasuk 3 jenis Blender
Total Produk ada 100, termasuk 5 jenis Hair Dryer
Total Produk ada 100, termasuk 2 jenis Mixer
```

Di awal kode program saya membuat class **Produk** seperti contoh sebelumnya. Isinya juga tidak ada perubahan, yakni property static **\$totalProduk** dan method static **cekProduk()**. Class **Produk** kemudian saya turunkan ke 3 buah class lain, yakni class **Blender**, class **HairDryer** dan class **Mixer**.

Di dalam setiap child class saya membuat sebuah method, isinya mengakses method static **cekProduk()** milik parent class (**Produk**), yang kemudian ditambah dengan sedikit string sebagai pembeda.

Perhatikan cara akses static method parent class di setiap child class, saya menggunakan keyword **self** di baris 12 , **parent** di baris 18, dan nama class **Produk** di baris 24. Semua cara ini bisa dipakai di dalam PHP. Perhatikan juga bahwa method di child class ini adalah method biasa, bukan static method. Ini juga memperlihatkan bahwa kita bisa mengakses method static dari dalam method biasa.

Setelah pendefenisian ke-4 class, saya membuat 3 object dari class **Blender**, **HairDryer** dan **Mixer** lalu mengakses setiap method dari masing-masing class. Hasilnya akan tampil string "Total Produk ada 100" milik class **Produk**, lalu disambung dengan string dari setiap class.

Static Increment

Dari pembahasan sebelumnya kita sudah melihat bahwa static property dan static method adalah kepunyaan class, bukan object seperti property dan method biasa. Selain itu, ada efek lain yang hanya bisa dimiliki oleh static property. Perhatikan kode berikut:

65.static_method_increment.php

```
1 <?php
2 class Produk {
3     private static $totalProduk = 0;
4
5     public function __construct(){
6         self::$totalProduk++;
7         echo "class Produk dibuat, total produk = ".self::$totalProduk."<br>";
8     }
9 }
10
11 $produkA = new Produk();
12 $produkB = new Produk();
13 $produkC = new Produk();
```

```
14 $produkD = new Produk();
```

Di dalam class Produk saya membuat static property `$totalProduk`. Property ini kemudian diinisialisasi dengan nilai 0.

Selanjutnya terdapat constructor. Di baris pertama constructor saya menjalankan operasi increment untuk property `$totalProduk` dengan perintah `self::$totalProduk++`. Ini sama artinya jika ditulis sebagai `self::$totalProduk = self::$totalProduk + 1`. Perintah ini akan menaikkan isi property `$totalProduk` sebanyak 1 angka. Di baris kedua constructor saya menampilkan gabungan string "class Produk dibuat, total produk = " dengan isi property `self::$totalProduk`.

Setelah pendefinisian class Produk, pada baris 11 – 14 saya membuat 4 object. Bisakah anda menebak seperti apa hasilnya? Setiap kali object Produk di buat, constructor otomatis akan dijalankan, sehingga terdapat 4 kali perintah echo:

```
class Produk dibuat, total produk = 1
class Produk dibuat, total produk = 2
class Produk dibuat, total produk = 3
class Produk dibuat, total produk = 4
```

Karena `$totalProduk` adalah static property, proses inisialisasi `$totalProduk = 0` hanya akan dijalankan 1 kali (di awal kode program saja). Setiap kali class Produk di instansiasi, perintah `self::$totalProduk++` akan dijalankan. Inilah yang membuat nilai `$totalProduk` terus naik dari 1, 2, 3 hingga 4.

Bagaimana jika `$totalProduk` kita ubah menjadi property biasa? Berikut hasilnya:

66.tanpa_static_method.php

```
1 <?php
2 class Produk {
3     private $totalProduk = 0;
4
5     public function __construct(){
6         $this->totalProduk++;
7         echo "class Produk dibuat, total produk = ".$this->totalProduk."<br>";
8     }
9 }
10
11 $produkA = new Produk();
12 $produkB = new Produk();
13 $produkC = new Produk();
14 $produkD = new Produk();
```

Hasil kode program:

```
class Produk dibuat, total produk = 1
class Produk dibuat, total produk = 1
class Produk dibuat, total produk = 1
```

```
class Produk dibuat, total produk = 1
```

Bisakah anda jelaskan kenapa property `$totalProduk` nilainya sama untuk setiap class (berisi angka 1)?

Alasannya adalah karena perintah `$totalProduk = 0` selalu dijalankan setiap kali proses instansiasi terjadi. Selain itu akan terdapat 4 buah property `$totalProduk`, yakni satu untuk setiap object. Ini berbeda jika `$totalProduk` di set sebagai static, yang artinya hanya ada 1 property `$totalProduk` kepunyaan class `Produk` yang dipakai bersama oleh ke-4 object.

Kode seperti ini cocok dipakai jika kita ingin membuat semacam counter untuk menghitung jumlah instansiasi class `Produk`. Jika terdapat 87 kali pembuatan class `Produk`, maka static property `$totalProduk` juga akan berisi 87. Inilah salah satu penerapan dari static property dalam *object oriented programming*.

Helper Class

Penerapan lain dari static method adalah untuk membuat sebuah **helper class** atau class bantu yang memang kurang cocok jika di instansiasi menjadi object. Class helper ini biasanya berisi kumpulan method untuk keperluan umum sehingga sering juga disebut sebagai **utility class**.

Sebagai contoh saya ingin membuat method `cekValidSKU()`. Method ini berfungsi untuk memeriksa apakah sebuah string membentuk kode SKU yang valid atau tidak. Misalnya dalam program saya, kode SKU yang valid harus terdiri dari 6 digit (3 huruf besar + 3 angka), seperti "AAA001", atau "XYZ001". Method `cekValidSKU()` akan mengembalikan nilai **true** jika kode SKU sesuai dan mengembalikan nilai **false** jika tidak sesuai.

Method `cekValidSKU()` ini kurang cocok jika di tempatkan ke dalam class `Produk` karena tidak secara langsung diperlukan untuk setiap produk.

Contoh lain, saya ingin membuat method `cekValidMerek()` untuk memeriksa apakah sebuah merek sudah ada di dalam database atau tidak. Jika merek "Samsung" sudah ada, method `cekValidMerek("Samsung")` akan mengembalikan nilai **true**. Atau `cekValidMerek("Asus")` akan mengembalikan nilai **false** jika merek "Asus" belum tersedia di dalam database.

Method `cekValidSKU()` dan `cekValidMerek()` terhitung sebagai method bantu yang kurang sesuai jika dimasukkan ke dalam class `Produk`. Untuk hal ini, kita bisa membuat class `ProdukHelper` yang berisi method-method tersebut.

Class `ProdukHelper` sendiri juga kurang pas jika dibuat sebagai object, karena yang kita butuhkan cuma isi methodnya saja. Sehingga setiap method bisa di set sebagai static agar bisa diakses tanpa perlu membuat instansiasi. Dalam kasus seperti ini, class `ProdukHelper` hanya berfungsi sebagai "container" atau penampung dari method-method bantu.

Berikut contoh kode programnya:

67.static_method_class_helper.php

```

1 <?php
2 class ProdukHelper {
3     public static function cekValidSKU($sku){
4         return preg_match("/^A-Z{3}[0-9]{3}$/, $sku);
5     }
6
7     public static function cekValidMerek($merek){
8         $semuaMerek=[ "Samsung", "LG", "Sony", "Philips", "Sharp", "Sanken"];
9         return in_array($merek,$semuaMerek);
10    }
11 }
12
13 if (ProdukHelper::cekValidSKU("AAA545")) {
14     echo "SKU AAA545 valid <br>";
15 }
16
17 if (ProdukHelper::cekValidSKU("AAa545")) {
18     echo "SKU AAa545 valid <br>";
19 }
20
21 if (ProdukHelper::cekValidMerek("Sharp")) {
22     echo "Merek Sharp tersedia <br>";
23 }
24
25 if (ProdukHelper::cekValidMerek("Samsung")) {
26     echo "Merek Samsung tersedia <br>";
27 }
28
29 if (ProdukHelper::cekValidMerek("Asus")) {
30     echo "Merek Asus tersedia <br>";
31 }

```

Hasil kode program:

```

SKU AAA545 valid
Merek Sharp tersedia
Merek Samsung tersedia

```

Seperti penjelasan sebelumnya, saya membuat class ProdukHelper dengan 2 buah method: `cekValidSKU()` dan `cekValidMerek()`.

Method `cekValidSKU()` memiliki 1 parameter `$sku`, yang akan saya cek menggunakan fungsi regular expression `preg_match("/^A-Z{3}[0-9]{3}$/, $sku)`. Fungsi `preg_match()` sendiri sudah langsung mengembalikan nilai **true** jika pola sesuai, atau **false** jika pola tidak sesuai. Dengan demikian saya tinggal me-return kembalian fungsi ini.

Method `cekValidMerek()` juga memiliki 1 parameter, yakni `$merek`. Di dalam method ini saya membuat array `$semuaMerek` yang berisi daftar merek. Dalam praktek yang sebenarnya, daftar merek ini bisa berasal dari database. Kemudian saya mengembalikan hasil fungsi `in_array($merek, $semuaMerek)`.

Fungsi `in_array()` adalah function bawaan PHP yang akan mengembalikan nilai `true` jika argument pertama ada di dalam array argument kedua. Karena nantinya argument pertama `$merek` akan diisi dengan string seperti `"Samsung"`, maka pemanggilan fungsi ini akan menjadi `in_array("Samsung", $semuaMerek)`, hasilnya `true` karena string `"Samsung"` ada di daftar array `$semuaMerek`.

Method `cekValidSKU()` dan `cekValidMerek()` saya set sebagai static, sehingga bisa diakses langsung dari class `ProdukHelper`.

Di baris 13 sampai 31 saya menguji penggunaan method-method ini. Jika hasilnya `true`, maka jalankan perintah `echo`. Jika hasilnya `false`, perintah `echo` tidak akan dijalankan. Inilah contoh penggunaan static method yang sering dipakai.

Teknik helper class seperti ini mengundang cukup banyak diskusi. Ada pendapat bahwa setiap method seharusnya melekat kepada object. Jika butuh helper class, berarti ada yang salah di design program yang dibuat.

Namun bahasa Java sendiri (yang sudah full OOP) juga memiliki helper class seperti `java.util.Arrays` dan `java.lang.Math`. Keduanya berisi berbagai static property dan static method untuk memproses array dan fungsi matematika di bahasa Java.

2.11. Class Constant

Class constant adalah sebuah konstanta yang terdapat di dalam class. Saya yakin anda sudah paham tentang perbedaan konstanta dan variabel. Keduanya sama-sama dipakai untuk menyimpan data, hanya saja konstanta bersifat tetap dan tidak bisa diubah sepanjang kode program.

Untuk membuat konstanta di dalam class, awali penulisannya dengan keyword `const` seperti contoh berikut:

```
1 <?php
2 class Produk {
3     public const KURSUSD = 15000;
4 }
```

Di sini saya membuat konstanta `KURSUSD` yang diisi dengan nilai 15000. Kebiasaan banyak programmer adalah menulis nama konstanta dalam huruf besar, tujuannya supaya mudah dibedakan dengan property atau variabel biasa. Ini bukanlah sebuah keharusan, bisa saja kita membuat konstanta dengan huruf kecil seperti `kursusd` atau `kursUSD`.

Tambahan *access modifier* seperti `public` baru didukung oleh PHP 5.3 ke atas. Untuk versi PHP di bawah itu, class constant tidak bisa diberikan hak akses dan akan menampilkan pesan error.

Class constant bisa diakses menggunakan nama class (seperti static method). Berikut

contohnya:

```
68.class_constant.php

1 <?php
2 class Produk {
3     public const KURSUSD = 15000;
4 }
5
6 echo Produk::KURSUSD;           // 15000
```

Di baris 6 saya mengakses konstanta KURSUSD yang caranya mirip seperti static method, yakni menggunakan scope resolution operator " :: ".

Bagaimana jika diakses dari object? Berikut percobaannya:

```
68a.class_constant_object.php
```

```
1 <?php
2 class Produk {
3     public const KURSUSD = 15000;
4 }
5
6 $produk01 = new Produk();
7 echo $produk01->KURSUSD;    // Warning: Undefined property: Produk::$KURSUSD
```

Ternyata tidak bisa. Ini memperlihatkan bahwa sama seperti static property, konstanta juga "melekat" ke dalam class, bukan ke object.

Jika hak akses konstanta diubah menjadi private, kita tidak bisa mengakses nilainya dari luar class:

```
69.class_constant_private.php
```

```
1 <?php
2 class Produk {
3     private const KURSUSD = 15000;
4 }
5
6 echo Produk::KURSUSD;
7 // Fatal error: Uncaught Error: Cannot access private const Produk::KURSUSD
```

Bagaimana jika konstanta ini diakses dari dalam class? Caranya sama seperti static property, yakni melalui keyword **self**:

```
70.class_constant_self.php
```

```
1 <?php
2 class Produk {
3     public $hargaUSD = 0;
4     private const KURSUSD = 15000;
5
6     public function hargaIDR(){
```

```

7     return $this->hargaUSD * self::KURSUSD;
8 }
9 }
10
11 $produk01 = new Produk();
12 $produk01->hargaUSD = 15;
13
14 echo $produk01->hargaIDR(); // 225000

```

Kali ini saya memodifikasi class Produk dengan tambahan property \$hargaUSD yang diset sebagai public, serta method hargaIDR().

Method hargaIDR() akan mengembalikan hasil perkalian property \$hargaUSD dengan konstanta KURSUSD. Karena \$hargaUSD adalah property biasa, maka di akses sebagai \$this->hargaUSD. Sedangkan untuk konstanta KURSUSD diakses sebagai self::KURSUSD.

Class Produk selanjutnya di instansiasi ke dalam object \$produk01 di baris 11. Property hargaUSD kemudian saya set dengan 15. Hasilnya, method hargaIDR() akan mengembalikan nilai 225000, yang di dapat dari $15 * 15000$.

Sebagai penutup mari kita coba mengubah isi konstanta setelah di definisikan:

71.class_constant_error.php

```

1 <?php
2 class Produk {
3     public static $kursSGD = 11000;
4     public const KURSUSD = 15000;
5 }
6
7 Produk::$kursSGD = 12000;
8 echo Produk::$kursSGD; // 12000
9
10 Produk::KURSUSD = 16000; // Parse error: syntax error, unexpected '='
11 echo Produk::KURSUSD;

```

Dalam contoh ini saya membuat static property \$kursSGD dan konstanta KURSUSD. Keduanya diberi nilai awal 11000 dan 15000. Isi static property \$kursSGD bisa saya tukar di baris 8, sedangkan konstanta KURSUSD akan menampilkan pesan error saat nilainya diubah pada baris 10.

2.12. Constructor Property Promotion

PHP 8 menghadirkan fitur baru yang disebut sebagai **constructor property promotion**. Fitur ini bisa dipakai untuk mempersingkat proses inisialisasi property yang biasa ditulis dalam constructor.

Sebagai contoh, di materi tentang constructor kita pernah bahas kode program berikut ini:

```

1 <?php
2 class Produk {
3     public $jenis;
4     public $merek;
5     public $stok;
6
7     public function __construct($jenis, $merek, $stok){
8         $this->jenis = $jenis;
9         $this->merek = $merek;
10        $this->stok = $stok;
11    }
12 }
13
14 $produk01 = new Produk("Televisi", "Samsung", 20);
15 $produk02 = new Produk("Mesin cuci", "LG", 10);
16
17 print_r ($produk01);
18 echo "<br>";
19 print_r ($produk02);

```

Hasil kode program:

```

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

```

Di sini class Produk saya definisikan dengan tiga buah property: \$jenis, \$merek dan \$stok. Kemudian proses inisialisasi ketiganya dilakukan dari dalam constructor. Jika dipanggil new Produk("Televisi", "Samsung", 20), maka string "Televisi" akan diinput ke dalam property \$jenis, string "Samsung" ke dalam \$merek, dan angka 20 ke dalam \$stok.

Pendefinisian property seperti ini sangat sering di pakai, oleh karena itu PHP 8 menyediakan cara yang lebih singkat:

72.constructor_property_promotion_1.php

```

1 <?php
2 class Produk {
3     public function __construct(public $jenis, public $merek, public $stok) {
4     }
5 }
6
7 $produk01 = new Produk("Televisi", "Samsung", 20);
8 $produk02 = new Produk("Mesin cuci", "LG", 10);
9
10 print_r ($produk01);
11 echo "<br>";
12 print_r ($produk02);

```

Di baris 3, constructor memiliki tambahan hak akses public sebelum penulisan nama parameter. Inilah yang dimaksud dengan *constructor property promotion*, dimana parameter constructor "dipromosikan" menjadi property. Parameter tersebut akan di proses PHP sama

seperti baris 3 – 10 pada kode program sebelumnya.

Dengan kata lain, perintah berikut:

```
1 class Produk {
2     public function __construct(public $jenis, public $merek, public $stok) {
3     }
4 }
```

Sama artinya dengan:

```
1 class Produk {
2     public $jenis;
3     public $merek;
4     public $stok;
5
6     public function __construct($jenis, $merek, $stok){
7         $this->jenis = $jenis;
8         $this->merek = $merek;
9         $this->stok = $stok;
10    }
11 }
```

Hak akses yang dipakai untuk penulisan constructor property promotion juga tidak harus `public`, bisa juga `private` atau `protected` tergantung kebutuhan.

Karena penulisan parameter constructor sekarang menjadi sedikit lebih panjang, bisa juga dipecah menjadi beberapa baris seperti kode berikut:

```
1 class Produk {
2     public function __construct(
3         public $jenis,
4         public $merek,
5         public $stok) {
6     }
7 }
```

Di dalam constructor, kita tetap bisa menulis fitur-fitur biasa yang ada di PHP seperti default argument, atau menambah perintah lain:

```
1 <?php
2 class Produk {
3     public function __construct(
4         public $jenis,
5         private $merek = "Maspion",
6         protected $stok = 10) {
7
8         echo "Produk $this->jenis dibuat <br>";
9     }
10 }
11
12 $produk01 = new Produk("Televisi", "Samsung", 20);
13 $produk02 = new Produk("Mesin cuci");
```

```
14  
15 print_r ($produk01);  
16 echo "<br>";  
17 print_r ($produk02);
```

Hasil kode program:

```
Produk Televisi dibuat  
Produk Mesin cuci dibuat  
Produk Object (  
    [jenis] => Televisi [merek:Produk:private] => Samsung [stok:protected] => 20 )  
Produk Object (  
    [jenis] => Mesin cuci [merek:Produk:private] => Maspion [stok:protected] => 10 )
```

Sekarang property `$merek` dan `$stok` saya isi dengan nilai awal (*default argument*). Selain itu keduanya juga memiliki hak akses `private` dan `protected`. Di dalam constructor terdapat tambahan perintah `echo` di baris 8, sehingga akan tampil setiap kali object Produk dibuat.

Saat kode program di jalankan, akan tampil teks "Produk Televisi dibuat" dan "Produk Mesin cuci dibuat". Ini berasal dari perintah instansiasi class Produk di baris 12 dan 13.

Fitur constructor property promotion cukup menarik untuk dipakai, sekedar penulisan singkat dari cara yang bisa kita gunakan.

Materi tentang constructor property promotion menutup pembahasan pertama tentang dasar pemrograman object di PHP.

Sepanjang bab ini kita telah mempelajari banyak hal terkait OOP dasar, mulai dari konsep **class** dan **object**, **property** dan **method**, **constructor** dan **destructor**, **inheritance**, **visibility**, **getter** dan **setter**, **static property** dan **static method**, **class constant**, hingga **constructor property promotion**.

Semoga materi dasar OOP PHP ini bisa anda pahami. Apabila diperlukan, boleh dibaca ulang karena berikutnya kita akan masuk ke advanced OOP PHP yang membahas materi lanjutan tentang pemrograman object di PHP.

3. Advanced OOP PHP

Bab kali ini akan membahas konsep pemrograman object yang sedikit lebih advanced daripada bab sebelumnya. Mayoritas materi yang akan kita pelajari mungkin tidak akan sering dipakai, tapi tetap saja ini merupakan bagian dari object oriented programming di PHP.

Sebagai contoh, kecil kemungkinan anda butuh **abstract class** ketika membuat website sistem informasi sekolah. Materi seperti ini baru akan terpakai jika membuat website sekelas **traveloka** (dengan tim tentunya).

Materi *advanced OOP* ini bukanlah sebuah keharusan (tidak harus dipakai), tapi lebih ke perancangan sistem. Dalam konsep OOP ada yang dinamakan "**design pattern**", yakni pola penyusunan class untuk membuat efek tertentu. Teknik *design pattern* ini juga butuh pemahaman mendalam tentang object oriented programming.

3.1. Abstract Class

Abstract class adalah class khusus yang berfungsi sebagai class dasar (*base class*) untuk class turunan. Menggunakan abstract class, kita bisa "memaksa" class turunan agar mengimplementasikan method tertentu.

Tujuan akhir dari abstract class adalah konsep OOP yang dikenal sebagai **polymorphism** (*polimorfisme*). Secara sederhana, **polymorphism** dipakai agar semua class turunan memiliki nama method yang sama meskipun method tersebut melakukan hal yang berbeda.

Sebagai analogi, semua produk elektronik memiliki tombol power yang dipakai untuk menghidupkan perangkat tersebut. Bentuk dan lokasi tombol power ini bisa berbeda-beda antar setiap perangkat, namun bisa di tebak bahwa jika ada produk elektronik baru, pasti ada tombol powernya.

Dengan *abstract class*, kita bisa membuat aturan "jika sebuah class adalah turunan dari *abstract class Produk*, maka pasti class tersebut memiliki method `cekHarga()`". *Abstract class* ini nantinya memiliki syarat khusus yang akan kita bahas secara detail.

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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekHarga(){
4         return 300000;
5     }
6 }
```

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 300000; }`. 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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekHarga();
4 }
5
6 class Televisi extends Produk{
7 }
8
9 $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`:

05.abstract_method_implement_ok.php

```
1 <?php
```

```

2 abstract class Produk {
3     abstract public function cekHarga();
4 }
5
6 class Televisi extends Produk{
7     public function cekHarga(){
8         return 3000000;
9     }
10}
11
12 $produk01 = new Televisi();
13 echo $produk01->cekHarga();

```

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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekHarga();
4 }
5
6 class Televisi extends Produk{
7     public function cekHarga(){
8         return 3000000;
9     }
10}
11
12 class MesinCuci extends Produk{
13     public function cekHarga(){
14         return 1500000;
15     }
16 }
17
18 $produk01 = new Televisi();
19 echo $produk01->cekHarga();
20
21 echo "<br>";
22
23 $produk01 = new MesinCuci();
24 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.

Dalam konsep OOP PHP, sebuah abstract class seharusnya memiliki minimal 1 buah abstract method. Namun PHP juga tidak memberikan pesan error jika kita membuat abstract class tanpa memiliki abstract method.

Sebuah abstract class juga bisa berisi property dan method biasa:

07.abstract_class_property_method.php

```

1 <?php
2 abstract class Produk {
3     private $stok = 200;
4     abstract public function cekHarga();
5
6     public function cekStok(){
7         return $this->stok;
8     }
9 }
10
11 class Televisi extends Produk{
12     public function cekHarga(){
13         return 3000000;
14     }
15 }
16
17
18 $produk01 = new Televisi();
19 echo $produk01->cekHarga(); // 3000000
20
21 echo "<br>";
22
23 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`.

Implementasi 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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekHarga($jumlah);
4 }
5
6 class Televisi extends Produk{
7     public function cekHarga($jumlah){
8         return 300000 * $jumlah;
9     }
10}
11
12 $produk01 = new Televisi();
13 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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekHarga($jumlah);
4 }
5
6 class Televisi extends Produk{
7     public function cekHarga(){
8         return 3000000;
9     }
10}
11
12 $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`:

10.abstract_method_visibility_error.php

```

1 <?php
2 abstract class Produk {

```

```

3   abstract public function cekHarga();
4 }
5
6 class Televisi extends Produk{
7   protected function cekHarga(){
8     return 300000;
9   }
10}
11
12 $produk01 = new Televisi();

```

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
1 <?php
2 abstract class Produk {
3   abstract protected function cekHarga();
4 }
5
6 class Televisi extends Produk{
7   public function cekHarga(){
8     return 300000;
9   }
10}
11
12 $produk01 = new Televisi();
13 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 ditimpak di dalam class turunan:

```

12.abstract_method_private_error.php
1 <?php
2 abstract class Produk {
3   abstract private function cekHarga();
4 }

```

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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekHarga();
4 }
5
6 abstract class Televisi extends Produk{
7     abstract public function cekTipe();
8 }
9
10 class TelevisiLED extends Televisi{
11     public function cekHarga(){
12         return 3000000;
13     }
14     public function cekTipe(){
15         return "TV LED";
16     }
17 }
18
19 $produk01 = new TelevisiLED();
20 echo $produk01->cekHarga(); // 3000000
21 echo "<br>";
22 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:

14.abstract_class_static.php

```

1 <?php
2 abstract class Produk {
3     public static $totalProduk = 100;
4
5     public static function cekProduk(){
6         return "Total Produk ada ".self::$totalProduk;
7     }
8 }
9

```

```

10 echo Produk::$totalProduk; // 100
11 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 (dari sekolah / kuliah), 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 ([wikipedia](#)).

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()`:

15.polymorfism.php

```

1 <?php
2 abstract class Produk {
3     abstract public function cekMerek();
4 }
5
6 class Televisi extends Produk{
7     public function cekMerek(){
8         return "Polytron";
9     }
10 }
11
12 class MesinCuci extends Produk{
13     public function cekMerek(){
14         return "Electrolux";
15     }
16 }
17
18 class LemariEs extends Produk{
19     public function cekMerek(){
20         return "Sharp";
21     }
22 }
23
24 $produk01 = new Televisi();
25 $produk02 = new MesinCuci();
26 $produk03 = new LemariEs();
27
28 echo $produk01->cekMerek(). "<br>"; // Polytron
29 echo $produk02->cekMerek(). "<br>"; // Electrolux
30 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

```

1 <?php
2 abstract class Produk {
3     abstract public function cekMerek();
4 }
5
6 class Televisi extends Produk{
7     public function cekMerek(){
8         return "Polytron";
9     }
10}
11
12 class MesinCuci extends Produk{
13     public function cekMerek(){
14         return "Electrolux";
15     }
16}
17
18 class LemariEs extends Produk{
19     public function cekMerek(){
20         return "Sharp";
21     }
22 }
23
24 $produk01 = new Televisi();
25 $produk02 = new MesinCuci();
26 $produk03 = new LemariEs();
27
28 function tampilanMerek($objectProduk){
29     return $objectProduk->cekMerek(). "<br>";
30 }
31
32 echo tampilanMerek($produk01);      // Polytron
33 echo tampilanMerek($produk02);      // Electrolux
34 echo tampilanMerek($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 `tampilanMerek()` dengan 1 parameter bernama `$objectProduk`. Sesuai namanya, parameter ini nantinya akan berisi object turunan class Produk. Fungsi `tampilanMerek()` ini mengembalikan nilai hasil `$objectProduk->cekMerek()`.

Mari kita bahas apa yang terjadi ketika perintah `echo tampilanMerek($produk01)` dijalankan.

Di baris 24 saya men-inisialisasi variabel `$produk01` sebagai object dari class Televisi. Variabel `$produk01` ini kemudian menjadi inputan argument untuk fungsi `tampilanMerek()` 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.

Penjelasan tentang *polymorphism* ini mungkin terasa cukup rumit. Tidak masalah jika anda perlu membacanya beberapa kali.

Mengenai fungsi `tampilkanMerek()` yang bisa diisi dengan parameter bertipe object, akan kita bahas secara detail di bagian tersendiri.

3.2. Interface

Interface adalah sebuah mekanisme di dalam OOP PHP untuk membuat 'kontrak' atau perjanjian *implementasi method*. Interface bisa juga disebut sebagai bentuk yang lebih khusus dari abstract class.

Secara garis besar, **interface** berfungsi sama seperti abstract class, yakni berisi kumpulan *signature method* yang harus di implementasikan. Setiap class yang menggunakan interface akan "dipaksa" membuat ulang method tersebut.

Berikut perbedaan antara **abstract class** dengan **interface**:

1. Abstract class bisa menjadi parent bagi class-class lain, sedangkan interface tidak berada di dalam struktur hierarki class sehingga tidak memiliki hubungan *child-parent* sebagaimana abstract class.
2. Abstract class bisa berisi konstanta, property dan method "biasa" (selain abstract method). Sedangkan interface hanya bisa berisi *signature method* dan konstanta saja. Dengan kata lain, interface tidak bisa diisi dengan property dan method biasa.
3. Interface dibuat menggunakan keyword `implement`.

Tujuan penggunaan interface juga mirip seperti abstract class, yakni membuat desain hubungan logika antar class (hierarki class). Hanya saja interface lebih ditujukan untuk kasus dimana terdapat abstract method yang kurang pas jika ditempatkan ke dalam abstract class.

Melanjutkan contoh class **Produk** kita sebelumnya, setiap produk seperti **Televisi**, **MesinCuci** dan **LemariEs** merupakan turunan dari abstract class **Produk**.

Misalkan beberapa produk akan di ekspor keluar negeri, sehingga saya perlu method **cekHargaUsd()** untuk mengetahui harga setiap barang dalam dollar Amerika (USD), serta method **cekNegara()** untuk mengetahui negara mana tujuan ekspor dari produk tersebut.

Method **cekHargaUsd()** dan **cekNegara()** ini menurut saya kurang pas jika ditempatkan ke dalam class **Produk** karena tidak semua produk akan di ekspor (hanya beberapa saja). Jika dibuat ke dalam class baru juga kurang tepat karena tidak masuk ke dalam struktur hierarki class. Saya ingin semua class hanya berasal dari turunan class **Produk** saja.

Dalam kasus seperti ini, method **cekHargaUsd()** dan **cekNegara()** menjadi kandidat yang pas untuk sebuah **interface**. Terlebih saya ingin setiap class produk yang akan di ekspor **harus** memiliki method **cekHargaUsd()** dan **cekNegara()**.

Berikut format dasar pembuatan interface **ProdukEkspor**:

```
interface ProdukEkspor {
    public function cekHargaUsd();
    public function cekNegara();
}
```

Interface dibuat menggunakan keyword **interface** yang diikuti dengan nama interface tersebut. Dalam contoh ini saya membuat interface dengan nama **ProdukEkspor**. Isi dari interface hanya bisa berupa *signature method*, yakni nama method serta parameter (jika ada). Implementasi dari method-method ini nantinya diserahkan kepada class yang menggunakan interface.

Agar sebuah class bisa menggunakan interface, tambahkan keyword **implements** setelah nama class. Sebagai contoh, jika saya ingin class **Televisi** menerapkan interface **ProdukEkspor**, maka kodenya adalah sebagai berikut:

```
class Televisi implements ProdukEkspor {

}
```

Dengan kode ini, di dalam class **Televisi** saya harus mendefinisikan ulang method **cekHargaUsd()** dan **cekNegara()** sebagaimana penulisan *signature method* dari interface **ProdukEkspor**. Jika tidak, PHP akan mengeluarkan pesan error.

Berikut kode program lengkap pembuatan interface **ProdukEkspor** serta implementasi method **cekHargaUsd()** dan **cekNegara()** di dalam class **Televisi**:

17.interface_basic.php

```
1 <?php
2 interface ProdukEkspor {
```

```

3   public function cekHargaUsd();
4   public function cekNegara();
5 }
6
7 class Televisi implements ProdukEkspor {
8   public function cekHargaUsd(){
9     return 185;
10  }
11  public function cekNegara(){
12    return ["Singapura", "Malaysia", "Thailand"];
13  }
14 }
15
16 $produk01 = new Televisi();
17 echo $produk01->cekHargaUsd();
18 echo "<br>";
19 echo implode(", ", $produk01->cekNegara());

```

Hasil kode program:

```

185
Singapura, Malaysia, Thailand

```

Sama seperti *abstract method*, implementasi dari setiap method yang ada di dalam interface diserahkan sepenuhnya kepada class. Di dalam class `Televisi` saya harus membuat ulang method `cekHargaUsd()` dan `cekNegara()`, dimana method `cekHargaUsd()` mengembalikan angka 185 dan method `cekNegara()` mengembalikan sebuah array `["Singapura", "Malaysia", "Thailand"]`.

Selanjutnya di baris 16 saya membuat object `$produk01` dari class `Televisi`, kemudian menampilkan hasil pemanggilan `$produk01->cekHargaUsd()` dan `$produk01->cekNegara()`.

Khusus untuk method `cekNegara()`, di baris 19 saya menggunakan fungsi `implode()` bawaan PHP untuk mengkonversi array menjadi string. Ini karena method `cekNegara()` mengembalikan sebuah array yang tidak bisa langsung di-echo.

Sebagai percobaan, saya akan membuat class `MesinCuci` dan menggunakan interface `ProdukEkspor`, tapi tidak membuat ulang method tersebut:

```

18.interface_error.php

1 <?php
2 interface ProdukEkspor {
3   public function cekHargaUsd();
4   public function cekNegara();
5 }
6
7 class MesinCuci implements ProdukEkspor {
8 }

```

Hasil kode program:

```
// Fatal error: Class MesinCuci contains 2 abstract methods and must therefore be  
declared abstract or implement the remaining methods (ProdukEkspor::cekHargaUsd,  
ProdukEkspor::cekNegara)
```

Hasilnya tampil pesan error karena class `MesinCuci` tidak mengimplementasikan ulang semua method yang ada di dalam interface `ProdukEkspor`.

Signature method yang ada di dalam interface juga harus memiliki hak akses `public`, tidak bisa di set sebagai `private` atau `protected`:

19.interface_visibility_error.php

```
1 <?php  
2 interface ProdukEkspor {  
3     private function cekHargaUsd();  
4     protected function cekNegara();  
5 }
```

Hasil kode program:

```
// Fatal error: Access type for interface method ProdukEkspor::cekHargaUsd() must  
be omitted
```

Atau kita bisa tidak menulis visibility sama sekali karena itu akan dianggap PHP sebagai `public`:

```
1 <?php  
2 interface ProdukEkspor {  
3     function cekHargaUsd();  
4     function cekNegara();  
5 }
```

Sebuah class juga bisa mengimplementasikan banyak interface sekaligus. Berikut contohnya:

20.interface_bulk.php

```
1 <?php  
2 interface ProdukEkspor {  
3     public function cekHargaUsd();  
4     public function cekNegara();  
5 }  
6  
7 interface ProdukMakanan {  
8     public function cekExpired();  
9 }  
10  
11 interface ProdukMakananBeku {  
12     public function cekSuhuMin();  
13 }  
14  
15 class Nugget implements ProdukEkspor, ProdukMakanan, ProdukMakananBeku {  
16     public function cekHargaUsd(){  
17         return 7.5;
```

```

18 }
19 public function cekNegara(){
20     return ["Singapura", "Malaysia", "Thailand"];
21 }
22 public function cekExpired(){
23     return "April 2019";
24 }
25 public function cekSuhuMin(){
26     return -14;
27 }
28 }
29
30 $produk01 = new Nugget();
31 echo $produk01->cekHargaUsd();
32 echo "<br>";
33 echo implode(", ", $produk01->cekNegara());
34 echo "<br>";
35 echo $produk01->cekExpired();
36 echo "<br>";
37 echo $produk01->cekSuhuMin();

```

Hasil kode program:

```

7.5
Singapura, Malaysia, Thailand
April 2019
-14

```

Dalam kode program ini saya membuat 3 buah interface: `ProdukEkspor`, `ProdukMakanan` dan `ProdukMakananBeku`.

Interface `ProdukMakanan` di rancang untuk produk makanan. Dimana nantinya saya perlu mengetahui kapan tanggal expired dari produk makanan tersebut. Oleh karena itu saya ingin semua produk makanan memiliki method `cekExpired()`.

Interface `ProdukMakananBeku` di tujuhan untuk produk makanan yang harus disimpan dalam suhu dingin. Oleh karena itu saya juga ingin setiap produk makanan beku memiliki method `cekSuhuMin()`. Method ini untuk mengetahui berapa suhu minimum penyimpanan makanan.

Di baris 20 saya membuat class `Nugget` yang mengimplementasikan ketiga interface tersebut. Perhatikan cara penulisannya, yakni dengan keyword `implements` dan diikuti dengan ketiga nama interface (dipisah dengan tanda koma).

Karena hal ini, di dalam class `Nugget` saya harus mendefinisikan ulang ke-4 method tersebut, yakni method `cekHargaUsd()` dan `cekNegara()` milik interface `ProdukEkspor`, method `cekExpired()` milik interface `ProdukMakanan`, serta method `cekSuhuMin()` milik interface `ProdukMakananBeku`.

Untuk method `cekHargaUsd()` akan mengembalikan nilai 7.5 yang artinya harga produk ini adalah US\$ 7.5. Method `cekNegara()` mengembalikan array 3 negara tujuan ekspor, yakni

["Singapura", "Malaysia", "Thailand"]. Kemudian method cekExpired() mengembalikan string "April 2019" yakni tanggal expired produk Nugget. Terakhir method cekSuhuMin() mengembalikan angka -14, dimana produk ini bisa disimpan dalam suhu minimum -14 derajat celcius.

Interface Inheritance

Interface juga bisa diturunkan ke dalam interface lain. Caranya sama seperti penurunan class, yakni menggunakan keyword extends:

21.interface_inheritance.php

```

1 <?php
2 interface ProdukEkspor {
3     public function cekHargaUsd();
4     public function cekNegara();
5 }
6
7 interface ProdukMakanan {
8     public function cekExpired();
9 }
10
11 interface ProdukMakananBeku extends ProdukMakanan {
12     public function cekSuhuMin();
13 }
14
15 class Nugget implements ProdukEkspor, ProdukMakananBeku {
16     public function cekHargaUsd(){
17         return 7.5;
18     }
19     public function cekNegara(){
20         return ["Singapura", "Malaysia", "Thailand"];
21     }
22     public function cekSuhuMin(){
23         return -14;
24     }
25 }
```

Hasil kode program:

Fatal error: Class Nugget contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (ProdukMakanan::cekExpired)

Perhatikan di baris 11, saya membuat interface ProdukMakananBeku sebagai turunan dari interface ProdukMakanan. Hasilnya, interface ProdukMakananBeku akan memiliki signature method cekExpired() milik interface ProdukMakanan.

Class Nugget hanya menggunakan interface ProdukEkspor dan ProdukMakananBeku saja (tidak menggunakan interface ProdukMakanan). Oleh kerena itu saya mencoba untuk tidak mengimplementasikan method cekExpired(), karena method tersebut milik interface

ProdukMakanan.

Hasilnya PHP menampilkan pesan error "komplain" karena saya tidak membuat ulang method `cekExpired()`. Meskipun method ini berada di interface `ProdukMakanan` yang tidak dipakai class `Nugget`, namun method ini secara langsung ada di dalam interface `ProdukMakananBeku`.

Interface Constant

Yang cukup unik dari interface adalah, kita diperbolehkan membuat konstanta:

22.interface_const.php

```

1 <?php
2 interface ProdukEkspor {
3     public function cekHargaUsd();
4     public function cekNegara();
5     public const biayaPajak = 0.5;
6 }
7
8 echo ProdukEkspor::biayaPajak; // 0.5

```

Di baris 5 saya membuat konstanta `biayaPajak` dan memberinya nilai 0.5. Cara pengaksesannya ini mirip seperti konstanta class, yakni dengan menggunakan scope resolution operator berupa tanda " :: ".

Masih berhubungan dengan class, penamaan interface di dalam PHP "berbagi tempat" dengan nama class. Maksudnya, tidak boleh ada nama class dan interface yang sama:

23.interface_naming.php

```

1 <?php
2 interface ProdukEkspor {
3     public function cekHargaUsd();
4     public function cekNegara();
5 }
6
7 class ProdukEkspor{
8 }

```

Hasil kode program:

```
// Fatal error: Cannot declare class ProdukEkspor, because the name is already in use
```

Di sini saya membuat interface bernama `ProdukEkspor`, kemudian membuat class dengan nama yang sama. Hasilnya terjadi error karena nama `ProdukEkspor` sudah dipakai oleh interface. Error seperti ini juga akan terjadi jika kita membuat 2 buah class menggunakan nama yang sama.

Menutup pembahasan kita tentang interface, pertanyaan cukup sering adalah *kapan harus menggunakan abstract class dan kapan harus menggunakan interface?*

Sebenarnya tidak ada rumus baku karena ini lebih ke seperti apa logika anda untuk merancang kode program (design class yang dipakai). Tapi tips singkatnya, jika method tersebut hampir selalu dipakai di dalam class turunan, maka tempatkan ke dalam abstract class. Namun jika method tersebut relatif jarang terpakai dan tidak secara langsung berhubungan dengan struktur class, maka tempatkan ke dalam interface.

3.3. Trait

Trait merupakan solusi PHP untuk mengatasi batasan *multiple inheritance*. **Multiple inheritance** adalah konsep dimana sebuah class bisa memiliki 2 parent atau lebih.

Sebagai contoh, perhatikan kode program berikut:

```
24.multiple_inheritance.php

1 <?php
2 class Televisi {
3     public function cekResolusi(){
4         return "Full HD";
5     }
6 }
7
8 class Smartphone{
9     public function cekOS(){
10    return "Android 9.0 (Pie)";
11 }
12 }
13
14 class SmartTV extends Televisi, Smartphone{
15 }
```

Di baris 2 saya membuat class `Televisi` dengan method `cekResolusi()`. Method ini mengembalikan string `"Full HD"`, yakni resolusi layar produk televisi.

Di baris 8 saya membuat class `Smartphone` dengan method `cekOS()`. Method ini dipakai untuk memeriksa sistem operasi dari produk smartphone, yang dalam contoh di atas berisi `"Android 9.0 (Pie)"`.

Di baris 14 saya membuat class `SmartTV`. Sebagai informasi, **smart TV** adalah sebutan untuk televisi dengan kemampuan mengakses internet. Beberapa produk smart TV ada yang berbasis sistem operasi android sebagaimana layaknya smartphone (memiliki processor dan RAM bawaan). Oleh karena itu, cukup logis kalau class `SmartTV` dibuat sebagai turunan dari class `Televisi` dan class `Smartphone`.

Inilah penerapan dari *multiple inheritance*, dimana class `SmartTV` diturunkan dari 2 class lain.

Akan tetapi, PHP tidak mendukung *multiple inheritance*, kode program di atas akan menghasilkan pesan error:

```
Parse error: syntax error, unexpected ',', expecting '{' on line 14
```

PHP menggunakan konsep **single inheritance**, dimana sebuah class hanya bisa diturunkan dari 1 class saja. Batasan seperti ini lebih ke keputusan design dari tim pengembang bahasa PHP. Dalam beberapa kasus, *multiple inheritance* juga bisa menimbulkan masalah baru yang dikenal sebagai [diamond problem](#), yakni kasus dimana method parent class yang satu bisa bentrok dengan method dari parent class lain (jika terdapat nama method yang sama).

Jadi, kalau tidak bisa menggunakan *multiple inheritance*, bagaimana solusi untuk contoh kita? Class `SmartTV` seharusnya bisa memiliki semua method yang ada di class `Televisi` dan `Smartphone`, ini agar tidak perlu menulis ulang method-method tersebut di class `SmartTV`.

Sebagai solusi, PHP 5.3 memperkenalkan *trait*. **Trait** ini mirip seperti *interface*, dimana setiap class nantinya bisa menggunakan lebih dari 1 trait. Namun berbeda dengan *interface*, di dalam *trait* kita bisa membuat implementasi method, tidak hanya *signature* method saja.

Berikut format dasar penulisan *trait*:

```
1 trait namaTrait {
2     public $property1;
3     public $property2;
4     public function fungs1() {
5         // isi dari fungs1 di sini...
6     }
7 }
```

Isi dari *trait* sendiri sama seperti class pada umumnya, yakni berupa property dan method.

Class `SmartTV` bisa saya tulis ulang sebagai berikut:

`25.trait_basic.php`

```
1 <?php
2 class Televisi {
3     public function cekResolusi(){
4         return "Full HD";
5     }
6 }
7
8 trait SmartElectronic{
9     public function cekOS(){
10        return "Android 9.0 (Pie)";
11    }
12 }
13
14 class SmartTV extends Televisi{
15     use SmartElectronic;
16     public function cekInfo(){
```

```

17     return "Smart TV ".$this->cekResolusi()." - ".$this->cekOS();
18 }
19 }
20
21 $produk01 = new SmartTV;
22 echo $produk01->cekInfo();

```

Hasil kode program:

```
Smart TV Full HD - Android 9.0 (Pie)
```

Di baris 8 saya mengubah class `Smartphone` menjadi trait `SmartElectronic`. Perubahan nama ini agar saya bisa membedakan antara sebuah produk dengan non-produk. Ini semata-mata pilihan design yang saya pakai. Jika pun anda ingin menggunakan trait `Smartphone`, itu pun tetap bisa berjalan. Trait `SmartElectronic` nantinya dirancang untuk menampung berbagai method untuk produk "smart", seperti `SmartTV`, `SmartWatch`, `SmartSpeaker`, dll.

Isi dari trait `SmartElectronic` sama seperti class `Smartphone` sebelumnya, yakni sebuah method `cekOS()` yang mengembalikan string "Android 9.0 (Pie)".

Di baris 14, saya membuat class `SmartTV` sebagai turunan dari class `Televisi`. Kemudian di baris 15 terdapat perintah `use SmartElectronic`. Inilah cara agar class `SmartTV` bisa mengakses trait `SmartElectronic`.

Pada class `SmartTV` ini saya membuat method `cekInfo()` yang akan mengakses method `cekResolusi()` milik class `Televisi`, serta method `cekOS()` milik trait `SmartElectronic`. Kedua method ini diakses dari pseudo-variable `$this` karena sudah menjadi bagian dari class `SmartTV`.

Sebuah class juga bisa menggunakan lebih dari 1 trait, berikut contohnya:

```
26.trait_multiple.php
```

```

1 <?php
2 trait SmartElectronic{
3     public function cekOS(){
4         return "Android 9.0 (Pie)";
5     }
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10        return "Konsumsi daya 0.8";
11    }
12 }
13
14 class SmartTV{
15     use SmartElectronic, LowWatt;
16     public function cekInfo(){
17         return "Smart TV ".$this->cekOS()." - ".$this->efisiensi();
18     }

```

```

19 }
20
21 $produk01 = new SmartTV;
22 echo $produk01->cekInfo();

```

Hasil kode program:

```
Smart TV Android 9.0 (Pie) - Konsumsi daya 0.8
```

Dalam contoh ini saya membuat 2 buah trait: `SmartElectronic` dan `LowWatt`. Trait `SmartElectronic` sama seperti sebelumnya yakni berisi method `cekOS()`. Sedangkan trait `LowWatt` berisi method `efisiensi()` yang akan mengembalikan string "Konsumsi daya 0.8".

Trait `LowWatt` saya rancang untuk produk yang menggunakan fitur hemat daya. Konsumsi daya 0.8 artinya produk tersebut hanya butuh konsumsi daya 80% dari produk biasa.

Kedua trait di input ke dalam class `SmartTV` dengan perintah `use SmartElectronic, LowWatt` di baris 15. Dengan perintah ini, class `SmartTV` sudah memiliki method `cekOS()` dan `efisiensi()`. Jika anda butuh menginput trait lain, tinggal di sambung dengan nama trait yang dipisah dengan tanda koma.

Urutan Prioritas Method

Jika terdapat method dengan nama yang sama antara parent class, trait dan class itu sendiri, urutannya adalah sebagai berikut:

1. Method di dalam class itu sendiri.
2. Method milik trait.
3. Method milik parent class.

Berikut contoh prakteknya:

```
27.trait_priority_1.php
```

```

1 <?php
2 class Televisi {
3     public function efisiensi(){
4         return "Konsumsi daya 1.0";
5     }
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10        return "Konsumsi daya 0.8";
11    }
12 }
13
14 class SmartTV extends Televisi{
15     use LowWatt;
16 }

```

```

17
18 $produk01 = new SmartTV;
19 echo $produk01->efisiensi();

```

Dalam contoh ini saya membuat method efisiensi() di class Televisi serta trait LowWatt. Kemudian class Televisi di turunkan ke dalam class SmartTV, dan class SmartTV juga menggunakan trait LowWatt.

Pertanyaannya, ketika kita memanggil method efisiensi() di dalam class SmartTV, method milik siapa yang akan dipakai? Berikut hasil kode program dari contoh di atas:

Konsumsi daya 0.8

Hasilnya method milik trait LowWatt lah yang akan dipakai. Di sini kita bisa ambil kesimpulan bahwa jika terdapat method dengan nama sama yang berasal dari trait dan parent element, method dari trait lah yang akan di prioritaskan.

Namun jika di dalam class SmartTV ternyata juga ada method efisiensi(), maka method milik class SmartTV lah yang akan dipakai:

28.trait_priority_2.php

```

1 <?php
2 class Televisi {
3     public function efisiensi(){
4         return "Konsumsi daya 1.0";
5     }
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10        return "Konsumsi daya 0.8";
11    }
12 }
13
14 class SmartTV extends Televisi{
15     use LowWatt;
16
17     public function efisiensi(){
18         return "Konsumsi daya 0.9";
19     }
20 }
21
22 $produk01 = new SmartTV;
23 echo $produk01->efisiensi();

```

Hasil kode program:

Konsumsi daya 0.9

Kali ini saya memodifikasi kode sebelumnya dengan menambah method efisiensi() ke dalam

class SmartTV. Ketika di panggil method `efisiensi()` pada baris 23, method milik class SmartTV lah yang akan diproses, bukan method `efisiensi()` milik trait LowWatt maupun parent class Televisi.

Conflict Resolution

Dalam bahasan sebelumnya kita telah mempelajari urutan prioritas ketika terdapat method dengan nama yang sama dalam **class**, **trait** dan **parent class**.

Sekarang bagaimana jika ternyata di trait yang berbeda terdapat method dengan nama yang sama? Berikut contohnya:

29.trait_conflict.php

```

1 <?php
2 trait SmartElectronic{
3     public function efisiensi(){
4         return "Konsumsi daya 1.1";
5     }
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10        return "Konsumsi daya 0.8";
11    }
12 }
13
14 class SmartTV{
15     use SmartElectronic, LowWatt;
16 }
17
18 $produk01 = new SmartTV;
19 echo $produk01->efisiensi()

```

Dalam kode program di atas saya membuat 2 buah trait: `SmartElectronic` dan `LowWatt`. Kedua trait sama-sama memiliki method dengan nama `efisiensi()`. Di baris 15, kedua trait di input ke dalam class `SmartTV`. Kemudian di baris 19 saya mengakses method `efisiensi()` dari object class `SmartTV`.

Berikut hasilnya:

```
Fatal error: Trait method efisiensi has not been applied, because there are
collisions with other trait methods on SmartTV
```

Terjadi error karena PHP "bingung" method `efisiensi()` mana yang harus dijalankan, apakah kepunyaan trait `SmartElectronic` atau kepunyaan trait `LowWatt`?

Untuk mengatasi hal ini, kita perlu memberi tahu PHP method mana yang akan dijalankan:

30.trait_conflict_insteadof.php

```

1 <?php
2 trait SmartElectronic{
3     public function efisiensi(){
4         return "Konsumsi daya 1.1";
5     }
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10        return "Konsumsi daya 0.8";
11    }
12 }
13
14 class SmartTV{
15     use SmartElectronic, LowWatt {
16         SmartElectronic::efisiensi insteadof LowWatt;
17     }
18 }
19
20 $produk01 = new SmartTV;
21 echo $produk01->efisiensi();

```

Hasil kode program:

Konsumsi daya 1.1

Perhatikan kode program di baris 15 – 17. Setelah perintah `use SmartElectronic, LowWatt,` diikuti blok kode program dalam tanda kurung kurawal. Di dalamnya saya menulis kode `SmartElectronic::efisiensi insteadof LowWatt.` Perintah ini adalah informasi kepada PHP bahwa jika di dalam class `SmartTV` dipanggil method `efisiensi()`, method dari trait `SmartElectronic`-lah yang akan dipakai.

Jika anda ingin sebaliknya, yakni method `efisiensi()` milik trait `LowWatt` yang akan dipakai, maka perintahnya menjadi `LowWatt::efisiensi insteadof SmartElectronic.`

Sekarang muncul pertanyaan, bagaimana cara mengakses method `efisiensi()` milik trait lain? PHP mengizinkan kita untuk memberi nama baru atau nama alias untuk method tersebut:

31.trait_conflict_as.php

```

1 <?php
2 trait SmartElectronic{
3     public function efisiensi(){
4         return "Konsumsi daya 1.1";
5     }
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10        return "Konsumsi daya 0.8";
11    }

```

```

12 }
13
14 class SmartTV{
15     use SmartElectronic, LowWatt {
16         SmartElectronic::efisiensi insteadof LowWatt;
17         LowWatt::efisiensi as efisiensiLow;
18     }
19 }
20
21 $produk01 = new SmartTV;
22 echo $produk01->efisiensi();
23 echo "<br>";
24 echo $produk01->efisiensiLow();

```

Hasil kode program:

```

Konsumsi daya 1.1
Konsumsi daya 0.8

```

Sekarang terdapat tambahan kode `LowWatt::efisiensi as efisiensiLow` di baris 17. Perintah ini dipakai untuk memberi nama baru (nama alias) dari method `efisiensi()` milik trait `LowWatt`. Kita bebas ingin memberikan nama apa saja, dalam contoh ini saya memberi nama `efisiensiLow()`.

Hasilnya, kedua method `efisiensi()` sudah bisa diakses. Jika dipanggil `$produk01->efisiensi()`, itu artinya method `efisiensi()` milik trait `SmartElectronic`. Jika dipanggil `$produk01->efisiensiLow()`, itu akan mengakses method `efisiensi()` milik trait `LowWatt`.

Trait di dalam Trait

Trait juga bisa dipakai di dalam trait lain. Caranya juga sama seperti di dalam class, yakni dengan keyword `use`:

```

32.trait_in_trait.php

1 <?php
2 trait SmartElectronic{
3     public function cekOS(){
4         return "Android 9.0 (Pie)";
5     }
6 }
7
8 trait LowWatt{
9     use SmartElectronic;
10    public function efisiensi(){
11        return $this->cekOS()." Konsumsi daya 0.8";
12    }
13 }
14
15 class SmartTV{
16     use LowWatt;
17 }

```

```

18
19 $produk01 = new SmartTV;
20 echo $produk01->efisiensi();      // Android 9.0 (Pie) Konsumsi daya 0.8

```

Di baris 9 terdapat perintah `use SmartElectronic` di dalam trait `LowWatt`. Maksudnya, saya ingin menginput trait `SmartElectronic` ke dalam trait `LowWatt`. Dengan perintah ini, method `cekOS()` milik `SmartElectronic` bisa diakses dari dalam trait `LowWatt`.

Trait Property

Di dalam trait, boleh terdapat property sebagaimana class biasa:

33.trait_property.php

```

1 <?php
2 trait SmartElectronic{
3     public $internet = "Telkom Indihome";
4     public function cekOS(){
5         return "Android 9.0 (Pie)";
6     }
7 }
8
9 class SmartTV{
10    use SmartElectronic;
11 }
12
13 $produk01 = new SmartTV;
14 echo $produk01->internet;      // Telkom Indihome
15 echo "<br>";
16 echo $produk01->cekOS();        // Android 9.0 (Pie)

```

Saya membuat property `$internet` di dalam trait `SmartElectronic`, kemudian menggunakannya di dalam class `SmartTV`. Hasilnya, di baris 14 saya bisa mengakses property ini melalui object dari class `SmartTV`.

Bagaimana dengan konstanta? Mari kita coba:

34.trait_const.php

```

1 <?php
2 trait SmartElectronic{
3     public $internet = "Telkom Indihome";
4     const KECEPATAN = "10 Mbps";
5     public function cekOS(){
6         return "Android 9.0 (Pie)";
7     }
8 }
9
10 echo SmartElectronic::KECEPATAN;
11 // Fatal error: Traits cannot have constants

```

Hasilnya tampil pesan error bahwa trait tidak bisa memiliki konstanta.

Trait Static Property dan Static Method

Selain memiliki property dan method biasa, trait juga bisa diisi dengan static property dan static method. Berikut contohnya:

35.trait_static.php

```

1 <?php
2 trait SmartElectronic{
3     public static $internet = "Telkom Indihome";
4     public static function cekOS(){
5         return "Android 9.0 (Pie)";
6     }
7 }
8
9 echo SmartElectronic::$internet;      // Telkom Indihome
10 echo "<br>";
11 echo SmartElectronic::cekOS();        // Android 9.0 (Pie)

```

Di dalam trait `SmartElectronic` saya membuat static property `$internet` dan static method `cekOS()`. Cara akses keduanya sama seperti class biasa, yakni dengan menulis nama trait, diikuti tanda " :: " serta nama static property dan static method yang akan diakses.

Trait Abstract Method

Yang juga unik, PHP mengizinkan sebuah trait memiliki abstract method. Akibatnya, semua class yang menggunakan trait tersebut harus mengimplementasikan ulang abstract method:

36.trait_abstract_method_1.php

```

1 <?php
2 trait SmartElectronic{
3     public function cekOS(){
4         return "Android 9.0 (Pie)";
5     }
6     abstract public function cekProcessor();
7 }
8
9 class SmartTV{
10     use SmartElectronic;
11 }

```

Hasil kode program:

Fatal error: Class `SmartTV` contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (`SmartTV::cekProcessor()`)

Error di atas terjadi karena saya tidak mengimplementasikan ulang abstract method `cekProcessor()` di dalam class `SmartTV`:

37.trait_abstract_method_2.php

```

1 <?php
2 trait SmartElectronic{
3     public function cekOS(){
4         return "Android 9.0 (Pie)";
5     }
6     abstract public function cekProcessor();
7 }
8
9 class SmartTV{
10    use SmartElectronic;
11    public function cekProcessor() {
12        return "Snapdragon 845";
13    }
14 }
15
16 $produk01 = new SmartTV;
17 echo $produk01->cekOS();           // Android 9.0 (Pie)
18 echo "<br>";
19 echo $produk01->cekProcessor();    // Snapdragon 845

```

Dengan abstract method, kita bisa memaksakan class untuk mengimplementasikan ulang sebuah method sebagaimana layaknya jika diturunkan dari abstract class maupun interface.

Trait Access Modifier

Pada saat menggunakan trait, kita bisa mengubah hak akses dari sebuah method yang ada di dalam trait. Berikut contohnya:

38.trait_access_modifier.php

```

1 <?php
2 trait SmartElectronic{
3     public function cekOS(){
4         return "Android 9.0 (Pie)";
5     }
6 }
7
8 class SmartTV{
9     use SmartElectronic { cekOS as protected; }
10 }
11
12 $produk01 = new SmartTV;
13 echo $produk01->cekOS();

```

Hasil kode program:

```
Fatal error: Uncaught Error: Call to protected method SmartTV::cekOS()
```

Di awal kode program saya membuat trait `SmartElectronic` yang berisi method `cekOS()`. Method `cekOS()` ini di set sebagai `public`.

Pada saat diinput ke dalam class `SmartTV`, di baris 9 saya membuat perintah `use SmartElectronic {cekOS as protected;}`. Kode ini akan mengubah hak akses dari method `cekOS()` menjadi `protected` khusus di dalam class `SmartTV` ini. Akibatnya akan tampil pesan error ketika saya mencoba mengakses method `cekOS()` dari luar class (baris 13).

Trait Name Collision

Sama seperti interface, trait juga "berbagi nama" dengan class. Jika sebuah trait sudah di definisikan dengan nama tertentu, kita tidak bisa membuat class maupun interface dengan nama yang sama:

39.trait_name_collision.php

```
1 <?php
2 trait SmartElectronic{ }
3
4 class SmartElectronic{ }
```

Hasil kode program:

```
Fatal error: Cannot declare class SmartElectronic, because the name is already in use
```

Error di atas terjadi karena saya mencoba membuat class bernama sama dengan nama trait.

Bersama-sama dengan abstract class dan interface, trait memberikan pilihan design alternatif dalam merancang kode program. Ketiga fitur ini (*abstract class, interface* dan *trait*) relatif jarang dipakai kecuali untuk website yang terbilang rumit. Namun mengetahui fitur-fitur ini setidaknya menambah pemahaman kita tentang konsep OOP PHP secara keseluruhan.

3.4. Magic Constant

Magic constant adalah konstanta bawaan PHP yang berisi berbagai informasi tentang kode program saat ini. Penulisan magic constant ditandai dengan garis bawah dua kali (*double underscore*) di awal dan di akhir konstanta. Magic constant sebenarnya tidak hanya untuk OOP PHP, tapi juga tersedia untuk pemrograman PHP prosedural.

Tabel berikut merangkum 7 magic constant yang tersedia di dalam PHP:

Nama Konstanta	Penjelasan
<code>__LINE__</code>	Berisi nomor baris
<code>__FILE__</code>	Berisi <i>full path</i> (nama folder) dan nama file
<code>__DIR__</code>	Berisi <i>full path</i> (nama folder)

Nama Konstanta	Penjelasan
__FUNCTION__	Berisi nama fungsi
__CLASS__	Berisi nama class
__TRAIT__	Berisi nama trait
__METHOD__	Berisi nama method
__NAMESPACE__	Berisi nama namespace

Isi setiap konstanta akan berbeda-beda tergantung dari mana konstanta tersebut dijalankan. Berikut contohnya:

40.magic_constant.php

```

1 <?php
2 echo "Kode ini berada di file: ".__FILE__."
3 echo "Yang berada di dalam folder: ".__DIR__."
4 echo "Perintah ini berasal dari baris: ".__LINE__."
5
6 function belajar_magic_constant(){
7     return "Kode ini berada di dalam fungsi: ".__FUNCTION__;
8 }
9
10 echo belajar_magic_constant();

```

Hasil kode program:

Kode ini berada di file: C:\xampp\htdocs\belajar_oop_php\bab_03\40.magic_constant.php

Yang berada di dalam folder: C:\xampp\htdocs\belajar_oop_php\bab_03

Perintah ini berasal dari baris: 4

Kode ini berada di dalam fungsi: belajar_magic_constant

Terlihat setiap konstanta menghasilkan nilai yang berbeda-beda. Dan berikut contoh penggunaan magic constant di dalam OOP PHP:

41.magic_constant_class.php

```

1 <?php
2 trait HardCover {
3     public function cekTrait(){
4         $psn = "Pesan ini berasal dari method ".__METHOD__;
5         $psn .= " di dalam trait ".__TRAIT__;
6         return $psn;
7     }
8 }

```

```

9
10 class Buku {
11     use HardCover;
12     public function cekClass(){
13         $psn = "Pesan ini berasal dari method ".__METHOD__;
14         $psn .= " di dalam class ".__CLASS__;
15         return $psn;
16     }
17 }
18
19 $produk01 = new Buku();
20 echo $produk01->cekTrait();
21 echo "<br>";
22 echo $produk01->cekClass();

```

Hasil kode program:

```

Pesan ini berasal dari method HardCover::cekTrait di dalam trait HardCover
Pesan ini berasal dari method Buku::cekClass di dalam class Buku

```

Dalam contoh ini saya menggunakan magic constant __METHOD__, __TRAIT__ dan __CLASS__. Setiap konstanta ini akan berisi nilai sesuai dimana konstanta tersebut diakses.

Terakhir, berikut contoh penggunaan dari magic constant __NAMESPACE__:

42.magic_constant_namespace.php

```

1 <?php
2 namespace DuniaIlkom;
3
4 class Buku {
5     public function cekClass(){
6         $psn = "Pesan ini berasal dari class ".__CLASS__;
7         $psn .= " di dalam namespace ".__NAMESPACE__;
8         return $psn;
9     }
10 }
11
12 $produk01 = new Buku();
13 echo $produk01->cekClass();

```

Hasil kode program:

```

Pesan ini berasal dari class DuniaIlkom\Buku di dalam namespace DuniaIlkom

```

Materi tentang **namespace** memang belum saya bahas, namun setidaknya kita bisa melihat bahwa PHP menyediakan magic constant untuk keperluan tersebut.

Magic constant ini sering dipakai untuk proses *debugging* (pencarian kesalahan). Jika kode program tersebut terdiri dari puluhan class yang saling "menurunkan", cukup sulit untuk menebak asal dari sebuah method. Kita bisa memanfaatkan beberapa konstanta ini untuk penelusuran lebih lanjut.

3.5. Magic Method

Magic method adalah method khusus yang dijalankan jika sebuah kondisi terjadi. Sama seperti magic constant, PHP menyediakan cara penulisan khusus untuk magic method, dimana nama method diawali dengan tanda garis bawah dua kali (*double underscore*).

PHP menyediakan cukup banyak magic method dan kita sudah mempelajari 2 diantaranya, yakni `__construct()` dan `__destruct()`. Kedua method ini akan dijalankan secara otomatis pada saat object dibuat dan ketiga object dihapus.

Setiap itu terdapat juga `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` dan `__debugInfo()`.

Saya tidak akan menjelaskan semua magic method ini karena sebagian besar juga sangat jarang dipakai. Kita akan bahas beberapa yang dianggap penting.

Magic Method `__toString()`

Method `__toString()` akan di jalankan ketika sebuah object "di tuntut" untuk menghasilkan nilai string. Ini bisa terjadi jika object tersebut berada di dalam perintah echo seperti contoh berikut:

43.magic_method_print_error.php

```

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

```

Hasil kode program:

```
Recoverable fatal error: Object of class Produk could not be converted to string
```

Dalam kode program ini saya langsung men-echo-kan object `$produk01`. Kode seperti ini memang tidak umum karena seharusnya yang di echo berupa property atau method, bukan langsung object. Hasilnya tampil pesan error karena object `Produk` tidak bisa dikonversi menjadi string.

Jika diinginkan, kita bisa membuat method `__toString()` di dalam class `Produk` yang bisa menangani situasi seperti ini:

44.magic_method_print.php

```

1 <?php
2 class Produk {
3     public function __toString(){

```

```

4     return "Ini berasal dari class Produk";
5 }
6 }
7
8 $produk01 = new Produk();
9 echo $produk01; // Ini berasal dari class Produk

```

Di dalam class Produk saya membuat method `__toString()` yang isinya mengembalikan string "Ini berasal dari class Produk". Hasilnya ketika object `$produk01` di-echo, tidak lagi terdapat error namun tampil hasil string sebagaimana isi dari method `__toString()`.

Magic method `__toString()` ini juga akan tetap aktif di dalam class turunan:

45.magic_method_print_extends.php

```

1 <?php
2 class Produk {
3     public function __toString(){
4         return "Ini berasal dari class Produk";
5     }
6 }
7
8 class Televisi extends Produk {
9 }
10
11 $produk01 = new Televisi();
12 echo $produk01; // Ini berasal dari class Produk

```

Variabel `$produk01` sekarang berisi object dari class `Televisi`. Class `Televisi` sendiri tidak memiliki method `__toString()` secara langsung, tapi di turunkan dari class `Produk`. Hasilnya, method `__toString()` milik class `Produk` yang akan dijalankan.

Magic Method `__get()`

Method `__get()` adalah magic method yang dijalankan ketika kita mengakses property yang tidak ada (atau tidak bisa diakses) dari luar class. Berikut contoh kasusnya:

46.property_access_error.php

```

1 <?php
2 class Produk {
3 }
4
5 $produk01 = new Produk();
6 echo $produk01->merek;

```

Hasil kode program:

Warning: Undefined property: Produk::\$merek

Dalam contoh di atas saya membuat class `Produk` tanpa isi apapun (class kosong). Class `Produk`

ini kemudian di instansiasi ke dalam variabel \$produk01. Di baris 6 saya mencoba mengakses property merek milik object \$produk01. Hasilnya terdapat error karena di class Produk tidak memiliki property apapun.

Agar kode di atas tidak menghasilkan error, saya harus mendefinisikan property merek di dalam class Produk terlebih dahulu. Atau alternatifnya saya bisa membuat magic method __get():

47.magic_method_get.php

```

1 <?php
2 class Produk {
3     public function __get($name) {
4         return "Maaf property '$name' tidak terdefinisi";
5     }
6 }
7
8 $produk01 = new Produk();
9 echo $produk01->merek; // Maaf property 'merek' tidak terdefinisi
10 echo "<br>";
11 echo $produk01->harga; // Maaf property 'harga' tidak terdefinisi
12 echo "<br>";
13 echo $produk01->tipe; // Maaf property 'tipe' tidak terdefinisi

```

Sekarang di dalam class Produk terdapat magic method __get(). Method __get() memiliki 1 parameter yang nantinya otomatis diisi PHP dengan nama property pada saat pemanggilan. Dalam contoh di atas, saya memberi nama parameter ini sebagai \$name (baris 3). Anda bisa saja memberikan nama lain.

Method __get() ini akan mengembalikan string "Maaf property '\$name' tidak terdefinisi". Ini saya buat sebagai pengganti pesan error jika property tidak ada di dalam class Produk. Variabel \$name nantinya akan diganti PHP dengan nama property yang dipanggil.

Ketika dijalankan property \$produk01->merek, string "merek" akan menjadi inputan untuk parameter \$name, sehingga hasilnya menjadi "Maaf property 'merek' tidak terdefinisi". Begitu juga dengan hasil pemanggilan dari \$produk01->harga dan \$produk01->tipe. Yang juga sama-sama tidak terdefinisi di dalam class Produk.

Kesimpulannya, jika kita mengakses property yang tidak tersedia atau tidak bisa diakses, PHP akan mencari magic method __get() terlebih dahulu. Jika ada, method tersebut akan dijalankan. Namun jika tidak ada, PHP akan mengembalikan pesan error "Undefined property".

Jika ternyata property itu tersedia, method __get() akan dilewati:

48.magic_method_get_property.php

```

1 <?php
2 class Produk {
3     public $merek = "Logitech";
4

```

```

5  public function __get($name) {
6      return "Maaf property '$name' tidak terdefinisi";
7  }
8 }
9
10 $produk01 = new Produk();
11 echo $produk01->merek; // Logitech
12 echo "<br>";
13 echo $produk01->harga; // Maaf property 'harga' tidak terdefinisi

```

Dalam class Produk saya menambah property \$merek dengan nilai "Logitech". Karena property \$merek terdefinisi di dalam class Produk, maka pemanggilan \$produk01->merek akan menghasilkan nilai "Logitech", bukan lagi pesan dari method __get().

Untuk menguji logika, bisakah anda menjelaskan hasil dari kode program berikut?

49.magic_method_get_property_access.php

```

1 <?php
2 class Produk {
3     private $merek = "Logitech";
4
5     public function __get($name) {
6         return "Maaf property '$name' tidak terdefinisi";
7     }
8 }
9
10 $produk01 = new Produk();
11 echo $produk01->merek; // Maaf property 'merek' tidak terdefinisi
12 echo "<br>";
13 echo $produk01->harga; // Maaf property 'harga' tidak terdefinisi

```

Dalam class Produk masih terdapat property \$merek dengan nilai "Logitech". Namun kenapa di baris 11 pemanggilan \$produk01->merek tidak menghasilkan "Logitech", tapi pesan string dari method __get()?

Kuncinya ada di pengertian dari method __get() itu sendiri, yakni akan dijalankan jika sebuah property tidak ada (**atau tidak bisa diakses**) dari luar class. Dalam kasus ini, property \$merek ada, tapi tidak bisa diakses dari luar class karena dibatasi oleh visibility private. Karena di set sebagai private, property \$merek hanya bisa diakses dari dalam class Produk saja.

Lebih jauh lagi, di dalam method __get() kita bisa membuat sebuah kondisi if untuk menampilkan berbagai pesan tergantung property yang diakses. Berikut contohnya:

50.magic_method_get_if.php

```

1 <?php
2 class Produk {
3     public function __get($name) {
4         if ($name == "merek"){
5             $hasil = "Logitech";
6         }

```

```

7     else if ($name == "harga"){
8         $hasil = 150000;
9     }
10    else if ($name == "tipe"){
11        $hasil = "Mouse";
12    }
13    else {
14        $hasil = "Maaf property '$name' tidak terdefinisi";
15    }
16    return $hasil;
17 }
18 }
19
20 $produk01 = new Produk();
21 echo $produk01->merek;      // Logitech
22 echo "<br>";
23 echo $produk01->harga;      // 150000
24 echo "<br>";
25 echo $produk01->tipe;       // Mouse
26 echo "<br>";
27 echo $produk01->warna;      // Maaf property 'warna' tidak terdefinisi

```

Dalam method `__get()` saya membuat semacam "filter" untuk memeriksa isi parameter `$name`. Terdapat 3 kondisi if yang akan menjalankan kode yang berbeda-beda.

Pada saat dipanggil `$produk01->merek`, string "merek" akan menjadi nilai input untuk parameter `$name`. Karena itu kondisi `if ($name == "merek")` akan menghasilkan nilai true (baris 4). Jika ini terjadi, input string "Logitech" ke variabel `$hasil`.

Selain itu terdapat juga kondisi untuk pemanggilan property `harga` dan `tipe`. Setiap kondisi akan memberikan nilai yang berbeda-beda ke dalam variabel `$hasil`. Jika ternyata property yang dipanggil tidak memenuhi ketiga kondisi ini, variabel `$hasil` akan diisi string "Maaf property '\$name' tidak terdefinisi". Terakhir, kembalikan isi variabel `$hasil` dengan perintah `return`.

Prinsip seperti ini bisa dikembangkan untuk membuat semacam "getter otomatis". Seperti yang pernah kita bahas di dalam bab sebelumnya, **getter** adalah sebutan untuk method yang menampilkan nilai property. Dengan method `__get()`, kita bisa membuat 1 method untuk mewakili banyak getter method:

51.magic_method_getter.php

```

1 <?php
2 class Produk {
3     private $merek = "Logitech";
4     private $harga = 150000;
5     private $tipe = "Mouse";
6
7     public function __get($name) {
8         if ($name == "merek"){
9             $hasil = strtoupper($this->merek);

```

```

10     }
11     else if ($name == "harga"){
12         $hasil = "Rp. ".number_format($this->harga,2,",",".");
13     }
14     else if ($name == "tipe"){
15         $hasil = "Tipe produk: ".$this->tipe;
16     }
17     else {
18         $hasil = "Maaf property '$name' tidak terdefinisi";
19     }
20     return $hasil;
21 }
22 }
23
24 $produk01 = new Produk();
25 echo $produk01->merek;      // LOGITECH
26 echo "<br>";
27 echo $produk01->harga;       // Rp. 150.000,00
28 echo "<br>";
29 echo $produk01->tipe;        // Tipe produk: Mouse
30 echo "<br>";
31 echo $produk01->warna;       // Maaf property 'warna' tidak terdefinisi

```

Kode program ini adalah hasil modifikasi dari kode sebelumnya. Sekarang di dalam class Produk saya sertakan property \$merek, \$harga dan \$tipe dengan nilainya masing-masing.

Ketiga property di set sebagai private agar tidak bisa diakses dari luar class. Konsekuensinya, jika kita ingin mengakses property ini dari luar class, maka harus membuat method getter untuk setiap property, seperti getMerek(), getHarga() dan getTipe().

Ide awal kenapa perlu getter adalah agar kita bisa melalukan sesuatu sebelum menampilkan isi property, misalnya mengubah huruf, men-format angka atau menambah sebuah string. Ini semua bisa dibuat di dalam 1 method __get().

Caranya, di dalam method __get(), kita harus filter terlebih dahulu apa property yang sedang dipanggil. Ini dibuat menggunakan kondisi **if else** untuk parameter \$name. Jika sesuai, kembalikan nilai property menggunakan variabel \$this.

Sebagai contoh, di baris 25 saya mencoba mengakses \$produk01->merek. Namun karena property \$merek di set sebagai private, maka nilainya tidak bisa diakses dari luar class. PHP kemudian akan mencari magic method __get() yang ternyata ada di dalam class Produk.

Di dalam method __get(), parameter \$name akan berisi string "merek", yakni sesuai dengan pemanggilan sebelumnya. String "merek" akan memenuhi kondisi **if (\$name == "merek")** di baris 8. Dengan demikian, blok kondisi if akan dijalankan, dimana variabel \$hasil diisi dengan strtoupper(\$this->merek).

Fungsi strtoupper() saya pakai untuk memperlihatkan bahwa kita bisa melalukan "sesuatu" terhadap property sebagaimana layaknya sebuah method getter. Fungsi strtoupper() akan mengkonversi nilai string menjadi huruf besar.

Hal yang sama juga di terapkan untuk kondisi `if ($name == "harga")` dimana saya menformat tampilan angka dengan fungsi `number_format()` bawaan PHP, plus menambah awalan string "Rp. ". Sedangkan untuk `if ($name == "tipe")`, saya menambah awalan string "Tipe produk: ".

Sebagai pengaman, jika property yang diakses bukan salah satu dari `merek`, `harga` dan `tipe`, variabel `$hasil` akan diisi dengan string "Maaf property '\$name' tidak terdefinisi".

Salah satu keunggulan dari cara membuat method getter seperti ini adalah jika kita punya banyak property yang perlu diproses dengan cara yang sama. Berikut contohnya:

52.magic_method_getter_all.php

```

1  <?php
2  class Produk {
3      private $merek = "Logitech";
4      private $tipe = "Mouse";
5
6      public function __get($name) {
7          if (($name == "merek") || ($name == "tipe")) {
8              $hasil = strtoupper($this->$name);
9          }
10         else {
11             $hasil = "Maaf property '$name' tidak terdefinisi";
12         }
13         return $hasil;
14     }
15 }
16
17 $produk01 = new Produk();
18 echo $produk01->merek;      // LOGITECH
19 echo "<br>";
20 echo $produk01->tipe;        // MOUSE
21 echo "<br>";
22 echo $produk01->warna;       // Maaf property 'warna' tidak terdefinisi

```

Dalam contoh ini saya menggabungkan akses getter untuk property `merek` dan `tipe`. Keduanya diproses dengan kondisi `if` di baris 7. Jika parameter `$name` berisi string `merek` atau `tipe`, perintah `$hasil = strtoupper($this->$name)` akan dijalankan. Bayangkan jika terdapat 10 property yang juga ingin diubah menjadi huruf besar, maka saya cukup menambah kondisi `if` lain, tidak perlu membuat method getter untuk setiap property.

Di balik keunggulannya, membuat getter dengan `__get()` juga memiliki kelemahan. Pertama, penulisan kode program menjadi agak susah dibaca. Kedua, PHP butuh waktu pemrosesan yang sedikit lebih lama pada saat memproses `__get()` dibandingkan jika ditulis menjadi method terpisah. Dan ketiga, mungkin tidak semua orang di tim kita paham cara kerja dari magic method ini, akibatnya butuh waktu untuk menjelaskan seperti apa cara kerja dari `__get()`.

Singkatnya, jika tidak terpaksa saya sarankan tetap buat getter terpisah, bukan memakai

magic method `__get()`. Jika tertarik membaca penjelasan yang lebih teknis, bisa kunjungi [Best practice: PHP Magic Methods __set and __get](#).

Magic Method `__set()`

Mirip seperti `__get()`, magic method `__set()` akan dijalankan pada saat kita berusaha menginput sebuah nilai ke dalam property yang tidak ada (atau tidak bisa diakses) dari sebuah class. Ini adalah *magic method* untuk setter.

Sebelum kita masuk ke pembahasan tentang `__set()`, saya ingin menunjukkan bahwa PHP membolehkan kita mengisi nilai ke sebuah property yang tidak terdefinisi sebelumnya:

53.method_access.php

```

1 <?php
2 class Produk {
3 }
4
5 $produk01 = new Produk();
6 $produk01->merek = "Logitech";
7 echo $produk01->merek; // "Logitech"

```

Di baris 2 – 3 saya membuat class `Produk` tanpa isi apapun. Class `Produk` ini kemudian diinstansiasi ke dalam variabel `$produk01` di baris 5. Lalu saya mengisi string "Logitech" ke dalam property `merek` dari object `$produk01` (baris 6).

Perhatikan bahwa property `merek` sebelumnya **tidak ada** di dalam class `Produk`. Artinya saya mencoba mengisi nilai ke sebuah property "baru" ke dalam sebuah class. PHP mengizinkan hal ini dan tidak tampil pesan error.

Apabila di dalam class `Produk` terdapat method `__set()`, maka proses input property seperti ini akan "dilewatkan" terlebih dahulu ke method `__set()`. Dengan demikian, saya bisa membuat kode program yang menampilkan pesan error apabila seseorang mencoba membuat property baru seperti kasus di atas:

54.magic_method_set.php

```

1 <?php
2 class Produk {
3     public function __set($name, $value) {
4         echo "Maaf property '$name' tidak tersedia dan tidak bisa diisi '$value'";
5     }
6 }
7
8 $produk01 = new Produk();
9 $produk01->merek = "Logitech";
10 echo "<br>";
11 $produk01->harga = 15000;
12 echo "<br>";
13 $produk01->tipe = "Mouse";

```

Hasil kode program:

```
Maaf property 'merek' tidak tersedia dan tidak bisa diisi 'Logitech'
Maaf property 'harga' tidak tersedia dan tidak bisa diisi '15000'
Maaf property 'tipe' tidak tersedia dan tidak bisa diisi 'Mouse'
```

Method `__set()` menerima 2 buah parameter. Parameter pertama dipakai untuk menampung nama property (saya set sebagai `$nama`), dan parameter kedua dipakai untuk menampung nilai yang diinput ke dalam property tersebut (saya set sebagai `$value`).

Sebagai contoh, di baris 9 terdapat perintah `$produk01->merek = "Logitech"`. Ini berarti saya mencoba mengisi nilai "Logitech" ke dalam property `merek`. Di dalam class `Produk` tidak ada property `$merek` sehingga PHP akan menjalankan *magic method* `__set()`.

Di dalam method `__set()`, parameter `$nama` akan berisi string "merek", yakni nama property yang dipanggil, sedangkan parameter `$value` akan berisi string "Logitech", yakni nilai yang diinput ke dalam property `merek`. Kedua nilai ini kemudian saya tampilkan ke dalam pesan `echo`. Hasilnya berupa teks "Maaf property 'merek' tidak tersedia dan tidak bisa diisi 'Logitech'".

Hal yang sama juga terjadi untuk perintah `$produk01->harga = 15000` di baris 11 dan perintah `$produk01->tipe = "Mouse"` di baris 13. Kedua perintah ini sama-sama mencoba membuat dan mengisi property baru ke dalam class `Produk`. Karena method `__set()` saya tulis seperti ini, maka setiap kali kode program mencoba membuat property baru, akan gagal.

Alternatif lain, saya bisa memanfaatkan method `__set()` untuk memproses nilai property baru sebelum diinput ke dalam class `Produk`:

55.magic_method_set_uppercase.php

```
1 <?php
2 class Produk {
3     public function __set($name, $value) {
4         $this->$name = strtoupper($value);
5     }
6 }
7
8 $produk01 = new Produk();
9 $produk01->merek = "Logitech";
10 $produk01->harga = 15000;
11 $produk01->tipe = "Mouse";
12
13 echo $produk01->merek;    // LOGITECH
14 echo "<br>";
15 echo $produk01->harga;    // 15000
16 echo "<br>";
17 echo $produk01->tipe;    // MOUSE
```

Di sini method `__set()` saya rancang untuk "melewatkannya" proses pembuatan property baru.

Sebagai contoh, di baris 9 terdapat perintah `$produk01->merek = "Logitech"`. Proses ini akan di teruskan ke dalam method `__set()` dimana parameter `$nama` akan berisi string "merek" dan parameter `$value` akan berisi string "Logitech".

Di baris 4, perintah `$this->$name` saya pakai untuk membuat property baru ke dalam class `Produk`, namun nilainya dilewatkan terlebih dahulu ke dalam function `strtoupper($value)`. Hasilnya, string "Logitech" akan di konversi ke huruf kapital.

Sekarang setiap kali ada percobaan membuat property baru di luar class, isi string akan dikonversi menjadi huruf besar, kecuali untuk nilai angka yang tidak berpengaruh apa-apa.

Sampai di sini saya yakin anda sudah bisa memahami fungsi dari method `__set()`, yakni menangani proses pembuatan property baru. Namun jika di dalam class `Produk` sudah ada property tersebut, maka method `__set()` tidak akan dipanggil oleh PHP:

56.magic_method_property.php

```

1 <?php
2 class Produk {
3     public $merek;
4     public function __set($name, $value) {
5         $this->$name = strtoupper($value);
6     }
7 }
8
9 $produk01 = new Produk();
10 $produk01->merek = "Logitech";
11 $produk01->harga = 15000;
12 $produk01->tipe = "Mouse";
13
14 echo $produk01->merek;    // Logitech
15 echo "<br>";
16 echo $produk01->harga;    // 15000
17 echo "<br>";
18 echo $produk01->tipe;    // MOUSE

```

Di awal class `Produk` sekarang terdapat property `$merek` (baris 3). Akibatnya, string "Logitech" tidak akan dikonversi menjadi huruf besar karena proses `$produk01->merek = "Logitech"` di baris 10 tidak akan dilewatkan ke method `__set()`.

Sekali lagi, ini karena fungsi method `__set()` yang hanya akan dipanggil pada saat kita berusaha menginput sebuah nilai ke dalam property yang tidak ada (atau tidak bisa diakses) dari sebuah class. Karena pengertian ini, method `__set()` juga bisa dipakai untuk membuat sebuah method **setter universal** seperti contoh berikut:

57.magic_method_setter.php

```

1 <?php
2 class Produk {
3     private $merek;

```

```

4  private $harga;
5
6  public function __set($name, $value) {
7      if ($name == "merek"){
8          if (is_string($value)) {
9              $this->merek = $value;
10         }
11     else {
12         echo "Error: merek harus berbentuk string <br>";
13     }
14 }
15
16 else if ($name == "harga"){
17     if (is_int($value)) {
18         $this->harga = $value;
19     }
20     else {
21         echo "Error: harga harus berbentuk angka <br>";
22     }
23 }
24
25 else {
26     echo "Maaf property '$name' tidak tersedia";
27 }
28 }
29 }
30
31 $produk01 = new Produk();
32 $produk01->merek = "Logitech";
33 $produk01->harga = 15000;
34
35 echo "<pre>";
36 print_r ($produk01);
37 echo "</pre>";

```

Hasil kode program:

```

Produk Object
(
    [merek:Produk:private] => Logitech
    [harga:Produk:private] => 15000
)

```

Kode program di atas sedikit panjang karena terdapat proses validasi property.

Di awal class Produk saya membuat property \$merek dan \$harga yang di-set sebagai private. Karena batasan ini, kedua property tidak bisa diakses dari luar class Produk. Setiap percobaan mengisi kedua property akan berakhir di method __set(). Kita bisa memanfaatkan method __set() ini untuk membuat proses validasi.

Kode program di baris 7 – 14 merupakan proses validasi untuk property \$merek. Isinya sama seperti validasi yang kita buat pada materi tentang setter dan getter (pada bab sebelumnya). Di

sini terdapat proses pemeriksaan apakah property \$merek diisi dengan string atau tipe data lain. Jika iya, jalankan perintah `$this->merek = $value`. Namun jika hasilnya false, tampilkan pesan "Error: merek harus berbentuk string
".

Baris 16 – 23 adalah kode validasi untuk property \$harga. Di sini saya ingin memeriksa apakah nilai yang diinput sudah berbentuk angka menggunakan kondisi `if (is_int($value))`.

Di akhir kode program saya menggunakan perintah `print_r ($produk01)` untuk melihat isi dari object `$produk01`.

Mari kita uji validasi ini dengan data yang salah:

58.magic_method_setter_2.php

```

1  <?php
2  class Produk {
3      // isi class produk sama seperti sebelumnya...
4      // isi class produk sama seperti sebelumnya...
5  }
6
7  $produk01 = new Produk();
8  $produk01->merek = 5000;
9  $produk01->harga = "mahal";
10 $produk01->tipe = "Mouse";
11
12 echo "<pre>";
13 print_r ($produk01);
14 echo "</pre>";

```

Hasil kode program:

```

Error: merek harus berbentuk string
Error: harga harus berbentuk angka
Maaf property 'tipe' tidak tersedia

```

```

Produk Object
(
    [merek:Produk:private] =>
    [harga:Produk:private] =>
)

```

Agar menghemat tempat, class `Produk` tidak lagi saya tampilkan (isinya sama seperti kode sebelumnya).

Di baris 8 saya mencoba input nilai `5000` ke property `merek`. Hasilnya, tampil pesan "Error: merek harus berbentuk string". Begitu juga di baris 9 ketika saya coba menginput string "`mahal`" ke property `harga`, hasilnya tampil teks "Error: harga harus berbentuk angka". Di baris 10 saya coba menginput nilai baru ke property yang sebelumnya tidak ada, hasilnya tampil pesan "Maaf property 'tipe' tidak tersedia".

Percobaan ini menunjukkan bahwa proses validasi yang kita rancang di dalam method `__set()`

sudah berjalan sebagaimana mestinya. Ini adalah semacam method setter *universal*. Jika terdapat 10 property bertipe string yang harus diperiksa, kita tidak perlu membuat 10 method setter untuk setiap property, tapi cukup memeriksanya di method `__set()`.

Namun kelemahan yang ada di method `__get()` juga berlaku untuk magic method `__set()`, dimana kode program menjadi lebih susah dibaca, butuh waktu pemrosesan yang sedikit lebih lama, dan jika ini dibuat untuk kerja tim, setiap orang harus paham cara kerja dari magic method PHP.

Menurut saya, salah satu penggunaan method `__set()` yang cukup "masuk akal" adalah untuk mencegah seseorang membuat property baru ke dalam class. Selain itu, sebaiknya tetap gunakan setter terpisah.

Magic Method `__call()`

Magic method `__call()` adalah method khusus yang dijalankan PHP ketika kita mengakses method yang tidak tersedia (atau tidak bisa di akses) dari sebuah class.

Secara bawaan, PHP akan menampilkan pesan error ketika kita mengakses method yang sebenarnya tidak ada:

59.method_calling_error.php

```

1 <?php
2 class Produk {
3 }
4
5 $produk01 = new Produk();
6 $produk01->tambah(3, 7, 8);

```

Hasil kode program:

```
Fatal error: Uncaught Error: Call to undefined method Produk::tambah()
```

Di baris 6, saya mencoba memanggil method `tambah()` dari object class `Produk`, hasilnya tampil pesan error karena method `tambah()` tidak terdefinisi .

Namun jika di dalam class itu terdapat method `__call()`, PHP akan memproses pemanggilan method yang "salah" ini ke dalam method `__call()`:

60.magic_method_call.php

```

1 <?php
2 class Produk {
3     public function __call($name,$arguments){
4         echo "Maaf method $name tidak tersedia";
5     }
6 }
7
8 $produk01 = new Produk();

```

```
9 $produk01->tambah(3, 7, 8); // Maaf method tambah tidak tersedia
```

Sekarang di dalam class Produk sudah terdapat method __call(). Method __call() menerima 2 parameter, yakni nama dari method yang dipanggil, serta argument dari pemanggilan method tersebut.

Di baris 9 saya memanggil method tambah() yang memang tidak ada di dalam class Produk. Karena pemanggilan ini, method __call() akan menerima string "tambah" di dalam parameter \$name yang selanjutnya saya input ke pesan kesalahan "Maaf method \$name tidak tersedia".

Parameter kedua untuk method __call() berisi argument pemanggilan method dalam bentuk array. Berikut contoh cara menampilkannya:

61.magic_method_call_args.php

```
1 <?php
2 class Produk {
3     public function __call($name,$arguments){
4         echo "Maaf method $name tidak tersedia <br>";
5         print_r($arguments);
6     }
7 }
8
9 $produk01 = new Produk();
10 $produk01->tambah(3, 7, 8);
```

Hasil kode program:

```
Maaf method tambah tidak tersedia
Array ( [0] => 3 [1] => 7 [2] => 8 )
```

Di dalam class __call() saya menambahkan perintah print_r(\$arguments). Ini dipakai untuk memperlihatkan apa yang diterima oleh parameter \$arguments, yakni sebuah array.

Karena di baris 10 saya menulis \$produk01->tambah(3, 7, 8), di dalam method __call(\$name, \$arguments), parameter \$name akan berisi string "tambah", sedangkan parameter \$arguments akan berisi array [3, 7, 8], yakni seluruh argumen yang ada saat pemanggilan.

Seandainya saya memanggil method \$produk01->setMerek("Xiaomi","Vivo","Oppo"), maka parameter \$name akan berisi string "setMerek", dan parameter \$arguments akan berisi array ["Xiaomi", "Vivo", "Oppo"].

Supaya lebih rapi, parameter \$arguments ini bisa kita proses menggunakan fungsi implode() PHP:

62.magic_method_callImplode.php

```
1 <?php
2 class Produk {
3     public function __call($name,$arguments){
4         echo "Maaf method $name dengan argument ". implode(", ",$arguments);
```

```

5     echo " tidak tersedia <br>";
6 }
7 }
8
9 $produk01 = new Produk();
10 $produk01->tambah(3, 7, 8);
11 $produk01->setMerek("Xiaomi", "Vivo", "Oppo");

```

Hasil kode program:

```

Maaf method tambah dengan argument 3, 7, 8 tidak tersedia
Maaf method setMerek dengan argument Xiaomi, Vivo, Oppo tidak tersedia

```

Fungsi `implode()` di baris 4 berguna untuk proses konversi array menjadi string. Fungsi ini tidak berhubungan dengan method `__call()`, hanya untuk mempercantik tampilan array.

Sama seperti magic method lainnya, jika di dalam class sudah terdapat method yang "seharusnya", maka magic method `__call()` ini tidak akan dipanggil:

63.magic_method_call_normal.php

```

1 <?php
2 class Produk {
3     public function tambah($a,$b,$c){
4         echo "Hasil = ".($a + $b + $c). "<br>";
5     }
6
7     public function __call($name,$arguments){
8         echo "Maaf method $name dengan argument ". implode(", ",$arguments);
9         echo " tidak tersedia <br>";
10    }
11 }
12
13 $produk01 = new Produk();
14 $produk01->tambah(3, 7, 8);
15 $produk01->setMerek("Xiaomi", "Vivo", "Oppo");

```

Hasil kode program:

```

Hasil = 18
Maaf method setMerek dengan argument Xiaomi, Vivo, Oppo tidak tersedia

```

Di awal class `Produk` saya mendefinisikan method `tambah()`, akibatnya pemanggilan di baris 14 tidak lagi di proses oleh method `__call()`, tapi langsung ke method `tambah()` yang ada di class `Produk`.

Seandainya method `tambah()` saya set menjadi `private`, maka pemanggilan dari luar class juga akan diproses oleh method `__call()`.

Sama seperti `__get()` dan `__set()`, method `__call()` juga bisa dibuat menjadi method universal dengan menambahkan beberapa kondisi if. Ini memang akan menambah

kompleksitas kode program, tapi dalam beberapa situasi mungkin bisa berguna.

Magic Method `__callStatic()`

Method `__callStatic()` berfungsi mirip seperti `__call()`, tapi secara khusus dipakai untuk *static method*. Magic method `__callStatic()` adalah method khusus yang dijalankan PHP ketika kita mengakses method static yang tidak tersedia (atau tidak bisa diakses) dari sebuah class.

Contoh berikut memperlihatkan bahwa method `__call()` tidak akan dijalankan jika yang kita akses adalah static method:

64. static_method_calling_error.php

```

1 <?php
2 class Produk {
3     public function __call($name,$arguments){
4         echo "Maaf method $name dengan argument ". implode(", ",$arguments);
5         echo " tidak tersedia <br>";
6     }
7 }
8
9 Produk::tambah(3, 7, 8);

```

Hasil kode program:

```
Fatal error: Uncaught Error: Call to undefined method Produk::tambah()
```

Untuk bisa "menangkap" static method ini, ubah nama method `__call()` menjadi `__callStatic()`:

65. magic_method_callstatic.php

```

1 <?php
2 class Produk {
3     public static function __callStatic($name,$arguments){
4         echo "Maaf method static $name dengan argument ". implode(", ",$arguments);
5         echo " tidak tersedia <br>";
6     }
7 }
8
9 Produk::tambah(3, 7, 8);

```

Hasil kode program:

```
Maaf method static tambah dengan argument 3, 7, 8 tidak tersedia
```

Sekarang, pemanggilan static method yang tidak ada di dalam class `Produk` akan dilewatkan terlebih dahulu dilewatkan ke dalam method `__callStatic()`.

Agar lebih lengkap, kita bisa membuat method `__call()` dan `__callStatic()` sekaligus:

66.magic_method_call_callstatic.php

```

1 <?php
2 class Produk {
3
4     public function __call($name,$arguments){
5         echo "Maaf method $name dengan argument ". implode(", ",$arguments);
6         echo " tidak tersedia <br>";
7     }
8
9     public static function __callStatic($name,$arguments){
10        echo "Maaf method static $name dengan argument ". implode(", ",$arguments);
11        echo " tidak tersedia <br>";
12    }
13 }
14
15 $produk01 = new Produk();
16 $produk01->tambah(3, 7, 8);
17 // Maaf method tambah dengan argument 3, 7, 8 tidak tersedia
18
19 Produk::tambah(3, 7, 8);
20 // Maaf method static tambah dengan argument 3, 7, 8 tidak tersedia

```

Kedua method memang relatif jarang dipakai, tapi bisa menjadi solusi untuk kasus-kasus yang spesifik.

Magic Method __isset()

Magic method `__isset()` akan dijalankan ketika sebuah property yang tidak ada (atau tidak bisa diakses) di periksa dengan fungsi `isset()` atau `empty()`. Jika anda pernah membuat kode program untuk proses validasi form, tentu tidak asing dengan kedua fungsi ini.

Fungsi `isset()` dipakai untuk memeriksa apakah sebuah variabel terdefinisi dan berisi sesuatu. Sedangkan fungsi `empty()` dipakai untuk memeriksa apakah sebuah variabel memiliki nilai yang dianggap kosong seperti 0, "" (string kosong), [] (array kosong), atau variabel tersebut diisi nilai NULL.

Fungsi `isset()` atau `empty()` bisa juga kita pakai untuk memeriksa property dari sebuah object:

67.property_isset_empty

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4     public $stok;
5     public $tipe = "";
6 }
7
8 $produk01 = new Produk();
9 var_dump(isset($produk01->merek)); // bool(true)
10 echo "<br>";

```

```

11 var_dump(isset($produk01->stok));      // bool(false)
12 echo "<br>";
13 var_dump(isset($produk01->tipe));       // bool(true)
14 echo "<br>";
15
16 var_dump(empty($produk01->merek));     // bool(false)
17 echo "<br>";
18 var_dump(empty($produk01->stok));       // bool(true)
19 echo "<br>";
20 var_dump(empty($produk01->tipe));       // bool(true)

```

Property \$merek dari class Produk berisi string "Sony", sehingga fungsi `isset()` akan mengembalikan nilai **true** karena property ini terdefinisi dan berisi sesuatu. Sedangkan fungsi `empty()` mengembalikan nilai **false** karena property \$merek tidak kosong.

Property \$stok dari class Produk tidak berisi apa-apa, sehingga fungsi `isset()` akan mengembalikan nilai **false** dan fungsi `empty()` mengembalikan nilai **true**.

Property \$tipe dari class Produk berisi string "" (string kosong), sehingga fungsi `isset()` akan mengembalikan nilai **true** karena property ini terdefinisi dan berisi sesuatu, meskipun itu adalah string kosong. Sedangkan fungsi `empty()` juga mengembalikan nilai **true** karena property \$tipe dianggap berisi nilai kosong.

Bagaimana jika ketiga property ini saya ubah menjadi protected? Berikut hasilnya:

68.property_isset_empty_protected.php

```

1 <?php
2 class Produk {
3     protected $merek = "Sony";
4     protected $stok;
5     protected $tipe = "";
6 }
7
8 $produk01 = new Produk();
9 var_dump(isset($produk01->merek));      // bool(false)
10 echo "<br>";
11 var_dump(isset($produk01->stok));       // bool(false)
12 echo "<br>";
13 var_dump(isset($produk01->tipe));       // bool(false)
14 echo "<br>";
15
16 var_dump(empty($produk01->merek));     // bool(true)
17 echo "<br>";
18 var_dump(empty($produk01->stok));       // bool(true)
19 echo "<br>";
20 var_dump(empty($produk01->tipe));       // bool(true)

```

Hasilnya, semua fungsi `isset()` mengembalikan nilai false karena property tidak bisa diakses sehingga dianggap tidak ada. Sedangkan fungsi `empty()` mengembalikan nilai true karena menganggap property yang tidak bisa diakses ini adalah kosong. Hal yang sama juga akan

terjadi jika property di set sebagai private.

Kembali ke method `__isset()`, method ini akan dipanggil ketika kita mengecek property yang tidak ada atau tidak bisa diakses. Salah satu tujuannya adalah untuk memeriksa apakah property tersebut benar-benar tidak ada atau hanya tidak bisa diakses dari luar class.

Berikut contoh kasusnya:

69.magic_method_isset.php

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4     protected $stok = 9;
5     private $tipe = "Televisi";
6
7     public function __isset($name) {
8         echo "Apakah property '$name' ada? ";
9         return isset($this->$name);
10    }
11 }
12
13 $produk01 = new Produk();
14 var_dump(isset($produk01->merek));
15 echo "<br>";
16 var_dump(isset($produk01->stok));
17 echo "<br>";
18 var_dump(isset($produk01->tipe));
19 echo "<br>";
20 var_dump(isset($produk01->warna));

```

Hasil kode program:

```

bool(true)
Apakah property 'stok' ada? bool(true)
Apakah property 'tipe' ada? bool(true)
Apakah property 'warna' ada? bool(false)

```

Sekarang di dalam class `Produk` terdapat tambahan method `__isset()`. Method ini menerima 1 argument berupa nama dari property yang diperiksa (saya simpan ke dalam variabel `$name`).

Di dalamnya terdapat perintah `echo` untuk menampilkan teks "Apakah property '\$name' ada?" beserta perintah `return isset($this->$name)`. Perintah return ini mengembalikan hasil isset **dari dalam class**, sehingga akan berdampak ke pemanggilan fungsi isset dari luas class (penjelasannya sesaat lagi).

Perhatikan juga bahwa class produk memiliki 3 property yang masing-masing di set dengan hak akses yang berbeda-beda.

Di baris 14 saya memeriksa `isset($produk01->merek)`. Hasilnya **true** karena di dalam class `Produk` memang terdapat property `merek` yang berisi string "Sony". Karena property `merek` bisa

diakses dari luar (public), maka method `__isset()` tidak akan dijalankan.

Di baris 16 saya memeriksa `isset($produk01->stok)`. Perintah ini seharusnya menghasilkan nilai **false** karena property `stok` tidak bisa diakses dari luar class `Produk` (di set sebagai `protected`). Namun karena di dalam class `Produk` terdapat method `__isset()`, PHP akan menjalankan magic method ini.

Hasilnya, akan tampil teks "Apakah property 'stok' ada?", dan mengembalikan nilai **true**. Artinya, property `stok` ada tapi tidak bisa diakses dari luar. Hal yang sama juga terjadi untuk perintah `isset($produk01->tipe)` di baris 18 karena property `tipe` saya set sebagai `private`.

Terakhir di baris 20 perintah `isset($produk01->warna)` menghasilkan nilai **false**. Ini berarti property `warna` memang tidak ada di dalam class `Produk`.

Magic method `__isset()` juga akan dipanggil jika property diperiksa dengan fungsi `empty()`. Berikut contohnya:

70.magic_method_isset_empty.php

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4     protected $stok = 9;
5     private $tipe = "Televisi";
6
7     public function __isset($name) {
8         echo "Apakah property '$name' kosong? ";
9         var_dump(empty($this->$name));
10    }
11 }
12
13 $produk01 = new Produk();
14 var_dump(empty($produk01->merek));
15 echo "<br>";
16 empty($produk01->stok);
17 echo "<br>";
18 empty($produk01->tipe);
19 echo "<br>";
20 empty($produk01->warna);

```

Hasil kode program:

```

bool(false)
Apakah property 'stok' kosong? bool(false)
Apakah property 'tipe' kosong? bool(false)
Apakah property 'warna' kosong? bool(true)

```

Untuk contoh dengan `empty()` ini, proses `var_dump()` saya tempatkan di dalam method `__isset()` agar kita bisa langsung melihat dari dalam class. Jika tanpa method `__isset()`, pemanggilan `empty($produk01->stok)` dan `empty($produk01->tipe)` akan menghasilkan boolean `true` karena kedua property tidak bisa diakses dari luar class.

Trik seperti ini sebenarnya relatif jarang dipakai karena jika sebuah property di set sebagai `protected` atau `private` maka itu memang tidak untuk diakses dari luar class.

Magic Method `__unset()`

Magic method `__unset()` akan dijalankan ketika sebuah property yang tidak ada (atau tidak bisa diakses) dihapus dengan fungsi `unset()`.

Kita memang belum membahas efek dari fungsi `unset()` untuk property. Fungsi `unset()` bisa dipakai untuk menghapus property dari sebuah object. Berikut contoh prakteknya:

71.property_unset.php

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4 }
5
6 $produk01 = new Produk();
7
8 echo "<pre>";
9 print_r($produk01);
10 echo "</pre>";
11
12 unset($produk01->merek);
13
14 echo "<pre>";
15 print_r($produk01);
16 echo "</pre>";

```

Hasil kode program:

```

Produk Object
(
    [merek] => Sony
)
Produk Object
(
)

```

Class `Produk` memiliki sebuah property `$merek` yang berisi string "Sony". Class ini kemudian saya instansiasi ke dalam object `$produk01`. Hasil dari fungsi `print_r()` di baris 9 memperlihatkan isi dari object `$produk01`, yakni sebuah property `$merek` sebagaimana isi class `Produk`.

Selanjutnya di baris 12 terdapat kode `unset($produk01->merek)`. Perintah ini akan menghapus property `$merek` dari object `$produk01`. Hasil dari `print_r ($produk01)` mengkonfirmasi bahwa property `$merek` sudah tidak ada lagi di dalam object `$produk01`.

Bagaimana jika kita menghapus property yang memang tidak ada?

72.property_unset_delete.php

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4 }
5
6 $produk01 = new Produk();
7
8 echo "<pre>";
9 print_r($produk01);
10 echo "</pre>";
11
12 unset($produk01->stok);
13
14 echo "<pre>";
15 print_r($produk01);
16 echo "</pre>";

```

Hasil kode program:

```

Produk Object
(
    [merek] => Sony
)
Produk Object
(
    [merek] => Sony
)

```

Di baris 12 saya mencoba menghapus property \$stok dari object \$produk01. Property ini memang tidak ada di dalam class Produk, namun perintah `unset($produk01->stok)` juga tidak menghasilkan pesan error.

Magic method `__unset()` bisa jadi solusi jika kita ingin menampilkan pesan kesalahan ketika sebuah property yang tidak ada dihapus menggunakan perintah `unset()`. Berikut contoh penerapannya:

73.magic_method_unset.php

```

1 <?php
2 class Produk {
3     public $merek = "Sony";
4
5     public function __unset($name){
6         echo "Maaf, property $name tidak ada / tidak bisa diakses";
7     }
8 }
9
10 $produk01 = new Produk();
11 unset($produk01->stok);

```

Hasil kode program:

```
Maaf, property stok tidak ada / tidak bisa diakses
```

Method `__unset()` menerima 1 nilai argument, yakni nama property yang akan dihapus.

Dengan cara ini, kita bisa menampilkan pesan error jika property yang dihapus ternyata tidak ada di dalam class `Produk`.

74.magic_method_unset_private.php

```
1 <?php
2 class Produk {
3     private $merek = "Sony";
4
5     public function __unset($name){
6         echo "Maaf, property $name tidak ada / tidak bisa diakses";
7     }
8 }
9
10 $produk01 = new Produk();
11 unset($produk01->merek);
```

Hasil kode program:

```
Maaf, property merek tidak ada / tidak bisa diakses
```

Sekarang property `merek` saya set menjadi private, akibatnya perintah `unset($produk01->merek)` di luar class tidak menemukan property `merek` dan akan menjalankan magic method `__unset()`.

Method Overloading

Di dalam PHP, seluruh magic method yang sudah kita bahas ini dikenal dengan istilah **overloading**, yakni membuat property dan method secara dinamis (dimana property dan method tersebut tidak ada sebelumnya).

Lebih jauh lagi, method `__get()`, `__set()`, `__isset()` dan `__unset()` dipakai untuk membuat **property overloading**. Sedangkan method `__call()` dan `__callStatic()` dipakai untuk membuat **method overloading**.

Istilah overloading ini sering membuat bingung, karena di bahasa pemrograman object lain seperti **Java** atau **C++**, method overloading punya pengertian lain.

Dalam bahasa Java, *method overloading* adalah sebutan untuk nama method yang sama tapi dengan jenis tipe data parameter yang berbeda. Berikut contoh method overloading dalam bahasa Java:

```
tambah(int a, int b)
tambah(float a, float b)
```

Meskipun memiliki nama yang sama, kedua method `tambah()` di atas berbeda satu sama lain. Teknik *method overloading* seperti ini hanya tersedia dalam bahasa pemrograman dengan aturan tipe data yang ketat (*strongly typed language*) seperti Java atau C++.

PHP adalah *loosely typed language* dimana sebuah variabel bisa menampung tipe data apa saja (tidak harus ditentukan terlebih dahulu seperti Java). Dengan demikian, konsep *method overloading* seperti ini tidak ada di dalam PHP.

Jadi method overloading di PHP **tidak sama** pengertiannya dengan method overloading dalam bahasa Java (dan pemrograman object lain). Ini membuat PHP sering di kritik karena dianggap "salah tafsir" tentang method overloading.

Pengertian method overloading yang banyak dipahami adalah yang seperti di Java, dimana kita membuat nama method yang sama tapi dengan tipe parameter yang berbeda. Namun PHP memiliki pengertian method overloading sendiri, yakni membuat property dan method secara dinamis.

3.6. Perbandingan Object

Materi kali ini akan membahas operasi perbandingan antar object di dalam PHP. Seperti yang kita ketahui, PHP memiliki 2 buah operasi perbandingan, yaitu operator "`==`" dan "`==`". Kedua operator ini memiliki perlakuan yang berbeda.

Pertama, kita akan lihat operator "sama dengan" (`==`) terlebih dahulu:

75.perbandingan_object.php

```

1 <?php
2 class Laptop {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8 }
9
10 $produk01 = new Laptop('Asus');
11 $produk02 = new Laptop('Asus');
12
13 if ($produk01 == $produk02) {
14     echo "Kedua object sama";
15 }
16 else {
17     echo "Kedua object tidak sama";
18 }
```

Hasil kode program:

Kedua object sama

Di sini saya membuat class Laptop dengan 1 property merek dan sebuah constructor untuk mengisi property merek tersebut. Class Laptop kemudian di instansiasi ke dalam variabel \$produk01 dan \$produk02. Kedua variabel diisi dengan object Laptop yang sama, yakni dengan argument berupa string "Asus".

Di baris 13 saya membandingkan variabel \$produk01 dan \$produk02 memakai operator "sama dengan" (==). Hasilnya tampil teks "Kedua object sama", yang berarti operasi perbandingan keduanya bernilai **true**.

Variabel \$produk01 dengan \$produk02 dianggap sama karena berisi object dari class yang sama (class Laptop) dan juga memiliki nilai yang sama, yakni property merek yang berisi string "Asus". Jika pada saat instansiasi \$produk02 saya isi dengan nilai argument yang berbeda, operasi perbandingan akan menghasilkan nilai **false**:

76.perbandingan_object_2.php

```

1 <?php
2 class Laptop {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8 }
9
10 $produk01 = new Laptop('Asus');
11 $produk02 = new Laptop('Acer');
12
13 if ($produk01 == $produk02) {
14     echo "Kedua object sama";
15 }
16 else {
17     echo "Kedua object tidak sama";
18 }
```

Hasil kode program:

Kedua object tidak sama

Karena variabel \$produk02 di instansiasi dengan perintag new Laptop('Acer'), maka object \$produk01 dan \$produk02 sudah dianggap berbeda.

Sekarang kita akan lihat operasi perbandingan kedua, yakni operator "identik dengan" (===). Berikut contohnya:

77.perbandingan_object_identik.php

```

1 <?php
2 class Laptop {
```

```

3  private $merek;
4
5  public function __construct($merek){
6      $this->merek = $merek;
7  }
8 }
9
10 $produk01 = new Laptop('Asus');
11 $produk02 = new Laptop('Asus');
12
13 if ($produk01 === $produk02) {
14     echo "Kedua object sama!";
15 }
16 else {
17     echo "Kedua object tidak sama!";
18 }
```

Hasil kode program:

Kedua object tidak sama!

Di sini object \$produk01 dan \$produk02 sama-sama di instansiasi dengan string "Asus", namun hasilnya adalah "Kedua object tidak sama!".

Operasi perbandingan "identik dengan" (===) hanya akan bernilai true **jika kedua object merujuk ke object yang sama, bukan hanya berisi nilai yang sama**. Untuk bisa memahami kalimat ini, kita harus membahas tentang perbedaan *copy by value* dan *copy by reference*.

Di buku **PHP Uncover** saya pernah membahas perbedaan antara *assignment by value* dengan *assignment by reference*. Konsep ini sangat berhubungan dengan pembahasan kita.

Assignment by value adalah proses input variabel dengan men-copy nilai (value) dari variabel lain. Berikut contohnya:

78.assignment_by_value.php

```

1 <?php
2 $a = 5;
3 $b = $a;
4
5 $a = 10;
6
7 echo $a; // 10
8 echo $b; // 5
```

Di awal kode program, variabel \$a saya isi dengan angka 5, kemudian terdapat operasi \$b = \$a. Operasi ini bisa dibaca sebagai "**copy nilai** yang ada di dalam variabel \$a ke dalam variabel \$b".

Dengan perintah ini, variabel \$a dan \$b adalah variabel yang berbeda. Perubahan nilai di variabel \$a menjadi 10 tidak berpengaruh apa-apa dengan nilai yang tersimpan di variabel \$b.

Secara teknis, variabel \$a dan \$b menggunakan alamat memory penyimpanan yang berbeda.

Berikutnya, mari kita lihat contoh dari **assignment by reference**:

79.assignment_by_reference.php

```

1 <?php
2 $a = 5;
3 $b =& $a;
4
5 $a = 10;
6
7 echo $a; // 10
8 echo $b; // 10

```

Assignment by reference adalah proses pengisian **alamat memory** sebuah variabel ke variabel lain. Lokasi alamat memory ini disebut juga dengan "reference".

Di baris 2 terdapat operasi \$a = 5. Perintah ini menginstruksikan PHP untuk menyediakan sebuah ruang memory untuk menyimpan angka 5. Ruang memory ini secara fisik ada di RAM komputer dan memiliki alamat atau nomor urut. Kita anggap angka 5 ini disimpan di alamat memory RAM nomor urut 5194.

Di baris 3 terdapat operasi \$b =& \$a. Perhatikan bahwa operator yang dipakai adalah "=&", ini menginstruksikan PHP untuk men-copy *alamat memory* milik variabel \$a ke dalam variabel \$b, (yakni alamat 5194). Hasilnya, variabel \$b juga akan berisi angka 5 karena baik variabel \$a dan variabel \$b sama-sama merujuk ke ruang memory yang sama (5194).

Ketika nilai salah satu variabel diubah, misalnya isi variabel \$a ditimpas menjadi 10, maka isi variabel \$b juga ikut berubah. Hal ini berbeda dengan *assignment by value* dimana variabel \$a dan \$b memiliki alamat memory terpisah satu sama lain.

Di sini kita bisa lihat bahwa operator assignment "biasa", yakni tanda sama dengan " = " dipakai untuk copy by value, sedangkan operator "=&" dipakai untuk proses copy by reference.

Namun jika dipakai untuk object, operator assignment " = " akan menjadi **copy by reference**. Berikut contoh prakteknya:

80.object_copy_by_reference.php

```

1 <?php
2 class Laptop {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8
9     public function cekMerek(){
10        return $this->merek;
11    }

```

```

12
13     public function setMerek($merek){
14         $this->merek = $merek;
15     }
16 }
17
18 $produk01 = new Laptop('Asus');
19 $produk02 = $produk01;
20
21 echo $produk01->cekMerek()."<br>"; // Asus
22 echo $produk02->cekMerek()."<br>"; // Asus
23
24 $produk02->setMerek('Dell')."<br>";
25
26 echo $produk01->cekMerek()."<br>"; // Dell
27 echo $produk02->cekMerek()."<br>"; // Dell

```

Untuk class Produk saya tambah 2 method baru, yakni cekMerek() dan setMerek(). Kedua method ini tidak lain adalah getter dan setter untuk property merek.

Di baris 18 saya membuat instansiasi class Produk ke dalam variabel \$produk01. String awal yang diinput adalah 'Asus'.

Di baris 19 terdapat perintah \$produk02 = \$produk01, perintah ini berarti saya ingin men-copy object \$produk01 ke dalam variabel \$produk02. Artinya, variabel \$produk01 dan \$produk02 sama-sama berisi object dari class Laptop. Perintah echo di baris 21 dan 22 mengkonfirmasi hal ini, dimana method cekMerek() sama-sama berisi string 'Asus'.

Kemudian di baris 24 saya mengubah nilai property merek dari \$produk02 dengan perintah \$produk02->setMerek('Dell'). Perhatikan hasil yang di dapat di baris 26, ternyata property merek milik \$produk01 juga ikut berubah, padahal yang kita tukar hanya property milik \$produk02.

Hal ini bisa terjadi karena operasi di baris 19, yakni perintah \$produk02 = \$produk01 adalah **copy by reference**. Perintah tersebut akan men-copy alamat memory (*reference*) dari object \$produk01 ke dalam variabel \$produk02. Artinya kedua variabel akan merujuk ke object Laptop yang sama.

Operasi perbandingan 'identik dengan' (===) yang sudah kita coba sebelumnya, hanya akan bernilai **true** jika kedua variabel berisi referensi ke object yang sama:

81.perbandingan_object_identik_2.php

```

1 <?php
2 class Laptop {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8 }

```

```

9
10 $produk01 = new Laptop('Asus');
11 $produk02 = $produk01;
12
13 if ($produk01 === $produk02) {
14     echo "Kedua object sama!";
15 }
16 else {
17     echo "Kedua object tidak sama!";
18 }
```

Hasil kode program:

Kedua object sama!

Kesimpulannya, operasi perbandingan "identik dengan" (==) hanya akan bernilai **true** jika kedua variabel **berisi referensi ke object yang sama**. Sedangkan operasi perbandingan "sama dengan" (==) akan bernilai **true** jika kedua variabel berisi nilai object yang sama atau berisi referensi ke object yang sama.

3.7. Object Clone

Dalam bahasan sebelumnya dijelaskan bahwa operasi assignment " = " akan men-copy alamat memory (reference) dari sebuah object, atau disebut sebagai *copy by reference*.

Bagaimana jika yang kita inginkan adalah men-copy nilai objectnya saja atau *copy by value*? PHP menyediakan perintah **clone** untuk keperluan ini.

82.object_clone.php

```

1 <?php
2 class Laptop {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8
9     public function cekMerek(){
10        return $this->merek;
11    }
12
13    public function setMerek($merek){
14        $this->merek = $merek;
15    }
16 }
17
18 $produk01 = new Laptop('Asus');
19 $produk02 = clone $produk01;
20
21 echo $produk01->cekMerek()."<br>"; // Asus
```

```

22 echo $produk02->cekMerek()."<br>";      // Asus
23
24 $produk02->setMerek('Dell')."<br>";
25
26 echo $produk01->cekMerek()."<br>";      // Asus
27 echo $produk02->cekMerek()."<br>";      // Dell

```

Ini class Laptop masih sama seperti contoh pada bahasan perbandingan object, yakni berisi property merek, sebuah constructor, serta method cekMerek() dan getMerek(). Selanjutnya class Laptop ini saya instansiasi ke dalam variabel \$produk01.

Pada baris 19, object di dalam variabel \$produk01 di "cloning" ke variabel \$produk02 dengan perintah **clone**. Hasilnya, variabel \$produk02 akan berisi seluruh property dan method yang sama dengan \$produk01, namun sekarang setiap property memiliki alamat masing-masing (tidak lagi menggunakan referensi yang sama).

Sebagai uji coba, di baris 24 saya mengubah isi property merek milik \$produk02 menjadi 'Dell'. Terlihat yang berubah hanya property merek di \$produk02 saja, sedangkan property merek milik \$produk01 tetap berisi 'Asus'.

Dengan penjelasan ini saya yakin anda sudah bisa membedakan maksud dari 2 perintah berikut:

```

1 $foo = $bar;
2 $foo = clone $bar;

```

Dengan menganggap variabel \$bar berisi object, perintah di baris 1 adalah operasi *copy by reference*, sedangkan perintah di baris 2 adalah operasi *copy by value*.

Magic Method `__clone()`

Masih berhubungan dengan clone, PHP menyediakan sebuah magic method `__clone()`. Method ini akan dijalankan pada saat sebuah object di cloning dengan perintah **clone**. Berikut contohnya:

83.magic_method_clone.php

```

1 <?php
2 class Laptop {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8
9     public function __clone(){
10        echo "Object Laptop di cloning... <br>";
11    }
12 }
13

```

```

14 $produk01 = new Laptop('Asus');
15 $produk02 = clone $produk01;
16 $produk03 = clone $produk02;

```

Hasil kode program:

```

Object Laptop di cloning...
Object Laptop di cloning...

```

Di dalam class `Laptop` terdapat method `__clone()` yang berisi perintah `echo "Object Laptop di cloning...
"`. Method `__clone()` dalam contoh ini memang tidak terlalu berguna, karena hanya berisi perintah `echo`, namun setidaknya bisa menjadi gambaran bahwa method ini akan dijalankan setiap kali object di cloning.

Shallow Copy dan Deep Copy

Bahasan kali ini mungkin terasa agak rumit, namun istilah *shallow copy* dan *deep copy* cukup sering dipakai terutama untuk mengukur seberapa dalam pengetahuan seseorang tentang konsep OOP di PHP.

Kedua istilah ini berkaitan tentang bagaimana PHP men-cloning sebuah object jika di dalam object tersebut terdapat referensi ke object lain.

Sebagai contoh awal, silahkan pelajari sejenak kode program berikut:

84.property_berisi_object.php

```

1 <?php
2 class Perusahaan {
3     public $nama;
4
5     public function __construct($nama){
6         $this->nama = $nama;
7     }
8 }
9
10 class Laptop {
11     public $merek;
12     public $suplier;
13
14     public function __construct($merek){
15         $this->merek = $merek;
16     }
17 }
18
19 $suplierLaptop = new Perusahaan("CV. Jaya Abadi");
20 $produk01 = new Laptop("Acer");
21 $produk01->suplier = $suplierLaptop;
22
23 echo "<pre>";
24 print_r($produk01);
25 echo "</pre>";

```

Hasil kode program:

```
Laptop Object
(
    [merek] => Acer
    [suplier] => Perusahaan Object
        (
            [nama] => CV. Jaya Abadi
        )
)
```

Dalam kode program ini saya membuat 2 buah class: Perusahaan dan Laptop.

Class Perusahaan (baris 2 – 8) memiliki property `nama` dan sebuah constructor yang dipakai untuk mengisi property tersebut. Property `nama` saya siapkan untuk menampung nama perusahaan.

Class Laptop (baris 10 – 17) memiliki 2 property yakni `merek` dan `suplier`, serta sebuah constructor yang dipakai untuk mengisi property `merek`.

Semua property di set sebagai `public` agar bisa diakses dari luar class. Ini semata-mata supaya kode program kita menjadi lebih singkat (tidak perlu method setter dan getter untuk setiap property).

Di baris 19 saya membuat variabel `$suplierLaptop` yang diisi dengan object hasil instansiasi class `Perusahaan("CV. Jaya Abadi")`. String "CV. Jaya Abadi" akan menjadi nilai untuk property `nama`.

Di baris 20 saya membuat variabel `$produk01` yang diisi dengan object hasil instansiasi class `Laptop("Acer")`.

Kemudian di baris 21 terdapat perintah `$produk01->suplier = $suplierLaptop`. Artinya, saya mengisi property `$suplier` milik `$produk01` dengan object `$suplierLaptop`. Cara ini memang baru pertama kali kita pakai, yakni **mengisi property dengan sebuah object**.

Property pada dasarnya sama seperti variabel biasa yang bisa diisi dengan data apapun, sebagai contoh, `$produk01` dan `$suplierLaptop` adalah variabel yang berisi **object**. Hal yang sama juga bisa diisi ke dalam property. Untuk kode program yang kompleks, property sebuah object bisa diisi dengan object lain.

Perintah `print_r($produk01)` di baris 24 memperlihatkan isi dari object `$produk01`. Bisa dilihat bahwa property `$suplier` berisi object `$suplierLaptop`.

Pertanyaan singkat, bagaimana cara mengakses property `$name` dari object `$produk01`? Berikut perintah yang dipakai:

```
echo $produk01->suplier->nama; // CV. Jaya Abadi
```

Perintah di atas berarti saya mengakses property `nama`, yang ada di dalam property `suplier`

milik object \$produk01.

Berikutnya, saya ingin mencopy object \$produk01 ke dalam object \$produk02. Caranya, gunakan perintah **clone**:

85.shallow_copy.php

```

1 <?php
2 class Perusahaan {
3     //...
4 }
5
6 class Laptop {
7     //...
8 }
9
10 $suplierLaptop = new Perusahaan("CV. Jaya Abadi");
11 $produk01 = new Laptop("Acer");
12 $produk01->suplier = $suplierLaptop;
13
14 $produk02 = clone $produk01;
15
16 echo "<pre>";
17 print_r($produk01);
18 echo "</pre>";
19
20 echo "<pre>";
21 print_r($produk02);
22 echo "</pre>";

```

Hasil kode program:

```

Laptop Object
(
    [merek] => Acer
    [suplier] => Perusahaan Object
        (
            [nama] => CV. Jaya Abadi
        )
)
Laptop Object
(
    [merek] => Acer
    [suplier] => Perusahaan Object
        (
            [nama] => CV. Jaya Abadi
        )
)

```

Untuk menghemat kode program, saya tidak lagi menampilkan isi dari class Perusahaan dan Laptop, namun isinya sama seperti kode sebelumnya.

Di baris 14 terdapat perintah `$produk02 = clone $produk01`. Artinya, saya ingin meng-copy object `$produk01` ke dalam `$produk02`. Kedua variabel akan berisi object yang isinya sama persis. Istilah "sama persis" di sini bukan berarti "identik", karena baik object `$produk01` dan `$produk02` disimpan di ruang memory yang berbeda.

Untuk membuktikan, saya akan mengubah isi property `merek` milik object `$produk02`:

86.shallow_copy_testing.php

```

1 <?php
2 class Perusahaan {
3     //...
4 }
5
6 class Laptop {
7     //...
8 }
9
10 $suplierLaptop = new Perusahaan("CV. Jaya Abadi");
11 $produk01 = new Laptop("Acer");
12 $produk01->suplier = $suplierLaptop;
13
14 $produk02 = clone $produk01;
15
16 $produk02->merek = "Apple";
17
18 echo "<pre>";
19 print_r($produk01);
20 echo "</pre>";
21
22 echo "<pre>";
23 print_r($produk02);
24 echo "</pre>";

```

Hasil kode program:

```

Laptop Object
(
    [merek] => Acer
    [suplier] => Perusahaan Object
        (
            [nama] => CV. Jaya Abadi
        )
)
Laptop Object
(
    [merek] => Apple
    [suplier] => Perusahaan Object
        (
            [nama] => CV. Jaya Abadi
        )
)

```

Setelah proses "cloning", di baris 16 saya mengubah property `merek` milik object `$produk02` menjadi '`Apple`'. Dari hasil `print_r()` terlihat bahwa sekarang isi property `merek` di kedua object sudah berbeda.

Percobaan selanjutnya, saya ingin mengubah isi property `nama` di object `$produk02`:

87.shallow_copy_testing_2.php

```

1  <?php
2  class Perusahaan {
3      //..
4  }
5
6  class Laptop {
7      //..
8  }
9
10 $suplierLaptop = new Perusahaan("CV. Jaya Abadi");
11 $produk01 = new Laptop("Acer");
12 $produk01->suplier = $suplierLaptop;
13
14 $produk02 = clone $produk01;
15
16 $produk02->merek = "Apple";
17 $produk02->suplier->nama = "CV. Maju Makmur";
18
19 echo "<pre>";
20 print_r($produk01);
21 echo "</pre>";
22
23 echo "<pre>";
24 print_r($produk02);
25 echo "</pre>";

```

Tambahan dalam kode program ini ada baris 17, dimana saya mengubah property `nama` milik object `$produk02` menjadi "`CV. Maju Makmur`". Dengan demikian, seharusnya sekarang property `nama` di `$produk01` dan `$produk02` akan berbeda (mirip seperti percobaan property `merek` sebelumnya).

Berikut hasil yang didapat:

```

Laptop Object
(
    [merek] => Acer
    [suplier] => Perusahaan Object
        (
            [nama] => CV. Maju Makmur
        )
)
Laptop Object
(

```

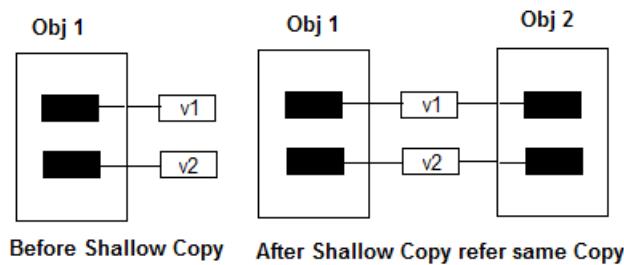
```
[merek] => Apple
[supplier] => Perusahaan Object
(
    [nama] => CV. Maju Makmur
)
)
```

Perhatikan isi property `nama`, keduanya berisi "CV. Maju Makmur"! Loh jika keduanya terpisah di memory, kenapa di sini perubahan di satu object masih berhubungan dengan object lain?

Inilah yang dimaksud dengan **shallow copy**. Secara default, ketika sebuah object di copy menggunakan perintah **clone**, PHP (dan mayoritas bahasa pemrograman object lain) akan **men-copy nilai** dari suatu property ke property lain. Tapi jika property tersebut merujuk ke object lain, maka yang di **copy hanya referensinya saja**.

Dengan demikian, property `nama` di dalam object `$produk01` dan `$produk02` masih terhubung karena yang di copy hanya referensinya saja (*copy by reference*), bukan nilai (*copy by value*).

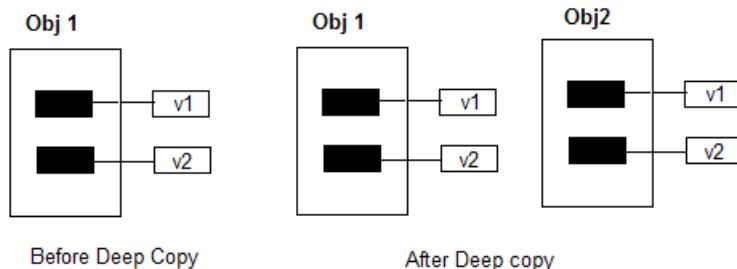
Gambar berikut mengilustrasikan apa yang terjadi:



Proses shallow copy (sumber gambar: net-informations.com)

Kotak hitam di dalam **Obj 1** merujuk ke property, sedangkan `v1` dan `v2` adalah nilai yang terhubung secara "reference". Ketika **Obj 1** di copy ke dalam **Obj 2**, referensi `v1` dan `v2` masih terhubung karena berisi alamat memory yang sama.

Sedangkan **deep copy** adalah proses copy object yang juga mencopy nilai reference, seperti ilustrasi berikut:



Proses deep copy (sumber gambar: net-informations.com)

Di dalam *deep copy*, apabila sebuah property berisi referensi ke object lain, referensi tersebut akan "diputus" agar masing-masing object menyimpan nilai yang berbeda.

Kenapa PHP melakukan *shallow copy* dan bukannya *deep copy* untuk perintah **clone**? Ini berkaitan dengan hal performa. Deep copy butuh proses yang cukup lama, apalagi jika object tersebut memiliki banyak property. Jika programmer butuh deep copy, silahkan diimplementasikan sendiri.

Jadi bagaimana cara membuat **deep copy**? Kita terpaksa memproses ini secara manual memakai bantuan magic method `__clone()`. Berikut contoh pembuatan *deep copy* dalam PHP:

88.deep_copy.php

```

1 <?php
2 class Perusahaan {
3     public $nama;
4
5     public function __construct($nama){
6         $this->nama = $nama;
7     }
8 }
9
10 class Laptop {
11     public $merek;
12     public $suplier;
13
14     public function __construct($merek){
15         $this->merek = $merek;
16     }
17
18     public function __clone() {
19         $this->suplier = clone $this->suplier;
20     }
21 }
22
23 $suplierLaptop = new Perusahaan("CV. Jaya Abadi");
24 $produk01 = new Laptop("Acer");
25 $produk01->suplier = $suplierLaptop;
26
27 $produk02 = clone $produk01;
28
29 $produk02->merek = "Apple";
30 $produk02->suplier->nama = "CV. Maju Makmur";
31
32 echo "<pre>";
33 print_r($produk01);
34 echo "</pre>";
35
36 echo "<pre>";
37 print_r($produk02);
38 echo "</pre>";

```

Prinsip dari *deep copy* adalah, jika terdapat property yang berisi object, maka pada saat proses **clone**, object tersebut juga harus di **clone**. Inilah fungsi dari method `__clone()` di dalam class `Laptop`.

Perintah `$this->suplier = clone $this->suplier` artinya saya ingin isi dari property `suplier` (yang berisi object) harus di **clone** terlebih dahulu, bukan sekedar *copy by reference*.

Sekarang, ketika `$produk01` di copy ke `$produk02` dengan perintah **clone**, property `suplier` akan berisi object `$suplierLaptop` yang berbeda. Perubahan di property `name` milik `$object2` tidak lagi ikut mengubah isi property `name` milik `$object1`.

Konsep perbandingan **shallow copy** dengan **deep copy** ini memang cukup rumit, dan hanya terjadi untuk property yang berisi object.

3.8. Method Chaining

Method chaining adalah istilah untuk menyebut penulisan method yang bisa disambung atau dirangkai satu sama lain, karena itu disebut dengan *chaining* (rantai).

Sebagai contoh, jika kita butuh mengisi 3 property ke dalam class `Laptop`, biasa ditulis sebagai berikut:

```
1 $produk01 = new Televisi();
2
3 $produk01->setMerek("Samsung");
4 $produk01->setJenisLayar("LED");
5 $produk01->setUkuranLayar("42");
```

Di sini saya membuat `$produk01` sebagai instansiasi dari class `Televisi`, kemudian mengakses method `setMerek`, `setJenisLayar`, dan `setUkuranLayar`. Cara ini tidak salah, namun dengan sedikit trik kita bisa buat proses pengisian seperti berikut:

```
1 $produk01 = new Televisi();
2
3 $produk01->setMerek("Samsung")->setJenisLayar("LED")->setUkuranLayar("42");
```

Sekarang proses pemanggilan method bisa disambung seperti rantai. Inilah yang dimaksud dengan *method chaining*. Teknik method chaining seperti ini cukup populer di library atau framework. Jika anda pernah menulis kode program dengan **jQuery**, hampir pasti sudah menggunakan cara penulisan seperti ini.

Prinsip kerja dari method chaining adalah, di dalam method kita harus mengembalikan object saat ini, yakni variabel `$this`. Kita akan lihat praktiknya sesaat lagi.

Sebelum ke sana, berikut kode program untuk pemanggilan method dengan cara "normal":

89.method_reguler.php

```
1 <?php
2
3 class Televisi {
4     private $merek;
5     private $jenisLayar;
```

```

6  private $ukuranLayar;
7
8  public function setMerek($merek){
9      $this->merek = $merek;
10 }
11 public function setJenisLayar($jenisLayar){
12     $this->jenisLayar = $jenisLayar;
13 }
14 public function setUkuranLayar($ukuranLayar){
15     $this->ukuranLayar = $ukuranLayar;
16 }
17
18 public function cekInfo(){
19     return "Televisi ".$this->jenisLayar." ".$this->merek." ".
20             $this->ukuranLayar." inch";
21 }
22
23 }
24
25 $produk01 = new Televisi();
26
27 $produk01->setMerek("Samsung");
28 $produk01->setJenisLayar("LED");
29 $produk01->setUkuranLayar("42");
30
31 echo $produk01->cekInfo();

```

Hasil kode program:

Televisi LED Samsung 42 inch

Dalam kode program ini saya membuat class `Televisi` dengan 3 property, yakni `merek`, `jenisLayar` dan `ukuranLayar`. Semua property di set sebagai private dan proses input dibuat menggunakan 3 buah method setter: `setMerek()`, `setJenisLayar()`, dan `setUkuranLayar()`.

Terdapat juga method `cekInfo()` untuk menampilkan isi dari ketiga property yang disambung dalam 1 string panjang. Sekarang mari kita ubah agar setiap method setter bisa dipanggil dengan cara *method chaining*:

90.method_chaining.php

```

1 <?php
2
3 class Televisi {
4     private $merek;
5     private $jenisLayar;
6     private $ukuranLayar;
7
8     public function setMerek($merek){
9         $this->merek = $merek;
10        return $this;
11    }
12    public function setJenisLayar($jenisLayar){

```

```

13     $this->jenisLayar = $jenisLayar;
14     return $this;
15 }
16 public function setUkuranLayar($ukuranLayar){
17     $this->ukuranLayar = $ukuranLayar;
18     return $this;
19 }
20
21 public function cekInfo(){
22     return "Televisi ".$this->jenisLayar." ".$this->merek." ".
23           $this->ukuranLayar." inch";
24 }
25
26 }
27
28 $produk01 = new Televisi();
29 echo $produk01->setMerek("Samsung")->setJenisLayar("LED")
30     ->setUkuranLayar("42")->cekInfo();

```

Perubahan dari kode program sebelumnya ada di perintah `return $this` pada akhir setiap method setter. Dengan mengembalikan object saat ini (variabel `$this`), pemanggilan method bisa disambung menjadi 1 perintah yang panjang.

Karena keterbatasan panjang halaman, tampilan `echo` di baris 29 terpaksa saya sambung ke baris 30. Tapi anda bisa menulisnya dalam 1 baris saja.

Selain itu bisa juga ditulis terpisah seperti berikut:

```

1 echo $produk01->setMerek("Samsung")
2     ->setJenisLayar("LED")
3     ->setUkuranLayar("42")
4     ->cekInfo();

```

Jika ditulis seperti ini, tanda titik koma harus berada di baris paling bawah, tidak di setiap baris.

Exercise

Seperti yang disinggung sebelumnya, *method chaining* cukup populer dipakai dalam framework atau library. Sebagai contoh, beberapa library database butuh penulisan query sebagai berikut:

```
$mahasiswa01->select("nim, nama")->from("mahasiswa")->where("nim='13012012'");
```

Hasil akhirnya menjadi sebuah string query MySQL:

```
SELECT nim, nama FROM mahasiswa WHERE nim = '13012012'
```

Bisakah anda merancang class untuk tampilan seperti ini? Kita tidak membahas sampai ke database, tapi hanya penyambungan string saja. Prinsipnya sama seperti kode

program kita sebelumnya, setiap method harus mengembalikan variabel **\$this**.

Di sini juga perlu membuat 3 method: `select()`, `from()` dan `where()`. Setiap pemanggilan method nantinya bisa disambung menjadi 1 string panjang.

```

1 $mahasiswa01 = new DisplayDatabase();
2 $mahasiswa01->select("nim, nama")->from("mahasiswa")
3         ->where("nim='13012012'");
4
5 echo $mahasiswa01->getQuery();
6 // SELECT nim, nama FROM mahasiswa WHERE nim = '13012012'
```

Pada awal kode program, saya mengisi variabel `$mahasiswa01` dengan hasil instansiasi class `DisplayDatabase`. Kemudian terdapat 3 method chaining yang diakses dari object `$mahasiswa01`. Terakhir, method `getQuery()` dipakai untuk menampilkan hasil akhir string.

Silahkan anda coba merancang isi class `DisplayDatabase` ini.

Baik, latihan ini mungkin sedikit sulit. Namun **selamat** jika anda berhasil. Berikut kode program yang saya pakai:

```

1 <?php
2
3 class DisplayDatabase {
4     private $query;
5
6     public function select($column){
7         $this->query = "SELECT $column ";
8         return $this;
9     }
10    public function from($table){
11        $this->query .= "FROM $table ";
12        return $this;
13    }
14    public function where($condition){
15        $this->query .= "WHERE $condition";
16        return $this;
17    }
18    public function getQuery(){
19        return $this->query;
20    }
21
22}
23
24
25 $mahasiswa01 = new DisplayDatabase();
26 $mahasiswa01->select("nim, nama")->from("mahasiswa")
27         ->where("nim='13012012');
```

```

29 echo $mahasiswa01->getQuery();
30 // SELECT nim, nama FROM mahasiswa WHERE nim = '13012012'

```

Tidak masalah jika kode program yang anda gunakan sedikit berbeda. Selama hasilnya berupa string `SELECT nim, nama FROM mahasiswa WHERE nim = '13012012'`, itu tidak masalah.

Dalam merancang class `DisplayDatabase` ini saya menggunakan 1 property untuk menampung semua string: `query`. Untuk setiap method setter, property `query` ini akan disambung menggunakan operator gabungan assignment (`.=`). Di dalam method `getQuery()`, saya tinggal mengembalikan isi string `query` ini.

Anda bisa saja menggunakan teknik menyimpan ke variabel terpisah seperti contoh class `Televisi` sebelumnya, kemudian digabung kembali menjadi string akhir.

3.9. Type Hinting

Type hinting adalah fitur PHP untuk membatasi jenis tipe data parameter yang bisa diterima oleh sebuah function atau method.

Dalam PHP, tipe data bisa dibagi menjadi 2 kelompok besar: *tipe data scalar* dan *tipe data composite*. **Tipe data scalar** (disebut juga sebagai tipe data dasar), adalah tipe data sederhana yang terdiri dari **integer**, **float**, **boolean** dan **string**. Sedangkan **tipe data composite** adalah tipe data bentukan yang terdiri dari kumpulan tipe data dasar, diantaranya **array** dan **object**.

Type hinting untuk tipe data composite sudah tersedia sejak PHP 5, sedangkan type hinting untuk tipe data scalar baru ditambahkan di PHP 7. Kita akan bahas type hinting untuk tipe data composite terlebih dahulu.

Array Type Hinting

Sebagai salah satu tipe data composite, array sudah memiliki fitur type hinting sejak PHP 5.1. Agar lebih mudah dipahami, kita akan bahas dengan contoh kasus.

Perhatikan kode program berikut ini:

```

92.function_with_array.php

1 <?php
2
3 function hitungRata2($data){
4     return ($data[0] + $data[1] + $data[2])/3;
5 }
6
7 echo hitungRata2([3, 6, 12]); // 7

```

Saya membuat fungsi `hitungRata2()` dengan sebuah parameter `$data`. Nantinya parameter

\$data ini akan berisi array. Fungsi hitungRata2() akan mengembalikan nilai rata-rata dari 3 element array tersebut.

Di baris 7 saya menjalankan perintah hitungRata2([3, 6, 12]). Nilai argument yang dikirim berupa array [3, 6, 12], dengan demikian hasil akhir dari pemanggilan fungsi ini adalah angka 7, yang didapat dari $(3+6+12)/3$.

Fungsi hitungRata2() tentunya hanya bisa menerima argument yang berbentuk array, tapi bagaimana jika kita input tipe data lain yang tidak berbentuk array?

93.function_with_array_0.php

```

1 <?php
2
3 function hitungRata2($data){
4     return ($data[0] + $data[1] + $data[2])/3;
5 }
6
7 echo hitungRata2(3, 6, 12); // 0

```

Sekarang di baris 7 saya menjalankan hitungRata2(3, 6, 12). Perhatikan bahwa argument untuk pemanggilan ini tidak berbentuk array, tapi 3 buah argument integer 3, 6 dan 12.

Hasilnya, PHP tetap menjalankan fungsi hitungRata2() dan tidak tampil pesan error apapun.

Dalam situasi seperti ini, pesan error akan sangat membantu. Jika fungsi hitungRata2() ditulis oleh orang lain, kita akan bingung kenapa rata-rata dari 3, 6 dan 12 adalah 0?

Type hinting menjadi solusinya. Dengan type hinting, kita bisa membuat semacam aturan tambahan untuk parameter fungsi :

94.function_array_type_hinting.php

```

1 <?php
2
3 function hitungRata2(array $data){
4     return ($data[0] + $data[1] + $data[2])/3;
5 }
6
7 echo hitungRata2(3, 6, 12);

```

Hasil kode program:

```
Fatal error: Uncaught TypeError: Argument 1 passed to hitungRata2() must be of the
type array, integer given
```

Type hinting berada tepat sebelum penulisan nama parameter. Dalam contoh di atas, tambahan keyword **array** di baris 3 sebelum \$data adalah type hinting, fungsinya untuk menginformasikan kepada PHP bahwa parameter \$data harus diisi dengan nilai argument yang berbentuk **array**, tidak boleh tipe data lain.

Ketika saya menjalankan perintah `hitungRata2(3, 6, 12)`, akan tampil pesan Fatal error yang berisi penjelasan bahwa argument untuk fungsi `hitungRata2()` harus berbentuk array.

Object Type Hinting

Prinsip penggunaan type hinting untuk object mirip seperti array. Bedanya, kita menulis nama class sebagai pengganti tipe data. Sebelum ke sana, mari lihat contoh class `Smartphone` berikut ini:

```
95.function_with_object.php

1 <?php
2 class Smartphone {
3     public $merek;
4     public $tipe;
5     public $harga;
6
7     public function __construct($merek,$tipe,$harga){
8         $this->merek = $merek;
9         $this->tipe = $tipe;
10        $this->harga = $harga;
11    }
12 }
13
14 function tampilkanSmartphone($hp){
15     return "Smartphone ".$hp->merek." ".$hp->tipe." di jual seharga Rp. "
16         .number_format($hp->harga,2,",",".");
17 }
18
19 $produk01 = new Smartphone("Xiaomi","Redmi Note 6",2799000);
20 $produk02 = new Smartphone("Samsung","Galaxy S9+",11999000);
21 $produk03 = new Smartphone("Apple","iPhone X",15700000);
22
23 echo tampilkanSmartphone($produk01);
24 echo "<br>";
25 echo tampilkanSmartphone($produk02);
26 echo "<br>";
27 echo tampilkanSmartphone($produk03);
```

Hasil kode program:

```
Smartphone Xiaomi Redmi Note 6 di jual seharga Rp. 2.799.000,00
Smartphone Samsung Galaxy S9+ di jual seharga Rp. 11.999.000,00
Smartphone Apple iPhone X di jual seharga Rp. 15.700.000,00
```

Di baris 2 saya membuat class `Smartphone` dengan 3 buah property: `merek`, `tipe` dan `harga`. Ketiga property di set sebagai `public` agar mudah diakses. Kemudian terdapat constructor untuk proses instansiasi ketiga property tersebut.

Di baris 14 terdapat fungsi `tampilkanSmartphone()`. Fungsi ini bukanlah method di dalam class `Smartphone`, tetapi sebuah function terpisah. Saya tulis seperti ini agar kita lebih mudah

membahas tentang *type hinting*.

Sesuai namanya, fungsi `tampilkanSmartphone()` dipakai untuk menampilkan ketiga property yang ada di dalam class `Smartphone`. Fungsi `tampilkanSmartphone()` butuh 1 parameter: `$hp`. Parameter `$hp` ini nantinya harus diisi dengan object dari class `Smartphone`.

Di baris 19 – 21 saya menginstansiasi 3 buah object dari class `Smartphone`, yakni `$produk01`, `$produk02`, dan `$produk03`. Ketiganya diisi dengan nilai yang berlainan. Kemudian di baris 23 – 27 ketiga object diinput ke dalam fungsi `tampilkanSmartphone()`. Hasilnya berupa string yang berasal dari setiap object.

Dalam kode program di atas, fungsi `tampilkanSmartphone()` hanya bisa menerima argument yang berasal dari class `Smartphone`. Ini karena di dalam fungsi tersebut terdapat perintah untuk mengakses property `merek`, `tipe` dan `harga`. Tapi bagaimana jika ternyata ada class lain?

95.function_with_object_2.php

```

1 <?php
2 class Smartphone {
3     public $merek;
4     public $tipe;
5     public $harga;
6
7     public function __construct($merek,$tipe,$harga){
8         $this->merek = $merek;
9         $this->tipe = $tipe;
10        $this->harga = $harga;
11    }
12 }
13
14 class Televisi {
15     public $merek;
16     public $tipe;
17     public $harga;
18
19     public function __construct($merek,$tipe,$harga){
20         $this->merek = $merek;
21         $this->tipe = $tipe;
22         $this->harga = $harga;
23     }
24 }
25
26 function tampilkanSmartphone($hp){
27     return "Smartphone ".$hp->merek." ".$hp->tipe." di jual seharga Rp. "
28         .number_format($hp->harga,2,",",".");
29 }
30
31 $produk01 = new Televisi("Samsung", "LED TV 40 inch UA40M5000",4499000);
32 $produk02 = new Smartphone("Samsung","Galaxy S9+",11999000);
33
34 echo tampilkanSmartphone($produk01);
35 echo "<br>";

```

```
36 echo tampilanSmartphone($produk02);
```

Hasil kode program:

```
Smartphone Samsung LED TV 40 inch UA40M5000 di jual seharga Rp. 4.499.000,00
Smartphone Samsung Galaxy S9+ di jual seharga Rp. 11.999.000,00
```

Sekarang terdapat 2 class: `Smartphone` dan `Televisi`. Kedua class ini sama-sama memiliki property `merek`, `tipe` dan `harga`. Dengan demikian, object `$produk01` yang merupakan instansiasi dari class `Televisi` juga bisa diinput ke dalam fungsi `tampilanSmartphone()`. Tapi hasilnya menjadi:

```
Smartphone Samsung LED TV 40 inch UA40M5000 di jual seharga Rp. 4.499.000,00
```

Tentu saja ini bukanlah sebuah smartphone, tapi televisi.

Kembali, fungsi `tampilanSmartphone()` hanya ditujukan untuk bisa menerima argument berupa object class `Smartphone`. Dalam kasus seperti inilah kita bisa membuat **object type hinting**. Caranya, tambahkan nama class di awal penulisan parameter:

97.function_object_type_hinting.php

```
1 <?php
2 class Smartphone {
3     //...
4 }
5
6 class Televisi {
7     //...
8 }
9
10 function tampilanSmartphone(Smartphone $hp){
11     return "Smartphone ".$hp->merek." ".$hp->tipe." di jual seharga Rp. "
12         .number_format($hp->harga,2,",",".");
13 }
14
15 $produk01 = new Televisi("Samsung", "LED TV 40 inch UA40M5000",4499000);
16 $produk02 = new Smartphone("Samsung", "Galaxy S9+",11999000);
17
18 echo tampilanSmartphone($produk01);
19 echo "<br>";
20 echo tampilanSmartphone($produk02);
```

Perhatikan tambahan keyword "Smartphone" sebelum penulisan parameter `$hp` di baris 10. Inilah yang dimaksud dengan **object type hinting**. Kita menginformasikan kepada PHP bahwa parameter untuk fungsi `tampilanSmartphone()` hanya boleh diisi dengan argument yang merupakan hasil instansiasi class `Smartphone`.

Karena di baris 18 saya tetap menginput `$produk01` (object dari class `Televisi`), maka akan tampil pesan error berikut:

Fatal error: Uncaught TypeError: Argument 1 passed to tampilanSmartphone() must be an instance of Smartphone, instance of Televisi given

Yup, PHP "protes" karena terdapat perbedaan jenis object.

Agar class `Televisi` juga bisa ditampilkan, saya akan buat function terpisah:

98.function_object_type_hinting_2.php

```

1 <?php
2 class Smartphone {
3     //...
4 }
5
6 class Televisi {
7     //...
8 }
9
10 function tampilanSmartphone(Smartphone $hp){
11     return "Smartphone ".$hp->merek." ".$hp->tipe." di jual seharga Rp. "
12         .number_format($hp->harga,2,",",".");
13 }
14
15 function tampilanTelevisi(Televisi $tv){
16     return "Televisi ".$tv->merek." ".$tv->tipe." di jual seharga Rp. "
17         .number_format($tv->harga,2,",",".");
18 }
19
20 $produk01 = new Televisi("Samsung", "LED TV 40 inch UA40M5000",4499000);
21 $produk02 = new Smartphone("Samsung","Galaxy S9+",11999000);
22
23 echo tampilanTelevisi($produk01);
24 echo "<br>";
25 echo tampilanSmartphone($produk02);
```

Hasil kode program:

Televisi Samsung LED TV 40 inch UA40M5000 di jual seharga Rp. 4.499.000,00
 Smartphone Samsung Galaxy S9+ di jual seharga Rp. 11.999.000,00

Sekarang class `Smartphone` dan `Televisi` sudah memiliki function masing-masing. Isi dari fungsi `tampilanSmartphone()` dan `tampilanTelevisi()` sebenarnya sangat mirip. Oleh karena itu saya akan memanfaatkan fitur turunan class (*inheritance*) yang digabung dengan type hinting, berikut contohnya:

99.function_object_type_hinting_inheritance.php

```

1 <?php
2 class Produk {
3     public $merek;
4     public $tipe;
5     public $harga;
6
```

```

7  public function __construct($merek,$tipe,$harga){
8      $this->merek = $merek;
9      $this->tipe = $tipe;
10     $this->harga = $harga;
11 }
12 }
13
14 class Smartphone extends Produk { }
15 class Televisi extends Produk { }
16
17 function tampilanProduk(Produk $barang){
18     return "Produk ".$barang->merek." ".$barang->tipe." di jual seharga Rp. "
19         .number_format($barang->harga,2,",",".");
20 }
21
22 $produk01 = new Televisi("Samsung", "LED TV 40 inch UA40M5000",4499000);
23 $produk02 = new Smartphone("Samsung","Galaxy S9+",11999000);
24
25 echo tampilanProduk($produk01);
26 echo "<br>";
27 echo tampilanProduk($produk02);

```

Hasil kode program:

```

Produk Samsung LED TV 40 inch UA40M5000 di jual seharga Rp. 4.499.000,00
Produk Samsung Galaxy S9+ di jual seharga Rp. 11.999.000,00

```

Di sini saya menulis ulang class `Smartphone` dan `Televisi`. Sekarang, kedua class di set sebagai turunan dari class `Produk`.

Di awal kode program, terdapat class `Produk` dengan 3 buah property: `merek`, `tipe` dan `harga`, serta sebuah constructor. Kode ini sama seperti yang ada di dalam class `Smartphone` dan `Televisi` sebelumnya.

Sekarang class `Smartphone` dan `Televisi` cukup diturunkan dari class `Produk`, dengan demikian keduanya otomatis memiliki property `merek`, `tipe` dan `harga` serta constructor yang sama seperti class `Produk`.

Proses menampilkan data akan di akses dari fungsi `tampilanProduk()`. Perhatikan tambahan type hinting `Produk` untuk parameter `$barang` di baris 17. Artinya, fungsi `tampilanProduk()` hanya bisa diisi dengan argument yang berasal dari class `Produk` saja.

Loh, bukankah `$produk01` dan `$produk02` bukan object dari class `Produk`? Betul, tapi ternyata type hinting juga berlaku untuk class turunan. Class `Smartphone` dan `Televisi` adalah turunan dari class `Produk` sehingga type hinting `Produk` juga berlaku untuk `$produk01` dan `$produk02`.

Selain class turunan, type hinting juga bisa dipakai untuk **interface**. Berikut contohnya:

100.function_object_type_hinting_interface.php

```

1 <?php
2 class Produk {

```

```

3  public $merek;
4  public $tipe;
5  public $harga;
6
7  public function __construct($merek,$tipe,$harga){
8      $this->merek = $merek;
9      $this->tipe = $tipe;
10     $this->harga = $harga;
11 }
12 }
13
14 interface SmartElectronic{
15     public function cekOS();
16 }
17
18 class Televisi extends Produk implements SmartElectronic{
19     public function cekOS(){
20         return "Android 9.0 (Pie)";
21     }
22 }
23
24 function tampilanProduk(SmartElectronic $barang){
25     return "Produk ".$barang->merek." ".$barang->tipe.", dengan "
26         .".$barang->cekOS()." di jual seharga Rp. "
27         .number_format($barang->harga,2,",","."));
28 }
29
30 $produk01 = new Televisi("Samsung", "LED TV 40 inch UA40M5000",4499000);
31 echo tampilanProduk($produk01);

```

Hasil kode program:

Produk Samsung LED TV 40 inch UA40M5000, dengan Android 9.0 (Pie) di jual seharga Rp. 4.499.000,00

Di baris 14 saya membuat sebuah interface `SmartElectronic`. Isinya berupa signature untuk method `cekOS()`.

Di baris 18 , class `Televisi` mengimplementasikan interface `SmartElectronic` ini, sehingga kita harus mendefinisikan ulang method `cekOS()` di dalam class `Televisi`. Isi dari method `cekOS()` akan mengembalikan string "Android 9.0 (Pie)".

Untuk fungsi `tampilanProduk()`, saya menulis type hinting berupa `SmartElectronic` (baris 24). Artinya, fungsi ini hanya bisa menerima argument yang menerapkan interface `SmartElectronic`.

Dari beberapa percobaan di atas kita bisa simpulkan bahwa type hinting bisa berlaku untuk **object**, **parent object**, serta **interface**. PHP memiliki operator khusus yakni **instanceof** yang bisa dipakai untuk memeriksa apakah sebuah object merupakan bagian dari class tertentu.

Operator `instanceof` mengembalikan nilai boolean **true** jika sebuah object merupakan bagian dari class tertentu, dan mengembalikan nilai **false** jika object tersebut bukan bagian dari class.

Berikut contoh penggunaannya:

101.instance_of.php

```

1 <?php
2 class Produk { }
3
4 interface SmartElectronic{
5     public function cekOS();
6 }
7
8 trait LowWatt{
9     public function efisiensi(){
10     return "Konsumsi daya 0.8";
11 }
12 }
13
14 class Televisi extends Produk implements SmartElectronic{
15     use LowWatt;
16     public function cekOS(){
17         return "Android 9.0 (Pie)";
18     }
19 }
20
21 $produk01 = new Televisi();
22
23 echo var_dump($produk01 instanceof Produk)."<br>";
24 echo var_dump($produk01 instanceof Televisi)."<br>";
25 echo var_dump($produk01 instanceof SmartElectronic)."<br>";
26 echo var_dump($produk01 instanceof LowWatt)."<br>";
27 echo var_dump($produk01 instanceof Smartphone)."<br>";

```

Hasil kode program:

```

bool(true)
bool(true)
bool(true)
bool(false)
bool(false)

```

Di sini, class `Televisi` adalah turunan dari class `Produk`, mengimplementasikan interface `SmartElectronic`, serta memakai trait `LowWatt`.

Dengan struktur seperti ini, object `$produk01` yang merupakan hasil instansi dari class `Televisi` akan bernilai **true** dari operasi `instanceof Produk`, `instanceof Televisi`, dan `instanceof SmartElectronic`. Tapi menghasilkan nilai **false** untuk operasi `instanceof LowWatt` dan `instanceof Smartphone`.

Ini menegaskan bahwa **trait** tidak dianggap sebagai bagian dari struktur class. Meskipun object `$produk01` menggunakan trait `LowWatt`, tapi hasil `instanceof` menghasilkan nilai **false**.

Dalam bahasan sebelumnya saya mempraktekkan type hinting untuk function. Prinsip yang

sama juga berlaku untuk method, tapi tentu kode programnya menjadi sedikit lebih kompleks karena di sini kita harus menginput object sebagai argument ke dalam method object lain. Berikut contohnya:

102.method_type_hinting.php

```

1 <?php
2 class Perusahaan {
3     private $namaPerusahaan;
4     private $kotaPerusahaan;
5
6     public function __construct($nama,$kota){
7         $this->namaPerusahaan = $nama;
8         $this->kotaPerusahaan = $kota;
9     }
10
11    public function cekPerusahaan(){
12        return $this->namaPerusahaan." dari kota ".$this->kotaPerusahaan;
13    }
14
15 }
16
17 class Smartphone {
18     private $merek;
19     private $suplier;
20
21     public function __construct($merek, Perusahaan $suplier){
22         $this->merek = $merek;
23         $this->suplier = $suplier;
24     }
25
26     public function cekSmartphone(){
27         return "Smartphone ".$this->merek.", di supply oleh ".
28             $this->suplier->cekPerusahaan();
29     }
30 }
31
32 $suplier01 = new Perusahaan("CV Maju Terus","Bandung");
33 $produk01 = new Smartphone("Xiaomi",$suplier01);
34
35 echo $produk01->cekSmartphone();

```

Kode program ini terdiri dari 2 class: Perusahaan dan Smartphone.

Class Perusahaan memiliki 2 property, yakni `namaPerusahaan` dan `kotaPerusahaan`. Kemudian terdapat constructor untuk mengisi kedua property ini. Selain itu juga ada method `cekPerusahaan()` untuk menampilkan teks yang berisi informasi perusahaan tersebut (diambil dari nilai property `namaPerusahaan` dan `kotaPerusahaan`).

Class Smartphone memiliki 2 property, yakni `merek` dan `suplier`. Property `merek` nantinya berisi nama merek smartphone (berupa string biasa), sedangkan property `suplier` akan berisi

object dari class Perusahaan. Untuk membatasi hal ini, saya menambahkan type hinting **Perusahaan** di constructor class Smartphone (baris 21). Artinya, class Smartphone hanya bisa di instansi jika argument keduanya di isi object dari class Perusahaan. Class Smartphone juga memiliki method `cekSmartphone()` yang menampilkan isi property merek dan suplier.

Selanjutnya di baris 32 saya mengisi variabel `$suplier01` sebagai object dari class Perusahaan. Kemudian di baris 33 mengisi variabel `$produk01` sebagai object dari class Smartphone. Perhatikan isi argument kedua dari proses instansi class Smartphone, isinya berupa object `$suplier01`.

Dengan type hinting ini, class Smartphone memiliki sebuah "proses validasi". Jika argument kedua constructor bukan berisi object dari class Perusahaan, maka akan tampil pesan error. Misalkan apabila saya menulis:

```
$produk01 = new Smartphone("Xiaomi","CV Jaya Abadi");
```

Hasilnya adalah:

```
Fatal error: Uncaught TypeError: Argument 2 passed to Smartphone::__construct() must be an instance of Perusahaan, string given
```

Bagi programmer yang melihat pesan kesalahan ini, bisa paham bahwa argument 2 harus diisi dengan "instance of Perusahaan", yakni harus diisi dengan object class Perusahaan.

Scalar Type Hinting

Materi ini sebenarnya tidak berhubungan langsung dengan pemrograman object, namun karena scalar type hinting termasuk fitur baru di PHP 7, tidak ada salahnya kita bahas sekilas.

PHP 7 membawa beberapa fitur baru, salah satunya type hinting untuk tipe data scalar, yakni type hinting untuk tipe data dasar: *integer*, *float*, *boolean* dan *string*. Dengan menerapkan type hinting, kita bisa membatasi tipe data yang bisa diterima oleh sebuah function. Namun penggunaannya perlu tambahan perintah khusus (akan kita bahas sesaat lagi).

Sebagai contoh awal, perhatikan kode program berikut:

103.scalar_function.php

```
1 <?php
2 function tambah($a, $b){
3     return $a + $b;
4 }
5
6 echo tambah(5,6);      // 11
7 echo "<br>";
8 echo tambah(6.5,8.9); // 15.4
9 echo "<br>";
10 echo tambah("5",6);   // 11
```

Di sini saya membuat fungsi `tambah()`. Isi fungsi ini sangat sederhana, yakni menambahkan 2

buah parameter. Tentu saja fungsi ini hanya bisa menerima argument yang bertipe angka (*number*), yakni integer dan float.

Di baris 10 saya menginput string "5" sebagai argument pertama untuk fungsi `tambah()`. Hasilnya seolah-olah tidak terjadi apa-apa. Ini karena PHP memiliki fitur **type juggling**, yakni bisa mengkonversi sebuah tipe data menjadi tipe data lain jika dibutuhkan.

Operator plus (+) adalah operator aritmatika untuk tipe data number. Jika kita memakai operator ini untuk tipe data yang bukan number, PHP akan mencoba mengkonversi nilai tersebut menjadi angka. Operasi "5" + 6 akan diproses sebagai 5 + 6. Untuk aplikasi sederhana proses konversi otomatis seperti ini tampak tidak masalah, namun sebenarnya sering menjadi sumber error. Inilah salah satu alasan PHP 7 menambahkan fitur type hinting.

Cara penambahan type hinting sama seperti array dan object sebelumnya, yakni dengan menulis tipe data sebelum penulisan parameter fungsi. Tipe data yang ditulis adalah `int`, `float`, `string` atau `bool`.

Berikut contohnya:

104. scalar_type_hinting.php

```

1 <?php
2 function tambah(int $a,int $b){
3     return $a + $b;
4 }
5
6 echo tambah(5,6);      // 11
7 echo "<br>";
8 echo tambah(6.5,8.9); // 14
9 echo "<br>";
10 echo tambah("5",6);   // 11

```

Di baris 2 saya menulis type hinting `int` untuk kedua parameter, yakni `int $a` dan `int $b`. Namun kenapa tidak tampil pesan error? Padahal di baris 8 argument yang dikirim bertipe float dan di baris 10 argument pertama bertipe string.

Ini terjadi karena PHP tidak menerapkan scalar type hinting dalam skala global. Secara default, scalar type hinting berperilaku mirip seperti type casting, yakni konversi antar tipe data.

Dalam contoh di atas, pemanggilan fungsi `tambah(6.5,8.9)` hasilnya adalah 14, bukan 15.4 seperti contoh sebelumnya. Alasannya, karena tambahan type hinting `int` akan mengkonversi nilai argument menjadi integer. Jika angka yang diinput berupa float, proses konversi ini akan menghapus angka koma. Nilai float 6.5 dikonversi menjadi 6, dan nilai float 8.9 dikonversi menjadi 8. Hasilnya $6 + 8 = 14$.

Yang sebenarnya terjadi mirip seperti kasus berikut:

```

1 <?php
2 $a = (int) 6.5;

```

```

3 $b = (int) 8.9;
4
5 echo $a;          // 6
6 echo $b;          // 8
7 echo $a + $b;    // 14

```

Jadi bagaimana jika yang di inginkan adalah type hinting "yang sebenarnya"? Yakni akan menghasilkan pesan error jika tipe data tidak sesuai? Caranya, kita perlu masuk ke **PHP strict mode**:

105.scalar_type_hinting_strict.php

```

1 <?php
2 declare(strict_types=1);
3
4 function tambah(int $a,int $b){
5     return $a + $b;
6 }
7
8 echo tambah(5,6);
9 echo "<br>";
10 echo tambah(6.5,8.9);
11 echo "<br>";
12 echo tambah("5",6);

```

Hasil kode program:

11

```
Fatal error: Uncaught TypeError: Argument 1 passed to tambah() must be of the type integer, float given
```

Di baris 2 terdapat tambahan perintah `declare(strict_types=1)`, ini dipakai agar PHP masuk ke **strict mode**. Jika ditulis seperti ini, barulah scalar type hinting menampilkan pesan error.

Error terjadi karena di baris 10 argument yang dikirim berupa float, bukan integer. Jika baris ini dihapus, akan terjadi error kedua yang berasal dari baris 12 karena argument yang dikirim berupa string.

PHP memutuskan bahwa scalar type hinting hanya menampilkan pesan error di dalam **strict mode** saja. Alasannya untuk menghindari masalah dengan kode program yang sudah ada. Karena bisa jadi ada beberapa kode program yang memang memanfaatkan proses konversi otomatis antar tipe data.

3.10. Typed Properties

PHP 7.4 membawa fitur baru yang mengizinkan kita membatasi tipe data sebuah property.

Cara penulisannya mirip seperti type hinting, hanya saja ditulis sebelum nama property.

Berikut contoh penggunaan dari typed properties:

105a.type_property_1.php

```

1 <?php
2 class Smartphone {
3     public string $merek;
4 }
5
6 $produk01 = new Smartphone();
7 $produk01->merek = "Oppo";
8
9 echo $produk01->merek; // Oppo

```

Di baris 3 terdapat tambahan keyword **string** antara **public** dan **\$merek**. Dengan tambahan tersebut, property **\$merek** hanya bisa diisi dengan tipe data string saja. Inilah yang dimaksud dengan **typed properties**.

Akan tetapi sebagaimana yang kita ketahui, PHP juga memiliki fitur *type juggling* yang bisa mengkonversi suatu tipe data menjadi tipe data lain secara otomatis:

105b.type_property_2.php

```

1 <?php
2 class Smartphone {
3     public string $merek;
4 }
5
6 $produk01 = new Smartphone();
7 $produk01->merek = 100;
8
9 var_dump($produk01->merek); // string(3) "100"

```

Meskipun property **\$merek** sudah dibatasi agar hanya bisa diisi dengan data **string** saja, namun kode di baris 7 tidak akan error. Angka 100 dikonversi PHP menjadi tipe data string pada saat proses input. Ini bisa dibuktikan dari hasil perintah **var_dump(\$produk01->merek)** di baris 9.

Jika kita tidak ingin proses konversi ini berlangsung otomatis, bisa menerapkan **strict mode**:

105c.type_property_3.php

```

1 <?php
2 declare(strict_types=1);
3
4 class Smartphone {
5     public string $merek;
6 }
7
8 $produk01 = new Smartphone();
9 $produk01->merek = 100; // Fatal error: Uncaught TypeError: Cannot assign int
10 // to property Smartphone::$merek of type string

```

Sekarang kode yang sama akan error karena kita berada dalam strict mode dengan tambahan

perintah `declare(strict_types=1)` pada awal kode program.

Semua tipe data dasar bisa diterapkan ke dalam *typed properties* seperti `int`, `float`, `boolean`, dll. Berikut contoh penggunaan *typed properties* untuk tipe data `int`:

105d.type_property_4.php

```

1 <?php
2 declare(strict_types=1);
3
4 class Smartphone {
5     public int $jumlah;
6 }
7
8 $produk01 = new Smartphone();
9 $produk01->jumlah = "100";
10
11 var_dump($produk01->jumlah);
12 // Fatal error: Uncaught TypeError: Cannot assign string to property
13 // Smartphone::$jumlah of type int

```

Kali ini saya membuat property `$jumlah` dengan batasan tipe data `int`. Karena berada dalam *strict mode*, kode di atas akan error. Perintah input di baris 7 harus diisi dengan tipe data integer seperti `$produk01->jumlah = 100`, tidak boleh `$produk01->jumlah = "100"`.

Typed Properties untuk Object

Lebih jauh lagi, *typed properties* juga bisa dipakai untuk tipe data composite seperti object. Berikut contoh penggunaannya:

105e.type_property_object_1.php

```

1 <?php
2 class Smartphone {
3     public Perusahaan $suplier;
4 }
5
6 $produk01 = new Smartphone();
7 $produk01->suplier = "PT. ABC";
8 // Fatal error: Uncaught TypeError: Cannot assign int to property
9 // Smartphone::$suplier of type Perusahaan

```

Di baris 3 saya membatasi property `$suplier` agar hanya bisa diisi object `Perusahaan` saja. Akibatnya, kode di baris 7 akan error karena nilai yang di input adalah string "PT. ABC".

Supaya sesuai dengan syarat tersebut, property `$suplier` hanya bisa diisi dengan object dari class `Perusahaan`:

105f.type_property_object_2.php

```

1 <?php
2 class Perusahaan { }

```

```

3
4 class Smartphone {
5     public Perusahaan $suplier;
6 }
7
8 $suplier01 = new Perusahaan();
9 $produk01 = new Smartphone();
10 $produk01->suplier = $suplier01;

```

Di awal kode program saya membuat class Perusahaan yang saat ini hanya berbentuk class kosong. Tapi itu tidak masalah karena kita hanya ingin membuktikan bahwa property \$suplier hanya bisa diisi dengan object dari class Perusahaan seperti perintah di baris 10. Dalam prakteknya nanti, class Perusahaan bisa saja berisi data yang lebih kompleks.

Null Typed Properties

Dalam kasus tertentu, mungkin saja kita ingin memberikan nilai `null` ke dalam sebuah property. Akan tetapi jika property tersebut sudah di-set tipe datanya, nilai `null` ini akan menghasilkan error:

105g.type_property_null_1.php

```

1 <?php
2 class Smartphone {
3     public Perusahaan $suplier;
4 }
5
6 $produk01 = new Smartphone();
7 $produk01->suplier = null;
8 // Fatal error: Uncaught TypeError: Cannot assign null to property
9 // Smartphone::$suplier of type Perusahaan

```

Solusinya, jika sebuah property boleh diisi dengan nilai `null`, tambah satu karakter tanda tanya "?" sebelum penulisan tipe data seperti contoh berikut:

105h.type_property_null_2.php

```

1 <?php
2 class Smartphone {
3     public ?Perusahaan $suplier;
4 }
5
6 $produk01 = new Smartphone();
7 $produk01->suplier = null;
8
9 var_dump($produk01->suplier); // NULL

```

Di baris 3, typed properties saya tulis sebagai `?Perusahaan`. Kode ini berarti property \$suplier bisa diisi dengan object dari class Perusahaan atau nilai `null`.

Tanda tanya ini juga bisa ditulis ke dalam tipe data dasar:

105i.type_property_null_3.php

```

1 <?php
2 class Smartphone {
3     public ?int $merek;
4 }
5
6 $produk01 = new Smartphone();
7 $produk01->merek = null;
8
9 var_dump($produk01->merek); // NULL

```

Kode program di baris 3 artinya property `$merek` hanya bisa menerima tipe data integer atau nilai `null`.

Union Typed Properties

PHP 8 mengembangkan fitur *typed properties* ini lebih jauh lagi, dimana kita bisa buat beberapa tipe data untuk sebuah property (tidak hanya satu). Sebagai contoh jika property `$jumlah` boleh menerima tipe data float dan integer, bisa ditulis sebagai berikut:

105j.type_property_union.php

```

1 <?php
2 declare(strict_types=1);
3
4 class Smartphone {
5     public float|int $jumlah;
6 }
7
8 $produk01 = new Smartphone();
9
10 $produk01->jumlah = 100;
11 var_dump($produk01->jumlah); // int(100)
12
13 $produk01->jumlah = 40.5;
14 var_dump($produk01->jumlah); // float(40.5)
15
16 $produk01->jumlah = "100"; // // Fatal error: Uncaught TypeError
17 var_dump($produk01->jumlah);

```

Perintah `public float|int $jumlah` di baris 5 berarti property `$jumlah` boleh diisi dengan tipe data float atau int. Dengan demikian kita bisa isi dengan angka 100 dan 40.5, namun tidak dengan string "100".

3.11. Nullsafe Operator

PHP 8 memperkenalkan fitur baru yang disebut **nullsafe operator**. Ini tidak lain pengembangan dari *null coalescing operator* yang sudah lebih dahulu hadir di PHP 7.

Sebagai pengingat, null coalescing operator bisa dipakai untuk memeriksa apakah sebuah variabel berisi nilai atau tidak:

```
echo $produk ?? $smartphone ?? "Laptop"; // Laptop
```

Kode program di atas akan memeriksa apakah variabel `$produk` terdefinisi atau tidak, jika iya maka nilainya akan ditampilkan, namun jika tidak akan masuk ke pemeriksaan variabel `$smartphone`. Jika `$smartphone` tidak terdefinisi juga, maka kembalikan string "Laptop".

Pembahasan lebih detail tentang *null coalescing operator* tersedia di buku [PHP Uncover](#).

Salah satu kekurangan *null coalescing operator* adalah tidak bisa dipakai untuk hasil pemanggilan method. Agar sampai ke pengertian ini kita akan bahas secara bertahap.

Pertama, silahkan pelajari sejenak kode program berikut ini:

`105k.nullsafe_operator_normal.php`

```
1 <?php
2 class Perusahaan {
3     public function __construct(public $nama, public $kota) {
4     }
5     public function getInfo(){
6         return "$this->nama dari kota $this->kota";
7     }
8 }
9
10 class Produk {
11     public ?Perusahaan $suplier;
12
13     public function __construct($suplier) {
14         $this->suplier = $suplier;
15     }
16 }
17
18 $perusahaan01 = new Perusahaan("PT. ABC","Bandung");
19 $produk01 = new Produk($perusahaan01);
20
21 echo $produk01->suplier->getInfo(); // PT. ABC dari kota Bandung
```

Di baris 2 – 8 terdapat pendefinisian class `Perusahaan`. Class ini memiliki constructor yang ditulis dengan format *constructor property promotion* (sudah kita bahas di akhir bab sebelumnya). Dengan demikian, parameter `$nama` dan `$kota` secara otomatis sudah menjadi property.

Selain itu terdapat juga method `getInfo()` yang akan mengembalikan string "`$this->nama dari kota $this->kota`".

Kemudian di baris 10 – 16 saya mendefinisikan class `Smartphone`. Class ini memiliki 1 property `$suplier` yang di set dengan tipe "`?Perusahaan`" (praktek dari *null typed properties*). Ini artinya

variabel `$suplier` hanya bisa diisi dengan object dari class Perusahaan atau nilai null. Proses pengisian property `$suplier` dilakukan dari constructor class Produk.

Di baris 18, variabel `$perusahaan01` diisi object dari hasil instansiasi class Perusahaan ("PT. ABC", "Bandung"). Selanjutnya variabel `$perusahaan01` ini menjadi nilai input ke dalam perintah instansiasi class Produk() di baris 19.

Sampai di sini, variabel `$produk01` sudah bisa dipakai untuk mengakses `$produk01->suplier->getInfo()` yang akan menampilkan string " PT. ABC dari kota Bandung".

Semuanya sudah berjalan normal. Akan tetapi jika ternyata variabel `$suplier` berisi nilai null, maka perintah `$produk01->suplier->getInfo()` di baris 21 akan menghasilkan error:

1051.nullsafe_operator_null.php

```

1  <?php
2  class Perusahaan {
3      public function __construct(public $nama, public $kota) {
4          }
5      public function getInfo(){
6          return "$this->nama dari kota $this->kota";
7      }
8  }
9
10 class Produk {
11     public ?Perusahaan $suplier;
12
13     public function __construct($suplier) {
14         $this->suplier = $suplier;
15     }
16 }
17
18 $perusahaan01 = new Perusahaan("PT. ABC", "Bandung");
19 $produk01 = new Produk(null);
20
21 echo $produk01->suplier->getInfo();
22 // Fatal error: Uncaught Error: Call to a member function getInfo() on null

```

Di baris 19 saya sengaja mengisi nilai null saat pembuatan object class Produk. Dalam prakteknya nanti, bisa saja nilai tersebut berasal dari luar kode program sehingga kita tidak bisa memastikan apakah object Produk dibuat dengan nilai yang bisa diakses, atau hanya nilai null.

Jika class Produk di instansiasi dengan nilai null, maka property `$suplier` juga berisi null sehingga perintah `$produk01->suplier->getInfo()` akan menghasilkan error. Error terjadi karena kita sedang mengakses method dari object yang tidak ada (object null).

Salah satu solusi dari masalah ini adalah dengan memeriksa isi property `$suplier` sebelum menjalankan `$produk01->suplier->getInfo()`:

105m.nullsafe_operator_check.php

```

1 <?php
2 ...
3 ...
4
5 $perusahaan01 = new Perusahaan("PT. ABC", "Bandung");
6 $produk01 = new Produk(null);
7
8 if ($produk01->suplier != null) {
9     echo $produk01->suplier->getInfo();
10 }

```

Sekarang perintah `echo $produk01->suplier->getInfo()` di baris 9 hanya akan di jalankan jika `$produk01->suplier` tidak berisi nilai `null`.

Kasus inilah yang menjadi target dari **nullsafe operator**. Caranya, ganti tanda panah " `->`" yang biasa dipakai untuk mengakses method menjadi " `?->`". Berikut contoh prakteknya:

105n.nullsafe_operator.php

```

1 <?php
2 ...
3 ...
4
5 $perusahaan01 = new Perusahaan("PT. ABC", "Bandung");
6 $produk01 = new Produk(null);
7
8 echo $produk01->suplier?->getInfo();

```

Kondisi if sebelumnya bisa diwakilkan oleh perintah *nullsafe operator* di baris 8. Perintah ini bisa dibaca: akses method `getInfo()` hanya jika property `suplier` berisi sesuatu. Jika `suplier` berisi nilai `null`, tidak perlu jalankan apa-apa.

Apabila diperlukan, *nullsafe operator* bisa di-chaining seperti contoh berikut:

```
$produk01->suplier?->agent?->karyawan?->getInfo()
```

Nullsafe operator yang kita bahas di sini lebih ke menyingkat penulisan. Jika dirasa masih agak bingung, silahkan pakai pemeriksaan kondisi if biasa.

3.12. Late Static Binding

Late static binding diperkenalkan di PHP 5.3 untuk mengatasi masalah dimana method static tidak mendukung konsep penurunan (*inheritance*). Materi *late static binding* memang sedikit rumit dan akan kita bahas secara perlahan. Sebelum ke sana, mari bahas sejenak tentang **early binding** dan **late binding**.

Secara sederhana, **early binding** adalah sebutan bahwa PHP sudah mengetahui apa yang harus dilakukan di awal (pada saat proses compile).

Perhatikan kode program berikut ini:

106.early_binding.php

```

1 <?php
2 class Produk {
3     protected $merek = "LG";
4
5     public function cekMerek(){
6         return "Produk dengan merek $this->merek tersedia";
7     }
8 }
9
10 class MesinCuci extends Produk {
11     public function cekMerek(){
12         return "Mesin cuci dengan merek $this->merek tersedia";
13     }
14 }
15
16 $produk01 = new Produk;
17 echo $produk01->cekMerek();
18
19 echo "<br>";
20
21 $produk02 = new MesinCuci;
22 echo $produk02->cekMerek();

```

Hasil kode program:

```

Produk dengan merek LG tersedia
Mesin cuci dengan merek LG tersedia

```

Dalam kode di atas terdapat 2 class: Produk dan MesinCuci.

Class Produk memiliki property merek yang langsung saya isi dengan string "LG", kemudian terdapat method cekMerek() yang menampilkan string "Produk dengan merek \$this->merek tersedia". Karena property merek sudah berisi string "LG", maka teks yang tampil akan menjadi "Produk dengan merek LG tersedia".

Class MesinCuci saya set sebagai turunan dari class Produk. Dengan demikian, class ini juga akan memiliki property merek dan cekMerek(). Namun di dalam class MesinCuci, method cekMerek() saya definisikan ulang, isinya sekarang menjadi "Mesin cuci dengan merek \$this->merek tersedia". Karena di dalam class ini tidak ada property merek, maka akan diambil dari property merek milik parent class Produk, yakni "LG".

Di baris 16 saya meng-instansiasi class Produk ke dalam object \$produk01 dan menjalankan method cekMerek(). Di baris 21 giliran class MesinCuci saya instansiasi ke dalam object \$produk02 dan menjalankan method cekMerek().

Tanpa melihat proses instansiasi di baris 16 - 22, kita sudah bisa menebak bahwa jika yang dijalankan adalah method cekMerek() milik class Produk, hasil akhirnya adalah "Produk dengan

merek LG tersedia". Dan jika yang dijalankan adalah method cekMerek() milik class MesinCuci, maka hasil akhirnya adalah "Mesin cuci dengan merek LG tersedia".

Di sini PHP bisa "menebak" hasil akhir dari setiap pemanggilan method cekMerek(). Inilah contoh dari **early binding** atau disebut juga dengan **compile time binding**.

Sedangkan untuk **late binding** atau **runtime binding**, PHP harus menunggu object apa yang memanggil untuk bisa menentukan hasil akhir. Berikut contohnya:

107.late_binding.php

```

1 <?php
2 class Produk {
3     protected $merek = "LG";
4
5     public function cekMerek(){
6         return "Produk dengan merek $this->merek tersedia";
7     }
8
9     public function getInfo(){
10        return $this->cekMerek();
11    }
12 }
13
14 class MesinCuci extends Produk {
15     public function cekMerek(){
16         return "Mesin cuci dengan merek $this->merek tersedia";
17     }
18 }
19
20 $produk01 = new Produk;
21 echo $produk01->getInfo();
22
23 echo "<br>";
24
25 $produk02 = new MesinCuci;
26 echo $produk02->getInfo();

```

Hasil kode program:

```

Produk dengan merek LG tersedia
Mesin cuci dengan merek LG tersedia

```

Perubahan dari kode program sebelumnya ada di baris 9 – 12, dimana saya menambah method baru ke dalam class Produk. Method getInfo() ini hanya berisi perintah untuk mengakses method \$this->cekMerek(). Karena class MesinCuci adalah turunan dari class Produk, maka class MesinCuci otomatis juga memiliki method yang sama.

Proses **late binding** terjadi di sini, dimana PHP harus melihat object apa yang menjalankan method getInfo() untuk bisa memproses hasil akhir. Variabel **\$this** dari pemanggilan \$this->cekMerek() ditentukan di akhir pemanggilan.

Jika yang mengakses adalah object dari class `Produk` (baris 21), maka hasil akhirnya menjadi "`Produk dengan merek LG tersedia`". Namun jika yang mengakses adalah object dari class `MesinCuci` (baris 26), hasil akhirnya adalah "`Mesin cuci dengan merek LG tersedia`".

Sekarang kita masuk ke **static late binding**. Istilah ini merupakan sebutan untuk late binding di dalam sebuah static method. Berikut contohnya:

108.late_static_binding_problem.php

```

1 <?php
2 class Produk {
3     protected static $merek = "LG";
4
5     public static function cekMerek(){
6         return "Produk dengan merek ".self::$merek." tersedia";
7     }
8
9     public static function getInfo(){
10        return self::cekMerek();
11    }
12 }
13
14 class MesinCuci extends Produk {
15     public static function cekMerek(){
16         return "Mesin cuci dengan merek ".self::$merek." tersedia";
17     }
18 }
19
20 echo Produk::getInfo();
21 echo "<br>";
22 echo MesinCuci::getInfo();
```

Hasil kode program:

```
Produk dengan merek LG tersedia
Produk dengan merek LG tersedia
```

Kode program di atas mirip seperti contoh sebelumnya, hanya saja semua property dan method saya ubah menjadi **static**. Dengan demikian *pseudo-variable* `$this` harus diganti menjadi `self::`. Selain itu karena `getInfo()` menjadi static method, kita tidak perlu melakukan instansiasi object tapi cukup diakses dengan nama class langsung, yakni `Produk::getInfo()` dan `MesinCuci::getInfo()`.

Namun perhatikan hasil yang tampil. Pemanggilan method `Produk::getInfo()` dan `MesinCuci::getInfo()` sama-sama menghasilkan teks "`Produk dengan merek LG tersedia`". Padahal class yang mengakses method ini berbeda. Seharusnya, hasil dari `MesinCuci::getInfo()` adalah "`Mesin cuci dengan merek LG tersedia`".

Ini terjadi karena operator `self::` seolah-olah "terikat" dengan class tempat method tersebut di definisikan. Method `getInfo()` dibuat di dalam class `Produk`, akibatnya meskipun diakses dari

class turunan, operator `self::` masih mengambil data yang ada di dalam class Produk, bukan dari class turunan. Inilah yang saya maksud bahwa static method tidak mendukung konsep penurunan (*inheritance*).

Masalah ini diatasi PHP dengan memperkenalkan operator baru, yakni "`static::`". Method `getInfo()` bisa ditulis ulang sebagai berikut:

109. late_static_binding.php

```

1 ...
2 public static function getInfo(){
3     return static::cekMerek();
4 }
5 ...
6 ...
7 echo Produk::getInfo();
8 echo "<br>";
9 echo MesinCuci::getInfo();

```

Dengan mengubah perintah `return self::cekMerek()` menjadi `return static::cekMerek()`, sekarang hasilnya akan menjadi:

```

Produk dengan merek LG tersedia
Mesin cuci dengan merek LG tersedia

```

Untuk penggunaan dasar, **static late binding** jarang kita pakai, tapi pengetahuan ini cukup penting untuk dipahami.

3.13. Anonymous Class

Anonymous class adalah class "tanpa nama" (*anonym*), yang digunakan untuk membuat class "sekali pakai". Class *anonym* ini hanya bisa diinstansiasi 1 kali saja, tidak bisa lebih.

Anonymous class sendiri masih relatif baru di PHP (diperkenalkan di PHP 7). Namun dalam bahasa pemrograman seperti JavaScript (terutama jQuery), anonymous class cukup sering dipakai, terutama untuk menulis nilai input ke sebuah method.

Langsung saja kita lihat contoh kode program cara membuat *anonymous class* di dalam PHP:

110. anonymous_class_basic.php

```

1 <?php
2 $produk01 = new class(){};
3
4 var_dump($produk01); // object(class@anonymous)#1 (0) { }

```

Untuk membuat *anonymous class*, kita memakai perintah `new class(){};`. Dalam contoh ini, saya membuat *anonymous class* yang langsung diisi ke dalam variabel `$produk01`. Dari hasil `var_dump()` terlihat bahwa PHP mengenali `$produk01` sebagai object dari "class@anonymous".

Dengan penulisan seperti ini, *anonymous class* hanya bisa di instansiasi 1 kali saja. Karena pembuatan class langsung diisi ke dalam object \$produk01.

Kita juga bisa mengisi *anonymous class* ini dengan berbagai komponen class "normal" seperti property dan method. Caranya, tulis isi class di antara tanda kurung kurawal:

111. anonymous_class_property_method.php

```

5  <?php
6  $produk01 = new class(){
7      private $merek = "Polytron";
8      public function getMerek(){
9          return $this->merek;
10     }
11 };
12
13 echo $produk01->getMerek(); // Polytron
14
15 echo $produk01->merek;
16 // Fatal error: Uncaught Error: Cannot access private property
17 // class@anonymous::$merek

```

Di sini saya menambah property `merek` dan method `getMerek()` ke dalam *anonymous class*. Perintah di baris 15 akan menghasilkan error karena saya mencoba mengakses property `private merek` dari luar class. Ini menunjukkan bahwa *anonymous class* berfungsi sebagaimana layaknya class biasa.

Lebih jauh lagi, *anonymous class* juga bisa memiliki **constructor**:

112. anonymous_class_construct.php

```

1  <?php
2  $produk01 = new class("Sony"){
3      private $merek;
4
5      public function __construct($foo){
6          $this->merek = $foo;
7      }
8
9      public function getMerek(){
10         return $this->merek;
11     }
12 };
13
14 echo $produk01->getMerek(); // Sony

```

Di baris 2, *anonymous class* saya buat dengan perintah `new class("Sony")`. String "Sony" di sini nantinya akan dikirim ke dalam constructor di baris 5 – 7, yang kemudian menjadi nilai awal untuk property `merek`.

Selain itu, *anonymous class* juga bisa diturunkan dari class lain serta mengimplementasikan

interface:

113.anonymous_class_inheritance.php

```

1 <?php
2 class Produk {
3     protected $merek;
4
5     public function __construct($foo){
6         $this->merek = $foo;
7     }
8 }
9
10 interface PunyaMerek {
11     public function getMerek();
12 }
13
14 $produk01 = new class("Hitachi") extends Produk implements PunyaMerek{
15     public function getMerek(){
16         return $this->merek;
17     }
18 };
19
20 echo $produk01->getMerek(); // Hitachi

```

Di awal kode program saya membuat class `Produk` serta sebuah interface `PunyaMerek`. Class `Produk` sendiri memiliki property `merek` dan sebuah constructor. Sedangkan interface `PunyaMerek` berisi *signature* dari method `getMerek()`.

Class `Produk` dan interface `PunyaMerek` ini selanjutnya saya turunkan ke dalam *anonymous class* di baris 14. Cara penulisannya kurang lebih sama seperti class biasa, yakni menggunakan perintah "`extends Produk implements PunyaMerek`".

Karena anonymous class memakai interface `PunyaMerek`, maka di dalam class anonym harus ditulis ulang method `getMerek()`. Method ini saya isi dengan perintah `return $this->merek` (baris 15 – 17).

Pada prakteknya, anonymous class biasa dipakai sebagai nilai input untuk sebuah argument. Perhatikan contoh fungsi `tampilkanSmartphone()` berikut ini:

114.anonymous_class_argument.php

```

1 <?php
2 function tampilkanSmartphone($hp){
3     return "Smartphone ".$hp->merek." ".$hp->tipe." di jual seharga Rp. "
4         .number_format($hp->harga,2,",",".");
5 }

```

Fungsi `tampilkanSmartphone()` butuh argument berupa object yang memiliki property `merek`, `tipe` dan `harga`, karena di dalam fungsi ini terdapat perintah untuk menampilkan `$hp->merek`, `$hp->tipe` dan `$hp->harga`. Fungsi ini sebenarnya sudah kita pakai pada saat pembahasan

type hinting.

Biasanya, untuk mengisi nilai argument \$hp kita harus membuat sebuah class terlebih dahulu, misalnya class Produk, kemudian membuat object dari class Produk ini dan baru menginputnya sebagai argument untuk fungsi tampilanSmartphone().

Atau sebagai alternatif, argument \$hp ini bisa diisi dengan object dari anonymous class:

114. anonymous_class_argument.php

```
1 echo tampilanSmartphone(new class{
2     public $merek= "Vivo";
3     public $tipe= "V11";
4     public $harga= "3599000";
5});
```

Hasil kode program:

Smartphone Vivo V11 di jual seharga Rp. 3.599.000,00

Dengan pemanggilan seperti ini, class anonym akan menjadi nilai awal untuk argument \$hp.

Menggunakan anonymous class seperti ini memang terlihat agak repot, tapi bisa jadi solusi untuk kasus-kasus tertentu.

3.14. stdClass

PHP memiliki prototype class atau class standard yang bernama **stdClass**. Ini adalah class bawaan PHP yang jika dipakai langsung, tidak berisi apapun (class kosong). Namun jika terdapat kode program yang dikonversi menjadi object, object tersebut akan berada di bawah class stdClass.

Berikut contoh pembuatan object dari class stdClass:

115. std_class_basic.php

```
1 <?php
2 $produk01 = new stdClass;
3 var_dump($produk01);
```

Hasil kode program:

object(stdClass)#1 (0) { }

Tidak berbeda seperti pembuatan object biasa. Dan object hasil dari stdClass ini juga berfungsi sebagaimana object biasa, misalnya kita bisa ditambah dengan property:

116. std_class_property.php

```
1 <?php
```

```

2 $produk01 = new stdClass;
3 $produk01->merek = "Oppo";
4 $produk01->tipe = "Find X";
5 $produk01->harga = 13499000;
6
7 echo $produk01->merek."<br>";
8 echo $produk01->tipe."<br>";
9 echo $produk01->harga."<br>";
10 echo "<br>";
11 var_dump($produk01);

```

Hasil kode program:

```

Oppo
Find X
13499000

```

```
object(stdClass)#1 (3) { ["merek"]=> string(4) "Oppo" ["tipe"]=> string(6) "Find X"
["harga"]=> int(13499000) }
```

Dalam contoh ini object \$produk01 saya tambah dengan 3 property: merek, tipe dan harga. Ini bisa dilakukan karena secara bawaan, semua object bisa ditambah dengan property baru walaupun di tulis dari luar class.

Pada prakteknya, kita jarang membuat langsung `stdClass` seperti ini. `stdClass` biasanya di dapat dari hasil pemrosesan PHP. Sebagai contoh, di dalam PHP terdapat perintah untuk `type casting`, yakni mengubah suatu tipe data menjadi tipe data lain. Jika kita mengubah sebuah tipe data menjadi object maka object tersebut adalah object dari class **`stdClass`**.

Berikut contohnya:

117. `std_class_type_casting_string.php`

```

1 <?php
2 $foo = "Belajar OOP PHP";
3 $bar = (object) $foo;
4
5 var_dump($bar);

```

Hasil kode program:

```
object(stdClass)#1 (1) { ["scalar"]=> string(15) "Belajar OOP PHP" }
```

Di sini saya mengkonversi variabel \$foo yang berisi string menjadi object. Object ini kemudian disimpan ke dalam variabel \$bar.

Hasil dari `var_dump()` memperlihatkan bahwa \$bar berisi sebuah object dari class `stdClass`. Kita juga bisa perhatikan bahwa jika sebuah string di konversi menjadi object, string tersebut akan menjadi nilai dari property "scalar":

118. std_class_type_casting_string_2.php

```
1 <?php
2 $foo = "Belajar OOP PHP";
3 $bar = (object) $foo;
4
5 echo $bar->scalar; // Belajar OOP PHP
```

Property "scalar" ini berlaku untuk semua tipe data dasar, yakni string, integer, float dan boolean.

Jika tipe data asal adalah *associative array*, maka nama index akan menjadi nama property. Berikut contohnya:

119. std_class_type_casting_array.php

```
1 <?php
2 $produk = ["merek" => "Oppo", "tipe" => "Find X", "harga" => 13499000];
3 $produk01 = (object) $produk;
4
5 echo $produk01->merek."<br>";
6 echo $produk01->tipe."<br>";
7 echo $produk01->harga."<br>";
8
9 var_dump($produk01 instanceof StdClass);
```

Hasil kode program:

```
Oppo
Find X
13499000
bool(true)
```

Hasil kode program di baris 9 memperlihatkan bahwa variabel \$produk01 berisi object yang menjadi bagian dari `stdClass`, di mana perintah `$produk01 instanceof StdClass` menghasilkan nilai **true**.

Kesimpulannya, `stdClass` adalah class khusus yang menjadi "class penampung" dari object yang di generate oleh PHP.

Materi tentang `stdClass` ini menutup bab tentang Advanced OOP PHP. Pada bab berikutnya kita masih membahas materi lanjutan OOP PHP, namun sengaja saya pisah agar bab ini tidak terlalu panjang.

4. Class dan Object Function

Dalam bab ini kita akan membahas beberapa fungsi atau function bawaan PHP yang berhubungan dengan class dan object. Daftar fungsi ini saya ambil dari php.net/manual/en/ref.classobj.php, berikut listnya:

- __autoload
- call_user_method_array
- call_user_method
- class_alias
- class_exists
- get_called_class
- get_class_methods
- get_class_vars
- get_class
- get_declared_classes
- get_declared_interfaces
- get_declared_traits
- get_object_vars
- get_parent_class
- interface_exists
- is_a
- is_object
- is_subclass_of
- method_exists
- property_exists
- trait_exists

Sebagian besar dari fungsi ini dipakai untuk mengambil informasi seputar class dan object, seperti apa saja daftar method yang tersedia di dalam sebuah class, atau memeriksa apakah sebuah object merupakan turunan dari class lain.

Karena banyak fungsi yang berperilaku mirip satu sama lain, beberapa akan di bahas secara bersamaan.

Khusus untuk konstanta `__autoload`, fungsi `call_user_method_array()` dan fungsi `call_user_method()`, tidak saya bahas karena sudah berstatus *deprecated* dan tidak tersedia lagi di PHP 7 dan PHP 8.

4.1. Function `is_object()`

Fungsi `is_object()` dipakai untuk memeriksa apakah sebuah variabel berisi object atau tidak. Fungsi ini butuh sebuah argument, yakni variabel yang akan diperiksa.

Fungsi `is_object()` mengembalikan nilai **true** jika variabel yang diperiksa berisi object, atau nilai **false** jika variabel tersebut bukan berisi object.

Berikut contoh penggunaannya:

```
01.is_object.php

10 <?php
11 class Produk { }
12 $produk01 = new Produk;
13 $produk02 = [1, 2, 3];
14
15 var_dump(is_object($produk01));
16 echo "<br>";
17 var_dump(is_object($produk02));
```

Hasil kode program:

```
bool(true)
bool(false)
```

Di sini saya membuat class `Produk` yang tidak berisi apa-apa (karena memang tidak diperlukan pada contoh kita). Variabel `$produk01` kemudian diisi dengan object hasil instansiasi class `Produk`, sedangkan variabel `$produk02` diisi sebuah array.

Hasilnya, perintah `is_object($produk01)` mengembalikan nilai boolean **true**, dan perintah `is_object($produk02)` mengembalikan nilai boolean **false**.

4.2. Function `is_a()`

Fungsi `is_a()` dipakai untuk memeriksa apakah sebuah object berasal dari class tertentu atau turunan dari class tersebut. Fungsi ini butuh 2 argument: variabel yang akan diperiksa, serta string yang berisi nama class.

Berikut contohnya:

02.is_a.php

```
1 <?php
2 class Produk { }
3 interface LaptopGaming { }
4 class Laptop extends Produk implements LaptopGaming { }
5
6 $laptop01 = new Laptop;
7
8 var_dump(is_a($laptop01,"Laptop"));
9 echo "<br>";
10 var_dump(is_a($laptop01,"Produk"));
11 echo "<br>";
12 var_dump(is_a($laptop01,"LaptopGaming"));
13 echo "<br>";
14 var_dump(is_a($laptop01,"Televisi"));
```

Hasil kode program:

```
bool(true)
bool(true)
bool(true)
bool(false)
```

Di awal kode program saya membuat class `Produk`, interface `LaptopGaming`, serta class `Laptop`. Class `Laptop` sendiri merupakan turunan dari class `Produk` serta menerapkan interface `LaptopGaming`.

Di baris 6 class `Laptop` di instansiasi ke dalam variabel `$laptop01`. Kemudian di baris 8 – 9 saya memeriksa isi dari variabel `$laptop01` dengan 4 kali fungsi `is_a()`.

Fungsi `is_a($laptop01,"Laptop")` artinya apakah variabel `$laptop01` berisi object yang merupakan atau turunan dari class `Laptop`? Betul, maka hasilnya **true**. Begitu pula dengan perintah `is_a($laptop01,"Produk")` dan `is_a($laptop01,"LaptopGaming")`. Perhatikan bahwa fungsi `is_a()` juga akan mengembalikan nilai **true** untuk class yang menerapkan interface.

Khusus di baris terakhir, fungsi `is_a($laptop01,"Televisi")` mengembalikan nilai **false** karena variabel `$laptop01` bukanlah turunan dari class `Televisi`.

Fungsi `is_a()` ini sebenarnya adalah penulisan lain dari perintah `instanceof`. Kode di atas juga bisa diubah dengan perintah `instanceof`:

03.instanceof.php

```
1 <?php
2 class Produk { }
3 interface LaptopGaming { }
4 class Laptop extends Produk implements LaptopGaming { }
5
6 $laptop01 = new Laptop;
7
8 var_dump($laptop01 instanceof Laptop);
```

```
9 echo "<br>";
10 var_dump($laptop01 instanceof Produk);
11 echo "<br>";
12 var_dump($laptop01 instanceof LaptopGaming);
13 echo "<br>";
14 var_dump($laptop01 instanceof Televisi);
```

Hasil kode program:

```
bool(true)
bool(true)
bool(true)
bool(false)
```

Anda apakah ingin menggunakan fungsi `is_a()` atau perintah `instanceof`.

4.3. Function `is_subclass_of()`

Fungsi `is_subclass_of()` mirip seperti fungsi `is_a()`, bedanya fungsi `is_subclass_of()` akan mengembalikan nilai `false` jika variabel yang diperiksa merupakan class saat ini.

Berikut contoh prakteknya:

04.is_subclass_of.php

```
1 <?php
2 class Produk { }
3 interface LaptopGaming { }
4 class Laptop extends Produk implements LaptopGaming { }
5
6 $laptop01 = new Laptop;
7
8 var_dump(is_subclass_of($laptop01, "Laptop"));
9 echo "<br>";
10 var_dump(is_subclass_of($laptop01, "Produk"));
11 echo "<br>";
12 var_dump(is_subclass_of($laptop01, "LaptopGaming"));
13 echo "<br>";
14 var_dump(is_subclass_of($laptop01, "Televisi"));
```

Hasil kode program:

```
bool(false)
bool(true)
bool(true)
bool(false)
```

Contoh di atas nyaris sama seperti sebelumnya, hanya saja sekarang saya menggunakan fungsi `is_subclass_of()`.

Perhatikan hasil dari pemanggilan `is_subclass_of($laptop01, "Laptop")` di baris 8, hasilnya

false karena variabel \$laptop01 berisi object dari class Laptop. Fungsi `is_subclass_of()` hanya mengembalikan nilai **true** jika object yang diperiksa merupakan turunan dari class yang diperiksa, tapi bukan dari class tersebut.

4.4. Function `get_class_*()`

Fungsi yang diawali dengan `get_class_` dipakai untuk mencari informasi mengenai sebuah class atau object. Kita akan langsung membahas 4 fungsi karena mirip satu sama lain yakni:

- `get_class()`
- `get_class_vars()`
- `get_class_methods()`
- `get_object_vars()`

Fungsi `get_class()` dipakai untuk mencari nama class dari sebuah object atau variabel. Fungsi ini butuh argument berupa variabel yang akan dicari nama class-nya. Hasilnya berupa string yang berisi nama class.

Fungsi `get_class_vars()` dipakai untuk mencari daftar property yang ada di dalam sebuah class. Fungsi ini butuh sebuah argument berupa string nama class. Hasilnya berupa array dari nama property yang ada.

Fungsi `get_class_methods()` dipakai untuk mencari daftar method yang ada di dalam sebuah class. Fungsi ini butuh sebuah argument berupa string nama class. Hasilnya berupa array dari nama method yang ada.

Fungsi `get_object_vars()` dipakai untuk mencari daftar property yang ada di dalam sebuah object. Fungsi ini butuh sebuah argument berupa variabel. Hasilnya berupa array dari nama property yang ada.

Berikut contoh praktik dari ke-4 fungsi ini:

05.get_class.php

```

1 <?php
2 class Produk {
3     public $merek = "Asus";
4     public $tipe = "ROG GX800";
5     protected $harga = 95000000;
6
7     public function getMerek() {
8         return $this->merek;
9     }
10
11    public function cekInfo() {
12        return $this->merek." ".$this->tipe." ".$this->harga;
13    }
14 }
15

```

```

16 $produk01 = new Produk;
17
18 echo "<b>Get Class</b>: ";
19 print_r(get_class($produk01));
20 echo "<br>";
21
22 echo "<b>Get Class Vars</b>: ";
23 print_r(get_class_vars("Produk"));
24 echo "<br>";
25
26 echo "<b>Get Class Methods</b>: ";
27 print_r(get_class_methods("Produk"));
28 echo "<br>";
29
30 echo "<b>Get Object Vars</b>: ";
31 print_r(get_object_vars($produk01));
32 echo "<br>";

```

Hasil kode program:

```

Get Class: Produk
Get Class Vars: Array ( [merek] => Asus [tipe] => ROG GX800 )
Get Class Methods: Array ( [0] => getMerek [1] => cekInfo )
Get Object Vars: Array ( [merek] => Asus [tipe] => ROG GX800 )

```

Pada awal kode program saya membuat class `Produk` dengan 3 property: `merek`, `tipe` dan `harga`. Khusus untuk property `harga` di set sebagai `private`. Class `Produk` ini juga memiliki method `getMerek()` dan `cekInfo()`. Class `Produk` kemudian di instansiasi ke dalam variabel `$produk01`.

Di baris 19, saya menjalankan fungsi `get_class($produk01)`. Hasilnya berupa string "Produk", yakni nama class dari object `$produk01`.

Di baris 23 saya menjalankan fungsi `get_class_vars("Produk")`. Hasilnya berupa array dari 2 property milik class `Produk`, yakni `merek` dan `tipe`. Property `harga` tidak tampil karena tidak bisa diakses dari luar class (menggunakan hak akses `protected`).

Di baris 27 saya menjalankan fungsi `get_class_methods("Produk")`. Hasilnya berupa array dari 2 method milik class `Produk`, yakni `getMerek()` dan `cekInfo()`.

Terakhir di baris 31 saya menjalankan fungsi `get_object_vars($produk01)`. Hasilnya mirip seperti `get_class_vars("Produk")`, yakni array yang berisi nama 2 property dari class `Laptop`: `merek` dan `tipe`.

Fungsi-fungsi ini terlihat tidak terlalu berguna karena isi dari class `Produk` bisa kita lihat langsung. Akan tetapi jika object tersebut bukan berasal dari kode program kita, fungsi ini bisa membantu. Berikut contohnya:

06.get_class_stdclass.php

```

1 <?php

```

```
2 $foo = "Belajar OOP PHP";
3 $bar = (object) $foo;
4
5 echo get_class($bar); // stdClass
```

Di sini saya menggunakan fungsi `get_class()` untuk mencari tau apa nama class dari variabel `$bar`. Ternyata variabel tersebut berisi object dari class `stdClass`.

4.5. Function `get_parent_class()`

Sesuai dengan namanya, fungsi `get_parent_class()` dipakai untuk mencari tau nama *parent class* dari class yang diperiksa. Fungsi ini butuh 1 argument berupa string dari nama class yang diperiksa dan mengembalikan nilai string nama class yang menjadi parent class (jika ada).

Berikut contohnya:

07.get_perent_class.php

```
1 <?php
2 class Produk { }
3 class Komputer extends Produk { }
4 class Laptop extends Komputer { }
5
6 echo "<b>Get Class</b>: ";
7 print_r(get_parent_class("Laptop"));
8 echo "<br>";
9
10 echo "<b>Get Class</b>: ";
11 var_dump(get_parent_class("Produk"));
12 echo "<br>";
```

Hasil kode program:

```
Get Class: Komputer
Get Class: bool(false)
```

Di sini saya membuat 3 buah class, yakni class `Produk`, `Komputer` dan `Laptop`. Class `Komputer` merupakan turunan dari class `Produk`, dan class `Laptop` merupakan turunan dari class `Komputer`.

Fungsi `get_parent_class("Laptop")` mengembalikan nilai "`Komputer`", yakni nama parent classnya. Sedangkan `get_parent_class("Produk")` mengembalikan nilai false karena class `Produk` tidak memiliki parent class.

4.6. Function `get_declared_*()`

Fungsi yang diawali dengan `get_declared_` dipakai untuk mencari informasi apa saja nama class, interface dan trait yang bisa diakses. Terdapat 3 fungsi yang tergabung dalam kelompok

ini:

- `get_declared_classes()`
- `get_declared_interfaces()`
- `get_declared_traits()`

Fungsi `get_declared_classes()` dipakai untuk menampilkan daftar nama class yang bisa diakses saat ini. Fungsi `get_declared_interfaces()` dipakai untuk menampilkan daftar nama interface, dan fungsi `get_declared_traits()` dipakai untuk menampilkan daftar nama trait.

Daftar nama class, interface dan trait ini tidak hanya yang ditulis di halaman saat ini, tapi seluruhnya termasuk bawaan PHP. Ketiganya tidak butuh argument apapun.

Berikut contoh pemanggilan ketiga fungsi ini:

08.get_declared.php

```
1 <?php
2 class Produk { }
3 interface LaptopGaming { }
4 trait LowWatt { }
5
6 class Laptop extends Produk implements LaptopGaming{ }
7
8 echo "<h1>Get Declared Class</h1>";
9 echo "<pre>";
10 print_r(get_declared_classes());
11 echo "</pre>";
12 echo "<br><hr>";
13
14 echo "<h1>Get Declared Interface</h1>";
15 echo "<pre>";
16 print_r(get_declared_interfaces());
17 echo "</pre>";
18 echo "<br><hr>";
19
20 echo "<h1>Get Declared Trait</h1>";
21 echo "<pre>";
22 print_r(get_declared_traits());
23 echo "</pre>";
```

Hasil kode program:

Get Declared Class

```
Array
(
    [0] => stdClass
    [1] => Exception
    ...
    [121] => PDOException
    [122] => PDO
    ....
```

Class dan Object Function

```
[131] => mysqli_sql_exception  
[132] => mysqli_driver  
[133] => mysqli  
[134] => mysqli_warning  
[135] => mysqli_result  
...  
[141] => Produk  
[142] => Laptop  
)
```

Get Declared Interface

```
Array  
(  
    [0] => Traversable  
    [1] => IteratorAggregate  
    [2] => Iterator  
    ...  
    [17] => SessionUpdateTimestampHandlerInterface  
    [18] => LaptopGaming  
)
```

Get Declared Trait

```
Array  
(  
    [0] => LowWatt  
)
```

Yup, hasil kode program di atas menampilkan **seluruh class, interface dan trait** bawaan PHP. Setidaknya terdapat 142 class, 18 interface dan 1 trait.

Fungsi `get_declared_classes()` memperlihatkan daftar semua class bawaan PHP, yang diikuti dengan nama class `Produk` dan `Laptop` yang memang saya buat di dalam kode program. Daftar class ini bisa berbeda-beda tergantung pengaturan PHP. Jika saya mengaktifkan extension atau menggunakan library PHP lain, maka daftar class ini bisa bertambah.

Dari hasil fungsi `get_declared_interfaces()` bisa terlihat bahwa PHP juga memiliki beberapa interface bawaan. Namun hasil fungsi `get_declared_traits()` memperlihatkan bahwa PHP tidak memiliki trait bawaan. Fungsi ini hanya menampilkan trait `LowWatt` yang berasal dari kode program kita.

4.7. Function * _exists()

Terdapat 5 fungsi yang diakhiri dengan kata `_exists()`, semuanya dipakai untuk memeriksa apakah sebuah **class, interface, trait, property** dan **method** tersedia atau tidak. Berikut fungsi yang akan kita bahas:

- `class_exists()`
- `interface_exists()`

- `trait_exists()`
- `property_exists()`
- `method_exists()`

Dari namanya, saya yakin anda sudah bisa menebak kegunaan dari setiap fungsi ini. Kelimanya akan mengembalikan nilai boolean **true** atau **false** tergantung apakah class / interface / trait / property / method yang sedang diperiksa tersedia atau tidak.

Sebagai contoh class, saya akan memakai kode berikut:

09.exist.php

```
1 <?php
2 class Produk { }
3 interface LaptopGaming { }
4 trait LowWatt { }
5
6 class Laptop extends Produk implements LaptopGaming{
7     use LowWatt;
8
9     public $merek = "Asus";
10    public $tipe = "ROG GX800";
11    protected $harga = 95000000;
12
13    public function getMerek() {
14        return $this->merek;
15    }
16
17    public function cekInfo() {
18        return $this->merek." ".$this->tipe." ".$this->harga;
19    }
20 }
```

Di awal kode program saya membuat class `Produk`, interface `LaptopGaming`, serta trait `LowWatt`. Ketiganya tidak berisi kode apa-apa karena memang tidak kita butuhkan untuk bahasan kali ini.

Kemudian di baris 6 saya membuat class `Laptop` yang merupakan turunan dari class `Produk`, meng-implementasikan interface `LaptopGaming` serta menggunakan trait `LowWatt`. Di dalam class `Produk` ini terdapat 3 property: `merek`, `tipe` dan `harga` serta 2 buah method: `getMerek()` dan `cekInfo()`. Class `Laptop` ini akan kita pakai untuk membahas kelima fungsi `*_exist()`.

Fungsi `class_exists()`

Fungsi `class_exists()` dipakai untuk memeriksa apakah sebuah class tersedia atau tidak. Fungsi ini butuh sebuah argument berbentuk string dari nama class. Berikut contoh penggunaannya:

```
1 echo "<b>class_exists('Laptop')</b>: ";
2 var_dump(class_exists('Laptop'));
```

```
3 echo "<br>";  
4  
5 echo "<b>class_exists('Televisi')</b>: ";  
6 var_dump(class_exists('Televisi'));  
7 echo "<br>";  
8  
9 echo "<b>class_exists('PDO')</b>: ";  
10 var_dump(class_exists('PDO'));  
11 echo "<br>";
```

Hasil kode program:

```
class_exists('Laptop'): bool(true)  
class_exists('Televisi'): bool(false)  
class_exists('PDO'): bool(true)
```

Dalam kode program di atas saya memeriksa apakah class Laptop, Televisi dan PDO tersedia.

Pemanggilan fungsi `class_exists('Laptop')` menghasilkan nilai **true** karena ada di dalam kode program. Pemanggilan fungsi `class_exists('Televisi')` menghasilkan **false** karena memang tidak tersedia. Kemudian pemanggilan fungsi `class_exists('PDO')` menghasilkan nilai **true** karena class **PDO** memang tersedia sebagai class bawaan PHP.

Fungsi `interface_exists()`

Fungsi `interface_exists()` dipakai untuk memeriksa apakah sebuah interface tersedia atau tidak. Fungsi ini butuh sebuah argument berbentuk string dari nama interface yang akan diperiksa. Berikut contohnya:

```
1 echo "<b>interface_exists('LaptopGaming')</b>: ";  
2 var_dump(interface_exists('LaptopGaming'));  
3 echo "<br>";  
4  
5 echo "<b>interface_exists('SmartphoneGaming')</b>: ";  
6 var_dump(interface_exists('SmartphoneGaming'));  
7 echo "<br>";  
8  
9 echo "<b>interface_exists('Traversable')</b>: ";  
10 var_dump(interface_exists('Traversable'));  
11 echo "<br>";
```

Hasil kode program:

```
interface_exists('LaptopGaming'): bool(true)  
interface_exists('SmartphoneGaming'): bool(false)  
interface_exists('Traversable'): bool(true)
```

Di sini saya ingin memeriksa apakah interface LaptopGaming, SmartphoneGaming dan Traversable tersedia atau tidak.

Pemanggilan fungsi `interface_exists('LaptopGaming')` menghasilkan nilai **true** karena

terdapat di dalam kode program. Fungsi `interface_exists('SmartphoneGaming')` menghasilkan **false** karena tidak ada di dalam kode program, serta pemanggilan fungsi `interface_exists('Traversable')` menghasilkan **true** karena merupakan interface bawaan PHP.

Fungsi `trait_exists()`

Fungsi `trait_exists()` dipakai untuk memeriksa apakah sebuah trait tersedia atau tidak. Fungsi ini butuh sebuah argument berbentuk string dari nama trait yang akan diperiksa. Berikut contoh penggunaannya:

```

1 echo "<b>trait_exists('LowWatt')</b>: ";
2 var_dump(trait_exists('LowWatt'));
3 echo "<br>";
4
5 echo "<b>trait_exists('HighPerformance')</b>: ";
6 var_dump(trait_exists('HighPerformance'));
7 echo "<br>";

```

Hasil kode program:

```

trait_exists('LowWatt'): bool(true)
trait_exists('HighPerformance'): bool(false)

```

Pemanggilan fungsi `trait_exists('LowWatt')` menghasilkan nilai **true** karena terdapat di dalam kode program sedangkan hasil dari fungsi `trait_exists('HighPerformance')` menghasilkan **false** karena tidak tersedia.

Fungsi `property_exists()`

Fungsi `property_exists()` dipakai untuk memeriksa apakah sebuah property tersedia atau tidak di dalam sebuah class. Fungsi ini butuh 2 buah argument berbentuk string. Argument pertama berupa nama class, dan argument kedua berupa nama property yang ingin dicari.

Berikut contoh penggunaannya:

```

1 echo "<b>property_exists('Laptop','merek')</b>: ";
2 var_dump(property_exists('Laptop','merek'));
3 echo "<br>";
4
5 echo "<b>property_exists('Laptop','harga')</b>: ";
6 var_dump(property_exists('Laptop','harga'));
7 echo "<br>";
8
9 echo "<b>property_exists('mysqli','affected_rows')</b>: ";
10 var_dump(property_exists('mysqli','affected_rows'));
11 echo "<br>";

```

Hasil kode program:

```
property_exists('Laptop','merek'): bool(true)
property_exists('Laptop','harga'): bool(true)
property_exists('mysqli','affected_rows'): bool(true)
```

Di baris 2 saya menjalankan fungsi `property_exists('Laptop','merek')` untuk memeriksa apakah di dalam class `Laptop` terdapat property `merek`. Hasilnya **true** karena ini ada di kode program.

Di baris 6 saya menjalankan fungsi `property_exists('Laptop','harga')`. Ini dipakai untuk memeriksa apakah di dalam class `Laptop` terdapat property `harga`. Hasilnya juga **true** karena ini ada di kode program. Perhatikan bahwa property `harga` ini sebenarnya berstatus `protected`, sehingga tidak bisa diakses dari luar class. Namun fungsi `property_exists()` tetap mengembalikan nilai **true**.

Di baris 10 terdapat pemanggilan fungsi `property_exists('mysqli','affected_rows')`. Fungsi ini berarti saya ingin memeriksa apakah di dalam class `mysqli` terdapat property yang bernama `affected_rows`. Hasilnya **true** karena property ini memang tersedia dan bagian dari class `mysqli`.

Fungsi `method_exists()`

Fungsi `method_exists()` dipakai untuk memeriksa apakah sebuah method tersedia atau tidak di dalam sebuah class. Fungsi ini butuh 2 buah argument berbentuk string. Argument pertama berupa nama class, dan argument kedua berupa nama method yang ingin dicari.

Berikut contoh penggunaannya:

```
1 echo "<b>method_exists('Laptop','getMerek')</b>: ";
2 var_dump(method_exists('Laptop','getMerek'));
3 echo "<br>";
4
5 echo "<b>method_exists('Laptop','getHarga')</b>: ";
6 var_dump(method_exists('Laptop','getHarga'));
7 echo "<br>";
8
9 echo "<b>method_exists('PDO','query')</b>: ";
10 var_dump(method_exists('PDO','query'));
11 echo "<br>";
```

Hasil kode program:

```
method_exists('Laptop','getMerek'): bool(true)
method_exists('Laptop','getHarga'): bool(false)
method_exists('PDO','query'): bool(true)
```

Di baris 2 saya menjalankan fungsi `method_exists('Laptop','getMerek')` untuk memeriksa apakah di dalam class `Laptop` terdapat method `getMerek`. Hasilnya **true** karena ini ada di kode program.

Di baris 6 saya menjalankan fungsi `method_exists('Laptop', 'getHarga')`. Ini dipakai untuk memeriksa apakah di dalam class Laptop terdapat method getHarga. Hasilnya **false** karena method ini memang tidak tersedia

Di baris 10 terdapat pemanggilan fungsi `method_exists('PDO', 'query')`. Fungsi ini berarti saya ingin memeriksa apakah di dalam class PDO terdapat method yang bernama query. Hasilnya **true** karena method ini tersedia dan bagian dari class PDO.

4.8. Function `class_alias()`

Fungsi `class_alias()` dipakai untuk membuat nama alias atau nama lain dari sebuah class. Fungsi ini butuh 2 buah argument bertipe string, dimana argument pertama adalah nama class asal, dan argument kedua berupa nama class alias.

Kedua class yang dihasilkan identik satu sama lain. Berikut contohnya:

10.class_alias.php

```
1 <?php
2 class Produk { }
3
4 class_alias('Produk', 'Barang');
5
6 $produk01 = new Produk;
7 $barang01 = new Barang;
8
9 var_dump($produk01 == $barang01);
10 echo "<br>";
11
12 var_dump($produk01 instanceof Produk);
13 echo "<br>";
14 var_dump($produk01 instanceof Barang);
15 echo "<br>";
16 var_dump($barang01 instanceof Produk);
17 echo "<br>";
18 var_dump($barang01 instanceof Barang);
```

Hasil kode program:

```
bool(true)
bool(true)
bool(true)
bool(true)
bool(true)
```

Di awal kode program saya membuat class Produk. Kemudian di baris 4 terdapat pemanggilan fungsi `class_alias('Produk', 'Barang')`. Artinya, saya ingin membuat Barang sebagai nama alias dari class Produk. Hasilnya, saya bisa menginisialisasi class Barang ke dalam `$barang01` di baris 7.

Object yang dibuat dari class alias dianggap sama satu sama lain. Operasi perbandingan object "sama dengan" di baris 9 menghasilkan nilai **true**. Begitu juga hasil pemeriksaan dengan keyword `instanceof` di baris 12 – 18 semuanya menghasilkan nilai **true**.

4.9. Function `get_called_class()`

Fungsi `get_called_class()` dipakai untuk mencari tahu class apa yang sedang memanggil sebuah method. Fungsi ini berguna untuk kasus **static late binding**. Berikut contoh penggunaannya:

11.get_called_class.php

```
1 <?php
2 class Produk {
3     public static function getInfo(){
4         return get_called_class();
5     }
6 }
7
8 class MesinCuci extends Produk {
9 }
10
11 echo Produk::getInfo();
12 echo "<br>";
13 echo MesinCuci::getInfo();
```

Hasil kode program:

Produk
MesinCuci

Kali ini saya membuat class `Produk` dengan 1 buah method static `getInfo()`. Di dalam method ini terdapat pemanggilan fungsi `get_called_class()`. Kemudian terdapat juga class `MesinCuci` yang diturunkan dari class `Produk`.

Ketika method `getInfo()` dipanggil dari class yang berbeda, fungsi `get_called_class()` akan menampilkan nama class yang sedang menjalankan method tersebut.

Dalam bab ini kita membahas berbagai fungsi bawaan PHP yang berhubungan dengan class dan object. Mayoritas dari fungsi-fungsi ini dipakai untuk memeriksa apakah sebuah method atau class tersedia dan bisa dipanggil. Pengetahuan ini akan bermanfaat terutama jika kita menggunakan class yang ditulis oleh programmer lain, atau ketika menggunakan library external.

Berikutnya, kita akan masuk ke materi tentang **namespace**.

5. Namespace

5.1. Pengertian Namespace

Namespace adalah fitur PHP untuk membuat semacam "folder virtual". Tujuannya untuk menghindari konflik jika terdapat nama class yang sama di 1 file. Situasi seperti ini bisa terjadi jika kita mengerjakan project besar yang dibuat oleh beberapa orang (dalam tim), atau menggunakan library PHP yang dibuat programmer lain.

Misalkan saya sedang membuat project website e-commerce. Untuk mengatur inventori barang saya menggunakan class `Produk`. Kemudian anggota tim lain membuat kode program pencari ongkos kirim yang sangat mungkin dia juga menggunakan class `Produk` dalam perancangan fitur tersebut.

Hasilnya terjadi error ketika kedua program digabung, karena dalam PHP tidak boleh terdapat 2 class dengan nama yang sama pada satu file.

Untuk menghindari masalah ini, saya bisa membuat aturan bahwa semua class harus diawali dengan nama programmer yang membuatnya. Jadi untuk class `Produk` nantinya terbagi menjadi class `AndreProduk` dan `RudiProduk`. Jika ada programmer lain yang ingin menggunakan class `Produk`, maka bisa memilih nama `BudiProduk`.

Aturan seperti ini bisa menjadi solusi untuk menghindari konflik. Namun nama class menjadi kurang rapi, dan itu juga baru untuk 1 class saja, belum termasuk nama class lain. Mari kita bahas dengan contoh kode program.

Sepanjang pembahasan materi **namespace** ini saya akan memakai 3 buah file: `file01.php`, `file02.php` dan `file03.php`. Ketiganya berada di satu folder yang sama dan akan saling terhubung dengan perintah `include()`.

Berikut isi dari file `file02.php`:

```
01.include_conflict\file02.php  
1 <?php  
2 class Produk {  
3     public $merek = "Miyako";  
4 }
```

Namespace

File ini berisi pendefinisian class Produk dengan 1 property merek. Property merek ini diisi string "Miyako" dengan hak akses public.

Untuk file01.php kodennya adalah sebagai berikut:

```
01.include_conflict\file01.php

1 <?php
2 include("file02.php");
3
4 $produk01 = new Produk();
5 echo $produk01->merek; // Miyako
```

Di baris 2 saya menggunakan fungsi `include("file02.php")` untuk meng-import class Produk dari file02.php. Dengan demikian di dalam file01.php ini saya bisa membuat object dari class Produk yang didefinisikan pada file file02.php. Hasilnya, property merek menampilkan string "Miyako".

Sampai di sini tidak ada masalah. Tapi ternyata terdapat file file03.php dengan kode program berikut:

```
01.include_conflict\file03.php

1 <?php
2 class Produk {
3     public $merek = "Maspion";
4 }
```

File ini juga mendefinisikan class Produk, namun dengan property merek yang berisi string "Maspion".

Artinya, file02.php dan file03.php sama-sama terdapat class Produk. Ini tidak menjadi masalah selama kedua file tersebut saling terpisah. Tapi bagaimana jika di dalam file01.php saya meng-import kedua file ini?

```
01.include_conflict\file01.php

1 <?php
2 include("file02.php");
3 include("file03.php");
4
5 $produk01 = new Produk();
6 echo $produk01->merek;
```

Hasil kode program:

```
Fatal error: Cannot declare class Produk, because the name is already in use
```

Hasilnya tampil pesan error karena terdapat 2 kali deklarasi class Produk. Masalah seperti inilah yang bisa diselesaikan dengan **namespace**.

5.2. Cara Penulisan Namespace

Namespace ditulis menggunakan keyword **namespace** dan diikuti dengan nama namespace-nya seperti contoh berikut:

02.penulisan_namespace\file02.php

```

1 <?php
2 namespace Miyako;
3
4 class Produk {
5     public $merek = "Miyako";
6 }
```

Penulisan namespace ada di baris ke-2. Perintah tersebut artinya, seluruh kode program yang ada dalam `file02.php` berada di dalam namespace **Miyako**.

Agar mudah dipahami, namespace ini bisa dianggap sebagai folder virtual. Perintah namespace **Miyako** artinya seluruh kode program di `file02.php` berada di dalam folder **Miyako**.

Kita bebas ingin menggunakan nama namespace apa saja sepanjang memenuhi syarat identifier, yakni tidak boleh diawali dengan angka dan tidak boleh terdapat spasi. Misalnya saya bisa membuat namespace `Andre`, namespace `rudi`, maupun namespace `unvocer08`.

Perintah namespace ini juga harus ditulis di baris paling awal. Tidak boleh ada kode PHP (maupun kode HTML) sebelum perintah namespace:

02.penulisan_namespace\file02.php

```

1 <?php
2 echo "Test";
3 namespace Miyako;
4
5 class Produk {
6     public $merek = "Miyako";
7 }
```

Hasil kode program:

Fatal error: Namespace declaration statement has to be the very first statement or after any declare call in the script

Error di atas tampil karena terdapat perintah `echo "Test";` di baris 2 sebelum perintah namespace. Namun terdapat beberapa pengecualian untuk hal ini, misalnya perintah `declare()` dan komentar diperbolehkan sebelum penulisan namespace:

02.penulisan_namespace\file02.php

```
1 <?php
```

Namespace

```
2 declare(strict_types=1);
3 //echo "Test";
4 namespace Miyako;
5
6 class Produk {
7     public $merek = "Miyako";
8 }
```

Baik, kita sudah menulis `namespace Miyako` di dalam `file02.php`. Jadi apa pengaruhnya?

Untuk melihat efek dari namespace ini, kita akan coba akses class `Produk` dari `file01.php`.

Berikut isi dari `file02.php` dan `file01.php`:

02.penulisan_namespace\file02.php

```
1 <?php
2 namespace Miyako;
3
4 class Produk {
5     public $merek = "Miyako";
6 }
```

02.penulisan_namespace\file01.php

```
1 <?php
2 include("file02.php");
3
4 $produk01 = new Produk();
5 echo $produk01->merek;
```

Isi dari `file01.php` sendiri tidak berubah dari contoh sebelumnya, yakni saya meng-import `file02.php` di baris 2, lalu mencoba membuat object dari class `Produk`. Bedanya, sekarang di dalam `file02.php` sudah menggunakan namespace `Miyako`.

Berikut hasil kode program:

```
Fatal error: Uncaught Error: Class 'Produk' not found
```

Betul, hasilnya adalah sebuah pesan error dimana `file01.php` tidak bisa menemukan deklarasi class `Produk`. Padahal di dalam `file02.php` terlihat jelas kode pembuatan class `Produk`.

Inilah efek dari penggunaan namespace `Miyako`. Sekarang, meskipun di dalam `file02.php` terdapat deklarasi class `Produk`, tapi itu tidak bisa dibaca oleh `file01.php`.

Untuk bisa mengakses class `Produk`, terdapat 2 cara.

Pertama, kita bisa menulis perintah `namespace Miyako` di awal `file01.php`:

02.penulisan_namespace\file01.php

```
1 <?php
2 namespace Miyako;
3 include("file02.php");
```

Namespace

```
4  
5 $produk01 = new Produk();  
6 echo $produk01->merek; // Miyako
```

Kembali di ingat bahwa namespace ibarat folder. Agar 2 buah file bisa saling berkomunikasi, keduanya harus berada di dalam folder yang sama. Begitu juga dengan kode program kita. Karena `file02.php` berada di dalam namespace `Miyako`, maka `file01.php` juga harus ikut berada di dalam namespace `Miyako`.

Cara kedua untuk mengakses class `Produk` di `file02.php` adalah dengan menulis nama namespace pada saat pemanggilan class:

02.penulisan_namespace\file01.php

```
1 <?php  
2 include("file02.php");  
3  
4 $produk01 = new Miyako\Produk();  
5 echo $produk01->merek; // Miyako
```

Perhatikan cara pembuatan object di baris 4, di sini saya menulis `new Miyako\Produk()`. Kembali, mirip seperti struktur folder, perintah `Miyako\Produk()` artinya saya masuk ke namespace `Miyako`, lalu akses class `Produk()`.

Inilah fungsi dasar dari namespace, yakni membuat sebuah class disimpan di dalam "folder khusus".

Sekarang mari kita gabung dengan `file03.php`. Untuk file ini saya memutuskan untuk menggunakan namespace `Maspion`:

02.penulisan_namespace\file03.php

```
1 <?php  
2 namespace Maspion;  
3  
4 class Produk {  
5     public $merek = "Maspion";  
6 }
```

Dengan penulisan seperti ini, `file03.php` berada di folder virtual dengan nama **Maspion**.

Karena `file02.php` dan `file03.php` sudah berada di namespace yang terpisah, maka keduanya sudah bisa kita akses dari `file01.php`:

02.penulisan_namespace\file01.php

```
1 <?php  
2 include("file02.php");  
3 include("file03.php");  
4  
5 $produk01 = new Miyako\Produk();
```

Namespace

```
6 echo $produk01->merek;           // Miyako
7
8 echo "<br>";
9
10 $produk02 = new Maspion\Produk();
11 echo $produk02->merek;           // Maspion
```

Perhatikan cara pengaksesan kedua class, yakni dengan menuliskan namespace masing-masing: `new Miyako\Produk()` dan `new Maspion\Produk()`.

Inilah fungsi dari namespace, dimana kita bisa menggabungkan 2 buah class dengan nama yang sama. Dengan catatan, kedua class tersebut berada di dalam namespace yang berbeda.

5.3. Cara Penulisan Sub-namespace

Sub-namespace adalah istilah untuk menyebut nama namespace yang "berjenjang". Ini bisa dipakai untuk project yang cukup besar dimana bisa saja namespace juga saling bentrok (terdapat 2 nama namespace yang sama).

Untuk membuat sub-namespace, pisahkan nama namespace dengan tanda back slash (" \"). Berikut contohnya:

03.sub_namespace\file02.php

```
1 <?php
2 namespace KipasAngin\Miyako;
3
4 class Produk {
5     public $merek = "Miyako";
6 }
```

03.sub_namespace\file03.php

```
1 <?php
2 namespace Elektronik\RiceCooker\Maspion;
3
4 class Produk {
5     public $merek = "Maspion";
6 }
```

Kembali sama seperti struktur folder, namespace `KipasAngin\Miyako` artinya kita menempatkan seluruh file di dalam folder `Miyako` yang berada di dalam folder `KipasAngin`.

Untuk mengakses setiap namespace dari `file01.php`, tulis secara lengkap alamat namespace-nya:

03.sub_namespace\file01.php

```
1 <?php
2 include("file02.php");
```

Namespace

```
3 include("file03.php");
4
5 $produk01 = new KipasAngin\Miyako\Produk();
6 echo $produk01->merek;           // Miyako
7
8 echo "<br>";
9
10 $produk01 = new Elektronik\RiceCooker\Maspion\Produk();
11 echo $produk01->merek;           // Maspion
```

PHP tidak membatasi "seberapa dalam" struktur namespace, namun kedalaman 1 - 3 sub-namespace sudah cukup untuk kebanyakan project.

5.4. Namespace Import dan Alias

Namespace **import** dan **alias** bisa dipakai untuk menyingkat penulisan sebuah class. Dalam contoh sebelumnya, menulis perintah `new Elektronik\RiceCooker\Maspion\Produk()` terasa sangat panjang dan tidak praktis. Apalagi jika dalam `file01.php` kita butuh beberapa kali membuat object dari class `Produk`.

Namespace **import** dan **alias** bisa menjadi solusi. Dengan *namespace import*, kita tidak harus menulis seluruh namespace untuk setiap pemanggilan class. Kemudian dengan *namespace alias* kita bisa memberikan nama lain untuk class atau namespace.

Berikut contoh penggunaan namespace **import** dan **alias**:

04.namespace_alias\file01.php

```
1 <?php
2 include("file02.php");
3 include("file03.php");
4
5 use KipasAngin\Miyako\Produk as MiyakoProduk;
6 use Elektronik\RiceCooker\Maspion\Produk as MaspionProduk;
7
8 $produk01 = new MiyakoProduk();
9 echo $produk01->merek;           // Miyako
10
11 echo "<br>";
12
13 $produk02 = new MaspionProduk();
14 echo $produk02->merek;           // Maspion
```

Penulisan namespace **import** dan **alias** ada di baris 5 dan 6. Keyword **use** dipakai untuk meng-import namespace, yang diikuti keyword **as** untuk membuat **alias**.

Di baris 5, saya membuat alias dari namespace `KipasAngin\Miyako\Produk` sebagai `MiyakoProduk`, dan di baris 6 saya membuat alias dari namespace `Elektronik\RiceCooker\Maspion\Produk` sebagai `MaspionProduk`.

Namespace

Dengan penulisan ini, sepanjang kode program di `file01.php`, class `MiyakoProduk` dan `MaspionProduk` bisa dipakai untuk menggantikan penulisan masing-masing namespace.

Alternatif lain, kita juga bisa mengimport sub-namespace saja, seperti contoh berikut:

`05.namespace_alias_sub\file01.php`

```
1 <?php
2 include("file02.php");
3 include("file03.php");
4
5 use KipasAngin\Miyako as Miyako;
6 use Elektronik\RiceCooker\Maspion as Maspion;
7
8 $produk01 = new Miyako\Produk();
9 echo $produk01->merek;           // Miyako
10
11 echo "<br>";
12
13 $produk02 = new Maspion\Produk();
14 echo $produk02->merek;           // Maspion
```

Di sini saya menyingkat penulisan namespace `KipasAngin\Miyako` menjadi `Miyako` serta namespace `Elektronik\RiceCooker\Maspion` menjadi `Maspion`. Namun karena yang dijadikan alias hanya sub-namespace saja, maka penulisan class tetap ditulis sebagai `Miyako\Produk()` dan `Maspion\Produk()`.

Jika perintah `use` ditulis tanpa pasangan `as` (hanya di import saja), maka sub-namespace terakhir dianggap sebagai nilai `as`. Maksudnya, kedua penulisan berikut dianggap sama:

```
use KipasAngin\Miyako as Miyako;
use KipasAngin\Miyako;
```

Begitu juga dengan kedua perintah berikut:

```
use Elektronik\RiceCooker;
use Elektronik\RiceCooker as RiceCooker;
```

Dengan kata lain, perintah `as` untuk membuat alias hanya diperlukan jika kita ingin memberi nama selain sub-namespace terakhir.

Untuk mengakses class `Produk` dari namespace `Elektronik\RiceCooker\Maspion`, saya juga bisa menulis kode berikut:

`06.namespace_alias_sub_tanpa_as\file01.php`

```
1 <?php
2 include("file03.php");
3
4 use Elektronik\RiceCooker\Maspion\Produk;
5
6 $produk02 = new Produk();
```

Namespace

```
7 echo $produk02->merek;
```

Baris ke-4 pada saat proses import namespace sama artinya dengan:

```
use Elektronik\RiceCooker\Maspion\Produk as Produk;
```

Sehingga di dalam file01.php, pemanggilan class Produk secara langsung akan memanggil namespace Elektronik\RiceCooker\Maspion\Produk.

5.5. Group use Declarations

Group use declarations merupakan fitur namespace yang baru ditambahkan pada PHP 7. Fitur ini dipakai untuk meng-import banyak namespace sekaligus, namun dengan syarat namespace yang akan di import memiliki struktur tertentu.

Sebagai contoh, saya akan tulis ulang file02.php dan file03.php sebagai berikut:

07.group_use\file02.php

```
1 <?php
2 namespace Produk\KipasAngin;
3
4 class MiyakoProduk {
5     public $merek = "Miyako";
6 }
```

07.group_use\file03.php

```
1 <?php
2 namespace Produk\KipasAngin;
3
4 class MaspionProduk {
5     public $merek = "Maspion";
6 }
```

Dengan struktur seperti ini, di file01.php saya bisa akses dengan kode berikut:

07.group_use\file01.php

```
1 <?php
2 include("file02.php");
3 include("file03.php");
4
5 use Produk\KipasAngin\MiyakoProduk;
6 use Produk\KipasAngin\MaspionProduk;
7
8 $produk01 = new MiyakoProduk();
9 echo $produk01->merek;      // Miyako
10
11 echo "<br>";
12
```

Namespace

```
13 $produk02 = new MaspionProduk();  
14 echo $produk02->merek; // Maspion
```

Tidak ada yang baru di sini. Kode program di baris 5 - 6 adalah perintah untuk mengimport class Produk\KipasAngin\MiyakoProduk dan Produk\KipasAngin\MaspionProduk.

Sebagai alternatif, penulisan namespace ini bisa ditulis memakai *group use declarations*:

```
use Produk\KipasAngin\{MaspionProduk,MiyakoProduk};
```

Penulisan seperti ini hanya bisa dibuat jika struktur namespace diawali dengan sub-namespace yang sama. Dalam contoh kita, kedua namespace diawali dengan Produk\KipasAngin\, yang berbeda hanya di bagian akhir (nama class).

Cara ini bisa dipakai untuk mempersingkat penulisan namespace, terutama jika kita harus mengimport banyak namespace sekaligus. Misalnya namespace berikut:

```
use Produk\Elektronik\Televisi;  
use Produk\Elektronik\MesinCuci;  
use Produk\Elektronik\Laptop;  
use Produk\Elektronik\Komputer as PC;
```

Bisa ditulis ulang menjadi:

```
use Produk\Elektronik\{Televisi, MesinCuci, Laptop, Komputer as PC};
```

5.6. Global dan Relative Namespace

Dari pembahasan kita sebelumnya, bisa terlihat bahwa namespace ini dipakai untuk membuat semacam "container" atau "encapsulation". Setiap kode program akan terpisah satu sama lain selama menggunakan namespace yang berbeda.

Global namespace atau disebut juga sebagai **global space** adalah ruang kode program tanpa namespace. Jika kita membuat kode program tanpa menulis namespace, maka kode itu sebenarnya berada di dalam *global namespace*. Dengan demikian, semua contoh kode program PHP yang kita bahas sejak awal buku (selain bab namespace ini) berada di dalam global namespace.

Relative namespace adalah ruang kode program yang menggunakan namespace. Misalnya file02.php yang menggunakan namespace Miyako, maka kode tersebut berada di dalam *relative namespace*, yakni Miyako.

Yang akan kita bahas sesaat lagi adalah, kode program yang berada di dalam relative namespace, **tidak bisa** secara langsung mengakses kode yang ada di dalam global namespace.

Pertama, perhatikan kode program berikut:

Namespace

08.namespace_global_relative\file02.php

```
1 <?php
2 namespace Miyako;
3
4 class Produk {
5     public $merek = "Miyako";
6 }
7
8 $produk01 = new Produk();
9 echo $produk01->merek; // Miyako
```

Kode program ini menggunakan `namespace Miyako` sehingga berada di dalam *relative namespace*. Mari kita coba akses class bawaan PHP, misalnya `stdClass`:

08.namespace_global_relative\file02.php

```
1 <?php
2 namespace Miyako;
3
4 class Produk {
5     public $merek = "Miyako";
6 }
7
8 $produk01 = new Produk();
9 echo $produk01->merek; // Miyako
10
11 $produk02 = new stdClass();
```

Di baris 11 saya mengisi variabel `$produk02` dari hasil instansiasi class `stdClass()`. Bagaimana hasilnya?

Fatal error: Uncaught Error: Class 'Miyako\stdClass' not found

Error ini terjadi karena selama kita berada di dalam *relative namespace*, semua pemanggilan class secara tidak langsung akan merujuk ke nama namespace-nya. Perintah `new stdClass()` akan dijalankan oleh PHP menjadi `new Miyako\stdClass()`. Inilah yang menyebabkan pesan error karena `stdClass` tidak ada di *relative namespace Miyako*, tapi berada di *global namespace*.

Jadi bagaimana cara mengakses global namespace? Caranya, tambahkan tanda backslash (" \ ") di awal penulisan class. Dengan demikian, untuk membuat instansiasi `stdClass()` di dalam namespace `Miyako`, perintahnya adalah sebagai berikut:

```
$produk02 = new \stdClass();
```

Penulisan seperti ini berlaku jika kita ingin mengakses berbagai class bawaan PHP karena semuanya ada di *global namespace*. Sama juga halnya jika kita meng-include file lain yang ada di dalam *global namespace*.

Untuk contoh praktik, `file03.php` saya modif menjadi sebagai berikut:

Namespace

09.namespace_global_relative_2\file03.php

```
1 <?php
2 class Produk {
3     public $merek = "Maspion";
4 }
```

File ini tidak lagi menggunakan namespace, sehingga class `Produk` dengan property `$merek = "Maspion"` berada di dalam *global namespace*.

Kemudian `file03.php` ini saya include ke dalam `file02.php`:

09.namespace_global_relative_2\file02.php

```
1 <?php
2 namespace Miyako;
3
4 include("file03.php");
5
6 class Produk {
7     public $merek = "Miyako";
8 }
9
10 $produk01 = new Produk();
11 echo $produk01->merek;
```

File ini diawali dengan namespace `Miyako`, artinya kita berada di dalam *relative namespace*, yakni `Miyako`. Kemudian di baris 4 terdapat perintah `include("file03.php")` sehingga class `Produk` di `file03.php` sudah bisa diakses dari `file02.php`.

Pertanyaannya, seperti apa hasil kode program di baris 11? Apakah tampil `Miyako`? `Maspion`? atau malah error karena terdapat 2 kali pendefinisian class `Produk`?

Berikut hasilnya:

Miyako

Karena pendefinisian *relative namespace* `Miyako` di baris 2, maka seluruh kode program yang dipakai untuk memanggil class secara tidak langsung akan ditambahkan namespace ini.

Perintah `$produk01 = new Produk()` di baris 10 secara tidak langsung sama dengan `$produk01 = new Miyako\Produk()`. Inilah yang menyebabkan perintah `echo $produk01->merek` menampilkan string `Miyako`.

Sekarang, bagaimana jika seperti ini?

09.namespace_global_relative_2\file02.php

```
1 <?php
2 namespace Miyako;
3
4 include("file03.php");
```

Namespace

```
5
6 class Produk {
7     public $merek = "Miyako";
8 }
9
10 $produk01 = new \Produk();
11 echo $produk01->merek;
```

Hasil kode program:

Maspion

Perhatikan di baris 10, sekarang perintahnya adalah `$produk01 = new \Produk();`. Bisakah anda menjelaskan kenapa tampil **Maspion**?

Yup, karena tambahan tanda backslash, PHP akan mencari class `Produk` di global namespace. dimana class `Produk` yang dimaksud berasal dari `file03.php`.

Lanjut, bagaimana dengan kode program berikut?

09.namespace_global_relative_2\file02.php

```
1 <?php
2 namespace Miyako;
3
4 include("file03.php");
5
6 class Produk {
7     public $merek = "Miyako";
8 }
9
10 $produk01 = new \Miyako\Produk();
11 echo $produk01->merek;
```

Di baris 10 sekarang penulisannya menjadi `new \Miyako\Produk()`. Meskipun terlihat seperti pengaksesan global namespace, tapi perintah ini diikuti dengan nama namespace `Miyako`, sehingga ini dipakai untuk mengakses class `Produk` yang ada di dalam namespace `Miyako`.

Cara pengaksesan class di dalam namespace ini mirip seperti konsep **absolute path** dan **relative path** di dalam struktur folder. Sebagai contoh, untuk menginput `file03.php` ke dalam `file02.php` cukup ditulis:

```
include("file03.php");
```

Ini adalah cara penulisan *relatif path*, dimana saya berasumsi `file03.php` dan `file02.php` berada di dalam 1 folder.

Saya juga bisa menggunakan **absolute path**, yakni dengan menulis:

```
include("C:/xampp/htdocs/belajar_oop_php/bab_05/09.namespace_global_relative_2/
file03.php");
```

Ini adalah penulisan sebuah file lengkap dengan seluruh struktur folder.

5.7. Unqualified, Qualified dan Fully Qualified Namespace

Berdasarkan cara penulisan, terdapat 3 jenis alamat di dalam namespace, yakni:

- Unqualified name
- Qualified name
- Fully qualified name

Secara tidak langsung, semua penulisan ini sudah kita coba sebelumnya.

Unqualified name adalah sebutan untuk pengaksesan class tanpa menulis nama namespace.

Misalnya di dalam `file02.php` yang menggunakan namespace `Miyako`, terdapat perintah berikut:

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

Penulisan "`Produk()`" adalah *unqualified name*. Kita tidak mendapat informasi apakah class `Produk()` ini berada di dalam sebuah namespace atau tidak. Jika ternyata kode tersebut berada di dalam namespace `Miyako`, maka perintah yang akan diproses PHP adalah `new Miyako\Produk()`.

Qualified name adalah cara pengaksesan class dengan menulis nama namespace, tapi tetap relatif kepada file saat ini. Misalnya di dalam `file01.php` yang menggunakan namespace `Elektronik`, terdapat perintah berikut:

```
$produk01 = new Miyako\Produk();
```

Artinya, saya ingin mengakses class `Produk()` yang berada di dalam namespace `Miyako`, relatif kepada file saat ini. Karena file saat ini juga menggunakan namespace `Elektronik`, maka yang akan dijalankan oleh PHP adalah `new Elektronik\Miyako\Produk()`

Fully qualified name adalah cara pengaksesan class dengan menulis lengkap dari global namespace. Misalnya di dalam `file01.php` terdapat perintah berikut:

```
$produk01 = new \Miyako\Produk();
```

Penulisan "`\Miyako\Produk()`" adalah *fully qualified name*, dimana perintah ini tidak bergantung kepada namespace yang terdapat di dalam sebuah file.

5.8. Multiple Namespace

Di awal penjelasan tentang namespace, saya menulis bahwa namespace harus ditulis di baris paling atas, namun dalam halaman tersebut juga bisa terdapat lebih dari 1 namespace (*multiple namespace*). PHP menyediakan 2 cara untuk melakukannya.

Namespace

Berikut cara pertama untuk membuat **multiple namespace**:

```
10.multiple_namespace\file01.php

1 <?php
2 namespace Elektronik\Miyako;
3
4 class Produk {
5     public $merek = "Miyako";
6 }
7
8 $produk01 = new Produk();
9 echo $produk01->merek;           // Miyako
10
11 echo "<br>";
12
13 namespace Elektronik\Maspion;
14
15 class Produk {
16     public $merek = "Maspion";
17 }
18
19 $produk02 = new Produk();
20 echo $produk02->merek;           // Maspion
```

Kode program ini terdiri dari 2 kelompok namespace. Baris 2 – 11 adalah tempat untuk namespace Elektronik\Miyako, sedangkan di baris 13 – 20 adalah tempat untuk namespace Elektronik\Maspion.

Kedua namespace ini seolah-olah membagi dua file01.php. Jika deklarasi namespace ini dihapus, akan tampil pesan error karena terdapat 2 kali deklarasi class Produk.

Cara pembuatan *multiple namespace* kedua adalah dengan memakai tanda kurung kurawal:

```
11.multiple_namespace_2\file01.php

1 <?php
2 namespace Elektronik\Miyako {
3     class Produk {
4         public $merek = "Miyako";
5     }
6
7     $produk01 = new Produk();
8     echo $produk01->merek;           // Miyako
9
10    echo "<br>";
11 }
12
13 namespace Elektronik\Maspion {
14     class Produk {
15         public $merek = "Maspion";
16     }
17 }
```

Namespace

```
18 $produk02 = new Produk();
19 echo $produk02->merek;           // Maspion
20 }
```

Hasilnya terlihat lebih rapi karena kedua namespace sudah dikelompokkan dan saling terpisah.

Penulisan menggunakan tanda kurung kurawal ini juga memiliki kelebihan yang tidak bisa dilakukan dengan cara pertama, yakni kita bisa beralih ke *global namespace*. Berikut contohnya:

12.multiple_namespace_3\file01.php

```
1 <?php
2 namespace Elektronik\Miyako {
3     class Produk {
4         public $merek = "Miyako";
5     }
6
7     $produk01 = new Produk();
8     echo $produk01->merek;           // Miyako
9
10    echo "<br>";
11 }
12
13 namespace {
14     class Produk {
15         public $merek = "Cosmos";
16     }
17
18     $produk02 = new Produk();
19     echo $produk02->merek;           // Cosmos
20 }
```

Perhatikan penulisan namespace di baris 13. Itu adalah perintah untuk beralih ke *global namespace*. Artinya kode program di baris 14 – 20 berada di global namespace, atau bisa disebut berada di area yang "tidak menggunakan namespace".

5.9. Namespace untuk Function dan Konstanta

Sepanjang bab ini kita telah melihat efek penggunaan namespace untuk class. Namespace sebenarnya juga berlaku untuk **function** dan **konstanta**, tetapi tidak untuk **variabel**. Artinya kita bisa membuat function (fungsi) dan konstanta dengan nama yang sama dalam 1 file, selama berada di namespace yang berbeda.

Sebagai contoh praktik, berikut modifikasi dari file02.php:

13.namespace_function\file02.php

```
1 <?php
2 namespace Miyako;
```

Namespace

```
3
4  class Produk {
5      public $merek = "Miyako";
6  }
7
8  function hidupkan(){
9      return "Wuzz...";
10 }
11
12 const JENIS = "Kipas Angin";
13 $harga = 150000;
```

Di dalam file ini saya mendefinisikan class `Produk`, function `hidupkan()`, konstanta `Jenis`, dan variabel `$harga`. Semuanya berada di dalam namespace `Miyako`.

Untuk bisa mengakses kode ini dari `file01.php` kita harus menggunakan aturan namespace:

13.namespace_function\file01.php

```
1  <?php
2  include("file02.php");
3
4  $produk01 = new Miyako\Produk();
5  echo $produk01->merek;
6  echo "<br>";
7  echo Miyako\hidupkan();
8  echo "<br>";
9  echo Miyako\JENIS;
10 echo "<br>";
11 echo $harga;
```

Hasil kode program:

```
Miyako
Wuzz...
Kipas Angin
150000
```

Untuk mengakses class `Produk` (baris 4), function `hidupkan()` (baris 7) dan konstanta `JENIS` (baris 9), semuanya menggunakan *qualified name* namespace, yakni dengan tambahan `"Miyako\"`. Jika nama namespace tidak ditulis, akan tampil pesan error karena PHP tidak bisa menemukannya.

Namun untuk variabel `$harga` di baris 11, tidak perlu diawali dengan nama namespace `"Miyako\"`. Ini karena variabel memang tidak terdampak pada aturan namespace.

Meskipun diakses menggunakan namespace (sama seperti class), namun **function** mendapat perlakuan yang sedikit berbeda dibandingkan **class**, terutama untuk function bawaan PHP.

Sebagaimana yang pernah kita coba untuk class `stdClass`, class bawaan PHP harus diakses dari *global namespace*, tidak bisa diakses langsung dari dalam *relative namespace*. Namun

Namespace

untuk function, PHP otomatis akan mencari di global namespace jika function tersebut tidak ditemukan.

Berikut praktek dari penjelasan ini:

14.namespace_function_2\file01.php

```
1 <?php
2 namespace Elektronik;
3
4 $waktu = new \DateTime();
5 echo $waktu->format('d-m-Y H:i:s');
6
7 echo "<br>";
8
9 echo date('d-m-Y H:i:s');
```

Hasil kode program:

```
09-04-2022 07:20:11
09-04-2022 07:20:11
```

Kode program untuk `file01.php` saya modifikasi ulang, yang kali ini diawali dengan `namespace Elektronik` di baris 2. Dengan demikian semua kode program di `file01.php` berada di `namespace Elektronik`.

Di baris 4 saya mengakses sebuah class bawaan PHP, yakni `DateTime`. Karena kita berada di dalam *relative namespace* `Elektronik`, maka untuk mengakses class ini harus ditambah tanda backslash "\ ", menjadi `new \DateTime()`. Ini karena semua class bawaan PHP berada di dalam *global namespace*.

Class `DateTime` dipakai untuk memproses data waktu (nanti akan kita bahas dalam bab tersendiri). Dengan perintah ini, variabel `$waktu` akan berisi object hasil instansiasi class `DateTime`. Kemudian di baris 5 saya mengakses property `format` dari object `$waktu`. Hasilnya tampil tanggal dan waktu hari ini (waktu pada saat kode dijalankan).

Di baris 9, saya mengakses fungsi `date()` bawaan PHP. Di sini tidak perlu menambah tanda backslash "\ " untuk masuk ke global namespace, karena PHP otomatis akan mencari function di global namespace jika di relative namespace tidak ditemukan. Namun jika ditulis sebagai `echo \date('d-m-Y H:i:s')` pun juga tidak masalah.

Perilaku ini juga berlaku untuk konstanta, dimana jika tidak ditemukan dalam namespace saat ini, PHP akan mencarinya di global namespace, tanpa perlu menulis tanda "\ ".

5.10. Namespace Magic Constant

PHP menyediakan magic constant `__NAMESPACE__` yang bisa dipakai untuk menampilkan nama namespace pada saat konstanta tersebut dijalankan. Berikut contohnya:

Namespace

15.namespace_constant\file01.php

```
1 <?php
2 namespace Elektronik;
3
4 echo __NAMESPACE__;
5 echo "<br>";
6
7 namespace Produk\Elektronik\KipasAngin;
8
9 echo __NAMESPACE__;
```

Hasil kode program:

```
Elektronik
Produk\Elektronik\KipasAngin
```

Magic constant `__NAMESPACE__` biasa dipakai untuk proses debugging (pencarian kesalahan). Misalnya jika tampil pesan error bahwa sebuah class tidak ditemukan, maka bisa cek di namespace mana kode saat ini berada.

Exercise

Materi tentang magic constant `__NAMESPACE__` ini menutup pembahasan kita tentang namespace. Untuk menguji pemahaman, saya membuat sebuah latihan sederhana.

Kita masih menggunakan `file01.php`, `file02.php` dan `file03.php`. Ketiganya berada di folder yang sama.

Berikut isi dari `file02.php`:

```
31 <?php
32 namespace Elektronik\Komputer;
33
34 class Laptop {
35     private $merek;
36     public function __construct($merek){
37         $this->merek = $merek;
38     }
39
40     public function getInfo(){
41         return "Laptop ". $this->merek .", Harga: 599000";
42     }
43 }
```

Dan berikut isi dari `file03.php`:

```
1 <?php
2 namespace Elektronik;
3
4 class Televisi {
5     private $merek;
6     private $harga;
```

Namespace

```
7
8  public function __construct($merek, $harga){
9    $this->merek = $merek;
10   $this->harga = $harga;
11 }
12
13  public function getInfo(){
14    return "Televisi ".$this->merek.", Harga: ".$this->harga;
15  }
16
17 }
```

Silahkan anda pelajari sejenak maksud dari setiap file ini. Dalam kedua file terdapat namespace serta sebuah class yang terdiri dari property dan method.

Tantangannya adalah, import / include kedua file ke dalam `file01.php`, lalu buat kode program agar tampil hasil berikut:

```
Laptop Lenovo, Harga: 599000
Televisi Sony, Harga: 5000000
```

Perhatikan bahwa selain perintah `include`, kita perlu membuat struktur namespace karena `file02.php` dan `file03.php` berada dalam namespace masing-masing.

Terdapat beberapa cara untuk mengakses class Laptop dan class Televisi, yakni mengakses langsung setiap class dengan menulis namespace-nya, melalukan import namespace, atau bisa juga menggunakan multiple namespace di `file01.php`. Saya akan menampilkan ketiga cara ini.

Cara pertama, bisa dengan mengakses langsung setiap class:

```
16.namespace_exercise\file01.php
```

```
1 <?php
2 // Cara 1
3
4 include("file02.php");
5 include("file03.php");
6
7 $produk01 = new Elektronik\Komputer\Laptop("Lenovo");
8 echo $produk01->getInfo();
9
10 echo "<br>";
11
12 $produk02 = new Elektronik\Televisi("Sony",5000000);
13 echo $produk02->getInfo();
```

Setelah proses include di baris 4 dan 5, saya mengakses class Laptop dan class Televisi lengkap dengan namespace-nya (baris 7 dan 12).

Cara kedua, adalah dengan proses import namespace:

```
1 <?php
2 // Cara 2
3
4 include("file02.php");
5 include("file03.php");
6
7 use Elektronik\Komputer\Laptop;
8 use Elektronik\Televisi;
9
10 $produk01 = new Laptop("Lenovo");
11 echo $produk01->getInfo();
12
13 echo "<br>";
14
15 $produk02 = new Televisi("Sony",5000000);
16 echo $produk02->getInfo();
```

Di sini saya melakukan proses import namespace di baris 7 dan 8. Perintah tersebut sebenarnya sama dengan:

```
use Elektronik\Komputer\Laptop as Laptop;
use Elektronik\Televisi as Televisi;
```

Dengan demikian, sepanjang file01.php class Laptop dan Televisi bisa dipakai tanpa harus menulis lengkap namespace-nya.

Cara ketiga, adalah menggunakan *multiple namespace*:

```
1 <?php
2 // Cara 3
3
4 namespace Elektronik\Komputer{
5     include("file02.php");
6
7     $produk01 = new Laptop("Lenovo");
8     echo $produk01->getInfo();
9
10    echo "<br>";
11 }
12
13 namespace Elektronik{
14     include("file03.php");
15
16     $produk02 = new Televisi("Sony",5000000);
17     echo $produk02->getInfo();
18 }
```

Kali ini saya mengakses setiap class dengan cara "masuk" ke namespace masing-masing

file.

Dari ketiga cara ini anda bebas ingin memakai cara yang mana saja. Ini lebih ke situasi dan kondisi pada saat pembuatan kode program. Meskipun begitu, cara kedua yakni dengan proses import namespace lebih umum dipakai.

Dalam bab ini kita telah membahas tentang **namespace**, yakni solusi untuk mencegah error yang terjadi akibat penggunaan nama class yang sama. Selain itu **namespace** juga membuat penamaan class menjadi lebih elegant, kita bisa menggunakan nama class yang pendek tanpa perlu khawatir bentrok dengan nama class yang ditulis oleh programmer lain.

Berikutnya, kita akan masuk ke materi tentang **autoload**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

6. Autoloading

Materi yang akan kita bahas kali ini masih berhubungan dengan solusi untuk menangani banyak file. Untuk aplikasi atau website yang besar, proses import file dengan perintah `include` dan `require` menjadi sedikit merepotkan. **Autoloading** bisa menjadi solusi untuk mengatasinya.

6.1. Pengertian Autoloading

Autoloading adalah fitur dalam PHP untuk meng-import file secara otomatis. Fitur ini baru terpakai ketika membuat project besar yang butuh banyak import file, misalnya seperti kasus berikut:

```
1 <?php
2
3 require "file01.php";
4 require "file02.php";
5 require "file03.php";
6 require "file04.php";
7 require "file05.php";
8
9 // ... kode PHP di sini...
```

Di sini saya melakukan import untuk 5 file, dan ini juga baru di 1 kode program. Jika terdapat 5 kode program lain, maka proses import ini juga harus ditulis ulang.

Dengan memanfaatkan *autoload*, kita bisa membuat sebuah mekanisme proses import file secara otomatis.

Dalam PHP terdapat 4 perintah untuk meng-import sebuah file PHP ke file PHP lain, yakni `include`, `required`, `include_once` dan `required_once`.

Perbedaan antara `include` dan `required` adalah pada pesan kesalahan yang tampil ketika file gagal di import. `Include` akan menampilkan error level **warning** dan PHP berusaha untuk melanjutkan kode program. Sedangkan `required` akan menampilkan **fatal error** dan kode program langsung berhenti.

Perbedaan antara `include` dengan `include_once`, serta `required` dan `required_once` adalah `include_once` dan `required_once` akan memastikan hanya ada 1 kali proses import. Dalam hal performa, `include_once` dan `required_once` sedikit lebih lambat

dibandingkan `include` dan `required` karena butuh proses pemeriksaan.

Sebagai tambahan, keempat perintah import ini adalah "*language construct*", yakni bukan function bawaan PHP biasa. Sehingga ke-4nya bisa dipanggil dengan atau tanpa tanda kurung. Perintah `require("file01.php")` dan `require "file01.php"` sama-sama boleh dipakai.

Penjelasan lebih jauh tentang `include`, `required`, `include_once` dan `required_once` sudah pernah saya bahas di buku **PHP Uncover**. Jika anda ragu, boleh dibaca kembali untuk sekedar pengingat.

6.2. Cara Penggunaan Autoloading

Ada beberapa syarat untuk bisa memanfaatkan fitur autoloading:

1. Autoloading baru "ter-trigger" atau baru dijalankan ketika sebuah class yang belum tersedia, di panggil dalam kode program.
2. Nama file yang akan di-import harus sama dengan nama class (atau memiliki pola nama yang berhubungan dengan nama class).
3. Jika class tersebut menggunakan namespace, struktur folder harus mengikuti penulisan namespace.

Ketiga syarat ini akan kita bahas dengan berbagai contoh kode program.

Untuk membuat autoloading, kita memakai fungsi `spl_autoload_register()`. Nama fungsi ini memang sedikit mengintimidasi karena cukup panjang. **SPL** sendiri adalah singkatan dari *Standard PHP Library*, yakni berbagai fungsi bawaan PHP untuk menyelesaikan masalah-masalah umum (yang sedikit '*advanced*'). Cukup banyak fungsi yang tergabung ke dalam SPL, jika tertarik anda bisa mengunjungi: [Standard PHP Library \(SPL\)](#).

Fungsi `spl_autoload_register()` pada dasarnya dipakai untuk menampung nama class yang tidak terdefinisi.

Perhatikan kode program berikut:

```
01.autoload_basic\01.error_class.php

1 <?php
2
3 $produk01 = new Mobil();
4 $produk02 = new SepedaMotor();
```

Hasil kode program:

```
Fatal error: Uncaught Error: Class 'Mobil' not found
```

Autoloading

Di baris 3 saya mencoba membuat object dari class `Mobil`. Namun karena class `Mobil` ini tidak terdefinisi, maka tampil error 'Class 'Mobil' not found'.

Di baris 4 saya membuat object dari class `SepedaMotor` yang juga tidak terdefinisi, namun tidak tampil pesan error karena kode program PHP sudah berhenti di baris 3 (disebabkan oleh **Fatal error**). Jika baris 3 di hapus atau di-set sebagai komentar, pesan error yang sama juga akan tampil:

01/autoload_basic\01.error_class.php

```
1 <?php
2
3 // $produk01 = new Mobil();
4 $produk02 = new SepedaMotor();
```

Hasil kode program:

Fatal error: Uncaught Error: Class 'SepedaMotor' not found

Ini adalah perilaku default dari kode PHP ketika menemukan class yang tidak terdefinisi.

Sekarang, saya akan membuat fungsi `spl_autoload_register()` untuk "menangkap" nama class yang tidak terdefinisi ini, berikut kode programnya:

01/autoload_basic\02.error_class.php

```
1 <?php
2 spl_autoload_register('cekClass');
3
4 function cekClass($foo) {
5     echo "Nama class yang dicari adalah: ".$foo;
6 }
7
8 $produk01 = new Mobil();
9 $produk02 = new SepedaMotor();
```

Fungsi `spl_autoload_register()` butuh argument berupa nama class. Dalam contoh ini di baris 1 saya mengisi nilai argument sebagai string 'cekClass'. Artinya, fungsi `spl_autoload_register()` akan mencari sebuah fungsi lain bernama `cekClass`.

Pendefinisian fungsi `cekClass()` ada di baris 4 – 6. Fungsi ini menerima 1 argument yang disimpan ke dalam variabel `$foo`. Isi dari fungsi `cekClass` sendiri hanya 1 baris, yakni perintah `echo "Nama class yang dicari adalah: ".$foo.`

Untuk kode program di baris 8 dan 9 masih sama seperti sebelumnya, yakni perintah untuk membuat object dari class `Mobil` dan `SepedaMotor`.

Berikut hasil dari kode program di atas:

```
Nama class yang dicari adalah: Mobil
Fatal error: Uncaught Error: Class 'Mobil' not found
```

Hasilnya tetap tampil error karena class Mobil tetap tidak ditemukan. Namun perhatikan hasil baris pertama, tampil string "Nama class yang dicari adalah: Mobil". String ini berasal dari fungsi `cekClass()` yang dijalankan oleh `spl_autoload_register()`.

Alur kerja kode di atas adalah sebagai berikut:

1. PHP menjalankan kode program dari baris paling awal secara berurutan ke bawah.
Baris 2 – 6 hanya berisi pendefinisian fungsi `spl_autoload_register()` dan `cekClass()`. Di sini tidak terdapat pemanggilan fungsi tersebut, sehingga kode hanya di periksa lalu di simpan di memory.
2. PHP sampai di baris 8 dan mencoba membuat object dari class `Mobil`. Tapi ternyata tidak ditemukan definisi class `Mobil`. Pada kondisi inilah PHP secara otomatis akan memanggil fungsi `spl_autoload_register()` serta mengirim sebuah string tentang class yang tidak ditemukan tadi, yakni string '`Mobil`'.
3. Di dalam fungsi `spl_autoload_register()` proses penanganan kemudian diserahkan lagi kepada fungsi `cekClass()`.
4. Dalam fungsi `cekClass()`, informasi nama class (yang dikirim PHP sebelumnya) ditampung ke dalam parameter `$foo`. Dengan demikian, parameter `$foo` sekarang berisi string '`Mobil`' yang kemudian diproses ke dalam perintah `echo`. Hasilnya, tampil string:
`Nama class yang dicari adalah: Mobil`.
5. Sampai di sini PHP tetap tidak menemukan pendefinisian class `Mobil` sehingga akhirnya menampilkan kembali pesan error `Class 'Mobil' not found`.

Kesimpulan dari alur di atas adalah, fungsi `spl_autoload_register()` akan dipanggil (atau ter-trigger) pada saat PHP menemukan kode program yang class-nya tidak terdefinisi.

Pada saat ini terjadi, fungsi `spl_autoload_register()` akan mendapatkan informasi mengenai nama class yang tidak terdefinisi tersebut. Di dalam class `spl_autoload_register()` inilah kita akan membuat proses **include** atau **required** file PHP yang dibutuhkan (tempat dimana class tadi di definisikan).

Sebagai info tambahan, jika saya menghapus baris 8 atau menjadikannya sebagai komentar, hasil yang tampil adalah sebagai berikut:

01.autoload_basic\02.error_class.php

```

1 <?php
2 spl_autoload_register('cekClass');
3
4 function cekClass($foo) {
5     echo "Nama class yang dicari adalah: ".$foo;
6 }
7
8 //$/produk01 = new Mobil();
9 $produk02 = new SepedaMotor();

```

Hasil kode program:

```
Nama class yang dicari adalah: SepedaMotor  
Fatal error: Uncaught Error: Class 'SepedaMotor' not found
```

Karena yang dicari adalah class SepedaMotor, maka PHP akan mengirim string 'SepedaMotor' ke pada fungsi `spl_autoload_register()`, yang selanjutnya ditampilkan dengan perintah echo.

Berikutnya, saya akan rangkai informasi nama class yang di dapat:

01.autoload_basic\03.require_autoload.php

```
1 <?php  
2 spl_autoload_register('cekClass');  
3  
4 function cekClass($foo) {  
5     echo "require '$foo.php';";  
6 }  
7  
8 $produk01 = new Mobil();  
9 $produk02 = new SepedaMotor();
```

Hasil kode program:

```
require 'Mobil.php';  
Fatal error: Uncaught Error: Class 'Mobil' not found
```

Perubahan dari kode program sebelumnya ada di baris 5, dari sebelumnya echo "Nama class yang dicari adalah: ".\$foo menjadi echo "require '\$foo.php'; ". Hasil yang tampil adalah **require 'Mobil.php'**; Sampai di sini semoga anda bisa menebak "ide" dari proses autoloading.

Apabila saya mengubah perintah di baris 5 dari echo "require '\$foo.php';" menjadi require "\$foo.php", maka setiap kali kode PHP menemukan class yang tidak terdefinisi, fungsi `cekClass()` akan segera menjalankan perintah require "\$foo.php". Misalkan class yang tidak terdefinisi adalah class Mobil, maka require "\$foo.php" akan diproses menjadi require "Mobil.php", yakni proses meng-import file Mobil.php ke dalam kode program saat ini.

Agar proses ini bisa berhasil, tentu saja kita harus mempersiapkan terlebih dahulu file Mobil.php dan juga SepedaMotor.php. Oleh karena itu saya akan membuat file mobil.php dan file sepedamotor.php, dimana masing-masing file berisi kode program berikut:

mobil.php

```
1 <?php  
2 class Mobil {  
3     private $merek;  
4  
5     public function __construct($merek){  
6         $this->merek = $merek;
```

Autoloading

```
7     }
8     public function getInfo(){
9         return "Mobil ".$this->merek;
10    }
11 }
```

sepedamotor.php

```
1 <?php
2 class SepedaMotor {
3     private $merek;
4
5     public function __construct($merek){
6         $this->merek = $merek;
7     }
8     public function getInfo(){
9         return "Sepeda Motor ".$this->merek;
10    }
11 }
```

Silahkan anda pelajari sejenak isi dari kedua file ini yang sebenarnya cukup sederhana.

Dalam file `mobil.php` saya mendefinisikan class `Mobil` dengan sebuah property `merek`, sebuah constructor, serta sebuah fungsi `getInfo()`. Constructor dipakai untuk mengisi property `merek`, dan fungsi `getInfo()` dipakai untuk menampilkan string `"Mobil ".$this->merek`. Hal yang sama juga di buat untuk file `sepedamotor.php`.

Kembali ke kode kita sebelumnya, jika proses import dilakukan secara manual, bisa menggunakan kode program berikut:

01.autoload_basic\04.tanpa_autoload.php

```
1 <?php
2 require 'mobil.php';
3 require 'sepedamotor.php';
4
5 $produk01 = new Mobil("Toyota");
6 echo $produk01->getInfo();
7
8 echo "<br>";
9
10 $produk02 = new SepedaMotor("Yamaha");
11 echo $produk02->getInfo();
```

Hasil kode program:

```
Mobil Toyota
Sepeda Motor Yamaha
```

Agar file di atas bisa berjalan, file `mobil.php` dan file `sepedamotor.php` harus disimpan dalam folder yang sama dengan file saat ini.

Tujuan kita menggunakan *autoloading* dengan fungsi `spl_autoload_register()` adalah agar proses import di baris 2 – 3 terjadi secara otomatis. Berikut kode programnya:

01.autoload_basic\05autoload_success.php

```
1 <?php
2 spl_autoload_register('cekClass');
3
4 function cekClass($foo) {
5     require "$foo.php";
6 }
7
8 $produk01 = new Mobil("Toyota");
9 echo $produk01->getInfo();
10
11 echo "<br>";
12
13 $produk02 = new SepedaMotor("Yamaha");
14 echo $produk02->getInfo();
```

Hasil kode program:

```
Mobil Toyota
Sepeda Motor Yamaha
```

Jika sudah memahami penjelasan sebelumnya, saya yakin anda bisa paham alur dari proses *autoloading* ini.

Begitu PHP memproses baris 8, ia tidak bisa menemukan kode yang mendefinisikan class `Mobil`, karena itu fungsi `spl_autoload_register()` akan dipanggil. Seolah-olah PHP mengatakan: "Hei fungsi `spl_autoload_register()`, saya menemukan class `Mobil` yang tidak terdefinisi, apakah kamu bisa memprosesnya?"

Fungsi `spl_autoload_register()` menyanggupi hal tersebut dan menjalankan fungsi `cekClass()`, tidak lupa menyertakan bahwa class yang dicari adalah class yang bernama `Mobil`.

Dalam fungsi `cekClass()`, nama class `Mobil` ini diterima oleh string `'$foo'`. Kemudian di baris 5 terdapat perintah `require "$foo.php"`, yang akan diproses oleh PHP menjadi `require "Mobil.php"`.

PHP kemudian memproses perintah `require "Mobil.php"` ini dan mencari apakah ada file `Mobil.php` di dalam folder saat ini. Ternyata ada, kemudian apakah di dalam file `Mobil.php` terdapat pendefinisian class `Mobil`? Juga ada! Maka kode program kembali ke baris 8 dan proses pembuatan class `Mobil` berhasil dijalankan.

PHP lanjut memproses baris berikutnya dan di baris 13 juga menemukan masalah yang sama, kali ini giliran class `SepedaMotor` yang tidak terdefinisi. Proses yang sama akan dijalankan kembali, dimana fungsi `spl_autoload_register()` secara otomatis akan mencari file `SepedaMotor.php`, kemudian meng-importnya.

Autoloading

Inilah alur kerja lengkap dari sebuah proses *autoloading* di dalam PHP. Memang tidak sepenuhnya otomatis karena kita harus membuat sendiri kode program untuk proses import. Agar lebih rapi, saya akan sedikit modifikasi kode program ini:

01.autoload_basic\05autoload_success_2.php

```
1 <?php
2 spl_autoload_register('cekClass');
3
4 function cekClass($className) {
5     require strtolower("$className.php");
6 }
7
8 $produk01 = new Mobil("Toyota");
9 echo $produk01->getInfo();
10
11 echo "<br>";
12
13 $produk02 = new SepedaMotor("Yamaha");
14 echo $produk02->getInfo();
```

Perubahannya tidak terlalu banyak, hanya mengganti nama parameter `$foo` menjadi `$className` agar lebih informatif. Karena parameter tersebut nantinya memang akan menampung nama class.

Di baris 5 saya juga menambah fungsi `strtolower("$className.php")`. Tujuannya untuk menghindari masalah dari penggunaan huruf besar dan kecil untuk nama file.

Dalam kode program kita disarankan membuat nama class menggunakan huruf besar di awal nama, seperti `Mobil` atau `SepedaMotor`. Namun untuk nama file, biasanya menggunakan huruf kecil semua, seperti `mobil.php` dan `sepedamotor.php`.

Untuk sistem operasi Windows, ini tidak masalah karena di Windows nama file bersifat *case insensitive*, dimana file `Mobil.php`, `mobil.php` dan `MOBIL.php` dianggap sama. Namun dalam sistem operasi Linux atau Mac OS, perbedaan huruf ini dianggap file yang berbeda. Oleh karena itu fungsi `strtolower()` dipakai untuk mengubah isi parameter `$className` menjadi huruf kecil semua.

Untuk uji coba, mari kita test dengan memanggil class lain:

01.autoload_basic\06autoload_test.php

```
1 <?php
2 spl_autoload_register('cekClass');
3
4 function cekClass($className) {
5     require strtolower("$className.php");
6 }
7
8 $produk01 = new Mobil("Toyota");
```

Autoloading

```
9 echo $produk01->getInfo();
10
11 echo "<br>";
12
13 $produk02 = new SepedaMotor("Yamaha");
14 echo $produk02->getInfo();
15
16 echo "<br>";
17
18 $produk03 = new Pesawat("Airbus");
19 echo $produk03->getInfo();
```

Hasil kode program:

```
Mobil Toyota
Sepeda Motor Yamaha
Warning: require(pesawat.php): failed to open stream: No such file or directory
Fatal error: require(): Failed opening required 'pesawat.php'
```

Di baris 18 saya mencoba membuat object dari class Pesawat. Hasilnya tampil pesan error bahwa file pesawat.php tidak ditemukan. Ini berarti proses autoloading sudah berjalan, dimana PHP secara otomatis akan mencari file pesawat.php ketika terdapat kode program yang memanggil class Pesawat.

Variasi lain dari penulisan *autoloading* adalah menambah sebuah kondisi if untuk memeriksa apakah file yang dicari ada atau tidak (sebelum dilakukan proses require). Ini bisa dipakai untuk menghindari pesan error seperti di atas. Berikut modifikasinya:

01.autoload_basic\07.autoload_check_file.php

```
1 <?php
2 spl_autoload_register('cekClass');
3
4 function cekClass($className) {
5     $fileName = strtolower("$className.php");
6
7     if (file_exists($fileName)) {
8         require $fileName;
9     } else {
10        die ("File $fileName tidak tersedia");
11    }
12 }
13
14 $produk01 = new Mobil("Toyota");
15 echo $produk01->getInfo();
16
17 echo "<br>";
18
19 $produk02 = new SepedaMotor("Yamaha");
20 echo $produk02->getInfo();
21
22 echo "<br>";
```

Autoloading

```
23  
24 $produk03 = new Pesawat("Airbus");  
25 echo $produk02->getInfo();
```

Hasil kode program:

```
Mobil Toyota  
Sepeda Motor Yamaha  
File pesawat.php tidak tersedia
```

Perubahan dari kode sebelumnya ada di isi function `cekClass()`, yakni baris 5 – 11. Di awal fungsi ini, saya membuat variabel bantu `$fileName` untuk menampung nama file.

Kemudian terdapat perintah pemeriksaan kondisi `if (file_exists($fileName))`. Fungsi `file_exists()` adalah fungsi bawaan PHP yang bisa dipakai untuk memeriksa apakah sebuah file ada atau tidak di dalam folder saat ini. Jika ada, hasilnya boolean `true`, atau `false` jika file tidak ditemukan.

Pada saat perintah `new Mobil("Toyota")` dan `new SepedaMotor("Yamaha")` di proses, hasilnya menjadi `true` karena file `mobil.php` dan `sepedamotor.php` memang tersedia. Namun ketika yang dijalankan adalah `new Pesawat("Airbus")`, hasilnya `false` karena file `pesawat.php` tidak ditemukan.

Jika fungsi `if (file_exists($fileName))` menghasilkan nilai `false`, maka proses `require` tidak perlu dijalankan.

Modifikasi lanjutan yang bisa kita buat adalah memakai *anonymous function* sebagai argument fungsi `spl_autoload_register()`.

Anonymous function sendiri adalah fitur yang relatif baru di PHP, yang disebut sebagai **closure**. Sama seperti anonymous class, anonymous function biasa dipakai untuk nilai argument fungsi. Berikut contoh perubahan untuk fungsi `spl_autoload_register()`:

01.autoload_basic\08.autoload_anonymous_function.php

```
1 <?php  
2 spl_autoload_register(function ($className) {  
3     $fileName = strtolower("$className.php");  
4  
5     if (file_exists($fileName)) {  
6         require $fileName;  
7     } else {  
8         die ("File $fileName tidak tersedia");  
9     }  
10});  
11  
12 $produk01 = new Mobil("Toyota");  
13 echo $produk01->getInfo();  
14  
15 echo "<br>";  
16
```

Autoloading

```
17 $produk02 = new SepedaMotor("Yamaha");
18 echo $produk02->getInfo();
19
20 echo "<br>";
21
22 $produk03 = new Pesawat("Airbus");
23 echo $produk03->getInfo();
```

Hasil kode program:

```
Mobil Toyota
Sepeda Motor Yamaha
File pesawat.php tidak tersedia
```

Proses pembuatan *anonymous function* hanya dengan memindahkan isi fungsi `cekClass()` ke dalam argument fungsi `spl_autoload_register()` di baris 2 – 10.

Anda bebas ingin menggunakan class terpisah seperti contoh sebelumnya, atau menggunakan *anonymous function* seperti ini.

Mengenal Function `__autoload()`

PHP sebenarnya memiliki fungsi autoloading lain yakni `__autoload()`. Cara kerja fungsi `__autoload()` sangat mirip seperti `spl_autoload_register()`, dan malah lebih sederhana.

Berikut contoh praktik dari fungsi `__autoload()`:

01.autoload_basic\09.autoload_function.php

```
1 <?php
2 function __autoload($className){
3     $fileName = strtolower("$className.php");
4
5     if (file_exists($fileName)) {
6         require $fileName;
7     } else {
8         die ("File $fileName tidak tersedia");
9     }
10 }
11
12 $produk01 = new Mobil("Toyota");
13 echo $produk01->getInfo();
14
15 echo "<br>";
16
17 $produk02 = new SepedaMotor("Yamaha");
18 echo $produk02->getInfo();
19
20 echo "<br>";
21
22 $produk03 = new Pesawat("Airbus");
23 echo $produk03->getInfo();
```

Fungsi `__autoload()` langsung menerima argument berupa nama class yang tidak terdefinisi. Kita tidak perlu memanggil class lain atau menggunakan *anonymous function* seperti pada fungsi `spl_autoload_register()`.

Namun fungsi `__autoload()` ini sudah berstatus [deprecated di PHP 7.2](#), sehingga sebaiknya tidak dipakai lagi. Kita disarankan beralih ke `spl_autoload_register()` untuk pembuatan autoloading.

Salah satu alasan kenapa fungsi `__autoload()` di anggap usang (*deprecated*) adalah karena fungsi ini hanya bisa dijalankan 1 kali saja. Jika kita memiliki 2 struktur file yang berbeda, fungsi `__autoload()` tidak bisa dipanggil 2 kali. Contoh kode program untuk kasus ini akan kita bahas sesaat lagi.

6.3. Multiple Autoloading

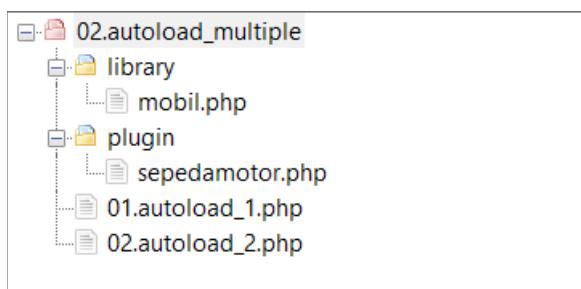
Dalam contoh autoloading sebelumnya, saya berasumsi file `mobil.php` dan `sepedamotor.php` berada dalam 1 folder. Namun sering kali file PHP disusun dalam folder-folder khusus agar lebih terorganisir.

Jika anda pernah mempelajari framework PHP seperti Code Igniter atau Laravel, di dalamnya terdiri dari banyak folder dan setiap folder berisi berbagai file PHP, misalnya di Code Igniter terdapat folder `libraries`, `helpers`, `core`, dsb.

Kita juga bisa membuat fitur autoloading untuk mencari file yang akan di load dari berbagai folder.

Sebagai contoh praktek, saya akan memindahkan file `mobil.php` ke dalam folder **library**. Kemudian file `sepedamotor.php` dipindahkan ke dalam folder **plugin**.

Berikut struktur folder yang akan kita pakai:



Gambar: struktur folder untuk file mobil.php dan sepedamotor.php

File `01autoload_1.php` dan `01autoload_2.php` adalah file kode program yang akan kita rancang. Di sinilah proses autoloading di tulis nantinya.

Jika berangkat dari file `01autoload_1.php`, maka class `Mobil` berada di `library/mobil.php` dan class `SepedaMotor` berada di `plugin/sepedamotor.php`.

Autoloading

Terdapat 2 cara yang bisa kita pakai untuk membuat proses autoloading ke dua file ini, yang pertama adalah dengan menjalankan fungsi `spl_autoload_register()` sebanyak 2 kali, yakni satu untuk setiap folder. Berikut kode programnya:

02.autoload_multiple\01autoload_1.php

```
1 <?php
2 spl_autoload_register('importLibraryClass');
3 spl_autoload_register('importPluginClass');
4
5 function importLibraryClass($className){
6     $filePath = strtolower("library/$className.php");
7
8     if (file_exists($filePath)) {
9         require $filePath;
10    }
11 }
12
13 function importPluginClass($className){
14     $filePath = strtolower("plugin/$className.php");
15
16     if (file_exists($filePath)) {
17         require $filePath;
18     }
19 }
20
21 $produk01 = new Mobil("Toyota");
22 echo $produk01->getInfo();
23
24 echo "<br>";
25
26 $produk02 = new SepedaMotor("Yamaha");
27 echo $produk02->getInfo();
```

Hasil kode program:

```
Mobil Toyota
Sepeda Motor Yamaha
```

Di baris 2 dan 3 saya membuat 2 kali fungsi `spl_autoload_register()`, masing-masing akan menjalankan fungsi `importLibraryClass()` dan `importPluginClass()`.

Isi dari kedua fungsi ini mirip seperti proses autoloading yang sudah kita pelajari sebelumnya. Yang berbeda adalah saya membuat variabel `$filePath` untuk merangkai nama folder, misalnya `strtolower("library/$className.php")` untuk mencari file di dalam folder **library**, serta `strtolower("plugin/$className.php")` untuk mencari file di folder **plugin**.

Kondisi `if (file_exists($filePath))` sangat berperan di sini. Ketika PHP menemukan perintah `new Mobil("Toyota")`, maka folder yang cari pertama kali adalah `library/mobil.php`, dan file tersebut memang tersedia.

Kemudian saat PHP menemukan perintah `new SepedaMotor("Yamaha")`, yang dicari adalah `library/sepedamotor.php`. Dan file ini tidak ditemukan, karena file sepedamotor.php ada di folder **plugin**, bukan **library**. Jika kita tidak membuat kondisi `if (file_exists($filePath))`, maka sampai di sini PHP akan menampilkan **fatal error** 'file not found' kemudian proses langsung berhenti.

Dengan tambahan kondisi `if (file_exists($filePath))`, maka PHP akan memeriksa terlebih dahulu apakah file yang dicari ada atau tidak. Jika ada, baru jalankan perintah `require $filePath`.

Proses yang dilakukan adalah sebagai berikut:

1. PHP menjalankan kode program dari baris paling awal secara berurutan ke bawah. Baris 2 – 19 hanya berisi pendefinisian fungsi sehingga kode hanya di periksa lalu di simpan di memory.
2. PHP sampai di baris 21 dan mencoba membuat object dari class `Mobil`. Tapi ternyata tidak ditemukan definisi class `Mobil`. Maka PHP secara otomatis akan memanggil fungsi `spl_autoload_register('importLibraryClass')`.
3. Dalam fungsi `importLibraryClass()`, variabel `$filePath` akan berisi `library/mobil.php`. Kemudian diperiksa `if (file_exists(library/mobil.php))?` **Ada** (true), maka jalankan perintah untuk men-import file tersebut, yakni `require library/mobil.php`.
4. Karena class sudah ditemukan, PHP lanjut ke baris 22 dan menjalankan perintah `echo $produk01->getInfo()`.
5. PHP sampai di baris 26 dan mencoba membuat object dari class `SepedaMotor`. Tapi ternyata class ini juga tidak ditemukan. Maka PHP secara otomatis akan memanggil fungsi `spl_autoload_register('importLibraryClass')` karena fungsi inilah yang berada paling awal.
6. Dalam fungsi `importLibraryClass()`, variabel `$filePath` akan berisi `library/sepedamotor.php`. Kemudian diperiksa `if (file_exists(library/sepedamotor.php))?` **Tidak ada** (false), karena di dalam folder library memang tidak terdapat file `sepedamotor.php`. Oleh karena itu perintah require tidak dijalankan.
7. PHP akan lanjut ke fungsi `spl_autoload_register('importPluginClass')`. Sekarang variabel `$filePath` akan berisi `plugin/sepedamotor.php`. Kemudian diperiksa `if (file_exists(plugin/sepedamotor.php))?` **Ada** (true), maka jalankan perintah untuk men-import file tersebut, yakni `require plugin/sepedamotor.php`.
8. Karena class sudah ditemukan, PHP lanjut ke baris 27 dan menjalankan perintah `echo $produk02->getInfo()`.

Inilah alur yang dijalankan jika terdapat 2 kali pemanggilan fungsi `spl_autoload_register()`.

Autoloading

Sekarang mari kita lihat alternatif cara penulisan kedua, yakni hanya menggunakan 1 fungsi `spl_autoload_register()`:

02/autoload_multiple\02autoload_2.php

```
1 <?php
2
3 spl_autoload_register(function ($className){
4
5     $filePath1 = strtolower("library/$className.php");
6     if (file_exists($filePath1)) {
7         require $filePath1;
8     }
9
10    $filePath2 = strtolower("plugin/$className.php");
11    if (file_exists($filePath2)) {
12        require $filePath2;
13    }
14
15 });
16
17
18 $produk01 = new Mobil("Toyota");
19 echo $produk01->getInfo();
20
21 echo "<br>";
22
23 $produk02 = new SepedaMotor("Yamaha");
24 echo $produk02->getInfo();
```

Di sini saya menggabungkan proses pemeriksaan folder library dan folder plugin di satu fungsi `spl_autoload_register()`. Konsep dasar yang dipakai kurang lebih sama seperti contoh sebelumnya.

6.4. Autoloading dengan Namespace

Jika class atau file yang akan di import menggunakan **namespace**, maka kita harus modifikasi alamat *path* yang dipakai oleh fungsi `spl_autoload_register()`. Sebagai contoh praktek, saya akan menambah namespace ke dalam class `Mobil` dan `SepedaMotor`:

mobil.php

```
1 <?php
2 namespace DuniaIlkom\Library;
3
4 class Mobil {
5     private $merek;
6
7     public function __construct($merek){
8         $this->merek = $merek;
9     }
10}
```

Autoloading

```
10  public function getInfo(){
11      return "Mobil ".$this->merek;
12  }
13 }
```

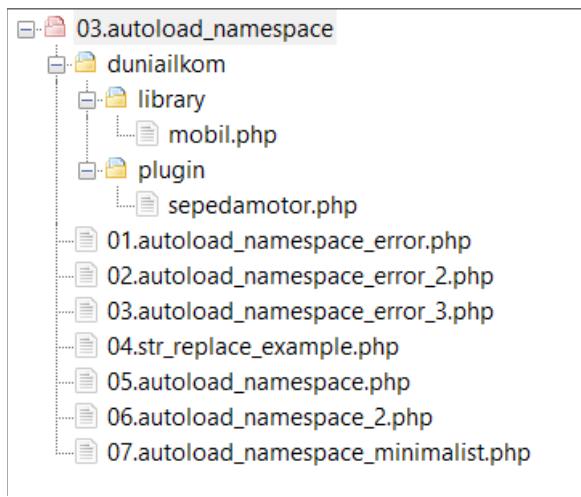
sepedamotor.php

```
1 <?php
2 namespace DuniaIlkom\Plugin;
3
4 class SepedaMotor {
5     private $merek;
6
7     public function __construct($merek){
8         $this->merek = $merek;
9     }
10    public function getInfo(){
11        return "Sepeda Motor ".$this->merek;
12    }
13 }
```

Sekarang, class `Mobil` berada di dalam namespace `DuniaIlkom\Library`, dan class `SepedaMotor` berada di dalam namespace `DuniaIlkom\Plugin`.

Agar proses autoloading dengan namespace bisa dilakukan, kita harus (atau setidaknya 'disarankan') untuk membuat struktur folder yang sama seperti namespace. Dengan demikian, saya akan membuat folder **duniaIlkom** yang di dalamnya terdapat folder **library** dan **plugin**.

Berikut struktur folder dari file `mobil.php` dan `sepedamotor.php`:



Gambar: struktur folder untuk file `mobil.php` dan `sepedamotor.php` di dalam folder `duniaIlkom`

Sekarang alamat path dari file `mobil.php` berada di `duniaIlkom/library/mobil.php` sedangkan untuk file `sepedamotor.php` alamat path-nya ada di `duniaIlkom/plugin/sepedamotor.php`. File dengan nomor 01,..., 02..., dst adalah file latihan yang akan kita bahas.

Mari kita coba akses file ini:

Autoloading

03/autoload_namespace\01autoload_namespace_error.php

```
1 <?php
2
3 spl_autoload_register(function ($className) {
4     $filePath = strtolower("duniailkom/library/$className.php");
5     echo $filePath;
6     require $filePath;
7 });
8
9 $produk01 = new Mobil("Toyota");
10 echo $produk01->getInfo();
```

Untuk fungsi `spl_autoload_register()`, saya langsung tulis dengan *anonymous function* sehingga tidak perlu memanggil fungsi terpisah.

Prinsip kerja proses import ini sama seperti sebelumnya, yakni di 'rangkai' terlebih dahulu ke dalam variabel `$filePath`, baru kemudian di import dengan perintah `require $filePath`.

Untuk variabel `$filePath` tidak lupa di tambah dengan urutan path yang baru menjadi `duniailkom/library/$className.php`.

Di baris 5 terdapat perintah `echo $filePath`. Ini saya sisipkan untuk memeriksa apakah hasil dari `$filePath` sudah sesuai atau belum.

Class `Mobil` kemudian di akses di baris 9, dan hasilnya adalah sebagai berikut:

```
duniailkom/library/mobil.php
Fatal error: Uncaught Error: Class 'Mobil' not found
```

Tampilan baris pertama merupakan hasil dari perintah `echo $filePath`. Bisa terlihat urutan nama file kita sudah sesuai, yakni `duniailkom/library/mobil.php`. Namun pada baris kedua tampil pesan error bahwa class `Mobil` tidak ditemukan. Bisakah anda menebak apa yang salah?

Saya bisa pastikan bahwa proses import file `mobil.php` sudah berhasil, karena kalau tidak error yang tampil adalah file `mobil.php` tidak bisa diakses.

Error di atas terjadi karena penggunaan **namespace**. Dalam file `mobil.php`, pendefinisian class `Mobil` berada di dalam namespace `duniailkom/library`, sehingga agar bisa mengakses class `Mobil` ini, kita juga harus menulis nama *namespace*-nya.

Dengan demikian, saya akan modifikasi ulang kode program di atas:

03/autoload_namespace\01autoload_namespace_error_2.php

```
1 <?php
2
3 spl_autoload_register(function ($className) {
4     $filePath = strtolower("duniailkom/library/" . $className . ".php");
5     echo $filePath;
6     require $filePath;
7 });
8
```

Autoloading

```
9 $produk01 = new DuniaIlkom\Library\Mobil("Toyota");
10 echo $produk01->getInfo();
```

Di baris 9 penulisan class Mobil sudah lengkap dengan namespace-nya, yakni new DuniaIlkom\Library\Mobil("Toyota"). Berikut hasil kode program:

```
duniaIlkom/library/duniaIlkom/library/mobil.php
Warning: require(duniaIlkom/library/duniaIlkom/library/mobil.php): failed to open stream: No such file or directory
```

Ternyata masih tidak bisa. Namun pesan error yang tampil adalah *failed to open stream*, yang artinya proses import file tidak berhasil di jalankan.

Penyebab dari error ini bisa dilihat pada hasil di baris pertama, yakni duniaIlkom/library/duniaIlkom/library/mobil.php. Teks ini berasal dari perintah echo \$filePath. Seharusnya, variabel \$filePath ini berisi alamat file duniaIlkom/library/mobil.php.

Jika anda masih ingat, ketika PHP menemukan sebuah class yang tidak terdefinisi, nama class tersebut akan dikirim ke argument fungsi `spl_autoload_register()`. Masalahnya, jika class tersebut dipanggil di dalam namespace, nama namespace itu juga ikut dikirim.

Dalam contoh kita, isi dari argument `$className` bukan lagi string 'Mobil', tapi 'DuniaIlkom\Library\Mobil', yakni hasil pemanggilan lengkap namespace beserta nama class. Berikut pembuktian dari hal ini:

03/autoload_namespace\03/autoload_namespace_error_3.php

```
1 <?php
2
3 spl_autoload_register(function ($className) {
4     echo $className;
5 });
6
7 $produk01 = new DuniaIlkom\Library\Mobil("Toyota");
8 echo $produk01->getInfo();
```

Hasil kode program:

```
DuniaIlkom\Library\Mobil
Fatal error: Uncaught Error: Class 'DuniaIlkom\Library\Mobil' not found
```

Kita bisa abaikan error di baris ke-2, namun perhatikan isi di baris pertama: `DuniaIlkom\Library\Mobil`. Hasil ini berasal dari perintah echo `$className` dalam fungsi `spl_autoload_register()`.

Sebenarnya hal ini bisa mempermudah pengaksesan file karena secara struktur, file `mobil.php` berada di folder `duniaIlkom/library`. Masalahnya ada di penggunaan tanda garis miring. Namespace ditulis menggunakan *backslash*, yakni tanda '\', sedangkan nama path ditulis menggunakan *forward slash*, yakni tanda '/'.

Maksudnya, jika saya langsung membuat seperti ini:

```
$filePath = strtolower("$className.php");
require $filePath;
```

Maka hasilnya menjadi:

```
require 'duniailkom\library\mobil.php';
```

Ini akan error, karena seharusnya yang ditulis adalah:

```
require 'duniailkom/library/mobil.php';
```

Sebagai solusi, kita bisa proses lebih lanjut menggunakan function `str_replace()` bawaan PHP. Function `str_replace()` berguna untuk mengubah karakter string, yang dalam kasus kita di pakai untuk mengubah tanda '\ ' menjadi '/':

03/autoload_namespace\04.str_replace_example.php

```
1 <?php
2 $filePath = 'duniailkom\library\mobil.php';
3 $hasil = str_replace('\\', '/', $filePath);
4
5 echo $hasil; // duniailkom/library/mobil.php
```

Fungsi `str_replace()` memerlukan 3 argument:

- Argumen pertama adalah untuk inputan karakter yang akan dicari, yakni karakter backslash '\'. Namun karena karakter backslash memiliki makna khusus di dalam string (sebagai *escape character*) maka harus ditulis 2 kali menjadi '\\'.
- Argument kedua yakni karakter pengganti, yakni karakter ' / '.
- Argument ketiga berupa string asal yang akan diubah.

Menggunakan fungsi `str_replace()`, kita sudah bisa mengkonversi string hasil namespace menjadi string path file. Berikut modifikasi dari kode program autoloading sebelumnya:

03/autoload_namespace\05.autoload_namespace.php

```
1 <?php
2 spl_autoload_register(function ($className) {
3     $filePath = strtolower("$className.php");
4     echo $filePath . "<br>";
5     $filePath = str_replace('\\', '/', $filePath);
6     echo $filePath . "<br>";
7     require $filePath;
8 });
9
10 $produk01 = new DuniaIlkom\Library\Mobil("Toyota");
11 echo $produk01->getInfo();
```

Hasil kode program:

```
duniailkom\library\mobil.php
duniailkom/library/mobil.php
Mobil Toyota
```

Sekarang pemanggilan class Mobil sudah berhasil!

Perhatikan juga hasil di baris 1 dan 2, ini berasal dari perintah echo \$filePath di dalam fungsi `spl_autoload_register()`. Saya menulis perintah echo agar kita bisa melihat isi dari \$filePath sebelum dan sesudah pemanggilan fungsi `str_replace()`.

Bagaimana jika namespace ini tidak langsung dipanggil dengan nama class tapi menggunakan proses *import* dan *alias*? Tidak masalah, karena PHP tetap mengirim nama namespace ke dalam fungsi `spl_autoload_register()`:

03/autoload_namespace\06autoload_namespace_2.php

```
1 <?php
2 spl_autoload_register(function ($className) {
3     $filePath = strtolower("$className.php");
4     $filePath = str_replace('\\', '/', $filePath);
5     require $filePath;
6 });
7
8 use DuniaIlkom\Library\Mobil as Mobil;
9 use DuniaIlkom\Plugin\SepedaMotor as SepedaMotor;
10
11 $produk01 = new Mobil("Toyota");
12 echo $produk01->getInfo();
13
14 echo "<br>";
15
16 $produk02 = new SepedaMotor("Yamaha");
17 echo $produk02->getInfo();
```

Hasil kode program:

```
Mobil Toyota
Sepeda Motor Yamaha
```

Dalam kode program ini isi fungsi `spl_autoload_register()` sama seperti sebelumnya. Di baris 8 dan 9 saya menggunakan proses *import namespace* dan *alias namespace*. Jika ditulis seperti ini, sebenarnya sama dengan:

```
use DuniaIlkom\Library\Mobil;
use DuniaIlkom\Plugin\SepedaMotor;
```

Kemudian saya membuat proses instansiasi untuk 2 buah class, yakni `Mobil` dan `SepedaMotor`. Keduanya langsung berjalan.

Kode program untuk pendefinisian class SepedaMotor berada di `duniailkom\plugin\sepedamotor.php`, namun proses autoloading secara otomatis sudah bisa mencari file ini. Alasannya karena struktur namespace saya buat sama dengan struktur fisik folder.

Fungsi `spl_autoload_register()` sebenarnya masih bisa dibuat lebih sederhana lagi, yakni dengan menyatukan semuanya dalam 1 baris:

`03.autoload_namespace\07.autoload_namespace_minimalist.php`

```

1 <?php
2 spl_autoload_register(function ($className) {
3     require str_replace('\\', '/', strtolower("$className.php"));
4 });
5
6 use DuniaIlkom\Library\Mobil as Mobil;
7 use DuniaIlkom\Plugin\SepedaMotor as SepedaMotor;
8
9 $produk01 = new Mobil("Toyota");
10 echo $produk01->getInfo();
11
12 echo "<br>";
13
14 $produk02 = new SepedaMotor("Yamaha");
15 echo $produk02->getInfo();

```

Pada baris 3, saya menggabungkan fungsi `strtolower()`, `str_replace()`, serta proses `require`. Penulisan seperti ini menjadi lebih ringkas namun sedikit lebih susah dibaca bagi yang belum terbiasa menggabungkan berbagai fungsi PHP ke dalam 1 baris perintah.

Dalam bab ini kita membahas tentang proses **autoloading**, yakni meng-import file PHP secara otomatis ke dalam file saat ini. Namun yang sudah kita lihat, proses autoloading ini butuh beberapa syarat tambahan, seperti nama file yang harus sama dengan nama class, serta nama namespace yang juga harus sama dengan struktur fisik folder.

Sama seperti namespace, autoloading lebih cocok dipakai untuk project yang relatif besar dengan banyak file terpisah.

Berikutnya kita akan masuk ke konsep pemrograman object yang hampir selalu tersedia di bahasa pemrograman yang menerapkan prinsip OOP, yakni **exception**.

7. Exception

Exception merupakan salah satu fitur standar dari *object oriented programming*. Hampir semua bahasa pemrograman yang menerapkan prinsip OOP juga memiliki **exception** (tidak hanya PHP saja). Dalam bab ini kita akan membahas lebih jauh apa itu exception dan bagaimana cara penggunaannya.

7.1. Pengertian Exception

Exception adalah sebuah mekanisme untuk mengelola error secara lebih "elegan". Dalam prakteknya nanti, PHP juga menyediakan class yang bernama **Exception**. Karena dipakai untuk mengelola error, istilah exception ini sering juga disebut sebagai *exception handling*, yakni cara penanganan exception.

Sebelum kita masuk ke praktik penggunaan exception, saya akan bahas secara bertahap bagaimana cara pengelolaan error yang selama ini kita pakai, terutama dalam *procedural programming*.

Perhatikan kode program berikut ini:

01.foo_function.php

```
1 <?php
2 function foo($a){
3     return 1/$a;
4 }
5
6 echo foo(2);      echo "<br>";
7 echo foo(100);    echo "<br>";
8 echo foo(0);       echo "<br>";
9 echo foo(-20);    echo "<br>";
```

Di awal kode program saya mendefinisikan fungsi `foo()` dengan sebuah parameter `$a`. Isi fungsi `foo()` sendiri sangat sederhana, yakni mengembalikan hasil operasi matematis dari `1/$a`.

Kemudian di baris 6 – 9 terdapat 4 kali pemanggilan fungsi `foo()` dengan nilai argument yang berbeda-beda: `foo(2)`, `foo(100)`, `foo(0)` dan `foo(-20)`. Berikut hasilnya:

0.5
0.01

Exception

```
Warning: Division by zero in C:\xampp\htdocs\belajar_oop_php\
bab_07\01.foo_function.php on line 3
INF
-0.05
```

Pemanggilan fungsi `foo(2)`, `foo(100)` dan `foo(-20)` tidak bermasalah. Namun muncul error pada saat menjalankan `foo(0)`.

Dalam matematika, sebuah angka tidak bisa dibagi dengan nol. Pada beberapa bahasa pemrograman, hasil dari operasi 1/0 adalah `undefined`, `NaN` (*not a number*), atau `infinity` (tidak berhingga). Dalam hal ini PHP memberikan pesan error level warning: "Division by zero", kemudian menampilkan hasil `INF` yang merupakan singkatan dari `infinity`.

Bagaimana jika kita tidak ingin pesan error ini tampil? Caranya bisa dengan me-nonaktifkan pengaturan error display PHP, atau menggunakan trik menambah tanda '@' di awal perintah yang di prediksi akan menghasilkan error:

02.error_handling_at.php

```
1 <?php
2 function foo($a){
3     return 1/$a;
4 }
5
6 echo foo(2);      echo "<br>";
7 echo foo(100);    echo "<br>";
8 echo @foo(0);     echo "<br>";
9 echo foo(-20);    echo "<br>";
```

Hasil kode program:

```
0.5
0.01
INF
-0.05
```

Sekarang tidak tampil lagi pesan error, namun ini juga bukan solusi yang baik. Jika terjadi pembagian dengan nol, maka besar kemungkinan ada sesuatu yang salah.

Alternatif lain, kita bisa membuat proses validasi di dalam fungsi `foo()`, yakni menampilkan pesan error jika argument `$a` diisi dengan angka nol. Berikut modifikasi kode program sebelumnya:

03.error_handling_if.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         echo "Error: Argument \$a tidak bisa diisi angka 0";
5     }
6     else {
```

Exception

```
7     return 1/$a;
8 }
9 }
10
11 echo foo(2);    echo "<br>";
12 echo foo(100);  echo "<br>";
13 echo foo(0);    echo "<br>";
14 echo foo(-20);  echo "<br>";
```

Hasil kode program:

```
0.5
0.01
Error: Argument $a tidak bisa diisi angka 0
-0.05
```

Sedikit lebih baik. Saya menambah sebuah kondisi `if ($a === 0)` ke dalam fungsi `foo()`. Jika kondisi ini bernilai `true`, yakni terdapat pemanggilan fungsi `foo()` dengan argument angka 0, maka tampilkan pesan "Error: Argument \$a tidak bisa diisi angka 0".

Bagaimana jika saya ingin membatasi bahwa fungsi `foo()` juga tidak bisa dipanggil dengan argument angka negatif? Tinggal buat kondisi if kedua:

04.error_handling_if_2.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         echo "Argument \$a tidak bisa diisi angka 0";
5     }
6     else if ($a < 0){
7         echo "Argument \$a tidak bisa diisi angka negatif";
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 echo foo(2);    echo "<br>";
15 echo foo(100);  echo "<br>";
16 echo foo(0);    echo "<br>";
17 echo foo(-20);  echo "<br>";
```

Hasil kode program:

```
0.5
0.01
Argument $a tidak bisa diisi angka 0
Argument $a tidak bisa diisi angka negatif
```

Dalam kasus tertentu, kita juga sering memakai fungsi `die()` untuk memaksa PHP berhenti memproses jika terjadi kesalahan:

Exception

05.error_handling_die.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         die("Argument \$a tidak bisa diisi angka 0");
5     }
6     else if ($a < 0){
7         die("Argument \$a tidak bisa diisi angka negatif");
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 echo foo(2);      echo "<br>";
15 echo foo(100);     echo "<br>";
16 echo foo(0);       echo "<br>";
17 echo foo(-20);    echo "<br>";
```

Hasil kode program:

```
0.5
0.01
Argument $a tidak bisa diisi angka 0
```

Sekarang begitu terjadi kesalahan, PHP akan langsung berhenti dan tidak mengeksekusi perintah lain.

Inilah beberapa cara yang biasa di pakai ketika menangani error dalam pemrograman prosedural PHP. Untuk kode program yang sederhana, cara tersebut sudah mencukupi. Namun dalam pemrograman object yang lebih rumit, kita bisa menggunakan **exception** sebagai cara alternatif penanganan kesalahan.

Berikut modifikasi kode program sebelumnya dengan menggunakan **exception**:

06.exception_basic.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 echo foo(2) . "<br>";
12 echo foo(100) . "<br>";
13 echo foo(0) . "<br>";
14 echo foo(-20) . "<br>";
```

Exception

Hasil kode program:

```
0.5  
0.01
```

```
Fatal error: Uncaught Exception: Argument $a tidak bisa diisi angka 0 in C:\xampp\htdocs\belajar_oop_php\bab_07\06.exception_basic.php:4 Stack trace: #0 C:\xampp\htdocs\belajar_oop_php\bab_07\06.exception_basic.php(13): foo(0) #1 {main} thrown in C:\xampp\htdocs\belajar_oop_php\bab_07\06.exception_basic.php on line 4
```

Perhatikan perintah di baris 4, yakni `throw new Exception("Argument \$a tidak bisa diisi angka 0")`. Inilah cara pembuatan pesan error menggunakan exception.

Hasilnya tampil pesan "Fatal error: Uncaught Exception" karena exception yang sudah "dilempar" (**throw**), harus "ditangkap" (**catch**) agar bisa di proses lebih lanjut. Untuk menangkap pesan exception kita perlu membahas sebuah struktur logika baru, yakni **try** dan **catch**.

7.2. Struktur try - catch

Pada contoh sebelumnya kita telah melihat sekilas cara pembuatan exception dengan perintah **throw**. Agar bisa diproses, kode program yang akan menghasilkan sebuah exception harus berada di dalam struktur **try - catch**.

Berikut konsep dasar penulisannya:

```
1  try {  
2      // kode program yang akan menghasilkan exception  
3      throw new Exception ("pesan kesalahan")  
4  }  
5  catch (Exception $e) {  
6      // kode program yang akan memproses exception  
7  }
```

Struktur **try - catch** ini mirip seperti struktur **if else**. Pertama, PHP akan mencoba memproses seluruh kode program yang ada di dalam block **try**. Apabila tidak ada perintah yang menghasilkan exception, maka block kode **catch** akan dilewati dan PHP lanjut memproses perintah setelah blok **catch**.

Namun jika terdapat perintah yang menghasilkan exception, maka PHP akan pindah ke bagian **catch** untuk "menangkap" exception tersebut. Jika tidak ada kode program yang memproses exception (tidak ada block kode **catch**), akan tampil "Fatal error: Uncaught Exception" seperti contoh kita sebelumnya.

Di awal perintah **catch**, terdapat baris `(Exception $e)`. Baris ini berfungsi mirip seperti argument di dalam sebuah function. Sepanjang block **catch**, variabel `$e` bisa dipakai untuk mengakses exception object, yakni object hasil exception yang "dilempar" dari block **try**.

Exception

Nama variabel \$e ini boleh bebas, tidak harus ditulis sebagai \$e, tapi bisa juga nama lain seperti \$ex, \$exception, atau \$objectException. Aturan penamaan variabel ini sama seperti argument biasa pada function.

Berikut contoh pengaksesan object hasil exception:

07.exception_try_catch.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 try {
12     echo foo(0);
13 }
14 catch (Exception $e) {
15     echo $e->getMessage();
16 }
```

Hasil kode program:

```
Argument $a tidak bisa diisi angka 0
```

Kode program ini bisa dibagi ke dalam 2 kelompok: pendefinisian fungsi foo() di baris 2 -9, serta block **try catch** di baris 11 – 16.

Isi dari fungsi foo() masih sama seperti sebelumnya, yakni "melemparkan" exception jika argument \$a berisi angka 0, atau mengembalikan hasil dari 1/\$a. Karena fungsi foo() bisa mengembalikan exception, maka proses pemanggilan fungsi ini harus berada di dalam blok **try catch**.

Pertama kali, PHP akan mencoba menjalankan isi dari blok **try** yang dalam contoh kita hanya terdiri dari 1 baris: echo foo(0). Jika ternyata hasil pemanggilan fungsi foo() menghasilkan exception, kode program langsung beralih ke block **catch**.

Pada block **catch**, argument \$e dipakai untuk menampung object exception yang tadinya sudah "dilemparkan". Seperti yang akan kita bahas sesaat lagi, object exception ini memiliki berbagai method, dimana salah satunya adalah method getMessage(). Jika dipanggil dengan perintah \$e->getMessage(), hasilnya berupa pesan error yang ditulis di dalam exception.

Pada saat pembuatan exception di dalam fungsi foo(), saya menulisnya sebagai throw new Exception("Argument \\$a tidak bisa diisi angka 0"). Teks inilah yang ditampilkan kembali menggunakan method \$e->getMessage().

Exception

Selain menampilkan teks, object exception juga berisi berbagai informasi mengenai error yang terjadi, seperti nama file tempat exception dijalankan, di baris ke berapa error terjadi, dll. Berikut berbagai method yang bisa diakses dari **exception** object:

08.exception_method.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0",99);
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 try {
12     echo foo(0);
13 }
14 catch (Exception $e) {
15     echo $e->getMessage() . "<br>";
16     echo $e->getCode() . "<br>";
17     echo $e->getFile() . "<br>";
18     echo $e->getLine() . "<br>";
19     echo $e->getTraceAsString() . "<br>";
20
21     echo "<pre>";
22     print_r( $e->getTrace() );
23     echo "</pre>";
24 }
```

Hasil kode program:

```
Argument $a tidak bisa diisi angka 0
99
C:\xampp\htdocs\belajar_oop_php\bab_07\08.exception_method.php
4
#0 C:\xampp\htdocs\belajar_oop_php\bab_07\08.exception_method.php(12): foo(0) #1
{main}
Array
(
    [0] => Array
        (
            [file] => C:\xampp\htdocs\belajar_oop_php\bab_07\08.exception_method.php
            [line] => 12
            [function] => foo
            [args] => Array
                (
                    [0] => 0
                )
        )
)
```

Dalam kode program ini saya mengakses 6 method dari exception object:

1. Method `$e->getMessage()` dipakai untuk menampilkan pesan error yang ditulis sebagai argument pertama pada saat pembuatan exception. Ini sudah kita bahas sebelumnya.
2. Method `$e->getCode()` dipakai untuk menampilkan kode error yang ditulis sebagai argument kedua pada saat pembuatan exception. Di baris 4, argument kedua (setelah pesan error) berupa angka 99, dengan demikian method `getCode()` juga akan menampilkan angka 99. Kode error ini bisa di pakai untuk pemrosesan lebih lanjut. Misalnya jika kode error yang dihasilkan sama, berarti error yang terjadi juga berjenis sama.
3. Method `$e->getFile()` dipakai untuk menampilkan alamat path serta nama file tempat exception terjadi. Dalam contoh ini adalah `C:\xampp\htdocs\belajar_oop_php\bab_07\08.exception_method.php`. Ini merupakan alamat file kode program yang saya tulis.
4. Method `$e->getLine()` dipakai untuk menampilkan baris tempat exception "dilempar". Karena penulisan exception ada di baris 4 (dalam fungsi `foo()`), maka hasil method ini berupa angka 4.

Ke-4 method di atas, yakni `getMessage()`, `getCode()`, `getFile()`, dan `getLine()`, dipakai untuk mencari **lokasi perintah exception**, bukan lokasi dari kode program yang menyebabkan exception terjadi.

Dalam contoh kita, kode program yang menyebabkan terjadinya exception ada di baris 12, yakni pada saat pemanggilan fungsi `foo(0)`. Untuk mencari lokasi ini, bisa memakai method `$e->getTraceAsString()` dan `$e->getTrace()`.

5. Method `$e->getTraceAsString()` dipakai untuk menampilkan **trace error** dalam bentuk string. Trace error sendiri berisi penjelasan detail tentang lokasi error yang akan dibahas dalam method `getTrace()`.
6. Method `$e->getTrace()` dipakai untuk menampilkan trace error dalam bentuk array multi-dimensi, yakni ada array di dalam array di dalam array. Tampilan seperti ini diperlukan karena bisa saja sumber error tersebut "terpendam" di dalam kode program yang saling memanggil (kita akan bahas contoh prakteknya nanti).

Isi dari `$e->getTrace()[0]` adalah informasi penyebab error pertama, `$e->getTrace()[1]` berisi informasi penyebab error kedua, `$e->getTrace()[2]` berisi informasi penyebab error ketiga, dst.

Dimensi kedua dari `$e->getTrace()` memiliki 4 element:

- [file]: berisi nama file tempat error terjadi, yakni: `C:\xampp\htdocs\belajar_oop_php\bab_07\08.exception_method.php`.

Exception

- [line]: berisi urutan baris terjadi error, yakni baris 12.
- [function]: berisi nama fungsi yang menyebabkan error, yakni foo.
- [args]: berisi array dari nama argument pada saat pemanggilan fungsi. Dalam contoh ini fungsi foo() hanya memiliki 1 argument, yakni angka 0 yang diinput pada saat pemanggilan fungsi foo(0).

Karena method `$e->getTrace()` menghasilkan array multi-dimensi, maka jika array trace ini ingin dipanggil secara langsung, caranya adalah `$e->getTrace()[0][file]`, `$e->getTrace()[0][line]`, dst.

Agar penulisan ini bisa dipahami, silahkan anda pelajari sejenak struktur array hasil dari perintah `print_r($e->getTrace())` dalam contoh sebelumnya.

Sebagai pengingat, method `$e->getLine()` dengan `$e->getTrace()[0][line]` sama-sama menampilkan urutan baris, namun method `$e->getLine()` berisi **lokasi exception**, sedangkan method `$e->getTrace()[0][line]` berisi urutan baris **tempat error** yang sebenarnya terjadi.

Dengan semua informasi ini, kita bisa membuat pesan error yang lebih rapi:

09.exception_method_pesan.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 try {
12     echo foo(0);
13 }
14 catch (Exception $e) {
15     echo "Terjadi error dalam file <b>".$e->getTrace()[0]["file"]."</b>,
16         di baris ke-".$e->getTrace()[0]["line"]." dengan keterangan <b>".
17         $e->getMessage()."</b>.";
```

Hasil kode program:

```
Terjadi error dalam file C:\xampp\htdocs\belajar_oop_php\
bab_07\09.exception_method_pesan.php, di baris ke-12 dengan keterangan Argument $a
tidak bisa diisi angka 0
```

Atau bisa juga di format dengan sedikit "design" seperti kode berikut:

Exception

10.exception_method_h1.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 try {
12     echo foo(0);
13 }
14 catch (Exception $e) {
15     echo "<h1 style='text-align:center'>==== Error ====</h1>";
16     echo "<hr>";
17     echo "<h2 style='text-align:center'>". $e->getMessage(). "</h2>";
18     echo "<p style='text-align:center'> Baris ke- ". $e->getTrace()[0]["line"] .
19         ", di dalam ". $e->getTrace()[0]["file"]. "</p>";
20 }
```



Gambar: Tampilan pesan error exception yang lebih rapi

Object hasil exception juga bisa langsung di echo yang akan menampilkan semua informasi exception dalam bentuk sebuah string panjang:

11.exception_echo.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 try {
12     echo foo(0);
13 }
```

Exception

```
14 catch (Exception $e) {  
15     echo $e;  
16 }
```

Hasil kode program:

```
Exception: Argument $a tidak bisa diisi angka 0 in C:\xampp\htdocs\belajar_oop_php\bab_07\11.exception_echo.php:4 Stack trace: #0 C:\xampp\htdocs\belajar_oop_php\bab_07\11.exception_echo.php(12): foo(0) #1 {main}
```

Jika anda masih ingat, sebuah object hanya bisa di-echo jika dalam class asal object tersebut terdapat method `__toString()`. Inilah yang dipakai secara internal oleh class **Exception** PHP.

Block **try catch** tidak hanya bisa diisi satu perintah saja, namun juga bisa banyak perintah sekaligus. Jika ada di antara perintah ini ada yang menghasilkan exception, pemrosesan block **try** langsung berhenti dan kode program akan "lompat" ke bagian **catch**. Berikut contohnya:

12.exception_try_catch_2.php

```
1 <?php  
2 function foo($a){  
3     if ($a === 0){  
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");  
5     }  
6     else {  
7         return 1/$a;  
8     }  
9 }  
10  
11 try {  
12     echo foo(2)    . "<br>";  
13     echo foo(100)   . "<br>";  
14     echo foo(0)     . "<br>";  
15     echo foo(-20)   . "<br>";  
16 }  
17 catch (Exception $e) {  
18     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"].  
19     " dengan keterangan <b>".$e->getMessage()."</b><br>";  
20 }  
21  
22 echo "Selesai";
```

Hasil kode program:

```
0.5  
0.01  
Terjadi error di baris ke-14 dengan keterangan Argument $a tidak bisa diisi angka 0  
Selesai
```

Kali ini saya menjalankan 4 kali fungsi `foo()` di dalam block **try**. Pemanggilan `foo(2)` dan `foo(100)` tidak ada masalah. Namun ketika sampai di `foo(0)`, ini akan menghasilkan exception. Pada saat itu juga PHP langsung lompat ke bagian **catch** untuk memproses exception.

Exception

Setelah itu, PHP akan lanjut memproses kode program setelah block **catch** (baris 22). Fungsi `foo(-20)` di baris 15 tidak akan diproses karena berada setelah kode program yang menghasilkan exception.

Bagaimana jika yang kita inginkan adalah fungsi `foo(-20)` ini tetap bisa diproses meskipun terjadi exception? Solusinya, pisah pemanggilan setiap fungsi `foo()` ke dalam blok **try catch** masing-masing. Maksudnya, setiap pemanggilan fungsi `foo()` berada di dalam 1 blok **try catch**:

13.exception_try_catch_terpisah.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 try {
12     echo foo(2)    . "<br>";
13 }
14 catch (Exception $e) {
15     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
16     " dengan keterangan <b>".$e->getMessage()."</b><br>";
17 }
18
19 try {
20     echo foo(100) . "<br>";
21 }
22 catch (Exception $e) {
23     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
24     " dengan keterangan <b>".$e->getMessage()."</b><br>";
25 }
26
27 try {
28     echo foo(0)    . "<br>";
29 }
30 catch (Exception $e) {
31     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
32     " dengan keterangan <b>".$e->getMessage()."</b><br>";
33 }
34
35 try {
36     echo foo(-20) . "<br>";
37 }
38 catch (Exception $e) {
39     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
40     " dengan keterangan <b>".$e->getMessage()."</b><br>";
41 }
```

Hasil kode program:

Exception

```
0.5  
0.01  
Terjadi error di baris ke-28 dengan keterangan Argument $a tidak bisa diisi angka 0  
-0.05
```

Sekarang, setiap pemanggilan fungsi `foo()` sudah saling terpisah. Jika terjadi exception, itu akan diproses oleh setiap blok dan tidak mempengaruhi blok lain. Jika kita membuat banyak pemrosesan exception seperti ini, akan lebih praktis membuat sebuah fungsi bantu untuk pemrosesan exception:

14.exception_try_catch_terpisah_function.php

```
1 <?php  
2 function foo($a){  
3     if ($a === 0){  
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");  
5     }  
6     else {  
7         return 1/$a;  
8     }  
9 }  
10  
11 function tampilkanException($e){  
12     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"].  
13     " dengan keterangan <b>".$e->getMessage()."</b><br>";  
14 }  
15  
16 try {  
17     echo foo(2) . "<br>";  
18 }  
19 catch (Exception $e) {  
20     tampilkanException($e);  
21 }  
22  
23 try {  
24     echo foo(100) . "<br>";  
25 }  
26 catch (Exception $e) {  
27     tampilkanException($e);  
28 }  
29  
30 try {  
31     echo foo(0) . "<br>";  
32 }  
33 catch (Exception $e) {  
34     tampilkanException($e);  
35 }  
36  
37 try {  
38     echo foo(-20) . "<br>";  
39 }  
40 catch (Exception $e) {  
41     tampilkanException($e);
```

Exception

42 }

Di baris 11 – 14 saya membuat fungsi `tampilkanException()`. Fungsi ini dirancang untuk menampilkan pesan error sesuai format yang di inginkan. Dalam setiap block `catch`, object exception akan dikirim sebagai argument untuk fungsi `tampilkanException()` ini.

Variasi lain dari pembuatan exception adalah kita bisa membuat pesan exception yang berbeda-beda. Dalam contoh sebelumnya, fungsi `foo()` hanya menghasilkan exception jika argument `$a` berisi angka 0. Bagaimana jika kita tambah proses validasi lain, misalnya membuat pesan error ketika argument `$a` diisi angka negatif? Berikut kode program yang bisa dipakai:

15.multiple_exception.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception("Argument \$a tidak bisa diisi angka 0");
5     }
6     else if ($a < 0){
7         throw new Exception("Argument \$a tidak bisa diisi angka negatif");
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 try {
15     echo foo(0);
16 }
17 catch (Exception $e) {
18     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"].
19     " dengan keterangan <b>".$e->getMessage()."</b><br>";
20 }
21
22 try {
23     echo foo(-20);
24 }
25 catch (Exception $e) {
26     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"].
27     " dengan keterangan <b>".$e->getMessage()."</b><br>";
28 }
```

Hasil kode program:

Terjadi error di baris ke-15 dengan keterangan Argument \$a tidak bisa diisi angka 0
Terjadi error di baris ke-24 dengan keterangan Argument \$a tidak bisa diisi angka negatif

Di baris 6 – 8 saya membuat proses validasi kedua untuk memeriksa apakah argument `$a` berisi angka kurang dari nol (angka negatif). Jika benar (`true`), lempar sebuah exception

dengan pesan "Argument \\$a tidak bisa diisi angka negatif". Pesan ini diinput sebagai argument dari object exception.

7.3. Custom Exception

Pada contoh sebelum ini kita membuat 2 buah object exception dengan pesan error yang berbeda-beda. Namun kedua object tersebut tetap 1 jenis, yakni sama-sama berasal dari class **Exception** bawaan PHP.

Untuk situasi yang lebih rumit, kita bisa membuat beberapa jenis class **Exception**, yang nantinya akan ditangkap oleh kondisi **catch** yang berbeda juga. Ini dikenal dengan istilah *custom exception*.

Custom exception adalah proses pembuatan class **Exception** baru dengan cara diturunkan dari class **Exception** bawaan PHP.

Misalnya saya ingin membuat 2 buah class Exception baru, perintah yang dipakai adalah sebagai berikut:

```
class NolException extends Exception{}
class NegatifException extends Exception{}
```

Di sini saya membuat 2 buah class baru: **NolException** dan **NegatifException**. Kedua class ini diturunkan dari class **Exception** bawaan PHP.

Selanjutnya, kedua class ini bisa "dilempar" sebagaimana exception biasa:

```
1 function foo($a){
2     if ($a === 0){
3         throw new NolException();
4     }
5     else if ($a < 0){
6         throw new NegatifException();
7     }
8     else {
9         return 1/$a;
10    }
11 }
```

Fungsi dari **custom exception** ini adalah agar kita bisa membuat berbagai block **catch** untuk menangkap exception sesuai dengan nama classnya.

16.custom_exception.php

```
1 try {
2     echo foo(0);
3 }
4 catch (NolException $e) {
5     echo "Argument tidak bisa diisi angka 0 <br>";
6 }
```

Exception

```
7  catch (NegatifException $e) {
8      echo "Argument tidak bisa diisi angka negatif <br>";
9  }
10
11 try {
12     echo foo(-20);
13 }
14 catch (NolException $e) {
15     echo "Argument tidak bisa diisi angka 0 <br>";
16 }
17 catch (NegatifException $e) {
18     echo "Argument tidak bisa diisi angka negatif <br>";
19 }
```

Hasil kode program:

```
Argument tidak bisa diisi angka 0
Argument tidak bisa diisi angka negatif
```

Pada masing-masing block **try**, diikuti dengan 2 buah block **catch**. Setiap block catch memiliki type hinting sesuai nama class exception yang akan ditangkap (satu untuk setiap class Exception).

Sebagai contoh, di baris 2 terdapat pemanggilan fungsi `foo(0)`. Di dalam fungsi `foo()`, argument 0 ini akan membuat kode program mengeksekusi perintah `throw new NolException()`, yakni "melempar" object **NolException**.

Karena yang dilempar adalah object **NolException**, maka block **catch** yang akan menangkap terdapat di baris 4 – 5, yakni sesuai dengan type hinting yang tertulis: `catch(NolException $e)`.

Berikutnya di baris 12 terdapat pemanggilan fungsi `foo(-200)`. Dalam fungsi `foo()`, argument dengan angka negatif akan membuat kode program mengeksekusi perintah `throw new NegatifException()`, yakni melempar object **NegatifException**.

Karena yang dilempar adalah object **NegatifException**, maka block **catch** yang akan "menangkap" ada di baris 17 – 19, sesuai dengan type hinting yang tertulis: `catch (NegatifException $e)`.

Penjelasan ini memperlihatkan bahwa block **catch** hanya akan memproses exception sesuai dengan type hinting yang tertulis.

Salah satu analogi yang bisa dipakai untuk menggambarkan **custom exception** adalah sebagai berikut:

Bayangkan kode program ibarat gedung bertingkat yang diproses secara berurutan dari lantai paling atas sampai lantai paling bawah. Ketika terjadi exception, sebuah batu besar akan dilempar keluar jendela (**throw exception**).

Selama proses "jatuh" ini, batu tersebut harus ditangkap dengan jaring agar tidak mengenai orang di bawah (**catch exception**). Jika jaringnya tidak ada, maka terjadilah *Fatal error: Uncaught Exception*.

Syarat lain, jaring penangkap ini harus sesuai dengan jenis batu yang dilempar (**custom exception**). Jika yang di lempar adalah batu warna merah, maka jaring yang disediakan juga harus berwarna merah. Jika ternyata jaring yang dipakai adalah jaring biru, itu tidak bisa dipakai untuk menangkap batu merah dan kembali terjadi *Fatal error: Uncaught Exception*.

Idealnya, kita harus menyediakan jaring untuk semua jenis batu.

Di dalam class **custom exception**, kita juga bisa membuat method tambahan yang nantinya bisa dipanggil pada block **catch**. Berikut contoh prakteknya:

17.custom_exception_2.php

```

1 <?php
2 class NolException extends Exception{
3     public function pesanKesalahan(){
4         return "Argument tidak bisa diisi angka 0, di baris "
5             . $this->getTrace()[0]["line"] . " <br>";
6     }
7 }
8
9 class NegatifException extends Exception{
10    public function pesanKesalahan(){
11        return "Argument tidak bisa diisi angka negatif, di baris "
12            . $this->getTrace()[0]["line"] . " <br>";
13    }
14 }
15
16 function foo($a){
17     if ($a === 0){
18         throw new NolException();
19     }
20     else if ($a < 0){
21         throw new NegatifException();
22     }
23     else {
24         return 1/$a;
25     }
26 }
27
28 try {
29     echo foo(0);
30 }
31 catch (NolException $e) {
32     echo $e->pesanKesalahan();
33 }
34 catch (NegatifException $e) {

```

Exception

```
35     echo $e->pesanKesalahan();
36 }
37
38 try {
39     echo foo(-20);
40 }
41 catch (NolException $e) {
42     echo $e->pesanKesalahan();
43 }
44 catch (NegatifException $e) {
45     echo $e->pesanKesalahan();
46 }
```

Hasil kode program:

```
Argument tidak bisa diisi angka 0, di baris 29
Argument tidak bisa diisi angka negatif, di baris 39
```

Di baris 2 – 14 saya kembali membuat class `NolException` dan `NegatifException` yang diturunkan dari class `Exception`. Namun sekarang terdapat tambahan definisi method `pesanKesalahan()`. Isinya berbentuk string pesan kesalahan yang sudah beberapa kali kita tulis.

Dengan tambahan method ini, ketika terjadi exception, method `pesanKesalahan()` bisa dipanggil dengan perintah `$e->pesanKesalahan()`.

Untuk menyederhanakan contoh kode program, saya memakai nama method yang sama pada class `NolException` dan `NegatifException`, yakni `pesanKesalahan()`. Pada prakteknya, anda bisa menggunakan nama method yang berbeda untuk setiap custom exception

Jika terdapat class exception yang tidak memiliki block `catch`, akan tampil pesan error meskipun itu adalah class `Exception` bawaan PHP:

18.custom_exception_error.php

```
1 <?php
2 class NolException extends Exception{
3     public function pesanKesalahan(){
4         return "Argument tidak bisa diisi angka 0, di baris "
5             . $this->getTrace()[0]["line"] . " <br>";
6     }
7 }
8
9 class NegatifException extends Exception{
10    public function pesanKesalahan(){
11        return "Argument tidak bisa diisi angka negatif, di baris "
12            . $this->getTrace()[0]["line"] . " <br>";
13    }
14 }
15
16 function foo($a){
17     if ($a === 0){
```

Exception

```
18     throw new NolException();
19 }
20 else if ($a < 0){
21     throw new NegatifException();
22 }
23 else if (!is_numeric($a)){
24     throw new Exception('Argument yang diinput bukan angka');
25 }
26 else {
27     return 1/$a;
28 }
29 }
30
31 try {
32     echo foo('a');
33 }
34 catch (NolException $e) {
35     echo $e->pesanKesalahan();
36 }
37 catch (NegatifException $e) {
38     echo $e->pesanKesalahan();
39 }
```

Hasil kode program:

```
Fatal error: Uncaught Exception: Argument yang diinput bukan angka
```

Pendefinisian class **NolException** dan **NegatifException** masih sama seperti sebelumnya.

Tambahan dalam kode program di atas ada di baris 24 – 26. Di sini saya membuat 1 lagi proses validasi untuk fungsi `foo()`. Jika yang diinput sebagai argument `$a` bukan tipe data angka, maka `throw new Exception('Argument yang diinput bukan angka')`.

Di baris 33, saya coba jalankan fungsi `foo('a')`. Karena argument yang diinput bukanlah angka, maka yang dilempar adalah class `Exception`. Namun di dalam block **catch** tidak terdapat kondisi yang menerima class ini. Yang tersedia hanyalah block **catch** untuk menangkap `NolException` dan `NegatifException`. Hasilnya, tampil pesan "Fatal error: Uncaught Exception".

Solusinya, kita harus membuat block **catch** ketiga untuk menangkap class `Exception` ini.

7.4. Function `set_exception_handler()`

PHP menyediakan fungsi khusus yang secara otomatis akan diproses ketika mendapati ada exception yang tidak ditangkap. Fungsi tersebut adalah `set_exception_handler()`.

Berikut contoh praktik penggunaannya:

```
19.set_exception_handler.php
```

```
1 <?php
```

Exception

```
2 class NolException extends Exception{}
3 class NegatifException extends Exception{}
4
5 set_exception_handler(function($e){
6     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
7     " dengan keterangan <b>".$e->getMessage()."</b><br>";
8 });
9
10 function foo($a){
11     if ($a === 0){
12         throw new NolException("Argument tidak bisa diisi angka 0");
13     }
14     else if ($a < 0){
15         throw new NegatifException("Argument tidak bisa diisi angka negatif");
16     }
17     else if (!is_numeric($a)){
18         throw new Exception('Argument yang diinput bukan angka');
19     }
20     else {
21         return 1/$a;
22     }
23 }
24
25 try {
26     echo foo('X');
27 }
28 catch (NolException $e) {
29     echo $e->getMessage();
30 }
```

Hasil kode program:

Terjadi error di baris ke-26 dengan keterangan Argument yang diinput bukan angka

Di baris 2 – 3 saya membuat class `NolException` dan `NegatifException` sebagai turunan dari class `Exception`. Kali ini tanpa method tambahan untuk menyederhanakan contoh kita.

Kemudian di baris 5 – 8 terdapat pendefinisian fungsi `set_exception_handler()`. Sebagai argument untuk fungsi ini adalah sebuah *anonymous function* dengan argument `$e` yang nantinya berisi object hasil exception. Dalam fungsi ini saya menulis string untuk menampilkan pesan kesalahan sama seperti contoh-contohnya.

Di baris 10 – 23 terdapat pendefinisian fungsi `foo()` dengan 3 kondisi yang akan menghasilkan exception, yakni apakah argument berupa angka 0, berupa angka negatif, atau bukan berupa angka. Setiap kondisi akan melempar exception yang berbeda-beda.

Di baris 25 – 30, saya menjalankan fungsi `foo('X')`. Pemanggilan ini cocok dengan kondisi di baris 17, yakni argument yang diinput bukan berbentuk angka, dengan demikian class `Exception` akan dilempar.

Namun di dalam block `catch`, tidak ada block kode yang bisa menangkap class `Exception`. Yang

Exception

tersedia hanya untuk class N0lException.

Dalam situasi seperti inilah fungsi `set_exception_handler()` dipanggil untuk menangani exception yang tidak ditangkap. Hasilnya, berupa pesan error yang sudah dirancang dalam fungsi ini. Jika fungsi `set_exception_handler()` tidak ada, PHP akan menampilkan "Fatal error: Uncaught Exception".

Sebagai pengganti dari anonymous function, fungsi `set_exception_handler()` juga bisa diisi dengan fungsi terpisah:

20.set_exception_handler_terpisah.php

```
1 <?php
2 set_exception_handler('tampilkanError');
3
4 function tampilkanError($e) {
5     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
6         " dengan keterangan <b>".$e->getMessage()."</b><br>";
7 }
```

Ketika fungsi `set_exception_handler()` dipanggil, proses selanjutnya diserahkan kepada fungsi `tampilkanError()`. Argument `$e` nantinya akan berisi object exception hasil "pelemparan" yang tidak ditangkap oleh blok `catch`.

7.5. Exception Trace

Di awal pembahasan tentang exception object, kita telah melihat bahwa object exception memiliki method `getTrace()`. Method ini mengembalikan array multi dimensi terkait sumber error yang akan menghasilkan exception.

Method `getTrace()` ini akan berguna untuk kode program yang sedikit kompleks. Berikut contohnya:

21.exception_trace_problem.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception('Argument tidak bisa diisi angka 0');
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 function bar($b){
12     return foo($b);
13 }
14
15 try {
```

Exception

```
16     echo bar(0);
17 }
18 catch (Exception $e) {
19     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
20     " dengan keterangan <b>".$e->getMessage()."</b><br>";
21 }
```

Hasil kode program:

```
Terjadi error di baris ke-12 dengan keterangan Argument tidak bisa diisi angka 0
```

Pada kode program ini saya membuat 2 buah function, yakni `foo()` dan `bar()`. Fungsi `foo()` kurang lebih sama seperti contoh-contoh kita sebelumnya, dimana jika argument `$a` diisi angka 0, `throw new Exception`.

Pendefinisian fungsi `bar()` ada di baris 11 – 13. Isinya hanya pemanggilan ulang dari fungsi `foo()`. Block `try` di baris 15 – 21 mencoba menjalankan perintah `bar(0)`, yang tentu akan menghasilkan **exception**.

Fokus utama kita ada di hasil yang tampil: "Terjadi error di baris ke-12". Padahal jika diteliti lebih lanjut, sumber error sebenarnya ada di baris 16, yakni perintah `bar(0)`. Yang ada di baris 12 hanyalah pemanggilan ulang fungsi `foo()`.

Kode program ini adalah versi sederhana dari "error terpendam", yakni kasus dimana laporan error dari PHP menunjuk ke baris yang tidak sepenuhnya asal masalah. Namun PHP juga tidak bisa disalahkan karena baris 12 pun memiliki peran dalam error ini.

Semua data error sebenarnya bisa terlihat di dalam array hasil method `getTrace()`. Berikut hasil yang didapat:

22.exception_trace_array.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception('Argument tidak bisa diisi angka 0');
5     }
6     else {
7         return 1/$a;
8     }
9 }
10
11 function bar($b){
12     return foo($b);
13 }
14
15 try {
16     echo bar(0);
17 }
18 catch (Exception $e) {
19     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
```

Exception

```
20      " dengan keterangan <b>".$e->getMessage().</b><br>";
21
22 echo "<pre>";
23 print_r( $e->getTrace() );
24 echo "</pre>";
25 }
```

Hasil kode program:

Terjadi error di baris ke-12 dengan keterangan Argument tidak bisa diisi angka 0

```
Array
(
    [0] => Array
        (
            [file] => C:\xampp\htdocs\belajar_oop_php\bab_07\22.exception_trace_array.php
            [line] => 12
            [function] => foo
            [args] => Array
                (
                    [0] => 0
                )
        )

    [1] => Array
        (
            [file] => C:\xampp\htdocs\belajar_oop_php\bab_07\22.exception_trace_array.php
            [line] => 16
            [function] => bar
            [args] => Array
                (
                    [0] => 0
                )
        )
)
```

Perubahan dari kode sebelumnya ada di baris 22 – 24, dimana saya menambah perintah `print_r($e->getTrace())` untuk melihat secara detail apa isi dari array hasil method `getTrace()`.

Seperti yang terlihat, array ini berisi penjelasan tentang 2 buah lokasi error. Dalam array pertama dengan index [0], element [line] berisi angka 12. Sedangkan di array kedua dengan index [1], element [line] berisi angka 16. Ini artinya PHP menelusuri bahwa error terjadi di baris 12 serta baris 16.

Semakin "dalam" sumber error, array hasil `$e->getTrace()` ini juga akan menampung semakin banyak data.

Agar hasil penelusuran error menjadi lebih rapi, saya bisa memanfaatkan perulangan **foreach** untuk menampilkan element [line],[function] dan [args] dari semua array:

Exception

23.exception_trace_foreach.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception('Argument tidak bisa diisi angka 0');
5     }
6     else if ($a < 0){
7         throw new Exception("Argument \$a tidak bisa diisi angka negatif");
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 function bar($b){
15     return foo($b);
16 }
17
18 try {
19     echo bar(-10);
20 }
21 catch (Exception $e) {
22     echo "Error: <b>".$e->getMessage()."</b><br>";
23
24     echo "<br>Trace error: <br>";
25
26     foreach ($e->getTrace() as $value) {
27         echo "Baris ke-".$value["line"];
28         echo ", error di function ".$value["function"];
29         echo ", dengan argument ".$value["args"][0]."<br>";
30     }
31 }
```

Hasil kode program:

Error: Argument \$a tidak bisa diisi angka negatif

Trace error:

Baris ke-15, error di function foo, dengan argument -10
Baris ke-19, error di function bar, dengan argument -10

Di dalam fungsi `foo()` sekarang terdapat 2 buah proses validasi, yakni apakah `$a` sama dengan nol, atau `$a` berisi angka negatif. Untuk kedua kondisi ini, lempar Exception object.

Proses pemanggilan fungsi `foo()` juga sama seperti sebelumnya, yakni dipanggil melalui fungsi `bar()`. Ketika block kode `try` di proses, perintah `echo bar(-10)` akan diproses ulang sebagai `foo(-10)`.

Sekarang kita masuk ke block `catch`, di baris 26 - 30 saya membuat sebuah perulangan `foreach` untuk memproses array hasil `$e->getTrace()`. Untuk setiap element hasil `$e->getTrace()`, tampilkan isi dari element `["line"]`, `["function"]` dan `["args"]`[0].

Exception

Agar bisa memahami cara kerja perulangan ini, setidaknya anda harus memahami cara kerja **foreach**, kemudian analisa struktur array hasil dari pemanggilan `$e->getTrace()`.

Mari kita test dengan pemanggilan fungsi yang "lebih terpendam":

24.exception_trace_foreach_2.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception('Argument tidak bisa diisi angka 0');
5     }
6     else if ($a < 0){
7         throw new Exception("Argument \${$a} tidak bisa diisi angka negatif");
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 function bar($b){
15     return foo($b);
16 }
17
18 function baz($b){
19     return bar($b);
20 }
21
22 try {
23     echo baz(-10);
24 }
25 catch (Exception $e) {
26     echo "Error: <b>". $e->getMessage(). "</b><br>";
27
28     echo "<br>Trace error: <br>";
29
30     foreach ($e->getTrace() as $value) {
31         echo "Baris ke-".$value ["line"];
32         echo ", error di function ".$value ["function"];
33         echo ", dengan argument ".$value ["args"][0]. "<br>";
34     }
35 }
```

Mayoritas kode program di atas sama seperti contoh sebelumnya, namun sekarang terdapat 3 buah function: `foo()`, `bar()` dan `baz()`.

Pada saat block **try** di proses, terdapat pemanggilan fungsi `baz(-10)`. Dalam pendefinisian fungsi `baz()`, isinya ternyata pemanggilan dari fungsi `bar()`, sehingga yang diproses berikutnya adalah `bar(-10)`. Di dalam pendefinisian fungsi `bar()` ternyata isinya juga pemanggilan fungsi lain, yakni `foo()`, sehingga yang diproses adalah `foo(-10)`. Barulah pada saat fungsi `foo()` ini diproses, terjadi exception.

Exception

Pertanyaannya, di baris manakah error terjadi? PHP punya 3 jawaban yang bisa dilihat dari pemanggilan `$e->getTrace()`. Berikut hasil dari kode program di atas:

```
Error: Argument $a tidak bisa diisi angka negatif
```

Trace error:

```
Baris ke-16, error di function foo, dengan argument -10
Baris ke-20, error di function bar, dengan argument -10
Baris ke-24, error di function baz, dengan argument -10
```

Terlihat bahwa ada 3 titik kemungkinan error, yakni di baris 16, 20 dan 24. Inilah manfaat dari pemanggilan method `getTrace()`.

Dalam prakteknya nanti, kita tidak harus menulis kode program untuk menampilkan trace seperti ini. Biasanya, pesan error default dari PHP juga akan menampilkan semua hasil trace, hanya saja tampilannya berupa 1 string panjang yang agak susah dibaca.

Sebagai contoh, saya akan coba menjalankan fungsi `baz(-10)` tanpa menggunakan block **try catch**, hasilnya sebagai berikut:

25.trace_tanpa_try.php

```
1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception('Argument tidak bisa diisi angka 0');
5     }
6     else if ($a < 0){
7         throw new Exception("Argument \$a tidak bisa diisi angka negatif");
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 function bar($b){
15     return foo($b);
16 }
17
18 function baz($b){
19     return bar($b);
20 }
21
22 echo baz(-10);
```

Hasil kode program:



Gambar: hasil pemanggilan fungsi foo(-10)

Terlihat PHP menampilkan semua hasil trace dalam 1 baris panjang yang cukup sulit untuk dibaca.

Alternatif lain kita bisa menggunakan library tambahan untuk mempermudah proses *debugging* seperti [xdebug](#). Aplikasi debugging ini cocok dipakai untuk project besar yang bisa saja ada method dari sebuah class yang memanggil method dari class lain, dimana isinya juga memanggil method dari class lain (dan seterusnya...).

7.6. Struktur try - catch - finally

PHP 5.5 memperkenalkan tambahan block baru ke dalam struktur try – catch, yakni **finally**. Block **finally** ini bersifat opsional dan ditulis setelah block catch.

Semua kode program yang ada di dalam block **finally** akan di eksekusi terlepas ada tidaknya exception, seperti contoh berikut:

26.tryCatch_Finally.php

```

1 <?php
2 function foo($a){
3     if ($a === 0){
4         throw new Exception('Argument tidak bisa diisi angka 0');
5     }
6     else if ($a < 0){
7         throw new Exception("Argument \$a tidak bisa diisi angka negatif");
8     }
9     else {
10        return 1/$a;
11    }
12 }
13
14 echo "Sebelum try... <br>";
15
16 try {
17     echo foo(-10). "<br>";
18 }
19 catch (Exception $e) {
20     echo "Terjadi error di baris ke-".$e->getTrace()[0]["line"] .
21     " dengan keterangan <b>".$e->getMessage()."</b><br>";

```

Exception

```
22 }
23 finally {
24     echo "Di dalam finally... <br>";
25 }
26
27 echo "Setelah try... <br>";
```

Hasil kode program:

```
Sebelum try...
Terjadi error di baris ke-18 dengan keterangan Argument $a tidak bisa diisi angka negatif
Di dalam finally...
Setelah try...
```

Pada baris 23 – 25 terdapat block **finally**. Meskipun pemanggilan fungsi `echo foo(-10)` di baris 17 menghasilkan exception, block **finally** tetap diproses sebelum PHP lanjut ke baris 27.

Block **finally** ini tidak terlalu sering dipakai, tapi bisa berguna untuk proses "bersih-bersih", seperti menutup koneksi ke database MySQL.

Sebagai penutup bab exception ini, saya ingin menampilkan penggunaan block **try - catch** dalam pemrograman yang sebenarnya:

27.membuat_koneksi_mysql.php

```
1 <?php
2 // Set agar error menggunakan exception
3 mysqli_report(MYSQLI_REPORT_STRICT);
4
5 // Tes buat koneksi dengan MySQL Server
6 try {
7     $mysqli = new mysqli('localhost','root','','coba');
8 }
9 catch (mysqli_sql_exception $e) {
10    echo "Hasil pesan exception adalah: ".$e->getMessage();
11 }
```

Kita memang belum membahas tentang cara membuat koneksi ke database MySQL menggunakan metode object **mysqli**. Namun jika anda sudah pernah mempelajari `mysqli` extension secara procedural, kode di atas tidak terlalu susah untuk dipahami.

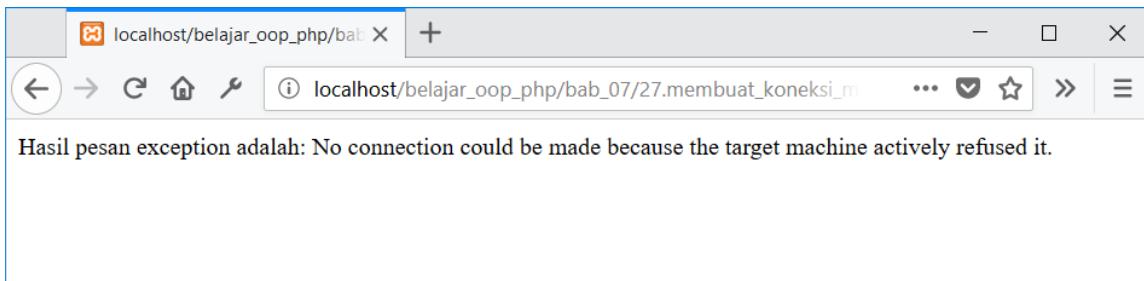
Di baris 3 saya menjalankan fungsi `mysqli_report(MYSQLI_REPORT_STRICT)`. Perintah ini adalah instruksi agar PHP agar menampilkan pesan error **mysqli** menggunakan exception. Ini perlu karena secara default pesan error `mysqli` tampil langsung sebagai string (bukan dengan exception).

Di baris 6 – 11 terdapat block **try - catch**. Pada baris 7 saya mencoba membuka koneksi dengan **MySQL Server**. Jika karena sesuatu hal proses ini gagal, PHP akan melempar sebuah exception.

Exception yang di lempar oleh class **mysqli** adalah **mysqli_sql_exception**. Inilah yang akan ditangkap dalam block **catch**. Isi dari block catch sendiri hanya 1 baris, dimana saya mengakses method `$e->getMessage()`.

Dalam PHP, semua class exception (apapun classnya), adalah turunan dari class **Exception** yang kita pelajari dalam bab ini. Jadi semua class tersebut akan memiliki method `getMessage()`, `getLine()`, `getTrace()`, dst.

Dengan sengaja tidak menjalankan MySQL Server, berikut hasil yang saya dapat:



Gambar: Pesan error karena mysqli gagal membuat koneksi ke MySQL Server

Tampilan ini memperlihatkan bahwa proses penanganan exception sudah berhasil.

Dalam bab ini kita telah membahas salah satu materi dasar OOP yang cukup penting, yakni **exception**. Hampir setiap kode program yang melibatkan object butuh penanganan error dalam bentuk exception.

Berikutnya, kita akan masuk ke bahasan tentang object bawaan PHP, yakni **DateTime Object**.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

=====

8. DateTime Object

PHP adalah bahasa pemrograman *multi paradigm*, yang artinya bisa ditulis menggunakan *procedural programming* serta *object oriented programming*. Oleh karena itu banyak fitur bawaan tersedia dalam bentuk function biasa dan juga dalam bentuk object.

Kali ini kita akan lihat praktik penerapan OOP di salah satu object bawaan PHP, yakni **DateTime Object**. Sesuai namanya, **DateTime** adalah object untuk mengolah tanggal dan waktu (**date** dan **time**). Object ini bisa dipakai untuk menghitung selisih waktu, men-format tampilan tanggal, dll.

Kita akan bahas beberapa fitur penting dari **DateTime Object**. Jika anda ingin mempelajari semua method dan property, bisa mengunjungi dokumentasi resmi di PHP Manual: [Date and Time Related Extensions](#).

Di buku **PHP Uncover** saya pernah membahas *Date and Time Function*, sekarang kita akan lihat penulisan versi objectnya. Dalam banyak hal, fitur yang tersedia dalam bentuk function juga tersedia dalam bentuk object.

8.1. DateTime Class

DateTime object berasal dari **DateTime class**. Object ini dipakai untuk menyimpan data tanggal dan waktu. Berikut cara pembuatannya:

```
$sekarang = new DateTime();
```

Dalam contoh di atas, variabel \$sekarang akan berisi object dari class **DateTime**. Jika class **DateTime** dipanggil tanpa argument seperti ini, yang disimpan adalah tanggal dan waktu sekarang (saat kode program dijalankan).

Untuk menampilkan tanggal dan waktu yang sudah disimpan, bisa menggunakan method **format()**:

```
01.datetime_object_basic.php
```

```
1 <?php
2 $sekarang = new DateTime();
3
4 echo $sekarang->format('d-m-Y'); // 09-04-2022
```

Date Time Object

Hasil perintah ini akan berbeda-beda tergantung kapan anda menjalankan kode tersebut.

Pola string yang dipakai oleh method `format()` sama seperti pada fungsi `date()` dalam pemrograman procedural PHP. Penjelasan lebih lengkap tentang method `format()` akan kita bahas sesaat lagi.

Kembali ke `DateTime`, object ini bisa menerima 2 argument opsional. Argument pertama berupa string yang berisi tanggal dan / atau waktu, serta argument kedua berupa `DateTimeZone` object.

Berikut format dasar penulisannya:

```
DateTime::__construct ([ string $time = "now" [, DateTimeZone $timezone = NULL ]] )
```

Penulisan di atas saya ambil dari PHP Manual. Tanda `DateTime::__construct` berarti ini adalah method constructor untuk proses instansiasi class `DateTime`. Tanda kurung siku "[" dan "]" menandakan bahwa kedua argument bersifat opsional.

Apabila pada saat proses instansiasi argument pertama tidak ditulis atau diisi `null`, maka akan diambil tanggal dan waktu sekarang. Ini sama artinya dengan `new DateTime('now')`.

Apabila argument kedua juga tidak ditulis, maka akan dipakai `timezone` default yang di set dari `php.ini`. Tentang `timezone` nantinya akan kita bahas secara terpisah.

Berikut percobaan lain dari pembuatan `DateTime` object:

02.datetime_object_argument.php

```
1 <?php
2 $sekarang = new DateTime(null);
3 echo $sekarang->format('d-m-Y');           // 09-04-2022
4
5 echo "<br>";
6
7 $sekarang = new DateTime('now');
8 echo $sekarang->format('d-m-Y');           // 09-04-2022
9
10 echo "<br>";
11
12 $hariKemerdekaan = new DateTime('17 Aug 1945');
13 echo $hariKemerdekaan->format('d-m-Y');   // 17-08-1945
14
15 echo "<br>";
16
17 $akhirTahun = new DateTime('2022/12/31');
18 echo $akhirTahun->format('d-m-Y');         // 31-12-2022
```

PHP mendukung beragam penulisan string sebagai inputan argument pertama `DateTime` object. Semua penulisan berikut akan menghasilkan tanggal 17 Agustus 1945:

DateTime Object

03.datetime_object_argument_2.php

```
1 <?php
2 $hariKemerdekaan_1 = new DateTime('17 Aug 1945');
3 $hariKemerdekaan_3 = new DateTime('17 August 1945');
4 $hariKemerdekaan_2 = new DateTime('17-08-1945');
5 $hariKemerdekaan_4 = new DateTime('17AUG1945');
6 $hariKemerdekaan_5 = new DateTime('1945-08-17');
7
8 echo $hariKemerdekaan_1->format('d-m-Y'); echo "<br>"; // 17-08-1945
9 echo $hariKemerdekaan_2->format('d-m-Y'); echo "<br>"; // 17-08-1945
10 echo $hariKemerdekaan_3->format('d-m-Y'); echo "<br>"; // 17-08-1945
11 echo $hariKemerdekaan_4->format('d-m-Y'); echo "<br>"; // 17-08-1945
12 echo $hariKemerdekaan_5->format('d-m-Y'); echo "<br>"; // 17-08-1945
```

Format input string ini sama dengan yang bisa diterima oleh fungsi `strtotime()` dalam pemrograman prosedural PHP. Daftar lengkapnya bisa anda lihat ke: [Date Formats](#) atau bisa baca kembali bab *Date and Time Function* di buku **PHP Uncover**.

Pada prakteknya, akan lebih mudah jika kita memakai cara penulisan standar, yakni tanggal dalam format `dd-mm-yyyy`.

`DateTime` object juga bisa diisi dengan tanggal, waktu, atau tanggal dan waktu sekaligus:

04.datetime_object_argument_time.php

```
1 <?php
2 $sekarang      = new DateTime();
3 $hariKemerdekaan = new DateTime('17 Aug 1945 11:30');
4 $nantiMalam    = new DateTime('10:30 PM');
5 $tahunBaru     = new DateTime('01-01-2022');
6 $akhirTahun    = new DateTime('31 Dec 2022 11:59:59');
7
8 echo $sekarang->format('d-m-Y, H:i:s');           echo "<br>";
9 echo $hariKemerdekaan->format('d-m-Y, H:i:s');     echo "<br>";
10 echo $nantiMalam->format('d-m-Y, H:i:s');        echo "<br>";
11 echo $tahunBaru->format('d-m-Y, H:i:s');          echo "<br>";
12 echo $akhirTahun->format('d-m-Y, H:i:s');         echo "<br>";
```

Hasil kode program:

```
09-04-2022, 06:40:33
17-08-1945, 11:30:00
09-04-2022, 22:30:00
01-01-2022, 00:00:00
31-12-2022, 11:59:59
```

Dari percobaan ini terlihat bahwa jika tanggal tidak diinput, itu dianggap tanggal hari ini. Namun jika waktu tidak diinput, akan dianggap pukul 00:00.

Format string penulisan waktu bisa dilihat ke [Time Formats](#), tapi biasanya kita cukup pakai penulisan umum, yakni *jam:menit:detik*.

Object hasil dari `DateTime` class juga bisa dibandingkan satu sama lain seperti contoh berikut:

04a.datetime_object_argument_time.php

```

1 <?php
2 $tgl1 = new DateTime('17 Aug 2022');
3 $tgl2 = new DateTime('17 Aug 1945');
4
5 var_dump($tgl1 < $tgl2);           echo "<br>";
6 var_dump($tgl1 > $tgl2);           echo "<br>";
7 var_dump($tgl1 === $tgl2);         echo "<br>";

```

Hasil kode program:

```

bool(false)
bool(true)
bool(false)

```

Dalam contoh ini variabel `$tgl1` berisi tanggal yang lebih baru (lebih besar) daripada `$tgl2`, sehingga hasil dari operasi perbandingan `$tgl1 > $tgl2` akan menghasilkan nilai **true**.

8.2. `DateTime::format()` Method

Class `DateTime` memiliki berbagai method yang salah satunya adalah method **`format()`**. Seperti yang sudah kita coba sebelumnya, method `format()` ini dipakai untuk menampilkan kembali informasi tanggal dan waktu yang tersimpan dalam `DateTime` object.

Dari dokumentasi PHP (PHP Manual), penulisan method dari sebuah class menggunakan tanda ' :: '. Ini bukan berarti method tersebut adalah sebuah *static method*, tapi hanya cara penulisan bahwa method tersebut adalah "kepunyaan" class tertentu.

Misalnya method `format()` milik `DateTime` class ditulis sebagai `DateTime::format()`. Cara penulisan seperti ini akan sering anda jumpai.

Method `format()` butuh 1 buah argument berupa format tanggal dan waktu dalam bentuk string. PHP menyediakan beragam bentuk format, apakah itu untuk tanggal saja, waktu saja atau tanggal dan waktu sekaligus.

Berikut contoh penggunaan dari method `format()`:

05.datetime_object_format.php

```

1 <?php
2 $sekarang = new DateTime();
3
4 echo $sekarang->format('d-m-Y');    echo "<br>"; // 09-04-2022
5 echo $sekarang->format('d/m/Y');    echo "<br>"; // 09/04/2022
6 echo $sekarang->format('D, d M Y'); echo "<br>"; // Sat, 09 Apr 2022
7 echo $sekarang->format('j F y');    echo "<br>"; // 9 April 22

```

```
8 echo $sekarang->format('d-m-Y, H:i:s'); echo "<br>"; // 09-04-2022, 06:41:28
```

Di baris 2 saya membuat instansiasi class `DateTime` tanpa argument, ini artinya tanggal dan waktu yang disimpan adalah tanggal dan waktu saat kode dijalankan.

Format string yang dipakai sebenarnya sama seperti fungsi `date()` dalam versi PHP prosedural. Berikut daftar karakter yang didukung beserta maknanya:

Format angka/nama hari:

- ◆ **d**: Angka hari dengan format 2 digit dan didahului angka nol: 01 hingga 31.
- ◆ **j**: Angka hari dengan format 2 digit dan tanpa angka nol: 1 hingga 31.
- ◆ **w**: Angka hari untuk satu minggu, 0 hingga 6.
- ◆ **z**: Angka hari untuk satu tahun, 0 hingga 365.
- ◆ **D**: Nama hari dengan singkatan 3 karakter bahasa inggris: Sun, Mon hingga Sat.
- ◆ **I**: Nama hari dalam bahasa inggris: Sunday, Monday hingga Saturday.
- ◆ **S**: Awalan 2 digit karakter dalam bahasa inggris untuk nama hari: st, nd, rd dan th. Ini biasanya digunakan dengan format "j".

Format angka/nama bulan:

- ◆ **m**: Angka bulan dengan format 2 digit dan didahului angka nol: 01 hingga 12.
- ◆ **n**: Angka bulan dengan format 2 digit dan tanpa angka nol: 1 hingga 12.
- ◆ **M**: Nama bulan dengan singkatan 3 karakter bahasa inggris: Jan, Feb hingga Dec.
- ◆ **F**: Nama bulan dalam bahasa inggris: January, February hingga December.
- ◆ **t**: Total jumlah hari dalam 1 bulan : 28 hingga 31.

Format angka/nama tahun:

- ◆ **L**: 1 jika tahun kabisat, 0 jika bukan tahun kabisat.
- ◆ **Y**: Angka tahun dengan 4 digit, seperti 1998, 2022 dan 2030
- ◆ **y**: Angka tahun dengan 2 digit, seperti 98, 02 dan 22

Format angka untuk waktu:

- ◆ **a**: am atau pm, dengan huruf kecil.
- ◆ **A**: AM atau PM, dengan huruf besar.
- ◆ **g**: Angka jam dalam format 12 jam, tanpa awalan nol: 1 hingga 12.
- ◆ **G**: Angka jam dalam format 24 jam, tanpa awalan nol: 0 hingga 23.
- ◆ **h**: Angka jam dalam format 12 jam, dengan awalan nol: 01 hingga 12.
- ◆ **H**: Angka jam dalam format 24 jam, dengan awalan nol: 01 hingga 23.
- ◆ **i**: Angka menit dengan awalan nol: 00 hingga 59.
- ◆ **s**: Angka detik dengan awalan nol: 00 hingga 59.

Versi lengkap dari format-format ini bisa dilihat ke [Date Function](#).

Contoh-contoh lain dari penggunaan format ini sudah pernah saya bahas di buku **PHP Uncover** (bab Date and Time Function).

8.3. DateTimeZone Class

DateTimeZone adalah class yang berisi data *timezone*. Object dari class **DateTimeZone** ini bisa diinput sebagai argument kedua pada proses instansiasi class **DateTime**. Fungsinya, untuk menentukan zona waktu yang dipakai oleh object **DateTime**.

DateTimeZone diperlukan karena zona waktu yang aktif dari bawaan PHP biasanya bukanlah zona waktu Indonesia. Sebagai contoh, dalam versi XAMPP yang saya pakai zona waktu default adalah **Europe/Berlin**. Ini bisa dilihat dari *phpinfo*:

date	
date/time support	enabled
timelib version	2021.11
"Olson" Timezone Database Version	2021.5
Timezone Database	internal
Default timezone	Europe/Berlin

Gambar: Default timezone bawaan XAMPP adalah Europe/Berlin

Efek dari zona waktu akan berpengaruh pada saat kita menampilkan waktu saat ini:

06.timezone.php

```
1 <?php
2 $sekarang = new DateTime();
3 echo $sekarang->format('d-m-Y, H:i:s'); // 09-04-2022, 06:56:33
```

Tampak tidak ada yang salah, dimana PHP menampilkan tanggal 09-04-2022 pukul 06:56:33.

Tapi kode program di atas sebenarnya saya jalankan pada pukul 11:56:33 WIB sesuai waktu di sistem komputer, yang artinya terdapat selisih -6 jam. Penyebabnya adalah karena PHP memakai zona waktu di kota **Berlin**.

Untuk mengatasi hal ini, kita bisa tambah **DateTimeZone** object sebagai argument kedua pada saat pembuatan **DateTime**.

Class **DateTimeZone** sendiri butuh sebuah argument bertipe string yang berisi kode *timezone*. Daftar string kode *timezone* ini bisa dilihat ke [List of Supported Timezones](#). Untuk Waktu Indonesia Barat (WIB) stringnya adalah **Asia/Jakarta**, sehingga cara pembuatan **DateTimeZone** object adalah sebagai berikut:

```
$zonaWIB = new DateTimeZone('Asia/Jakarta');
```

Kemudian, variabel **\$zonaWIB** ini kita input sebagai argument kedua saat proses instansiasi

DateTime Object

class DateTime:

07.timezone_wib.php

```
1 <?php
2 $zonaWIB = new DateTimeZone('Asia/Jakarta');
3 $sekarang = new DateTime('now', $zonaWIB);
4
5 echo $sekarang->format('d-m-Y, H:i:s'); // 09-04-2022, 11:59:04
```

Hasilnya, waktu yang tampil sudah sesuai dengan waktu di komputer yang saya pakai.

Argument pertama pada saat instansiasi class DateTime (baris 2) tidak boleh kosong karena kita perlu mengisi argument kedua dengan DateTimeZone object. Oleh karena itu argument pertama diinput string 'now' untuk tanggal dan waktu saat ini.

Alternatif penulisan lain yang lebih singkat adalah dengan langsung membuat DateTimeZone object pada saat instansiasi class DateTime. Berikut contohnya:

08.timezone_wib_wita_wit.php

```
1 <?php
2 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
3 echo "Waktu WIB: ".$sekarang->format('d-m-Y, H:i:s');
4
5 echo "<br>";
6
7 $sekarang = new DateTime(null, new DateTimeZone('Asia/Makassar'));
8 echo "Waktu WITA: ".$sekarang->format('d-m-Y, H:i:s');
9
10 echo "<br>";
11
12 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jayapura'));
13 echo "Waktu WIT: ".$sekarang->format('d-m-Y, H:i:s');
```

Hasil kode program:

```
Waktu WIB: 09-04-2022, 12:00:24
Waktu WITA: 09-04-2022, 13:00:24
Waktu WIT: 09-04-2022, 14:00:24
```

Di sini saya membuat 3 zona waktu yang ada di Indonesia. String inputan yang dipakai untuk pembuatan DateTimeZone object adalah Asia/Jakarta untuk waktu **WIB**, Asia/Makassar untuk waktu **WITA** dan Asia/Jayapura untuk waktu **WIT**.

Jika kita ingin mengatur *timezone* agar berlaku sepanjang kode program, bisa memakai fungsi *date_default_timezone_set()*:

09.date_default_timezone_set.php

```
1 <?php
2 date_default_timezone_set("Asia/Jakarta");
```

DateTime Object

```
3  
4 $sekarang = new DateTime();  
5 echo $sekarang->format('d-m-Y, H:i:s'); // 09-04-2022, 12:00:53
```

Dengan deklarasi `date_default_timezone_set("Asia/Jakarta")`, maka sepanjang kode program akan memakai zona waktu Asia/Jakarta secara default. Kita tidak perlu lagi menulis timezone pada saat pembuatan `DateTime` object.

Jika yang diinginkan adalah mengubah pengaturan `timezone` secara global (untuk seluruh file PHP), bisa dengan mengubah pengaturan `date.timezone` pada file `php.ini`. Mengenai caranya sudah pernah saya bahas di buku **PHP Uncover**.

Untuk melihat zona waktu yang tersimpan di dalam sebuah object `DateTime`, kita bisa menggunakan fungsi `var_dump()`:

10.var_dump_datetime.php

```
1 <?php  
2 $sekarang = new DateTime();  
3  
4 echo "<pre>";  
5 var_dump($sekarang);  
6 echo "</pre>";
```

Hasil kode program:

```
object(DateTime)#1 (3) {  
    ["date"]=>  
        string(26) "2022-04-09 07:01:17.894172"  
    ["timezone_type"]=>  
        int(3)  
    ["timezone"]=>  
        string(13) "Europe/Berlin"  
}
```

Terlihat di bagian akhir bahwa `DateTime` object `$sekarang` di set dengan timezone `Europe/Berlin`.

8.4. `DateTime::add()` Method dan `DateInterval` Class

`DateTime` object yang kita buat sebelumnya bisa diproses lebih lanjut seperti ditambah 2 minggu lagi atau bisa juga dipakai untuk mencari nama hari di 10 tahun mendatang. Untuk keperluan ini, tersedia method `add()` dari `DateTime` object. Method `add()` ini butuh argument dalam bentuk dari `DateInterval` object.

`DateInterval` adalah class khusus yang berisi durasi tanggal dan waktu. Class ini butuh argument bertipe string berisi jumlah durasi waktu atau interval yang ingin ditambah. Berikut

contoh penggunaannya:

11.datetime_add.php

```
1 <?php
2 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
3 $duaMinggu = new DateInterval('P2W');
4 $sekarang->add($duaMinggu);
5
6 echo $sekarang->format('j F Y'); // 23 April 2022
```

Di baris 2 saya mengisi variabel \$sekarang dengan `DateTime` object untuk tanggal sekarang, yakni tanggal pada saat kode dijalankan (09 April 2022).

Di baris 3 terdapat perintah untuk membuat `DateInterval` object dengan durasi 2 minggu. Caranya adalah dengan menulis 'P2W' sebagai argument pada saat proses instansiasi. Object `DateInterval` kemudian disimpan ke dalam variabel \$duaMinggu.

Perintah `$sekarang->add($duaMinggu)` di baris 4 akan menambah waktu yang tersimpan di dalam object \$sekarang sebanyak 2 minggu. Hasilnya, object \$sekarang akan berisi tanggal 23 April 2022, hasil penambahan 2 minggu dari tanggal 09 April 2022.

Argument pada saat pembuatan `DateInterval` berbentuk string dengan format tertentu. String ini harus diawali dengan huruf 'P' (singkatan dari *Period*), kemudian diikuti dengan pasangan angka dan inisial pola waktu.

Berikut inisial pola waktu untuk `DateInterval` object:

- ◆ **Y**: year (tahun)
- ◆ **M**: month (bulan)
- ◆ **D**: day (hari)
- ◆ **W**: weeks (minggu). Dikonversi menjadi hari secara internal sehingga tidak bisa dikombinasikan dengan pola waktu 'D'.
- ◆ **H**: hour (jam)
- ◆ **M**: minute (menit)
- ◆ **S**: second (detik)

Pola waktu ini bisa digabung satu sama lain dengan syarat secara berurutan dari waktu paling lama dari kiri. Selain itu antara tanggal dan waktu harus ditambah dengan karakter "T".

Sebagai contoh, pola string 'P300D' artinya periode 300 hari, pola string 'P3Y1M2W' artinya periode 3 tahun 1 bulan 2 minggu, pola string 'P1D4H' artinya adalah 1 hari 4 jam. Pola 'P3M3DT3H' artinya periode 3 bulan 3 hari dan 3 jam.

Berikut contoh praktik penggunaan pola-pola ini:

DateTime Object

12.dateinterval_object.php

```
1 <?php
2 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
3 $durasi = new DateInterval('P3Y1M2W');
4 $sekarang->add($durasi);
5
6 echo $sekarang->format('j F Y'); // 23 May 2025
7
8 echo "<br>";
9
10 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
11 $durasi = new DateInterval('P300D');
12 $sekarang->add($durasi);
13
14 echo $sekarang->format('j F Y'); // 3 February 2023
```

Alternatif penulisan yang lebih singkat meskipun menjadi sedikit susah dibaca, adalah dengan membuat DateInterval object langsung pada saat pemanggilan method add():

13.dateinterval_object_singkat.php

```
1 <?php
2 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
3 $sekarang->add(new DateInterval('P3Y1M2W'));
4 echo $sekarang->format('j F Y'); // 23 May 2025
5
6 echo "<br>";
7
8 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
9 $sekarang->add(new DateInterval('P300D'));
10 echo $sekarang->format('j F Y'); // 3 February 2023
```

Agar lebih ringkas lagi, kita bisa memanfaatkan method chaining seperti penulisan berikut:

14.dateinterval_object_singkat_2.php

```
1 <?php
2 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
3 echo $sekarang->add(new DateInterval('P3Y1M2W'))->format('j F Y');
4 // 23 May 2025
5
6 echo "<br>";
7
8 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
9 echo $sekarang->add(new DateInterval('P300D'))->format('j F Y');
10 // 3 February 2023
```

Perhatikan kode program di baris 3 dan 9, di sini saya menggabung banyak penulisan sekaligus: pembuatan object DateInterval, pemanggilan method add(), serta menampilkan hasilnya dengan method format().

Exercise

Sebagai latihan, bisakah anda menampilkan nama hari, tanggal dan waktu untuk 2 kondisi berikut:

- ◆ 1000 hari dari tanggal sekarang.
- ◆ 9 tahun 9 bulan 9 hari 9 jam 9 menit dan 9 detik sejak tanggal 17 Agustus 1945 pukul 10:30.

Untuk membuat ini kita butuh `DateTime` object, `DateInterval` object, method `DateTime::add()` serta `DateTime::format()`.

Silahkan coba anda rancang kode programnya.

Baik, berikut kode program yang saya gunakan:

```
1 <?php
2 $sekarang = new DateTime(null, new DateTimeZone('Asia/Jakarta'));
3 $durasi = new DateInterval('P1000D');
4 echo $sekarang->add($durasi)->format('l, j F Y H:i:s');
5
6 echo "<br>";
7
8 $hariKemerdekaan = new DateTime('17-08-1945 10:30',
9 new DateTimeZone('Asia/Jakarta'));
10 $durasi = new DateInterval('P9Y9M9DT9H9M9S');
11 echo $hariKemerdekaan->add($durasi)->format('l, j F Y H:i:s');
```

Hasil kode program:

```
Friday, 3 January 2025 12:05:16
Thursday, 26 May 1955 19:39:09
```

8.5. `DateTime::modify()`

Method `modify()` mirip seperti `add()`, yakni mengubah tanggal dan waktu yang tersimpan di dalam `DateTime` object. Namun method `modify()` bisa juga dipakai untuk proses pengurangan tanggal.

Method `modify()` butuh sebuah argument bertipe string. String inilah yang dipakai untuk menentukan durasi tanggal dan waktu yang ingin ditambah atau dikurangi. Format penulisannya bisa berbentuk bahasa inggris seperti "1 day", "1 week", "yesterday", "2 weeks ago", "2 months 5 days ago", dst.

Berikut contoh penggunaan dari method `modify()`:

Date Time Object

16.modify_method.php

```
1 <?php
2 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
3 $sekarang->modify('1 day');
4
5 echo $sekarang->format('j F Y'); // 10 April 2022
```

Di baris 2 saya membuat `DateTime` object untuk tanggal hari ini dan menyimpannya ke dalam variabel `$sekarang`.

Kemudian di baris 3 dipanggil method `modify()` dengan argument `'1 day'`, artinya saya ingin menambah tanggal yang tersimpan di dalam `DateTime` sebanyak 1 hari. Hasilnya, variabel `$sekarang` berisi tanggal `10 April 2022`.

Kita juga bisa menulis tanda plus " + " atau minus " - " ke dalam string seperti `modify("+1 month")` atau `modify("-1 month")`:

17.modify_method_min_plus.php

```
1 <?php
2 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
3 $sekarang->modify('+1 month');
4 echo $sekarang->format('j F Y');
5
6 echo "<br>";
7
8 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
9 $sekarang->modify('-1 month');
10 echo $sekarang->format('j F Y');
```

Hasil kode program:

```
9 May 2022
9 March 2022
```

Tanda plus " + " dipakai untuk menambah data tanggal dan waktu, sedangkan tanda minus " - " dipakai untuk mengurangi tanggal dan waktu. Pada saat saya menjalankan kode program di atas, tanggal saat ini adalah 09 April 2022, sehingga 1 bulan berikutnya ada di 9 May 2022, dan 1 bulan sebelumnya ada di 9 March 2022.

Format inputan argument untuk method `add()` cukup beragam, seperti yang terlihat dari tabel berikut:

Day-based Notations		
Format	Description	Examples
'yesterday'	Midnight of yesterday	"yesterday 14:00"
'midnight'	The time is set to 00:00:00	
'today'	The time is set to 00:00:00	
'now'	Now - this is simply ignored	
'noon'	The time is set to 12:00:00	"yesterday noon"
'tomorrow'	Midnight of tomorrow	
'back of' hour	15 minutes past the specified hour	"back of 7pm", "back of 15"
'front of' hour	15 minutes before the specified hour	"front of 5am", "front of 23"
'first day of'	Sets the day of the first of the current month. This phrase is best used together with a month name following it.	"first day of January 2008"
'last day of'	Sets the day to the last day of the current month. This phrase is best used together with a month name following it.	"last day of next month"
ordinal space dayname space 'of'	Calculates the x-th week day of the current month.	"first sat of July 2008"
'last' space dayname space 'of'	Calculates the last week day of the current month.	"last sat of July 2008"
number space? (unit 'week')	Handles relative time items where the value is a number.	"+5 weeks", "12 day", "-7 weekdays"
ordinal space unit	Handles relative time items where the value is text.	"fifth day", "second month"
'ago'	Negates all the values of previously found relative time items.	"2 days ago", "8 days ago 14:00", "2 months 5 days ago", "2 months ago 5 days", "2 days ago"
dayname	Moves to the next day of this name.	"Monday"
reltext space 'week'	Handles the special format "weekday + last/this/next week".	"Monday next week"

Gambar: Berbagai format inputan yang didukung method DateTime::modify()

Daftar lebih lengkap bisa anda lihat ke PHP Manual: [Supported Date and Time Formats](#).

Berikut contoh penggunaannya:

18.modify_method_example.php

```

1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $sekarang = new DateTime();
5 echo $sekarang->modify('+10 month +100 day ')->format('j F Y');
6
7 echo "<br>";
8
9 $merdeka = new DateTime('17-08-1945 10:30');
10 echo $merdeka->modify('-3 day -10 hour -10 minute')->format('D, j F Y H:i:s');
```

Hasil kode program:

```

20 May 2023
Tue, 14 August 1945 00:20:00
```

Di sini saya menggunakan teknik *method chaining*, dimana hasil dari method `modify()`

langsung disambung dengan pemanggilan method `format()`.

Di baris 4 – 5 saya ingin mencari tanggal untuk 10 bulan dan 100 hari ke depan. Karena sekarang tanggal 9 April 2022, maka hasilnya ada di 20 May 2023.

Pada baris 9 – 10 saya ingin menampilkan tanggal untuk 3 hari 10 jam dan 10 menit sebelum tanggal 17-08-1945 10:30. Hasilnya adalah `Tue, 14 August 1945 00:20:00`. Perhatikan penggunaan tanda minus " - " di setiap komponen agar method `modify()` melakukan proses pengurangan.

8.6. DateTime::diff()

Method `diff()` bisa dipakai untuk menghitung selisih tanggal dan waktu dari 2 buah `DateTime` object. Berikut contoh penggunaannya:

19.diff_method.php

```

1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $tanggal1 = new DateTime('12-08-2016');
5 $tanggal2 = new DateTime('25-06-2022');
6
7 $interval = $tanggal1->diff($tanggal2);
8 echo $interval->format('%a hari');      // 2143 hari

```

Di baris 4 dan 5 terdapat perintah untuk membuat 2 buah `DateTime` object dengan tanggal yang berbeda. Kemudian di baris 7 terdapat perintah `$tanggal1->diff($tanggal2)` yang akan mencari selisih antara `$tanggal2` dengan `$tanggal1`.

Hasil dari pemanggilan method `diff()` ini berupa `DateInterval` object yang saya simpan ke dalam variabel `$interval`. Pada baris 8, isi `DateInterval` selanjutnya di tampilkan dengan perintah `echo $interval->format('%a hari')`.

Ada beberapa hal yang bisa kita bahas dari contoh di atas. Pertama, method `diff()` ini melekat ke `DateTime` object. Kedua, variabel `$tanggal1` dan `$tanggal2` sama-sama berasal dari `DateTime` class. Sehingga sebenarnya terdapat 2 cara pemanggilan:

```

$tanggal1->diff($tanggal2);
$tanggal2->diff($tanggal1);

```

Karena yang ingin dicari adalah selisih tanggal, maka tidak masalah ingin memakai penulisan yang mana. Sebagai contoh, selisih antara angka 7 dan 4 adalah 3. Cara menghitungnya bisa dengan $7 - 4$ atau $4 - 7$. Nantinya kita juga bisa mencari apakah tanggal tersebut selisih lebih atau selisih kurang, yakni apakah $+3$ atau -3 .

Untuk menampilkan durasi tanggal yang tersimpan di dalam `DateInterval` object, yakni hasil

dari pemanggilan method `diff()`, harus menggunakan format tertentu. Berikut daftar format yang disediakan PHP:

The following characters are recognized in the <code>format</code> parameter string. Each format character must be prefixed by a percent sign (%).		
format character	Description	Example values
%	Literal %	%
Y	Years, numeric, at least 2 digits with leading 0	01, 03
y	Years, numeric	1, 3
M	Months, numeric, at least 2 digits with leading 0	01, 03, 12
m	Months, numeric	1, 3, 12
D	Days, numeric, at least 2 digits with leading 0	01, 03, 31
d	Days, numeric	1, 3, 31
a	Total number of days as a result of a <code>DateTime::diff()</code> or (<i>unknown</i>) otherwise	4, 18, 8123
H	Hours, numeric, at least 2 digits with leading 0	01, 03, 23
h	Hours, numeric	1, 3, 23
I	Minutes, numeric, at least 2 digits with leading 0	01, 03, 59
i	Minutes, numeric	1, 3, 59
S	Seconds, numeric, at least 2 digits with leading 0	01, 03, 57
s	Seconds, numeric	1, 3, 57
F	Microseconds, numeric, at least 6 digits with leading 0	007701, 052738, 428291
f	Microseconds, numeric	7701, 52738, 428291
R	Sign "-" when negative, "+" when positive	-, +
r	Sign "-" when negative, empty when positive	-,

Tabel: format string untuk tampilan DateInterval object

Penulisan format ini harus diawali dengan tanda persen '%' untuk setiap karakter. Sebagai contoh, untuk menampilkan jumlah selisih dalam satuan bulan dan hari, format penulisannya adalah '`%m bulan %d hari`'.

Khusus untuk format hari, tersedia dalam 3 buah pilihan: '`%a`', '`%d`' dan '`%D`'. Bedanya, '`%a`' akan menampilkan total seluruh hari, tidak terpengaruh dengan format lain. Sedangkan '`%d`' dan '`%D`' akan menampilkan hari setelah dikurangi dengan format tahun dan bulan. Berikut contoh praktiknya:

20.diff_method_2.php

```

1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $tanggal1 = new DateTime('12-08-2016');
5 $tanggal2 = new DateTime('25-06-2022');
6
7 $interval = $tanggal1->diff($tanggal2);
8
9 echo $interval->format('%a hari');
```

DateTime Object

```
10 echo "<br>";
11 echo $interval->format('%y tahun %m bulan %d hari');
12 echo "<br>";
13 echo $interval->format('%y tahun %m bulan');
```

Hasil kode program:

```
2143 hari
5 tahun 10 bulan 13 hari
5 tahun 10 bulan
```

Saya masih menggunakan tanggal yang sama seperti contoh sebelumnya, yakni selisih antara tanggal '12-08-2016' dengan '25-06-2022'. Selisih ini dihitung dengan perintah `$tanggal1->diff($tanggal2)` yang kemudian disimpan ke dalam variabel `$interval`.

Di baris 9, 11 dan 13 saya menampilkan isi dari variabel `$interval` menggunakan format yang berbeda-beda.

Kita juga bisa menambah tanda ' %R ' atau ' %r ' untuk menampilkan apakah tanggal ini selisih kurang atau selisih lebih:

21.diff_method_3.php

```
1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $tanggal1 = new DateTime('12-08-2016');
5 $tanggal2 = new DateTime('25-06-2022');
6
7 $interval = $tanggal1->diff($tanggal2);
8 echo $interval->format('%R %y tahun %m bulan %d hari');
9
10 echo "<br>";
11
12 $interval = $tanggal2->diff($tanggal1);
13 echo $interval->format('%R %y tahun %m bulan %d hari');
```

Hasil kode program:

```
+ 5 tahun 10 bulan 13 hari
- 5 tahun 10 bulan 13 hari
```

Perhitungan tanda ini didapat dari tanggal mana yang dipakai untuk proses pengurangan.

Exercise

Buatlah kode program untuk menghitung umur anda dengan ketelitian sampai detik.

Latihan ini sebenarnya cukup mudah, kita hanya perlu merangkai format tampilan

sampai ke detik:

```
1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $sekarang = new DateTime();
5 $tanggallahir = new DateTime('15-07-1998 20:45:52');
6
7 $format = '%y tahun %m bulan %d hari %h jam %i menit %s detik';
8 echo $sekarang->diff($tanggallahir)->format($format);
```

Hasil kode program:

```
23 tahun 8 bulan 24 hari 15 jam 30 menit 5 detik
```

Dengan asumsi bahwa 'Andi' lahir di tanggal 15-07-1998 20:45:52, maka sampai dengan kode program dijalankan, umur Andi sudah 23 tahun 8 bulan 24 hari 15 jam 30 menit 5 detik.

8.7. DateTimeImmutable Class

Jika anda membuka dokumentasi PHP tentang [DateTime](#), terdapat 2 buah variasi class: `DateTime` dan `DateTimeImmutable`. Kedua class ini sangat mirip, yakni sama-sama dipakai untuk menyimpan data tanggal dan waktu.

Selain itu keduanya juga memiliki method yang sama. Sebagai contoh, untuk menampilkan tanggal kita bisa memakai `DateTime::format()` dan `DateTimeImmutable::format()`, untuk mengubah data tanggal bisa menggunakan `DateTime::modify()` dan `DateTimeImmutable::modify()`.

Perbedaan antara `DateTime` dengan `DateTimeImmutable` adalah dari cara pemrosesan data tanggal yang disimpan.

Jika menggunakan `DateTime`, isi tanggal yang ada di dalam object akan berubah setiap kali sebuah method dijalankan, sedangkan untuk `DateTimeImmutable`, isi tanggal dalam object tersebut bersifat tetap dan tidak bisa diubah lagi setelah proses instansiasi. Dalam programming, sesuatu yang tidak bisa diubah ini dikenal juga dengan istilah **immutable**.

Agar pengertian ini bisa lebih dipahami, kita akan bahas contoh dengan contoh praktek. Pertama, saya akan tampilkan kembali cara penggunaan `DateTime` object:

23.datetime_object.php

```
1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $sekarang = new DateTime();
```

DateTime Object

```
5 $sekarang->add(new DateInterval('P2W'));
6 $sekarang->modify('-1 month');
7 echo $sekarang->format('j F Y');
```

Hasil kode program:

23 March 2022

Saya yakin anda sudah bisa memahami maksud dari kode di atas. Kode tersebut saya jalankan pada 9 April 2022 sehingga inilah tanggal yang diisi ke dalam variabel \$sekarang hasil dari proses instansiasi DateTime object.

Kemudian object \$sekarang ditambah 2 minggu menggunakan method add(new DateInterval('P2W')) serta dikurangi 1 bulan dengan method modify('-1 month'). Hasilnya, variabel \$sekarang berisi tanggal 23 March 2022.

Selanjutnya, mari kita lihat praktik dari DateTimeImmutable:

24.datetimeimmutable_object.php

```
1 <?php
2 date_default_timezone_set("Asia/Jakarta");
3
4 $sekarang = new DateTimeImmutable();
5 $sekarang->add(new DateInterval('P2W'));
6 $sekarang->modify('-1 month');
7 echo $sekarang->format('j F Y');
```

Hasil kode program:

9 April 2022

Perubahan dari contoh sebelumnya ada di baris 4. Variabel \$sekarang akan berisi object dari instansiasi class DateTimeImmutable. Sisa kode yang lain sama persis seperti sebelumnya. Namun hasilnya adalah 9 April 2022.

Kode program di atas tidak menghasilkan error, namun kenapa pemanggilan method add() dan modify() tidak berdampak apa-apa kepada object \$sekarang?

Alasannya karena data tanggal yang terdapat di dalam DateTimeImmutable object tidak akan berubah. Isi dari variabel \$sekarang akan terus bernilai 9 April 2022 dari hasil proses instansiasi di baris 4.

Jika kita menggunakan DateTimeImmutable object, pemanggilan method seperti add() dan modify() akan menghasilkan nilai baru berupa object. Object baru inilah yang akan menyimpan hasil pemanggilan method tersebut. Berikut penjelasannya:

25.datetimeimmutable_object_2.php

```
1 <?php
2 date_default_timezone_set("Asia/Jakarta");
```

DateTime Object

```
3  
4 $sekarang = new DateTimeImmutable();  
5 $hasil1 = $sekarang->add(new DateInterval('P2W'));  
6 echo $hasil1->format('j F Y');  
7  
8 echo "<br>";  
9  
10 $hasil2 = $sekarang->modify('-1 month');  
11 echo $hasil2->format('j F Y');
```

Hasil kode program:

```
23 April 2022  
9 March 2022
```

Perhatikan kode program di baris 5 dan 10, variabel \$hasil1 dan \$hasil2 dipakai untuk menampung hasil pemanggilan method add() dan modify() dari object \$sekarang. Kembali, object \$sekarang itu sendiri tidak berubah, tapi mengembalikan sebuah object baru.

Jadi mana yang lebih baik? `DateTime` atau `DateTimeImmutable`?

Jawabannya tergantung kepada keperluan kita, apakah nanti butuh data tanggal awal? Jika perlu maka pakai `DateTimeImmutable` karena akan terus menyimpan tanggal awal. Namun jika tanggal awal ini tidak perlu dan kita hanya butuh hasilnya saja, akan lebih praktis jika menggunakan `DateTime`.

Perbedaan cara kerja `DateTime` dengan `DateTimeImmutable` menjadi contoh ideal dari prinsip pemrograman object di PHP. Nanti kita akan lihat bahwa beberapa class bawaan PHP ada yang bersifat **mutable** (data di dalam object tersebut bisa di modifikasi), atau ada juga yang bersifat **immutable** (data di dalam object tidak bisa di modifikasi).

Dalam bab ini kita telah membahas cara pengolahan tanggal dan waktu (**Date and Time**) di PHP menggunakan versi object. PHP sendiri memiliki cukup banyak object bawaan yang bisa membantu kita untuk memecahkan masalah.

Berikutnya, kita masih membahas object bawaan PHP yang dipakai untuk berkomunikasi dengan database MySQL, yakni **mysqli object**.

9. Mysqli Object

Pemrosesan database menjadi salah satu fokus utama ketika membuat aplikasi web. Dari sekian banyak pilihan, MySQL masih menjadi aplikasi database yang paling banyak di pakai dengan PHP.

PHP menyediakan 3 cara untuk berkomunikasi dengan MySQL, yakni melalui **mysql extension**, **mysqli extension** serta **PDO extension**. Untuk *mysql extension* sebaiknya tidak dipakai lagi karena sudah berstatus *deprecated* dan juga tidak tersedia di PHP versi 7 ke atas. Kita disarankan untuk menggunakan *mysqli extension* dan PDO.

Mysqli extension hadir dalam 2 "varian rasa", yakni penulisan prosedural menggunakan function, serta penulisan dengan konsep pemrograman object.

Dalam bab ini kita akan fokus membahas cara penggunaan *mysqli extension* menggunakan konsep pemrograman object ini (OOP). Untuk **PDO** akan dibahas terpisah dalam bab berikutnya.

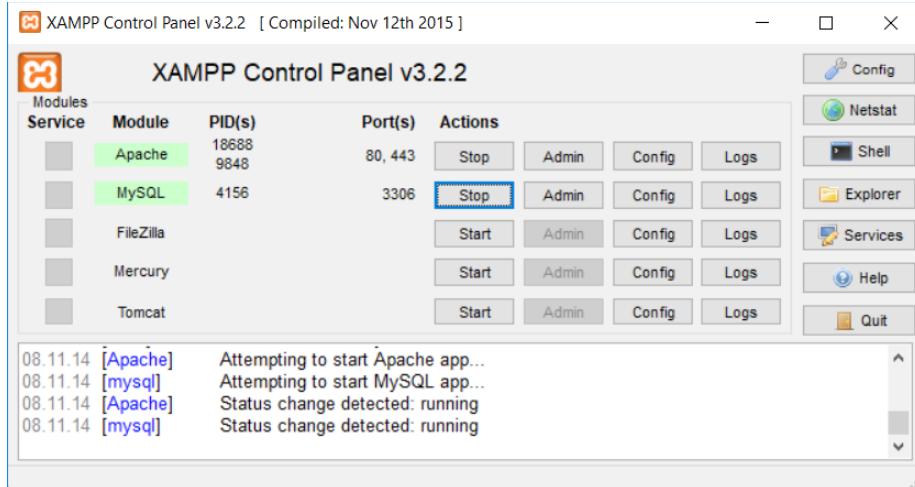
Agar pembahasan kita bisa lebih banyak ke materi *advanced*, saya berasumsi anda sudah pernah menulis kode program PHP menggunakan *mysqli extension* secara prosedural (misalnya dari buku **PHP Uncover**).

Selain itu dasar pemahaman query MySQL juga akan sangat membantu (misalnya dari buku **MySQL Uncover**). Saya tidak membahas lagi tentang maksud dari setiap perintah dasar SQL seperti SELECT, INSERT, UPDATE dan DELETE.

Sama seperti pembahasan tentang **DateTime** object, *mysqli extension* "rasa OOP" juga hadir dalam berbagai class. Setidaknya kita akan membahas 3 diantaranya, yakni **mysqli** class, **mysqli_result** class, dan **mysqli_stmt** class.

Secara singkat, **mysqli** class digunakan untuk membuat koneksi dengan database MySQL. Kemudian **mysqli_result** class dipakai untuk memproses hasil yang didapat dari query SELECT (menampilkan tabel). Serta **mysqli_stmt** class dipakai untuk memproses *prepared statement*. Setiap class ini memiliki berbagai property dan method yang akan kita bahas secara bertahap.

Sebelum mulai praktek, pastikan **MySQL Server** sudah aktif. Caranya, klik tombol **start** di kolom MySQL pada XAMPP Control Panel:



Gambar: Jalankan Web Server Apache dan Database Server MySQL di XAMPP Control Panel

Sepanjang pembahasan saya juga memakai user **default** bawaan XAMPP, yakni user **root** tanpa password. Jika anda mengubah password untuk user **root**, silahkan tukar contoh kode program nanti dengan password tersebut.

Aplikasi database bawaan XAMPP sebenarnya adalah **MariaDB**, bukan lagi MySQL. MariaDB sendiri merupakan "cloningan MySQL" yang sengaja dibuat oleh kalangan open-source sejak MySQL dibeli oleh perusahaan **Oracle**.

Dalam penggunaan umum, tidak ada perbedaan kode program PHP ketika mengakses MariaDB maupun MySQL. Namun agar lebih familiar, saya tetap memakai istilah MySQL meskipun pada praktiknya kita menggunakan MariaDB.

9.1. Mysqli Class

Mysqli class adalah class yang dipakai untuk mengelola koneksi antara PHP dengan MySQL database server. Dalam mysqli versi prosedural, koneksi ini dibuat menggunakan function **mysqli_connect()**.

Agar bisa dipakai, **mysqli class** harus di instansiasi menjadi **mysqli object**. Proses instansiasi ini sama seperti pembuatan object biasa, yakni menggunakan perintah **new**:

```
$nama_variabel = new mysqli([argument_1], [argument_2], [argument_3], ...)
```

Pada saat proses instansiasi, **mysqli object** bisa menerima 6 argument yang semuanya opsional. Berikut format dasar penulisan **mysqli object** yang saya ambil dari [PHP Manual](#):

```
mysqli::__construct (
[ string $host = ini_get("mysqli.default_host")
[, string $username = ini_get("mysqli.default_user")]
[, string $passwd = ini_get("mysqli.default_pw")]
[, string $dbname = ""
```

Mysqli Object

```
[, int $port = ini_get("mysqli.default_port")
[, string $socket = ini_get("mysqli.default_socket")]
]]]] ] )
```

Penulisan di atas memang sedikit "mengintimidasi". Kita bisa sederhanakan sebagai berikut:

```
$nama_variabel = new mysqli([$host], [$username], [$password], [$dbname], [$port],
[$socket])
```

Penjelasan argument:

- ◆ **\$host**: Diisi dengan alamat server atau IP address tempat MySQL server berada. Karena kita menjalankan MySQL Server di komputer yang sama dengan Web Server Apache, maka bisa diisi dengan "localhost" atau "127.0.0.1".
- ◆ **\$username**: Diisi dengan nama user MySQL Server, misalnya "root".
- ◆ **\$password**: Diisi dengan password dari user MySQL yang ada ditulis pada argument kedua.
- ◆ **\$dbname**: Diisi dengan database yang ingin dipakai.
- ◆ **\$port**: Diisi dengan nama port MySQL yang digunakan. Secara default, MySQL menggunakan port 3306.
- ◆ **\$socket**: Diisi dengan alamat file yang mengatur socket, yakni mekanisme cara komunikasi antara web server dengan database server.

Dari ke-6 argument ini, umumnya kita hanya perlu mengisi 3 atau 4 argument saja, yakni **\$host**, **\$username**, **\$password** dan **\$dbname**.

Setiap argument berada di dalam tanda kurung siku karena semuanya bersifat opsional (boleh tidak ditulis). Jika argument tersebut tidak diisi, PHP akan mencarinya di file konfigurasi **php.ini**.

Sebagai contoh, apabila **\$username** tidak ditulis, PHP akan mengambil nilai yang ada di pengaturan **mysqli.default_user** di file **php.ini**:

```
1174 ; Default socket name for local MySQL connects. If empty, uses the built-in
1175 ; MySQL defaults.
1176 ; http://php.net/mysqli.default-socket
1177 mysqli.default_socket=
1178
1179 ; Default host for mysql_connect() (doesn't apply in safe mode).
1180 ; http://php.net/mysqli.default-host
1181 mysqli.default_host=
1182
1183 ; Default user for mysql_connect() (doesn't apply in safe mode).
1184 ; http://php.net/mysqli.default-user
1185 mysqli.default_user=
```

Pengaturan **mysqli.default_user** di file **php.ini** yang masih kosong

Namun bawaan XAMPP, konfigurasi **mysqli.default_user** ini tidak berisi nama user apapun (kosong), sehingga kita harus mengisi argument ini pada saat pembuatan **mysqli** object.

Mysqli Object

Kembali ke cara pembuatan **mysqli object**, saya akan input argument untuk \$host, \$username, dan \$password:

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "");
3
4 echo "<pre>";
5 print_r($mysqli);
6 echo "</pre>";
```

Perintah di baris 2 artinya: buat **mysqli** object untuk berkomunikasi dengan MySQL Server yang berada di komputer "localhost", kemudian login sebagai user "root" dengan password "" (tanpa password).

Hasil instansiasi ini disimpan ke dalam variabel \$mysqli. Nama variabel ini boleh bebas dan tidak harus \$mysqli. Nama lain yang sering dipakai adalah \$link, \$conn, \$connect atau bisa juga ditulis sebagai \$koneksi. Nama variabel \$mysqli cukup sering di jumpai karena sekaligus menunjukkan bahwa ini adalah variabel yang berisi **mysqli** object.

Selanjutnya variabel \$mysqli saya akses menggunakan perintah print_r(\$mysqli) di baris 5 untuk melihat apa saja isi object ini. Berikut hasilnya:

```
1 mysqli Object
2 (
3     [affected_rows] => 0
4     [client_info] => mysqlnd 8.0.0
5     [client_version] => 80000
6     [connect_errno] => 0
7     [connect_error] =>
8     [errno] => 0
9     [error] =>
10    [error_list] => Array
11        (
12        )
13
14    [field_count] => 0
15    [host_info] => localhost via TCP/IP
16    [info] =>
17    [insert_id] => 0
18    [server_info] => 5.5.5-10.4.17-MariaDB
19    [server_version] => 100417
20    [sqlstate] => 00000
21    [protocol_version] => 10
22    [thread_id] => 9
23    [warning_count] => 0
24 )
```

Jika anda tidak mendapatkan hasil di atas atau tampil error, silahkan periksa kembali apakah MySQL Server sudah dijalankan atau apakah pernah mengubah password root MySQL.

Di baris 1 terlihat bahwa ini adalah sebuah **mysqli Object**. Kemudian di baris 2 – 24 terdapat puluhan property yang bisa kita akses dari **mysqli object**. Nantinya nilai property ini akan berubah setiap kali perintah MySQL di jalankan.

Property `affected_rows` di baris 3 berisi informasi mengenai jumlah baris tabel yang terdampak oleh sebuah perintah *query*. Saat ini nilainya 0 karena kita belum menjalankan perintah *query* apapun.

Property `client_info` dan `client_version` di baris 4-5 menunjukkan versi MySQL Client yang dipakai oleh PHP.

Property `connect_errno` dan `connect_error` di baris 6-7 berisi nomor dan pesan kesalahan. Keduanya akan berisi nilai jika koneksi ke MySQL server gagal di proses. Dalam contoh di atas isi property ini bernilai 0 dan string kosong "" karena koneksi tidak mengalami masalah.

Property `server_info` di baris 18 berisi informasi mengenai versi MySQL Server yang sedang terhubung. Karena alasan tertentu, MariaDB memberikan kode awal 5.5.5 sebelum versi server yang sebenarnya, yakni **MariaDB 10.4.17**.

Setiap property ini kita bisa akses secara individu dari **mysqli object**:

02.mysql_connect_property.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "");
3
4 echo $mysqli->affected_rows; echo "<br>";
5 echo $mysqli->client_info; echo "<br>";
6 echo $mysqli->connect_errno; echo "<br>";
7 echo $mysqli->server_info; echo "<br>";
```

Hasil kode program:

```
0
mysqlnd 8.0.0
0
5.5.5-10.4.17-MariaDB
```

Penanganan Error mysqli Object

Di antara property milik **mysqli object**, `connect_errno` dan `connect_error` sangat bermanfaat untuk memeriksa kesalahan yang terjadi seperti contoh berikut:

03.mysql_connect_error.php

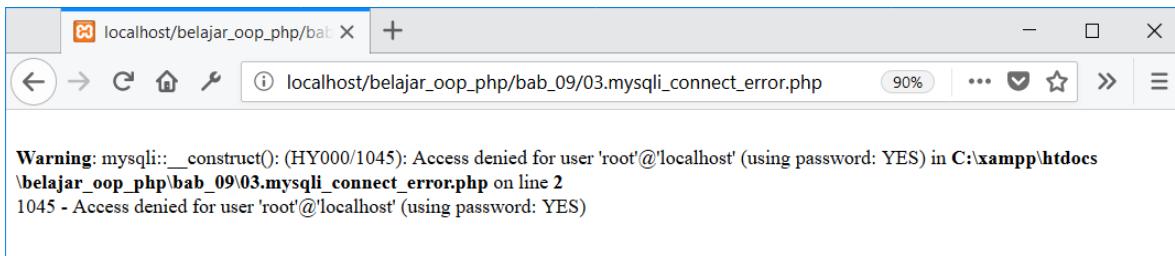
```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "x");
3 echo $mysqli->connect_errno, " - ", $mysqli->connect_error;
```

Pada saat proses instansiasi **mysqli object** di baris 2, saya sengaja menginput huruf 'x' di

Mysqli Object

argument ketiga, yakni inputan untuk password user root. Ini akan menyebabkan error karena password root seharusnya diisi dengan string kosong.

Berikut hasilnya:



Gambar: Error koneksi dengan MySQL Server

Terdapat 2 buah pesan error. Pertama adalah pesan warning di baris 1-2 yang langsung berasal dari PHP: "Warning: mysqli::__construct(): (HY000/1045): Access denied...". Dan yang kedua adalah pesan di baris 3 berupa: "1045 - Access denied for user 'root'@'localhost' (using password: YES)".

Pesan di baris 3 inilah yang berasal dari perintah echo \$mysqli->connect_errno, " - ", \$mysqli->connect_error.

Dalam situasi tertentu, bisa saja PHP tidak menampilkan pesan error bawaan. Ini umum terjadi di lingkungan web hosting atas alasan keamanan (web yang sudah online).

Teknik yang sering dipakai adalah menempatkan isi property connect_errno atau connect_error ke dalam sebuah kondisi **if**.

Jika tidak ada masalah (koneksi berhasil dilakukan), property connect_errno akan berisi nilai 0 dan property connect_error akan berisi string kosong. Namun jika ternyata koneksi bermasalah, kedua property ini akan berisi "sesuatu":

04.mysqli_connect_error_die.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "x");
3
4 if ($mysqli->connect_error) {
5     die('Koneksi bermasalah (' . $mysqli->connect_errno . ') '
6         . $mysqli->connect_error);
7 }
8
9 echo "Jalankan query MySQL...";
```

Di baris 4, terdapat pemeriksaan kondisi, yakni **if (\$mysqli->connect_error)**. Kondisi ini akan bernilai **true** jika property \$mysqli->connect_error berisi suatu string, namun akan bernilai **false** jika property \$mysqli->connect_error berisi string kosong.

Di dalam blok kondisi **if** terdapat fungsi **die()**. Gunanya adalah, jika ternyata koneksi ke MySQL bermasalah, tampilkan pesan error dan proses cukup berhenti sampai di sini.

Perintah di baris 9 dan seterusnya adalah perintah yang butuh koneksi ke MySQL. Daripada tampil pesan error lanjutan, proses cukup kita "cut" sampai di baris 5. Namun jika ternyata koneksi berjalan lancar, perintah `die()` akan dilompati dan proses lanjut ke baris 9.

Kita bisa membuat penulisan kode program yang lebih *elegant* daripada menggunakan fungsi `die()`, yakni menggunakan **Exception**. Idenya adalah, tempatkan kode program untuk membuat **mysqli** object di dalam struktur **try**. Jika terjadi error, lempar sebuah exception untuk ditangkap di bagian **catch**:

05.mysqli_connect_error_exception.php

```

1 <?php
2 try {
3     $mysqli = new mysqli("localhost", "root", "x");
4
5     if ($mysqli->connect_error) {
6         throw new Exception('Koneksi bermasalah (' . $mysqli->connect_errno . ') ' .
7             . $mysqli->connect_error);
8     }
9     echo "Jalankan query MySQL...";
10 }
11 catch (Exception $e) {
12     echo $e->getMessage();
13 }
```

Hasilnya sama seperti kode sebelumnya. Baris ke 9 serta seluruh kode program di dalam block **try** tidak akan dijalankan jika **Exception** sudah "dilempar". Nantinya kode program lain yang berhubungan dengan MySQL akan berada di dalam block **try** ini.

Dalam contoh di atas kita membuat exception secara manual. Sebenarnya, **mysqli** object juga bisa di set agar menampilkan pesan error langsung dalam bentuk exception, namun belum aktif. Secara default, seluruh pesan error dari **mysqli** object tampil sebagai **Warning**.

Untuk mengubah pesan error dari **mysqli** object menjadi exception, kita harus jalankan fungsi `mysqli_report(MYSQLI_REPORT_STRICT)` di awal kode program. Dengan adanya perintah ini, jika proses pembuatan mysqli object mengalami kendala, PHP akan melempar exception secara otomatis. Exception yang dilempar berasal dari class `mysqli_sql_exception`.

Berikut contoh prakteknya:

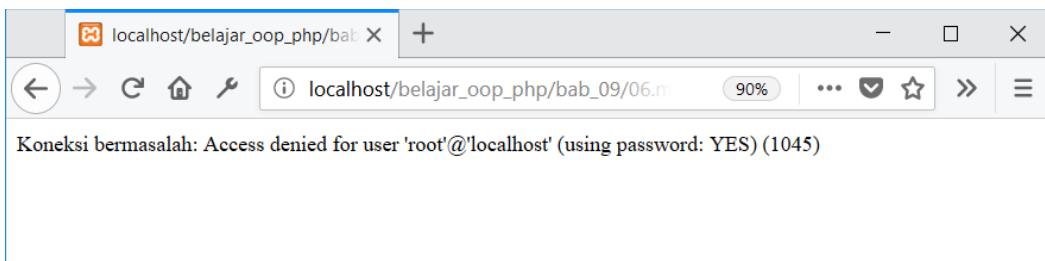
06.mysqli_connect_sql_exception.php

```

1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "x");
6     echo "Jalankan query MySQL...";
7 }
8 catch (mysqli_sql_exception $e) {
```

Mysqli Object

```
9     echo "Koneksi bermasalah: ".$e->getMessage(). " (".$e->getCode()." );  
10 }  
11
```



Gambar: Hasil pesan error dengan Exception mysqli object

Sekarang di dalam block **try** kita cukup membuat **mysqli** object saja. Jika terjadi sesuatu yang salah, **mysqli** object otomatis melempar **mysqli_sql_exception**. Exception ini akan ditangkap oleh block **catch** baris 8 yang kemudian menampilkan isi dari `$e->getMessage()` dan `$e->getCode()`.

Pesan error yang tampil sekarang juga hanya berasal dari exception. Tidak ada lagi pesan error "Warning: mysqli::__construct(): (HY000/1045): Access denied..." seperti yang tampil jika kita memproses error **mysqli** object menggunakan fungsi `die()`.

Dalam contoh di atas, perintah `echo` di baris ke 6 tidak akan diproses karena pembuatan **mysqli** object di baris 5 sudah langsung melempar sebuah exception.

Jika anda bingung tentang penjelasan tentang exception ini, boleh kembali sejenak ke bab tentang **Exception**.

Sebagai uji coba, mari kita tes dengan cara mematikan MySQL server dan jalankan kode di atas. Berikut hasilnya:



Gambar: Hasil tampilan pesan error karena MySQL Server dimatikan

Pesan error langsung di tangani oleh exception yang sudah kita rancang.

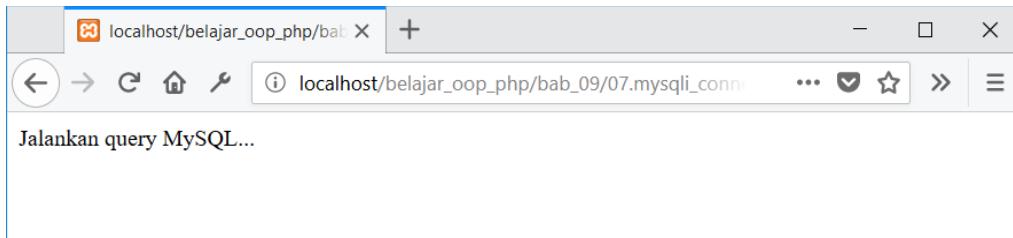
Sekarang aktifkan kembali MySQL Server, dan ubah argument untuk password menjadi kosong:

07.mysql_connect_sql_exception_2.php

```
1 <?php  
2 mysqli_report(MYSQLI_REPORT_STRICT);
```

Mysqli Object

```
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6     echo "Jalankan query MySQL..."; 
7 }
8 catch (mysqli_sql_exception $e) {
9     echo "Koneksi bermasalah: ". $e->getMessage(). " (".$e->getCode()." );"
10 }
```



Gambar: Tidak ada pesan error yang tampil

Karena password untuk `root` sudah benar (berupa string kosong), proses pembuatan `mysqli object` di baris 5 tidak lagi menghasilkan exception. Hasilnya perintah `echo` di baris 6 sukses dijalankan.

Teknik exception seperti ini akan kita pakai hingga akhir bab. Nantinya akan kita lihat bahwa selain membuat kode program menjadi lebih rapi dan terstruktur, struktur **try - catch** ini juga lebih mudah ditulis untuk penanganan error lain. Jika terjadi sesuatu yang salah, kita tinggal lempar exception baru dan otomatis sudah di tangani oleh block **catch** yang sudah ada.

Menutup mysqli Object

Jika proses koneksi dengan database MySQL sudah selesai, kita bisa tutup dengan method `mysqli::close()` seperti contoh berikut:

08.mysql_close.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "");
3
4 // Perintah query MySQL
5 // Perintah query MySQL
6 // Perintah query MySQL
7
8 $mysqli->close();
```

Method `mysqli::close()` sebenarnya tidak harus di tulis karena PHP otomatis juga akan menutup semua koneksi ke MySQL begitu file PHP selesai di proses. Namun tidak ada salahnya menutup koneksi jika memang tidak diperlukan lagi. Ini akan membebaskan ruang memory yang dipakai oleh PHP, terutama jika masih banyak kode PHP yang akan di eksekusi.

Karena kita sudah merancang kode program menggunakan **try - catch**, method `mysqli::close()` ini sangat pas ditulis pada block **finally**:

09.mysqli_close_finally.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6 }
7 catch (mysqli_sql_exception $e) {
8     echo "Koneksi bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
9 }
10 finally {
11     if (isset($mysqli)) {
12         $mysqli->close();
13     }
14 }
```

Dalam struktur **try - catch - finally**, block **finally** akan selalu dijalankan terlepas ada tidaknya exception. Namun saya harus membuat kondisi `if (isset($mysqli))` di baris 11 karena bisa saja **mysqli** object gagal dibuat. Jika ini terjadi, pemanggilan method `$mysqli->close()` malah akan menghasilkan pesan error.

Perintah `$mysqli->close()` hanya akan dijalankan jika variabel `$mysqli` berisi sesuatu. Atau dengan kata lain, method `$mysqli->close()` hanya perlu di tutup jika koneksi masih terbuka.

Membuat Database Dengan mysqli Object

Baik, proses koneksi antara PHP dengan database MySQL sudah aktif. Selanjutnya kita akan bahas cara menjalankan query MySQL menggunakan **mysqli** object.

Sebagai bahan praktek, saya ingin merancang sebuah mini project sederhana, yakni membuat database, tabel dan mengisinya secara langsung dari PHP.

Proses pembuatan database dan tabel ini sebenarnya lebih sering dibuat dari **phpmyadmin**. Namun dengan menulis kodennya secara langsung di PHP akan membawa keuntungan tersendiri. Nantinya kita memiliki sebuah file yang bisa dipakai untuk meng-generate database dan tabel secara otomatis.

Pengetahuan ini sangat berguna dalam pembuatan aplikasi komersil. Katakanlah anda sudah selesai merancang aplikasi **Sistem Informasi Sekolah**. Cukup repot jika setiap kali menginstall aplikasi, databasenya harus dibuat dulu dari phpmyadmin.

Akan lebih praktis jika kita menyediakan semacam file booting atau file generate yang ketika dijalankan, otomatis membuat semua database dan tabel. Proses ini memang butuh persiapan yang lebih sedikit lebih lama, tapi sangat praktis dalam jangka panjang.

Untuk praktek ini saya akan membuat kode program untuk meng-generate database **ilkoom** dan tabel **barang**. Kita akan buat database "**ilkoom**" terlebih dahulu:

10.mysql_query.php

```

1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6     $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
7     $mysqli->query($query);
8 }
9 catch (mysqli_sql_exception $e) {
10    echo "Koneksi bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
11 }
12 finally {
13     if (isset($mysqli)) {
14         $mysqli->close();
15     }
16 }
```

Karena kita menggunakan blok **try-catch-finally**, maka semua kode program di tempatkan dalam block **try**.

Setelah proses koneksi di baris 5, saya membuat variabel `$query` di baris 6. Variabel ini berisi perintah query MySQL untuk membuat database `ilkoom`. *Clausa* atau tambahan perintah `IF NOT EXISTS` berfungsi agar query tidak error seandainya database dengan nama `ilkoom` sudah ada sebelumnya.

Untuk menjalankan query MySQL, caranya adalah dengan mengakses method `mysqli::query()`. Method `mysqli::query()` butuh sebuah argument bertipe string yang berisi perintah query MySQL. Karena perintah ini sudah ada di dalam variabel `$query`, maka penulisannya menjadi `$mysqli->query($query)` seperti yang terlihat di baris 7.

Saya selalu sarankan agar query MySQL ditulis secara terpisah, bukan langsung diinput ke dalam perintah `mysqli::query()` seperti berikut:

```
$mysqli->query("CREATE DATABASE IF NOT EXISTS ilkoom");
```

Karena dengan memisahkannya menjadi sebuah variabel, query tersebut bisa di-echo untuk proses *debugging*:

```
$query = "CREATE DATABASE IF NOT EXISTS ilkoom";
echo $query;
```

Jika query yang dirancang cukup kompleks, kita bisa test hasil `echo $query` ini ke dalam phpmyadmin atau cmd Windows yang sedang mengakses MySQL Server.

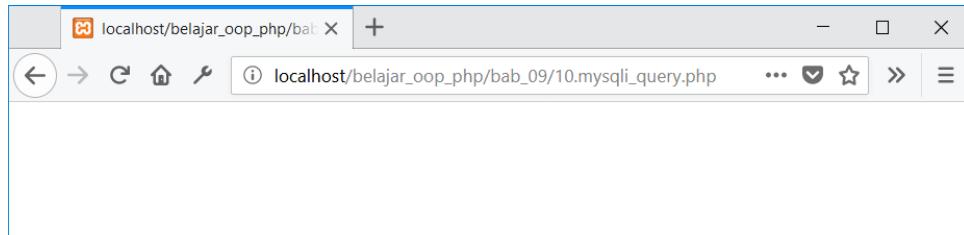
Dengan menjalankan kode di atas, maka database `ilkoom` sudah tersedia di dalam MySQL Server. Kode ini memang tidak menampilkan pesan apapun karena method `mysqli::query()` tidak secara otomatis memberitahu kita apakah query tersebut sukses atau gagal.

Mysqli Object

Sebagai bukti, silahkan tukar isi variabel \$query di baris 6 menjadi sebagai berikut:

```
$query = "CREATE DATABASE IF NOT EXISTS ilkoom";
```

Saya sengaja mengganti query "CREATE" menjadi "CREAT". Query ini tidak akan bisa diproses oleh MySQL dan seharusnya menghasilkan error. Namun ketika dijalankan, tidak tampil pesan apapun!



Gambar: Tidak ada pemberitahuan apakah query berhasil atau gagal

Di sini kita tidak tau apakah query yang ditulis sudah diproses oleh PHP atau belum. Dan apabila tidak bisa diproses, bagian mana yang salah juga tidak dijelaskan. Solusinya, kita bisa akses property **mysqli::error** dan **mysqli::errno**.

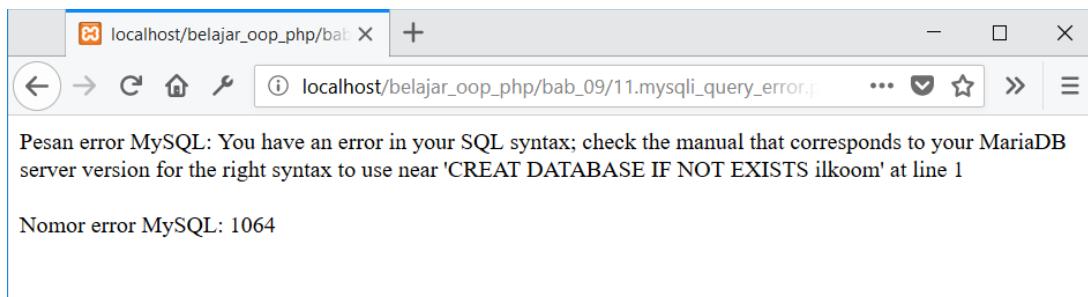
Ketika PHP memproses sebuah query MySQL (menjalankan method **mysqli::query()**), isi dari **mysqli** object otomatis di update oleh PHP. Jika query tersebut bermasalah, property **mysqli::error** dan **mysqli::errno** akan menyimpan informasi pesan error dan kode error yang berasal dari MySQL Server.

Berikut contoh kode programnya:

11.mysql_query_error.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6     $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
7     $mysqli->query($query);
8     echo "Pesan error MySQL: ".$mysqli->error;
9     echo "<br><br>";
10    echo "Nomor error MySQL: ".$mysqli->errno;
11 }
12 catch (mysqli_sql_exception $e) {
13     echo "Koneksi bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
14 }
15 finally {
16     if (isset($mysqli)) {
17         $mysqli->close();
18     }
19 }
```

Mysqli Object



Gambar: Pesan error karena query tidak bisa diproses MySQL

Kode program ini nyaris sama seperti sebelumnya. Yang berbeda adalah isi variabel \$query yang diubah menjadi "CREAT" serta pengaksesan property \$mysqli->error dan \$mysqli->errno di baris 8 dan 10. Hasilnya, tampil pesan error dari MySQL bahwa ada yang salah dari penulisan query dengan nomor error 1064.

Dengan demikian kita bisa memakai property \$mysqli->error sebagai patokan apakah sebuah query bisa dipahami oleh MySQL atau tidak. Lebih lanjut bisa ditulis sebagai berikut:

12.mysql_query_error_die.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6     $query = "CREAT DATABASE IF NOT EXISTS ilkoom";
7     $mysqli->query($query);
8     if ($mysqli->error){
9         die("Query bermasalah: ".$mysqli->error. " (".$mysqli->errno.")");
10    };
11 }
12 catch (mysqli_sql_exception $e) {
13     echo "Koneksi bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
14 }
15 finally {
16     if (isset($mysqli)) {
17         $mysqli->close();
18     }
19 }
```

Hasil kode program:

```
Query bermasalah: You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near 'CREAT
DATABASE IF NOT EXISTS ilkoom' at line 1 (1064)
```

Perhatikan baris 8 – 10, saya memeriksa isi dari \$mysqli->error. Jika property ini berisi sesuatu, maka kondisi if (\$mysqli->error) akan bernilai **true**. Akibatnya perintah die() akan dijalankan.

Saya memakai perintah die() dengan alasan yang sama pada saat pembuatan **mysqli** object,

Mysqli Object

yakni jika query ini gagal atau tidak dipahami oleh MySQL, maka stop sampai di situ.

Akan tetapi juga sama seperti sebelumnya, kode program yang menggunakan fungsi die() menjadi kandidat yang tepat untuk kita ubah menjadi sebuah **Exception**:

13.mysql_query_error_exception.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6     $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
7     $mysqli->query($query);
8     if ($mysqli->error){
9         throw new Exception($mysqli->error, $mysqli->errno);
10    };
11 }
12 catch (Exception $e) {
13     echo "Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
14 }
15 catch (mysqli_sql_exception $e) {
16     echo "Koneksi bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
17 }
18 finally {
19     if (isset($mysqli)) {
20         $mysqli->close();
21     }
22 }
```

Tidak seperti proses pembuatan mysqli object, PHP tidak memiliki mekanisme bawaan untuk menghasilkan Exception jika query MySQL gagal di proses. Oleh karena itu kita terpaksa melempar exception secara manual (baris 8-10). Exception ini nanti akan ditangkap oleh block catch di baris 12 – 14.

Di dalam PHP, semua exception berasal dari class **Exception**, termasuk **mysqli_sql_exception**. Jadi daripada membuat 2 buah block **catch**, saya menyederhanakannya jadi 1 block saja:

14.mysql_query_error_exception_2.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6     $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
7     $mysqli->query($query);
8     if ($mysqli->error){
9         throw new Exception($mysqli->error, $mysqli->errno);
10    };
11 }
12 catch (Exception $e) {
13     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." );
```

Mysqli Object

```
14 }
15 finally {
16     if (isset($mysqli)) {
17         $mysqli->close();
18     }
19 }
```

Hasil kode program:

Koneksi / Query bermasalah: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'CREATE DATABASE IF NOT EXISTS ilkoom' at line 1 (1064)

Block **catch** di baris 12 akan menangkap semua Exception, termasuk `mysqli_sql_exception` yang di-generate PHP pada saat pembuatan `mysqli` object.

Sebagai uji coba, jika password user root saya isi dengan huruf 'x', pesan error yang tampil adalah sebagai berikut:

Koneksi / Query bermasalah: Access denied for user 'root'@'localhost' (using password: YES) (1045)

Pesan error yang tampil memang sangat membantu pada saat pembuatan kode program (proses development). Namun jadi **sangat berbahaya** jika pesan ini tampil ketika aplikasi sudah rilis dan bisa diakses di internet.

Data sensitif seperti nama database, nama tabel atau nama kolom bisa dimanfaatkan untuk membobol aplikasi kita. Oleh karena itu selalu sembunyikan pesan error ketika aplikasi sudah selesai. Idealnya, di sistem yang sudah jadi, pesan error di alihkan ke file log di server, bukan ditampilkan langsung dalam web browser.

Dengan memproses pesan error menggunakan exception, akan memudahkan kita dalam mengatur hal ini karena semua pesan error diproses dalam satu tempat saja. Jika aplikasi sudah selesai dibuat (siap untuk di pakai user), cukup hapus perintah echo di block **catch** untuk diganti dengan perintah lain, misalnya menulis error ke file log.

Jika memakai cara biasa seperti perintah `die()`, pesan error jadi tersebar di berbagai tempat.

Selanjutnya, bagaimana menampilkan pesan jika query MySQL sukses dijalankan? Kita tinggal menambah block **else** saja:

15.mysqli_create_database.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
```

```

6
7 // Buat database "ilkoom" (jika belum ada)
8 $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
9 $mysqli->query($query);
10 if ($mysqli->error){
11     throw new Exception($mysqli->error, $mysqli->errno);
12 }
13 else {
14     echo "Database 'ilkoom' berhasil di buat / sudah tersedia <br>";
15 };
16 }
17 catch (Exception $e) {
18     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()."";
19 }
20 finally {
21     if (isset($mysqli)) {
22         $mysqli->close();
23     }
24 }
```

Hasil kode program:

Database 'ilkoom' berhasil di buat / sudah tersedia

Block **else** di baris 13 – 15 terhubung dengan kondisi `if ($mysqli->error)` di baris 10. Block **else** ini hanya akan di eksekusi jika `$mysqli->error` bernilai **false**. Dengan kata lain, kita bisa memastikan bahwa query berhasil di eksekusi jika `$mysqli->error` tidak berisi string apapun.

Sampai di sini kita sudah membahas 4 property **mysqli** object yang berhubungan dengan error, yakni:

- ◆ `mysqli::connect_error`
- ◆ `mysqli::connect_errno`
- ◆ `mysqli::error`
- ◆ `mysqli::error`

Dua property pertama yakni `mysqli::connect_error` dan `mysqli::connect_errno` dipakai untuk menampilkan pesan kesalahan pada saat **pembuatan koneksi ke MySQL** (pada saat pembuatan **mysqli** object).

Sedangkan dua property terakhir yakni `mysqli::error` dan `mysqli::error` dipakai untuk menampilkan pesan kesalahan pada saat **pemrosesan query**.

Memilih Database Dengan mysqli Object

Dari kode program sebelumnya kita telah berhasil membuat database **ilkoom**. Langkah berikutnya adalah memilih database ini sebagai database aktif dengan menjalankan method `mysqli::select_db()`. Method ini butuh sebuah argument berupa nama database yang akan

Mysqli Object

dipakai.

16.mysqli_select_db.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6
7     // Buat database "ilkoom" (jika belum ada)
8     $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
9     $mysqli->query($query);
10    if ($mysqli->error){
11        throw new Exception($mysqli->error, $mysqli->errno);
12    }
13    else {
14        echo "Database 'ilkoom' berhasil di buat / sudah tersedia <br>";
15    }
16
17    // Pilih database "ilkoom"
18    $mysqli->select_db("ilkoom");
19    if ($mysqli->error){
20        throw new Exception($mysqli->error, $mysqli->errno);
21    }
22    else {
23        echo "Database 'ilkoom' berhasil di pilih <br>";
24    }
25
26 }
27 catch (Exception $e) {
28     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
29 }
30 finally {
31     if (isset($mysqli)) {
32         $mysqli->close();
33     }
34 }
```

Hasil kode program:

```
Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
```

Proses pemilihan database ilkoom ada di baris 18 – 24. Teknik yang saya pakai sama seperti sebelumnya, yakni jalankan perintah \$mysqli->select_db("ilkoom"), kemudian periksa apakah terjadi error dengan cara melihat isi dari \$mysqli->error.

Membuat Tabel Dengan mysqli Object

Untuk membuat tabel, kita kembali menggunakan method mysqli::query(), namun query yang dijalankan adalah CREATE TABLE. Berikut kode programnya:

Mysqli Object

17.mysql_create_table.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6
7     // Kode program untuk pembuatan database ilkoom...
8     // Kode program untuk pemilihan database ilkoom...
9     // ...
10
11    // Buat tabel "barang"
12    $query = "CREATE TABLE barang (
13        id_barang INT PRIMARY KEY AUTO_INCREMENT,
14        nama_barang VARCHAR(50),
15        jumlah_barang INT,
16        harga_barang DEC,
17        tanggal_update TIMESTAMP
18    )";
19    $mysqli->query($query);
20    if ($mysqli->error){
21        throw new Exception($mysqli->error, $mysqli->errno);
22    }
23    else {
24        echo "Tabel 'barang' berhasil di buat <br>";
25    };
26
27 }
28 catch (Exception $e) {
29     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
30 }
31 finally {
32     if (isset($mysqli)) {
33         $mysqli->close();
34     }
35 }
```

Hasil kode program:

```
Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
Tabel 'barang' berhasil di buat
```

Untuk menghemat tempat, kode program yang dipakai untuk membuat dan memilih database **ilkoom** tidak lagi saya tulis. Jika ingin melihat versi lengkapnya, bisa membuka file **belajar_oop_php.zip**. Ini juga berlaku untuk file-file kita selanjutnya.

Query yang saya pakai untuk membuat tabel ada di baris 12 – 25. Kembali anda bisa melihat teknik penulisan yang saya pakai, yakni:

1. Rancang query dan simpan ke dalam variabel \$query.

2. Jalankan method `$mysqli->query($query)`.
3. Cek apakah ada error dengan kondisi `if ($mysqli->error)`.

Untuk tabel **barang** saya rancang dengan 5 kolom:

- ◆ `id_barang`
- ◆ `nama_barang`
- ◆ `jumlah_barang`
- ◆ `harga_barang`
- ◆ `tanggal_update`

Khusus untuk kolom `id_barang` di set sebagai PRIMARY KEY AUTO_INCREMENT sehingga MySQL akan meng-generate nomor id secara otomatis. Sedangkan kolom `tanggal_update` dipakai untuk menampung data tanggal TIMESTAMP MySQL, yakni dalam bentuk: `yyyy-mm-dd hh:ii:ss`.

Kode program yang kita rancang ini sebenarnya saya siapkan sebagai file "reset tabel barang". Maksudnya, ketika kita "berkreasi" dengan berbagai query seperti mengedit data, menghapus baris, dst, tabel barang ini sudah tidak "asli" lagi.

Saya ingin dengan menjalankan sebuah file PHP, data di tabel barang kembali ke bentuk awal. Namun jika kode di atas dijalankan 2 kali, hasilnya sebagai berikut:

```
Database 'ilkoom' berhasil dibuat / sudah tersedia
Database 'ilkoom' berhasil dipilih
Koneksi / Query bermasalah: Table 'barang' already exists (1050)
```

Keterangan di baris 3 adalah pesan error dari MySQL. Ini terjadi karena kita mencoba menjalankan query "CREATE TABLE barang" sekali lagi, padahal tabel barang sudah tersedia.

Sebagai solusi, saya akan menjalankan query "DROP TABLE IF EXISTS barang" sebelum query pembuatan tabel barang:

```
18.mysql_drop_table_if_exist.php

1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6
7     // Kode program untuk pembuatan database ilkoom...
8     // Kode program untuk pemilihan database ilkoom...
9     // ...
10
11    // Hapus tabel "barang" (jika ada)
12    $query = "DROP TABLE IF EXISTS barang";
13    $mysqli->query($query);
14    if ($mysqli->error){
15        throw new Exception($mysqli->error, $mysqli->errno);
16    }
}
```

Mysqli Object

```
17
18 // Buat tabel "barang"
19 $query = "CREATE TABLE barang (
20     id_barang INT PRIMARY KEY AUTO_INCREMENT,
21     nama_barang VARCHAR(50),
22     jumlah_barang INT,
23     harga_barang DEC,
24     tanggal_update TIMESTAMP
25 )";
26 $mysqli->query($query);
27 if ($mysqli->error){
28     throw new Exception($mysqli->error, $mysqli->errno);
29 }
30 else {
31     echo "Tabel 'barang' berhasil di buat <br>";
32 };
33
34 }
35 catch (Exception $e) {
36     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
37 }
38 finally {
39     if (isset($mysqli)) {
40         $mysqli->close();
41     }
42 }
```

Hasil kode program:

```
Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
Tabel 'barang' berhasil di buat
```

Di baris 12 – 16 saya menjalankan query "DROP TABLE IF EXISTS barang". Query ini akan memerintahkan MySQL Server untuk menghapus tabel **barang** jika sudah ada sebelumnya. Dengan demikian, setiap kali file ini dijalankan, tabel **barang** akan selalu di buat ulang.

Mengisi Tabel Dengan mysqli Object

Setelah tabel barang selesai dibuat, saatnya kita isi dengan query **INSERT**:

19.mysql_insert_table.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6
7     // Kode program untuk pembuatan database ilkoom...
8     // Kode program untuk pemilihan database ilkoom...
9     // Kode program untuk pembuatan tabel barang...
10    // ...
```

Mysqli Object

```
11 // Isi tabel "barang"
12 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
13 $timestamp = $sekarang->format("Y-m-d H:i:s");
14
15 $query = "INSERT INTO barang
16     (nama_barang, jumlah_barang, harga_barang, tanggal_update) VALUES
17     ('TV Samsung 43NU7090 4K',5,5399000,'$timestamp'),
18     ('Kulkas LG GC-A432HLHU',10,7600000,'$timestamp'),
19     ('Laptop ASUS ROG GL503GE',7,16200000,'$timestamp'),
20     ('Printer Epson L220',14,2099000,'$timestamp'),
21     ('Smartphone Xiaomi Pocophone F1',25,4750000,'$timestamp')
22     ;";
23 $mysqli->query($query);
24 if ($mysqli->error){
25     throw new Exception($mysqli->error, $mysqli->errno);
26 }
27 else {
28     echo "Tabel 'barang' berhasil di isi <br>";
29 };
30
31 }
32 catch (Exception $e) {
33     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
34 }
35
36 finally {
37     if (isset($mysqli)) {
38         $mysqli->close();
39     }
40 }
```

Hasil kode program:

```
Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
Tabel 'barang' berhasil di buat
Tabel 'barang' berhasil di isi
```

Sebelum query **INSERT**, di baris 13 – 14 saya menyiapkan data tanggal saat ini. Caranya dengan membuat **DateTime** object, lalu di format sebagai "Y-m-d H:i:s". Jika sekarang adalah tanggal 04-01-2019 pukul 14:18:25 maka isi variabel **\$timestamp** adalah 2019-01-04 14:18:25. Format ini akan dipakai untuk tipe data **TIMESTAMP** pada kolom **tanggal_update**.

Proses pengisian tabel, yakni query **INSERT** berada di baris 16 – 30. Di sini saya mengisi langsung 5 baris data ke dalam tabel **barang**. Query yang dipakai tidak terlalu rumit, yakni menggunakan format:

```
INSERT... (nama_kolom1, nama_kolom2,...) VALUES (data1, data2, ... )
```

Khusus untuk kolom **id_barang** tidak perlu diinput karena kolom tersebut di set sebagai **AUTO INCREMENT**.

Mysqli Object

Query insert akan mengubah "sesuatu" ke dalam tabel. Informasi mengenai jumlah tabel yang diubah akibat sebuah query bisa diakses dari property `mysql::affected_rows`. Sehingga daripada membuat pesan:

```
echo "Tabel 'barang' berhasil di isi <br>";
```

Akan lebih informatif jika ditulis menjadi:

```
echo "Tabel 'barang' berhasil di isi ".$mysqli->affected_rows." baris data <br>";
```

Sampai di sini proses pembuatan database dan tabel sudah selesai. Berikut kode program lengkapnya:

20.mysqli_generate.php

```
1  <?php
2  mysqli_report(MYSQLI_REPORT_STRICT);
3
4  try {
5      $mysqli = new mysqli("localhost", "root", "");
6
7      // Buat database "ilkoom" (jika belum ada)
8      $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
9      $mysqli->query($query);
10     if ($mysqli->error){
11         throw new Exception($mysqli->error, $mysqli->errno);
12     }
13     else {
14         echo "Database 'ilkoom' berhasil di buat / sudah tersedia <br>";
15     }
16
17     // Pilih database "ilkoom"
18     $mysqli->select_db("ilkoom");
19     if ($mysqli->error){
20         throw new Exception($mysqli->error, $mysqli->errno);
21     }
22     else {
23         echo "Database 'ilkoom' berhasil di pilih <br>";
24     }
25
26     // Hapus tabel "barang" (jika ada)
27     $query = "DROP TABLE IF EXISTS barang";
28     $mysqli->query($query);
29     if ($mysqli->error){
30         throw new Exception($mysqli->error, $mysqli->errno);
31     }
32
33     // Buat tabel "barang"
34     $query = "CREATE TABLE barang (
35             id_barang INT PRIMARY KEY AUTO_INCREMENT,
36             nama_barang VARCHAR(50),
37             jumlah_barang INT,
38             harga_barang DEC,
```

Mysqli Object

```
39         tanggal_update TIMESTAMP
40         );
41     $mysqli->query($query);
42     if ($mysqli->error){
43         throw new Exception($mysqli->error, $mysqli->errno);
44     }
45     else {
46         echo "Tabel 'barang' berhasil di buat <br>";
47     };
48
49 // Isi tabel "barang"
50 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
51 $timestamp = $sekarang->format("Y-m-d H:i:s");
52
53 $query = "INSERT INTO barang
54     (nama_barang, jumlah_barang, harga_barang, tanggal_update) VALUES
55     ('TV Samsung 43NU7090 4K',5,5399000,'$timestamp'),
56     ('Kulkas LG GC-A432HLHU',10,7600000,'$timestamp'),
57     ('Laptop ASUS ROG GL503GE',7,16200000,'$timestamp'),
58     ('Printer Epson L220',14,2099000,'$timestamp'),
59     ('Smartphone Xiaomi Pocophone F1',25,4750000,'$timestamp')
60     ;";
61 $mysqli->query($query);
62 if ($mysqli->error){
63     throw new Exception($mysqli->error, $mysqli->errno);
64 }
65 else {
66     echo "Tabel 'barang' berhasil di isi ".$mysqli->affected_rows."
67         baris data <br>";
68 };
69 }
70 catch (Exception $e) {
71     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
72 }
73 finally {
74     if (isset($mysqli)) {
75         $mysqli->close();
76     }
77 }
```

Hasil kode program:

```
Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
Tabel 'barang' berhasil di buat
Tabel 'barang' berhasil di isi 5 baris data
```

File 20.mysql_generate.php ini akan menjadi file "master reset", dimana setiap kali dijalankan, tabel **barang** otomatis di-generate ulang dan di reset dengan data yang sama seperti di atas.

Terakhir, anda bisa buka **phpmyadmin** untuk memeriksa apakah tabel barang ini memang sudah ada atau belum:

	<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Sort by key:	None	
		+ Options					
	← T →		id_barang	nama_barang	jumlah_barang	harga_barang	tanggal_update
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	TV Samsung 43NU7090 4K	5	5399000 2019-01-04 15:12:36
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	Kulkas LG GC-A432HLHU	10	7600000 2019-01-04 15:12:36
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	Laptop ASUS ROG GL503GE	7	16200000 2019-01-04 15:12:36
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	Printer Epson L220	14	2099000 2019-01-04 15:12:36
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	5	Smartphone Xiaomi Pocophone F1	25	4750000 2019-01-04 15:12:36

Isi tabel barang dilihat dari phpmyadmin

9.2. Mysqli_result Class

Mysqli_result adalah class khusus yang dipakai untuk proses menampilkan data tabel, yakni mengolah hasil dari query **SELECT**. Dibandingkan pembuatan tabel, proses menampilkan data tabel lebih sering kita pakai. **Mysqli_result** memiliki cukup banyak method yang akan kita bahas secara detail.

Mysqli_result object didapat dari hasil pemanggilan method **mysqli::query()**, seperti contoh berikut:

21.mysql_result_object.php

```

1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "", "ilkoom");
6
7     // Ambil semua data di tabel barang
8     $query = "SELECT * FROM barang";
9     $result = $mysqli->query($query);
10    if ($mysqli->error){
11        throw new Exception($mysqli->error, $mysqli->errno);
12    }
13    else {
14        echo "<pre>";
15        print_r($result);
16        echo "</pre>";
17        $result->free();
18    };
19 }
20 catch (Exception $e) {
21     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
22 }
```

Mysqli Object

```
23 finally {
24     if (isset($mysqli)) {
25         $mysqli->close();
26     }
27 }
```

Hasil kode program:

```
mysqli_result Object
(
    [current_field] => 0
    [field_count] => 5
    [lengths] =>
    [num_rows] => 5
    [type] => 0
)
```

Dalam program ini, perintah pembuatan **mysqli** object di baris 5 memakai 4 argument, yakni alamat komputer "localhost", user "root", password root berupa string kosong "", serta nama database "ilkoom".

Dengan menambahkan argument ke-4 berupa nama database, proses koneksi langsung memilih database **ilkoom** sebagai database default. Kita tidak perlu lagi menjalankan method `$mysqli->select_db("ilkom")` seperti sebelumnya.

Variabel `$query` di baris 8 saya isi dengan string "SELECT * FROM barang". Ini adalah perintah query MySQL untuk menampilkan seluruh isi tabel **barang**.

Di baris 9 terdapat perintah `$result = $mysqli->query($query)`. Ini merupakan proses pembuatan **mysqli_result** object. **Mysqli_result** object kemudian ditampung ke dalam variabel `$result`.

Nama variabel yang dipakai untuk menampung hasil `$mysqli->query($query)` ini boleh bebas, namun banyak programmer yang menggunakan nama `$result` agar mencerminkan kalau ini adalah hasil (`result`) dari proses query MySQL. Selain itu karena isinya berupa **mysqli_result** object.

Di baris 10 – 12 terdapat proses pemeriksaan `$mysqli->error` seperti sebelumnya. Jika tidak ada masalah, block **else** akan dijalankan.

Dalam block **else** terdapat perintah `print_r($result)` di baris 15. Ini saya tulis agar kita bisa melihat property apa saja yang tersedia dari variabel `$result`. Dari hasil tampilan kode program terlihat bahwa **mysqli_result** object memiliki 5 property: `current_field`, `field_count`, `lengths`, `num_rows` dan `type`.

Dari kelima property ini, yang akan sering kita akses adalah `num_rows` dan `field_count`. Property `mysqli_result::num_rows` berisi informasi total jumlah baris hasil query `SELECT` (jumlah record). Sedangkan property `mysqli_result::field_count` berisi total kolom tabel hasil dari query `SELECT`.

Dalam contoh ini property `num_rows` berisi angka 5 karena terdapat 5 record di dalam tabel `barang`, Begitu juga dengan property `field_count` yang berisi angka 5 karena tabel `barang` memiliki 5 kolom.

Di baris 17 terdapat kode `$result->free()`. Ini adalah perintah untuk menghapus memory yang dipakai oleh **mysqli_result** object. Sama seperti method `$mysqli->close()`, method `$result->free()` juga tidak harus dijalankan untuk setiap **mysqli_result** object, karena PHP otomatis akan mengosongkan ruang memory begitu halaman selesai di proses.

Namun jika kita menjalankan banyak query dalam 1 file PHP, method `$result->free()` akan memperkecil kebutuhan memory, terlebih jika query `SELECT` mengambil banyak data dari MySQL Server.

Kembali ke property **mysqli_result** object, dalam contoh berikut saya ingin mengakses property `num_rows` dan `field_count` secara langsung:

Untuk mempersingkat kode program, mulai dari contoh ini dan seterusnya saya tidak lagi menulis kode untuk penanganan error dan exception. Namun dalam kode program "asli", sebaiknya tetap tulis kode tersebut.

22.mysql_result_num_rows.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Ambil data dari tabel barang
5 $query = "SELECT * FROM barang WHERE id_barang = 1";
6 $result = $mysqli->query($query);
7
8 // Tampilkan jumlah baris dan kolom
9 echo "Terdapat ".$result->field_count." kolom & ".$result->num_rows." baris";
10 $result->free();
```

Hasil kode program:

Terdapat 5 kolom & 1 baris

Di contoh ini query yang dijalankan adalah `"SELECT * FROM barang WHERE id_barang = 1"`. Clusa `WHERE id_barang = 1` membatasi data yang diambil dari tabel barang hanya 1 baris saja. Ini terlihat dari hasil perintah `echo` di baris 7, dimana property `$result->num_rows` menghasilkan nilai 1. Sedangkan untuk property `$result->field_count` tetap berisi angka 5 karena data yang diambil terdiri dari 5 kolom.

Isi property `$result->num_rows` ini sering dibuat sebagai patokan dari ada tidaknya sebuah record di dalam tabel, seperti contoh berikut:

23.mysql_result_num_rows_if.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan data dari tabel barang
5 $query = "SELECT * FROM barang WHERE id_barang = 100";
6 $result = $mysqli->query($query);
7
8 if ($result->num_rows === 0) {
9     echo "Data tidak ditemukan";
10 }
11 else {
12     echo "Data tersedia";
13 }
14
15 $result->free();
```

Hasil kode program:

Data tidak ditemukan

Kondisi WHERE yang saya pakai dalam query SELECT adalah "id_barang = 100". Tentu saja data ini tidak tersedia dalam tabel barang karena id_barang paling besar cuma 5. Hasilnya, property \$result->num_rows akan menghasilkan nilai 0.

Inilah yang kemudian dipakai untuk block **if - else** di baris 8 – 13. Yakni jika kondisi `if($result->num_rows === 0)` menghasilkan nilai **true**, bisa dipastikan data tidak tersedia di dalam tabel.

Menampilkan Tabel Dengan mysqli_result Object

Dalam bahasan sebelumnya kita telah lihat bahwa **mysqli_result** object dipakai untuk menampung hasil dari perintah query MySQL, terutama query **SELECT**.

Data yang tersimpan di dalam **mysqli_result** object bisa ditampilkan dalam 4 bentuk:

1. Normal Array (dengan index berupa angka numeric)
2. Associative array (dengan index berupa nama kolom)
3. Normal Array atau associative array (dengan index berupa angka numeric atau nama kolom)
4. Object (dengan property berupa nama kolom)

Jika anda sudah pernah menggunakan mysqli versi procedural, ketiga metode di atas tidak lain sebutan untuk fungsi `mysqli_fetch_row()`, `mysqli_fetch_assoc()`, dan `mysqli_fetch_object()`.

Untuk mengambil data dari **mysqli_result** sebagai *normal array*, kita bisa menggunakan method `mysqli_result::fetch_row()` atau `mysqli_result::fetch_array(MYSQLI_NUM)`.

Mysqli Object

Berikut contoh penggunaannya:

24.mysql_result_fetch_row.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 $row = $result->fetch_row();
9
10 echo "<pre>";
11 print_r($row);
12 echo "</pre>";
13
14 $result->free();
```

Hasil kode program:

```
Array
(
    [0] => 1
    [1] => TV Samsung 43NU7090 4K
    [2] => 5
    [3] => 5399000
    [4] => 2019-01-04 16:59:53
)
```

Query yang saya jalankan adalah "SELECT * FROM barang", yakni ambil semua data yang ada di dalam tabel barang.

Setelah menjalankan query di baris 6, di baris 8 terdapat perintah \$row = \$result->fetch_row(), artinya: "ambil data yang tersimpan di dalam \$result sebagai normal array, kemudian simpan ke variabel \$row".

Di baris 11 variabel \$row ini saya tampilkan dengan perintah print_r(\$row). Hasilnya terlihat bahwa \$row berisi sebuah "array normal", maksudnya index dari array ini menggunakan nomor, dimana \$row[0] berisi "1", \$row[1] berisi "TV Samsung 43NU7090 4K", dst.

Kita juga bisa mengakses data ini secara langsung:

25.mysql_result_fetch_row_2.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 $row = $result->fetch_row();
```

Mysqli Object

```
9 echo $row[0]; echo " | ";
10 echo $row[1]; echo " | ";
11 echo $row[2]; echo " | ";
12 echo $row[3]; echo " | ";
13 echo $row[4];
14
15 $result->free();
```

Hasil kode program:

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-04 16:59:53
```

Di sini isi dari array \$row saya tampilkan secara langsung, dimana index array berurutan sesuai nomor kolom dari tabel **barang**.

Method `fetch_row()` hanya memberikan hasil untuk 1 baris saja pada setiap pemanggilan. Jika saya ingin menampilkan data baris kedua, method `fetch_row()` ini harus dijalankan lagi. Sehingga cara yang paling pas untuk menampilkan semua data adalah dengan membuat sebuah perulangan while, seperti contoh berikut:

26.mysql_result_fetch_row_while.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_row()){
9     echo $row[0]; echo " | ";
10    echo $row[1]; echo " | ";
11    echo $row[2]; echo " | ";
12    echo $row[3]; echo " | ";
13    echo $row[4];
14    echo "<br>";
15 }
16
17 $result->free();
```

Hasil kode program:

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-04 16:59:53
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-04 16:59:53
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-04 16:59:53
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-04 16:59:53
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-04 16:59:53
```

Perulangan while di baris 8 – 15 akan terus dijalankan untuk setiap baris data yang ada di dalam tabel **barang**.

Sebagai alternatif, kita juga bisa memakai method `fetch_array(MYSQLI_NUM)` sebagai alias atau

Mysqli Object

nama lain dari method `$result->fetch_row()`:

27.mysql_result_fetch_array_num.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_array(MYSQLI_NUM)){
9     echo $row[0];    echo " | ";
10    echo $row[1];   echo " | ";
11    echo $row[2];   echo " | ";
12    echo $row[3];   echo " | ";
13    echo $row[4];
14    echo "<br>";
15 }
16
17 $result->free();
```

Hasil yang di dapat akan sama seperti sebelumnya.

Selain mengambil data tabel sebagai "normal array", kita juga bisa mengambilnya sebagai *associative array*. **Associative array** adalah sebutan untuk array dengan *key* yang bisa berisi karakter non-angka, seperti huruf.

Untuk mengambil data dari `mysql_result` sebagai associative array, tersedia method `mysql_result::fetch_assoc()` atau `mysql_result::fetch_array(MYSQLI_ASSOC)`. Berikut contoh penggunaannya:

28.mysql_result_fetch_assoc_while.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_assoc()){
9     echo $row['id_barang'];      echo " | ";
10    echo $row['nama_barang'];   echo " | ";
11    echo $row['jumlah_barang']; echo " | ";
12    echo $row['harga_barang'];  echo " | ";
13    echo $row['tanggal_update'];
14    echo "<br>";
15 }
16
17 $result->free();
```

Hasil kode program:

Mysqli Object

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-04 16:59:53
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-04 16:59:53
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-04 16:59:53
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-04 16:59:53
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-04 16:59:53
```

Sekarang variabel `$row` berisi *associative array* dengan nama kolom sebagai *key*. Untuk menampilkan data dari `nama_barang`, diakses sebagai `$row['nama_barang']`. Dalam banyak hal, *associative array* ini lebih informatif daripada normal array seperti method sebelumnya.

Sebagai alternatif, kita juga bisa memakai method `fetch_array(MYSQLI_ASSOC)` sebagai alias atau nama lain dari method `$result->fetch_assoc()`:

29.mysql_result_fetch_array_assoc.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_array(MYSQLI_ASSOC)){
9     echo $row['id_barang'];    echo " | ";
10    echo $row['nama_barang'];  echo " | ";
11    echo $row['jumlah_barang']; echo " | ";
12    echo $row['harga_barang']; echo " | ";
13    echo $row['tanggal_update'];
14    echo "<br>";
15 }
16
17 $result->free();
```

Hasil yang di dapat akan sama seperti sebelumnya.

Meskipun jarang dipakai, terdapat method `mysqli_result::fetch_array(MYSQLI_ASSOC)` jika kita ingin mengkombinasikan *normal array* dan *associative array*, seperti contoh berikut:

30.mysql_result_fetch_array_both.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_array(MYSQLI_BOTH)){
9     echo $row['id_barang'];    echo " | ";
10    echo $row[1];              echo " | ";
11    echo $row['jumlah_barang']; echo " | ";
12    echo $row[3];              echo " | ";
13    echo $row['tanggal_update'];
```

Mysqli Object

```
14     echo "<br>";
15 }
16
17 $result->free();
```

Di dalam perulangan, array \$row bisa diakses sebagai *normal array* seperti \$row[1], dan juga bisa sebagai *associative array* seperti \$row['tanggal_update']. Meskipun lebih praktis, cara ini jarang dipakai karena menggunakan ruang memory yang lebih banyak. Secara internal, array \$row harus menampung 2 buah array, satu untuk *normal array* dan satu lagi untuk *associative array*.

Sebagai tambahan, jika method `fetch_array()` dipanggil tanpa argument, itu sama dengan `mysqli_result::fetch_array(MYSQLI_ASSOC)`. Dengan kata lain, perintah di baris 8 bisa juga ditulis sebagai:

```
while ($row = $result->fetch_array()){
```

Cara ketiga untuk menampilkan data dari `mysqli_result` adalah sebagai **object**. Maksudnya, variabel \$row akan berisi object dengan nama kolom tabel sebagai property. Untuk keperluan ini, kita memakai method `mysqli_result::fetch_object`. Berikut contoh penggunaannya:

31.mysql_result_fetch_object.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_object()){
9     echo $row->id_barang;      echo " | ";
10    echo $row->nama_barang;    echo " | ";
11    echo $row->jumlah_barang;  echo " | ";
12    echo $row->harga_barang;   echo " | ";
13    echo $row->tanggal_update;
14    echo "<br>";
15 }
16
17 $result->free();
```

Hasil kode program:

1	TV Samsung 43NU7090 4K	5	5399000	2019-01-04 16:59:53
2	Kulkas LG GC-A432HLHU	10	7600000	2019-01-04 16:59:53
3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-04 16:59:53
4	Printer Epson L220	14	2099000	2019-01-04 16:59:53
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-04 16:59:53

Dengan memakai method `fetch_object()`, isi tabel akan berbentuk object. Misalnya untuk mengakses isi dari kolom `jumlah_barang`, bisa dari perintah `$row->jumlah_barang`.

Sebagai rangkuman, berikut 7 method yang bisa dipakai untuk menampilkan isi **mysqli_result** object:

- ◆ `mysqli_result::fetch_row()`: normal array.
- ◆ `mysqli_result::fetch_assoc()`: associative array.
- ◆ `mysqli_result::fetch_array(MYSQLI_NUM)`: normal array.
- ◆ `mysqli_result::fetch_array(MYSQLI_ASSOC)`: associative array.
- ◆ `mysqli_result::fetch_array(MYSQLI_BOTH)`: normal array dan associative array.
- ◆ `mysqli_result::fetch_array()`: normal array dan associative array.
- ◆ `mysqli_result::fetch_object()`: object.

Kita bebas ingin menggunakan cara yang mana. Ada yang lebih suka menggunakan normal array karena penulisannya lebih singkat. Ada yang lebih suka associative array karena lebih informatif, serta ada yang memilih menggunakan object agar lebih pas dengan OOP.

Mengambil Data Tabel Dari **mysqli_result** Object

Dalam beberapa situasi, yang kita butuhkan adalah mengambil semua data tabel yang tersimpan di dalam **mysqli_result**, bukan langsung menampilkannya. Untuk keperluan ini tersedia method `mysqli_result::fetch_all()`.

Berbeda dengan method `fetch_row()` atau `fetch_assoc()` yang mengambil baris per baris, method `fetch_all()` mengambil seluruh hasil query `SELECT` dan menyimpannya sebagai array 2 dimensi.

Berikut contoh penggunaan method `mysqli_result::fetch_all()`:

32. `mysqli_result_fetch_all.php`

```

1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Ambil semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 $arr = $result->fetch_all();
9
10 echo "<pre>";
11 print_r($arr);
12 echo "</pre>";
13
14 $result->free();
```

Hasil kode program:

Mysqli Object

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => TV Samsung 43NU7090 4K
            [2] => 5
            [3] => 5399000
            [4] => 2019-01-04 16:59:53
        )

    [1] => Array
        (
            [0] => 2
            [1] => Kulkas LG GC-A432HLHU
            [2] => 10
            [3] => 7600000
            [4] => 2019-01-04 16:59:53
        )

    ...
    ...
)
```

Query yang dipakai sama seperti contoh kita sebelumnya, yakni "SELECT * FROM barang". Ini akan mengambil seluruh isi tabel **barang** dan diproses oleh kode program di baris 5-6.

Di baris 8, saya menjalankan perintah `$result->fetch_all()` yang hasilnya disimpan ke dalam variabel `$arr`. Method `fetch_all()` ini akan mengambil seluruh hasil query dalam bentuk array. Perintah `print_r($arr)` di baris 11 memperlihatkan isi dari variabel `$arr`.

Jika dipanggil tanpa argument, method `fetch_all()` akan menyimpan data tabel dalam bentuk "normal array", dimana key atau index array menggunakan angka. Ini sebenarnya sama dengan `fetch_all(MYSQLI_NUM)`.

Alternatif lain adalah method `fetch_all(MYSQLI_ASSOC)` yang akan mengambil hasil query dalam bentuk *associative array*. Berikut contohnya:

33.mysql_result_fetch_all_assoc.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Ambil semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 $arr = $result->fetch_all(MYSQLI_ASSOC);
9
10 echo "<pre>";
11 print_r($arr);
12 echo "</pre>";
13
14 $result->free();
```

Hasil kode program:

```
Array
(
    [0] => Array
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            [harga_barang] => 5399000
            [tanggal_update] => 2019-01-04 16:59:53
        )

    [1] => Array
        (
            [id_barang] => 2
            [nama_barang] => Kulkas LG GC-A432HLHU
            [jumlah_barang] => 10
            [harga_barang] => 7600000
            [tanggal_update] => 2019-01-04 16:59:53
        )

    ...
    ...
)
```

Perbedaan dengan kode program sebelumnya hanya di baris 8, yang kali ini saya menjalankan perintah `$arr = $result->fetch_all(MYSQLI_ASSOC)`. Seperti yang terlihat, variabel `$arr` berisi associative array dimana index atau key array menggunakan nama kolom tabel.

Teknik menyimpan keseluruhan hasil query ke dalam sebuah variabel ini sangat berguna, karena kita bisa memisahkan kode PHP yang dipakai untuk mengakses database dengan kode PHP yang nantinya dipakai untuk menampilkan data tersebut.

Ini umum dijumpai dalam framework yang menerapkan prinsip MVC (Model-View-Controller), dimana kode program yang mengakses database, terpisah dengan kode program yang akan dipakai untuk menampilkan data tersebut.

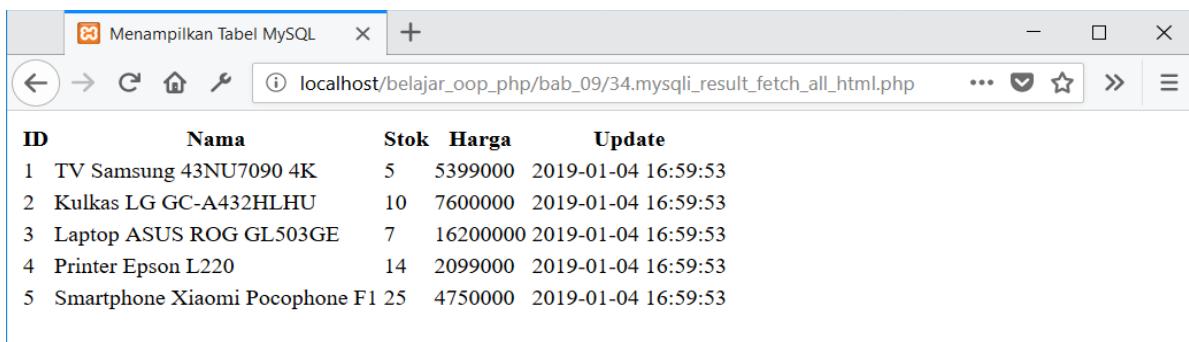
Pemisahan ini membuat penulisan kode menjadi lebih rapi, seperti contoh berikut:

34.mysql_result_fetch_all_html.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Ambil semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7 $arr = $result->fetch_all(MYSQLI_ASSOC);
8 $result->free();
9 ?>
10 <!DOCTYPE html>
11 <html lang="id">
```

Mysqli Object

```
12 <head>
13   <meta charset="UTF-8">
14   <title>Menampilkan Tabel MySQL</title>
15 </head>
16 <body>
17 <table>
18   <tr>
19     <th>ID</th>
20     <th>Nama</th>
21     <th>Stok</th>
22     <th>Harga</th>
23     <th>Update</th>
24   </tr>
25   <?php foreach ($arr as $key => $val) {?>
26   <tr>
27     <td><?php echo $val['id_barang']; ?></td>
28     <td><?php echo $val['nama_barang']; ?></td>
29     <td><?php echo $val['jumlah_barang']; ?></td>
30     <td><?php echo $val['harga_barang']; ?></td>
31     <td><?php echo $val['tanggal_update']; ?></td>
32   </tr>
33   <?php } ?>
34 </table>
35 </body>
36 </html>
```



The screenshot shows a web browser window titled "Menampilkan Tabel MySQL". The address bar indicates the URL is "localhost/belajar_oop_php/bab_09/34.mysql_result_fetch_all_html.php". The main content area displays a table with five columns: ID, Nama, Stok, Harga, and Update. The table contains five rows of data, each representing a product from a database. The data is as follows:

ID	Nama	Stok	Harga	Update
1	TV Samsung 43NU7090 4K	5	5399000	2019-01-04 16:59:53
2	Kulkas LG GC-A432HLHU	10	7600000	2019-01-04 16:59:53
3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-04 16:59:53
4	Printer Epson L220	14	2099000	2019-01-04 16:59:53
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-04 16:59:53

Gambar: Tampilan tabel barang yang di format dalam bentuk tabel HTML

Sama seperti sebelumnya, di baris 7 saya menyimpan semua hasil query "SELECT * FROM barang" ke dalam variabel \$arr menggunakan method \$result->fetch_all(MYSQLI_ASSOC). Kemudian blok PHP di tutup pada baris 9 dan proses pengaksesan ke database selesai.

Karena kita sudah berada di luar blok PHP, kode program di baris 10 dan seterusnya adalah "milik" HTML. Di sini saya membuat struktur kode HTML lengkap, mulai dari <!DOCTYPE html>, <head>, <body>, dst.

Di baris 17 – 34 terdapat tag <table>. Di sinilah saya ingin menampilkan isi dari variabel \$arr. Karena \$arr adalah sebuah array, kita bisa memakai perulangan **foreach** untuk menampilkan semua isinya (kode program di baris 25 – 33). Jika anda sudah pernah menampilkan isi data MySQL ke dalam tabel HTML, tentu tidak asing dengan teknik seperti ini.

Agar tampilannya lebih menarik, saya ingin tambah sedikit kode CSS serta merapikan format

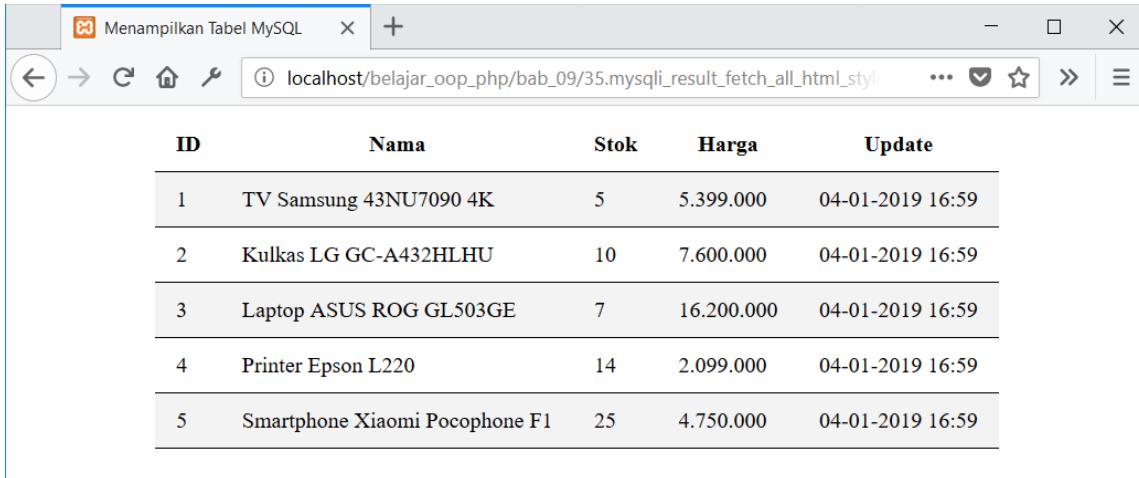
tampilan data:

35.mysql_result_fetch_all_html_style.php

```

1 <?php
2 // isi kode PHP sama seperti contoh sebelumnya
3 $arr = $result->fetch_all(MYSQLI_ASSOC);
4 // ...
5 ?>
6
7 <!DOCTYPE html>
8 <html lang="id">
9 <head>
10    <meta charset="UTF-8">
11    <title>Menampilkan Tabel MySQL</title>
12    <style>
13        table {
14            border-collapse: collapse;
15            margin: 0 auto;
16        }
17        td, th {
18            border-bottom: 1px solid black;
19        }
20        th,td {
21            padding: 10px 15px;
22        }
23        tr:nth-child(even) {background-color: #f2f2f2;}
24    </style>
25 </head>
26 <body>
27 <table>
28     <tr>
29         <th>ID</th>
30         <th>Nama</th>
31         <th>Stok</th>
32         <th>Harga</th>
33         <th>Update</th>
34     </tr>
35     <?php foreach ($arr as $key => $val) {?>
36     <tr>
37         <td><?php echo $val['id_barang']; ?></td>
38         <td><?php echo $val['nama_barang']; ?></td>
39         <td><?php echo $val['jumlah_barang']; ?></td>
40         <td><?php echo number_format($val['harga_barang'], 0, ',', '.'); ?></td>
41         <td><?php $tanggal = new DateTime($val['tanggal_update']);
42             echo $tanggal->format("d-m-Y H:i"); ?></td>
43     </tr>
44     <?php } ?>
45 </table>
46 </body>
47 </html>
```

Mysqli Object



The screenshot shows a web browser window with the title "Menampilkan Tabel MySQL". The URL is "localhost/belajar_oop_php/bab_09/35.mysql_result_fetch_all_html_style.php". The page displays a table of product data with the following columns: ID, Nama, Stok, Harga, and Update. The data is as follows:

ID	Nama	Stok	Harga	Update
1	TV Samsung 43NU7090 4K	5	5.399.000	04-01-2019 16:59
2	Kulkas LG GC-A432HLHU	10	7.600.000	04-01-2019 16:59
3	Laptop ASUS ROG GL503GE	7	16.200.000	04-01-2019 16:59
4	Printer Epson L220	14	2.099.000	04-01-2019 16:59
5	Smartphone Xiaomi Pocophone F1	25	4.750.000	04-01-2019 16:59

Gambar: Tampilan tabel barang yang sudah di-style dengan CSS

Tidak ada perubahan kode PHP di bagian awal. Variabel tetap \$arr dipakai untuk menyimpan hasil pemanggilan method \$result->fetch_all(MYSQLI_ASSOC). Kemudian di baris 12 – 24 saya menulis sedikit kode CSS agar tampilan tabel menjadi lebih menarik.

Saat memproses data di baris 35 – 44, saya mengolah "data mentah" dari kolom harga_barang dan tanggal_update.

Untuk kolom harga_barang, nilainya di tampilkan memakai fungsi number_format(). Tujuannya agar sesuai dengan format penulisan angka di Indonesia, dimana tanda titik dipakai untuk memisah angka ribuan.

Untuk kolom tanggal_update, nilainya juga saya format menjadi "dd-mm-yyyy hh:mm". Proses ini memakai DateTime object serta method DateTime::format().

Hasilnya, tampilan data tabel barang menjadi lebih rapi.

Exercise

Seperti yang sudah kita lihat, method mysqli_result::fetch_all(MYSQLI_ASSOC) mengambil keseluruhan data tabel dalam sekali proses, sedangkan method mysqli_result::fetch_assoc() hanya mengambil 1 baris dalam sekali jalan.

Sebagai tantangan, bisakah anda membuat kode program untuk mengisi variabel \$arr menggunakan method mysqli_result::fetch_assoc()?

Tipsnya, gunakan perulangan **while** untuk meng-generate array \$arr.

Latihan ini sebenarnya lebih ke menguji pemahaman seputar pengolahan array. Intinya adalah bagaimana cara merangkai sebuah array 2 dimensi menggunakan perulangan **while**.

Berikut kode yang saya gunakan:

36.mysql_result_exercise_1.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Ambil semua data di tabel barang
5 $query = "SELECT * FROM barang";
6 $result = $mysqli->query($query);
7
8 $i=0;
9 while ($row = $result->fetch_array(MYSQLI_ASSOC)){
10     $arr[$i]['id_barang'] = $row['id_barang'];
11     $arr[$i]['nama_barang'] = $row['nama_barang'];
12     $arr[$i]['jumlah_barang'] = $row['jumlah_barang'];
13     $arr[$i]['harga_barang'] = $row['harga_barang'];
14     $arr[$i]['tanggal_update']= $row['tanggal_update'];
15     $i++;
16 }
17 $result->free();
18
19 echo "<pre>";
20 print_r($arr);
21 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => Array
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            [harga_barang] => 5399000
            [tanggal_update] => 2019-01-04 16:59:53
        )

    [1] => Array
        (
            [id_barang] => 2
            [nama_barang] => Kulkas LG GC-A432HLHU
            [jumlah_barang] => 10
            [harga_barang] => 7600000
            [tanggal_update] => 2019-01-04 16:59:53
        )

    ...
)
```

Sebelum perulangan while, di baris 8 saya menyiapkan variabel bantu \$i. Variabel ini

nantinya dipakai sebagai penanda urutan array.

Di dalam perulangan while, nilai variabel \$i menjadi index array pertama. Sedangkan untuk index array kedua menggunakan nama kolom sesuai isi data tabel barang (ingat, karena kita sedang merancang sebuah array 2 dimensi). Variabel \$i akan naik sebanyak 1 angka dengan perintah \$i++ di baris 15.

Hasilnya, nilai kolom id_barang untuk baris pertama akan berada di \$arr[0] ['id_barang'], nilai kolom id_barang untuk baris kedua akan berada di \$arr[1] ['id_barang'], dst.

Proses pembuatan \$arr secara manual menggunakan method mysqli_result::fetch_assoc() ini kadang lebih bermanfaat daripada langsung mengambil semua data menggunakan mysqli_result::fetch_all(). Alasannya karena kita bisa mengolah data sebelum diinput ke dalam array.

Dalam contoh dengan mysqli_result::fetch_all(), saya menformat tampilan kolom harga_barang dan tanggal_update ketika akan ditampilkan. Jika kita menggunakan mysqli_result::fetch_assoc(), proses ini bisa dilakukan pada saat pembuatan array \$arr:

37.mysql_fetch_array_process.php

```

1  <?php
2  $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4  // Ambil semua data di tabel barang
5  $query = "SELECT * FROM barang";
6  $result = $mysqli->query($query);
7
8  $i=0;
9  while ($row = $result->fetch_array(MYSQLI_ASSOC)){
10    $arr[$i]['id_barang'] = $row['id_barang'];
11    $arr[$i]['nama_barang'] = $row['nama_barang'];
12    $arr[$i]['jumlah_barang'] = $row['jumlah_barang'];
13    $arr[$i]['harga_barang'] = number_format($row['harga_barang'], 0, ',', '.');
14
15    $tanggal = new DateTime($row['tanggal_update']);
16    $arr[$i]['tanggal_update']= $tanggal->format("d-m-Y H:i");
17    $i++;
18 }
19 ?>
20
21 <!DOCTYPE html>
22 <html lang="id">
23 <head>
24   <meta charset="UTF-8">
25   <title>Menampilkan Tabel MySQL</title>
26   <style>
27     table {
28       border-collapse: collapse;

```

Mysqli Object

```
29         margin: 0 auto;
30     }
31     td, th {
32         border-bottom: 1px solid black;
33     }
34     th,td {
35         padding: 10px 15px;
36     }
37     tr:nth-child(even) {background-color: #f2f2f2;}
38 </style>
39 </head>
40 <body>
41 <table>
42     <tr>
43         <th>ID</th>
44         <th>Nama</th>
45         <th>Stok</th>
46         <th>Harga</th>
47         <th>Update</th>
48     </tr>
49     <?php foreach ($arr as $key => $val) {?>
50     <tr>
51         <td><?php echo $val['id_barang']; ?></td>
52         <td><?php echo $val['nama_barang']; ?></td>
53         <td><?php echo $val['jumlah_barang']; ?></td>
54         <td><?php echo $val['harga_barang']; ?></td>
55         <td><?php echo $val['tanggal_update']; ?></td>
56     </tr>
57     <?php } ?>
58 </table>
59 </body>
60 </html>
```

Perhatikan perulangan while di baris 9 – 18. Di sini saya mengambil data menggunakan method `fetch_array(MYSQLI_ASSOC)`.

Pada saat proses pembuatan array `$arr`, data mentah dari MySQL bisa diolah terlebih dahulu, seperti di baris 13 dan 16. Di sini saya menggunakan fungsi `number_format()` serta `DateTime` object yang sama seperti sebelumnya. Sehingga, variabel `$arr` sudah berisi data akhir yang tinggal di tampilan di baris 49 – 57.

Menghapus Data Tabel

`Mysqli_result` object hanya dipakai untuk proses menampilkan hasil tabel (query `SELECT`). Untuk menghapus data tabel, kita tinggal menjalankan query `DELETE` dari method `mysqli::query()` seperti contoh berikut:

38.mysql_delete_query.php

```
1 <?php
2 $mysql = new mysqli("localhost", "root", "", "ilkoom");
3
```

Mysqli Object

```
4 // Tampilkan semua data di tabel barang
5 $query = "DELETE FROM barang WHERE id_barang = 1";
6 $mysqli->query($query);
7 echo "Terdapat ".$mysqli->affected_rows." baris yang telah dihapus"
```

Hasil kode program:

Terdapat 1 baris yang telah dihapus

Query "DELETE FROM barang WHERE id_barang = 1" artinya saya ingin menghapus data di tabel barang untuk baris yang memiliki `id_barang = 1`. Ini akan menghapus baris "TV Samsung 43NU7090 4K" dari tabel barang.

Di baris 7 saya mengakses property `$mysqli->affected_rows` untuk mendapatkan informasi jumlah baris tabel yang dihapus oleh query DELETE.

Untuk melihat isi tabel barang ini, silahkan jalankan kembali kode program sebelumnya (materi tentang method `fetch_all()`)

ID	Nama	Stok	Harga	Update
2	Kulkas LG GC-A432HLHU	10	7.600.000	04-01-2019 16:59
3	Laptop ASUS ROG GL503GE	7	16.200.000	04-01-2019 16:59
4	Printer Epson L220	14	2.099.000	04-01-2019 16:59
5	Smartphone Xiaomi Pocophone F1	25	4.750.000	04-01-2019 16:59

Gambar: isi tabel barang dengan id 1 sudah dihapus

Terlihat bahwa baris dengan `id_barang 1` sudah tidak ada lagi di dalam tabel.

Property `$mysqli->affected_rows` memberikan informasi terkait jumlah baris yang dipengaruhi oleh sebuah query. Jika query DELETE menghapus 2 baris, maka property ini juga akan berisi angka 2, seperti contoh berikut:

39.mysql_delete_query_2.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Tampilkan semua data di tabel barang
5 $query = "DELETE FROM barang WHERE id_barang = 4 OR id_barang = 5";
6 $mysqli->query($query);
7 echo "Terdapat ".$mysqli->affected_rows." baris yang telah dihapus <br>";
```

Hasil kode program:

Terdapat 2 baris yang telah dihapus

ID	Nama	Stok	Harga	Update
2	Kulkas LG GC-A432HLHU	10	7.600.000	04-01-2019 16:59
3	Laptop ASUS ROG GL503GE	7	16.200.000	04-01-2019 16:59

Dalam contoh ini, query "DELETE FROM barang WHERE id_barang = 4 OR id_barang = 5" akan menghapus 2 buah baris tabel, yakni baris dengan id_barang 4 dan 5.

Multi Query

Dalam situasi tertentu, kadang kita ingin menjalankan banyak query sekaligus. Misalnya setelah query `INSERT`, langsung diikuti dengan query `UPDATE`. Biasanya untuk yang seperti ini saya tetap menjalankan 2 buah query terpisah agar kode programnya menjadi lebih mudah dikelola.

Namun jika anda harus menjalankan 2 query secara langsung, PHP menyediakan method `mysqli::multi_query()`. Method ini mirip seperti method `mysqli::query()` yang sering kita pakai. Hanya saja untuk `mysqli::multi_query()` bisa menjalankan 2 query atau lebih. Di antara perintah query tersebut dipisah dengan tanda titik koma " ; ".

Berikut contoh penggunaan `mysqli::multi_query()`:

Karena pada contoh sebelumnya kita menghapus data dari tabel **barang**, silahkan reset ulang tabel barang dengan cara menjalankan file `20.mysql_generate.php`.

40.mysql_multi_query.php

```

1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Generate tanggal hari ini
5 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
6 $timestamp = $sekarang->format("Y-m-d H:i:s");
7
8 // Jalankan 3bh query
9 $query = "INSERT INTO barang
10    (nama_barang, jumlah_barang, harga_barang, tanggal_update)
11   VALUES ('Hardisk Eksternal WD My Passport 2TB',9,1120000,'$timestamp');
12   UPDATE barang SET jumlah_barang = 5, tanggal_update = '$timestamp'
13   WHERE id_barang=3;
14   UPDATE barang SET harga_barang = 4500000, tanggal_update = '$timestamp'
15   WHERE id_barang=5";
16
17 $mysqli->multi_query($query);
18 echo "Terdapat ".$mysqli->affected_rows." baris yang ditambah <br>";

```

Hasil kode program:

Terdapat 1 baris yang ditambah

Di baris 5 dan 6, saya membuat kode program untuk meng-generate tanggal hari ini.

Kemudian di baris 9 – 15 terdapat 3 buah query MySQL yang langsung digabung ke dalam 1 string panjang. Tiga query tersebut adalah proses INSERT untuk 1 data baru, serta proses UPDATE untuk 2 buah data yang sudah tersedia di dalam tabel barang. Setiap query dipisah dengan tanda titik koma.

Method `$mysqli->multi_query($query)` di baris 17 dipakai untuk menjalankan ketiga query ini. Berikut hasil akhir dari tabel barang setelah menjalankan query di atas. Kembali untuk melihat isi tabel barang ini, cukup jalankan kode program yang kita buat ketika membahas method `$mysqli_result::fetch_all()`:

ID	Nama	Stok	Harga	Update
1	TV Samsung 43NU7090 4K	5	5.399.000	07-01-2019 17:11
2	Kulkas LG GC-A432HLHU	10	7.600.000	07-01-2019 17:11
3	Laptop ASUS ROG GL503GE	5	16.200.000	07-01-2019 18:12
4	Printer Epson L220	14	2.099.000	07-01-2019 17:11
5	Smartphone Xiaomi Pocophone F1	25	4.500.000	07-01-2019 18:12
6	Hardisk Eksternal WD My Passport 2TB	9	1.120.000	07-01-2019 18:12

Gambar: hasil akhir dari tabel barang

Terlihat data "Hardisk Eksternal WD My Passport 2TB" sudah berhasil ditambah. Selain itu tanggal update untuk id 3 dan 5 juga sudah berubah.

Method `$mysqli::multi_query()` juga memiliki kelemahan tersendiri. Yakni jika setelah pemanggilan method `multi_query()` kita menjalankan method `$mysqli::query()` lagi, harus jalankan method `$mysqli->more_results` dan `$mysqli->next_result()`. Kalau tidak, akan tampil pesan error:

Commands out of sync; you can't run this command now (2014)

Berikut contoh penggunaanya:

41. `mysqli_multi_query_2.php`

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
```

```

4 // Generate tanggal hari ini
5 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
6 $timestamp = $sekarang->format("Y-m-d H:i:s");
7
8 // Jalankan 3bh query
9 $query = "INSERT INTO barang
10    (nama_barang, jumlah_barang, harga_barang, tanggal_update)
11   VALUES ('Hardisk Eksternal WD My Passport 2TB',9,1120000,'$timestamp');
12   UPDATE barang SET jumlah_barang = 5, tanggal_update = '$timestamp'
13 WHERE id_barang=3;
14   UPDATE barang SET harga_barang = 4500000, tanggal_update = '$timestamp'
15 WHERE id_barang=5";
16
17 $mysqli->multi_query($query);
18 echo "Terdapat ".$mysqli->affected_rows." baris yang ditambah <br>";
19
20 // Proses sinkronisasi dengan MySQL (akibat $mysqli->multi_query)
21 while( $mysqli->more_results())
22 {
23   $mysqli->next_result();
24 }
25
26 // Ambil semua data di tabel barang
27 $query = "SELECT * FROM barang";
28 $result = $mysqli->query($query);
29 $arr = $result->fetch_all(MYSQLI_ASSOC);

```

Kode program di baris 1 – 18 sama seperti sebelumnya. Namun kali ini di baris 28 saya menjalankan perintah `$mysqli->query($query)`. Agar perintah ini bisa di proses, kita harus tulis "proses sinkronisasi" seperti yang ada di baris 21 – 24. Bisa dibilang bahwa pemanggilan method `$mysqli->more_results()` dan `$mysqli->next_result()` dipakai untuk "bersih-bersih memory" akibat penggunaan method `$mysqli->multi_query($query)` di baris 17.

Saya pribadi sangat jarang menggunakan method `multi_query()`. Kode program akan lebih rapi jika setiap perintah query dijalankan satu per satu. Toh meskipun dalam 1 halaman terdapat 10 kali pemanggilan method `query()`, semuanya tetap diproses dalam 1 kali pengaksesan file.

Namun jika ada situasi dimana anda harus menjalankan beberapa perintah query sekaligus, method `$mysqli::multi_query()` bisa jadi solusi.

Validasi Data

Data yang terdapat di dalam tabel MySQL biasanya berasal dari form HTML. Sehingga siapa saja bisa menginput informasi yang tidak sesuai. Atau yang lebih parah, mencoba membobol sistem menggunakan metode seperti SQL Injection.

Proses validasi data menjadi sebuah "rutinitas" yang hampir selalu kita rancang saat memproses data form. Misalnya memakai fungsi `trim()` untuk menghapus tambahan spasi, menggunakan fungsi `strip_tags()` untuk menghapus kode-kode HTML, memeriksa tipe data

yang diinput, membuat pengecekan dengan *regular expression*, serta metode validasi lainnya.

Di sini saya tidak akan membahas semua teknik-teknik validasi form karena itu cukup panjang dan bisa jadi bab tersendiri (seperti di buku PHP Uncover). Namun kita akan lihat beberapa masalah validasi yang bisa terjadi saat proses penyimpanan ke database MySQL. Materi ini juga sebagai persiapan sebelum masuk ke topik tentang **prepared statement**.

Sebelum itu, silahkan reset tabel barang karena dalam praktek sebelum ini kita mengubah beberapa data. Caranya, akses kembali file 20.mysqli_generate.php.

42.mysqli_validate_int.php

```

1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Ambil data di tabel barang
5 $query = "SELECT * FROM barang WHERE id_barang = 5";
6 $result = $mysqli->query($query);
7
8 while ($row = $result->fetch_assoc()){
9     echo $row['id_barang'];      echo " | ";
10    echo $row['nama_barang'];   echo " | ";
11    echo $row['jumlah_barang']; echo " | ";
12    echo $row['harga_barang'];  echo " | ";
13    echo $row['tanggal_update'];
14    echo "<br>";
15 }
16 $result->free();

```

Hasil kode program:

5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41

Tidak ada yang baru dari kode program di atas, dimana saya menjalankan query "SELECT * FROM barang WHERE id_barang = 5" kemudian menggunakan perulangan **while** untuk menampilkan data.

Pada prakteknya, kondisi `id_barang = 5` dalam penulisan query di atas sering berasal dari form HTML seperti contoh berikut:

43.mysqli_validate_int_2.php

```

1 <?php
2 $_GET['id_barang'] = "5";
3 $id_barang = $_GET['id_barang'];
4
5 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
6
7 // Ambil data di tabel barang
8 $query = "SELECT * FROM barang WHERE id_barang = $id_barang";

```

Mysqli Object

```
9 $result = $mysqli->query($query);
10
11 while ($row = $result->fetch_assoc()){
12     echo $row['id_barang'];      echo " | ";
13     echo $row['nama_barang'];    echo " | ";
14     echo $row['jumlah_barang'];  echo " | ";
15     echo $row['harga_barang'];   echo " | ";
16     echo $row['tanggal_update'];
17     echo "<br>";
18 }
19 $result->free();
```

Hasil kode program:

```
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41
```

Saya "mensimulasikan" data yang berasal dari form dengan kode program di baris 2. Kita anggap user menginput angka 5 di isian form lalu men-klik tombol submit. Dengan demikian, variabel `$_GET['id_barang']` berisi angka 5. Variabel ini selanjutnya di pindahkan ke variabel `$id_barang` di baris 3.

Penulisan query sekarang menjadi "SELECT * FROM barang WHERE id_barang = `$id_barang`", yang ketika di jalankan akan diproses sebagai "SELECT * FROM barang WHERE id_barang = 5".

Selanjutnya perhatikan kode program berikut:

```
44.mysqli_validate_int_inject.php
```

```
1 <?php
2 $_GET['id_barang'] = "5 OR id_barang = 2";
3 $id_barang = $_GET['id_barang'];
4
5 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
6
7 // Ambil data di tabel barang
8 $query = "SELECT * FROM barang WHERE id_barang = $id_barang";
9 $result = $mysqli->query($query);
10
11 while ($row = $result->fetch_assoc()){
12     echo $row['id_barang'];      echo " | ";
13     echo $row['nama_barang'];    echo " | ";
14     echo $row['jumlah_barang'];  echo " | ";
15     echo $row['harga_barang'];   echo " | ";
16     echo $row['tanggal_update'];
17     echo "<br>";
18 }
19 $result->free();
```

Hasil kode program:

```
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-07 21:09:41
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41
```

Mysqli Object

Di baris 2 si user bukan lagi mengetik angka 5, tapi string "5 OR id_barang = 2" ke dalam `$_GET['id_barang']`. Ini merupakan sebuah teknik SQL injection, karena yang diinput merupakan kode SQL yang dirancang sedemikian rupa.

Akibatnya, query yang diproses di baris 8 menjadi "SELECT * FROM barang WHERE id_barang = 5 OR id_barang = 2". Sehingga tampil 2 baris data, dimana seharusnya 1 user hanya bisa menampilkan 1 data saja.

Salah satu cara untuk mencegah masalah ini adalah dengan mengkonversi inputan user menjadi tipe data integer:

```
1  $_GET['id_barang'] = "5 OR id_barang = 2";
2  $id_barang = (int) $_GET['id_barang'];
```

Tambahan perintah `(int)` di baris 2 membuat isi variabel `$_GET['id_barang']` di konversi sebagai tipe data integer. Di dalam PHP, jika sebuah string dikonversi menjadi integer, string tersebut hanya akan dibaca hingga huruf pertama.

Sebagai contoh, hasil dari `(int) "5 OR id_barang = 2"` adalah 5. Atau hasil dari `(int) "12R78"` adalah 12. Berikut kode program lengkap dari teknik ini:

45.mysql_validate_int_conversion.php

```
1  <?php
2  $_GET['id_barang'] = "5 OR id_barang = 2";
3  $id_barang = (int) $_GET['id_barang'];
4
5  $mysqli = new mysqli("localhost", "root", "", "ilkoom");
6
7  // Ambil data di tabel barang
8  $query = "SELECT * FROM barang WHERE id_barang = $id_barang";
9  $result = $mysqli->query($query);
10
11 while ($row = $result->fetch_assoc()) {
12  echo $row['id_barang']; echo " | ";
13  echo $row['nama_barang']; echo " | ";
14  echo $row['jumlah_barang']; echo " | ";
15  echo $row['harga_barang']; echo " | ";
16  echo $row['tanggal_update'];
17  echo "<br>";
18 }
19 $result->free();
```

Hasil kode program:

5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41

Teknik konversi memakai perintah `(int)` seperti ini hanya bisa dipakai jika kondisi yang kita input ke query berupa angka.

Bagaimana dengan tipe string? Berikut contoh kasusnya:

Mysqli Object

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 $_GET['nama_barang'] = "Buku Moody's";
5 $nama_barang = $_GET['nama_barang'];
6
7 try {
8     $mysqli = new mysqli("localhost", "root", "", "ilkoom");
9
10    // Ambil data di tabel barang
11    $query = "SELECT * FROM barang WHERE nama_barang = '$nama_barang'";
12    $result = $mysqli->query($query);
13
14    if ($mysqli->error){
15        throw new Exception($mysqli->error, $mysqli->errno);
16    }
17    else {
18        if ($result->num_rows === 0) {
19            echo "Data tidak ditemukan";
20        }
21        else {
22            echo "Data tersedia";
23        }
24    }
25 }
26 catch (Exception $e) {
27     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
28 }
29 finally {
30     if (isset($mysqli)) {
31         $mysqli->close();
32     }
33 }
```

Hasil kode program:

Koneksi / Query bermasalah: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''s'' at line 1 (1064)

Di sini saya membuat proses pencarian untuk kolom `nama_barang`. Nama barang yang dicari adalah "Buku Moody's" yang disimpan ke dalam variabel `$_GET['nama_barang']`. Kembali ini sebagai simulasi bahwa proses pencarian berasal dari form HTML yang diinput oleh user.

Variabel `$_GET['nama_barang']` kemudian dipindahkan ke variabel `$nama_barang`, yang kemudian menjadi nilai input untuk query "SELECT * FROM barang WHERE nama_barang = '\$nama_barang'".

Akan tetapi, keyword yang dipakai untuk pencarian ini memiliki karakter khusus yang tidak bisa diproses oleh MySQL, yakni tanda kutip satu di kata "Moody's". Akibatnya, tampil pesan error, padahal output dari kode program di atas saya rancang agar menampilkan string "Data tidak ditemukan" atau "Data tersedia" tergantung ada tidaknya hasil pencarian di tabel

barang.

Solusinya, kita bisa memakai method `mysqli::real_escape_string()` untuk men-escape karakter khusus (seperti tanda kutip satu) agar bisa dipahami oleh MySQL:

47. `mysqli_validate_string_real_escape_string.php`

```

1  <?php
2  mysqli_report(MYSQLI_REPORT_STRICT);
3
4  $_GET['nama_barang'] = "Buku Moody's";
5
6  try {
7      $mysqli = new mysqli("localhost", "root", "", "ilkoom");
8
9      $nama_barang = $mysqli->real_escape_string($_GET['nama_barang']);
10
11     // Ambil data di tabel barang
12     $query = "SELECT * FROM barang WHERE nama_barang = '$nama_barang'";
13     $result = $mysqli->query($query);
14
15     if ($mysqli->error){
16         throw new Exception($mysqli->error, $mysqli->errno);
17     }
18     else {
19         if ($result->num_rows === 0) {
20             echo "Data tidak ditemukan";
21         }
22         else {
23             echo "Data tersedia";
24         }
25     }
26 }
27 catch (Exception $e) {
28     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
29 }
30 finally {
31     if (isset($mysqli)) {
32         $mysqli->close();
33     }
34 }
```

Hasil kode program:

Data tidak ditemukan

Di baris 4, variabel `$_GET['nama_barang']` tetap diisi dengan string "Buku Moody's". Namun di baris 9, variabel `$nama_barang` diisi dengan hasil dari pemanggilan method `$mysqli->($GET['nama_barang'])`.

Proses pemanggilan method `real_escape_string()` ini harus saya tempatkan setelah pembuatan mysqli object di baris 7.

Sekarang tidak lagi tampil pesan error, namun teks "Data tidak ditemukan" karena nama barang "" memang tidak tersedia di dalam tabel.

Method `mysqli::real_escape_string()` sering dipakai untuk memfilter data dari SQL Injection. Method ini akan men-escape atau memproses karakter khusus yang berbentuk NUL (ASCII 0), \n, \r, \, ', ", dan Control-Z.

Dengan 2 contoh kasus ini, kita bisa mengambil kesimpulan bahwa proses validasi menjadi salah satu aspek yang tidak bisa dilupakan.

Untuk data dengan tipe data angka, harus dikonversi menggunakan perintah (`int`) atau (`float`) agar kita bisa memastikan tidak ada perintah SQL yang disisipkan ke dalam data tersebut.

Sedangkan untuk inputan string, bisa menggunakan method `mysqli::real_escape_string()` untuk "membersihkannya" dari karakter-karakter khusus.

Alternatif yang lebih aman dan lebih modern adalah menggunakan teknik **prepared statement**, yang akan segera kita bahas.

9.3. Mysqli_stmt Class

Mysqli_stmt adalah class khusus yang dipakai untuk menjalankan query MySQL sebagai *prepared statement*.

Prepared statement sendiri merupakan teknik memproses query MySQL dengan memisahkan antara query dengan data yang akan input. Nantinya, query disiapkan terlebih dahulu (proses `prepare`), kemudian data diinput (proses `bind`), dan baru query dijalankan (proses `execute`).

Dengan demikian, di dalam prepared statement terdapat 3 langkah yang harus di proses secara berurutan: **prepare**, **bind**, dan **execute**.

Seperti yang akan kita lihat nanti, sebenarnya prepared statement ini tampak lebih ribet daripada menjalankan query seperti yang sudah kita lakukan seperti sebelumnya. Namun prepared statement membawa beberapa keunggulan:

1. Query yang ditulis dengan prepared statement otomatis ter-validasi. Artinya, prepared statement kebal terhadap MySQL injection.
2. Untuk query yang berulang, kita hanya perlu menulis 1 perintah query saja (proses `prepare`), kemudian menginput data beberapa. Cara ini lebih efisien dibandingkan menjalankan query satu per satu.

Inti dari pembuatan *prepared statement* ada di proses **prepare** dan **bind**. Dalam proses `prepare`, kita membuat query seperti biasa namun untuk bagian data input diganti dengan tanda tanya "?", seperti contoh berikut:

Mysqli Object

```
"SELECT * FROM barang WHERE id_barang = ?"
```

atau

```
"INSERT INTO barang VALUES (?, ?, ?, ?, ?)"
```

Dalam **mysqli** object, proses *prepared* ini menggunakan method **mysqli::prepare()**. Method ini butuh sebuah argument bertipe string yang berisi query kemudian mengembalikan sebuah **mysqli_stmt** object. Berikut contoh pemanggilannya:

```
$stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");
```

Dengan perintah ini, variabel `$stmt` akan berisi sebuah **mysqli_stmt** object. Sepanjang proses pembuatan prepared statement, variabel inilah yang akan terus kita akses. Nama variabel penampung **mysqli_stmt** object ini boleh bebas, tidak harus `$stmt`.

Langkah selanjutnya adalah proses **bind**, yakni mengisi tanda tanya yang sudah kita siapkan pada saat *prepare*. Proses *bind* dijalankan dengan method **mysqli_stmt::bind_param()**.

Perhatikan bahwa method `bind_param()` ini adalah "kepunyaan" dari **mysqli_stmt** object.

Method **mysqli_stmt::bind_param()** butuh minimal 2 argument, yakni jenis tipe data serta variabel yang akan diinput. Disebut "minimal" karena argument method bisa lebih dari 2 tergantung jumlah tanda tanya yang terdapat di query.

Berikut contoh penggunaan dari **mysqli_stmt::bind_param()**:

```
1 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");  
2 $id_barang = 5;  
3 $stmt->bind_param("i", $id_barang);
```

Perintah di baris 1 adalah proses *prepare* seperti yang sudah kita bahas sebelumnya. Kemudian di baris 2 saya membuat variabel `$id_barang` yang diisi angka 5.

Proses **bind** terdapat di baris 3. Argument pertama diisi dengan string "i". Huruf i di sini mewakili **integer**, yakni tipe data dari tanda tanya yang ada di dalam query. Data untuk tanda tanya itu sendiri ada di argument kedua, yakni `$id_barang`.

Dengan kata lain, perintah di baris 3 bisa dibaca: "Ganti tanda tanya di query *prepare* dengan isi variabel integer dari `$id_barang`".

Seandainya pada saat *prepare* terdapat dua buah tanda tanya, maka method `bind_param()` akan butuh 3 argument, seperti contoh berikut:

```
1 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?  
2          OR id_barang = ?");  
3 $id_barang_1 = 5;  
4 $id_barang_2 = 1;  
5 $stmt->bind_param("ii", $id_barang_1, $id_barang_2);
```

Dalam proses *prepare* di baris 1 – 2, query tersebut memiliki 2 buah tanda tanya. Oleh karena

itu di bagian *bind* (baris 5) kita harus menginput nama variabel untuk kedua tanda tanya ini plus tipe datanya. Argument pertama pada saat proses *bind* adalah "ii" yang berarti "integer integer", yakni tipe data untuk variabel \$id_barang_1 dan \$id_barang_2.

Terdapat 4 buah pilihan tipe data pada saat proses *bind*, yakni:

- i = integer
- d = double
- s = string
- b = blob (binary)

Sebagai contoh lain, berikut penulisan perintah *prepare* dan *bind* untuk query **INSERT** ke tabel barang:

```
1 $stmt = $mysqli->prepare("INSERT INTO barang (nama_barang,
2 jumlah_barang, harga_barang, tanggal_update) VALUES (?, ?, ?, ?)");
3
4 $stmt->bind_param("siis", $nama_barang, $jumlah_barang,
5                     $harga_barang, $tanggal_update);
```

Argument pertama dari method *bind_param()* adalah "siis" yang bersesuaian dengan variabel \$nama_barang yang bertipe **string**, \$jumlah_barang bertipe **integer**, \$harga_barang bertipe **integer**, dan \$tanggal_update bertipe **string**. Tentu saja kita harus mengisi setiap variabel ini dengan data yang sebenarnya.

Sebagai info tambahan, nilai argument untuk method *bind_param()* harus berupa variabel, tidak bisa di input dengan data langsung. Sebagai contoh, perintah berikut akan menghasilkan error:

```
1 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");
2 $stmt->bind_param("i", 5);
```

Meskipun secara logika kode program di atas sudah benar, namun PHP tidak membolehkan penulisan seperti ini. Angka 5 di dalam argument kedua method *bind_param()* harus berbentuk variabel.

Posisi penulisan variabel ini bisa sebelum atau sesudah pemanggilan method *bind_param()*:

```
1 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");
2
3 $stmt->bind_param("i", $id_barang);
4 $id_barang = 5;
```

Setelah *prepare* dan *bind*, langkah terakhir adalah *execute* yakni menjalankan query tersebut dengan method **mysqli_stmt::execute()**. Untuk query seperti **INSERT**, **UPDATE** dan **DELETE**, proses menjalankan query sudah selesai:

```
1 $stmt = $mysqli->prepare("DELETE FROM barang WHERE id_barang = ?");
```

Mysqli Object

```
2  
3 $stmt->bind_param("i", $id_barang);  
4 $id_barang = 5;  
5  
6 $stmt->execute();
```

Namun untuk query **SELECT**, nantinya kita butuh pemrosesan lebih lanjut untuk menampilkan hasil yang didapat dari MySQL. Misalnya menjalankan method **mysqli_stmt::get_result()** yang akan mengembalikan **mysqli_result** object untuk kemudian ditampilkan dengan berbagai method milik **mysqli_result** seperti **mysqli_result::fetch_row()**, **mysqli_result::fetch_assoc()**, dll:

```
1 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");  
2  
3 $stmt->bind_param("i", $id_barang);  
4 $id_barang = 5;  
5  
6 $stmt->execute();  
7  
8 $result = $stmt->get_result();  
9 while ($row = $result->fetch_assoc()) {  
10 echo $row['id_barang'];  
11 // ...  
12 }
```

Terakhir, terdapat method opsional untuk menghapus memory dan menutup proses prepared statement dengan method **mysqli_stmt::free_result()** serta **mysqli_stmt::close()**:

```
1 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");  
2  
3 $stmt->bind_param("i", $id_barang);  
4 $id_barang = 5;  
5  
6 $stmt->execute();  
7  
8 $result = $stmt->get_result();  
9 while ($row = $result->fetch_assoc()) {  
10 echo $row['id_barang'];  
11 // ...  
12 }  
13  
14 $stmt->free_result();  
15 $stmt->close();
```

Baik, cukup dengan teorinya, kita akan lihat contoh kode program dari *prepared statement* menggunakan mysqli object.

Silahkan pelajari sejenak kode berikut ini:

48.mysqli_prepared_select.php

```
1 <?php
```

Mysqli Object

```
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "", "ilkoom");
6
7     // Proses prepare
8     $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");
9
10    // Proses bind
11    $id_barang = 5;
12    $stmt->bind_param("i", $id_barang);
13
14    // Proses execute
15    $stmt->execute();
16
17    // Proses menampilkan hasil query
18    $result = $stmt->get_result();
19    if ($mysqli->error){
20        throw new Exception($mysqli->error, $mysqli->errno);
21    }
22    else {
23        while ($row = $result->fetch_assoc()){
24            echo $row['id_barang'];      echo " | ";
25            echo $row['nama_barang'];   echo " | ";
26            echo $row['jumlah_barang']; echo " | ";
27            echo $row['harga_barang'];  echo " | ";
28            echo $row['tanggal_update'];
29            echo "<br>";
30        }
31    }
32
33    // Hapus memory dan tutup prepared statement
34    $stmt->free_result();
35    $stmt->close();
36 }
37 catch (Exception $e) {
38     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
39 }
40 finally {
41     if (isset($mysqli)) {
42         $mysqli->close();
43     }
44 }
```

Hasil kode program:

```
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41
```

Kode program di atas merupakan rangkuman dari penjelasan kita sebelumnya. Proses **prepare** ada di baris 8, proses **bind** di baris 11-12, dan proses **execute** di baris 15.

Di baris 18 terdapat perintah `$result = $stmt->get_result()`. Variabel `$result` di sini akan berisi **mysqli_result** object hasil pemrosesan *prepared statement*.

Jika anda masih ingat, `mysqli_result` object ini sudah kita bahas cukup detail sebelumnya.

Tidak ada perbedaan antara `mysqli_result` object hasil *prepared statement* dengan `mysqli_result` object hasil `mysqli::query()` biasa. Untuk menampilkan hasilnya saya menggunakan perulangan `while ($row = $result->fetch_assoc())` di baris 23 – 30.

Kita juga bisa menggunakan method lain untuk memproses hasil ini, misalnya memakai `$result->fetch_row()`, `$result->fetch_object()`, atau `$result->fetch_array()`.

Dalam contoh di atas saya juga membuat kode program untuk *error handling*. Ini memperlihatkan bahwa tidak ada perbedaan penanganan error antara *prepared statement* dengan method `query()` biasa.

Bagaimana jika ada 2 data yang diinput? Berikut contohnya:

49. `mysqli_prepared_select_2.php`

```

1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Proses prepare
5 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?
6     OR nama_barang = ?");
7
8 // Proses bind
9 $id_barang = 5;
10 $nama_barang = "TV Samsung 43NU7090 4K";
11 $stmt->bind_param("is", $id_barang, $nama_barang);
12
13 // Proses execute
14 $stmt->execute();
15
16 // Proses menampilkan hasil query
17 $result = $stmt->get_result();
18 while ($row = $result->fetch_assoc()){
19     echo $row['id_barang'];    echo " | ";
20     echo $row['nama_barang'];  echo " | ";
21     echo $row['jumlah_barang']; echo " | ";
22     echo $row['harga_barang'];  echo " | ";
23     echo $row['tanggal_update'];
24     echo "<br>";
25 }
26
27 // Hapus memory dan tutup prepared statement
28 $stmt->free_result();
29 $stmt->close();

```

Hasil kode program:

```

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-07 21:09:41
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41

```

Di sini query *prepared* yang saya pakai adalah "SELECT * FROM barang WHERE id_barang = ?

OR `nama_barang = ?`. Karena ada 2 buah tanda tanya, maka proses **bind** juga butuh 2 buah data inputan, yakni `$id_barang`, `$nama_barang`. Kedua variabel ini di set sebagai `"is"` karena `$id_barang` bertipe integer, dan `$nama_barang` bertipe string.

Salah satu fitur dari *prepared statement* adalah, kita bisa menjalankan query yang sama beberapa kali dengan isian data yang berbeda-beda. Berikut contoh kode programnya:

50.mysqli_prepared_reuse.php

```

1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Buat prepared statement untuk ambil data barang
5 $query = "SELECT * FROM barang WHERE id_barang = ?";
6 $stmt = $mysqli->prepare($query);
7
8 // Proses bind
9 $stmt->bind_param("i", $id_barang);
10
11 // Input data 1
12 $id_barang = 5;
13 $stmt->execute();
14 $result = $stmt->get_result();
15 $row = $result->fetch_row();
16 echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
17 $stmt->free_result();
18
19 echo "<br>";
20
21 // Input data 2
22 $id_barang = 1;
23 $stmt->execute();
24 $result = $stmt->get_result();
25 $row = $result->fetch_row();
26 echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
27 $stmt->free_result();
28
29 // Tutup prepared statement
30 $stmt->close();

```

Di sini saya menulis query **prepare** di baris 5-6, kemudian proses **bind** di baris 9. Perintah untuk bind adalah `$stmt->bind_param("i", $id_barang)`, namun isi variabel `$id_barang` ini belum dibuat.

Pada baris 12, saya mengisi variabel `$id_barang = 2` yang diikuti dengan proses **execute**. Kode program di baris 14 – 16 akan menampilkan hasilnya menggunakan method `$stmt->get_result()`.

Pada baris 22, saya kembali mengisi variabel `$id_barang` dengan nilai 1, kemudian kembali menjalankan method `$stmt->execute()` untuk memproses prepared statement dan menampilkan hasilnya.

Mysqli Object

Dalam kode program ini, perintah query cukup ditulis 1 kali di awal, yakni di baris 5-6. Namun data untuk proses **bind** bisa dijalankan berkali-kali dengan data yang berbeda. Proses ini lebih efisien dibandingkan menjalankan seluruh query secara terpisah.

Sebelumnya juga dijelaskan bahwa kita bisa memproses hasil query **SELECT** dari prepared statement dengan berbagai method **mysqli_result** object, misalnya memakai **mysqli_result::fetch_object()** seperti contoh berikut:

51.mysqli_prepared_fetch_object.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Buat prepared statement untuk ambil data barang
5 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");
6
7 // Proses bind
8 $stmt->bind_param("i", $id_barang);
9 $id_barang = 3;
10
11 // Proses execute
12 $stmt->execute();
13
14 // Proses menampilkan hasil query
15 $result = $stmt->get_result();
16 $row = $result->fetch_object();
17 echo $row->id_barang; echo " | ";
18 echo $row->nama_barang; echo " | ";
19 echo $row->jumlah_barang; echo " | ";
20 echo $row->harga_barang; echo " | ";
21 echo $row->tanggal_update;
22
23 $stmt->free_result();
24 $stmt->close();
```

Hasil kode program:

```
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-07 21:09:41
```

Di sini saya tidak menggunakan perulangan while karena hasil dari query **SELECT** bisa dipastikan hanya berisi 1 baris data saja. Dan karena memakai method **\$row = \$result->fetch_object()**, maka cara pengaksesan data akan menggunakan object, seperti **\$row->id_barang**, dst.

Jika tanpa perulangan while, kita bisa memakai teknik **method chaining** untuk menggabungkan perintah di baris 15 dan 16, menjadi:

52.mysqli_prepared_method_chaining.php

```
1 <?php
2 // Proses menampilkan hasil query
```

Mysqli Object

```
3 $row = $stmt->get_result()->fetch_object();  
4  
5 echo $row->id_barang;      echo " | ";  
6 echo $row->nama_barang;    echo " | ";  
7 echo $row->jumlah_barang;  echo " | ";  
8 echo $row->harga_barang;   echo " | ";  
9 echo $row->tanggal_update;
```

Penulisan prepared statement juga mendukung karakter khusus seperti % atau _ yang sering dipakai untuk proses query SELECT ... LIKE. Berikut contohnya:

53.mysqli_prepared_select_like.php

```
1 <?php  
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");  
3  
4 // Buat prepared statement untuk mencari nama barang  
5 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE nama_barang LIKE ?");  
6  
7 // Proses bind  
8 $stmt->bind_param("s", $nama_barang);  
9 $nama_barang = "%kulkas%";  
10  
11 // Proses execute  
12 $stmt->execute();  
13  
14 // Proses menampilkan hasil query  
15 $result = $stmt->get_result();  
16 while ($row = $result->fetch_assoc()){  
17     echo $row['id_barang'];      echo " | ";  
18     echo $row['nama_barang'];    echo " | ";  
19     echo $row['jumlah_barang'];  echo " | ";  
20     echo $row['harga_barang'];   echo " | ";  
21     echo $row['tanggal_update'];  
22     echo "<br>";  
23 }  
24  
25 $stmt->free_result();  
26 $stmt->close();
```

Hasil kode program:

```
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-07 21:09:41
```

Di sini query yang saya pakai adalah "SELECT * FROM barang WHERE nama_barang LIKE ?", kemudian variabel yang di-bind berupa "%kulkas%", sehingga hasil akhir query menjadi: "SELECT * FROM barang WHERE nama_barang LIKE '%kulkas%'". Perintah query ini akan mencari nama barang yang memiliki kata "kulkas".

Mysqli_stmt object sebenarnya memiliki 1 lagi cara menampilkan data hasil query SELECT, yakni menggunakan perpaduan method `mysqli_stmt::bind_result()` dan `mysqli_stmt::fetch()`.

Method `bind_result()` dan `fetch()` ini sebenarnya sudah tidak terlalu sering dipakai karena kurang fleksibel dibandingkan method `get_result()`. Namun untuk versi PHP 5.3 ke bawah, inilah satu-satunya cara untuk menampilkan hasil query `SELECT` dalam mysqli prepared statement, sehingga mungkin anda akan menemukan penulisan seperti ini di kode PHP lama.

Mari kita bahas dengan contoh kode program berikut ini:

54.mysqli_prepared_fetch.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Buat prepared statement untuk ambil data barang
5 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang = ?");
6
7 // Proses bind
8 $stmt->bind_param("i", $id_barang);
9 $id_barang = 5;
10
11 // Proses execute
12 $stmt->execute();
13
14 // Proses menampilkan hasil query
15 $stmt->bind_result($a, $b, $c, $d, $e);
16 $stmt->fetch();
17
18 echo $a." | ".$b." | ".$c." | ".$d." | ".$e;
19
20 $stmt->free_result();
21 $stmt->close();
```

Hasil kode program:

```
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41
```

Yang perlu kita bahas hanya kode program di baris 15 – 18, yakni proses menampilkan hasil query.

Dalam contoh ini, query yang dijalankan adalah `"SELECT * FROM barang WHERE id_barang = 5"` (query akhir setelah proses `execute`). Tanda bintang `*` artinya saya ingin mengambil semua kolom yang ada di tabel barang, yakni 5 buah kolom.

Karena terdapat 5 kolom, maka pemanggilan method `bind_result()` di baris 15 juga butuh 5 buah argument. Argument ini berupa nama variabel yang akan berpasangan dengan nilai setiap kolom dari tabel barang. Dalam contoh di atas, saya menulis kelima argument dengan nama `$a`, `$b`, `$c`, `$d` dan `$e`.

Kemudian diikuti dengan pemanggilan method `$stmt->fetch()` di baris 17. Hasilnya, variabel `$a` akan berisi nilai yang berasal dari kolom `id_barang`. Kolom `$b` akan berisi nilai dari kolom `nama_barang`, dst. Inilah hasil dari perintah `echo` di baris 18.

Mysqli Object

Agar lebih informatif, argument pada saat pemanggilan method `$stmt->bind_result()` bisa kita isi dengan nama variabel yang sesuai dengan nama kolom tabel:

55.mysqli_prepared_fetch_2.php

```
1 <?php
2 // Proses menampilkan hasil query
3 $stmt->bind_result($id_barang, $nama_barang, $jumlah_barang,
4                     $harga_barang, $tanggal_update);
5 $stmt->fetch();
6
7 echo $id_barang;      echo " | ";
8 echo $nama_barang;    echo " | ";
9 echo $jumlah_barang;  echo " | ";
10 echo $harga_barang;   echo " | ";
11 echo $tanggal_update;
```

Bagaimana jika hasil query terdiri lebih dari 1 baris? Kita bisa memasukkan method `$stmt->fetch()` ke dalam perulangan **while** seperti contoh berikut:

56.mysqli_prepared_fetch_while.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Buat prepared statement untuk ambil data barang
5 $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang <= ?");
6
7 // Proses bind
8 $stmt->bind_param("i", $id_barang);
9 $id_barang = 3;
10
11 // Proses execute
12 $stmt->execute();
13
14 // Proses menampilkan hasil query
15 $stmt->bind_result($id_barang, $nama_barang, $jumlah_barang,
16                     $harga_barang, $tanggal_update);
17
18 while ($stmt->fetch()){
19     echo $id_barang;      echo " | ";
20     echo $nama_barang;    echo " | ";
21     echo $jumlah_barang;  echo " | ";
22     echo $harga_barang;   echo " | ";
23     echo $tanggal_update;
24     echo "<br>";
25 }
26
27 $stmt->free_result();
28 $stmt->close();
```

Hasil kode program:

Mysqli Object

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-07 21:09:41
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-07 21:09:41
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-07 21:09:41
```

Tergantung "kesukaan", anda boleh memakai method `mysqli_stmt::get_result()` atau `mysqli_stmt::bind_result()` untuk menampilkan hasil query `SELECT`. Namun dalam banyak hal method `get_result()` lebih fleksibel dibandingkan method `bind_result()`.

Sepanjang materi tentang prepared statement ini kita baru melihat praktek dari query `SELECT`. Bagaimana dengan query lain seperti `INSERT`, `UPDATE` atau `DELETE`? Prinsipnya sama saja, malah lebih sederhana karena tidak perlu menampilkan hasil query tersebut:

57.mysqli_prepared_insert.php

```
1 <?php
2 $mysqli = new mysqli("localhost", "root", "", "ilkoom");
3
4 // Buat format tanggal hari ini
5 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
6 $timestamp = $sekarang->format("Y-m-d H:i:s");
7
8 // Buat prepared statement untuk input data barang
9 $stmt = $mysqli->prepare("INSERT INTO barang (nama_barang,
10 jumlah_barang, harga_barang, tanggal_update) VALUES (?,?,?,?)");
11
12 // Proses bind
13 $stmt->bind_param("siis", $nama_barang, $jumlah_barang,
14                     $harga_barang, $tanggal_update);
15
16 $nama_barang = "Sharp Microwave Oven R-728(K)";
17 $jumlah_barang = 20;
18 $harga_barang = 1250500;
19 $tanggal_update = $timestamp;
20
21 // Proses execute
22 $stmt->execute();
23 echo "Terdapat ".$mysqli->affected_rows." baris yang ditambah <br>";
24
25 $stmt->close();
```

Hasil kode program:

Terdapat 1 baris yang ditambah

Di baris 5-6 terdapat kode program untuk membuat format tanggal hari ini. Kemudian di baris 9-10 saya mempersiapkan query untuk proses `INSERT`. Karena ini adalah prepared statement, maka semua nilai data diganti dengan tanda tanya.

Di baris 13 – 14 adalah kode untuk proses bind. Argument pertama dari method `bind_param()` adalah "siis" yang mewakili tipe data dari semua nilai input, dimana variabel `$nama_barang` yang bertipe string, `$jumlah_barang` bertipe integer, `$harga_barang` bertipe integer, dan

Mysqli Object

\$tanggal_update bertipe string.

Nilai dari setiap variabel ini ditulis di baris 16 – 19. Yang segera diikuti dengan perintah \$stmt->execute() di baris 22 untuk memproses query INSERT ini.

Di baris 23 saya mengakses property \$mysqli->affected_rows untuk mendapat informasi jumlah perubahan data yang dilakukan oleh proses INSERT.

Proses INSERT menjadi kandidat paling pas untuk proses "bind berulang", dimana kita bisa menginput beberapa data secara terpisah, dengan 1 query prepare. Berikut contoh prakteknya:

58.mysqli_prepared_insert_reuse.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "", "ilkoom");
6
7     // Buat format tanggal hari ini
8     $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
9     $timestamp = $sekarang->format("Y-m-d H:i:s");
10
11    // Buat prepared statement untuk input data barang
12    $stmt = $mysqli->prepare("INSERT INTO barang (nama_barang,
13        jumlah_barang, harga_barang, tanggal_update) VALUES (?, ?, ?, ?)");
14
15    $stmt->bind_param("siis", $nama_barang, $jumlah_barang,
16                      $harga_barang, $tanggal_update);
17
18    // Input data 1
19    $nama_barang = "Cosmos CRJ-8229 - Rice Cooker";
20    $jumlah_barang = 4;
21    $harga_barang = 299000;
22    $tanggal_update = $timestamp;
23
24    $stmt->execute();
25    echo "Terdapat ".$mysqli->affected_rows." baris yang ditambah <br>";
26
27    // Input data 2
28    $nama_barang = "Philips Blender HR 2157";
29    $jumlah_barang = 11;
30    $harga_barang = 629000;
31    $tanggal_update = $timestamp;
32
33    $stmt->execute();
34    echo "Terdapat ".$mysqli->affected_rows." baris yang ditambah <br><br>";
35
36    $stmt->close();
37
38    // Proses prepare untuk menampilkan semua isi tabel barang
39    $stmt = $mysqli->prepare("SELECT * FROM barang WHERE id_barang");
```

Mysqli Object

```
40
41 // Proses execute
42 $stmt->execute();
43
44 // Proses menampilkan hasil query
45 $result = $stmt->get_result();
46 while ($row = $result->fetch_assoc()){
47     echo $row['id_barang'];    echo " | ";
48     echo $row['nama_barang'];  echo " | ";
49     echo $row['jumlah_barang']; echo " | ";
50     echo $row['harga_barang']; echo " | ";
51     echo $row['tanggal_update'];
52     echo "<br>";
53 }
54
55 // Hapus memory dan tutup prepared statement
56 $stmt->free_result();
57 $stmt->close();
58 }
59 catch (Exception $e) {
60     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
61 }
62 finally {
63     if (isset($mysqli)) {
64         $mysqli->close();
65     }
66 }
```

Hasil kode program:

```
Terdapat 1 baris yang ditambah
Terdapat 1 baris yang ditambah
```

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-07 21:09:41
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-07 21:09:41
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-07 21:09:41
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-07 21:09:41
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-07 21:09:41
6 | Sharp Microwave Oven R-728(K) | 20 | 1250500 | 2019-01-09 17:08:40
7 | Cosmos CRJ-8229 - Rice Cooker | 4 | 299000 | 2019-01-09 17:56:27
8 | Philips Blender HR 2157 | 11 | 629000 | 2019-01-09 17:56:27
```

Kode program di atas cukup panjang karena selain menampilkan kode untuk error handling, juga terdapat 2 query *prepared statement*, satu untuk proses input (**INSERT**) dan satu lagi untuk menampilkan data tabel (**SELECT**).

Perintah atau query yang dipakai untuk proses input sama seperti contoh sebelumnya. Hanya saja kali ini method `$stmt->execute()` dipanggil 2 kali untuk 2 data yang berbeda. Kelompok pertama di baris 19 – 25, dan kelompok kedua di baris 28 – 34.

Setelah proses **INSERT**, kemudian disambung dengan pembuatan prepared statement kedua di baris 39. Ini sebenarnya bukanlah prepared statement "asli", karena kita tidak memiliki data

input (tidak ada tanda tanya di dalam query). Karena itu, proses bind tidak diperlukan dan langsung lompat ke proses execute di baris 42.

Di baris 45 – 53 saya membuat perulangan while dari hasil `$stmt->get_result()`. Hasilnya, tampil semua isi tabel barang.

9.4. Mysqli Transaction

Di dalam MySQL, ada fitur yang dinamakan *transaction*. Fitur *transaction* dipakai untuk mengeksekusi beberapa perintah (query) sebagai satu kesatuan. Ketika terjadi masalah atau hal lain, seluruh query bisa dibatalkan atau di proses seluruhnya.

Misalnya untuk proses pembelian, kita harus menambah transaksi di tabel `order` dan mengurangi jumlah stok di tabel `barang`. Kedua query ini harus diproses sekaligus, jika query kedua gagal maka query pertama juga harus dibatalkan. Fitur *transaction* memastikan kedua query (atau lebih) di proses secara bersamaan atau tidak sama sekali.

Lebih lanjut tentang *transaction* ini saya bahas detail di buku **MySQL Uncover**. Intinya, kita memakai query `START TRANSACTION` untuk memulai proses *transaction*, kemudian menulis perintah query seperti biasa. Jika query tersebut tidak ada masalah, jalankan perintah `COMMIT`. Namun jika karena suatu hal kita ingin membatalkan seluruh query, jalankan perintah `ROLLBACK`.

Sebenarnya untuk menjalankan fitur *transaction* bisa langsung dengan menulis perintah query ke dalam method `mysqli::query()`, seperti:

```

1 <?php
2 $mysqli->query("START TRANSACTION");
3
4 // jalankan query Lain seperti biasa
5 // ...
6 // ...
7
8 // jika query OK, permanenkan hasil query
9 $mysqli->query("COMMIT");
10
11 // atau jika query bermasalah, batalkan semua query
12 $mysqli->query("ROLLBACK");

```

Selain itu PHP juga menyediakan beberapa method yang secara khusus dipakai untuk memproses *transaction*.

Untuk memulai proses *transaction*, kita bisa memakai salah satu method berikut:

- ◆ `$mysqli->query("START TRANSACTION");`
- ◆ `$mysqli->begin_transaction();`
- ◆ `$mysqli->autocommit(FALSE);`

Mysqli Object

Untuk "mempermanenkan" hasil transaction bisa menjalankan method:

- ◆ \$mysqli->query("COMMIT");
- ◆ \$mysqli->commit()
- ◆ \$mysqli->autocommit(TRUE);

Dan untuk membatalkan transaction bisa menggunakan method:

- ◆ \$mysqli->query("ROLLBACK");
- ◆ \$mysqli->rollback();

Berikut contoh kode program dari sebuah proses *transaction*:

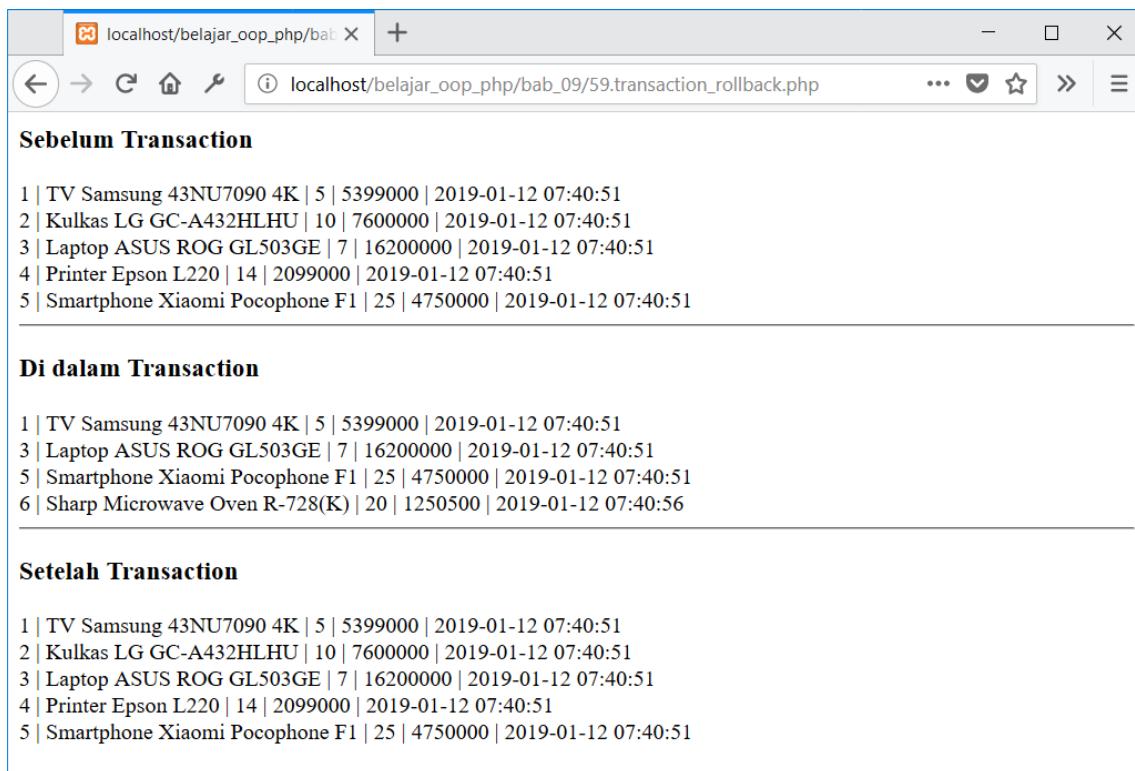
Seperti biasa, karena kita akan menggunakan tabel **barang**, silahkan reset ulang tabel ini dengan cara menjalankan file 20.mysqli_generate.php.

59.transaction_rollback.php

```
1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 // Buat format tanggal hari ini
5 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
6 $timestamp = $sekarang->format("Y-m-d H:i:s");
7
8 try {
9     $mysqli = new mysqli("localhost", "root", "", "ilkoom");
10
11    // Tampilkan isi tabel sebelum transaction
12    echo "<h3>Sebelum Transaction</h3>";
13    $result = $mysqli->query("SELECT * FROM barang");
14    while ($row = $result->fetch_array(MYSQLI_NUM)){
15        echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
16        echo "<br>";
17    }
18    echo "<hr>";
19
20    // Mulai transaction
21    $mysqli->begin_transaction();
22    $mysqli->query("DELETE FROM barang WHERE id_barang = 2");
23    $mysqli->query("DELETE FROM barang WHERE id_barang = 4");
24    $mysqli->query("INSERT INTO barang VALUES (NULL,
25                      'Sharp Microwave Oven R-728(K)',20,1250500,'$timestamp')");
26
27    // Tampilkan isi tabel selama transaction
28    echo "<h3>Di dalam Transaction</h3>";
29    $result = $mysqli->query("SELECT * FROM barang");
30    while ($row = $result->fetch_array(MYSQLI_NUM)){
31        echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
32        echo "<br>";
33    }
34    echo "<hr>";
```

Mysqli Object

```
35 // Batalkan query transaction
36 $mysqli->rollback();
37
38 // Tampilkan isi tabel di setelah transaction
39 echo "<h3>Setelah Transaction</h3>";
40 $result = $mysqli->query("SELECT * FROM barang");
41 while ($row = $result->fetch_array(MYSQLI_NUM)){
42     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
43     echo "<br>";
44 }
45 }
46 }
47 catch (Exception $e) {
48     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
49 }
50 finally {
51     if (isset($mysqli)) {
52         $mysqli->close();
53     }
54 }
```



Gambar: Hasil dari proses transaction yang di rollback

Kode di atas cukup panjang karena menulis perintah untuk menampilkan isi tabel barang di 3 tempat, yakni sebelum transaction, selama transaction dan setelah proses transaction.

Di awal program, terdapat kode untuk meng-generate tanggal hari ini (baris 5-6), yang nantinya akan dipakai untuk query `INSERT`.

Setelah pembuatan koneksi di baris 9, kode program di baris 12 – 18 dipakai untuk

menampilkan isi tabel barang. Di sini saya menjalankan query "SELECT * FROM barang" lalu memakai perulangan while (`$row = $result->fetch_array(MYSQLI_NUM)`) untuk menampilkan isi tabel barang menggunakan index numerik (normal array).

Dari hasil tampilan, terlihat isi tabel barang terdiri dari 5 baris. Nantinya kita akan periksa ulang selama di dalam transaction dan setelah proses transaction.

Di baris 21 terdapat pemanggilan method `$mysqli->begin_transaction()`. Ini adalah perintah untuk memulai transaction. Apapun query yang akan kita jalankan setelah perintah ini belum permanen dan hanya di proses di memory.

Di baris 22 dan 23, saya menjalankan query `DELETE` untuk menghapus data dari tabel barang untuk `id_barang` 2 dan 4. Kemudian di baris 24 terdapat query `INSERT` yang akan menambah sebuah data baru ke dalam tabel barang.

Isi tabel barang akan ditampilkan dengan kode program di baris 28 – 34. Kode ini saya seperti yang saya pakai di awal program. Hasilnya, di tabel barang hanya ada 4 baris. Barang "Kulkas LG" dan "Printer Epson" sudah tidak ada lagi, dan terdapat barang baru "Sharp Microwave". Inilah hasil dari query `DELETE` dan `INSERT` yang kita jalankan sebelumnya.

Namun karena sesuatu hal, saya berubah pikiran dan menjalankan perintah `$mysqli->rollback()` di baris 37. Perintah ini akan membatalkan semua query yang dijalankan sejak method `$mysqli->begin_transaction()` di baris 21. Hasilnya, isi tabel barang kembali seperti semula. Inilah efek dari penggunaan transaction.

Jika kita ingin hasil query menjadi permanen, tinggal ganti perintah `$mysqli->rollback()` di baris 37 menjadi `$mysqli->commit()`.

Dalam contoh ini saya tidak memakai prepared statement, namun prinsip dasarnya tetap sama. Method `begin_transaction()`, `rollback()`, dan `commit()` "melekat" ke **mysqli** object, bukan **mysqli_stmt** object. Sehingga jika kita menggunakan prepared statement, yang akan berubah hanyalah di pemanggilan query baris 22 – 24, yakni ketika menjalankan query "normal" seperti `SELECT`, `INSERT`, `UPDATE` dan `DELETE`.

Dalam bab ini kita telah membahas cukup detail tentang menampilkan data dari database MySQL menggunakan **mysqli object**. Selain itu juga dibahas tentang *prepared statement* sebagai alternatif penulisan query yang lebih aman.

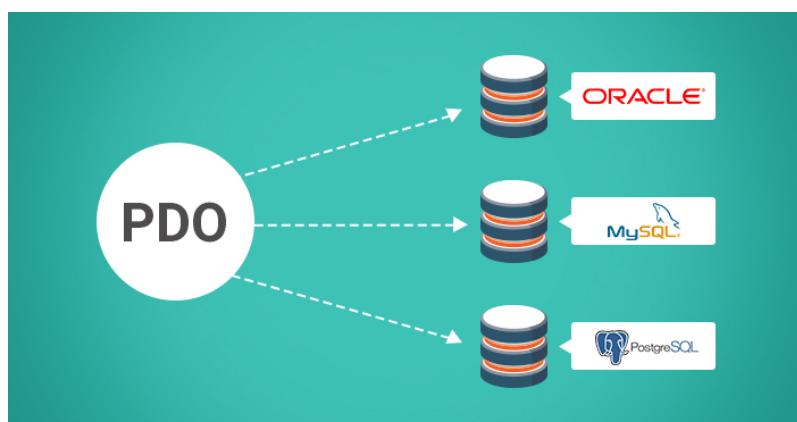
Berikutnya kita juga masih membahas cara pengaksesan tabel MySQL, namun kali ini menggunakan **PDO** (PHP Data Objects).

10. PDO

Sebagai alternatif dari **mysqli**, PHP menyediakan **PDO** untuk mengakses database. Selain lebih fleksibel (karena bisa dipakai untuk mengakses berbagai jenis aplikasi database), PDO juga menyediakan beberapa fitur yang tidak tersedia di mysqli. Dalam bab ini kita akan kupas secara mendalam apa itu PDO serta bagaimana cara penggunaannya.

10.1. Pengertian PDO

PDO (singkatan dari **PHP Data Object**), adalah object yang berfungsi untuk berkomunikasi dengan database. PDO ini mirip seperti **mysqli** yang baru saja kita bahas, akan tetapi PDO bersifat universal, yakni bisa dipakai untuk mengakses berbagai aplikasi database mulai dari MySQL / MariaDB, SQLite, Microsoft SQL Server, Oracle, PostgreSQL, dll.



Ilustrasi PDO (sumber gambar: cloudways.com)

Secara teknis, ketika kita menggunakan **mysqli** extension, PHP langsung terhubung dengan MySQL Server. Tetapi jika menggunakan PDO, terdapat 1 lapisan (*layer*) di atasnya. PDO hadir sebagai antar muka (*interface*) universal yang menyediakan 1 cara untuk mengakses berbagai jenis database.

Konsep PDO ini dapat digambarkan sebagai berikut:

PHP PDO → Database Driver → Database Server

PDO bekerja dengan metode yang disebut "**data-access abstraction layer**". Artinya, apapun jenis database server yang digunakan, kode PHP yang ditulis tetap sama. PDO lah yang akan menerjemahkan kode tersebut agar bisa dipahami aplikasi database tujuan. Dengan mempelajari cara penggunaan PDO, secara otomatis kita juga bisa membuat kode program

untuk berbagai jenis database.

Keunggulan utama PDO ada di satu cara universal dalam mengakses berbagai jenis database, bukan untuk berpindah dari satu jenis database ke database lain.

Yang harus dipahami adalah, setiap aplikasi database punya fitur unik yang tidak dimiliki oleh database lain. Jadi meskipun terdapat kode program yang menggunakan PDO, tetapi bukan cara yang mudah untuk berpindah dari satu database server ke database server lain (terutama di aplikasi yang sudah jadi).

Di PHP 8, PDO mendukung setidaknya 11 jenis Interface/Database Server:

1. CUBRID
2. MS SQL Server
3. Firebird
4. IBM
5. Informix
6. MySQL
7. MS SQL Server
8. Oracle
9. ODBC and DB2
10. PostgreSQL
11. SQLite

Daftar lengkapnya bisa dilihat ke: [PDO Drivers](#).

10.2. Mengaktifkan PDO Extension

PDO telah aktif secara default di PHP versi 5.1 ke atas, tetapi tidak semua database driver bisa dipakai. Karena alasan performa, PHP me-nonaktifkan mayoritas PDO database driver seperti Oracle atau PostgreSQL.

Untuk melihat driver database apa saja yang telah aktif, jalankan static method `PDO::getAvailableDrivers()`:

```
01.pdo_getavailabledrivers.php
1 <?php
2     print_r(PDO::getAvailableDrivers());
```

Berikut hasil yang saya dapat:

```
Array ( [0] => mysql [1] => sqlite )
```

Terlihat bahwa driver PDO yang aktif hanya ada 2, yakni **MySQL** dan **SQLite**. Artinya, kita

hanya bisa memakai PDO untuk mengakses kedua database ini saja. Bagaimana dengan yang lain? harus diaktifkan dari file konfigurasi PHP: `php.ini`.

Jika anda menginstall XAMPP di drive C, lokasi file `php.ini` ada di `C:\xampp\php\php.ini`.

Silahkan buka dengan aplikasi text editor, kemudian cari kata "pdo". Pada versi PHP yang saya gunakan, pengaturannya ada di baris 900-an:

```

895 | extension=mbstring
896 | extension=exif      ; Must be after mbstring as it depends on it
897 | extension=mysqli
898 | ;extension=oci8_12c ; Use with Oracle Database 12c Instant Client
899 | ;extension=odbc
900 | ;extension=openssl
901 | ;extension=pdo_firebird
902 | extension=pdo_mysql
903 | ;extension=pdo_oci
904 | ;extension=pdo_odbc
905 | ;extension=pdo_pgsql
906 | extension=pdo_sqlite
907 | ;extension=pgsql

```

Pengaturan PDO driver extension

Dari gambar di atas, pengaturan PDO *driver extension* ada di baris 902 – 906, yakni baris yang diawali dengan "extension=pdo_ ", inilah driver database PDO yang tersedia di PHP. Terlihat driver yang telah aktif hanya `pdo_mysql` dan `pdo_sqlite`.

Untuk mengaktifkan sebuah driver, hapus tanda titik koma (;) di awal baris. Sebagai contoh, saya akan mengaktifkan `extension=pdo_pgsql` yang merupakan driver untuk database

PostgreSQL:

```

895 | extension=mbstring
896 | extension=exif      ; Must be after mbstring as it depends on it
897 | extension=mysqli
898 | ;extension=oci8_12c ; Use with Oracle Database 12c Instant Client
899 | ;extension=odbc
900 | ;extension=openssl
901 | ;extension=pdo_firebird
902 | extension=pdo_mysql
903 | ;extension=pdo_oci
904 | ;extension=pdo_odbc
905 | extension=pdo_pgsql
906 | extension=pdo_sqlite
907 | ;extension=pgsql

```

Aktifkan `pdo_pgsql` driver extension

Save file `php.ini`, kemudian restart web server Apache (matikan dan hidupkan kembali melalui XAMPP Control Panel).

Untuk memastikan apakah driver telah aktif atau belum, jalankan kembali method `PDO::getAvailableDrivers()` dan berikut adalah hasil yang saya dapat:

```
Array ( [0] => mysql [1] => pgsql [2] => sqlite )
```

Terlihat, driver **PostgreSQL** untuk PDO telah aktif.

Meskipun driver PDO untuk sebuah database telah aktif, tetap tidak bisa langsung

dipakai karena kita juga harus menginstall aplikasi database tersebut.

Sebagai contoh, agar bisa mengakses database PostgreSQL dari PHP, kita harus menginstall aplikasi PostgreSQL server terlebih dahulu dan menjalankannya.

Karena keterbatasan tempat untuk membahas cara instalasi PostgreSQL (serta aplikasi database lain), dalam materi ini kita hanya membahas cara pemakaian PDO untuk database MySQL atau MariaDB saja.

10.3. Membuat PDO Object

Dalam banyak hal, cara kerja PDO hampir sama dengan mysqli versi object, dimana kita membuat **PDO object** sebagai penghubung dengan database (dikenal sebagai "database handler"), lalu menjalankan berbagai method dari object ini.

Berikut format dasar pembuatan PDO object (PDO constructor):

```
PDO::__construct(string $dsn[, string $username[, string $passwd[, array $options]]])
```

Terdapat 4 argument yang bisa kita isi pada saat pembuatan PDO object:

- ◆ **\$dsn**: berisi data **DSN**, yakni informasi seputar database server (akan kita bahas sesaat lagi).
- ◆ **\$username**: nama user yang akan mengakses database, misalnya `root`.
- ◆ **\$passwd**: berisi password dari **\$username**.
- ◆ **\$options**: berbagai pengaturan tambahan dalam bentuk array.

Selain **\$dsn**, tiga argumen lain bersifat opsional dan kadang tidak diperlukan untuk driver database tertentu. Misalnya untuk koneksi ke database SQLite tidak perlu menginput **\$username** dan **\$passwd**.

Argument pertama pada saat pembuatan PDO object adalah **DSN** (singkatan dari **Data Source Name**). DSN ini berbentuk string dengan format:

```
nama_driver_pdo:<pengaturan_khusus_driver>
```

Nama driver PDO adalah nama driver yang di dapat dari hasil pemanggilan method `PDO::__getAvailableDrivers()`. Misalnya untuk database MySQL, nama drivernya adalah "`mysql`", sedangkan untuk PostgreSQL, nama drivernya adalah "`pgsql`".

Setelah nama driver, disambung dengan tanda titik dua ":" , kemudian diikuti dengan pengaturan lain tergantung driver yang dipakai.

Untuk MySQL, informasi yang harus dicantumkan adalah alamat komputer (**host**) tempat MySQL Server berada, yang dalam contoh kita berupa `localhost`. Pengaturan opsional lain

berupa nama port (port), nama database (dbname), serta character set (charset) yang akan dipakai.

Pengaturan ini ditulis dalam format `nama_pengaturan=nilai`, yang dipisah dengan tanda titik koma. Berikut contoh penulisan DSN lengkap untuk koneksi ke database MySQL:

```
mysql:host=localhost;port=3306;dbname=ilkoom;charset=utf8mb
```

DSN di atas bisa dibaca: "Akses driver `mysql` di `localhost` dengan nomor port `3306`, kemudian langsung pakai database bernama `ilkoom` dengan character set `utf8mb4`".

Tidak semua pengaturan ini harus ditulis, minimal hanya perlu alamat host saja. Nama database juga tidak harus ditentukan di awal, serta nomor port dan charset bisa memakai nilai default bawaan MySQL.

Dengan demikian, berikut contoh pembuatan **PDO** object untuk mengakses database MySQL:

02.pdo_connect.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost", "root", "");
3 var_dump($pdo); // object(PDO)#1 (0) { }
```

Variabel `$pdo` saya pakai sebagai penampung dari **PDO** object. Nama variabel ini boleh bebas, tidak harus `$pdo`. PDO object ini berisi koneksi ke database MySQL yang ada di `localhost`, kemudian masuk sebagai user `root` dengan password kosong.

Perintah `var_dump($pdo)` di baris 3 saya tambah untuk melihat bahwa variabel `$pdo` berisi sebuah `object(PDO)`.

Jika kita ingin menulis DSN lengkap juga tidak masalah:

03.pdo_connect_full_DSN.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;port=3306;dbname=ilkoom;charset=utf8mb4",
3                 "root", "");
4 var_dump($pdo); // object(PDO)#1 (0) { }
```

Kode di atas artinya saya ingin masuk ke MySQL server yang berada di `localhost` dengan nomor port `3306`, langsung memakai database `ilkoom`, menggunakan charset `utf8mb4`, serta login sebagai user `root` dengan password kosong.

Karena cukup panjang, penulisan argument untuk PDO bisa dipisah menjadi variabel tersendiri agar lebih rapi:

04.pdo_connect_variable.php

```
1 <?php
2 $host    = "127.0.0.1";
3 $port    = "3306";
```

PDO

```
4 $db      = "ilkoom";
5 $charset = "utf8mb4";
6 $user    = "root";
7 $pass    = "";
8
9 $dsn = "mysql:host=$host;port=$port;dbname=$db;charset=$charset";
10 $pdo = new PDO($dsn, $user, $pass);
```

Dari segi isi, tidak ada perbedaan dengan contoh kita sebelumnya. Hanya saja kali ini pembuatan PDO object tampak lebih rapi karena dipecah menjadi variabel-variabel.

Namun demi menghemat tempat, dalam contoh selanjutnya saya "terpaksa" memakai versi yang 1 baris saja.

Walaupun kita hanya membahas cara penggunaan PDO ke database MySQL, berikut contoh pembuatan koneksi untuk database **Microsotf SQL Server**, **Sybase** dan **SQLite**:

```
$pdo = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
$pdo = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");
$pdo = new PDO("sqlite:my/database/path/database.db");
```

Terlihat bahwa argument pembuatan PDO object bisa berbeda antar aplikasi database. Untuk SQLite tidak perlu menulis user dan password, tapi cukup alamat ke lokasi file database SQLite saja.

Setelah PDO object selesai digunakan, kita bisa menutupnya dengan mengisi nilai **NULL** ke dalam variabel penampung PDO, seperti contoh berikut:

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost", "root", "");
3 //...
4 //...
5 $pdo = NULL;
```

Yup, PDO tidak menyediakan method khusus untuk menutup koneksi seperti halnya **mysqli::close()**, jadi terpaksa dibuat manual dengan cara mengisi nilai **NULL**.

Proses pemberian nilai **NULL** ini sebenarnya juga tidak harus ditulis karena PHP otomatis menutup koneksi begitu halaman selesai di proses.

10.4. Error handling PDO Object

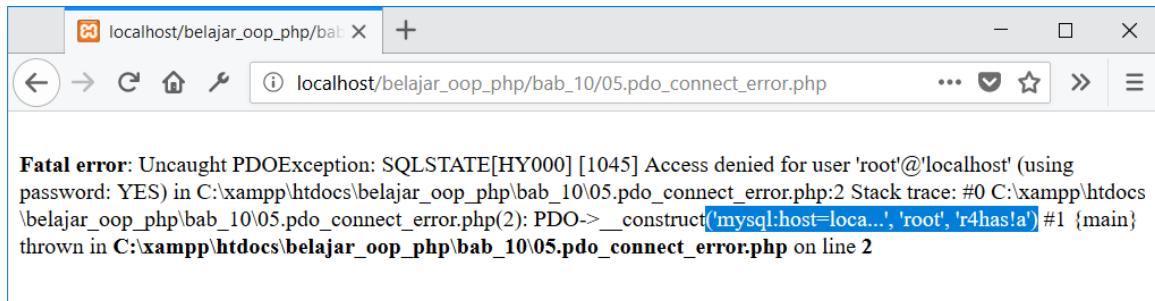
Penanganan error pada saat pembuatan PDO object butuh perhatian khusus. Karena secara default jika koneksi ke database tidak bisa di proses, PHP akan menampilkan pesan error yang di dalamnya terdapat seluruh informasi, termasuk nama user dan password!

Berikut contoh kasusnya:

PDO

05 pdo_connect_error.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "r4has!a");
```



Tampilan pesan error pada saat pembuatan PDO Object

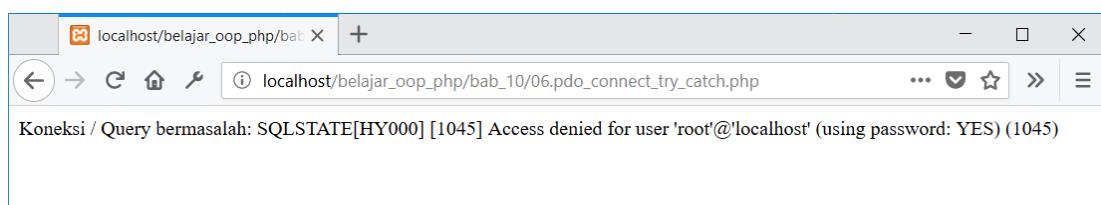
Pada saat pembuatan PDO object, saya mengisi password user root dengan string "r4has!a". Hasilnya tampil error karena user root seharusnya tanpa password. Namun yang jadi masalah adalah, dalam pesan error ini terlihat informasi sensitif berupa semua argument pembuatan PDO object. Tentu saja ini sangat berbahaya.

Solusi yang paling pas adalah mematikan *error reporting* agar semua pesan error tidak lagi bisa terlihat. Ini bisa dilakukan dari file `php.ini` atau dengan menjalankan fungsi `error_reporting(0)` di baris paling atas file PHP. Namun pilihan ini baru pas dilakukan saat aplikasi yang kita buat sudah selesai.

Alternatif lain adalah menggunakan block **try – catch**. Jika anda perhatikan, pesan error di atas diawali dengan `Fatal error: Uncaught PDOException`. Ini adalah pesan error ketika sebuah exception tidak ditangkap. Artinya, secara bawaan PDO sudah "melempar" sebuah exception jika terjadi error. Kita tinggal menangkap `PDOException` ini:

06 pdo_connect_try_catch.php

```
1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "r4has!a");
4 }
5 catch (\PDOException $e) {
6     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
7 }
8 finally {
9     $pdo=NULL;
10 }
```



Tampilan pesan error PDO dengan exception handling

Dengan menggunakan block try-catch, kita bisa mengontrol hasil tampilan error. Dalam contoh ini saya hanya menampilkan informasi dari `$e->getMessage()` yang berisi pesan error dan `$e->getCode()` yang berisi nomor kode error. Ini sama seperti yang kita pakai dalam bab **mysqli** object sebelumnya.

Class exception yang dihasilkan pada saat pembuatan PDO bernama `PDOException`, awalan tanda "\\" pada block catch (`\PDOException $e`) adalah kode untuk *global namespace*. Dalam contoh ini sebenarnya tidak perlu ditulis karena kita sedang berada di global namespace (saya tidak menggunakan namespace apapun). Jika berada di dalam namespace, awalan "\\" ini perlu ditulis.

10.5. Menjalankan Query dengan method PDO::exec()

Proses pembuatan koneksi dengan database sudah selesai, selanjutnya kita akan bahas cara menjalankan perintah query MySQL.

Terdapat beberapa method yang bisa dipakai untuk menjalankan query:

- ◆ `PDO::exec()`
- ◆ `PDO::query()`
- ◆ `PDO::prepare()` dan `PDO::execute()`

Method pertama, yakni `PDO::exec()` hanya bisa dipakai untuk query yang tidak mengembalikan hasil, seperti query `INSERT`, `UPDATE` dan `DELETE`.

Method kedua, yakni `PDO::query()` lebih fleksibel karena bisa dipakai untuk memproses hasil dari query `SELECT`, serta query lain seperti `INSERT`, `UPDATE` dan `DELETE`.

Dan method ketiga, yakni `PDO::prepare()` dan `PDO::execute()` dipakai untuk memproses *prepared statement*.

Kita akan bahas ketiga method ini yang dimulai dari `PDO::exec()` terlebih dahulu.

Seperti yang disinggung sebelumnya, method `PDO::exec()` hanya bisa dipakai untuk query yang tidak butuh menampilkan hasil, artinya kita tidak bisa memakai method ini untuk memproses query `SELECT`.

Method `PDO::exec()` butuh sebuah argument berupa perintah query yang akan dijalankan. Kemudian method ini mengembalikan salah satu dari 2 nilai:

- ◆ **False**, jika perintah query yang ditulis terdapat error.
- ◆ **Angka integer**, berisi jumlah baris yang dipengaruhi oleh perintah query (*affected rows*).

Berikut contoh penggunaannya:

Sepanjang bab ini saya masih memakai tabel barang yang kita buat pada bab sebelumnya.
Silahkan reset ulang dengan menjalankan file bab09\20.mysqli_generate.php.

07 pdo_exec_update.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "UPDATE barang SET jumlah_barang = 100 WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6     if ($count !== FALSE) {
7         echo "Query Ok, ada $count baris yang di update";
8     }
9 }
10 catch (\PDOException $e) {
11     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
12 }
13 finally {
14     $pdo=NULL;
15 }
```

Hasil kode program:

Query Ok, ada 1 baris yang di update

Di baris 3 saya membuat PDO object yang langsung mengakses database ilkoom.

Kemudian di baris 4 terdapat pendefinisian variabel \$query yang berisi perintah "UPDATE barang SET jumlah_barang = 100 WHERE id_barang = 3". Artinya, saya ingin mengubah data jumlah_barang menjadi 100 untuk baris yang memiliki id_barang = 3.

Perintah \$pdo->exec(\$query) dipakai untuk menjalankan query, yang hasilnya di tumpung ke dalam variabel \$count. Variabel \$count ini akan berisi FALSE jika terdapat error, atau jumlah affected rows.

Di baris 6-8 terdapat block if(\$count !== FALSE). Kondisi ini hanya akan bernilai TRUE jika variabel \$count berisi nilai selain FALSE, yang berarti query berhasil di proses. Isi dari block if sendiri berupa string yang menampilkan jumlah baris terdampak dari hasil query (*affected rows*).

Kita tidak bisa memakai kondisi if(\$count) saja karena ada kemungkinan isi variabel \$count bernilai 0 yang akan dikonversi PHP menjadi FALSE. Ini terjadi jika query yang dijalankan tidak berdampak apa-apa, misalnya ketika kita menghapus baris yang tidak ada, atau mengupdate baris dengan nilai baru yang sama.

Jika kode di atas dijalankan ulang, hasilnya berupa:

Query Ok, ada 0 baris yang di update

Karena meskipun query UPDATE berhasil di jalankan, tapi tidak ada perubahan nilai di tabel barang.

Sebagai contoh kedua, saya ingin menjalankan query DELETE menggunakan method PDO::exec():

08 pdo_exec_delete.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELETE FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6     if ($count !== FALSE) {
7         echo "Query Ok, ada $count baris yang di hapus";
8     }
9 }
10 catch (\PDOException $e) {
11     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()."";
12 }
13 finally {
14     $pdo=NULL;
15 }
```

Hasil kode program:

Query Ok, ada 1 baris yang di hapus

Di sini saya ingin menghapus baris dengan `id_barang = 3` dari tabel barang. Selain perubahan perintah query, tidak ada perbedaan dengan contoh kita sebelumnya.

10.6. Error Handling Query

Salah satu perubahan di PHP 8 berhubungan dengan tampilan pesan error query di PDO.

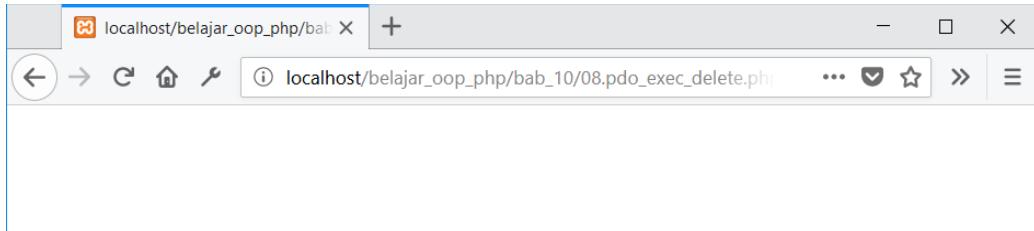
Sebelumnya di PHP 7, error query PDO tidak langsung tampil (harus kita aktifkan secara manual). Sedangkan di PHP 8 secara default pesan error PDO sudah langsung tampil. Perbedaan ini sebenarnya tidak terlalu berdampak karena tetap bisa diatur sesuai keinginan.

Oleh karena itu bagi yang sudah menggunakan PHP 8, bahasan ini boleh dilewati. Bagian ini khusus bagi yang memakai PHP 7 atau PHP 5.6 saja.

Di PHP 7, secara default PDO menyembunyikan pesan error query. Untuk membuktikan, silahkan tukar isi variabel `$query` dari kode program sebelumnya menjadi:

```
$query = "DELET FROM barang WHERE id_barang = 3";
```

Query ini seharusnya error karena di MySQL tidak terdapat perintah "DELET". Bagaimana hasilnya?



Tampilan default PDO jika terjadi query error

Tidak tampil pesan error apapun! Ini merupakan kasus yang sama seperti di **mysqli** object. Tentu saja hal ini bisa membuat pusing karena kita tidak tau apa yang menyebabkan error.

Jika menggunakan PHP 8, kode di atas akan menampilkan error.

Di dalam PDO, ketika sebuah query tidak berhasil dijalankan (error), informasi mengenai pesan error bisa diakses dari method `PDO::errorCode()` dan `PDO::errorInfo()`. Berikut hasil dari pemanggilan kedua method ini:

09 pdo_exec_error_info.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELET FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6
7     echo "<pre>";
8     var_dump($pdo->errorCode());
9     var_dump($pdo->errorInfo());
10    echo "</pre>";
11
12    if ($count !== FALSE) {
13        echo "Query Ok, ada $count baris yang dihapus";
14    }
15 }
16 catch (\PDOException $e) {
17     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
18 }
19 finally {
20     $pdo=NULL;
21 }
```

Hasil kode program:

```

string(5) "42000"
array(3) {
[0]=>
string(5) "42000"
```

```
[1]=>
int(1064)
[2]=>
string(185) "You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near 'DELET
FROM barang WHERE id_barang = 3' at line 1"
}
```

Kembali, jika kode ini dijalankan pada PHP 8, hasilnya sedikit berbeda. Tampilan di atas hanya terlihat di PHP 7 dan PHP 5.6.

Di baris 8 dan 9 saya menggunakan perintah `var_dump()` untuk melihat hasil dari `$pdo->errorCode()` dan `$pdo->errorInfo()`.

Method `$pdo->errorCode()` menghasilkan kode error dalam format `SQLSTATE`, yakni 5 digit kode error alfanumerik. Kode ini bersifat universal untuk semua aplikasi database. Jika anda tertarik, arti dari setiap kode `SQLSTATE` bisa diakses ke en.wikipedia.org/wiki/SQLSTATE.

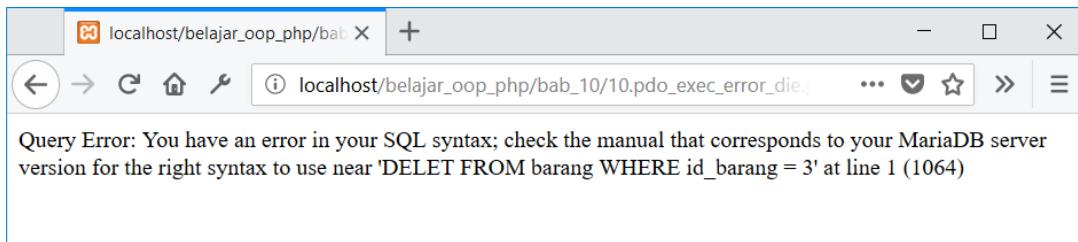
Method `$pdo->errorInfo()` mengembalikan nilai dalam bentuk array 3 element:

1. Element pertama berisi kode `SQLSTATE`, yakni sama dengan hasil pemanggilan `$pdo->errorCode()`.
2. Element kedua berisi nomor error milik MySQL.
3. Element ketiga berisi keterangan error dari MySQL.

Dari ketiga element ini, informasi error yang perlu kita ketahui adalah kode error MySQL dan pesan error MySQL, yakni element ke-2 dan ke-3. Berikut revisi kode program sebelumnya:

10.pdo_exec_error_die.php

```
1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELETE FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6
7     if ($count !== FALSE) {
8         echo "Query Ok, ada $count baris yang dihapus";
9     }
10    else {
11        die("Query Error: ".$pdo->errorInfo()[2]." (".$pdo->errorInfo()[1].")");
12    }
13 }
14 catch (\PDOException $e) {
15     echo "Koneksi / Query bermasalah: ".$e->getMessage()." (".$e->getCode().")";
16 }
17 finally {
18     $pdo=NULL;
19 }
```



Hasil error dari PDO

Di baris 10 – 12 saya menambah block **else** dari kondisi `if($count !== FALSE)`. Artinya, blok **else** hanya dijalankan jika terdapat error di penulisan query. Isinya berupa fungsi `die()` untuk menghentikan kode program dan menampilkan pesan error yang tersimpan di `$pdo->errorInfo()[2]` serta kode error di `$pdo->errorInfo()[1]`.

Jika anda mengikuti materi di bab sebelumnya tentang **mysqli**, saya yakin bisa menebak arah dari kode pembahasan kita. Yup... daripada menggunakan fungsi `die()`, lebih baik pesan error ini diproses sebagai sebuah **exception**. Lagi pula kita sudah memiliki block kode **catch** di akhir kode program:

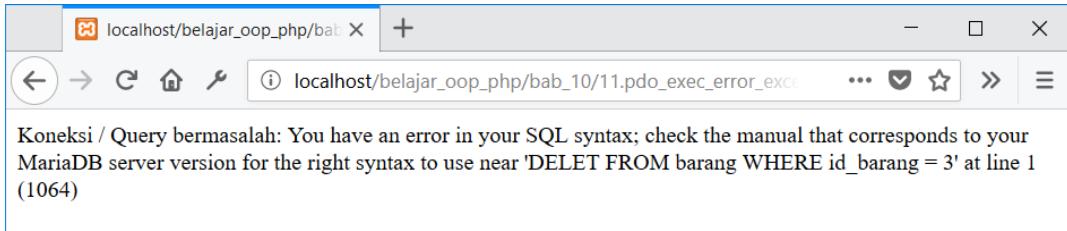
11.pdo_exec_error_exception.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELET FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6
7     if ($count !== FALSE) {
8         echo "Query Ok, ada $count baris yang dihapus";
9     }
10    else {
11        throw new Exception($pdo->errorInfo()[2], $pdo->errorInfo()[1]);
12    }
13 }
14 catch (\Exception $e) {
15     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
16 }
17 finally {
18     $pdo=NULL;
19 }
```

Sekarang di dalam block **else** terdapat perintah untuk membuat exception. Sebagai argument, diinput pesan error yang tersimpan di `$pdo->errorInfo()[2]` dan `$pdo->errorInfo()[1]`.

Di baris 14 saya menukar kondisi **catch** dari sebelumnya `catch(\PDOException $e)` menjadi `catch(\Exception $e)`. Ini bertujuan agar block **catch** bersifat global dan bisa menangkap semua exception.



Hasil error PDO dengan exception yang dibuat manual

Harap dibedakan bahwa exception yang kita buat di sini dipakai untuk menampilkan **pesan kesalahan dari perintah query**. Sedangkan exception yang kita bahas di awal bab adalah pesan kesalahan pada saat **pembuatan PDO object**.

10.7. Pengaturan PDO dengan method PDO::setAttribute()

Cara menampilkan pesan error query yang kita bahas sebelumnya secara khusus ditujukan untuk PHP 7 ke bawah. Sedangkan di PHP 8 tidak perlu lagi karena exception sudah "dilempar" secara otomatis.

Perbedaan ini berkaitan dengan pengaturan PDO yang bisa di-set dari method khusus bernama `PDO::setAttribute()`. Method `PDO::setAttribute()` butuh 2 argument, yakni aturan yang ingin diubah serta nilai dari aturan tersebut.

Sebagai contoh, berikut perintah agar PDO memproses pesan error query sebagai exception:

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Argument pertama, yakni `PDO::ATTR_ERRMODE` berisi keterangan bahwa kita ingin mengubah pengaturan **error mode** di PDO. Terdapat 3 konstanta nilai yang bisa dipilih untuk

`PDO::ATTR_ERRMODE`:

- ◆ `PDO::ERRMODE_SILENT`
- ◆ `PDO::ERRMODE_WARNING`
- ◆ `PDO::ERRMODE_EXCEPTION`

`PDO::ERRMODE_SILENT` adalah pilihan default di PHP 7, dimana PDO "menyembunyikan" semua pesan error. Untuk menampilkannya, kita harus akses dari method `PDO::errorCode()` dan `PDO::errorInfo()` seperti contoh sebelumnya.

`PDO::ERRMODE_WARNING` dipakai untuk menampilkan error sebagai pesan *warning*, kemudian PHP akan lanjut memproses kode program berikutnya.

`PDO::ERRMODE_EXCEPTION` dipakai untuk menampilkan error sebagai exception. Inilah yang menjadi pengaturan default di PHP 8 sehingga error query sudah langsung tampil.

Dari ketiga pilihan ini, sebaiknya set ke `PDO::ERRMODE_EXCEPTION` agar jika terjadi error query,

PDO otomatis melempar sebuah exception. Berikut contoh prakteknya:

12.pdo_exec_error_set_exception.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "DELETE FROM barang WHERE id_barang = 3";
7     $count = $pdo->exec($query);
8
9     if ($count !== FALSE) {
10        echo "Query Ok, ada $count baris yang dihapus";
11    }
12 }
13 catch (\PDOException $e) {
14     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
15 }
16 finally {
17     $pdo=NULL;
18 }
```

Di baris 4 terdapat pemanggilan method `$pdo->setAttribute()` yang berisi pengaturan agar PDO memproses pesan error query sebagai exception.

Dengan demikian, kita tidak butuh lagi kode untuk membuat exception secara manual di PHP. Jika query yang dijalankan method `$pdo->exec()` salah ketik atau tidak dipahami oleh MySQL, PDO langsung melempar sebuah exception.

Exception yang dilempar berasal dari class `\PDOException`, yakni sama seperti exception yang dipakai saat pembuatan PDO object. Sehingga kita bisa kembali menulis block **catch** sebagai `catch(\PDOException $e)` di baris 13.

Sebagai alternatif penulisan, pengaturan PDO ini juga bisa ditulis sebagai argument ke-4 pada saat pembuatan PDO object (PDO constructor). Berikut cara penulisannya:

13.pdo_exec_error_set_exception_constuctor.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "",
4                     [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
5 // ...
6 // ...
```

Argument ke-4 ini ditulis dalam bentuk *associative array*, yakni dengan format `<nama_pengaturan> => <nilai_pengaturan>`. Jika terdapat beberapa pengaturan, pisah dengan tanda koma sebagaimana sebuah array:

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "",
4                     [ PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
5                     PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC ]);
6 // ...
7 // ...

```

Maksud dari `PDO::ATTR_DEFAULT_FETCH_MODE` akan kita bahas sesaat lagi. Untuk sementara yang perlu dipahami adalah bahwa pengaturan PDO bisa ditulis sebagai argument ke-4 pada saat pembuatan PDO object, atau memakai method `PDO::setAttribute()`.

10.8. Menjalankan Query dengan method PDO::query()

Cara kedua untuk menjalankan query di PDO adalah melalui method `PDO::query()`. Berbeda dengan method `PDO::exec()` yang tidak bisa memproses hasil query `SELECT`, method `PDO::query()` bisa dipakai untuk menjalankan semua perintah query, termasuk `SELECT`, `INSERT`, `UPDATE`, `DELETE`, dll.

Prinsip kerja dari method `PDO::query()` ini sama seperti method `mysqli::query()`, yakni butuh sebuah argument berupa perintah query MySQL dan mengembalikan sebuah object yang bisa kita proses lebih lanjut.

Jika method `mysqli::query()` mengembalikan `mysqli_result` object, maka method `PDO::query()` akan mengembalikan `PDOStatement` object. Di dalam `PDOStatement` object inilah hasil query seperti data tabel disimpan untuk kemudian bisa kita proses dengan berbagai method lanjutan.

Apabila query yang ditulis error atau tidak bisa dipahami oleh MySQL, method `PDO::query()` akan mengembalikan nilai boolean `FALSE` dan otomatis melempar exception jika pengaturan `PDO::ERRMODE_EXCEPTION` aktif.

Berikut contoh pembuatan `PDOStatement` object:

14.pdo_query_stmt_obj.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
5
6     $query = "SELECT * FROM barang";
7     $stmt = $pdo->query($query);
8     var_dump($stmt);
9     $stmt = NULL;
10 }
11 catch (\PDOException $e) {
12     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";

```

```

13 }
14 finally {
15     $pdo=NULL;
16 }
```

Hasil kode program:

```
object(PDOStatement)#2 (1) { ["queryString"]=> string(20) "SELECT * FROM barang" }
```

Fokus kita ada di baris 6 – 9. Di baris 6 saya membuat variabel \$query yang berisi perintah query "SELECT * FROM barang".

Selanjutnya di baris 7 adalah proses pembuatan **PDOStatement** object. Hasil pemanggilan method \$pdo->query(\$query) disimpan ke dalam variabel \$stmt. Variabel \$stmt inilah yang berisi **PDOStatement** object. Perintah var_dump(\$stmt) di baris 8 memperlihatkan hal ini.

Di baris 9 saya mengisi nilai NULL ke dalam variabel \$stmt. Ini adalah cara untuk menghapus **PDOStatement** object jika sudah tidak diperlukan lagi. Sebenarnya ini sama seperti method \$mysqli_result::free(), namun karena PDO tidak memiliki method seperti itu maka terpaksa di hapus manual dengan cara mengisi nilai NULL.

Kembali, proses penghapusan **PDOStatement** object ini bersifat opsional dan tidak harus ditulis. Artinya, variabel penampung **PDOStatement** object (\$stmt) tidak harus diisi nilai NULL di akhir kode program.

Apabila method **PDO**::**query()** dipakai untuk menjalankan perintah MySQL yang mengubah data (INSERT, UPDATE dan DELETE), kita bisa mengakses method **PDOStatement**::**rowCount()** untuk mengetahui jumlah baris yang terdampak (*affected rows*), berikut contohnya:

14a.pdo_query_stmt_obj_rowcount.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "UPDATE barang SET jumlah_barang = 99";
7     $stmt = $pdo->query($query);
8     if ($stmt !== FALSE) {
9         echo "Query Ok, ada ".$stmt->rowCount()." baris yang di update";
10    }
11    $stmt = NULL;
12 }
13 catch (\PDOException $e) {
14     echo "Koneksi / Query bermasalah: ".$e->getMessage()." (".$e->getCode().")";
15 }
16 finally {
17     $pdo=NULL;
18 }
```

Hasil kode program:

```
Query Ok, ada 5 baris yang di update
```

Di baris 6 saya menjalankan query "UPDATE barang SET jumlah_barang = 99". Query ini akan meng-update kolom `jumlah_barang` menjadi 99 untuk **seluruh** baris yang ada di dalam tabel barang (karena perintah UPDATE ini tidak memiliki kondisi WHERE).

Namun fokus utama kita ada di baris 8 – 10, dimana saya membuat kondisi `if($stmt !== FALSE)`. Kondisi ini akan dijalankan jika tidak ada error di query MySQL, sebab method `query()` di baris 7 akan mengembalikan nilai FALSE jika terdapat error.

Jika query tidak error, maka method `$stmt->rowCount()` di baris 8 akan menampilkan jumlah kolom yang terdampak dari hasil query UPDATE tersebut.

Berikutnya, bagaimana menampilkan data yang tersimpan di dalam **PDOStatement** object?

Kita bisa memakai salah satu dari method `PDOStatement::fetch`, atau

`PDOStatement::fetchAll`.

Silahkan reset ulang tabel barang dengan menjalankan file `bab09\20.mysqli_generate.php` agar isi kolom `jumlah_barang` kembali seperti semula.

10.9. Menampilkan hasil Query dengan PDOStatement::fetch()

Method `PDOStatement::fetch()` dipakai untuk memproses hasil dari **PDOStatement** object secara baris per baris. Artinya, jika kita ingin menampilkan seluruh data tabel, method ini harus ditempatkan ke dalam perulangan **while**.

Method `PDOStatement::fetch()` bisa diisi dengan satu argument yang akan mengatur seperti apa proses pengambilan data. Argument ini berbentuk konstanta dengan berbagai pilihan:

- ◆ `PDO::FETCH_NUM`: menampilkan data sebagai *numeric array*, dengan index berupa nomor urutan kolom.
- ◆ `PDO::FETCH_ASSOC`: menampilkan data sebagai *associative array*, dengan index berupa nama kolom.
- ◆ `PDO::FETCH_BOTH`: menampilkan data sebagai *numeric array* dan *associative array* sekaligus. Ini adalah pengaturan default jika method `fetch()` dipanggil tanpa argument.
- ◆ `PDO::FETCH_OBJ`: menampilkan data sebagai object, dengan nama kolom sebagai property.
- ◆ `PDO::FETCH_LAZY`: menampilkan data sebagai *numeric array*, *associative array*, dan object sekaligus.
- ◆ `PDO::FETCH_COLUMN`: menampilkan 1 kolom data.

Selain daftar di atas, masih ada beberapa pilihan lain yang cukup rumit, lengkapnya bisa ke manual PHP di [PDOStatement::fetch](#).

Berikut contoh penggunaan dari `PDOStatement::fetch(PDO::FETCH_NUM)`:

Agar kode program kita lebih ringkas, saya tidak lagi menampilkan seluruh kode program (terutama yang dipakai untuk *error handling*). Versi lengkapnya bisa anda buka dari file `belajar_oop_php.zip`

15.pdo_query_fetch_num.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
8     echo $row[0];    echo " | ";
9     echo $row[1];    echo " | ";
10    echo $row[2];   echo " | ";
11    echo $row[3];   echo " | ";
12    echo $row[4];
13    echo "<br>";
14 }
```

Hasil kode program:

```

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-17 15:02:47
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-17 15:02:47
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-17 15:02:47
```

Terlihat bahwa cara penggunaan method `$stmt->fetch(PDO::FETCH_NUM)` sama seperti method `$result->fetch_row()` di **mysqli** object.

Setiap pemanggilan method `$stmt->fetch(PDO::FETCH_NUM)` akan mengembalikan 1 baris saja, yang dalam contoh ini ditampung ke dalam variabel `$row`. Untuk menampilkan semua baris, harus diproses menggunakan perulangan **while**.

Jika ingin mengambil data tabel sebagai *associative array*, kita bisa memakai method `PDOStatement::fetch(PDO::FETCH_ASSOC)`:

16.pdo_query_fetch_assoc.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
```

PDO

```
6
7 while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
8     echo $row['id_barang'];      echo " | ";
9     echo $row['nama_barang'];    echo " | ";
10    echo $row['jumlah_barang'];  echo " | ";
11    echo $row['harga_barang'];   echo " | ";
12    echo $row['tanggal_update'];
13    echo "<br>";
14 }
```

Cara ini mirip seperti method `$result->fetch_assoc()` di `mysqli` object, dimana kita mengakses data tabel dengan nama kolom sebagai key atau index array.

Selanjutnya, berikut contoh penggunaan dari `PDOStatement::fetch(PDO::FETCH_BOTH)`:

17.pdo_query_fetch_both.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_BOTH)){
8     echo $row['id_barang'];      echo " | ";
9     echo $row[1];               echo " | ";
10    echo $row['jumlah_barang'];  echo " | ";
11    echo $row[3];               echo " | ";
12    echo $row['tanggal_update'];
13    echo "<br>";
14 }
```

Dengan memakai konstanta `PDO::FETCH_BOTH`, kita bisa mengakses array `$row` menggunakan *numeric array* maupun *associative array*. Ini adalah pilihan default jika method `fetch()` dipanggil tanpa menulis argument seperti contoh berikut:

```
1 ...
2 while($row = $stmt->fetch()){
3 ...
```

Nantinya kita juga bisa mengubah pengaturan default ini.

Jika menggunakan konstanta `PDO::FETCH_OBJ`, maka proses menampilkan data tabel akan memakai pemanggilan object:

18.pdo_query_fetch_obj.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
```

PDO

```
7 while ($row = $stmt->fetch(PDO::FETCH_OBJ)){
8     echo $row->id_barang;      echo " | ";
9     echo $row->nama_barang;    echo " | ";
10    echo $row->jumlah_barang;  echo " | ";
11    echo $row->harga_barang;   echo " | ";
12    echo $row->tanggal_update;
13    echo "<br>";
14 }
```

Cara menampilkan ini sama seperti method `$result->fetch_object()` di versi **mysqli**, dimana kita memakai format `$row->nama_kolom`.

Yang tidak ada di versi **mysqli** adalah, mengakses *numeric array*, *associative array* dan *object* sekaligus. Di PDO ini bisa dilakukan dengan method `PDOStatement::fetch(PDO::FETCH_LAZY)`:

19.pdo_query_fetch_lazy.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_LAZY)){
8     echo $row['id_barang'];      echo " | ";
9     echo $row[1];               echo " | ";
10    echo $row->jumlah_barang;  echo " | ";
11    echo $row[3];               echo " | ";
12    echo $row->tanggal_update;
13    echo "<br>";
14 }
```

Di sini saya mengakses variabel `$row` dengan 3 cara: *numeric array*, *associative array* serta *object*.

Pilihan konstanta lain untuk `PDOStatement::fetch()` adalah `PDO::FETCH_COLUMN`, yang akan mengembalikan array untuk 1 kolom (bukan berbentuk baris). Berikut contoh prakteknya:

20.pdo_query_fetch_column.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT nama_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_COLUMN)){
8     echo $row." | ";
9 }
```

Hasil kode program:

TV Samsung 43NU7090 4K | Kulkas LG GC-A432HLHU | Laptop ASUS ROG GL503GE | Printer

Epson L220 | Smartphone Xiaomi Pocophone F1 |

Di sini query yang saya jalankan adalah "SELECT nama_barang FROM barang", ini dipakai untuk mengambil semua nilai dari kolom `nama_barang`. Dengan memakai method `fetch(PDO::FETCH_COLUMN)`, variabel `$row` langsung berisi data kolom. Kita tidak perlu lagi menulis judul kolom sebagai key array. Jika menggunakan method `$stmt->fetch(PDO::FETCH_ASSOC)`, maka perlu menulis key array seperti `$row['nama_barang']`.

Sebelumnya telah dijelaskan bahwa konstanta `PDO::FETCH_BOTH` adalah pilihan default jika method `fetch()` dipanggil tanpa argument. Kita bisa mengubah pengaturan ini dari method `PDO::setAttribute()`. Caranya, gunakan `PDO::ATTR_DEFAULT_FETCH_MODE` sebagai argument pertama, lalu pilih salah satu konstanta method `fetch()` sebagai argument kedua.

Sebagai contoh, jika saya ingin pilihan default method `fetch()` adalah `PDO::FETCH_LAZY`, maka kode programnya adalah sebagai berikut:

21.pdo_query_fetch_setattribute.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3 $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
4 $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_LAZY);
5
6 $query = "SELECT * FROM barang";
7 $stmt = $pdo->query($query);
8
9 while ($row = $stmt->fetch()){
10    echo $row['id_barang'];
11    echo $row[1];
12    echo $row->jumlah_barang;
13    echo $row[3];
14    echo $row->tanggal_update;
15    echo "<br>";
16 }
```

Dengan tambahan kode program di baris 4, maka ketika method `fetch()` dipanggil tanpa argument seperti di baris 9, secara otomatis menggunakan konstanta `PDO::FETCH_LAZY`.

Alternatif penulisan lain adalah pada saat pembuatan PDO object, seperti:

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "",
4                     [ PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
5                       PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ ]);
6 // ...
7 // ...
```

Sekarang jika method `PDOStatement::fetch()` dipanggil tanpa argument, akan memakai konstanta `PDO::FETCH_OBJ` secara default.

10.10. Menampilkan hasil Query dengan PDOStatement::fetchAll()

Method `PDOStatement::fetchAll()` adalah variasi lain dari `PDOStatement::fetch()` yang baru saja kita pelajari. Melihat dari namanya, bisa di tebak bahwa method `fetchAll()` dipakai untuk mengambil seluruh data hasil query `SELECT`, bukan lagi baris per baris sebagaimana `fetch()`.

Array hasil pemanggilan method `PDOStatement::fetchAll()` berbentuk 2 dimensi karena akan menampung 1 tabel lengkap (memiliki dimensi kolom dan baris). Ini mirip seperti method `mysqli_stmt::fetch_all()` yang kita bahas pada bab tentang `mysqli` object.

Method `PDOStatement::fetchAll()` juga butuh 1 argument berupa konstanta yang akan menentukan seperti apa array hasil pemanggilan. Berikut beberapa pilihan konstanta tersebut:

- ◆ `PDO::FETCH_NUM`
- ◆ `PDO::FETCH_ASSOC`
- ◆ `PDO::FETCH_BOTH`
- ◆ `PDO::FETCH_OBJ`
- ◆ `PDO::FETCH_CLASS`
- ◆ `PDO::FETCH_COLUMN`
- ◆ `PDO::FETCH_KEY_PAIR`

Terlihat bahwa semua konstanta yang ada di method `PDOStatement::fetch()` sebelumnya juga bisa dipakai untuk `PDOStatement::fetchAll()`, kecuali `PDO::FETCH_LAZY`. Jika konstanta tidak ditulis, yang akan dipakai adalah `PDO::FETCH_BOTH`.

Berikut contoh penggunaan dari `PDOStatement::fetchAll()`:

22.pdo_query_fetchall_num.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_NUM);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo "<br>".$arr[2][1];

```

Hasil kode program:

PDO

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => TV Samsung 43NU7090 4K
            [2] => 5
            [3] => 5399000
            [4] => 2019-01-17 15:02:47
        )

    [1] => Array
        (
            [0] => 2
            [1] => Kulkas LG GC-A432HLHU
            [2] => 10
            [3] => 7600000
            [4] => 2019-01-17 15:02:47
        )

    ...
)
```

Kulkas LG GC-A432HLHU

Di baris 7 saya menjalankan method `$stmt->fetchAll(PDO::FETCH_NUM)` dan menyimpan hasilnya ke dalam variabel `$arr`. Karena menggunakan konstanta `PDO::FETCH_NUM`, maka `$arr` akan berisi *numeric array*.

Perintah `print_r($arr)` di baris 9 memperlihatkan struktur dari array `$arr`. Sebagai contoh, untuk menampilkan isi kolom `nama_barang` (kolom ke-2) dari baris ke-3 perintahnya adalah `$arr[2][1]`. Jika kita ingin menampilkan semua nilai yang ada di dalam `$arr`, bisa memakai perulangan **foreach**, yang caranya sama seperti di pembahasan tentang **mysqli object**.

Berikut contoh penggunaan dari `PDOStatement::fetchAll(PDO::FETCH_ASSOC)`:

23.pdo_query_fetchall_assoc.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_ASSOC);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo "<br>".$arr[2]["nama_barang"];
```

Hasil kode program:

PDO

```
Array
(
    [0] => Array
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            ...
        )
)
```

Laptop ASUS ROG GL503GE

Jika menggunakan konstanta PDO::FETCH_ASSOC, index array dimensi pertama tetap berupa nomor (yang menandakan urutan baris), namun untuk dimensi kedua (urutan kolom) akan memakai nama kolom yang berasal dari MySQL. Dengan demikian, untuk menampilkan isi kolom `nama_barang` dari baris ke-3 perintahnya adalah `$arr[2]["nama_barang"]`.

Selanjutnya, berikut contoh penggunaan dari `PDOStatement::fetchAll(PDO::FETCH_OBJ)`:

24.pdo_query_fetchall_obj.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_OBJ);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo $arr[2]->nama_barang;
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            ...
        )
)
```

Laptop ASUS ROG GL503GE

Kembali, index dari dimensi pertama (untuk baris) tetap berbentuk angka. Namun dimensi kedua (untuk kolom) menggunakan penulisan object. Sehingga apabila ingin menampilkan isi kolom `nama_barang` dari baris ke-3 perintahnya adalah `$arr[2]->nama_barang`.

Konstanta berikutnya untuk method `fetchAll()` adalah PDO::FETCH_CLASS. Konstanta ini mirip seperti PDO::FETCH_OBJ, terutama jika dipanggil tanpa argument kedua. Maksudnya, kedua

pemanggilan ini akan menampilkan hasil yang sama:

```
13 ...
14 $arr = $stmt->fetchAll(PDO::FETCH_OBJ);
15 $arr = $stmt->fetchAll(PDO::FETCH_CLASS);
16 ...
```

Bedanya, jika menggunakan PDO::FETCH_CLASS, kita bisa menginput nama class sebagai argument kedua dari method `fetchAll()`.

Jika diperhatikan, object untuk setiap baris di dalam variabel `$arr` di-set oleh PHP sebagai **stdClass** object, yakni object "generik" bawaan PHP. Kita bisa mengatur agar PHP memakai class lain yang telah siapkan sebelumnya. Caranya, input nama class sebagai argument kedua dari method `fetchAll(PDO::FETCH_CLASS)`:

25.pdo_query_fetchall_obj_class.php

```
1 <?php
2 class MyClass{}
3
4 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
5 $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
6
7 $query = "SELECT * FROM barang";
8 $stmt = $pdo->query($query);
9
10 $arr = $stmt->fetchAll(PDO::FETCH_CLASS, "MyClass");
11 echo "<pre>";
12 print_r($arr);
13 echo "</pre>";
14
15 echo $arr[2]->nama_barang;
```

Hasil kode program:

```
Array
(
    [0] => MyClass Object
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            ...
        )
)
```

Laptop ASUS ROG GL503GE

Di baris 2 saya membuat class `MyClass` yang memang tidak berisi apa-apa. Class ini kemudian diinput sebagai argument kedua saat pemanggilan method `fetchAll()` di baris 10. Hasilnya, setiap baris sekarang merupakan *instance* dari class `MyClass`.

Cara seperti ini bisa dipakai untuk membuat teknik yang lebih rumit, misalnya mengisi class `MyClass` dengan property lain atau magic method `__set()` yang akan memproses nilai inputan.

Berikut contoh prakteknya:

26 pdo_query_fetchall_obj_class_set.php

```

1 <?php
2 class IlkoomBarang{
3     public $nama_toko = "Ilkoom Store";
4     public function __set($name, $value) {
5         $this->$name = strtoupper($value);
6     }
7 }
8
9 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
10
11 $query = "SELECT * FROM barang";
12 $stmt = $pdo->query($query);
13
14 $arr = $stmt->fetchAll(PDO::FETCH_CLASS, "IlkoomBarang");
15 echo "<pre>";
16 print_r($arr);
17 echo "</pre>";

```

Hasil kode program:

```

Array
(
    [0] => IlkoomBarang Object
        (
            [nama_toko] => Ilkoom Store
            [id_barang] => 1
            [nama_barang] => TV SAMSUNG 43NU7090 4K
            [jumlah_barang] => 5
            [harga_barang] => 5399000
            [tanggal_update] => 2019-01-17 15:02:47
        )

    [1] => IlkoomBarang Object
        (
            [nama_toko] => Ilkoom Store
            [id_barang] => 2
            [nama_barang] => KULKAS LG GC-A432HLHU

```

Di baris 2 - 7 saya membuat class IlkoomBarang. Class ini memiliki property \$nama_toko yang diisi dengan string "Ilkoom Store". Kemudian terdapat *magic method* __set() yang saya rancang agar setiap pengisian property yang tidak ada di dalam class, nilainya di proses dulu dengan fungsi strtoupper().

Pada saat pemanggilan method \$stmt->fetchAll(PDO::FETCH_CLASS, "IlkoomBarang") di baris 14, data dari tabel barang akan diinput ke dalam object IlkoomBarang. Buktinya, terdapat tambahan property [nama_toko] => Ilkoom Store di setiap object.

Selain itu karena terdapat magic method __set(), maka setiap nilai yang berasal dari tabel

barang akan di konversi menjadi huruf besar. Ini bisa dilihat dari isi property `nama_barang`. Untuk kolom lain tidak ada perubahan karena berisi angka.

Berikutnya, method `fetchAll()` juga bisa menerima inputan konstanta `PDO::FETCH_COLUMN`. Pilihan ini akan mengambil 1 kolom tabel, seperti contoh berikut:

27.pdo_query_fetchall_column.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT harga_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_COLUMN);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo $arr[2];

```

Hasil kode program:

```

Array
(
    [0] => 5399000
    [1] => 7600000
    [2] => 16200000
    [3] => 2099000
    [4] => 4750000
)
16200000

```

Karena yang diambil hanya 1 kolom, maka variabel `$arr` hanya berisi 1 dimensi saja. Jika ingin menampilkan data ketiga, tinggal akses `$arr[2]`.

Apabila hasil query `SELECT` mengembalikan lebih dari 1 kolom, maka kolom paling awal yang akan diambil oleh `fetchAll(PDO::FETCH_COLUMN)`:

28.pdo_query_fetchall_column_many.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_COLUMN);
8
9 echo "<pre>";
10 print_r($arr);
11 echo "</pre>";

```

Hasil kode program:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
```

Query yang dijalankan adalah "SELECT * FROM barang", artinya ambil semua nilai yang ada di tabel barang (5 baris dan 5 kolom). Namun karena kita menggunakan method `fetchAll(PDO::FETCH_COLUMN)`, maka yang diambil hanya kolom pertama saja, yakni kolom `id_barang` dalam contoh ini.

Konstanta `PDO::FETCH_COLUMN` juga bisa digabung dengan konstanta lain untuk proses filter data, misalnya seperti ini:

```
1 ...
2 $arr = $stmt->fetchAll(PDO::FETCH_COLUMN | PDO::FETCH_UNIQUE);
3 ...
```

Tambahan `PDO::FETCH_UNIQUE` akan mengembalikan nilai kolom yang unik saja. Artinya, jika terdapat data yang berulang, hanya diambil 1 nilai. Ini kurang lebih sama seperti hasil penambahan clausa `DISTINCT` di perintah query MySQL.

Konstanta terakhir yang akan kita bahas untuk method `fetchAll()` adalah `PDO::FETCH_KEY_PAIR`. Ini dipakai untuk membuat pasangan key – value array dari 2 kolom tabel. Kolom pertama akan menjadi `key` dan kolom kedua berisi `value`. Berikut contoh penggunaannya:

29 pdo_query_fetchall_fetch_key_pair.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT id_barang,harga_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_KEY_PAIR);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo $arr[3];
```

Hasil kode program:

```
Array
(
```

PDO

```
[1] => 5399000
[2] => 7600000
[3] => 16200000
[4] => 2099000
[5] => 4750000
)
16200000
```

Query yang saya jalankan adalah "SELECT id_barang,harga_barang FROM barang". Hasilnya berbentuk 2 kolom data, yakni kolom **id_barang** dan **harga_barang**.

Karena konstanta yang dipakai adalah PDO::FETCH_KEY_PAIR, maka kolom **id_barang** akan menjadi key dari arrar \$arr, sedangkan kolom **harga_barang** menjadi nilai atau isi dari array tersebut.

Berikut contoh untuk kolom lain:

30 pdo_query_fetchall_fetch_key_pair_2.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT nama_barang,harga_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_KEY_PAIR);
8
9 echo "<pre>";
10 print_r($arr);
11 echo "</pre>";
12
13 echo $arr["TV Samsung 43NU7090 4K"]."<br>";
14 echo $arr["Kulkas LG GC-A432HLHU"]."<br>";
15 echo $arr["Printer Epson L220"]."<br>";
```

Hasil kode program:

```
Array
(
    [TV Samsung 43NU7090 4K] => 5399000
    [Kulkas LG GC-A432HLHU] => 7600000
    [Laptop ASUS ROG GL503GE] => 16200000
    [Printer Epson L220] => 2099000
    [Smartphone Xiaomi Pocophone F1] => 4750000
)
5399000
7600000
2099000
```

Sekarang pasangan kolom yang saya pilih adalah **nama_barang** dengan **harga_barang**. Hasilnya, dengan mengakses echo \$arr["TV Samsung 43NU7090 4K"] akan tampil harga barang untuk baris tersebut.

Penggunaan konstanta PDO::FETCH_KEY_PAIR hanya bisa dipakai untuk data yang hasilnya berbentuk 2 kolom, tidak boleh lebih atau kurang.

10.11. Prepared Statement dengan PDO

Agar query yang ditulis lebih aman, terutama jika terdapat data yang berasal dari form, lebih baik menggunakan **prepared statement**. Untungnya, prepared statement di PDO lebih sederhana (dan juga lebih fleksibel) dibandingkan prepared statement versi **mysqli**.

Langkah yang dipakai tetap sama, yakni **prepare** (mempersiapkan query), **bind** (menghubungkan data dengan query) dan **execute** (menjalankan query). Dalam PDO, proses bind dan execute bisa dilakukan dengan 1 perintah saja.

Selain itu PDO tidak butuh object khusus untuk menjalankan prepared statement. Object yang diperlukan tetap **PDOStatement** yang selama ini kita pakai menjalankan method **fetch()** dan **fetchAll()**. Ini berbeda dengan mysqli yang butuh **mysqli_stmt** object untuk menjalankan prepared statement.

Karena tidak butuh object baru, maka kita bisa langsung menampilkan hasil prepared statement dengan method **fetch()** dan **fetchAll()** yang telah dibahas sebelumnya.

Berikut contoh penulisan prepared statement dengan PDO:

31.pdo_prepared_fetchall_num.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE id_barang = ?";
5 $stmt = $pdo->prepare($query);
6 $stmt->execute([4]);
7
8 $arr = $stmt->fetchAll(PDO::FETCH_NUM);
9 echo "<pre>";
10 print_r($arr);
11 echo "</pre>";
12
13 echo $arr[0][1];

```

Hasil kode program:

```

Array
(
    [0] => Array
        (
            [0] => 4
            [1] => Printer Epson L220
            [2] => 14
            [3] => 2099000
            [4] => 2019-01-17 15:02:47

```

PDO

```
)  
)  
Printer Epson L220
```

Di baris 4 saya membuat query **SELECT** dengan kondisi **WHERE id_barang = ?**. Tanda tanya ini nantinya kita input secara terpisah.

Di baris 5, query yang tersimpan di dalam variabel \$query di jalankan dengan method \$pdo->prepare(\$query), inilah proses **prepare**. Sama seperti pemanggilan method \$pdo->query(), method \$pdo->prepare() ini mengembalikan **PDOStatement** object yang saya simpan ke dalam variabel \$stmt.

Pemanggilan method \$stmt->execute([4]) di baris 6 adalah proses **bind** sekaligus **execute** dari prepared statement. Argument yang diisi ke dalam method execute() adalah nilai pengganti tanda tanya di query prepared. Dalam hal ini saya ingin mengisi angka 4 agar query yang diproses menjadi: "SELECT * FROM barang WHERE id_barang = 4". Argument untuk method execute() harus berbentuk **array**, sehingga ditulis sebagai [4].

Setelah proses execute, sisa kode program di baris 8 – 13 sama seperti pembahasan kita sebelumnya, dimana saya menampilkan isi query dengan method fetchAll(PDO::FETCH_NUM). Hasilnya, variabel \$arr berisi tabel barang dalam format *numeric array*. Isinya hanya ada 1 baris karena terdapat batasan kondisi WHERE id_barang = 4.

Bagaimana jika ada 2 kondisi? Tidak masalah, cukup tambahkan dua buah tanda tanya dan input 2 buah argument ke dalam method execute() seperti contoh berikut:

32 pdo_prepared_fetch_assoc.php

```
1 <?php  
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");  
3  
4 $query = "SELECT * FROM barang WHERE id_barang = ? OR  
5       nama_barang = ?";  
6 $stmt = $pdo->prepare($query);  
7 $stmt->execute([1, "Printer Epson L220"]);  
8  
9 while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){  
10 echo $row['id_barang'];    echo " | ";  
11 echo $row['nama_barang']; echo " | ";  
12 echo $row['jumlah_barang']; echo " | ";  
13 echo $row['harga_barang']; echo " | ";  
14 echo $row['tanggal_update'];  
15 echo "<br>";  
16 }
```

Hasil kode program:

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47  
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47
```

Kali ini query yang di-prepare adalah "SELECT * FROM barang WHERE id_barang = ? OR nama_barang = ?". Karena ada dua buah tanda tanya, maka array yang diinput ke dalam argument method `execute()` juga perlu 2 buah nilai, yakni [1, "Printer Epson L220"]. Dengan demikian, query akhir akan menjadi `SELECT * FROM barang WHERE id_barang = 1 OR nama_barang = "Printer Epson L220"`.

Kemudian saya menggunakan perulangan **while** dan method `$stmt->fetch(PDO::FETCH_ASSOC)` untuk menampilkan hasil tabel barang.

Sampai di sini kita bisa lihat perbedaan antara *prepared statement* di **mysqli** dengan **PDO**. Di PDO, kita tidak butuh proses **bind** secara manual, serta tidak perlu juga menginput jenis tipe data seperti method `bind()` di mysqli.

Di dalam PDO, semua data inputan dianggap sebagai string. MySQL sendiri tidak akan komplain jika tipe data angka juga diinput sebagai string. Maksudnya, kedua query berikut bisa diproses sebagaimana mestinya:

```
SELECT harga_barang FROM barang WHERE id_barang = 2
SELECT harga_barang FROM barang WHERE id_barang = '2'
```

Di dalam MySQL, kolom `id_barang` saya set sebagai integer, namun MySQL tetap bisa memproses walaupun data untuk kolom tersebut diinput sebagai integer. PDO memanfaatkan hal ini sehingga kita tidak perlu mengatur tipe data setiap inputan.

Perbedaan lain adalah, proses **bind** di method `execute()` tidak harus berbentuk variabel, tapi bisa diisi dengan data langsung. Namun jika diinginkan, kita juga bisa mengisinya sebagai variabel seperti contoh berikut:

33.pdo_prepared_fetch_variable.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE id_barang = ? OR
5         nama_barang = ?";
6 $stmt = $pdo->prepare($query);
7
8 $id = 1;
9 $nama = "Printer Epson L220";
10 $stmt->execute([$id, $nama]);
11
12 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
13     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
14     echo "<br>";
15 }
```

Hasil kode program:

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47

4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47

Query yang dipakai sama seperti sebelumnya, namun di baris 8 dan 9 saya menyiapkan variabel \$id dan \$nama untuk diinput ke dalam method `execute()` di baris 10. Selain itu memakai method `$stmt->fetch(PDO::FETCH_NUM)` untuk menampilkan data tabel.

10.12. Prepared Statement dengan Named Parameters

Sebelumnya kita memakai tanda tanya " ? " sebagai *placeholder* atau penanda inputan query untuk prepared statement. Di PDO, terdapat penulisan lain yang dikenal sebagai **named parameter**.

Dengan *named parameter*, kita bisa menggunakan "nama" yang diawali dengan tanda titik dua sebagai penanda *placeholder*, seperti ":id", ":nama", atau ":harga_barang". Kemudian pada saat proses bind, nama ini diisi menggunakan *associative array*. Berikut contoh penggunaannya:

34.pdo_prepared_named_parameters.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE id_barang = :id OR
5         nama_barang = :nama";
6 $stmt = $pdo->prepare($query);
7
8 $stmt->execute(['id'=>1, 'nama'=>"Printer Epson L220"]);
9
10 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
12     echo "<br>";
13 }
```

Hasil kode program:

```

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47
```

Perhatikan cara penulisan query di baris 4-5, saya memakai :id dan :nama sebagai pengganti dari tanda tanya " ? ", inilah cara penulisan *named parameters*. Pada saat proses **bind** menggunakan `execute()` di baris 8, argument method ditulis sebagai *associative array*, yakni berbentuk `['id'=>1, 'nama'=>"Printer Epson L220"]`.

Salah satu keuntungan dari penggunaan named parameter adalah kita tidak terikat dengan urutan *placeholder*. Selama nama index dalam *associative array* sesuai dengan nama *placeholder*, tidak pengaruh urutannya seperti contoh berikut:

35.pdo_prepared_named_parameters_2.php

```

1 <?php
```

```

2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE jumlah_barang < :jumlah OR
5         harga_barang > :harga";
6 $stmt = $pdo->prepare($query);
7
8 $stmt->execute(['harga'=>5000000, 'jumlah'=>15]);
9
10 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11     echo $row[0]." | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
12     echo "<br>";
13 }

```

Hasil kode program:

```

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-17 15:02:47
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-17 15:02:47
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47

```

Pada saat penulisan query di baris 4 – 5, kondisi yang dipakai adalah "WHERE jumlah_barang < :jumlah OR harga_barang > :harga". Secara berurutan, penulisan placeholder-nya adalah :jumlah, lalu :barang. Namun di dalam argument method execute() ditulis sebagai ['harga'=>5000000, 'jumlah'=>15]. Ini tidak jadi masalah karena patokan dari named parameter adalah nama, bukan posisi.

Ini berbeda jika menggunakan tanda tanya "?" dimana kita harus menulis argument sesuai urutan ketika menjalankan method execute(). Karena itu pula penggunaan tanda tanya ini disebut sebagai **positional placeholder** atau **positional parameters**.

Di dalam method execute(), named parameter ini kadang ditulis juga dengan menyertakan tanda titik dua ":" , tapi ini tidak wajib dan PHP Manual juga tidak mengatur tentang hal ini. Kita bisa menggunakan salah satu perintah berikut:

```

$stmt->execute(['harga'=>5000000, 'jumlah'=>15]);
$stmt->execute([':harga'=>5000000, ':jumlah'=>15]);

```

10.13. Multiple Execution Prepared Statement

Salah satu keunggulan dari prepared statement (selain keamanan data input), adalah kita mengeksekusi query yang sama lebih dari 1 kali dengan nilai yang berbeda-beda, yakni *multiple execution*.

Materi ini juga sudah kita praktekkan di versi **mysqli** object, dan berikut contohnya di dalam PDO:

36 pdo_prepared_multiple_execution.php

```

1 <?php
2
3 // Buat format tanggal hari ini
4 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
5 $timestamp = $sekarang->format("Y-m-d H:i:s");
6
7 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
8
9 $query = "INSERT INTO barang (nama_barang, jumlah_barang,
10 harga_barang, tanggal_update) VALUES (:nama,:jumlah,:harga,:tanggal)";
11
12 $stmt = $pdo->prepare($query);
13
14 // Input data 1
15 $nama = "Cosmos CRJ-8229 - Rice Cooker";
16 $jumlah = 4;
17 $harga = 299000;
18 $tanggal = $timestamp;
19
20 $stmt->execute(['nama'=>$nama, 'jumlah'=>4, 'harga'=>$harga,
21 'tanggal'=>$tanggal]);
22 echo "Query Ok, ".$stmt->rowCount()." baris berhasil ditambah <br>";
23
24 // Input data 2
25 $arr_input = [
26 'nama' => "Philips Blender HR 2157",
27 'jumlah' => 11,
28 'harga' => 629000,
29 'tanggal' => $timestamp
30 ];
31
32 $stmt->execute($arr_input);
33 echo "Query Ok, ".$stmt->rowCount()." baris berhasil ditambah <br>";
34
35 echo "<hr>";
36
37 // Tampilkan data barang
38 $query = "SELECT * FROM barang";
39 $stmt = $pdo->query($query);
40
41 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
42 echo $row[0]." | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
43 echo "<br>";
44 }

```

Hasil kode program:

Query Ok, 1 baris berhasil ditambah
 Query Ok, 1 baris berhasil ditambah

1	TV Samsung 43NU7090 4K	5	5399000	2019-01-17 15:02:47
2	Kulkas LG GC-A432HLHU	10	7600000	2019-01-17 15:02:47
3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-17 15:02:47

```

4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-17 15:02:47
6 | Cosmos CRJ-8229 - Rice Cooker | 4 | 299000 | 2019-01-21 07:57:32
7 | Philips Blender HR 2157 | 11 | 629000 | 2019-01-21 07:57:32

```

Dalam kode program ini saya menambah 2 buah data baru ke dalam tabel `barang` dengan 1 penulisan query prepared.

Di awal program terdapat kode untuk menggenerate tanggal hari ini yang disimpan ke dalam variabel `$timestamp`. Kemudian di baris 9-10 adalah penulisan *prepared query* `INSERT`, dimana saya memakai *named parameter* untuk 4 data inputan.

Sebagai data pertama, saya membuat 5 variabel di baris 15 – 18, yang kemudian diinput ke dalam method `execute()` di baris 20.

Tanpa menulis ulang query, saya membuat data kedua dengan bentuk *associative array* `$arr_input` di baris 25 – 30, yang kemudian dipakai untuk pemanggilan method `execute()` di baris 32.

Di sini kita telah menjalankan 2 buah proses `execute()` untuk 1 query prepared. Sebagai latihan, anda bisa membuat kode program yang sama, tapi menggunakan *positioning parameter*, yakni memakai tanda tanya " ? " pada saat pembuatan query prepared.

10.14. Transaction Query dengan PDO

Untuk membuat transaction, PDO menyediakan 3 method:

- ◆ `PDO::beginTransaction()`
- ◆ `PDO::rollBack()`
- ◆ `PDO::commit()`

Perhatikan bahwa ketiga method ini "melekat" ke **PDO** object, bukan ke **PDOStatement** object.

Cara penggunaannya mirip seperti di versi **mysqli**, yakni kita membuka proses transaction dengan menjalankan method `PDO::beginTransaction()` lalu menulis query seperti biasa menggunakan prepared statement maupun tidak.

Jika terjadi masalah, semua query bisa dibatalkan dengan memanggil method `PDO::rollBack()`. Atau jika sudah oke, jalankan method `PDO::commit()` untuk mempermanenkan hasil query.

Berikut contoh kode program dari proses transaction di PDO:

Karena dalam contoh sebelumnya kita menambah data baru ke tabel `barang`, silahkan reset ulang dengan menjalankan file `bab09\20.mysql_generate.php`.

PDO

37.pdo_prepared_transaction.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $pdo->beginTransaction();
5
6 $query = "DELETE FROM barang WHERE id_barang = ?";
7 $stmt = $pdo->prepare($query);
8 $stmt->execute([2]);
9
10 $query = "DELETE FROM barang WHERE nama_barang = :nama";
11 $stmt = $pdo->prepare($query);
12 $stmt->execute(['nama'=>"TV Samsung 43NU7090 4K"]);
13
14 // Tampilkan isi tabel selama transaction
15 echo "<h3>Di dalam Transaction</h3>";
16 $query = "SELECT * FROM barang";
17 $stmt = $pdo->query($query);
18
19 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
20     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
21     echo "<br>";
22 }
23
24 $pdo->rollBack(); // atau $pdo->commit()
25
26 echo "<hr>";
27
28 // Tampilkan isi tabel setelah transaction
29 echo "<h3>Setelah Transaction</h3>";
30 $query = "SELECT * FROM barang";
31 $stmt = $pdo->query($query);
32
33 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
34     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
35     echo "<br>";
36 }
```

Hasil kode program:

Di dalam Transaction

3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-21 08:31:33
4	Printer Epson L220	14	2099000	2019-01-21 08:31:33
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-21 08:31:33

Setelah Transaction

1	TV Samsung 43NU7090 4K	5	5399000	2019-01-21 08:31:33
2	Kulkas LG GC-A432HLHU	10	7600000	2019-01-21 08:31:33
3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-21 08:31:33
4	Printer Epson L220	14	2099000	2019-01-21 08:31:33
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-21 08:31:33

Setelah pembuatan koneksi dengan database MySQL, di baris 4 saya membuka proses

transaction dengan memanggil method `$pdo->beginTransaction()`.

Kemudian di baris 6 - 12 terdapat query `DELETE` yang akan menghapus tabel barang dengan kondisi `id_barang = 2` dan `nama_barang = TV Samsung 43NU7090 4K`, hasilnya 2 baris data akan dihapus.

Namun di baris 24 saya menjalankan method `$pdo->rollBack()` yang akan membatalkan semua query sebelumnya. Sekarang, isi tabel barang sudah kembali seperti semula.

10.15. Proses Bind Manual untuk Prepared Statement

Dalam query MySQL, `LIMIT` adalah perintah tambahan yang berfungsi membatasi hasil data. Sebagai contoh, untuk menampilkan 3 barang dengan harga termurah, kita bisa memakai query:

```
SELECT * FROM barang ORDER BY harga_barang LIMIT 3
```

Namun PDO memiliki sebuah masalah ketika inputan `LIMIT` ini berasal dari prepared statement. Berikut contoh kasusnya:

38 pdo_prepared_limit_problem.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "SELECT * FROM barang ORDER BY harga_barang LIMIT :batas";
7     $stmt = $pdo->prepare($query);
8     $stmt->execute(['batas'=>3]);
9
10    while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11        echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
12        echo "<br>";
13    }
14    $stmt = NULL;
15 }
16 catch (\PDOException $e) {
17     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
18 }
19 finally {
20     $pdo=NULL;
21 }
```

Hasil kode program:

```
Koneksi / Query bermasalah: SQLSTATE[42000]: Syntax error or access violation: 1064
You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near ''3'' at line 1 (42000)
```

Di sini saya menjalankan prepared statement dengan `LIMIT :batas`. Nilai `:batas` akan diinput dari perintah `$stmt->execute(['batas'=>3])`, namun hasilnya terjadi error.

Bisa di pastikan bahwa tidak ada yang salah dari penulisan query, hasil akhir yang diharapkan adalah "SELECT * FROM barang ORDER BY harga_barang LIMIT 3". Lalu dimana salahnya?

Ini berhubungan dengan cara PDO yang otomatis mengkonversi semua inputan prepared statement sebagai string. Akibatnya, kode di atas akan menghasilkan query sebagai berikut:

```
SELECT * FROM barang ORDER BY harga_barang LIMIT '3'
```

Perhatikan tambahan tanda kutip di bagian `LIMIT '3'`, inilah yang menyebabkan error. Untuk nilai batasan `LIMIT`, MySQL hanya bisa menerima nilai angka (integer), tidak bisa berbentuk string. Yang seharusnya dijalankan adalah `LIMIT 3`, bukan `LIMIT '3'`.

Jadi bagaimana solusinya? Ternyata PDO juga masih mengizinkan proses **bind** manual menggunakan method `PDOStatement::bindValue()` atau `PDOStatement::bindParam()`. Dengan memakai salah satu dari method ini, kita bisa mengatur tipe data nilai inputan.

Berikut contoh kode program dari penggunaan `PDOStatement::bindValue()`:

39 pdo_prepared_bind_value.php

```
1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "SELECT * FROM barang ORDER BY harga_barang LIMIT :batas";
7     $stmt = $pdo->prepare($query);
8     $stmt->bindValue('batas', 3, PDO::PARAM_INT);
9     $stmt->execute();
10
11    while ($row = $stmt->fetch(PDO::FETCH_NUM)){
12        echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
13        echo "<br>";
14    }
15    $stmt = NULL;
16 }
17 catch (\PDOException $e) {
18     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
19 }
20 finally {
21     $pdo=NULL;
22 }
```

Hasil kode program:

4	Printer Epson L220	14	2099000	2019-01-21 08:31:33
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-21 08:31:33
1	TV Samsung 43NU7090 4K	5	5399000	2019-01-21 08:31:33

Perbedaan dengan contoh sebelumnya hanya di baris 8 dan 9. Di baris 8 terdapat pemanggilan method `bindValue()`. Method ini butuh 3 buah argument:

1. Named parameter, yang dalam contoh ini berupa string 'batas'.
2. Nilai untuk named parameter, dalam contoh ini adalah angka 3.
3. Konstanta untuk tipe data dari named parameter. Karena angka 3 ingin dikirim sebagai integer, maka konstanta-nya adalah `PDO::PARAM_INT`.

Untuk parameter ke-3 ini, konstanta tipe data yang tersedia adalah:

- ◆ `PDO::PARAM_BOOL`, untuk tipe data boolean.
- ◆ `PDO::PARAM_NULL`, untuk nilai NULL.
- ◆ `PDO::PARAM_INT`, untuk tipe data integer.
- ◆ `PDO::PARAM_STR`, untuk tipe data string, char, atau varchar.
- ◆ `PDO::PARAM_LOB`, untuk tipe data blob.

Jika argument ketiga dari method `bindValue()` tidak diisi, nilai default adalah `PDO::PARAM_STR` (dianggap sebagai string).

Dengan demikian, perintah `$stmt->bindValue('batas', 3, PDO::PARAM_INT)` bisa dibaca: "tukar named parameter batas dengan angka 3 dan kirim ke MySQL sebagai tipe data integer". Hasilnya, query `SELECT...LIMIT` bisa diproses sebagaimana mestinya.

Di baris 8, pemanggilan method `execute()` tidak butuh lagi tambahan argument karena proses **bind** sudah kita lakukan terpisah.

Pemanggilan method `bindValue()` ini tidak untuk `LIMIT` saja, tapi juga bisa dipakai mengisi nilai prepared statement biasa. Jika terdapat lebih dari 1 nilai, method `bindValue()` harus dipanggil beberapa kali (satu untuk setiap nilai) seperti contoh berikut:

40.pdo_prepared_bind_value_2.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3 $query = "SELECT * FROM barang WHERE id_barang = :id OR
4           nama_barang = :barang";
5
6 $stmt = $pdo->prepare($query);
7 $stmt->bindValue('id', 5, PDO::PARAM_INT);
8 $stmt->bindValue('barang', "Printer Epson L220", PDO::PARAM_STR);
9 $stmt->execute();
10
11 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
12   echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
13   echo "<br>";
14 }
```

Hasil kode program:

```
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-21 08:31:33
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-21 08:31:33
```

Di baris 7-8 terdapat 2 kali pemanggilan method `bindValue()`, karena di dalam query juga butuh 2 buah nilai inputan, yakni `:id` dan `:barang`.

Dalam 2 contoh ini saya memakai penulisan *named parameter*, bagaimana jika prepared statement di tulis memakai *positional parameters*, yakni memakai tanda tanya ?

Untuk *positional parameters*, argument pertama dari method `bindValue()` diisi dengan urutan posisi tanda "?" dimulai dari angka 1 untuk tanda tanya pertama, angka 2 untuk tanda tanya kedua, dst. Jika menggunakan *positional parameters*, kode program sebelumnya bisa ditulis sebagai berikut:

41.pdo_prepared_bind_value_positional.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3 $query = "SELECT * FROM barang WHERE id_barang = ? OR
4           nama_barang = ?";
5
6 $stmt = $pdo->prepare($query);
7 $stmt->bindValue(1, 5, PDO::PARAM_INT);
8 $stmt->bindValue(2, "Printer Epson L220", PDO::PARAM_STR);
9 $stmt->execute();
```

Karena terdapat 2 buah tanda tanya, maka argument pertama dari method `bindValue()` ditulis sebagai 1 dan 2, sesuai dengan urutan tanda tanya yang ada di query.

Alternatif method untuk membuat proses bind secara manual adalah `PDOStatement::bindParam()`. Method ini juga butuh 3 buah argument sebagaimana method `bindValue()`, namun untuk `bindParam()`, nilai harus diisi dalam bentuk variabel, tidak bisa berisi angka langsung.

Berikut contoh kode program dari penggunaan `PDOStatement::bindParam()`:

42.pdo_prepared_bind_param.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang ORDER BY harga_barang LIMIT :batas";
5 $stmt = $pdo->prepare($query);
6 $stmt->bindParam('batas', $batas, PDO::PARAM_INT);
7 $batas=3;
8 $stmt->execute();
9
10 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11   echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
```

```
12 echo "<br>";
13 }
```

Hasil kode program:

```
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-21 08:31:33
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-21 08:31:33
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-21 08:31:33
```

Di sini saya kembali memakai query "LIMIT :batas". Perhatikan cara pemanggilan method `bindParam()` di baris 6. Sebagai argument kedua, kita tidak bisa langsung menginput angka 3 seperti di `bindValue()`, tapi harus diisi dengan variabel. Dalam contoh ini saya menggunakan variabel `$batas` yang isinya diinput di baris 7. Jika ditulis langsung sebagai `bindParam('batas', 3, PDO::PARAM_INT)` maka akan menghasilkan error.

Di dalam PDO, proses bind manual seperti ini memang bukan sebuah keharusan. Dalam banyak hal, menginput langsung argument ke dalam method `execute()` lebih praktis daripada memanggil method `bindValue()` atau `bindParam()`. Pengecualiannya untuk kasus-kasus khusus seperti LIMIT, atau inputan yang sensitif terhadap tipe data.

Exercise

Sebagai penutup serta menguji pemahaman tentang PDO (dan juga perintah query MySQL), saya tantang anda membuat kode program untuk meng-generate tabel **mahasiswa** serta menampilkan isinya.

Latihan ini mirip seperti kode program yang dipakai untuk me-reset tabel **barang**. Dimana jika kode ini dijalankan kembali, tabel **mahasiswa** otomatis di reset ulang. Urutan kode yang perlu dibuat adalah:

1. Buat koneksi dengan MySQL menggunakan PDO.
2. Buat database iloom jika belum ada (CREATE DATABASE).
3. Hapus tabel mahasiswa jika ada (DROP DATABASE).
4. Buat tabel mahasiswa (CREATE TABLE).
5. Isi tabel mahasiswa (INSERT).
6. Tampilkan isi tabel mahasiswa (SELECT).

Berikut tampilan dan isi dari tabel mahasiswa:

NIM	Nama	Tempat Lahir	Tanggal Lahir	Fakultas	Jurusan	IPK
13012012	James Situmorang	Medan	02 - 04 - 1995	Kedokteran	Kedokteran Gigi	2.70
14005011	Riana Putria	Padang	23 - 11 - 1996	FMIPA	Kimia	3.10
15002032	Rina Kumala Sari	Jakarta	28 - 06 - 1997	Ekonomi	Akuntansi	3.40
15021044	Rudi Permana	Bandung	22 - 08 - 1994	FASILKOM	Ilmu Komputer	2.90
15003036	Sari Citra Lestari	Jakarta	31 - 12 - 1997	Ekonomi	Manajemen	3.50

Isi tabel mahasiswa

Tipe data data untuk setiap kolom boleh bebas, yang penting semua data bisa tersimpan ke database. Khusus untuk kolom tanggal lahir, sesuaikan dengan format tipe data DATE dari MySQL, yakni dengan format yyyy-mm-dd.

Tips: karena dalam kode program ini perlu membuat database `ilkoom` dan menggunakananya, maka pada saat koneksi (pembuatan PDO object), kita tidak bisa langsung menggunakan database. Setelah database `ilkoom` di buat, baru pilih dengan perintah query "USE `ilkoom`".

Tips lagi: Silahkan lihat kode program yang dipakai untuk me-reset tabel barang sebagai panduan (file `bab09\20.mysql_generate.php`).

Berikut tampilan akhir ketika file dijalankan:

```

Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
Tabel 'mahasiswa' berhasil di buat
Tabel 'mahasiswa' berhasil di isi 5 baris data

Tabel Mahasiswa

13012012 | James Situmorang | Medan | 1995-04-02 | Kedokteran
14005011 | Riana Putria | Padang | 1996-11-23 | FMIPA
15002032 | Rina Kumala Sari | Jakarta | 1997-06-28 | Ekonomi
15003036 | Sari Citra Lestari | Jakarta | 1997-12-31 | Ekonomi
15021044 | Rudi Permana | Bandung | 1994-08-22 | FASILKOM

```

Proses pembuatan (reset) tabel mahasiswa

Selamat jika anda berhasil!

Apabila butuh untuk menyamakan kode program (atau menyerah), silahkan buka file `bab_10\43 pdo_exercise.php`.

Sepanjang bab ini kita telah membahas cara penggunaan PDO untuk mengakses database MySQL. Skill ini cukup penting karena jika membahas object oriented programming PHP, PDO adalah cara ideal untuk mengakses database.

Jika ke depannya anda ingin menggunakan framework (seperti Code Igniter atau Laravel), framework tersebut biasanya juga memiliki data *abstraction layer* sendiri, yakni "PDO versi framework". Dengan memahami cara kerja PDO, setidaknya anda sudah memiliki dasar yang kuat mengenai cara kerja koneksi database di PHP. Serta memiliki pilihan apakah ingin memakai cara yang dari framework, atau tetap menggunakan PDO.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

11. Case Study: Database Query Builder

Di awal buku saya pernah menulis bahwa proses pembuatan kode program dengan OOP butuh sebuah perencanaan. Alasannya karena dalam OOP kita dituntut untuk merancang kode program yang bisa di-“scaling”, yakni tetap mudah dikelola seiring meningkatnya kompleksitas aplikasi. Akan sangat repot jika harus merombak ulang seluruh kode program hanya untuk mengakomodasi beberapa fitur baru.

Misalnya saat ini kita diminta membuat aplikasi Sistem Informasi Sekolah untuk sebuah yayasan pendidikan. Yayasan ini memiliki 3 jenjang sekolah, yakni SD, SMP dan SMA. Setelah aplikasi jadi, ternyata ada permintaan untuk menambah jenjang PAUD (Pendidikan Anak Usia Dini).

Jika di awal kita sudah mempersiapkan semuanya dengan baik, maka tidak ada masalah. Penambahan ini bisa diatasi dengan membuat beberapa tabel baru di database. Sebaliknya, jika kode program dibuat tanpa perencanaan, sangat mungkin harus merombak total seluruh kode program.

Tiga prinsip dasar pemrograman object: *inheritance*, *polymorphism* dan *encapsulation* bisa dipakai untuk mengatasi masalah ini.

Salah satunya adalah dengan *encapsulation*, dimana kita memisahkan kode program menjadi modul-modul mandiri, terpisah dengan kode program utama. Misalnya kode program yang berhubungan dengan database menjadi satu modul, kode program untuk memproses form menjadi satu modul, kode program yang menangani tampilan menjadi 1 modul, dst.

Dengan perencanaan yang matang, setiap modul nantinya bisa dipakai ulang untuk aplikasi lain, tidak hanya untuk 1 aplikasi saja. Proses validasi form misalnya, akan selalu dibutuhkan untuk setiap halaman yang memiliki form. Daripada membuat proses validasi berulang kali, akan lebih baik kita buat sebuah modul terpisah.

Di dalam programming, modul-modul ini sering juga disebut sebagai **library**, **plugin**, atau **add-on** yang sama-sama berarti sebuah "kode tambahan". Namun proses pembuatan library memang butuh waktu yang kadang kala bisa lebih rumit daripada langsung membuat kode program tanpa library. Kita harus memikirkan bagaimana caranya agar library tersebut bersifat universal, mudah digunakan dan juga bisa dikembangkan lebih jauh lagi.

Terdapat 2 pilihan, apakah membuat library sendiri atau menggunakan library yang sudah jadi. PHP merupakan bahasa pemrograman yang sudah matang sehingga tersedia banyak pilihan

library yang ditulis oleh berbagai programmer professional. Untungnya lagi, mayoritas library ini bisa di dapat dengan gratis.

Kembali, karena konsep OOP yang memang mendorong pemisahan kode program, maka hampir semua library dibuat menggunakan pemrograman object.

Satu aplikasi biasanya butuh beberapa library, mulai dari mengakses database, memproses form, menangani cookie, hingga mengirim email. Beberapa library ada yang di-bundle menjadi 1 paket lengkap, dan itulah yang menjadi **framework**. Secara sederhana, framework terdiri dari kumpulan library siap pakai untuk memudahkan pembuatan aplikasi.

Sebagai contoh, jika menggunakan framework **Code Igniter**, proses mengambil tabel barang dari database cukup dengan 1 perintah berikut:

```
$query = $this->db->get('barang');
```

Jika kita menggunakan **mysqli** atau **PDO**, kode programnya tentu akan lebih panjang.

Dalam bab ini (dan juga bab berikutnya) kita tidak akan belajar cara menggunakan library yang sudah jadi, tapi membuat library itu sendiri. Studi kasus ini menjadi praktik yang bagus dalam penerapan konsep OOP.

11.1. MySQL Query Builder

Apa yang akan kita rancang adalah sebuah library untuk memudahkan pengaksesan database. Dalam framework PHP seperti *Code Igniter* atau *Laravel*, teknik ini dikenal sebagai **Query Builder**.

Nantinya, kita hanya perlu memanggil beberapa method untuk mengakses database, kemudian method itulah yang akan melakukan "kerja" membuat dan menjalankan perintah query MySQL. Sebagai contoh, untuk menampilkan seluruh tabel barang, kita bisa menggunakan kode berikut:

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->get('barang');
```

Di baris 2 saya menggunakan perintah `require` bawaan PHP untuk meng-import file `DB.php`. Di dalam file `DB.php` ini terdapat pendefinisian **class** `DB` yang berisi berbagai method. Inilah class yang akan kita rancang sesaat lagi.

Method `DB::getInstance()` di baris 3 dipakai untuk membuat object dari class `DB`. Perintah instansiasi ini akan memproses semua "urusan" dengan MySQL, yang salah satunya membuka koneksi ke database MySQL.

Di baris 5 saya menampung hasil dari perintah `$DB->get('barang')` ke dalam variabel `$tabelBarang`. Method `get()` adalah salah satu method yang tersedia di dalam class `DB`. Method ini dipakai untuk mengambil data tabel MySQL. Hasilnya, variabel `$tabelBarang` akan berisi seluruh data tabel `barang` dalam bentuk array.

Anda bisa perhatikan bahwa kita tidak menulis satu pun perintah query MySQL. Ini semua akan di proses oleh method `$DB->get()`.

Ketika method `$DB->get('barang')` dipanggil, class `DB` akan memproses argument `'barang'` menjadi query `"SELECT * FROM barang"`. Inilah yang dimaksud bahwa class `DB` yang akan kita rancang merupakan sebuah **query builder**, dimana library tersebut bisa meng-generate dan menjalankan query MySQL secara otomatis.

Contoh lain, proses input ke tabel `barang` bisa dilakukan dengan perintah berikut:

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->insert('barang',[  

6     'nama_barang' => 'Philips Blender HR 2157',  

7     'jumlah_barang' => 11,  

8     'harga_barang' => 629000  

9 ]);
```

Perintah di baris 2 dan 3 masih sama seperti sebelumnya.

Di baris 5 saya mengakses method `$DB->insert()` yang dipakai untuk proses `INSERT` ke database. Argument pertama method berupa nama tabel yang akan diinput, yakni `'barang'`. Kemudian argument kedua berisi *associative array* yang terdiri dari pasangan nama kolom dan nilai.

Apa yang akan dilakukan oleh class `DB` adalah mengkonversi perintah di baris 5 – 9, menjadi query berikut:

```
INSERT INTO barang ('nama_barang', 'jumlah_barang', 'harga_barang') VALUES  
( 'Philips Blender HR 2157', 11, 629000 )
```

Contoh terakhir, untuk menghapus data tabel bisa dilakukan dengan kode berikut:

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->delete('barang',[ 'id_barang' , '=' , 4 ]);
```

Perintah di baris 5 artinya saya ingin menghapus data di tabel `barang` dengan kondisi kolom `'id_barang' = 4`. Secara internal nantinya pemanggilan method ini akan dikonversi menjadi perintah query berikut:

```
DELETE FROM barang WHERE id_barang = 4
```

Itulah beberapa contoh method yang akan kita rancang. Total, class DB memiliki sekitar 15 method untuk berbagai keperluan, termasuk update data, pencarian dengan query LIKE, menangani kondisi WHERE, memeriksa nilai nilai unik, dll. Semuanya akan kita bahas secara bertahap.

Dapat juga dilihat bahwa class DB sangat memudahkan pembuatan aplikasi. Kode program di file utama menjadi lebih singkat karena "kerja keras" dalam mengakses database sudah ditangani oleh class DB.

Dalam perancangan class DB, saya akan memakai PDO *prepared statement*. Class DB di sini berperan sebagai sebuah "layer" atau "interface" di atas PDO. Dengan mengakses class DB, kita tidak perlu menulis langsung method-method PDO. Si pengguna library (bisa jadi itu adalah programmer lain), juga tidak perlu memahami prinsip kerja di dalam class DB, tapi cukup cara pakainya saja.

11.2. Class DB

Library database yang akan kita rancang adalah sebuah **class** yang saya beri nama DB. Class DB ini disimpan dalam file DB.php. Anda bebas jika ingin menggunakan nama class serta nama file yang berbeda. Proses pengaksesan class DB dilakukan dari file terpisah, yang dalam contoh ini saya akses dari index.php.

Dengan demikian, kita butuh 2 buah file: DB.php dan index.php. Sepanjang pembahasan nanti, kedua file ini akan diakses secara bergantian. Silahkan buka teks editor dan ketik kode program berikut ke dalam file DB.php:

```
01.DB_class/DB.php
```

```
1 <?php
2 class DB{
3
4 }
```

Yup, hanya berupa class kosong yang akan kita isi sesaat lagi. Berikutnya, buat file index.php yang berisi kode program berikut:

```
01.DB_class/index.php
```

```
1 <?php
2 require 'DB.php';
3 $DB = new DB();
4
5 echo "<pre>";
6 var_dump($DB);
7 echo "</pre>";
```

Di baris 2 saya menggunakan perintah `require` bawaan PHP untuk meng-import file `DB.php`. Karena langsung ditulis dengan nama file, artinya file `index.php` harus berada di dalam folder yang sama dengan file `DB.php`. Jika file `DB.php` berada di folder lain, silahkan tambah alamat path ke dalam perintah `require`.

Selanjutnya di baris 3 saya membuat variabel `$DB` sebagai object hasil instansiasi dari class `DB`. Object `$DB` ini kemudian ditampilkan dengan fungsi `var_dump()` di baris 5 – 7. Fungsi `var_dump()` di sini semata-mata untuk melihat apa isi dari variabel `$DB`.

Berikut hasil tampilan dari file `index.php`:

```
object(DB)#1 (0) {  
}
```

Hasilnya, perintah `var_dump($DB)` memperlihatkan bahwa variabel `$DB` sudah berisi object dari class `DB`.

11.3. Class DB: Constructor

Langkah pertama dalam pembuatan sebuah library database adalah membuka koneksi ke database server, yang dalam contoh kita menggunakan database MySQL (atau tepatnya MariaDB bawaan XAMPP).

Karena proses koneksi ini selalu dibutuhkan, akan sangat pas jika ditulis ke dalam constructor. Informasi seputar *host*, *nama database*, *user name* dan *password* MySQL akan saya simpan ke dalam *private property* seperti kode berikut:

02.DB_constructor/DB.php

```
1 <?php  
2 class DB{  
3  
4     // Property untuk koneksi ke database mysql  
5     private $_host = '127.0.0.1';  
6     private $_dbname = 'ilkoom';  
7     private $_username = 'root';  
8     private $_password = '';  
9  
10    // Property internal dari class DB  
11    private $_pdo;  
12 }
```

Dalam kode program ini saya membuat 5 buah property. Kelima property menggunakan hak akses `private` sehingga hanya bisa diakses dari dalam class `DB` saja.

Setiap property diawali dengan tanda garis bawah "`_`" (*underscore*) semata-mata agar mudah dibedakan dengan property yang memiliki hak akses lain. Maksudnya, jika sebuah property diawali dengan *underscore*, maka saya bisa pastikan itu adalah `private` property. Teknik ini

hanya kebiasaan beberapa programmer dan bukan sebuah kewajiban. Anda boleh mengikuti cara ini atau menggunakan nama variabel biasa.

Sesuai dengan namanya, property `$_host` dipakai untuk menyimpan alamat komputer host atau IP address dari komputer tempat MySQL server berada. Karena MySQL server ada di komputer yang sama, maka saya bisa memakai alamat 'localhost' atau IP address '127.0.0.1'.

Berikutnya property `$_dbname` dipakai untuk menampung nama database. Dalam contoh ini saya menggunakan database 'ilkoom'. Artinya, di MySQL server harus sudah tersedia database bernama `ilkoom`.

Property `$_username` dan `$_password` dipakai untuk menampung nama username dan password untuk mengakses ke MySQL Server. Dalam contoh ini saya akan login sebagai user `root` dengan password berupa string kosong.

Property terakhir yakni `$_pdo` di siapkan untuk menampung koneksi PDO dari PHP ke database MySQL. Isi untuk property ini akan kita tulis di dalam constructor sebagai berikut:

02.DB_constructor/DB.php

```

1 <?php
2 class DB{
3
4     // Property untuk koneksi ke database mysql
5     private $_host = '127.0.0.1';
6     private $_dbname = 'ilkoom';
7     private $_username = 'root';
8     private $_password = '';
9
10    // Property internal dari class DB
11    private $_pdo;
12
13    // Constructor untuk pembuatan PDO Object
14    public function __construct(){
15        try {
16            $this->_pdo = new PDO('mysql:host='.$this->_host.';dbname='.$this->_dbname,
17                                  $this->_username, $this->_password);
18            $this->_pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
19        } catch (PDOException $e){
20            die("Koneksi / Query bermasalah: ".$e->getMessage().
21                " (".$e->getCode()."')");
22        }
23    }
24
25 }
```

Kode program ini adalah sambungan dari sebelumnya.

Di dalam method constructor (baris 14 - 23), saya langsung memanggil class PDO untuk membuat koneksi ke database. Hasil koneksi ini ditampung ke dalam property `$this->_pdo` yang sudah kita siapkan sebelumnya. Sekedar pengingat, `$this` adalah variabel khusus yang

merujuk ke "object saat ini".

Sebagai argument dari pembuatan PDO object, saya merangkai isi dari empat private property: `$_host`, `$_dbname`, `$_username` dan `$_password`. Kode programnya tampak rumit karena penambahan variabel `$this` untuk mengakses setiap private property. Pada dasarnya perintah di baris 16 – 17 sama seperti berikut:

```
$this->_pdo = new PDO("mysql:host=127.0.0.1;dbname=ilkoom", "root", "");
```

Selanjutnya baris 18 berisi pengaturan agar pesan error di tampilkan sebagai sebuah exception. Exception ini akan di tangkap oleh kode program di baris 19 – 22. Tidak ada hal yang baru di sini karena sama persis seperti yang kita pakai pada bab tentang PDO.

Saya memakai fungsi `die()` untuk menampilkan pesan error. Artinya jika class `DB` gagal mengakses database, fungsi `die()` akan langsung menghentikan kode program.

Sebagai uji coba, silahkan akses kembali file `index.php`. Jika tidak ada masalah akan tampil hasil berikut:

```
object(DB)#1 (5) {  
    ["_host":"DB":private]=>  
    string(9) "127.0.0.1"  
    ["_dbname":"DB":private]=>  
    string(6) "ilkoom"  
    ["_username":"DB":private]=>  
    string(4) "root"  
    ["_password":"DB":private]=>  
    string(0) ""  
    ["_pdo":"DB":private]=>  
    object(PDO)#2 (0) {  
    }  
}
```

Hasil ini berasal dari perintah `var_dump($DB)` di dalam file `index.php`. Isinya memperlihatkan seluruh property yang ada di dalam class `DB`. Perhatikan bahwa property `_pdo` di bagian terakhir sudah berisi `object(PDO)`, artinya proses koneksi dengan database MySQL sukses dilakukan.

Untuk menguji jika terjadi kesalahan, saya akan tukar isi property `$_password` di dalam class `DB` dengan sebuah karakter 'x':

```
private $_password = 'x';
```

Berikut hasil dari `index.php`:

```
Koneksi / Query bermasalah: SQLSTATE[HY000] [1045] Access denied for user  
'root'@'localhost' (using password: YES) (1045)
```

Tampil pesan error karena seharusnya user `root` tidak memiliki password. Namun pesan error

ini juga menunjukkan bahwa block try – catch yang kita tulis sudah berhasil menangkap exception. Silahkan ubah kembali isi property `$_password` menjadi string kosong.

Sepanjang bab ini saya akan mengakses tabel barang yang ada di database `ilkoom`. Tabel barang ini sudah kita buat sebelumnya, atau anda bisa jalankan file `generate_tabel_barang.php` untuk membuat ulang tabel barang beserta isinya.

11.4. Class DB: Singleton Pattern

Constructor class `DB` yang baru saja kita tulis sudah berjalan dengan baik, namun juga masih bisa disempurnakan lebih lanjut.

Untuk aplikasi yang kompleks, dalam satu halaman kadang perlu mengakses database MySQL beberapa kali. Jika halaman tersebut dibuat secara berurutan dari atas ke bawah, proses instansiasi class `DB` hanya perlu dilakukan sekali di awal (biasanya di baris paling atas).

Namun jika halaman tersebut terdiri dari beberapa modul atau library yang saling memanggil satu sama lain, ada kemungkinan class `DB` di instansiasi lebih dari 1 kali. Pada kode program kita saat ini, setiap kali class `DB` di instansiasi, koneksi ke database MySQL akan selalu dibuat ulang.

Sebagai praktek, silahkan buka file `DB.php` lalu tambah 1 perintah `echo` ke dalam constructor class `DB`:

03.DB_constructor_problem/DB.php

```
1 <?php
2 class DB{
3 ...
4 ...
5     public function __construct(){
6         try {
7             ...
8                 $this->_pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
9                 echo "Koneksi dibuat <br>";
10            } catch (PDOException $e){
11                ...
12            }
13        }
14    }
```

Saya menyisipkan perintah `echo "Koneksi dibuat
"` di baris 9 agar kita bisa melihat hasil tampilan setiap kali constructor class `DB` diakses.

Selanjutnya, buka file `index.php` dan isi dengan kode berikut:

03.DB_constructor_problem/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = new DB();
4 $DB = new DB();
5 $DB = new DB();
```

Dalam file `index.php` ini saya melakukan 3 kali instansiasi class `DB`. Bagaimana hasilnya?

```
Koneksi dibuat
Koneksi dibuat
Koneksi dibuat
```

Seperti yang terlihat, constructor class `DB` akan dipanggil sebanyak 3 kali, yakni satu untuk setiap proses instansiasi. Artinya, koneksi ke MySQL juga akan dibuat sebanyak 3 kali.

Dalam kondisi ideal, koneksi ke MySQL cukup dilakukan 1 kali saja meskipun class `DB` dibuat berulang kali. Salah satu solusi untuk menerapkan konsep seperti ini adalah menggunakan teknik *singleton pattern*, yang merupakan bagian dari konsep pemrograman object yang disebut sebagai **Design Pattern**.

Sekilas Tentang Design Pattern

Dalam ilmu programming (terutama pemrograman berorientasi objek) terdapat istilah yang dikenal sebagai **Design Pattern**. Secara sederhana, *design pattern* adalah teknik atau cara penulisan tertentu untuk membuat kode program "berperilaku khusus".

Perilaku khusus yang saya maksud adalah kode program tersebut bisa dibentuk supaya mengikuti aturan-aturan tertentu, misalnya hanya bisa di instansiasi 1 kali saja, atau sebuah object hanya bisa dibuat dengan memanggil method dari class induk.

Istilah **Design Pattern** mulai populer sejak tahun 1995, yang berasal dari buku "*Design Patterns: Elements of Reusable Object-Oriented Software*". Buku ini ditulis oleh 4 ilmuwan komputer ternama yang dikenal sebagai **The "Gang of Four"**, yakni Erich Gamma, Richard Helm, Ralph Johnson dan John Vlissides ([wikipedia](#)).

Buku tersebut berisi 23 pola atau programming pattern, diantaranya *builder pattern*, *factory method pattern*, *adapter pattern*, dan termasuk juga *singleton pattern*. Sampai saat ini buku tersebut masih menjadi patokan dasar bagi yang ingin belajar *design pattern*.

Apakah saya juga harus belajar design pattern?

Idealnya **iya**, karena teknik *design pattern* ini dibuat berdasarkan pengalaman programmer-programmer sebelumnya yang sudah lebih dulu menemui masalah yang sama.

Namun design pattern termasuk materi advanced yang cukup rumit. Agar bisa memahami design pattern, seorang programmer harus paham konsep pemrograman object secara mendalam serta pernah beberapa kali membuat project dengan konsep OOP.

Selain itu ada juga pendapat bahwa *design pattern* ini terlalu rumit dan relatif jarang dipakai dalam pembuatan aplikasi "normal". Kebanyakan pola / *pattern* baru diperlukan jika anda membuat library atau framework sendiri. Jika hanya sebagai pengguna framework (seperti yang akan banyak kita lakukan), cukup ikuti alur yang sudah ditetapkan oleh framework tersebut.

Saran saya, jika anda memang berencana menjadi programmer professional, *design pattern* ini menjadi salah satu materi yang diperlukan. Seseorang yang sudah menguasai *design pattern*, pasti mahir dalam pemrograman berorientasi objek. Sebaliknya, bagi yang paham dengan OOP belum tentu menguasai *design pattern*.

Tantangan lain (setidaknya untuk saat ini) anda harus mencari referensi dalam buku bahasa inggris karena materi *design pattern* yang berbahasa indonesia masih cukup langka.

Kembali ke class DB, kita akan rancang kode program untuk membuat **singleton pattern**. Tujuannya agar constructor class DB hanya bisa dipanggil 1 kali saja.

Pertama, buat sebuah `static` property yang nanti akan dipakai untuk menampung object dari class DB. Property ini saya beri nama `$_instance` dan diberi nilai awal `null`. :

```
private static $_instance = null;
```

Berikutnya, ubah hak akses constructor class DB menjadi `private`:

```
private function __construct(){
    ...
}
```

Loh, kalau constructor dibuat private, bukannya kita tidak akan bisa meng-instansiasi class DB?

Betul, karena constructor menjadi private, kita tidak akan bisa membuat object dari class DB **di luar class**. Namun proses pembuatan object masih bisa dilakukan dari dalam class DB itu sendiri, yakni dari static method `getInstance()` berikut:

```
1 public static function getInstance(){
2     if(!isset(self::$_instance)) {
3         self::$_instance = new DB();
4     }
5     return self::$_instance;
```

```
6 }
```

Inilah method yang digunakan untuk membuat **singleton pattern**. Kondisi `if` di baris 2 akan memeriksa apakah property `$_instance` berisi sesuatu atau tidak, yakni `if(!isset(self::$_instance))`.

Jika property `$_instance` tidak berisi sesuatu (atau berisi `null`), maka perintah di baris 3 akan dijalankan. Perintah ini dipakai untuk mengisi property `$_instance` dengan object dari class `DB`. Perhatikan cara penulisannya, yakni `self::$_instance = new DB()`. Sebagai pengingat, `self` adalah keyword khusus yang merujuk ke class saat ini. Kita tidak menggunakan `$this` karena `$_instance` adalah sebuah static property.

Jika ternyata property `$_instance` berisi sesuatu (`not null`), maka kondisi `if` akan dilewati dan method ini mengembalikan isi dari property `$_instance` tersebut.

Berikut kode program lengkap dari file `DB.php`:

04.DB_constructor_singleton/DB.php

```

1 <?php
2 class DB{
3
4     // Property untuk koneksi ke database mysql
5     private $_host = '127.0.0.1';
6     private $_dbname = 'ilkoom';
7     private $_username = 'root';
8     private $_password = '';
9
10    // Property internal dari class DB
11    private static $_instance = null;
12    private $_pdo;
13
14    // Constructor untuk pembuatan PDO Object
15    private function __construct(){
16        try {
17            $this->_pdo = new PDO('mysql:host='.$this->_host.';dbname='.$this->_dbname,
18                                  $this->_username, $this->_password);
19            $this->_pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
20            echo "Koneksi dibuat <br>";
21        } catch (PDOException $e){
22            die("Koneksi / Query bermasalah: ".$e->getMessage().
23                 " (".$e->getCode()."")");
24        }
25    }
26
27    // Singleton pattern untuk membuat class DB
28    public static function getInstance(){
29        if(!isset(self::$_instance)) {
30            self::$_instance = new DB();
31        }
32        return self::$_instance;
33    }

```

```
34  
35 }
```

Sekarang proses pembuatan object dari class DB juga sudah berubah. Kita tidak bisa lagi menggunakan perintah `$DB = new DB()` karena constructor class DB sudah "di kunci" dengan hak akses `private`. Proses pembuatan object dari class DB sekarang menggunakan method `DB::getInstance()`.

Berikut isi file `index.php` yang dipakai untuk membuat object class DB:

04.DB_constructor_singleton/index.php

```
1 <?php  
2 require 'DB.php';  
3  
4 $DB = DB::getInstance();  
5  
6 echo "<pre>";  
7 var_dump($DB);  
8 echo "</pre>";
```

Hasil kode program:

Koneksi dibuat

```
object(DB)#1 (5) {  
    ["_host":"DB":private]=>  
    string(9) "127.0.0.1"  
    ["_dbname":"DB":private]=>  
    string(6) "ilkoom"  
    ["_username":"DB":private]=>  
    string(4) "root"  
    ["_password":"DB":private]=>  
    string(0) ""  
    ["_pdo":"DB":private]=>  
    object(PDO)#2 (0) {  
    }  
}
```

Di sini proses pembuatan object dari class DB di lakukan dengan perintah `$DB = DB::getInstance()`. Hasil fungsi `var_dump($DB)` memperlihatkan bahwa variabel `$DB` sudah berisi object dari class DB.

Inilah cara pembuatan **singleton pattern**, dimana class DB hanya bisa di-instansiasi satu kali saja. Atau lebih tepatnya, constructor class DB hanya bisa diakses 1 kali sepanjang kode program. Sebagai pembuktian, kita akan coba panggil method `DB::getInstance()` beberapa kali:

04.DB_constructor_singleton/index.php

```
1 <?php  
2 require 'DB.php';
```

```
3  
4  $DB = DB::getInstance();  
5  $DB = DB::getInstance();  
6  $DB = DB::getInstance();
```

Hasil kode program:

Koneksi dibuat

Teks "Koneksi dibuat" hanya muncul 1 kali. Teks ini berasal dari perintah echo yang sebelumnya kita sisip ke dalam constructor class DB. Ini membuktikan bahwa constructor dari class DB hanya dijalankan 1 kali saja, meskipun terdapat 3 buah perintah untuk membuat object class DB.

Dengan demikian, class DB menjadi lebih efisien. Kita bisa memastikan bahwa koneksi ke database MySQL hanya akan dilakukan 1 kali sepanjang kode program, tidak peduli berapa kali class DB di-instansiasi.

Sebelum kita lanjut ke materi berikutnya, silahkan hapus kembali perintah echo "Koneksi dibuat
" dari constructor class DB. Dan untuk menghemat tempat, saya tidak menulis lagi kode program untuk constructor class DB serta method getInstance(). Sepanjang sisa pembahasan, kedua method ini dianggap sudah tersedia di dalam class DB.

11.5. Class DB: Method runQuery

Kita sudah berhasil membuat koneksi ke database MySQL, serta memastikan koneksi hanya dilakukan 1 kali saja dengan teknik *singleton pattern*. Sekarang kita akan buat method untuk memproses query, yakni runQuery().

Method runQuery() saya rancang sebagai method "generik". Maksudnya, method ini akan dipakai sebagai dasar untuk method-method lain. Method runQuery() hanya berfungsi untuk menjalankan query saja, belum sampai meng-generate perintah query secara otomatis.

Berikut kode program yang dipakai untuk membuat method runQuery():

05.DB_method_runQuery/DB.php

```
1  <?php  
2  class DB{  
3  
4      // kode program untuk property dan constructor class DB  
5      // ...  
6      // ...  
7  
8      public function runQuery($query, $bindValue = []){  
9          try {  
10              $stmt = $this->_pdo->prepare($query);  
11              $stmt->execute($bindValue);  
12          }  
13      }
```

```

13     catch (PDOException $e){
14         die("Koneksi / Query bermasalah: ".$e->getMessage().
15             " (".$e->getCode()."')");
16     }
17     return $stmt;
18 }
19
20 }
```

Method `runQuery()` menerima 2 buah argument. Argument pertama, yakni `$query` dipakai untuk menampung perintah query dalam bentuk string, misalnya `'SELECT * FROM barang'` sedangkan argument kedua yakni `$bindValue` dipakai untuk menampung nilai prepared statement dalam bentuk array. Argument `$bindValue` memiliki nilai default berupa array kosong sehingga bersifat opsional (boleh tidak ditulis).

Baris pertama method `runQuery()` langsung disambut dengan block try-catch. Di baris 10 saya membuat variabel `$stmt` yang dipakai untuk menampung hasil perintah `$this->_pdo->prepare($query)`. Hasilnya, variabel `$stmt` akan berisi sebuah **PDOStatement object**.

Jika anda lihat kembali constructor class `DB`, property `$this->_pdo` saya buat untuk menampung hasil koneksi ke MySQL. Dengan demikian perintah `$this->_pdo->prepare($query)` berfungsi menjalankan sebuah query MySQL.

Di baris 11 terdapat perintah `$stmt->execute($bindValue)`. Perintah ini dipakai untuk proses **bind** dan **execute** dari PDO prepared statement. Nilai yang ada di dalam array `$bindValue` akan berpasangan dengan tanda tanya atau *placeholder* `"?"` yang nantinya terdapat di string `$query`.

Selanjutnya terdapat block **catch** di baris 13 – 16 yang dipakai untuk menampilkan pesan error jika terdapat kesalahan perintah query. Terakhir di baris 17 saya mengembalikan isi variabel `$stmt`.

Method `runQuery()` ini mungkin cukup susah dipahami karena kita belum lihat bagaimana bentuk prakteknya. Berikut contoh pemanggilan method `runQuery()` dari file `index.php`:

05.DB_method_runQuery/index.php

```

1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->runQuery('SELECT * FROM barang');
6 $tabelBarang = $result->fetchAll(PDO::FETCH_OBJ);
7
8 echo "<pre>";
9 print_r($tabelBarang);
10 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
    (
        [id_barang] => 1
        [nama_barang] => TV Samsung 43NU7090 4K
        [jumlah_barang] => 5
        [harga_barang] => 5399000
        [tanggal_update] => 2019-03-11 17:35:28
    )

    [1] => stdClass Object
    (
        [id_barang] => 2
        [nama_barang] => Kulkas LG GC-A432HLHU
        [jumlah_barang] => 10
        [harga_barang] => 7600000
        [tanggal_update] => 2019-03-11 17:35:28
    )
    ...
    ...
)
```

Perintah di baris 2 – 3 dipakai untuk membuat object dari class DB. Object ini kemudian disimpan ke dalam variabel \$DB.

Di baris 4 saya mengakses method runQuery() dengan perintah \$result = \$DB->runQuery('SELECT * FROM barang'). Artinya, string 'SELECT * FROM barang' dikirim sebagai argument pertama untuk method runQuery(). Di dalam method runQuery(), string ini akan dijalankan sebagai perintah query MySQL dan menghasilkan sebuah **PDO Statement object**.

PDO Statement object ini saya tampung ke dalam variabel \$result. Untuk menampilkan data tabel, di baris 5 saya menjalankan perintah \$tabelBarang = \$result->fetchAll(PDO::FETCH_OBJ). Hasilnya, variabel \$tabelBarang berisi seluruh tabel barang dalam bentuk array object.

Jika saya ingin menampilkan 1 baris pertama dari tabel barang dengan perintah echo, bisa menggunakan kode berikut:

05.DB_method_runQuery/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->runQuery('SELECT * FROM barang');
6 $tabelBarang = $result->fetchAll(PDO::FETCH_OBJ);
7
8 echo $tabelBarang[0]->id_barang." | ";
9 echo $tabelBarang[0]->nama_barang." | ";
10 echo $tabelBarang[0]->jumlah_barang." | ";
11 echo $tabelBarang[0]->harga_barang." | ";
```

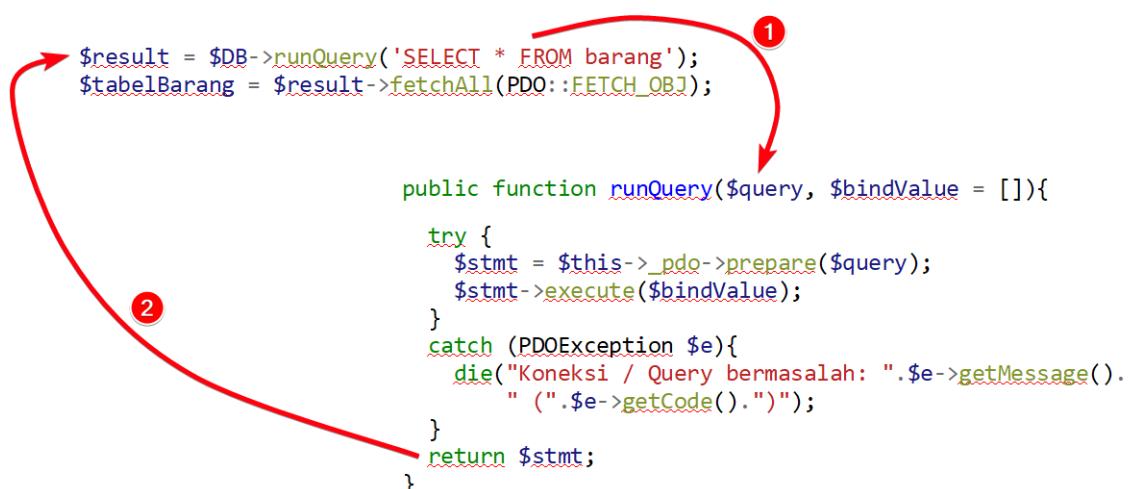
```
12 echo $tabelBarang[0]->tanggal_update." | ";
```

Hasil kode program:

```
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-03-11 17:35:28 |
```

Intinya adalah bagaimana cara menampilkan array tabel yang dipanggil dengan metode `fetchAll(PDO::FETCH_OBJ)`. Penjelasan lebih lanjut sudah kita bahas dalam bab tentang PDO.

Saya sangat sarankan untuk meluangkan waktu sejenak memahami alur proses pemanggilan method `runQuery()` ini. Terutama mengenai cara string 'SELECT * FROM barang' di kirim dan diterima oleh method `runQuery()`.



Ilustrasi pemanggilan method runQuery()

Di sini method `runQuery()` hanya dipakai untuk menjalankan query saja. Pada saat method dijalankan, string 'SELECT * FROM barang' akan dikirim ke dalam variabel `$query` sebagai argument pertama (1). Kemudian method `runQuery()` akan memproses query tersebut dan menghasilkan **PDO statement object** yang disimpan ke dalam variabel `$stmt`. Variabel `$stmt` kemudian di-return ke dalam variabel `$result` (2).

Tapi tunggu dulu, bagaimana dengan perintah `$stmt->execute($bindValue)`? Meskipun agak 'aneh', prepared statement tetap bisa dipakai untuk query yang tidak memiliki data *placeholder*. Karena dalam contoh ini kita menjalankan query tanpa nilai *placeholder*, yang akan dijalankan adalah sebagai berikut:

```
try {
    $stmt = $this->_pdo->prepare('SELECT * FROM barang');
    $stmt->execute([]);
}
```

Sekarang mari kita panggil method `runQuery()` dengan data placeholder:

06.DB_method_runQuery_prepared/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->runQuery('SELECT * FROM barang WHERE id_barang = ?',[4]);
6 $tabelBarang = $result->fetchAll(PDO::FETCH_OBJ);
7
8 echo "<pre>";
9 print_r($tabelBarang);
10 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 4
            [nama_barang] => Printer Epson L220
            [jumlah_barang] => 14
            [harga_barang] => 2099000
            [tanggal_update] => 2019-03-11 17:35:28
        )
)
```

Di baris 5 method `runQuery()` dipanggil dengan 2 buah argument. Argument pertama berupa string `'SELECT * FROM barang WHERE id_barang = ?'` dan argument kedua berupa array [4]. Dalam perintah query saya menambah clausa `WHERE id_barang = ?`, di sini terdapat tanda tanya `'?'` yang nilainya akan diambil dari argument kedua, yakni 4.

Dalam method `runQuery()`, perintah yang dijalankan adalah sebagai berikut:

```
try {
    $stmt = $this->_pdo->prepare('SELECT * FROM barang WHERE id_barang = ?');
    $stmt->execute([4]);
}
```

Artinya saya ingin mengambil data di tabel barang yang memiliki kolom `id_barang = 4`.

Agar lebih rapi, argument method `runQuery()` bisa ditulis sebagai variabel terpisah:

06.DB_method_runQuery_prepared/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $query = 'SELECT * FROM barang WHERE id_barang = ?';
6 $arr = [4];
7
8 $result = $DB->runQuery($query,$arr);
```

```
9 $tabelBarang = $result->fetchAll(PDO::FETCH_OBJ);
10
11 echo "<pre>";
12 print_r($tabelBarang);
13 echo "</pre>";
```

Di sini saya memisahkan string query ke dalam variabel \$query di baris 5 dan data placeholder ke variabel \$arr di baris 6.

Sebagai latihan, bisakah anda menjalankan sebuah query `INSERT` menggunakan method `getQuery()`? Data yang diinput ke tabel barang adalah sebagai berikut:

- ✓ nama_barang = 'Cosmos CRJ-8229 - Rice Cooker'
- ✓ jumlah_barang = 4
- ✓ harga_barang = 299000

Syarat tambahan: gunakan prepared statement. Silahkan coba rancang kode programnya.

Baik, berikut kode yang saya gunakan:

07.DB_method_runQuery_insert/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $query = 'INSERT INTO barang (nama_barang, jumlah_barang, harga_barang)
6           VALUES (?, ?, ?)';
7 $arr = ['Cosmos CRJ-8229 - Rice Cooker', 4, 299000];
8
9 // jalankan proses insert
10 $DB->runQuery($query, $arr);
11
12 // tampilkan semua tabel barang
13 $result = $DB->runQuery('SELECT * FROM barang');
14 $tabelBarang = $result->fetchAll(PDO::FETCH_OBJ);
15
16 echo "<pre>";
17 print_r($tabelBarang);
18 echo "</pre>";
```

Hasil kode program:

```
Array
(
    ...
    ...
    [5] => stdClass Object
        (
            [id_barang] => 6
            [nama_barang] => Cosmos CRJ-8229 - Rice Cooker
            [jumlah_barang] => 4
```

```
[harga_barang] => 299000  
[tanggal_update] => 2019-03-12 07:59:53  
)  
)
```

Method `runQuery()` pada dasarnya bisa menerima query apapun. Jika query tersebut berbentuk prepared statement, maka untuk setiap tanda tanya "?" harus berpasangan dengan data placeholder.

Dalam variabel `$query` terdapat 3 buah tanda tanya, maka akan berpasangan dengan 3 buah nilai di dalam variabel `$arr`. Di baris 13 – 18 saya menampilkan kembali isi seluruh tabel barang untuk memastikan query `INSERT` sudah berhasil berjalan.

Sedikit tambahan, dalam query `INSERT` di baris 5 saya tidak menginput nilai tanggal untuk kolom `tanggal_update`. Namun ketika ditampilkan, kolom ini sudah terisi sendiri.

Pada saat pembuatan tabel barang (dari file `generate_tabel_barang.php`), kolom `tanggal_update` di set sebagai `timestamp`. Di dalam MySQL, kolom dengan tipe data `timestamp` otomatis berisi tanggal server sekarang apabila tidak diinput. Ini kurang lebih sama seperti kolom `id_barang` yang di set sebagai `AUTO_INCREMENT`, dimana nilainya juga otomatis di generate oleh MySQL.

11.6. Class DB: Method `getQuery`

Method `getQuery()` merupakan sedikit tambahan dari method `runQuery()`. Method ini saya rancang agar langsung mengembalikan data tabel dalam bentuk array object, yakni hasil dari method `fetchAll(PDO::FETCH_OBJ)`. Secara internal, method `getQuery()` juga memanggil method `runQuery()`.

Berikut kode program dari method `getQuery()`:

08.DB_method_getQuery/DB.php

```
1 <?php  
2 class DB{  
3  
4     // kode program untuk property dan constructor class DB  
5     // kode program untuk method runQuery()  
6     // ...  
7  
8     public function getQuery($query,$bindValue = []){  
9         return $this->runQuery($query,$bindValue)->fetchAll(PDO::FETCH_OBJ);  
10    }  
11 }
```

Seperti yang terlihat, di dalam method `getQuery()` ini saya memanggil ulang method `runQuery()`, namun dengan tambahan langsung disambung dengan memanggil method `fetchAll(PDO::FETCH_OBJ)`.

Dengan penulisan seperti ini, maka method `getQuery()` secara khusus dipakai untuk query `SELECT`. Berikut cara penggunaan method ini dari file `index.php`:

08.DB_method_getQuery/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 // tampilkan semua tabel barang
6 $tabelBarang = $DB->getQuery('SELECT * FROM barang WHERE id_barang = ?',[2]);
7
8 echo "<pre>";
9 print_r($tabelBarang);
10 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 2
            [nama_barang] => Kulkas LG GC-A432HLHU
            [jumlah_barang] => 10
            [harga_barang] => 7600000
            [tanggal_update] => 2019-03-12 07:59:50
        )
)
```

Hasil akhir dari method `getQuery()` sudah langsung berbentuk array object yang saya simpan ke dalam variabel `$tabelBarang`. Untuk query `SELECT`, akan lebih praktis jika menggunakan method `getQuery()`, karena kita tidak perlu memanggil lagi method `fetchAll(PDO::FETCH_OBJ)`.

Namun kelemahannya, method `getQuery()` "sudah terkunci" untuk menampilkan hasil dalam bentuk array object. Jika anda ingin mengambil nilai tabel dalam bentuk *associative array* atau *numeric array* maka silahkan pakai method `runQuery()` dan jalankan method `fetchAll(PDO::FETCH_ASSOC)` atau `fetchAll(PDO::FETCH_NUM)` secara manual.

11.7. Class DB: Method get

Kita sudah membuat 2 buah method dasar, yakni `runQuery()` dan `getQuery()`. Kali ini saya akan masuk ke method yang berbentuk query builder, dimana kita tidak perlu lagi menulis query secara langsung. Method ini saya beri nama `get()`, yang berfungsi untuk mengambil semua isi dari sebuah tabel. Method `get()` ini cuma butuh 1 argument, yakni nama tabel yang ingin ditampilkan.

Berikut cara pengaksesan method get() dari file index.php:

09.DB_method_get/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 // tampilkan semua tabel barang
6 $tabelBarang = $DB->get('barang');
7
8 echo "<pre>";
9 print_r($tabelBarang);
10 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            [harga_barang] => 5399000
            [tanggal_update] => 2019-03-12 07:59:50
        )
    ...
)
```

Perhatikan baris ke 6, itulah cara pemanggilan method get(). Jika dipanggil \$DB->get('barang'), maka akan mengembalikan array object dari tabel barang. Atau jika dipanggil \$DB->get('user'), akan mengembalikan array object dari tabel user.

Silahkan anda coba rancang method get() ini di dalam class DB.

Baik, berikut kode yang saya pakai untuk membuat method get():

09.DB_method_get/DB.php

```
1 <?php
2 class DB{
3
4     // kode program untuk property dan constructor class DB
5     // kode program untuk method runQuery() dan getQuery()
6     // ...
7
8     public function get($tableName){
9         $query = "SELECT * FROM {$tableName}";
10        return $this->getQuery($query);
11    }
}
```

```
12 }
```

Isi dari method `get()` hanya 2 baris, dimana pada baris 9 saya membuat variabel `$query` yang berisi string "SELECT * FROM {\$tableName}". Variabel `$tableName` sendiri berasal dari argument method.

Di baris 10, terdapat perintah `return $this->getQuery($query)`. Artinya, method `get()` akan mengembalikan nilai hasil pemanggilan method `getQuery($query)`.

Silahkan anda pahami sebentar alur proses pemanggilan method `get()` karena terdapat "pemanggilan berantai". Di dalam method `get()`, prosesnya diserahkan ke dalam method `getQuery()`. Kemudian di dalam method `getQuery()`, proses kembali diserahkan ke dalam method `runQuery()`.

Meskipun terkesan rumit, tapi pemanggilan berantai seperti ini membuat kode program menjadi lebih efisien karena kita bisa menghindari adanya kode program yang berulang.

Saya bisa saja menulis method `get()` sebagai berikut:

```
1 <?php
2 class DB{
3
4     public function get($tableName,$bindValue = []){
5         $query = "SELECT * FROM {$tableName}";
6         try {
7             $stmt = $this->_pdo->prepare($query);
8             $stmt->execute($bindValue);
9         }
10        catch (PDOException $e){
11            die("Koneksi / Query bermasalah: ".$e->getMessage().
12                " (".$e->getCode()."')");
13        }
14        return $stmt->fetchAll(PDO::FETCH_OBJ);
15    }
16 }
```

Dengan penulisan seperti ini, method `get()` bisa berdiri sendiri, tidak bergantung dengan method lain seperti `getQuery()`.

Namun kode program untuk class `DB` akan banyak yang sama. Proses exception misalnya, sudah kita buat di dalam method `runQuery()`. Akan lebih efisien jika proses exception ini diserahkan ke method tersebut agar cukup di satu tempat saja, tidak perlu ditulis berulang di setiap method. Begitu juga halnya dengan method `fetchAll(PDO::FETCH_OBJ)` yang juga sudah tersedia di dalam method `getQuery()`.

Nantinya kita akan buat sekitar 8 method lagi, yang jika tidak dipisah maka proses penanganan exception perlu ditulis ulang di setiap method.

Skill untuk bisa memecah sebuah method menjadi method-method kecil seperti ini memang sangat berguna, tapi hanya bisa dilakukan dengan banyaknya latihan **re-factoring** kode

program, yakni menulis ulang kode yang sudah jadi dan membuatnya lebih efisien lagi.

Sedikit bocoran, seluruh method untuk class DB ini saya tulis ulang sekitar 3 kali. Setiap kali penulisan, saya mencoba membuat class DB lebih efisien lagi dengan cara memecah method untuk mengurangi kode yang berulang.

11.8. Class DB: Method select

Method `select()` yang akan kita buat ini masih berhubungan dengan method `get()`. Saya ingin merancang sebuah mekanisme agar bisa memilih kolom mana saja yang akan ditampilkan. Dengan kode yang ada sekarang, method `get()` selalu menampilkan semua kolom yang ada di tabel.

Untuk cara kerja method `select()` ini saya terinspirasi dengan *query builder* bawaan framework **Code Igniter**. Untuk menentukan kolom yang akan diambil, kita akan panggil method `select()` terlebih dahulu.

Berikut konsep penggunaan method `select()` dari file `index.php`:

10.DB_method_get_select/index.php

```

1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 // pilih kolom nama_barang
6 $DB->select('nama_barang');
7 $tabelBarang = $DB->get('barang');
8
9 echo "<pre>";
10 print_r($tabelBarang);
11 echo "</pre>";

```

Hasil kode program:

```

Array
(
    [0] => stdClass Object
        (
            [nama_barang] => TV Samsung 43NU7090 4K
        )

    ...
    ...

    [5] => stdClass Object
        (
            [nama_barang] => Cosmos CRJ-8229 - Rice Cooker
        )

```

)

Di baris 6 terdapat perintah `$DB->select('nama_barang')`. Method `select()` ini akan men-set "sesuatu" di class `DB`, sehingga ketika dipanggil method `$DB->get('barang')`, akan berisi kolom `nama_barang` saja.

Bagaimana cara membuatnya?

Pertama, di dalam class `DB` saya akan tambah sebuah private property `$_columnName`. Property ini dipakai untuk menampung nama kolom dari hasil pemanggilan method `select()`:

```
private $_columnName = "*";
```

Property `$_columnName` berisi nilai awal karakter bintang '*' agar ketika method `select()` tidak dipanggil, maka tampilkan semua kolom.

Berikutnya, buat method `select()` yang dipakai untuk mengubah isi property `$_columnName`:

```
public function select($columnName){  
    $this->_columnName = $columnName;  
}
```

Artinya, ketika method `select('nama_barang')` dipanggil, string 'nama_barang' akan menimpa isi property `$_columnName`.

Terakhir, kita perlu modifikasi method `get()` sebagai berikut:

10.DB_method_get_select/DB.php

```
1  public function get($tableName){  
2      $query = "SELECT {$this->_columnName} FROM {$tableName}";  
3      $this->_columnName = "*";  
4      return $this->getQuery($query);  
5  }
```

Perubahannya ada di string `$query`. Setelah perintah `SELECT`, terdapat pemanggilan variabel `$this->_columnName`. Dalam perintah query MySQL, bagian ini adalah tempat untuk menulis nama kolom.

Misalnya jika kita ingin menampilkan kolom `nama_barang` dari tabel `barang`, query yang harus ditulis adalah `'SELECT nama_barang FROM barang'`. Atau jika ingin menampilkan seluruh kolom, maka querinya adalah `'SELECT * FROM barang'`. Bagian nama kolom inilah yang kita ambil dari property `$this->_columnName`.

Di baris 3 terdapat tambahan perintah `$this->_columnName = "*"`. Ini dipakai untuk me-reset ulang isi property `$_columnName` agar kembali menjadi '*'. Karena kalau tidak, pemanggilan method `get()` berikutnya akan terpengaruh dengan pemanggilan method `select()` sebelumnya.

Dengan tambahan ini, method `get()` menjadi lebih fleksibel karena kita bisa menentukan

nama kolom yang ingin diambil. Jika method `select()` tidak dipanggil, artinya tampilan seluruh kolom.

Berikut contoh pemanggilan method `select()` dan `get()`:

```
// ambil seluruh kolom
$tabelBarang = $DB->get('barang');

// ambil kolom nama_barang
$DB->select('nama_barang');
$tabelBarang = $DB->get('barang');

// ambil kolom nama_barang, id_barang dan harga_barang
$DB->select('nama_barang,id_barang,harga_barang');
$tabelBarang = $DB->get('barang');

// ambil seluruh kolom
$tabelBarang = $DB->get('barang');
```

Khusus untuk pemanggilan yang terakhir, akan menampilkan seluruh kolom karena setiap kali method `get()` dipanggil nilai property `$_columnName` akan di reset ulang menjadi tanda bintang `"*"`.

Agar lebih "canggih" lagi, saya ingin menerapkan konsep **method chaining** ke dalam method `select()`. Maksudnya agar kita bisa memanggil method `select()` dan `get()` dengan cara berikut:

11.DB_method_get_select_chaining/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->select('nama_barang')->get('barang');
6 $tabelBarang = $DB->select('harga_barang, nama_barang')->get('barang');
```

Di baris 5 dan 6, saya menyambung pemanggilan method `select()` dan `get()`. Untuk mendapatkan fitur seperti ini, kita hanya perlu menambah 1 baris ke dalam method `select()`:

11.DB_method_get_select_chaining/DB.php

```
1 <?php
2 class DB{
3     // ...
4     // ...
5
6     public function select($columnName){
7         $this->_columnName = $columnName;
8         return $this;
9     }
10
11 }
```

Yup, hanya perlu menambahkan baris perintah `return $this` di baris 8, maka secara otomatis kita sudah bisa bisa melakukan *method chaining*.

11.9. Class DB: Method orderBy

Method `orderBy()` di rancang agar kita bisa mengatur urutan baris tampilan tabel. Dalam MySQL, ini bisa dilakukan dengan tambahan clausa '`ORDER BY <nama_kolom> ASC`' atau '`ORDER BY <nama_kolom> DESC`'.

Teknik yang akan saya pakai sama seperti method `select()`, yakni dengan memanggil method `orderBy()` sebelum menjalankan query `get()`. Berikut contoh penggunaannya dari file `index.php`:

12.DB_method_get_orderBy/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->select('id_barang,nama_barang');
6 $DB->orderBy('id_barang','DESC');
7 $tabelBarang = $DB->get('barang');
8
9 echo "<pre>";
10 print_r($tabelBarang);
11 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 6
            [nama_barang] => Cosmos CRJ-8229 - Rice Cooker
        )

    [1] => stdClass Object
        (
            [id_barang] => 5
            [nama_barang] => Smartphone Xiaomi Pocophone F1
        )

    ...
    [5] => stdClass Object
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
        )
)
```

)

Di baris 5 saya menjalankan method `select()` untuk menentukan kolom apa saja yang akan diambil. Dalam contoh ini kolom tersebut adalah '`id_barang, nama_barang`'.

Di baris 6 terdapat pemanggilan method `orderBy('id_barang', 'DESC')`. Method ini diisi dengan 2 buah argument. Argument pertama berupa nama kolom yang menjadi patokan pengurutan, yakni `id_barang`. Argument kedua berupa metode pengurutan, apakah akan diurut secara menaik ('`ASC`') atau secara menurun ('`DESC`').

Terakhir pada baris 7 terdapat perintah untuk menjalankan method `$DB->get('barang')`.

Hasil kode program dari baris 5 – 7 akan menghasilkan query MySQL sebagai berikut:

```
SELECT id_barang, nama_barang FROM barang ORDER BY id_barang DESC
```

Dan seperti yang terlihat, array `$tabelBarang` ditampilkan dari `id_barang` 6, 5, 4 sampai 1, yakni di urutkan menurun berdasarkan kolom `id_barang`.

Bagaimana cara pembuatannya? Kurang lebih mirip seperti pembuatan method `select()`. Pertama, saya butuh sebuah *private property* `$_orderBy`:

```
private $_orderBy = "";
```

Property ini tidak memiliki nilai awal karena akan diisi dari dalam method `orderBy()`:

```
public function orderBy($columnName, $sortType = 'ASC'){
    $this->_orderBy = "ORDER BY {$columnName} {$sortType}";
    return $this;
}
```

Di sini property `$_orderBy` diisi dengan string "`ORDER BY {$columnName} {$sortType}`". Variabel `$columnName` berasal dari argument pertama, dan variabel `$sortType` berasal dari argument kedua. Khusus untuk variabel `$sortType`, memiliki nilai opsional '`ASC`' yang akan dipakai jika method `orderBy()` dipanggil tanpa menyertakan nilai argument `$sortType`.

Di baris terakhir terdapat perintah `return $this` untuk membuat efek *method chaining* seperti di method `select()` sebelumnya.

Langkah terakhir adalah memodifikasi method `get()`:

12.DB_method_get_orderBy/DB.php

```
1 public function get($tableName){
2     $query = "SELECT {$this->_columnName} FROM {$tableName} {$this->_orderBy}";
3     $this->_columnName = "*";
4     $this->_orderBy = "";
5     return $this->getQuery($query);
6 }
```

Perubahan ada di baris ke-2, yakni isi dari variabel `$query`. Terdapat tambahan variabel

`->{$this->_orderBy}` di akhir string. Kemudian di baris 4 terdapat perintah `$this->_orderBy = ""` yang dipakai untuk me-reset ulang isi property `$_orderBy`.

Dengan penambahan ini, kita sudah bisa memakai method `orderBy()` untuk mengatur urutan tampilan kolom. Karena juga mendukung *method chaining*, pemanggilan method bisa dirangkai sebagai berikut :

```
$tabelBarang = $DB->select('harga_barang, nama_barang')->orderBy('harga_barang')
->get('barang');
```

Yang artinya sama dengan pemanggilan query:

```
SELECT harga_barang, nama_barang FROM barang ORDER BY harga_barang ASC
```

Karena method `orderBy()` dipanggil tanpa menyertakan argument kedua, maka akan memakai nilai default ASC.

11.10. Class DB: Method get (condition)

Class DB yang kita rancang sudah memiliki *query builder* untuk proses pengambilan tabel, pemilihan nama kolom serta pengurutan. Sekarang kita akan tambah dengan kondisi WHERE.

Namun daripada membuat sebuah method baru, saya akan modifikasi method `get()` agar bisa menerima argument berupa sebuah kondisi WHERE. Kondisi WHERE ini akan diinput sebagai argument kedua. Selain itu karena kita menerapkan prepared statement, maka butuh argument ketiga untuk data placeholder.

Berikut contoh pemanggilan method get dengan tambahan kondisi WHERE:

13.DB_method_get_condition/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->get('barang', 'WHERE id_barang = ?',[2]);
```

Perintah di baris ke-5 akan diproses sebagai: `SELECT * FROM barang WHERE id_barang = 2`

Dan berikut modifikasi untuk method `get()` di class DB:

13.DB_method_get_condition/DB.php

```
1 <?php
2 class DB{
3
4     public function get($tableName, $condition = "", $bindValue =[]){
5         $query = "SELECT {$this->_columnName} FROM {$tableName} {$condition}
6             {$this->_orderBy}";
7         $this->_columnName = "*";
8         $this->_orderBy = "";
```

```
9     return $this->getQuery($query, $bindValue);
10    }
11 }
```

Method `get()` mendapat 2 tambahan argument, yakni `$condition` dan `$bindValue`. Variabel `$condition` akan ditambah ke dalam string `$query`, yakni setelah `{$tableName}` dan sebelum `{$this->_orderBy}`. Sedangkan variabel `$bindValue` langsung diteruskan sebagai argument kedua untuk pemanggilan method `getQuery()` di baris 9.

Perhatikan bahwa argument `$condition` dan `$bindValue` memiliki nilai default, yang berarti keduanya bersifat opsional. Jika kedua argument tidak ditulis, maka itu berarti tanpa kondisi WHERE. Dari halaman `index.php`, kita tetap bisa memanggil method `get()` dengan hanya 1 argument saja:

13.DB_method_get_condition/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->get('barang'); // ambil semua isi tabel barang
```

Lebih jauh lagi, method `get()` sudah bisa menangani pembuatan query yang cukup kompleks, seperti contoh berikut:

13.DB_method_get_condition/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->select('harga_barang, nama_barang');
6 $DB->orderBy('harga_barang', 'DESC');
7 $tabelBarang = $DB->get('barang', 'WHERE harga_barang > ?',[5000000]);
```

Perintah query di baris 5 – 7 sama dengan berikut:

```
SELECT harga_barang, nama_barang FROM barang
WHERE harga_barang > 5000000 ORDER BY harga_barang DESC
```

Sebagai latihan, silahkan anda coba panggil method `select()`, `orderBy()` dan `get()` di atas menjadi 1 baris perintah dengan teknik method chaining.

11.11. Class DB: Method `getWhere`

Pengambilan data tabel dengan kondisi WHERE cukup sering kita lakukan, oleh karena itu saya akan buat method khusus untuk keperluan ini, yakni `getWhere()`. Method `getWhere()` ini merupakan pengembangan dari method `get()` dengan kondisi WHERE yang baru saja kita buat.

Dalam method get(), kondisi WHERE harus ditulis lengkap seperti contoh berikut:

```
$tabelBarang = $DB->get('barang', 'WHERE harga_barang > ?', [5000000]);
```

Untuk method getWhere(), saya ingin query tersebut tidak lagi tulis, tapi cukup datanya saja:

```
$tabelBarang = $DB->getWhere('barang', ['harga_barang', '>', 5000000]);
```

Di sini, argument kedua berbentuk array dengan 3 komponen, yakni nama **kolom kondisi**, **operator**, dan **nilai**. Nantinya, kode program di dalam method getWhere() lah yang akan meng-generate perintah query.

Contoh lain, query `SELECT * FROM barang WHERE id_barang = 3` bisa dijalankan dengan perintah:

```
$tabelBarang = $DB->getWhere('barang', ['id_barang', '=', 3]);
```

Bagaimana rancangan method getWhere() ini? Berikut kode program yang saya gunakan:

14.DB_method_getWhere

```
1  <?php
2  class DB{
3      // ...
4      // ...
5
6      public function getWhere($tableName, $condition){
7          $queryCondition ="WHERE {$condition[0]} {$condition[1]} ?";
8          return $this->get($tableName,$queryCondition,[${condition[2]}]);
9      }
10 }
```

Method getWhere() butuh 2 buah argument yang ditampung ke dalam variabel `$tableName` dan `$condition`.

Di baris 7, variabel `$queryCondition` dipakai untuk membuat ulang cluasa WHERE menggunakan nilai dari `${condition[0]}` dan `${condition[1]}` serta sebuah tanda tanya "?". Ini diperlukan karena kita akan menjalankan kondisi menggunakan PDO prepared statement.

Sebagai contoh, ketika dipanggil perintah berikut:

```
$tabelBarang = $DB->getWhere('barang', ['harga_barang', '>', 5000000]);
```

String 'barang' akan ditampung ke dalam variabel `$tableName`, sedangkan array `['harga_barang', '>', 5000000]` ditampung ke dalam `$condition`. Untuk mengakses setiap element array `$condition`, kita harus menggunakan index `$condition[0]`, `$condition[1]` dan `$condition[2]`.

Dengan demikian, perintah "WHERE `${condition[0]}` `${condition[1]}` ?" akan dikonversi menjadi: "WHERE `harga_barang > ?`".

Terakhir, di baris 8 proses query di alihkan ke method `get()` dengan mengirim 3 buah argument, yakni `$tableName`, `$queryCondition`, dan `[$condition[2]]`. Dalam contoh kita, perintah di baris 8 ini akan dikonversi sebagai berikut:

```
return $this->get('barang', "WHERE harga_barang > ?", [5000000]);
```

Sisa pemrosesan selanjutnya akan dikerjakan oleh method `get()`. Inilah salah satu keuntungan dari teknik memecah method, dimana method yang satu akan saling memanggil method lain untuk bersama-sama memecahkan sebuah masalah.

Keuntungan lain dengan memanggil method `get()` secara internal, adalah method `getWhere()` secara otomatis juga bisa memanfaatkan *method chaining* milik method `get()`, seperti contoh berikut:

14.DB_method_getWhere/index.php

```
1  <?php
2  require 'DB.php';
3  $DB = DB::getInstance();
4
5  $tabelBarang = $DB->select('nama_barang,jumlah_barang')
6      ->getWhere('barang',[ 'id_barang' , '=' , 5]);
7
8  echo "<pre>";
9  print_r($tabelBarang);
10 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [nama_barang] => Smartphone Xiaomi Pocophone F1
            [jumlah_barang] => 25
        )
)
```

Perintah di baris 5 akan diproses oleh class `DB` sebagai:

```
SELECT nama_barang, jumlah_barang FROM barang WHERE id_barang = 5
```

Dengan class `DB`, kita tidak perlu menulis manual query ini, tapi cukup mengakses method yang tersedia.

11.12. Class DB: Method `getWhereOnce`

Dalam banyak situasi, kadang kita ingin mengambil data pertama dari hasil query `SELECT`, atau kita yakin bahwa hasil query tersebut hanya akan mengembalikan 1 data saja. Dengan method

getWhere() yang sudah tersedia, ini bisa dilakukan namun dengan sedikit kendala.

Bisakah anda menebak apa yang salah dari kode berikut?

15.DB_method_getWhere_problem/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->getWhere('barang',[ 'id_barang' , '=' ,5]);
6
7 echo $tabelBarang->nama_barang;
```

Hasil kode program:

Warning: Attempt to read property "nama_barang" on array in index.php on line 7

Error ini terjadi karena \$tabelBarang->nama_barang tidak bisa diakses. Cara pengaksesan yang benar adalah \$tabelBarang[0]->nama_barang. Tambahan angka atau index [0] berhubungan dengan metode yang dipakai untuk mengambil data.

Secara internal, method getWhere() menggunakan method getQuery() untuk mengambil data, yakni menggunakan method fetchAll(PDO::FETCH_OBJ) dari PDO. Hasilnya, variabel \$tabelBarang berisi **array dari object**, bukan hanya object saja.

Dalam contoh di atas variabel \$tabelBarang hanya berisi 1 baris data, tetapi saja kita harus tulis index [0] seperti \$tabelBarang[0]->nama_barang.

Saya memutuskan membuat method khusus untuk keperluan ini, yakni getWhereOnce(). Method getWhereOnce() akan mengembalikan data langsung berbentuk object, bukan lagi array dari object. Dengan demikian, kita tidak harus menulis index [0] setiap kali ingin menampilkan hasil tabel.

Dengan method getWhereOnce(), pemanggilan di atas bisa dijalankan sebagai berikut:

16.DB_method_getWhereOnce/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->getWhereOnce('barang',[ 'id_barang' , '=' ,5]);
6
7 echo $tabelBarang->nama_barang;
```

Hasil kode program:

Smartphone Xiaomi Pocophone F1

Dan berikut kode program untuk membuat method getWhereOnce() dalam class DB:

16.DB_method_getWhereOnce/DB.php

```

1 <?php
2 class DB{
3     // ...
4     // ...
5
6     public function getWhereOnce($tableName, $condition){
7         $result = $this->getWhere($tableName,$condition);
8         if (!empty($result)) {
9             return $result[0];
10        } else {
11            return false;
12        }
13    }
14
15 }

```

Prinsip kerja dari method `getWhereOnce()` sebenarnya hanya dengan memanggil method `getWhere()[0]`. Namun ada sedikit tambahan pemeriksaan kondisi.

Di baris 7, hasil pemanggilan method `getWhere()` saya simpan ke dalam variabel `$result`. Kemudian isinya diperiksa menggunakan kondisi `if(!empty($result))`. Jika variabel `$result` berisi suatu nilai, maka kembalikan hasil dari `$result[0]`. Namun jika isi variabel `$result` ternyata kosong, yang berarti data tidak ditemukan, maka kembalikan nilai **false**.

Dengan penulisan seperti ini, method `getWhereOnce()` hanya mengembalikan 1 baris pertama dari hasil query, meskipun query tersebut bisa mengambil lebih dari 1 baris data. Sebagai contoh, jika yang dijalankan adalah perintah berikut:

```
$tabelBarang = $DB->getWhereOnce('barang',[ 'harga_barang' , '>', 5000000]);
```

Maka variabel `$tabelBarang` hanya akan berisi baris pertama dari barang yang harganya lebih dari 5000000. Meskipun sebenarnya ada banyak barang dengan harga lebih dari 5000000 di dalam tabel.

Method `getWhereOnce()` ini cocok dipakai untuk query yang hanya mengembalikan 1 baris saja, seperti query dengan kondisi `WHERE id_barang = 5`. Query ini hanya akan berisi 1 baris karena kolom `id_barang` sudah di set sebagai PRIMARY KEY yang tidak bisa diisi nilai ganda.

Sebenarnya method `getWhereOnce()` bisa saja ditulis seperti ini:

```

1 public function getWhereOnce($tableName, $condition){
2     return $this->getWhere($tableName,$condition)[0];
3 }

```

Namun kode ini akan error jika method `getWhere()` tidak mengembalikan apa-apa, yakni kondisi dimana data tidak ditemukan di dalam tabel. Karena itulah kita perlu membuat sebuah

pemeriksaan if(!empty(\$result)) terlebih dahulu.

11.13. Class DB: Method getLike

Method `getLike()` saya rancang untuk menjalankan query `SELECT` dengan kondisi `LIKE`. Dalam bahasa SQL, perintah `LIKE` dipakai dalam proses pencarian.

Konsep pembuatan method `getLike()` ini mirip seperti method `getWhere()`, dimana saya akan rancang string tambahan, lalu memanggil method `get()` secara internal.

Berikut contoh pemanggilan method `getLike()` dari halaman `index.php`:

17.DB_method_getLike/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->getLike('barang', 'nama_barang', '%kulkas%');
6
7 echo "<pre>";
8 print_r($tabelBarang);
9 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 2
            [nama_barang] => Kulkas LG GC-A432HLHU
            [jumlah_barang] => 10
            [harga_barang] => 7600000
            [tanggal_update] => 2019-03-12 07:59:50
        )
)
```

Method `getLike()` perlu 3 buah argument, yakni nama tabel, kolom yang akan dicari, dan pola pencarian. Pemanggilan method `getLike('barang', 'nama_barang', '%kulkas%')` di baris 5 akan meng-generate query berikut:

```
SELECT * FROM barang WHERE nama_barang LIKE '%kulkas%'
```

Query di atas akan mencari semua data barang yang memiliki kata "kulkas" di kolom `nama_barang`.

Berikut kode yang saya pakai untuk membuat method `getLike()` dalam class DB:

17.DB_method_getLike/DB.php

```
1 <?php
2 class DB{
3     // ...
4     // ...
5
6     public function getLike($tableName, $columnLike, $search){
7         $queryLike = "WHERE {$columnLike} LIKE ?";
8         return $this->get($tableName,$queryLike,[{$search}]);
9     }
10 }
```

Ketika dijalankan perintah `$DB->getLike('barang', 'nama_barang', '%kulkas%')`, maka di dalam method `getLike()`, variabel `$queryLike` akan berisi string `"WHERE nama_barang LIKE ?"`. Tanda tanya "?" diperlukan karena di dalam class `DB` kita memproses semua query sebagai *prepared statement*. Nilai pengganti untuk tanda tanya ini (nilai placeholder), yakni pola pencarian ditampung ke dalam argument `$search`.

Selanjutnya, proses bisa kita alihkan ke method `get()` dengan mengirim argument berupa `$tableName`, `$queryLike`, dan `[{$search}]`. Khusus untuk argument `$search`, sengaja menggunakan tanda kurung siku "[..]" agar dikonversi menjadi tipe data array. Ini diperlukan karena argument ini akan menjadi inputan untuk method `execute()` di dalam method `get()`. Dalam PDO prepared statement, method `execute()` harus menerima inputan dalam bentuk array, meskipun isinya hanya 1 data saja.

Dengan memanggil method `get()` secara internal, maka method `getLike()` otomatis juga mendukung berbagai method tambahan seperti `select()`, `orderBy()` serta teknik *method chaining* sebagaimana yang kita rancang untuk method `get()`. Berikut contohnya:

17.DB_method_getLike/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $tabelBarang = $DB->select('nama_barang,id_barang')
6             ->getLike('barang','nama_barang','%k%');
7
8 echo "<pre>";
9 print_r($tabelBarang);
10 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [nama_barang] => TV Samsung 43NU7090 4K
            [id_barang] => 1
```

```
)  
[1] => stdClass Object  
(  
    [nama_barang] => Kulkas LG GC-A432HLHU  
    [id_barang] => 2  
)  
  
[2] => stdClass Object  
(  
    [nama_barang] => Cosmos CRJ-8229 - Rice Cooker  
    [id_barang] => 6  
)  
)
```

Pemanggilan method `getLike()` di baris 5-6 akan menjalankan query berikut:

```
SELECT nama_barang, id_barang FROM barang WHERE nama_barang LIKE '%k%'
```

Query ini akan mencari seluruh barang dimana kolom `nama_barang` mengandung huruf 'k', ini cocok dengan data TV Samsung 43NU7090 4K, Kulkas LG GC-A432HLHU dan Cosmos CRJ-8229 - Rice Cooker.

11.14. Class DB: Method check

Method `check()` saya rancang untuk memeriksa apakah sebuah data ada di dalam tabel. Ini sering dipakai untuk proses validasi form, misalnya memeriksa apakah sebuah username sudah ada di dalam tabel atau belum.

Teknik yang akan saya pakai adalah dengan menjalankan method `rowCount()` bawaan PDO. Method `rowCount()` akan mengembalikan jumlah baris dari hasil query `SELECT`. Jika method ini menghasilkan nilai 0 maka artinya tidak ada data. Jika hasilnya 1 atau lebih, artinya data tersebut ada di dalam tabel.

Berikut contoh penggunaan method `check()` dari halaman `index.php`:

18.DB_method_check/index.php

```
1 <?php  
2 require 'DB.php';  
3 $DB = DB::getInstance();  
4  
5 $result = $DB->check('barang', 'id_barang', '4');  
6 echo $result;
```

Hasil kode program:

1

Angka 1 di sini berarti dalam tabel barang hanya terdapat 1 baris yang memiliki `id_barang` = 4.

Berikut contoh lainnya:

18.DB_method_check/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->check('barang', 'id_barang', '10');
6 echo $result;
```

Hasil kode program:

0

Angka 0 berarti dalam tabel barang tidak ditemukan baris yang memiliki `id_barang = 10`.

Berikut kode program yang saya pakai untuk membuat method `check()`:

18.DB_method_check/DB.php

```
1 <?php
2 class DB{
3     // ...
4     // ...
5
6     public function check($tableName, $columnName, $dataValues){
7         $query = "SELECT {$columnName} FROM {$tableName} WHERE {$columnName} = ? ";
8         return $this->runQuery($query,[$dataValues])->rowCount();
9     }
10 }
```

Method `check()` butuh 3 buah argument, yakni nama tabel, nama kolom dan nilai. Ketiganya ditampung ke dalam variabel `$tableName`, `$columnName`, dan `$dataValues`. Di baris 7, saya menulis seluruh query yang dibutuhkan dari argument ini.

Sebagai contoh, jika yang dipanggil adalah:

```
$result = $DB->check('barang', 'id_barang', '10');
```

Maka variabel `$query` akan berisi string:

```
$query = "SELECT id_barang FROM barang WHERE id_barang = ?"
```

Tanda tanya "?" di akhir string diperlukan karena kita menggunakan prepared statement. Kemudian variabel `$query` ini akan dikirim ke method `runQuery()`, beserta variabel `$dataValues`.

Hasil pemanggilan method `runQuery()` berbentuk **PDO Statement object** yang langsung disambung dengan method `rowCount()` di akhir baris 8.

Method `check()` ini bisa langsung dipakai untuk pengecekan kondisi, seperti contoh berikut:

18.DB_method_check/index.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 if ($DB->check('barang','id_barang','4')) {
6     echo "ID barang 4 tersedia";
7 }
```

Hasil kode program:

```
ID barang 4 tersedia
```

Di baris 5 saya langsung menulis `if ($DB->check('barang','id_barang','4'))`, ini bisa dilakukan karena PHP akan mengkonversi angka menjadi tipe data boolean.

Method `check()` akan mengembalikan nilai 0 jika data tidak ditemukan, yang akan dikonversi menjadi boolean **false**. Namun jika method `check()` mengembalikan angka selain 0, itu akan dikonversi menjadi **true**.

Method `check()` ini menjadi method terakhir untuk proses `SELECT`, berikutnya kita akan masuk ke method untuk memproses query `INSERT`, `UPDATE` dan `DELETE`. Ketiga query ini menjadi tantangan tersendiri karena harus kita proses sebagai prepared statement.

11.15. Class DB: Method insert

Sesuai dengan namanya, method `insert()` dipakai untuk memproses query `INSERT`. Method `insert()` yang akan kita rancang ini menjadi salah satu method yang paling rumit di dalam class `DB`. Ini karena jumlah data yang diinput bisa berbeda-beda, serta kita juga tetap akan menggunakan prepared statement.

Berikut contoh penggunaan method `insert()`:

19.DB_method_insert/insert_method.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->insert('barang', [
6     'nama_barang' => 'Philips Blender HR 2157',
7     'jumlah_barang' => 11,
8     'harga_barang' => 629000
9 ]);
```

Method `insert()` saya rancang dengan 2 buah argument, yang pertama berupa nama tabel yang akan ditambahkan, dan kedua berupa *associative array* yang berisi pasangan nama kolom

dan nilai untuk kolom tersebut.

Tugas kita adalah, bagaimana cara mengonversi pemanggilan method `insert()` di baris 5 – 9 menjadi query berikut:

```
INSERT into barang (nama_barang, jumlah_barang, harga_barang) VALUES ('Philips  
Blender HR 2157', 11, 629000)
```

Untuk keperluan ini kita perlu "membongkar" isi *associative array*. Dan akan sedikit rumit karena kita butuh query versi *prepared statement*. Maksudnya, query yang diperlukan oleh method `insert()` adalah sebagai berikut:

```
INSERT into barang (nama_barang, jumlah_barang, harga_barang) VALUES (?, ?, ?)
```

Dimana nilai dari setiap *placeholder* akan dikirim terpisah, yakni dalam bentuk array ['Philips
Blender HR 2157', 11, 629000]

Proses pembuatan method `insert()` akan saya bahas secara bertahap. Nantinya kita butuh berbagai function bawaan PHP seperti `array_keys()`, `array_values()`, `implode()`, serta `str_repeat()`.

Kita akan mulai dengan memecah *associative array*. Dalam PHP, *associative array* adalah array yang key atau indexnya berupa string seperti contoh berikut:

```
1 $foo = [  
2     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',  
3     'jumlah_barang' => 4,  
4     'harga_barang' => 299000  
5 ];
```

Untungnya PHP menyediakan function bawaan untuk mengambil key dan value secara terpisah, yakni menggunakan function `array_keys()` dan `array_values()`. Sesuai namanya, function `array_keys()` dipakai untuk mengambil nilai **key** dari sebuah array, sedangkan function `array_values()` dipakai untuk mengambil nilai atau **value** dari sebuah array.

Berikut contoh penggunaannya:

19.DB_method_insert/insert_function_1.php

```
1 <?php  
2 $foo = [  
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',  
4     'jumlah_barang' => 4,  
5     'harga_barang' => 299000  
6 ];  
7  
8 $dataKeys = array_keys($foo);  
9 $dataValues = array_values($foo);  
10  
11 print_r($dataKeys);
```

```
12 echo "<br>";  
13 print_r($dataValues);
```

Hasil kode program:

```
Array ( [0] => nama_barang [1] => jumlah_barang [2] => harga_barang )  
Array ( [0] => Cosmos CRJ-8229 - Rice Cooker [1] => 4 [2] => 299000 )
```

Terlihat kita sudah bisa memisahkan antara key dan value dari associative array \$foo. Seluruh key dari array \$foo tersimpan ke dalam variabel \$dataKeys, dan untuk value dari variabel \$foo tersimpan ke dalam variabel \$dataValues.

Agar pembahasan kita lebih sederhana, saya akan buat sebuah fungsi insert(). Di dalam fungsi insert() ini kita akan coba rancang kode program untuk meng-generate query INSERT, dan jika sudah berhasil, baru nanti dipindahkan ke dalam class DB:

19.DB_method_insert/insert_function_2.php

```
1 <?php  
2 insert('barang',[  
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',  
4     'jumlah_barang' => 4,  
5     'harga_barang' => 299000  
6 ]);  
7  
8 function insert($tableName, $data){  
9     $dataKeys = array_keys($data);  
10    $dataValues = array_values($data);  
11 }
```

Di awal kode program saya memanggil fungsi insert() dengan argument yang sama persis seperti method insert() sebelumnya. Argument pertama berupa nama tabel dan argument kedua berbentuk associative array.

Kode program untuk function insert() itu sendiri ada di baris 8 – 10. Argument \$tableName akan menampung nama tabel, dan argument \$data dipakai untuk menampung associative array.

Fungsi insert() ini sangat pas sebagai simulasi method insert() untuk class DB nanti. Tujuan akhir kita adalah bagaimana meng-generate sebuah query INSERT dalam bentuk prepared statement.

Jika function insert() dipanggil sebagai berikut:

```
insert('barang',[  
    'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',  
    'jumlah_barang' => 4,  
    'harga_barang' => 299000  
]);
```

Saya ingin nanti bisa men-generate query:

```
'INSERT INTO barang (nama_barang, jumlah_barang, harga_barang) VALUES (?, ?, ?)' ;
```

Saat ini kita sudah berhasil mengumpulkan semua **key** dari array `$data` yang disimpan ke dalam variabel `$dataKeys`. Langkah berikutnya adalah memproses array `$dataKeys` ini menjadi sebuah string.

Sekarang variabel `$dataKeys` berisi array dengan 3 element:

```
['nama_barang', 'jumlah_barang', 'harga_barang']
```

Kita akan rancang kode program agar array ini bisa menjadi string: '(`nama_barang`, `jumlah_barang`, `harga_barang`)'. Untuk proses konversi dari array menjadi string, PHP sudah menyediakan fungsi bawaan yakni `implode()`. Berikut hasilnya:

19.DB_method_insert/insert_function_3.php

```
1 <?php
2 insert('barang',[ 
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',
4     'jumlah_barang' => 4,
5     'harga_barang' => 299000
6 ]);
7
8 function insert($tableName, $data){
9     $dataKeys = array_keys($data);
10    $dataValues = array_values($data);
11
12    $result= implode(', ', $dataKeys);
13
14    echo($result);
15 }
```

Hasil kode program:

```
nama_barang, jumlah_barang, harga_barang
```

Di baris 12, saya menjalankan fungsi `implode(' ', '$dataKeys')`. Hasilnya, seluruh array akan disatukan menjadi string dengan tanda koma sebagai pemisah. Sekarang tinggal menambahkan string tanda kurung di awal dan akhir:

19.DB_method_insert/insert_function_4.php

```
1 <?php
2 insert('barang',[ 
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',
4     'jumlah_barang' => 4,
5     'harga_barang' => 299000
6 ]);
7
```

```
8 function insert($tableName, $data){  
9     $dataKeys = array_keys($data);  
10    $dataValues = array_values($data);  
11  
12    $result= '('.implode(', ', $dataKeys).')';  
13    echo($result);  
14 }
```

Hasil kode program:

```
(nama_barang, jumlah_barang, harga_barang)
```

Di baris 12 saya menambah awalan "(" dan akhiran ")", agar string berada di dalam tanda kurung. Dengan demikian, string ini kita bisa rangkai sebagai berikut:

19.DB_method_insert/insert_function_5.php

```
1 <?php  
2 insert('barang',[  
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',  
4     'jumlah_barang' => 4,  
5     'harga_barang' => 299000  
6 ]);  
7  
8 function insert($tableName, $data){  
9     $dataKeys = array_keys($data);  
10    $dataValues = array_values($data);  
11  
12    echo "INSERT INTO {$tableName} (".$implode(', ', $dataKeys).")";  
13 }
```

Hasil kode program:

```
INSERT INTO barang (nama_barang, jumlah_barang, harga_barang)
```

Sip, bagian awal query sudah selesai. Selanjutnya, bagaimana cara membuat tanda tanya sejumlah data yang akan diinput?

Sebagaimana yang sudah kita pelajari, prepared statement menggunakan placeholder berupa tanda tanya "?". Jumlah tanda tanya ini harus sesuai dengan jumlah data yang diinput. Kita bisa mengetahui jumlah data inputan dari total element yang ada di dalam array \$data.

Fungsi count() bisa dipakai untuk keperluan ini. Dalam contoh kita, count(\$data) akan menghasilkan angka 3. Artinya untuk tempat placeholder perlu 3 buah tanda tanya.

Untuk membuat 3 buah tanda tanya secara dinamis, bisa memakai perulangan for atau bisa juga menggunakan fungsi str_repeat() bawaan PHP. Berikut hasil penggunaannya:

19.DB_method_insert/insert_function_6.php

```
1 <?php  
2 insert('barang',[
```

```
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',
4     'jumlah_barang' => 4,
5     'harga_barang' => 299000
6 ];
7
8 function insert($tableName, $data){
9     $dataKeys = array_keys($data);
10    $dataValues = array_values($data);
11
12    echo str_repeat('?', ', count($data));
13 }
```

Hasil kode program:

```
? , ? , ? ,
```

Fungsi `str_repeat()` akan mengulang karakter yang diinput sebagai argument pertama sebanyak nilai di argument kedua. Perintah di baris 12 akan diproses sebagai `echo str_repeat('?', ', 3)`. Hasilnya, string `'?,'` akan di ulang sebanyak 3 kali.

Namun ada sedikit masalah. Kita harus menghapus tanda koma terakhir agar hasilnya menjadi:

```
? , ? , ?
```

Untuk ini kita akan memakai sedikit "trik". Daripada mengulang tanda tanya dan koma `'?,'` sejumlah data yang ada, saya akan kurangi 1 dan menulis manual tanya tanya terakhir.

Perintahnya menjadi sebagai berikut:

```
echo str_repeat('?', ', count($data)-1) . '?' ;
```

Yang akan diproses adalah: `str_repeat('?', ', 2)` lalu ditambah 1 karakter `'?'` di bagian akhir.

Placeholder ini juga perlu tambahan tanda kurung di awal dan akhir string:

19.DB_method_insert/insert_function_7.php

```
1 <?php
2 insert('barang',[
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',
4     'jumlah_barang' => 4,
5     'harga_barang' => 299000
6 ]);
7
8 function insert($tableName, $data){
9     $dataKeys = array_keys($data);
10    $dataValues = array_values($data);
11
12    echo '(' . str_repeat('?', ', count($data)-1) . '?)';
13 }
```

Hasil kode program:

(?, ?, ?)

Done. Kita sudah berhasil membuat karakter placeholder secara dinamis. Berikutnya tinggal merangkai semua string menjadi sebuah perintah query **INSERT**:

19.DB_method_insert/insert_function_8.php

```
1 <?php
2 insert('barang',[  
3     'nama_barang' => 'Cosmos CRJ-8229 - Rice Cooker',  
4     'jumlah_barang' => 4,  
5     'harga_barang' => 299000  
6 ]);  
7  
8 function insert($tableName, $data){  
9     $dataKeys = array_keys($data);  
10    $dataValues = array_values($data);  
11    $placeholder = ('.str_repeat('?',', count($data)-1) . '?');  
12  
13    echo "INSERT INTO {$tableName} (".$implode(', ', $dataKeys).")  
14        VALUES {$placeholder}";  
15 }
```

Hasil kode program:

```
INSERT INTO barang (nama_barang, jumlah_barang, harga_barang) VALUES (?, ?, ?)
```

Inilah hasil akhir dari perancangan fungsi `insert()`, dimana kita membuat query **INSERT prepared statement** yang dihasilkan secara dinamis. Untuk uji coba, mari tes dengan data yang berbeda:

19.DB_method_insert/insert_function_9.php

```
1 <?php
2 insert('user',[  
3     'username' => 'Andi',  
4     'email' => 'andi@gmail.com',  
5     'umur' => 15,  
6     'sekolah' => 'SMA N 7 lumut ijo',  
7     'alamat' => 'Jl. Perintis no 9'  
8 ]);  
9  
10 function insert($tableName, $data){  
11     $dataKeys = array_keys($data);  
12     $dataValues = array_values($data);  
13     $placeholder = ('.str_repeat('?',', count($data)-1) . '?');  
14  
15     echo "INSERT INTO {$tableName} (".$implode(', ', $dataKeys).")  
16         VALUES {$placeholder}";  
17     echo "<br>";  
18     print_r($dataValues);  
19 }
```

Hasil kode program:

```
INSERT INTO user (username, email, umur, sekolah, alamat) VALUES (?, ?, ?, ?, ?)

Array ( [0] => Andi
       [1] => andi@gmail.com
       [2] => 15 [3] => SMA N 7 lumut ijo
       [4] => Jl. Perintis no 9 )
```

Argument kedua pada saat pemanggilan fungsi insert di baris 2 – 8 berisi 5 buah element.

Hasilnya, query `INSERT` juga akan memiliki lima buah tanda tanya untuk *placeholder*. Ini sesuai dengan apa yang kita inginkan.

Dan, tiba saatnya membuat method `insert()` di dalam class `DB`. Berikut kode yang diperlukan:

19.DB_method_insert/DB.php

```
1 <?php
2 class DB{
3     // ...
4     // ...
5
6     public function insert($tableName, $data){
7         $dataKeys = array_keys($data);
8         $dataValues = array_values($data);
9         $placeholder = (''.str_repeat('?', ', count($data)-1) . '?');
10
11        $query = "INSERT INTO {$tableName} (".$implode(', ', $dataKeys).")
12            VALUES {$placeholder}";
13        $this->runQuery($query,$dataValues);
14    }
15 }
```

Isi method `insert()` ini sama seperti yang kita rancang sebelumnya. Sebagai tambahan, terdapat pemanggilan method `$this->runQuery($query,$dataValues)` di baris akhir. Method `runQuery()` lah yang akan selanjutnya memproses query `INSERT`.

Mari kita jalankan dengan proses insert yang sebenarnya:

19.DB_method_insert/insert_method.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->insert('barang', [
6     'nama_barang' => 'Philips Blender HR 2157',
7     'jumlah_barang' => 11,
8     'harga_barang' => 629000
9 ]);
10
11 // tampilkan semua tabel barang
12 $tabelBarang = $DB->get('barang');
```

```
13
14 echo "<pre>";
15 print_r($tabelBarang);
16 echo "</pre>";
```

Hasil kode program:

```
Array
(
    ...
    ...
    [6] => stdClass Object
        (
            [id_barang] => 7
            [nama_barang] => Philips Blender HR 2157
            [jumlah_barang] => 11
            [harga_barang] => 629000
            [tanggal_update] => 2019-03-14 14:23:56
        )
)
```

Setelah pemanggilan method `insert()` di baris 5, saya juga menjalankan method `get('barang')` di baris 12 untuk menampilkan isi dari tabel barang. Seperti yang terlihat, barang 'Philips Blender HR 2157' sukses diinput ke dalam tabel barang.

11.16. Class DB: Method count

Jika anda perhatikan, method `insert()` yang baru saja kita buat tidak mengembalikan nilai apapun. Ini bisa diatasi dengan menulis perintah `return true` di akhir method `insert()`. Ini tidak wajib ditulis karena tidak berpengaruh apa-apa ke dalam method `insert()`.

Jika pun ternyata query yang ditulis salah atau tidak sesuai, block kode **try-catch** di method `runQuery()` akan langsung menghentikan proses yang ada.

Selain itu kadang kita butuh kepastian mengenai jumlah baris yang baru saja di input (*affected rows*). Untuk hal ini, saya akan merancang method `count()`. Method `count()` pada dasarnya mirip seperti method `check()`, dimana secara internal akan mengakses method `rowCount()` bawaan PDO.

Berikut contoh pemanggilan method `count()` dari file `index.php`:

```
20.DB_method_insert_count/index.php

1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->insert('barang',[  
    'nama_barang' => 'Mouse Gaming Razer Basilisk',  
]
```

```
7     'jumlah_barang' => 25,
8     'harga_barang' => 1250000
9 ];
10
11 if($result) {
12     echo "Terdapat ".$DB->count()." data yang ditambah";
13     // Terdapat 1 data yang ditambah
14 }
```

Di baris 5, saya menyimpan hasil pemanggilan method `insert()` ke dalam variabel `$result`. Jika query berjalan sebagaimana mestinya, variabel `$result` akan berisi nilai boolean **true**.

Nilai dari `$result` kemudian di periksa dalam sebuah kondisi `if` di baris 11. Jika isinya boolean **true** (query `INSERT` berhasil dijalankan), maka perintah `echo` di baris 12 akan di proses. Di sini terdapat pemanggilan method `$DB->count()` yang akan berisi jumlah baris yang diinput oleh query `INSERT` sebelumnya.

Untuk membuat sistem seperti, kita perlu modifikasi class `DB` dengan beberapa penambahan perintah. Pertama, saya akan buat sebuah private property `$_count` di awal class `DB`:

```
private $_count = 0;
```

Property ini dipakai untuk menampung hasil dari pemanggilan method `rowCount()` bawaan PDO.

Kemudian, saya akan modif method `insert()` sebagai berikut:

```
1 public function insert($tableName, $data){
2     $dataKeys = array_keys($data);
3     $dataValues = array_values($data);
4     $placeholder = (''.str_repeat('?', ', count($data)-1) . '?');
5
6     $query = "INSERT INTO {$tableName} (".$implode(', ', $dataKeys).")"
7             VALUES {$placeholder}";
8     $this->_count = $this->runQuery($query, $dataValues)->rowCount();
9     return true;
10 }
```

Perubahannya ada di baris 8 dan 9.

Di baris 8, pemanggilan method `runQuery()` saya sambung dengan method `rowCount()`, kemudian hasilnya disimpan ke dalam property `$this->_count`. Dengan demikian, setiap kali method `insert()` dijalankan, property `$this->_count` juga akan berisi sebuah nilai, yakni jumlah kolom yang terdampak, atau *affected rows*.

Di baris 9 terdapat perintah `return true` agar method `insert()` mengembalikan nilai **true** jika sukses dijalankan.

Agar isi property `$_count` ini bisa diakses, kita perlu sebuah method getter untuk mengambil nilainya:

```
1 public function count(){
2     return $this->_count;
3 }
```

Sekarang, setiap kali method `insert()` dijalankan, jumlah baris yang di insert bisa diakses dari method `count()`. Nantinya, method `count()` ini juga bisa dipakai untuk mencari info jumlah baris hasil query lain seperti `UPDATE` dan `DELETE`.

11.17. Class DB: Method update

Membuat method `update()` punya tantangan tersendiri yang lebih rumit daripada method `insert()`. Untuk query `INSERT`, kita hanya perlu informasi seputar nama tabel dan data yang akan di input. Sedangkan query `UPDATE` butuh 3 hal, yakni nama tabel, data yang akan diubah, serta kondisi yang dipakai untuk mencari baris yang akan di update.

Namun karena sebelumnya sudah selesai membuat method `insert()`, pembuatan method `update()` akan terasa sedikit mudah karena menggunakan teknik yang mirip.

Langsung saja kita lihat contoh pemanggilan method `update()` ini:

21.DB_method_update/update_method.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->update('barang',
6                 ['nama_barang' => 'Smartphone iPhone XR',
7                  'harga_barang' => 17999000],
8                 ['id_barang', '=', 5]);
9
```

Di baris 5 terdapat pemanggilan method `$DB->update()`. Method `update()` butuh 3 buah argument. Argument pertama, yang dalam contoh ini berupa string '`barang`', merujuk ke nama tabel yang akan di update, yakni tabel `barang`.

Argument kedua berupa *associative array* yang berisi pasangan nama kolom dan nilai baru. Dalam contoh ini saya ingin mengubah nilai untuk kolom '`nama_barang`' menjadi '`Smartphone iPhone XR`' dan kolom '`harga_barang`' menjadi `17999000`.

Argument ketiga berbentuk array dengan 3 buah element. Ini dipakai untuk menentukan kondisi `WHERE` dari data yang akan di update. Dalam contoh ini, array `['id_barang', '=', 5]` sama artinya dengan kondisi `WHERE id_barang = 5`.

Secara keseluruhan, pemanggilan method `$DB->update()` di baris 5 akan menjalankan query berikut:

```
UPDATE barang SET nama_barang = 'Smartphone iPhone XR', harga_barang = 17999000
WHERE id_barang = 5
```

Namun tugas kita akan lebih berat, karena harus membuat versi *prepared statement* sebagai berikut:

```
UPDATE barang SET nama_barang = ?, harga_barang = ? WHERE id_barang = ?
```

Nantinya data *placeholder* akan di simpan dalam array tersendiri dengan nilai:

```
['Smartphone iPhone XR', 17999000, 5]
```

Baik, mari kita mulai proses pembuatan method `update()` untuk class DB.

Sama seperti proses pembuatan method `insert()`, kita akan pakai fungsi `update()` untuk mempermudah proses perancangan query. Setelah query berhasil di-generate, baru kemudian pindahkan ke dalam class DB:

21.DB_method_update/update_function_1.php

```
1 <?php
2 update('barang',
3     ['nama_barang' => 'Smartphone iPhone XR',
4      'harga_barang' => 17999000],
5     ['id_barang', '=', 5]);
6
7 function update($tableName, $data, $condition){
8     // ...
9 }
```

Masalah pertama yang akan kita pecahkan adalah bagaimana memproses pemanggilan fungsi `update()` di baris 2 agar bisa menghasilkan string berikut:

```
UPDATE nama_tabel SET nama_kolom_1 = ?, nama_kolom_2 = ?
```

Untuk nama tabel, sudah bisa langsung diakses karena tersimpan di dalam argument pertama, yakni variabel `$tableName`. Dengan demikian awal string bisa dibuat sebagai berikut:

```
$query = "UPDATE {$tableName} SET ";
```

Kemudian untuk membuat pasangan nama kolom dan tanya tanya '?' placeholder, saya susun dengan sebuah perulangan `foreach`:

```
foreach ($data as $key => $val){
    $query .= "$key = ?, ";
}
```

Kita perlu sebuah perulangan karena jumlah kolom yang akan di update tidak bisa ditentukan, bisa 2, 3 atau 10 sesuai dengan array yang ditulis dalam argument kedua.

Perulangan di atas akan dijalankan sejumlah element yang ada di dalam array `$data`. Dalam setiap perulangan, variabel `$query` akan disambung dengan nilai `"$key = ?, "`. Variabel `$key` ini merujuk ke nama key dari setiap element `$data`.

Berikut kode program fungsi update() kita sejauh ini:

21.DB_method_update/update_function_1.php

```
1 <?php
2 update('barang',
3     [ 'nama_barang' => 'Smartphone iPhone XR',
4     'harga_barang' => 17999000],
5     [ 'id_barang', '=', 5]);
6
7 function update($tableName, $data, $condition){
8     $query = "UPDATE {$tableName} SET ";
9     foreach ($data as $key => $val){
10         $query .= "{$key} = ?, " ;
11     }
12     echo $query;
13 }
```

Hasil kode program:

UPDATE barang SET nama_barang = ?, harga_barang = ?,

Sepintas query sudah sesuai. Namun ada masalah dengan tanda koma dan sebuah spasi di akhir string. Dua karakter ini berasal dari proses perulangan foreach. Untuk menghapusnya, saya akan memakai fungsi substr() bawaan PHP:

21.DB_method_update/update_function_2.php

```
1 <?php
2 update('barang',
3     [ 'nama_barang' => 'Smartphone iPhone XR',
4     'harga_barang' => 17999000],
5     [ 'id_barang', '=', 5]);
6
7 function update($tableName, $data, $condition){
8     $query = "UPDATE {$tableName} SET ";
9     foreach ($data as $key => $val){
10         $query .= "{$key} = ?, " ;
11     }
12     $query = substr($query, 0, -2);
13     echo $query;
14 }
```

Hasil kode program:

UPDATE barang SET nama_barang = ?, harga_barang = ?

Di baris 12, fungsi substr(\$query, 0, -2) akan menghapus 2 karakter terakhir yang terdapat di dalam string \$query.

Langkah berikutnya adalah menyambung string \$query dengan kondisi WHERE, yakni mencari baris mana yang akan di update. Berikut kode program yang saya pakai:

21.DB_method_update/update_function_3.php

```
1 <?php
2 update('barang',
3     ['nama_barang' => 'Smartphone iPhone XR',
4      'harga_barang' => 17999000],
5     ['id_barang', '=', 5]);
6
7 function update($tableName, $data, $condition){
8     $query = "UPDATE {$tableName} SET ";
9     foreach ($data as $key => $val){
10         $query .= "{$key} = ?, ";
11     }
12     $query = substr($query, 0, -2);
13     $query .= " WHERE {$condition[0]} {$condition[1]} ?";
14     echo $query;
15 }
```

Hasil kode program:

```
UPDATE barang SET nama_barang = ?, harga_barang = ? WHERE id_barang = ?
```

Proses penambahan kondisi WHERE cukup sederhana karena nama kolom dan operator sudah tersedia di element ke-1 dan ke-2 argument \$condition.

Sampai di sini proses pembuatan query sudah selesai, kita akan masuk ke perancangan data sebagai pengganti *placeholder*.

Data untuk nilai *placeholder* berasal dari dua buah array, yakni array \$data dan \$condition. Contoh berikut memperlihatkan isi dari kedua argument ini:

21.DB_method_update/update_function_4.php

```
1 <?php
2 update('barang',
3     ['nama_barang' => 'Smartphone iPhone XR',
4      'harga_barang' => 17999000],
5     ['id_barang', '=', 5]);
6
7 function update($tableName, $data, $condition){
8     print_r($data);
9     echo "<br>";
10    print_r($condition);
11 }
```

Hasil kode program:

```
Array ( [nama_barang] => Smartphone iPhone XR [harga_barang] => 17999000 )
Array ( [0] => id_barang [1] => = [2] => 5 )
```

Hasil akhir yang kita perlukan adalah sebuah array yang berisi nilai gabungan dari kedua argument, yakni dalam bentuk:

[Smartphone iPhone XR, 17999000, 5]

Isinya berupa seluruh *value* dari array `$data`, serta element ke-3 dari array `$condition`. Ini akan berpasangan dengan tanda tanya *placeholder* dari query prepared statement yang sudah kita siapkan sebelumnya.

Untuk mengambil nilai atau *value* dari associative array `$data` caranya cukup mudah, yakni menggunakan fungsi `array_values()` sebagaimana yang kita pakai pada saat pembuatan method `insert()`:

```
$dataValues = array_values($data);
```

Sekarang variabel `$dataValues` sudah berisi nilai atau *value* dari associative array `$data`. Kita akan tambah 1 nilai lagi, yakni element ke-3 dari array `$condition`.

Untuk keperluan ini saya akan memakai fungsi `array_push()` bawaan PHP. Fungsi `array_push()` berguna untuk menambah 1 nilai baru ke dalam sebuah array. Nilai baru ini akan berada di posisi paling akhir:

```
array_push($dataValues,$condition[2]);
```

Perintah ini artinya, tambah nilai yang tersimpan di `$condition[2]` ke posisi terakhir array `$dataValues`.

Berikut kode program lengkap proses pengambilan value array:

21.DB_method_update/update_function_5.php

```
1 <?php
2 update('barang',
3     ['nama_barang' => 'Smartphone iPhone XR',
4      'harga_barang' => 17999000],
5     ['id_barang','=',5]);
6
7 function update($tableName, $data, $condition){
8     $dataValues = array_values($data);
9     array_push($dataValues,$condition[2]);
10
11    print_r($dataValues);
12 }
```

Hasil kode program:

```
Array ( [0] => Smartphone iPhone XR [1] => 17999000 [2] => 5 )
```

Akhirnya, variabel `$dataValues` sudah berisi semua nilai yang kita butuhkan. Mari test seluruh fungsi `update()` dengan data yang berbeda:

21.DB_method_update/update_function_6.php

```
1 <?php
```

Case Study: Database Query Builder

```
2 update('user',[  
3     'username' => 'Andi',  
4     'email' => 'andi@gmail.com',  
5     'umur' => 15,  
6     'sekolah' => 'SMA N 7 lumut ijo',  
7     'alamat' => 'Jl. Perintis no 9'],  
8     ['id_user','=' ,85]);  
9  
10 function update($tableName, $data, $condition){  
11     $query = "UPDATE {$tableName} SET ";  
12     foreach ($data as $key => $val){  
13         $query .= "{$key} = ?, " ;  
14     }  
15     $query = substr($query,0,-2);  
16     $query .= " WHERE {$condition[0]} {$condition[1]} ?";  
17  
18     $dataValues = array_values($data);  
19     array_push($dataValues,$condition[2]);  
20  
21     echo $query;  
22     echo "<pre>";  
23     print_r($dataValues);  
24     echo "</pre>";  
25 }
```

Hasil kode program:

```
UPDATE user SET username = ?, email = ?, umur = ?, sekolah = ?, alamat = ? WHERE  
id_user = ?
```

```
Array  
(  
    [0] => Andi  
    [1] => andi@gmail.com  
    [2] => 15  
    [3] => SMA N 7 lumut ijo  
    [4] => Jl. Perintis no 9  
    [5] => 85  
)
```

Dalam contoh ini saya ingin meng-update tabel `user` dengan 5 buah data, yakni `username`, `email`, `umur`, `sekolah` dan `alamat`. Kondisi update adalah kolom `id_user` = 5.

Seperti yang terlihat, query prepared statement berhasil di generate dan disimpan ke dalam variabel `$query`. Kemudian terdapat 6 buah data placeholder yang disimpan dalam variabel `$dataValues`.

Karena proses pembuatan query sudah selesai, tinggal memindahkannya ke dalam class DB:

```
21.DB_method_update/DB.php
```

```
1 <?php  
2 class DB{
```

Case Study: Database Query Builder

```
3 // ...
4 // ...
5
6
7 public function update($tableName, $data, $condition){
8     $query = "UPDATE {$tableName} SET ";
9     foreach ($data as $key => $val){
10         $query .= "{$key} = ?, ";
11     }
12     $query = substr($query,0,-2);
13     $query .= " WHERE {$condition[0]} {$condition[1]} ?";
14
15     $dataValues = array_values($data);
16     array_push($dataValues,$condition[2]);
17
18     $this->runQuery($query,$dataValues)->rowCount();
19 }
20
21 }
```

Isi method `update()` ini tinggal di copy dari hasil percobaan fungsi `update()` sebelumnya. Dan sama seperti method `insert()`, di baris 18 proses menjalankan query dialihkan ke method `runQuery()`. Mari kita test proses update yang sebenarnya:

21.DB_method_update/update_method.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $DB->update('barang',
6             ['nama_barang' => 'Smartphone iPhone XR',
7              'harga_barang' => 17999000],
8             ['id_barang','=',5]);
9
10 // tampilkan semua tabel barang
11 $tabelBarang = $DB->getWhere('barang',[ 'id_barang','=',5]);
12
13 echo "<pre>";
14 print_r($tabelBarang);
15 echo "</pre>";
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 5
            [nama_barang] => Smartphone iPhone XR
            [jumlah_barang] => 25
            [harga_barang] => 17999000
            [tanggal_update] => 2019-03-16 07:50:09
        )
)
```

```
)
```

Hasilnya, tabel barang dengan `id_barang = 5` sukses di update.

Sentuhan terakhir, saya ingin menambah kode program untuk mengetahui jumlah baris yang di update. Caranya juga sama seperti method `insert()`, yakni mengakses method `rowCount()` bawaan PDO dan menyimpannya ke dalam private property `$_count`:

22.DB_method_update_count/DB.php

```
1 <?php
2 class DB{
3
4     // ...
5     // ...
6
7     public function update($tableName, $data, $condition){
8         $query = "UPDATE {$tableName} SET ";
9         foreach ($data as $key => $val){
10             $query .= "{$key} = ?, ";
11         }
12         $query = substr($query,0,-2);
13         $query .= " WHERE {$condition[0]} {$condition[1]} ?";
14
15         $dataValues = array_values($data);
16         array_push($dataValues,$condition[2]);
17
18         $this->_count = $this->runQuery($query,$dataValues)->rowCount();
19         return true;
20     }
21 }
```

Perubahan dari kode sebelumnya ada di baris 18 dan 19, dimana terdapat perintah untuk meng-update isi property `$_count`, serta perintah `return true`.

Dengan tambahan kode ini, kita bisa mendapat info mengenai jumlah baris kolom yang di update:

22.DB_method_update_count/DB.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->update('barang',
6                         ['nama_barang' => 'Dummy Product',
7                          'harga_barang' => 999999],
8                         ['id_barang','>',3]);
9
10 if($result) {
11     echo "Terdapat ".$DB->count()." data yang diubah";
12     // Terdapat 4 data yang diubah
```

```
13 }
```

Dalam kode program ini kondisi update saya tulis sebagai `['id_barang', '>', 3]`. Artinya, seluruh barang yang memiliki `id_barang` lebih dari 3 akan di update. Dari hasil pemanggilan method `$DB->count()` terlihat bahwa ada 4 baris data yang berhasil di update.

Jika kode di atas saya jalankan sekali lagi, hasilnya menjadi "Terdapat 0 data yang diubah". Ini terjadi karena method `DB->count()` hanya akan mengembalikan jumlah tabel yang terdampak (*affected rows*). Jika sebuah query `UPDATE` tidak melakukan perubahan apapun, maka hasilnya 0, meskipun query tersebut sukses dijalankan.

Karena alasan ini pula kita tidak bisa berpatokan kepada hasil `DB->count()` mengenai sukses atau tidaknya sebuah query. Jika hasilnya 0, bukan berarti itu query gagal dijalankan, tapi hanya tidak ada baris tabel yang berubah.

11.18. Class DB: Method delete

Method `delete()` ini menjadi method terakhir yang akan kita buat untuk class DB. Sesuai dengan namanya, method `delete()` dipakai untuk menghapus sebuah baris tabel.

Berikut contoh pemanggilannya dari halaman `index.php`:

23.DB_method_delete/index_1.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->delete('barang',[ 'id_barang' , '=' , 4]);
6
7 if($result) {
8     echo "Terdapat ".$DB->count()." data yang dihapus";
9 }
```

Method `delete()` saya rancang dengan 2 buah argument. Argumen pertama berupa nama tabel yang akan dihapus, dan argument kedua berupa kondisi WHERE untuk proses penghapusan.

Berikut kode yang dipakai untuk membuat method `delete()` di class DB:

23.DB_method_delete/DB.php

```
1 <?php
2 class DB{
3
4     // ..
5     // ..
6     public function delete($tableName, $condition){
7         $query = "DELETE FROM {$tableName} WHERE {$condition[0]} {$condition[1]} ? ";
8         $this->_count = $this->runQuery($query,[ $condition[2] ])->rowCount();
```

```
9     return true;
10    }
11 }
```

Proses pembuatan query `DELETE` cukup sederhana, dimana saya merancang sebuah string prepared statement di baris 7. Semua data yang diperlukan tinggal diambil dari argument `$tableName` dan `$condition`. Di baris 8, proses lanjutan diserahkan ke pada method `runQuery()` serta meng-update isi property `$this->_count`.

Sehingga ketika method `delete()` dipanggil dengan kode berikut:

```
$DB->delete('barang',[ 'id_barang' , '=' ,4]);
```

Akan di translate menjadi query: "DELETE FROM barang WHERE id_barang = ?".

Sebagai latihan, bisakah anda membuat kode program untuk menghapus isi tabel barang yang memiliki `id_barang` kurang dari 5?

Berikut kode program yang dibutuhkan:

23.DB_method_delete/index_2.php

```
1 <?php
2 require 'DB.php';
3 $DB = DB::getInstance();
4
5 $result = $DB->delete('barang',[ 'id_barang' , '<' ,5]);
6
7 if($result) {
8     echo "Terdapat ".$DB->count()." data yang dihapus";
9     // Terdapat 4 data yang dihapus
10 }
```

Hasilnya, 4 buah data yang memiliki `id_barang` < 5 akan di hapus dari tabel barang.

Method `delete()` ini menutup studi kasus kita dalam membuat sebuah library query builder MySQL. Meskipun terasa rumit, tapi ini masih sebuah query builder versi sederhana.

Cukup banyak batasan query MySQL yang belum bisa kita proses, sebagai contoh class `DB` ini tidak menyediakan method query builder untuk lebih dari 1 kondisi, seperti `SELECT * FROM barang WHERE id_barang = 2 AND id_barang = 5`. Termasuk query yang kompleks seperti `JOIN`.

Jika anda tertarik, silahkan kembangkan class `DB` ini lebih jauh lagi. Nantinya class ini bisa dipakai untuk berbagai project. Daripada membuat ulang semua kode program untuk mengakses database MySQL, kita tinggal mengcopy isi class `DB`. Inilah prinsip dari `code reuse`, atau filosofi **DRY** (*don't repeat yourselves*) yang bisa diterapkan dari pemrograman berorientasi objek.

Exercise

Sebagai persiapan untuk bab berikutnya, kita akan buat sedikit latihan:

1. Rancang sebuah kode PHP untuk meng-generate tabel `user` yang berisi 3 buah kolom:

- `username` VARCHAR(50) PRIMARY KEY
- `password` VARCHAR(255)
- `email` VARCHAR(100)

Tabel `user` ini berada di dalam database `ilkoom`. Teknik yang dipakai sama seperti yang pernah kita lakukan, yakni buat sebuah file `generate`. Dengan menjalankan file tersebut, otomatis tabel `user` sudah langsung dibuat. Anda boleh menggunakan mysqli object maupun PDO.

2. Menggunakan method `insert()` dari class DB, input 2 data berikut ke dalam tabel `user`:

username: alex, password: alex123, email: alexsaja@yahoo.com,
username: rina, password: 123456, email: rinapunya@gmail.com

3. Menggunakan method `get()` dari class DB, tampilkan seluruh isi tabel `user`.

4. Menggunakan method `select()` dan `get()` dari class DB, tampilkan kolom `username` dan `email` dari seluruh isi tabel `user`.

5. Menggunakan method `update()`, tukar alamat email dari username `alex` menjadi `alex999@gmail.com`.

6. Menggunakan method `getWhereOnce()` dari class DB, tampilkan `username` dan `email` dari username `alex`.

7. Menggunakan method `delete()` dari class DB, hapus `username alex` dari tabel `user`.

Kode program untuk jawaban dari latihan ini bisa anda lihat di folder `belajar_oop_php\bab_11\24.DB_latihan\`

Dalam bab ini kita telah merancang studi kasus penerapan konsep OOP dengan membuat sebuah library query builder MySQL, yakni class `DB`. Class `DB` ini sepenuhnya modular sehingga bisa dipakai untuk project-project lain.

Studi kasus berikutnya kita masih membuat library lain, kali ini dipakai untuk proses validasi form, yakni class `Validate`.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini. Hak cipta eBook sudah terdaftar di Depkumham RI. Pelanggaran akan dituntut sesuai UU yang berlaku.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

12. Case Study: Validate Class

Dalam bab sebelumnya kita telah bahas studi kasus pembuatan **database query builder class**. Dengan class tersebut, proses komunikasi ke database menjadi lebih mudah dan singkat. Terlebih hampir setiap aplikasi yang melibatkan PHP butuh menyimpan data ke database MySQL.

Hal lain yang juga hampir selalu ada di setiap aplikasi web adalah *form processing*. Pekerjaan ini sangat menantang namun juga membosankan. Menantang karena banyak hal yang harus dipikirkan, terutama bagaimana mencegah data yang tidak valid agar tidak masuk ke database (proses validasi). Namun juga membosankan karena untuk setiap form, kita perlu membuat proses validasi. Tidak jarang form-form ini mirip satu sama lain.

Oleh karena itu proses validasi form menjadi kandidat paling pas untuk dibuatkan class tersendiri. Tujuannya, agar kita hanya perlu menulis 1 buah class validasi yang bisa dipakai untuk semua form. Dengan memanfaatkan teknik pemrograman object, class bisa dibuat terpisah dan dapat dipakai dalam berbagai project.

Sama seperti studi kasus class DB, sepanjang pembuatan class Validate kita akan bahas bagaimana teknik yang saya pakai untuk memecahkan masalah. Harap juga diingat bahwa ada banyak solusi untuk membuat proses validasi. Dalam hal ini saya lebih mencari titik tengah antara "mudah dipelajari" dengan "performa".

Jika hanya berfokus kepada performa, teknik yang diperlukan bisa lebih rumit, misalnya melibatkan berbagai trik *design pattern*. Agar tidak terlalu panjang, class Validate yang akan kita rancang juga saya batasi untuk jenis validasi umum saja. Namun class ini sangat fleksibel dan bisa dikembangkan lebih lanjut sesuai kebutuhan anda.

12.1. Mempersiapkan Form Tambah Barang

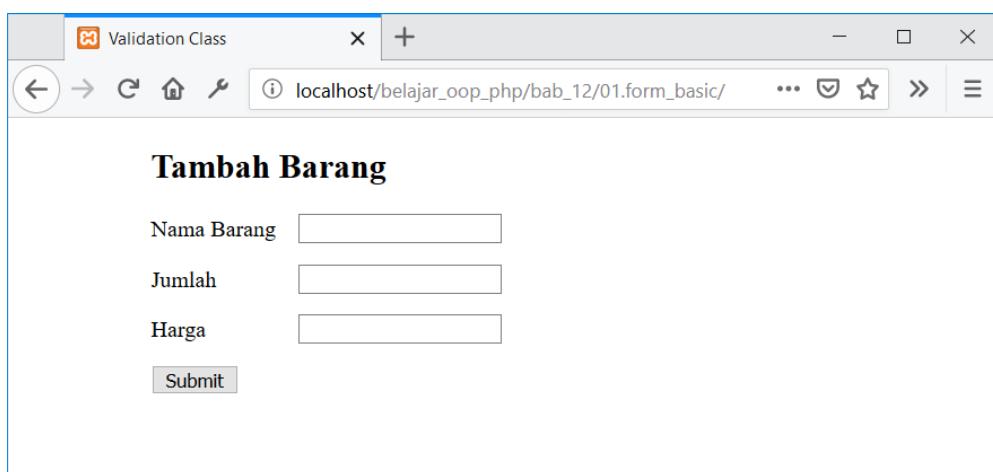
Sebagai bahan praktek, kita butuh sebuah form. Form ini akan saya pakai dalam beberapa pembahasan ke depan:

```
01.form_basic/index.html

1  <!doctype html>
2  <html lang="id">
3    <head>
4      <meta charset="utf-8">
5      <title>Validation Class</title>
```

Case Study: Validate Class

```
6   </head>
7   <style>
8     .container {
9       margin: 20px auto;
10      width: 500px;
11    }
12   form > div {
13     margin: 15px 0;
14   }
15   label {
16     display:inline-block;
17     width:100px;
18   }
19 </style>
20 <body>
21   <div class="container">
22     <h2>Tambah Barang</h2>
23     <div class="pesan-error">
24     </div>
25     <form method="post">
26       <div>
27         <label for="nama_barang">Nama Barang</label>
28         <input type="text" name="nama_barang">
29       </div>
30       <div>
31         <label for="jumlah_barang">Jumlah</label>
32         <input type="text" name="jumlah_barang">
33       </div>
34       <div>
35         <label for="harga_barang">Harga</label>
36         <input type="text" name="harga_barang">
37       </div>
38       <div>
39         <input type="submit" value="Submit">
40       </div>
41     </form>
42   </div>
43 </body>
44 </html>
```



Gambar: Form untuk tambah barang

Kode di atas hanya terdiri dari HTML dan sedikit kode CSS untuk merapikan tampilan form.

Dalam halaman ini terdapat 3 inputan form dari tag `<input type="text">` dengan atribut `name = "nama_barang"`, `name = "jumlah_barang"`, dan `name = "harga_barang"`. Atribut `name` inilah yang nantinya kita perlukan ketika mengakses nilai setiap inputan.

Di baris 23 – 24 terdapat sebuah tag `<div class="pesan-error">` yang saat ini belum berisi kode apapun. Nantinya di bagian ini pesan error akan ditampilkan. Form dikirim dengan `method="post"` dan karena tidak terdapat atribut `action`, maka isian form secara default akan dikirim ke halaman yang sama.

Baik, mari kita buat kode program untuk menampilkan setiap inputan form. Tempatkan kode PHP berikut di bagian paling atas (sebelum kode HTML):

02.form_basic_submit/index.php

```

1  <?php
2  if (!empty($_POST)) {
3      if (isset ($_POST['nama_barang'])) {
4          echo $_POST['nama_barang']. "<br>";
5      }
6      if (isset ($_POST['jumlah_barang'])) {
7          echo $_POST['jumlah_barang']. "<br>";
8      }
9      if (isset ($_POST['harga_barang'])) {
10         echo $_POST['harga_barang']. "<br>";
11     }
12 }
13 ?>
14
15 <!doctype html>
16 <html lang="id">
17 ...

```

Di baris 2 terdapat kondisi `if (!empty($_POST))`. Ini saya pakai untuk mendeteksi apakah halaman diakses setelah tombol **submit** form di tekan. Jika iya, maka global variabel `$_POST` akan berisi "sesuatu". Kondisi `if (!empty($_POST))` akan bernilai **true** jika global variabel `$_POST` tidak kosong (*not empty*).

Jika `$_POST` terdeteksi tidak kosong, maka kita bisa lanjut untuk menampilkan semua inputan form. Kita bisa saja langsung menulis `echo $_POST['nama_barang']`, namun untuk jaga-jaga saya kembali memeriksa isi dari setiap element yang ada di dalam `$_POST`. Maksudnya, hanya jika `$_POST['nama_barang']` terdefinisi, baru tampilkan isinya.

Pemeriksaan kondisi seperti ini sebenarnya tidak perlu untuk tag `<input type="text">`, karena jika pun form tidak diisi, element `$_POST` akan tetap berisi string kosong. Namun ini tidak berlaku untuk `<input type="checkbox">` dan `<input type="radio">`, dimana jika inputan checkbox atau radio tidak di pilih, perintah `echo` akan menghasilkan error karena variabel tersebut tidak terdefinisi.

Berikut tampilan form ketika setelah diisi:

The screenshot shows a browser window titled "Validation Class". The address bar displays "localhost/belajar_oop_php/bab_12/02.form_basic". The main content area shows a form titled "Tambah Barang" with three input fields: "Nama Barang" (with value "Laptop Acer"), "Jumlah" (with value "5"), and "Harga" (with value "6599000"). Below the form is a "Submit" button.

Gambar: Tampilan isian form tambah barang

Karena kita memproses kode PHP di bagian atas, maka hasilnya juga tampil di sudut kanan atas. Agar kode program kita tidak terlalu kompleks, hasil tampilan form memang "ala kadarnya". Tapi yang penting kita bisa sudah melihat hasil form.

12.2. Class Input: Method get

Dalam bab ini kita akan membuat 2 buah class. Yakni class `Validate` untuk proses validasi, serta class `Input` yang saya rancang untuk menampung berbagai method bantu.

Method pertama untuk class `Input` adalah `get()`. Method ini berfungsi sebagai shortcut dari kondisi `if` yang kita tulis sebelumnya. Daripada memeriksa satu per satu inputan form, akan lebih baik dibuat method khusus supaya kode program menjadi lebih rapi.

Nantinya, saya ingin mengubah kode program berikut:

```
if (isset ($_POST['nama_barang'])) {  
    echo $_POST['nama_barang']. "<br>";  
}
```

Menjadi:

```
echo Input::get('nama_barang'). "<br>";
```

Dari pemanggilan ini bisa ditebak kalau `get()` merupakan static method dari class `Input`.

Silahkan buat sebuah file baru bernama `Input.php`, lalu isi dengan kode berikut:

03.class_input_get/Input.php

```
1 <?php  
2 class Input{
```

```

3
4     public static function get($item) {
5         if (isset($_POST[$item])) {
6             return trim($_POST[$item]);
7         }
8         else if (isset($_GET[$item])) {
9             return trim($_GET[$item]);
10        }
11        return '';
12    }
13
14 }
```

Class `Input` saat ini berisi sebuah static method `get()`. Method `get()` butuh satu argument `$item`. Nantinya, `$item` akan berisi nilai atribut `name` yang berasal dari inputan form.

Prinsip kerja dari method `get()` cukup sederhana, yakni jika `$_POST[$item]` terdefinisi, maka kembalikan nilai `trim($_POST[$item])`. Jika tidak, coba cari apakah `$_GET[$item]` terdefinisi, jika iya, kembalikan nilai `trim($_GET[$item])`. Jika ternyata tidak ada di array `$_POST` maupun `$_GET`, maka kembalikan string kosong ''.

Fungsi `trim()` saya tambah untuk menghapus spasi di awal dan akhir inputan form. Tambahan spasi ini bisa jadi masalah karena dapat menyebabkan gagal login atau salah validasi.

Dengan method `get()`, kita tidak perlu khawatir tampil pesan error ketika mengakses inputan form yang belum terdefinisi. Berikut modifikasi kode sebelumnya:

03.class_input_get/index.php

```

1 <?php
2 require 'Input.php';
3
4 if (!empty($_POST)) {
5     echo Input::get('nama_barang'). "<br>";
6     echo Input::get('jumlah_barang'). "<br>";
7     echo Input::get('harga_barang'). "<br>";
8 }
9
10 ?>
11 <!doctype html>
12 <html lang="id">
13 ...
```

Dengan ini, proses pemeriksaan inputan form akan ditangani oleh method `get()`.

Selain itu kita juga bisa membuat kode program untuk proses re-populate form memanfaatkan method `get()`:

03.class_input_get/index.php

```

1 ...
2     <form method="post">
```

```

3   <div>
4     <label for="nama_barang">Nama Barang</label>
5     <input type="text" name="nama_barang"
6       value="<?php echo Input::get('nama_barang') ?>">
7   </div>
8   <div>
9     <label for="jumlah_barang">Jumlah</label>
10    <input type="text" name="jumlah_barang"
11      value="<?php echo Input::get('jumlah_barang') ?>">
12  </div>
13  <div>
14    <label for="harga_barang">Harga</label>
15    <input type="text" name="harga_barang"
16      value="<?php echo Input::get('harga_barang') ?>">
17  </div>
18  <div>
19    <input type="submit" value="Submit">
20  </div>
21 </form>
22 ...

```

Di sini saya mengisi nilai atribut `value` dari setiap inputan form dengan hasil pemanggilan method `get()` di baris 6, 11, dan 16. Pemeriksaan kondisi dalam method `get()` ini secara otomatis mengantisipasi 2 kondisi:

- Saat form dibuka pertama kali di web browser, variabel global `$_POST` belum berisi nilai apapun, sehingga method `get()` akan mengembalikan string kosong '' . Hasilnya, inputan form tidak berisi teks apapun.
- Jika form sudah diisi dan user men-klik tombol **submit**, maka method `get()` sudah otomatis berisi data. Data inilah yang akan menjadi nilai awal dari atribut `value`. Ini sangat memudahkan user karena tidak harus menginput ulang semua inputan form jika ada validasi yang gagal.

Untuk uji coba, silahkan isi form lalu klik tombol submit:

The screenshot shows a web browser window with the title "Validation Class". The address bar displays "localhost/belajar_oop_php/bab_12/03.class_input". The main content area shows a form titled "Tambah Barang" with three input fields:

Nama Barang	Asus Zenfone Max Pro M2
Jumlah	8
Harga	3750000

Below the form is a "Submit" button.

Gambar: Tampilan hasil submit form tambah barang

Sekarang selain nilai form tampil di sudut kiri atas, setiap inputan form juga sudah memiliki nilai hasil dari proses sebelumnya.

12.3. Function filter_var()

Proses "bersih-bersih" inputan form bisa dibagi ke dalam 2 bagian: **sanitizing** dan **validation**. Saya tetap memakai istilah bahasa inggris karena agak rancu ketika diterjemahkan ke dalam bahasa Indonesia.

Sanitizing adalah proses pembersihan data. Salah satunya sudah kita buat ke dalam method `get()`, yakni memakai fungsi `trim()` bawaan PHP untuk menghapus tambahan spasi di awal dan akhir nilai. Biasanya tambahan spasi ini karena user tidak sengaja menekan tombol spasi.

Selain itu, mayoritas inputan form juga harus dibersihkan dari tambahan kode HTML. Misalnya untuk nama barang, bisa saja ada user yang menginput `<h1>Laptop Dell</h1>`. Tambahan tag `<h1>` ini harusnya tidak boleh ada. Atau lebih parah ada yang coba menginput tag `<script>` untuk menjalankan kode JavaScript. Teknik injeksi kode seperti ini sangat berbahaya.

Oleh karena itu, inputan form perlu di bersihkan dan inilah yang disebut sebagai proses **sanitizing**.

Sedangkan **Validation** adalah proses memeriksa inputan form apakah sudah sesuai dengan aturan yang kita inginkan atau belum. Misalnya nama barang minimal harus 6 karakter, harga barang harus berupa angka, jumlah barang tidak boleh negatif, dsb.

Kita akan bahas proses **sanitizing** terlebih dahulu, untuk validation akan dibahas pada bagian tersendiri.

Ada banyak cara untuk membuat proses sanitizing. Di dalam studi kasus buku **PHP Uncover**, saya membuatnya dengan teknik berikut:

```
$nim = htmlentities(strip_tags(trim($_POST["nim"])));
```

Di sini terdapat gabungan 3 fungsi sekaligus, `trim()` untuk menghapus spasi, `strip_tags()` untuk menghapus tambahan tag-tag HTML, serta `htmlentities()` untuk mengubah karakter khusus agar menjadi kode HTML entity.

Ketiga fungsi ini masih bisa dipakai. Namun saya ingin menggunakan fungsi sanitizing khusus yang disediakan oleh PHP, yakni **filter_var()**.

Fungsi `filter_var()` merupakan fungsi "all in one", maksudnya fungsi ini menyediakan proses sanitizing untuk berbagai tipe data. Bagaimana proses sanitizing dilakukan, akan diatur dari inputan argument fungsi. Berikut format dasar dari fungsi `filter_var()`:

```
filter_var(nilai_asal, KONSTANTA_1, [KONSTANTA_2, ...])
```

Argument pertama, 'nilai_asal' di isi nilai yang akan "dibersihkan". Ini biasanya berasal dari

inputan form. Kemudian sebagai argument kedua terdapat KONSTANTA_1 yang akan menentukan seperti apa proses sanitizing berjalan. Kadang diperlukan juga konstanta lain yang ditulis sebagai argument ketiga, keempat, dst.

PHP menyediakan cukup banyak konstanta untuk mengatur validasi, namun kita akan lihat yang cukup umum saja. Jika tertarik, bisa melihat daftar lengkapnya ke [sanitize filters](#) di PHP Manual.

String Sanitizing

Konstanta pertama yang akan kita bahas adalah FILTER_SANITIZE_STRING. Konstanta ini dipakai untuk membersihkan tipe data string. Berikut contoh penggunaannya:

04.function_filter_var/1.filter_var_FILTER_SANITIZE_STRING.php

```

1 <?php
2
3 echo filter_var("<b>FooBar</b>", FILTER_SANITIZE_STRING);
4 //FooBar
5
6 echo filter_var("<script onclick='danger()'>FooBar</script>",
7 FILTER_SANITIZE_STRING);
8 //FooBar
9
10 echo filter_var(" FooBar__ _ ", FILTER_SANITIZE_STRING);
11 // FooBar__ _
12
13 echo filter_var("&Foo'Bar câfè", FILTER_SANITIZE_STRING);
14 //&Foo';Bar câfè

```

Pada contoh ini saya menginput berbagai string awal ke fungsi `filter_var()`. Di baris 3 dan 6 terlihat tag HTML akan dihapus dari string, yakni `` dan `<script>`.

Contoh di baris 10 juga memperlihatkan bahwa fungsi `filter_var()` dengan konstanta `FILTER_SANITIZE_STRING` tidak melakukan proses `trim()`, dimana tambahan spasi di awal dan akhir karakter tetap dibiarkan.

Pada baris 13, tanda kutip satu ' akan di konversi menjadi kode HTML entity `'`, namun untuk karakter lain seperti â atau è tetap dalam bentuk aslinya. Sebagai tambahan, karakter HTML entity `'` hanya di proses secara internal. Jika dilihat dari web browser, tetap tampil sebagai tanda kutip satu.

Kesimpulannya, secara default konstanta `FILTER_SANITIZE_STRING` akan menghapus tambahan tag-tag HTML serta mengkonversi tanda kutip.

Int Sanitizing

Untuk membersihkan karakter integer (angka bulat), terdapat konstanta `FILTER_SANITIZE_NUMBER_INT`. Berikut contoh penggunaannya:

04.function_filter_var/2.filter_var_FILTER_SANITIZE_NUMBER_INT.php

```
1 <?php
2
3 echo filter_var(12, FILTER_SANITIZE_NUMBER_INT);
4 //12
5
6 echo filter_var(12.45, FILTER_SANITIZE_NUMBER_INT);
7 //1245
8
9 echo filter_var("FooBar", FILTER_SANITIZE_NUMBER_INT);
10 //
11
12 echo filter_var("99FooBar9", FILTER_SANITIZE_NUMBER_INT);
13 //999
```

Dari 4 percobaan ini bisa terlihat FILTER_SANITIZE_NUMBER_INT akan menghapus seluruh karakter selain angka dan mengambil hanya bagian angka saja. Di baris 6, angka 12.45 dibersihkan menjadi 1245, sedangkan di baris 12, string "99FooBar9" dibersihkan menjadi 999. Proses pembersihan seperti ini mungkin tidak cocok untuk sebagian aplikasi, misalnya angka 12.45 akan lebih baik di proses sebagai 12, bukan 1245.

Float Sanitizing

Konstanta FILTER_SANITIZE_NUMBER_FLOAT bisa dipakai untuk membuat proses sanitizing tipe data float atau angka pecahan. Berikut contohnya:

04.function_filter_var/3.filter_var_FILTER_SANITIZE_NUMBER_FLOAT.php

```
1 <?php
2
3 echo filter_var(12, FILTER_SANITIZE_NUMBER_FLOAT);
4 //12
5
6 echo filter_var(12.45, FILTER_SANITIZE_NUMBER_FLOAT);
7 //1245
8
9 echo filter_var(12.45, FILTER_SANITIZE_NUMBER_FLOAT,
10                 FILTER_FLAG_ALLOW_FRACTION);
11 //12.45
12
13 echo filter_var("12AA.45", FILTER_SANITIZE_NUMBER_FLOAT,
14                  FILTER_FLAG_ALLOW_FRACTION);
15 //12.45
```

Perhatikan hasil di baris 6, di situ angka 12.45 malah di proses menjadi 1245. Saya sendiri juga heran kenapa proses sanitizing tipe data float malah menghapus tanda titik yang dipakai untuk memisahkan bilangan pecahan.

Namun dengan tambahan konstanta kedua, yakni FILTER_FLAG_ALLOW_FRACTION, barulah proses pembersihan berjalan seperti yang diinginkan. Sama seperti pada tipe data integer,

proses sanitizing tipe data float juga akan membuang tambahan karakter non-angka seperti di baris 13.

Email Sanitizing

Proses pembersihan karakter juga tersedia untuk email, dimana fungsi `filter_var()` akan menghapus karakter yang tidak boleh ada di sebuah alamat email. Untuk keperluan ini kita menggunakan konstanta `FILTER_SANITIZE_EMAIL`. Berikut contohnya:

04.function_filter_var/4.filter_var_FILTER_SANITIZE_EMAIL.php

```
1 <?php
2
3 echo filter_var("foo@bar.com", FILTER_SANITIZE_EMAIL);
4 //foo@bar.com
5
6 echo filter_var("f0o@b4r$#?*.com", FILTER_SANITIZE_EMAIL);
7 //f0o@b4r$#?*.com
8
9 echo filter_var("(foo)@bar.co./id", FILTER_SANITIZE_EMAIL);
10 //foo@bar.co.id
```

Meskipun jarang terlihat, tapi mayoritas karakter yang agak aneh seperti \$, #, ?, dan * tetap dianggap valid untuk sebuah alamat email, ini terlihat dari hasil percobaan di baris 6. Di baris 7, karakter tanda kurung (,), dan / merupakan karakter yang tidak boleh ada di alamat email sehingga akan dibersihkan.

URL Sanitizing

Konstanta sanitizing terakhir yang akan kita bahas adalah untuk alamat URL, dengan konstanta `FILTER_SANITIZE_URL`:

04.function_filter_var/5.filter_var_FILTER_SANITIZE_URL.php

```
1 <?php
2
3 echo filter_var("https://www.duniaikom.com", FILTER_SANITIZE_URL);
4 //foo@bar.com
5
6 echo filter_var("https://www. duniaikom. com?s=php", FILTER_SANITIZE_URL);
7 //https://www.duniaikom.com?s=php
8
9 echo filter_var("https://www. duniaikom. com?s='câfè'^&*()", 
10           FILTER_SANITIZE_URL);
11 //https://www.duniaikom.com?s='cf'^&*()
12
13 echo filter_var("https://www. duniaikom. com?s=<script>test<script>", 
14           FILTER_SANITIZE_URL);
15 //https://www.duniaikom.com?s=<script>test<script>
```

Konstanta `FILTER_SANITIZE_URL` akan menghapus karakter yang tidak boleh ada di URL seperti karakter â dan è.

Namun perhatikan di baris 13, saya mencoba menyisipkan tag `<script>` ke dalam URL, dan ini tidak dihapus.

Secara umum, proses sanitizing hanya membersihkan karakter yang membuat sebuah URL tidak valid, namun tidak sampai ke membersihkan karakter berbahaya seperti tag `<script>`. Ini karena teks '`<script>`' tetap dianggap valid. Dengan demikian, kita bisa pertimbangkan memakai fungsi tambahan lain untuk proses sanitizing URL, seperti `htmlentities()` atau `strip_tags()`.

12.4. Class Input: Method `runSanitize`

Setelah mempelajari cara penggunaan fungsi `filter_var()`, kita akan coba terapkan untuk memproses form. Saya ingin setiap inputan form akan melewati proses sanitize terlebih dahulu.

Dan agar lebih fleksibel, saya akan buat method khusus di dalam class `Input`, yakni method `runSanitize()`. Method `runSanitize()` butuh 2 buah argument, berupa nilai form yang akan dibersihkan, serta tipe data yang sesuai. Berikut contoh pemanggilan method ini:

```
echo Input::runSanitize('<b>Hello</b>', 'string');
echo Input::runSanitize('12abc3', 'int'). "<br>";
```

Untuk contoh pertama, hasil akhirnya akan menampilkan 'Hello', dimana proses sanitizing tipe data `string` akan menghapus tambahan tag ``, sedangkan untuk contoh kedua akan mengembalikan nilai 123 sebagai hasil proses sanitizing tipe data `int`.

Method `runSanitize()` saya rancang sebagai static method, yang akan menjalankan proses sanitizing berbeda tergantung inputan tipe data di argument kedua. Berikut kode program yang dipakai untuk membuat method ini:

```
05.class_input_runSanitize/Input.php

1  <?php
2  class Input{
3
4      public static function get($item) {
5          // ...
6      }
7
8      public static function runSanitize($value,$sanitizeType){
9          switch($sanitizeType) {
10              case 'string':
11                  $sanitizeValue = filter_var($value, FILTER_SANITIZE_STRING);
12              break;
13              case 'int':
```

Case Study: Validate Class

```
14     $sanitizeValue = filter_var($value, FILTER_SANITIZE_NUMBER_INT);
15     break;
16     case 'float':
17         $sanitizeValue = filter_var($value, FILTER_SANITIZE_NUMBER_FLOAT,
18                                     FILTER_FLAG_ALLOW_FRACTION);
19         break;
20     case 'email':
21         $sanitizeValue = filter_var($value, FILTER_SANITIZE_EMAIL);
22         break;
23     case 'url':
24         $sanitizeValue = filter_var($value, FILTER_SANITIZE_URL);
25         break;
26     }
27     return $sanitizeValue;
28 }
29
30 }
```

Kondisi switch – case berfungsi untuk menjalankan kode program yang sesuai dengan tipe data. Sebagai contoh, jika argument \$sanitizeType berisi string 'int', maka perintah di baris 14 yang akan dijalankan. Kode ini men-sanitizing inputan \$value dan menyimpan hasilnya ke dalam variabel \$sanitizeValue. Terakhir, variabel \$sanitizeValue di kembalikan dengan perintah return \$sanitizeValue pada baris 27.

Dengan ini, kita sudah bisa menggunakan method runSanitize() dari dalam form:

05.class_input_runSanitize/index.php

```
1 <?php
2 require 'Input.php';
3
4 if (!empty($_POST)) {
5     echo Input::runSanitize(Input::get('nama_barang'), 'string'). "<br>";
6     echo Input::runSanitize(Input::get('jumlah_barang'), 'int'). "<br>";
7     echo Input::runSanitize(Input::get('harga_barang'), 'float'). "<br>";
8 }
9
10 ?>
11 <!doctype html>
12 <html lang="id">
13 ...
```

Saya langsung mengkombinasikan pemanggilan method Input::runSanitize() dengan Input::get(). Dimana argument pertama dari method runSanitize() langsung berasal dari method get().

Berikut hasil percobaannya:

The screenshot shows a web browser window titled "Validation Class". The address bar displays the URL "localhost/belajar_oop_php/bab_12/05.class_input_.php". The main content area shows a form titled "Tambah Barang". The form has three input fields: "Nama Barang" containing "<i>Mouse Logitech</i>", "Jumlah" containing "5buah13", and "Harga" containing "78903.aaaa9". Below the form is a "Submit" button.

Gambar: Tampilan hasil sanitizing isian form

Sip, sekarang semua inputan form sudah "dibersihkan". Jika dibutuhkan, anda juga bisa memakai method `runSanitize()` di atribut `value` tag `<input>` agar proses re-populate form menghasilkan nilai yang sama seperti hasil sanitize.

Namun perlu juga diingat bahwa proses sanitize ini tidak wajib dilakukan. Dalam banyak hal, syarat validasi akan memiliki aturan yang lebih ketat daripada sanitize.

Misalnya dalam proses validasi nanti, bisa saja kita membuat aturan bahwa nama barang hanya bisa diisi dengan karakter alfanumerik saja (huruf dan angka), sehingga otomatis karakter "<" yang dipakai untuk membuat tag HTML sudah tidak memenuhi syarat dan langsung tampil pesan error.

Atau misalnya jumlah barang kita batasi hanya bisa diisi angka saja, yang juga akan menampilkan pesan error jika teks "5buah13" diinput ke dalam form.

12.5. Function validate()

Proses sanitizing sudah selesai, saatnya kita masuk ke inti dari pembahasan studi kasus bab ini, yaitu membuat **proses validasi**.

Validasi pada dasarnya adalah membuat syarat khusus untuk setiap inputan form. Jika syarat ini tidak dipenuhi, tampilkan pesan error agar user bisa memperbaiki dan mengisi nilai yang sesuai.

Misalnya dalam form kita mensyaratkan nama barang, jumlah barang dan harga barang harus diisi (tidak boleh kosong), maka ketika user lupa mengisi salah satu inputan, akan tampil pesan error. Atau untuk jumlah barang harus diisi angka positif, jika tidak tampilkan juga pesan error.

Syarat validasi bisa berbeda untuk setiap form. Selain itu banyak teknik yang bisa dipakai untuk membuatnya. Dalam studi kasus ini, kita akan rancang sebuah class **Validate** yang berisi berbagai aturan validasi.

Namun sebelum sampai ke pembuatan class, saya ingin bahas dalam bentuk function terlebih dahulu. Ini semata-mata agar lebih mudah dipahami daripada langsung lompat ke bentuk class. Jika dirasa sudah cukup, baru kita masuk ke pembuatan class. Cara ini sama seperti yang saya pakai ketika membahas method `insert()` dan `update()` di bab sebelumnya.

Menampilkan Pesan Error

Hal pertama yang akan kita buat adalah menampilkan pesan error. Teknik yang saya pakai yakni dengan menyimpan semua pesan error ke sebuah array. Jika terdapat syarat validasi yang tidak terpenuhi, tambah 1 pesan error ke dalam array ini.

Berikut contoh penggunaannya:

06.function_validate/01.error_array.php

```
1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     if(empty(Input::get('nama_barang'))) {
9         $error[] = "Nama barang tidak boleh kosong";
10    }
11
12    if(empty(Input::get('jumlah_barang'))) {
13        $error[] = "Jumlah barang tidak boleh kosong";
14    }
15
16    if(empty(Input::get('harga_barang'))) {
17        $error[] = "Harga barang tidak boleh kosong";
18    }
19
20 }
21
22 ?>
23 <!doctype html>
24 <html lang="id">
25 ...
```

Di baris 4 terdapat pendefinisian array `$error`. Array inilah yang akan menampung seluruh pesan error.

Kemudian di baris 8 - 18 terdapat kode untuk validasi pertama kita, yakni memeriksa apakah inputan form sudah diisi atau belum. Caranya, gabungkan fungsi `empty()` dengan pemanggilan `Input::get('nama_barang')`.

Perintah di baris 8 – 10 bisa dibaca: Jika nilai inputan `nama_barang` kosong, maka isi teks "Nama barang tidak boleh kosong" ke dalam array `$error`. Begitu juga untuk inputan form

jumlah_barang dan harga_barang.

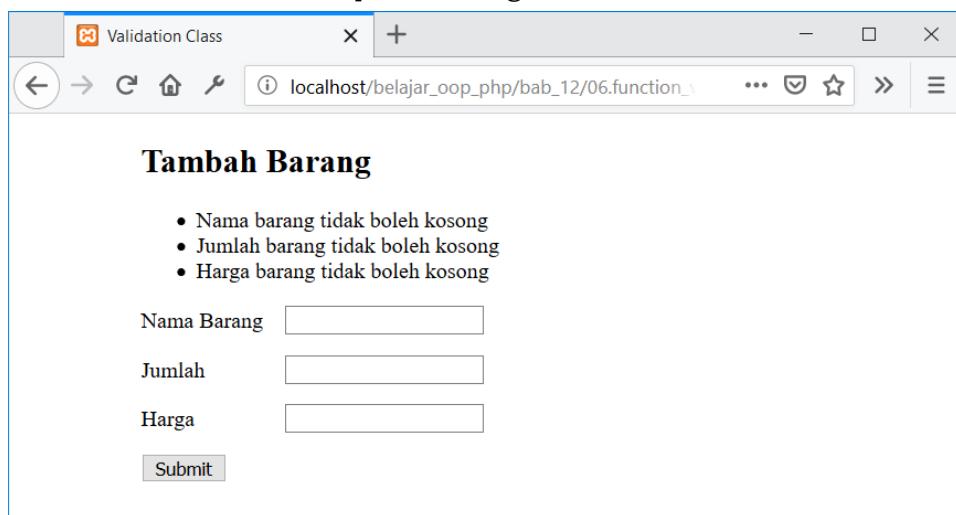
Untuk menyederhanakan kode program, saya tidak melakukan proses *sanitizing* ke hasil inputan form. Kita akan menambahkannya nanti.

Dengan kode program ini, ketika form di submit tanpa mengisi apapun, variabel \$error akan berisi 3 element, yakni pesan error untuk setiap inputan. Untuk menampilkan pesan error, tinggal buat sebuah perulangan `foreach` sebagai berikut:

06.function_validate/01.error_array.php

```
1 ...  
2 <body>  
3   <div class="container">  
4     <h2>Tambah Barang</h2>  
5     <div class="pesan-error">  
6       <ul>  
7         <?php  
8           foreach ($error as $value) {  
9             echo "<li>$value</li>";  
10            }  
11          ?>  
12        </ul>  
13      </div>  
14      <form method="post">  
15 ...
```

Pesan error saya tampilkan di dalam tag `<div class="pesan-error">` yang berada sebelum tag `<form>`. Isi variabel \$error akan ditampilkan sebagai sebuah unordered list:



Gambar: Tampilan error hasil validasi

Selama salah satu inputan form tidak diisi, maka akan tampil pesan error.

Function check_required

Kode untuk validasi kita sudah berjalan sebagaimana mestinya, namun cara penulisan seperti itu tidak fleksibel. Akan lebih baik jika dibuat sebuah fungsi khusus untuk memeriksa apakah sebuah inputan form masih kosong atau tidak.

Untuk keperluan ini, saya akan membuat fungsi **check_required()**. Fungsi ini nantinya butuh 2 buah argument. Argument pertama berupa nama inputan form yang berasal dari atribut name, dan argument kedua berupa string untuk meng-generate pesan error.

Berikut kode programnya:

06.function_validate/02.check_required_func.php

```

1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     function check_required($item, $itemLabel){
9         $formValue = Input::get($item);
10
11     global $error;
12     if (empty($formValue)) {
13         $error[] = "$itemLabel tidak boleh kosong";
14     }
15 }
16
17 check_required('nama_barang', 'Nama barang');
18 check_required('jumlah_barang', 'Jumlah barang');
19 check_required('harga_barang', 'Harga barang');
20 }
21
22 ?>
23 <!doctype html>
24 <html lang="id">
25 ...

```

Agar lebih mudah dipahami, kita mulai baca dari cara pemanggilan fungsi **check_required()** di baris 17:

```
check_required('nama_barang', 'Nama barang');
```

Argument pertama, 'nama_barang' adalah inputan form yang akan diperiksa. Nilainya harus sesuai dengan isi atribut name. Argument kedua, 'Nama barang' berisi string yang akan dipakai untuk membuat pesan error. Argument kedua ini tidak lain nama inputan form namun dengan format yang mudah dibaca.

Di dalam fungsi **check_required()**, argument pertama 'nama_barang' akan ditampung oleh

parameter \$item, sedangkan argument 'Nama Barang' akan ditampung ke \$itemLabel.

Kemudian di baris 9 saya menggunakan method get() dari class Input untuk mengambil nilai inputan form. Hasilnya disimpan ke dalam variabel \$formValue.

Ketiga variabel ini: \$item, \$itemValue, dan \$itemLabel akan banyak kita pakai sepanjang pembuatan proses validasi, dan juga di pembuatan class validate nantinya. Oleh karena itu sangat penting agar bisa membedakan ketiganya.

Setelah mendapat nilai inputan form di dalam variabel \$itemValue, pada baris 12 saya memeriksa isinya menggunakan kondisi if(empty(\$formValue)). Jika kondisi ini bernilai **true**, yang berarti nilai isian form kosong, maka input sebuah string "\$itemLabel tidak boleh kosong" ke dalam variabel \$error.

Karena \$itemLabel sudah berisi string 'Nama barang', maka pesan error yang tersimpan menjadi "Nama barang tidak boleh kosong".

Sebelumnya di baris 11 terdapat perintah global \$error. Ini berhubungan dengan scope atau ruang lingkup sebuah variabel. Keyword **global** diperlukan karena saya ingin mengakses variabel \$error yang berada di luar fungsi.

Dalam PHP, sebuah variabel yang ada di dalam function memiliki ruang lingkup yang terbatas, dimana hanya bisa diakses dari dalam function itu sendiri.

Jika tanpa menggunakan keyword **global**, maka kode program di baris 13 akan membuat variabel \$error baru yang hanya bisa diakses dari dalam function tersebut. Padahal yang kita inginkan adalah mengisi variabel \$error yang sudah disiapkan di luar function. Tambahan keyword **global** menginstruksikan PHP agar mengakses variabel \$error global, bukan variabel yang ada di dalam function.

Dengan fungsi `check_required()`, pembuatan syarat validasi menjadi lebih singkat. Untuk inputan form lain, cukup panggil fungsi ini beberapa kali:

```
check_required('jumlah_barang', 'Jumlah barang');
check_required('harga_barang', 'Harga barang');
```

Hasilnya, fungsi `check_required()` secara otomatis sudah memeriksa nilai form. Jika terdapat inputan form lain, tinggal lakukan hal yang sama. Ini jauh lebih baik daripada tanpa function.

Function `check_min_char`

Validasi kedua yang akan kita buat adalah untuk mengatur jumlah karakter minimal. Ini sering dipakai saat membuat username atau password. Agar lebih aman membatasi username minimal 6 karakter, jika kurang dari itu, tampilkan pesan error.

Teknik yang dipakai masih sama seperti sebelumnya, yakni memakai fungsi terpisah yang akan saya beri nama `check_min_char()`:

06.function_validate/03.min_char_func.php

```

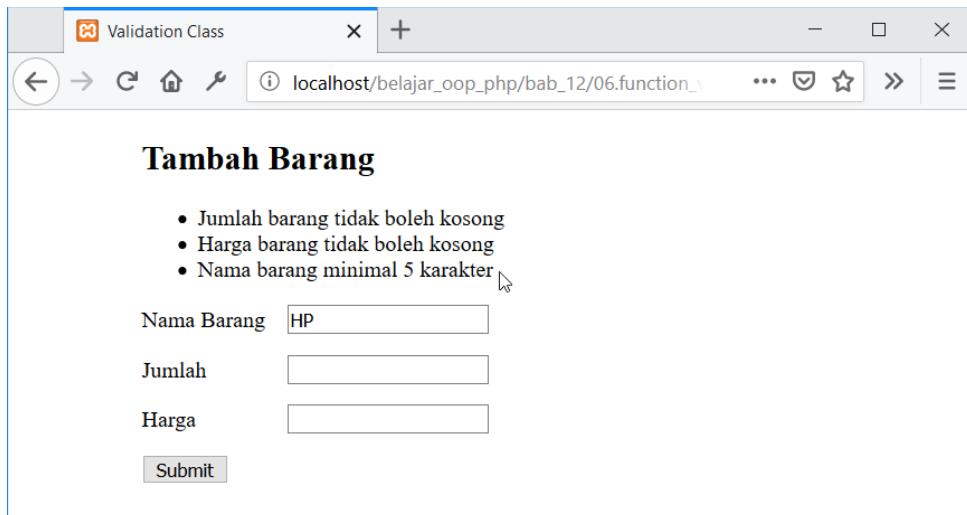
1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     function check_required($item, $itemLabel){
9         // ...
10    }
11
12     function check_min_char($item, $itemLabel, $ruleValue){
13         $formValue = Input::get($item);
14
15         global $error;
16         if (strlen($formValue) < $ruleValue) {
17             $error[] = "$itemLabel minimal $ruleValue karakter";
18         }
19     }
20
21     check_required('nama_barang', 'Nama barang');
22     check_required('jumlah_barang', 'Jumlah barang');
23     check_required('harga_barang', 'Harga barang');
24
25     check_min_char('nama_barang', 'Nama barang', 5);
26 }
27
28 ?>
29 <!doctype html>
30 <html lang="id">
31 ...

```

Fungsi `check_min_char()` butuh tambahan argument ketiga yang dipakai untuk menentukan jumlah karakter. Pemanggilan fungsi `check_min_char('nama_barang', 'Nama Barang', 5)` di baris 25 artinya saya ingin panjang inputan `nama_barang` minimal 5 karakter.

Di dalam fungsi `check_min_char()`, argument ketiga ini ditampung ke dalam parameter `$ruleValue`. Kondisi yang diperiksa adalah `if (strlen($formValue) < $ruleValue)`, yakni apakah total jumlah karakter `$formValue` kurang dari `$ruleValue`. Perintah `strlen()` sendiri merupakan fungsi bawaan PHP untuk mengetahui jumlah karakter di sebuah string.

Jika kita sengaja menginput nama barang yang kurang dari 5 karakter, akan tampil pesan error:



Gambar: Tampilan error karena karakter nama barang kurang dari 5

Function check_numeric

Khusus untuk inputan yang berbentuk angka seperti jumlah barang dan harga barang, kita harus memastikan nilai yang diinput memang angka, bukan huruf.

Untuk keperluan ini, bisa memanfaatkan fungsi `is_numeric()` bawaan PHP. Fungsi `is_numeric()` mengembalikan nilai `true` jika argument yang diperiksa berbentuk angka, atau `false` jika bukan angka. Berikut contohnya:

```
1 <?php
2 var_dump(is_numeric(45));           // true
3 var_dump(is_numeric('45'));         // true
4 var_dump(is_numeric('4.5'));        // true
5 var_dump(is_numeric('45a'));        // false
```

Selanjutnya tinggal membawa fungsi ini ke dalam syarat validasi kita, yang saya beri nama fungsi `check_numeric()`:

06.function_validate/04.check_numeric_func.php

```
1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     function check_required($item, $itemLabel){
9         //...
10    }
11
12    function check_min_char($item, $itemLabel, $ruleValue){
13        //...
14    }
```

```
15 function check_numeric($item, $itemLabel){  
16     $formValue = Input::get($item);  
17  
18     global $error;  
19     if (!is_numeric($formValue)) {  
20         $error[] = "$itemLabel harus diisi angka";  
21     }  
22 }  
23  
24 check_required('nama_barang', 'Nama barang');  
25 check_required('jumlah_barang', 'Jumlah barang');  
26 check_required('harga_barang', 'Harga barang');  
27  
28 check_min_char('nama_barang', 'Nama barang', 5);  
29  
30 check_numeric('jumlah_barang', 'Jumlah barang');  
31 check_numeric('harga_barang', 'Harga barang');  
32 }  
33  
34  
35 ?>  
36 <!doctype html>  
37 <html lang="id">  
38 ...
```

Cara pemanggilan dan isi fungsi `check_numeric()` mirip seperti `check_required()`, hanya saja sekarang kondisi yang diperiksa adalah `if (!is_numeric($formValue))`. Artinya jika `$formValue` tidak terdiri dari angka, isi pesan ke variabel `$error`.

Berikut hasil tampilan jika saya mengisi karakter non angka ke dalam inputan jumlah barang dan harga barang:



Gambar: Tampilan error karena nilai yang diinput bukan angka

Function Validate

Cara pembuatan validasi yang kita pakai sudah berjalan, tapi ada beberapa hal yang harus

diperbaiki.

Pertama, pemanggilan fungsi masih kurang praktis. Untuk setiap pemanggilan kita harus selalu menulis nama inputan form (argument pertama), serta string untuk men-generate pesan error (argument kedua). Selain itu di setiap fungsi juga harus mengakses kembali method `Input::get()`. Dengan kata lain ada banyak kode program yang berulang.

Kedua, validasi kita akan menampilkan semua pesan error sekaligus. Sebagai contoh, berikut hasil yang tampil jika saya tidak mengisi nilai apapun pada setiap inputan form:

Gambar: Error form akan ditampilkan semua

Ini tidak elegan sama sekali. Jika terdapat 10 syarat validasi, total akan tampil 10 pesan error, dan itu juga hanya untuk 1 inputan form.

Harusnya, pesan error cukup tampil 1 kali untuk setiap inputan form. Misalnya jika inputan nama barang tidak diisi nilai apapun, pesan error yang tampil cukup "*Nama barang tidak boleh kosong*". Pesan error "*Nama barang minimal 5 karakter*" tidak perlu tampil karena validasi pertama belum selesai di perbaiki.

Begitu juga untuk inputan jumlah barang. Jika pesan error "*Jumlah barang tidak boleh kosong*" tampil, maka pesan "*Jumlah barang harus diisi angka*" tidak perlu tampil.

Di sini kita perlu sebuah mekanisme agar pesan error cukup tampil 1 kali untuk setiap inputan. Jika validasi pertama tidak lolos, maka proses pemeriksaan berhenti sampai di situ dan tampilkan pesan error. Hanya jika validasi pertama lolos, baru masuk ke validasi kedua, dst. Dengan kata lain, harus ada sebuah urutan prioritas validasi.

Dalam contoh kita, validasi `check_min_char()` mestinya baru dijalankan jika validasi `check_required()` tidak menghasilkan error.

Untuk mengatasi masalah ini, saya akan rombak ulang proses validasi. Semuanya akan di

tangani oleh 1 fungsi saja, yakni fungsi `validate()`. Di dalam fungsi `validate()` inilah kita tempatkan kode-kode validasi. Kode programnya memang menjadi sedikit lebih kompleks, tapi juga lebih efisien.

Fungsi `validate()` nantinya butuh 3 argument. Dua argument pertama sama seperti sebelumnya, yakni nama inputan form serta string untuk meng-generate pesan error. Sedangkan argument ketiga akan berisi sebuah associative array yang berisi syarat validasi.

Berikut contoh pemanggilan fungsi `validate()`:

```
06.function_validate/05.validate.php

1 function validate($item,$itemLabel,$rules) {
2     // kode untuk proses validasi
3 }
4
5 validate('nama_barang','Nama barang', [
6     'required' => true,
7     'min_char' => 5
8 ]);
```

Perhatikan isi argument ketiga yang berbentuk associative array. Nantinya, associative array ini yang akan menjadi syarat validasi.

Pemanggilan fungsi `validate()` di baris 5 – 8 bisa dibaca: "Proses validasi untuk inputan `nama_barang`, dengan syarat tidak boleh kosong dan minimal berisi 5 digit karakter".

Untuk inputan form lain, caranya juga sama:

```
06.function_validate/05.validate.php

1 validate('jumlah_barang','Jumlah barang', [
2     'required' => true,
3     'numeric' => true,
4 ]);
5
6 validate('harga_barang','Harga barang', [
7     'required' => true,
8     'numeric' => true,
9 ]);
```

Kedua pemanggilan fungsi ini berarti saya ingin memeriksa inputan `jumlah_barang` dan `harga_barang` agar tidak boleh kosong dan harus bertipe angka.

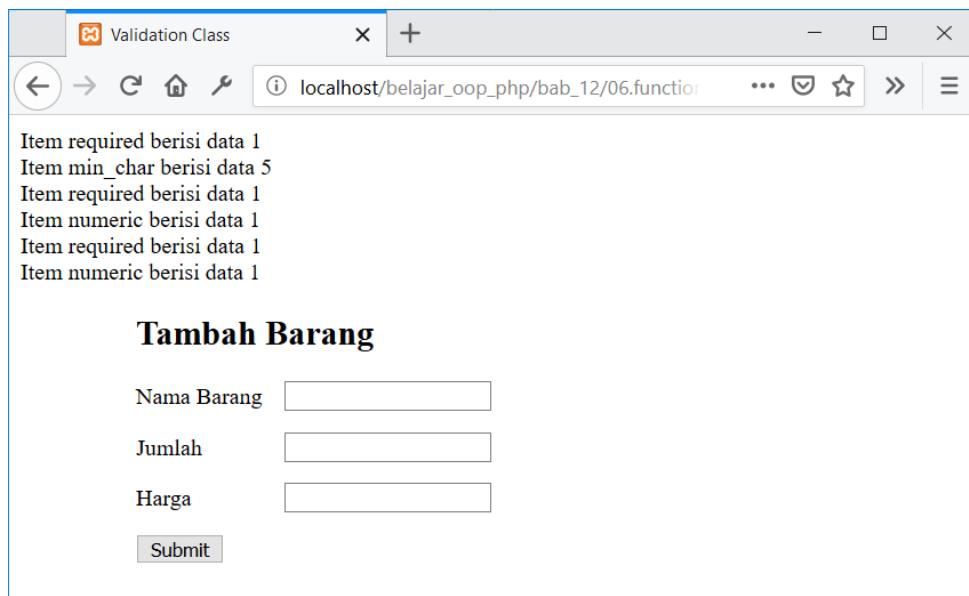
Penulisan associative array untuk argument ketiga sepenuhnya "suka-suka kita", maksudnya '`'required' => true` dan '`'numeric' => true`' adalah teks yang saya pilih sendiri. Anda pun bisa mengubahnya dengan nama lain. Nantinya, di dalam fungsi `validate-lah` kita akan rancang kode program untuk memprosesnya.

Baik, mari lanjut ke perancangan isi fungsi `validate()`. Yang harus dilakukan pertama kali adalah membaca ketiga argument yang ada. Berikut kode programnya:

06.function_validate/06.foreach.php

```
1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     function validate($item,$itemLabel,$rules) {
9         $formValue = Input::get($item);
10        global $error;
11
12        foreach ($rules as $rule => $ruleValue) {
13            echo "Item $rule berisi data $ruleValue <br>";
14        }
15    }
16
17    validate('nama_barang','Nama barang', [
18        'required' => true,
19        'min_char' => 5
20    ]);
21
22    validate('jumlah_barang','Jumlah barang', [
23        'required' => true,
24        'numeric' => true,
25    ]);
26
27    validate('harga_barang','Harga barang', [
28        'required' => true,
29        'numeric' => true,
30    ]);
31
32 }
33
34 ?>
35 <!doctype html>
36 <html lang="id">
37 ...
```

Hasil kode program:



Gambar: Hasil proses associative array fungsi validasi

Dalam kode program ini terdapat 3 kali pemanggilan fungsi `validate()`, yakni di baris 17, 22 dan 27. Setiap pemanggilan akan mengirim 3 buah argument.

Di dalam fungsi `validate()`, argument pertama disimpan ke dalam parameter `$item`. Di baris 9, parameter `$item` kemudian dipakai untuk mengambil nilai inputan form memakai method `Input::get()` dan disimpan ke dalam variabel `$itemValue`. Ini sama seperti yang kita lakukan sebelumnya.

Associative array yang menjadi argument ketiga disimpan ke dalam parameter `$rules`. Untuk menampilkan isinya, saya menggunakan perulangan `foreach` di baris 12 – 14. Di dalam perulangan `foreach` ini, variabel `$rule` akan berisi **key** dari associative array, sedangkan nilainya disimpan ke dalam `$ruleValue`.

Jika method `validate()` dipanggil sebagai berikut:

```
1 validate('nama_barang', 'Nama barang', [
2     'required' => true,
3     'min_char' => 5
4 ]);
```

Maka dalam iterasi atau perulangan pertama, string 'required' disimpan ke dalam `$rule`, sedangkan nilai `true` disimpan ke dalam `$ruleValue`. Untuk iterasi kedua, string 'min_char' disimpan ke dalam `$rule`, dan 5 disimpan ke dalam `$ruleValue`.

Karena dalam contoh sebelumnya terdapat 3 kali pemanggilan fungsi `validate()`, maka total ada 6 syarat validasi, yakni 2 untuk setiap inputan form sesuai dengan isi dari argument ketiga. Berikut nilai yang ditampilkan oleh perulangan `foreach`:

```
Item required berisi data 1
Item min_char berisi data 5
```

```
Item required berisi data 1
Item numeric berisi data 1
Item required berisi data 1
Item numeric berisi data 1
```

Nantinya, setiap inputan form bisa saja memiliki 3, atau bahkan 10 syarat validasi. Syarat validasi cukup ditulis sebagai associative array di argument ketiga dari fungsi validate().

Silahkan anda pelajari sejenak proses yang ada, terutama bagaimana cara memproses associative array agar bisa menjadi tampilan seperti di atas. Sebagai catatan, angka 1 yang tampil sebenarnya berasal dari boolean **true** yang dikonversi menjadi 1 karena perintah echo.

Validate: required

Sekarang kita akan fokus merancang validasi di dalam perulangan **foreach**. Validasi pertama sama seperti sebelumnya, yakni memastikan isian form tidak kosong. Berikut kode program yang diperlukan:

06.function_validate/07.case_required.php

```
1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     function validate($item,$itemLabel,$rules) {
9         $formValue = Input::get($item);
10        global $error;
11
12        foreach ($rules as $rule => $ruleValue) {
13
14            switch($rule) {
15                case 'required':
16                    if ($ruleValue === TRUE && empty($formValue)) {
17                        $error[$item] = "$itemLabel tidak boleh kosong";
18                    }
19                    break;
20            }
21        }
22    }
23 }
24
25 validate('nama_barang','Nama barang', [
26     'required' => true,
27     'min_char' => 5
28 ]);
29
30 // ...
```

Di dalam perulangan **foreach**, saya membuat sebuah block **switch-case**. Ini dipakai untuk

memeriksa satu per satu syarat validasi yang ditulis dalam associative array (argument ketiga fungsi validate()).

Jika anda sudah memahami kode program kita sebelumnya, maka di dalam perulangan foreach variabel \$rule akan berisi **key** dari associative array. Inilah yang akan menjadi nilai penentu block switch-case dengan cara memeriksa switch(\$rule) di baris 14. Misalnya jika isi dari \$rule berupa string 'required', maka jalankan blok kode program antara baris 16 – 18.

Agar bisa memahami kode program di atas, saya akan tulis kembali maksud dari setiap variabel:

- **\$item**: berisi string nama inputan form (berasal dari nilai atribut name dari tag <input>)
- **\$itemLabel**: berisi string nama inputan form untuk membuat pesan error
- **\$formValue**: nilai inputan form yang diisi oleh user (diakses dari Input::get(\$item))
- **\$rule**: key dari syarat validasi (berasal dari associative array)
- **\$ruleValue**: nilai dari syarat validasi (berasal dari associative array)

Dengan demikian, perintah berikut:

```
1 case 'required':  
2     if ($ruleValue === TRUE && empty($formValue)) {  
3         $error[$item] = "$itemLabel tidak boleh kosong";  
4     }  
5     break;
```

Artinya jika syarat validasi berisi string 'required', maka periksa apakah nilai syarat tersebut TRUE **dan** apakah isi inputan form kosong (maksud kondisi if di baris 2). Jika kedua kondisi terpenuhi, input sebuah pesan error dengan key \$item ke dalam array \$error.

Hasilnya berupa perintah berikut:

```
$error['nama_barang'] = "Nama Barang tidak boleh kosong";
```

Terakhir terdapat perintah break di baris 5 yang dipakai untuk keluar dari kondisi switch-case.

Validate: min_char

Syarat validasi kedua adalah memeriksa jumlah karakter. Berikut kode yang diperlukan:

06.function_validate/08.min_char.php

```
1 <?php  
2 require 'Input.php';  
3  
4 $error = [];  
5  
6 if (!empty($_POST)) {  
7
```

Case Study: Validate Class

```
8  function validate($item,$itemLabel,$rules) {
9      $formValue = Input::get($item);
10     global $error;
11
12     foreach ($rules as $rule => $ruleValue) {
13
14         switch($rule) {
15
16             case 'required':
17                 if ($ruleValue === TRUE && empty($formValue)) {
18                     $error[$item] = "$itemLabel tidak boleh kosong";
19                 }
20                 break;
21
22             case 'min_char' :
23                 if (strlen($formValue) < $ruleValue) {
24                     $error[$item] = "$itemLabel minimal $ruleValue karakter";
25                 }
26                 break;
27             }
28         }
29     }
30 }
31
32 validate('nama_barang','Nama barang', [
33     'required' => true,
34     'min_char'  => 5
35 ]);
36
37 // ...
```

Tambahan kode program ada di baris 22 – 26, yakni jika \$rule berisi string 'min_char', maka periksa if (strlen(\$formValue) < \$ruleValue). Jika hasilnya **true**, yang berarti panjang string form kurang dari syarat \$ruleValue, maka input pesan error ke dalam array \$error.

Pemanggilan fungsi validate() di baris 32 akan memproses syarat berikut:

```
case 'min_char' :
    if (strlen($formValue) < 5) {
        $error[nama_barang] = "Nama barang minimal 5 karakter";
    }
break;
```

Struktur penulisan validasi kita juga sudah mengatasi satu masalah lain. Sekarang seluruh proses validasi hanya dilakukan oleh 1 fungsi saja, sehingga kita tidak perlu mengambil nilai Input::get(\$item) berulang kali.

Untuk membuat syarat validasi lain, tinggal tambah blok case baru yang umumnya hanya butuh 2 - 3 baris kode program. Ini jauh lebih praktis.

Break Loop

Masalah validasi kedua yang harus kita pecahkan adalah membuat sebuah skala prioritas validasi. Yakni jika sebuah validasi sudah gagal, maka tidak perlu lagi memeriksa kondisi lain.

Sebagai contoh, misalnya sebuah inputan form memiliki syarat berikut:

```
[ 'required' => true,
  'min_char' => 5 ];
```

Maka apabila 'required' => true sudah menghasilkan error, maka syarat 'min_char' => 5 tidak perlu di proses lagi. Hanya jika syarat 'required' => true tidak menghasilkan error, barulah syarat 'min_char' => 5 di periksa.

Untuk membuat kondisi ini, kita perlu tambah sebuah pemeriksaan kondisi di akhir perulangan `foreach`:

06.function_validate/09.break_loop.php

```

1  <?php
2  require 'Input.php';
3
4  $error = [];
5
6  if (!empty($_POST)) {
7
8      function validate($item,$itemLabel,$rules) {
9          $formValue = Input::get($item);
10         global $error;
11
12         foreach ($rules as $rule => $ruleValue) {
13
14             switch($rule) {
15
16                 case 'required':
17                     if ($ruleValue === TRUE && empty($formValue)) {
18                         $error[$item] = "$itemLabel tidak boleh kosong";
19                     }
20                     break;
21
22                 case 'min_char' :
23                     if (strlen($formValue) < $ruleValue) {
24                         $error[$item] = "$itemLabel minimal $ruleValue karakter";
25                     }
26                     break;
27             }
28
29             // cek jika sudah ada error di item yang sama, langsung keluar dari looping
30             if (!empty($error[$item])) {
31                 break;
32             }
33         }
34     }
```

```
35    }
```

Tambahan perintah ada di baris 30 – 32. Perintah ini berada di luar block switch-case namun masih di dalam perulangan foreach.

Kondisi ini berarti untuk setiap iterasi atau perulangan foreach, cek apakah variabel \$error dengan key \$item sudah berisi sesuatu. Jika iya, itu artinya sudah ada validasi yang gagal dan break untuk langsung keluar dari perulangan.

Kembali seperti contoh sebelumnya:

```
[ 'required' => true,
  'min_char' => 5 ];
```

Jika syarat validasi 'required' => true sudah menghasilkan pesan error, maka program akan langsung keluar dari proses validasi. Dengan demikian, kita bisa pastikan pesan error yang tampil hanya 1 buah untuk setiap inputan form.

Validate: max_char

Dengan kode kita saat ini, akan lebih mudah menambah syarat validasi lainnya. Caranya, cukup membuat kondisi case baru. Sebagai contoh, saya ingin membuat validasi **max_char** untuk membatasi jumlah karakter maksimum. Maka bisa menggunakan kondisi case berikut:

```
1  case 'max_char' :
2    if (strlen($formValue) > $ruleValue) {
3      $error[$item] = "$itemLabel maksimal $ruleValue karakter";
4    }
5  break;
```

Kode programnya mirip seperti seperti **min_char**, hanya saja kali ini menggunakan operator lebih besar dari '>' :

Berikut kode program lengkap dari function validate() kita sejauh ini:

06.function_validate/10.max_char.php

```
1  <?php
2  require 'Input.php';
3
4  $error = [];
5
6  if (!empty($_POST)) {
7
8    function validate($item,$itemLabel,$rules) {
9      $formValue = Input::get($item);
10     global $error;
11
12     foreach ($rules as $rule => $ruleValue) {
13
14       switch($rule) {
```

Case Study: Validate Class

```
15     case 'required':
16         if ($ruleValue === TRUE && empty($formValue)) {
17             $error[$item] = "$itemLabel tidak boleh kosong";
18         }
19         break;
20
21     case 'min_char' :
22         if (strlen($formValue) < $ruleValue) {
23             $error[$item] = "$itemLabel minimal $ruleValue karakter";
24         }
25         break;
26
27     case 'max_char' :
28         if (strlen($formValue) > $ruleValue) {
29             $error[$item] = "$itemLabel maksimal $ruleValue karakter";
30         }
31         break;
32
33
34     }
35
36
37 // cek jika sudah ada error di item yang sama, Langsung keluar dari looping
38     if (!empty($error[$item])) {
39         break;
40     }
41
42 }
43 }
44
45 validate('nama_barang','Nama barang', [
46     'required' => true,
47     'min_char' => 5,
48     'max_char' => 10
49 ]);
50
51 validate('jumlah_barang','Jumlah barang', [
52     'required' => true,
53     'numeric' => true,
54 ]);
55
56 validate('harga_barang','Harga barang', [
57     'required' => true,
58     'numeric' => true,
59 ]);
60
61 }
62
63 ?>
64 <!doctype html>
65 <html lang="id">
66 ...
```

The screenshot shows a web browser window with the title 'Validation Class'. The URL in the address bar is 'localhost/belajar_oop_php/bab_12/06.function_validate/11.numeric.php'. The page content is titled 'Tambah Barang' and contains the following validation rule: '• Nama barang maksimal 10 karakter'. Below the rule are three input fields: 'Nama Barang' with value 'Mobil Toyota Avanza', 'Jumlah' with value 'Lima', and 'Harga' with value '229000000'. At the bottom is a 'Submit' button.

Gambar: Hasil proses associative array fungsi validasi

Error di atas tampil karena barang 'Mobil Toyota Avanza' sudah lebih dari 10 karakter. Ini membuktikan proses validasi sudah berjalan karena pada inputan **nama barang** saya membuat syarat validasi berikut:

```
'required' => true,  
'min_char' => 5,  
'max_char' => 10
```

Ini bisa dibaca: **nama barang** tidak boleh kosong, minimal 5 karakter dan maksimal 10 karakter. Syarat ini hanya untuk percobaan saja karena seharusnya panjang nama barang tidak perlu dibatasi.

Perhatikan juga inputan **Jumlah** saya isi dengan string 'Lima'. Ini tidak akan error karena validasi angka belum kita buat.

Validate: numeric

Sesuai dengan namanya, validasi **numeric** dipakai untuk memeriksa apakah sebuah inputan berisi angka atau tidak. Berikut kondisi case yang diperlukan:

06.function_validate/11.numeric.php

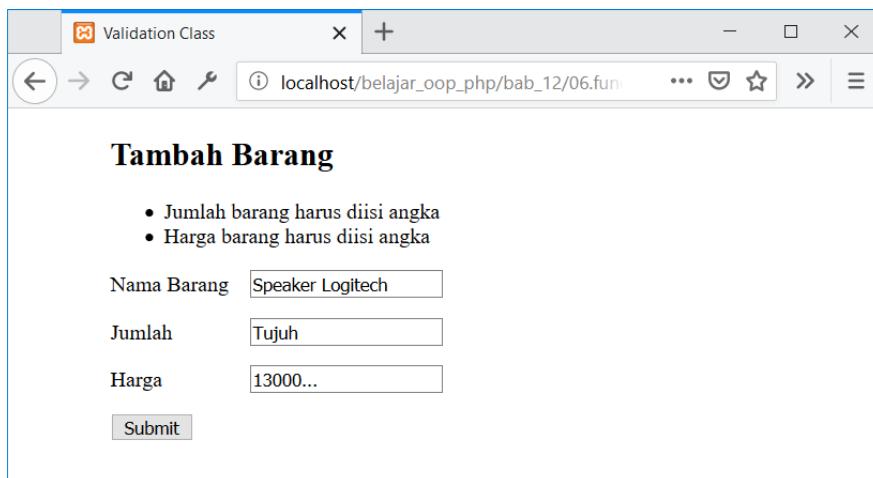
```
1  case 'numeric' :  
2      if ($ruleValue === TRUE && !is_numeric($formValue)) {  
3          $error[$item] = "$itemLabel harus diisi angka";  
4      }  
5  break;
```

Jika syarat validasi tertulis 'numeric' => true, **dan** jika fungsi `!is_numeric($formValue)` menghasilkan true (artinya `$formValue` bukan numerik), maka tambah pesan error "`$itemLabel harus diisi angka`".

Syarat validasi ini cocok untuk inputan yang harus berbentuk angka seperti jumlah barang dan harga barang. Berikut contoh pemanggilannya:

06.function_validate/11.numeric.php

```
1 validate('jumlah_barang','Jumlah barang', [
2     'required' => true,
3     'numeric' => true,
4 ]);
5
6 validate('harga_barang','Harga barang', [
7     'required' => true,
8     'numeric' => true,
9 ]);
```



Gambar: Error karena yang diinput bukan angka

Dalam contoh di atas, tampil pesan error karena saya mengisi teks 'Tujuh' dan '13000...' ke dalam inputan form jumlah dan harga, dimana kedua inputan ini memiliki syarat validasi **numeric**.

Validate: min_value dan max_value

Validasi berikutnya adalah `min_value` dan `max_value`. Ini dipakai untuk membatasi angka inputan agar tidak boleh kurang atau lebih dari nilai tertentu.

Sebagai contoh, inputan jumlah dan harga barang tidak boleh diisi angka negatif atau 0, karena memang tidak masuk akal jika sebuah barang memiliki harga negatif. Batasan inilah yang kita buat dengan syarat `min_value` dan `max_value`.

Berikut kode program yang diperlukan:

06.function_validate/12.min_max_value.php

```
1 case 'min_value' :
2     if ($formValue < $ruleValue) {
3         $error[$item] = "$itemLabel minimal $ruleValue";
4     }
5     break;
6
7 case 'max_value' :
```

```
8     if ($formValue > $ruleValue) {  
9         $error[$item] = "$itemLabel maksimal $ruleValue";  
10    }  
11    break;
```

Isi kedua validasi hanya membandingkan `$formValue` dan `$ruleValue` apakah lebih kecil atau lebih besar, kemudian tampilkan pesan error yang sesuai.

Berikut contoh pemanggilan syarat validasi ini:

06.function_validate/12.min_max_value.php

```
1 validate('jumlah_barang','Jumlah barang', [  
2     'required' => true,  
3     'numeric' => true,  
4     'min_value' => 0,  
5     'max_value' => 110,  
6 ]);
```

Penulisan di atas artinya saya ingin form `jumlah_barang`:

1. Harus diisi (tidak boleh kosong).
2. Harus diisi angka (numeric).
3. Nilai tidak boleh kurang dari 0.
4. Nilai tidak boleh lebih dari 0.

Proses validasi akan diproses secara berurutan dari atas ke bawah.

The screenshot shows a browser window titled "Validation Class". The address bar says "localhost/belajar_oop_php/bab_12/06.fun". The main content area has a title "Tambah Barang". Below it, there is a list of validation errors: "• Jumlah barang maksimal 110". There are three input fields: "Nama Barang" with value "Sepeda Wimcycle", "Jumlah" with value "111", and "Harga" with value "2000000". A "Submit" button is at the bottom. The "Jumlah" field is highlighted in red, indicating it is the current focus or has an error.

Gambar: Error karena nilai melebihi syarat validasi

Hasil di atas menampilkan pesan error karena saya mengisi angka 111 ke dalam inputan jumlah barang, dimana syarat maksimal hanya 110.

Validate + Sanitize

Setelah membuat beberapa syarat validasi dasar, kita akan menambahkan kembali proses sanitizing ke dalam fungsi `validate()`. Caranya adalah dengan menulis syarat sanitize ke associative array argument ketiga pemanggilan fungsi `validate()`. Berikut contoh

pemanggilannya:

```

1 validate('nama_barang','Nama barang', [
2   'sanitize' => 'string',
3   'required' => true,
4   'min_char' => 5,
5 ]);
```

Di baris 2 terdapat syarat 'sanitize' => 'string', yang berarti saya ingin inputan `nama_barang` harus melewati proses sanitize terlebih dahulu.

Untuk memprosesnya di dalam fungsi `validate()`, saya putuskan untuk tidak menempatkan kode sanitize ke dalam block switch-case, tapi sebelum itu. Tujuannya agar sanitize dilakukan terlebih dahulu sebelum masuk ke proses validasi.

Karena jika validasi dilakukan sebelum sanitizing, ada kemungkinan nilai menjadi string kosong. Misal, user menginput nama barang sebagai '`<script></script>`', ini akan lolos validasi karena sudah lebih 5 karakter, namun di dalam sanitizing semua teks akan terhapus sebab tidak boleh ada tag HTML.

Untuk memproses sanitizing, saya akan pakai fungsi `array_key_exists()` bawaan PHP. Fungsi `array_key_exists()` berguna untuk memeriksa apakah sebuah key ada di dalam array atau tidak. Hasilnya **true** jika ada, atau **false** jika tidak ada. Ini akan memudahkan proses pemeriksaan isi associative array:

06.function_validate/13.sanitize.php

```

1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8   function validate($item,$itemLabel,$rules) {
9     $formValue = Input::get($item);
10
11   // jalankan proses sanitize (jika disyaratkan)
12   if (array_key_exists('sanitize',$rules)) {
13     $formValue = Input::runSanitize($formValue,$rules['sanitize']);
14   }
15
16   global $error;
17
18   //.... lanjutan fungsi validate()
```

Tambahan proses sanitizing ada di baris 12-14. Kondisi `if (array_key_exists('sanitize', $rules))` di pakai untuk memeriksa apakah di dalam array `$rules` terdapat key bernama 'sanitize'.

Jika ada, maka ambil nilai dari key tersebut menggunakan kode `$rules['sanitize']`, dan input ke dalam method `Input::runSanitize()`. Hasil proses sanitize menjadi nilai baru dari `$formValue`.

Artinya, yang akan di validasi adalah hasil yang sudah di sanitize. Berikut pesan error dengan tambahan syarat `'sanitize' => 'string'` ke dalam inputan nama barang:

The screenshot shows a browser window with the title 'Validation Class'. The address bar says 'localhost/belajar_oop_php/bab_12/06.fun'. The main content is a form titled 'Tambah Barang'. It has three fields: 'Nama Barang' with value '<p></p>', 'Jumlah' with value '12', and 'Harga' with value '500'. Above the form, there is an error message: '• Nama barang tidak boleh kosong'. At the bottom is a 'Submit' button.

Gambar: Error karena inputan nama barang dianggap kosong

Saya mengisi nama barang dengan teks '`<p></p>`', namun hasilnya tetap dianggap kosong. Ini membuktikan proses sanitizing sudah berjalan sehingga teks '`<p></p>`' akan dihapus dan inputan form menjadi kosong.

Kita juga bisa memakai sanitizing untuk inputan jumlah barang:

```
1 validate('jumlah_barang', 'Jumlah barang', [
2     'sanitize' => 'int',
3     'required' => true,
4     'numeric' => true,
5     'min_value' => 0,
6     'max_value' => 110,
7 ]);
```

Sekarang inputan `jumlah_barang` akan melewati proses sanitizing dengan tipe data `int` terlebih dahulu.

Namun ada sedikit pertimbangan. Jika ditulis seperti ini, maka jumlah barang otomatis akan "dibersihkan". Apabila diinput teks '5a4', maka inputan form akan di-sanitizing menjadi 54.

Saya merasa ini kurang pas karena kita juga sudah memiliki validasi `numeric`. Jika ditambah syarat `'sanitize' => 'int'`, maka validasi numeric tidak berefek lagi. Semua inputan akan di bersihkan oleh proses sanitizing sebelum sampai ke proses validasi numeric.

Sampai di sini kita harus memutuskan apakah memilih sanitizing atau validation. Kali ini saya lebih memilih untuk menampilkan pesan error validation saja. Artinya, jika diinput teks '5a4' akan tampil pesan error. Ini semata-mata pilihan pribadi, anda boleh jika lebih memilih proses

sanitizing.

Repopulate

Setelah melewati proses sanitizing, kita juga harus merevisi kembali cara re-populate inputan form. Saya ingin jika ada yang menginput teks '<p>speaker</p>', inputan form akan menjadi 'speaker' tanpa ada tambahan tag <p>. Maksudnya, proses re-populate form harus berasal dari hasil sanitizing.

Terdapat 2 cara yang bisa kita lakukan. Pertama, langsung memanggil method `Input::runSanitize()` dari dalam atribut `value`. Jika sebelumnya isi atribut `value` adalah sebagai berikut:

```
<input type="text" name="nama_barang"
value="<?php echo Input::get('nama_barang') ?>">
```

Maka bisa diubah menjadi:

```
<input type="text" name="nama_barang"
value="<?php echo Input::runSanitize(Input::get('nama_barang'), 'string') ?>">
```

Sekarang setelah user mengisi inputan form dan men-klik tombol submit, proses repopulate akan melewati method `Input::runSanitize()` terlebih dahulu. Namun cara ini kurang fleksibel karena tidak semua inputan harus di sanitizing.

Cara kedua adalah mengambil hasil proses validasi dan menjadikannya sebagai nilai untuk atribut `value`. Ini bisa dilakukan dengan membuat fungsi `validate()` mengembalikan nilai inputan form yang sudah di sanitizing.

Jika sebelumnya kita menjalankan fungsi `validate()` sebagai berikut:

```
1 validate('nama_barang', 'Nama barang', [
2   'sanitize' => 'string',
3   'required' => true,
4   'min_char' => 5,
5 ]);
```

Maka sekarang saya ingin mengubahnya menjadi seperti ini:

```
1 $nama_barang = validate('nama_barang', 'Nama barang', [
2   'sanitize' => 'string',
3   'required' => true,
4   'min_char' => 5,
5 ]);
```

Sekarang terdapat tambahan variabel `$nama_barang` di baris 1. Variabel `$nama_barang` akan menampung nilai inputan form yang sudah melewati proses sanitizing di dalam fungsi `validate()`.

Agar fungsi validate() bisa mengembalikan nilai, cukup tambah perintah `return $formValue` di akhir fungsi:

06.function_validate/14.repopulate.php

```
1 function validate($item,$itemLabel,$rules) {
2     // seluruh proses sanitizing dan validation ada di sini..
3     // ...
4
5     return $formValue;
6 }
```

Perintah `return $formValue` di baris 5 akan mengembalikan nilai yang tersimpan di variabel `$formValue`. Variabel `$formValue` sendiri berasal dari inputan form yang sudah di-sanitizing. Dengan demikian, nilai untuk atribut `value` bisa ditulis sebagai berikut:

```
<input type="text" name="nama_barang"
value="<?php if (isset($nama_barang)) { echo $nama_barang; } ?>">
```

Tambahan kondisi `if (isset($nama_barang))` diperlukan karena jika form tampil pertama kali, variabel `$nama_barang` belum terdefinisi dan baru ada saat form telah di submit.

Berikut kode program lengkap dari fungsi validasi kita sejauh ini:

```
1 <?php
2 require 'Input.php';
3
4 $error = [];
5
6 if (!empty($_POST)) {
7
8     function validate($item,$itemLabel,$rules) {
9         $formValue = Input::get($item);
10
11        // jalankan proses sanitize untuk setiap item (jika disyaratkan)
12        if (array_key_exists('sanitize',$rules)) {
13            $formValue = Input::runSanitize($formValue,$rules['sanitize']);
14        }
15
16        global $error;
17
18        foreach ($rules as $rule => $ruleValue) {
19
20            switch($rule) {
21
22                case 'required':
23                    if ($ruleValue === TRUE && empty($formValue)) {
24                        $error[$item] = "$itemLabel tidak boleh kosong";
25                    }
26                    break;
27
28                case 'min_char' :
29                    if (strlen($formValue) < $ruleValue) {
```

Case Study: Validate Class

```
30         $error[$item] = "$itemLabel minimal $ruleValue karakter";
31     }
32     break;
33
34     case 'max_char' :
35         if (strlen($formValue) > $ruleValue) {
36             $error[$item] = "$itemLabel maksimal $ruleValue karakter";
37         }
38     break;
39
40     case 'numeric' :
41         if ($ruleValue === TRUE && !is_numeric($formValue)) {
42             $error[$item] = "$itemLabel harus diisi angka";
43         }
44     break;
45
46     case 'min_value' :
47         if ($formValue < $ruleValue) {
48             $error[$item] = "$itemLabel minimal $ruleValue";
49         }
50     break;
51
52     case 'max_value' :
53         if ($formValue > $ruleValue) {
54             $error[$item] = "$itemLabel maksimal $ruleValue";
55         }
56     break;
57
58 }
59
60 // cek jika sudah ada error di item yang sama, Langsung keluar dari looping
61 if (!empty($error[$item])) {
62     break;
63 }
64
65 }
66 // kembalikan nilai yang sudah Lewat proses sanitize
67 return $formValue;
68 }
69
70 $nama_barang = validate('nama_barang', 'Nama barang', [
71     'sanitize' => 'string',
72     'required' => true,
73     'min_char' => 5,
74 ]);
75
76 $jumlah_barang = validate('jumlah_barang', 'Jumlah barang', [
77     'required' => true,
78     'numeric' => true,
79     'min_value' => 0,
80     'max_value' => 110,
81 ]);
82
83 $harga_barang = validate('harga_barang', 'Harga barang', [
84     'required' => true,
```

Case Study: Validate Class

```
85      'numeric' => true,
86      'min_value' => 0,
87  ]);
88
89 }
90
91 ?>
92 <!doctype html>
93 <html lang="id">
94   <head>
95     <meta charset="utf-8">
96     <title>Validation Class</title>
97   </head>
98   <style>
99     .container {
100       margin: 20px auto;
101       width: 500px;
102     }
103     form > div {
104       margin: 15px 0;
105     }
106     label {
107       display:inline-block;
108       width:100px;
109     }
110   </style>
111   <body>
112     <div class="container">
113       <h2>Tambah Barang</h2>
114       <div class="pesan-error">
115         <ul>
116           <?php
117             foreach ($error as $value) {
118               echo "<li>$value</li>";
119             }
120           ?>
121         </ul>
122       </div>
123       <form method="post">
124         <div>
125           <label for="nama_barang">Nama Barang</label>
126           <input type="text" name="nama_barang"
127             value="<?php if (isset($nama_barang)) { echo $nama_barang; } ?>">
128         </div>
129         <div>
130           <label for="jumlah_barang">Jumlah</label>
131           <input type="text" name="jumlah_barang"
132             value="<?php if (isset($jumlah_barang)) { echo $jumlah_barang; } ?>">
133         </div>
134         <div>
135           <label for="harga_barang">Harga</label>
136           <input type="text" name="harga_barang"
137             value="<?php if (isset($harga_barang)) { echo $harga_barang; } ?>">
138         </div>
139         <div>
```

```

140      <input type="submit" value="Submit">
141    </div>
142  </form>
143</div>
144</body>
145</html>
```

Silahkan pelajari sebentar kode program di atas, dan saya sangat sarankan untuk coba input berbagai nilai yang akan menampilkan pesan error validasi. Dalam bahasan berikutnya kita akan "angkat" seluruh kode di fungsi validate() ke dalam class **Validate**.

12.6. Validate Class

Secara struktur, fungsi validate() sudah menjadi kode program yang fleksibel. Kita bisa dengan mudah membuat syarat validasi baru hanya dengan menambah kondisi case .

Namun agar lebih *portable* dan mudah dipakai untuk project-project lain (serta agar sesuai dengan materi buku ini), kita akan memindahkannya ke sebuah class yang saya beri nama **Validate class**.

Sebelum itu, berikut cara pemanggilan class Validate dari dalam form:

```

07.class_validate/index.php

1  <?php
2  require 'Input.php';
3  require 'Validate.php';
4
5  $error = [];
6
7  if (!empty($_POST)) {
8
9    $validate = new Validate($_POST);
10
11   $nama_barang = $validate->setRules('nama_barang', 'Nama barang', [
12     'sanitize' => 'string',
13     'required' => true,
14     'min_char' => 5,
15   ]);
16
17   $jumlah_barang = $validate->setRules('jumlah_barang', 'Jumlah barang', [
18     'required' => true,
19     'numeric' => true,
20     'min_value' => 0,
21     'max_value' => 110,
22   ]);
23
24   $harga_barang = $validate->setRules('harga_barang', 'Harga barang', [
25     'required' => true,
26     'numeric' => true,
27     'min_value' => 0,
```

```

28     ]);
29
30 }
31
32 ?>
33 <!doctype html>
34 <html lang="id">
35 ...

```

Di baris 3 terdapat perintah untuk meng-import file `Validate.php` ke dalam kode program kita saat ini. Nantinya, di dalam file `Validate.php` inilah pendefinisian class `Validate` berada.

Di baris 9 terdapat perintah untuk membuat object dari class `Validate`, yang disimpan ke dalam variabel `$validate`. Class `Validate` juga butuh sebuah argument berupa variabel yang berisi inputan form. Karena form kita menggunakan `method = 'post'`, maka saya mengirim global variabel `$_POST` sebagai argument.

Jika nanti terdapat form lain yang menggunakan `method = 'get'`, proses pembuatan class `Validate` juga bisa memakai `new Validate($_GET)`.

Selanjutnya terdapat pembuatan syarat validasi untuk setiap inputan form. Ini mirip seperti pemanggilan fungsi `validate()` yang sudah kita pelajari sebelumnya. Hanya saja sekarang diakses dari method `$validate->setRules()`. Dengan demikian, di dalam class `Validate` kita akan membuat sebuah method `setRules()` untuk memproses seluruh syarat validasi.

Method `setRules()` sendiri menerima 3 argument yang sama persis seperti di fungsi `validate()`, yakni nama inputan form, string yang akan dipakai untuk membuat pesan error, serta associative array yang berisi syarat validasi. Method `setRules()` juga mengembalikan nilai, berupa hasil inputan form yang sudah di *sanitizing*. Kembali, ini sama persis seperti fungsi `validate()`.

Baik, kita akan masuk ke pembuatan class `Validate()`. Pertama, buat file baru `Validate.php` lalu tulis kode program berikut:

```

07.class_validate/Validate.php

1  <?php
2  class Validate{
3      private $_errors = array();
4      private $_formMethod = null;
5
6      public function __construct($formMethod){
7          $this->_formMethod = $formMethod;
8      }
9
10     public function setRules($item, $itemLabel, $rules){
11         }
12
13 }

```

Di awal class `Validate` saya membuat 2 buah *private property*, yakni `$_errors` dan `$_formMethod`. Property `$_errors` nantinya dipakai untuk menampung semua pesan error yang berasal dari proses validasi. Sedangkan property `$_formMethod` dipakai untuk menampung global variabel yang dipakai untuk mengirim form, nilainya salah satu dari variabel `$_POST` atau `$_GET`.

Di baris 6 – 8 terdapat constructor dari class `Validate`. Constructor ini butuh sebuah parameter `$formMethod`. Variabel `$formMethod` nantinya akan berisi salah satu dari global variable `$_POST` atau `$_GET`. Parameter `$formMethod` ini kemudian disimpan kembali ke dalam private property `$this->_formMethod`.

Dengan demikian, jika class `Validate` di instansiasi dengan perintah `new Validate($_POST)`, maka property `$this->_formMethod` akan berisi `$_POST`.

Di baris 10 terdapat method `setRules()`. Method inilah yang akan memproses semua proses sanitizing dan validasi form. Seperti yang akan kita lihat, kode program untuk method ini berasal dari fungsi `validate()` yang sudah kita bahas sebelumnya.

Berikut isi dari method `setRules()`:

07.class_validate/Validate.php

```

1  <?php
2  // ...
3  public function setRules($item, $itemLabel, $rules){
4      $formValue = trim($this->_formMethod[$item]);
5
6      // jalankan proses sanitize untuk setiap item (jika disyaratkan)
7      if (array_key_exists('sanitize', $rules)) {
8          $formValue = Input::runSanitize($formValue, $rules['sanitize']);
9      }
10
11     foreach ($rules as $rule => $ruleValue) {
12
13         switch($rule) {
14
15             case 'required':
16                 if ($ruleValue === TRUE && empty($formValue)) {
17                     $this->_errors[$item] = "$itemLabel tidak boleh kosong";
18                 }
19                 break;
20
21             case 'min_char' :
22                 if (strlen($formValue) < $ruleValue) {
23                     $this->_errors[$item] = "$itemLabel minimal $ruleValue karakter";
24                 }
25                 break;
26
27             case 'max_char' :
28                 if (strlen($formValue) > $ruleValue) {
29                     $this->_errors[$item] = "$itemLabel maksimal $ruleValue karakter";

```

```

30         }
31     break;
32
33     case 'numeric' :
34         if ($ruleValue === TRUE && !is_numeric($formValue)) {
35             $this->_errors[$item] = "$itemLabel harus diisi angka";
36         }
37     break;
38
39     case 'min_value' :
40         if ($formValue < $ruleValue) {
41             $this->_errors[$item] = "$itemLabel minimal $ruleValue";
42         }
43     break;
44
45     case 'max_value' :
46         if ($formValue > $ruleValue) {
47             $this->_errors[$item] = "$itemLabel maksimal $ruleValue";
48         }
49     break;
50 }
51
52 // cek jika sudah ada error di item yang sama, Langsung keluar dari looping
53 if (!empty($this->_errors[$item])) {
54     break;
55 }
56
57 }
58 // kembalikan nilai yang sudah Lewat proses sanitize
59 return $formValue;
60 }
61
62 }

```

Di baris 4 saya mengisi variabel \$formValue dari hasil trim(\$this->_formMethod[\$item]). Tapi tunggu dulu, kenapa kita tidak menggunakan method Input::get() seperti di fungsi validate()?

Jika anda masih ingat, property \$_formMethod tidak lain berisi global array \$_POST. Sehingga untuk mengambil nilai form yang diinput user, kita bisa akses langsung dari \$_formMethod ini. Sebagai contoh, jika inputan yang diperiksa adalah nama_barang, maka \$this->_formMethod[\$item] akan di proses menjadi \$_POST['nama_barang'].

Selanjutnya sisa kode program sama persis seperti fungsi validate(), hanya saja sekarang pesan error akan ditampung ke dalam property \$this->_errors.

Untuk membuktikan tidak ada masalah, silahkan akses form dan coba input beberapa nilai. Ketika men-klik tombol submit, ternyata tidak tampil pesan error apapun. Alasannya karena pesan error masih berada dalam private property \$_error. Kita harus rancang sebuah method untuk mengambil pesan ini dari class Validate.

12.7. Validate Class: Method getError

Untuk bisa menampilkan pesan error dari class Validate ke dalam form, kita akan buat satu method khusus bernama `getError()`. Method `getError()` sendiri sangat sederhana, berisi sebuah getter untuk property `$_getError`:

08.class_validate_getError/Validate.php

```

1  <?php
2  class Validate{
3
4      //...isi class validate
5
6      public function getError(){
7          return $this->_errors;
8      }
9
10     public function passed(){
11         return empty($this->_errors) ? true : false;
12     }
13
14 }
```

Selain membuat method `getError()`, di baris 10 juga terdapat method lain, yakni `passed()`. Method ini akan mengembalikan boolean `true` atau `false` tergantung apakah property `$_error` berisi sesuatu atau tidak.

Di baris 11 saya menggunakan operator ternary PHP, yakni `? :`. Ini merupakan penulisan singkat dari:

```

1  <?php
2  if (empty($this->_errors)) {
3      return true;
4  }
5  else {
6      return false;
7 }
```

Method `passed()` bisa dijadikan acuan apakah form sudah lolos validasi atau tidak.

Kembali ke halaman form, saya akan mengakses kedua method ini dengan cara berikut:

08.class_validate_getError/index.php

```

1  <?php
2  require 'Input.php';
3  require 'Validate.php';
4
5  $error = [];
6
7  if (!empty($_POST)) {
8
9      $validate = new Validate($_POST);
```

Case Study: Validate Class

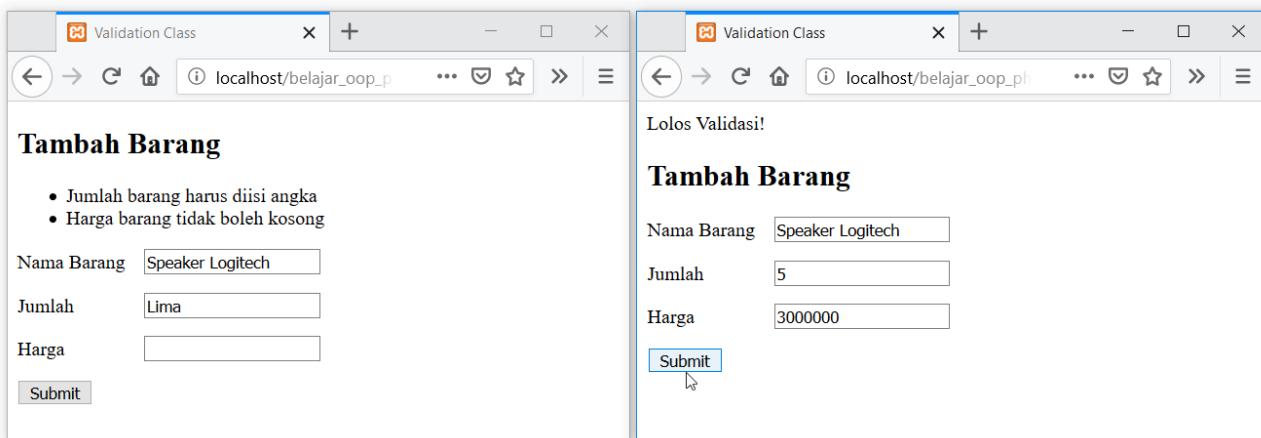
```
10
11     $nama_barang = $validate->setRules('nama_barang', 'Nama barang', [
12         //...
13     ]);
14
15     if($validate->passed()) {
16         echo "Lolos Validasi!";
17     } else {
18         $error = $validate->getError();
19     }
20
21 }
22
23 ?>
24 <!doctype html>
25 <html lang="id">
26 ...
```

Tambahan kode program ada di baris 15 – 19.

Pertama, kondisi `if($validate->passed())` di pakai untuk memeriksa apakah semua syarat validasi sudah terpenuhi atau tidak. Method `$validate->passed()` sendiri mengembalikan nilai `true` jika tidak ada pesan error. Jika hasilnya `true`, artinya semua syarat validasi sudah terpenuhi dan tampilan teks **Lolos Validasi!**.

Namun jika ternyata method `$validate->passed()` menghasilkan nilai `false`, ambil array pesan error menggunakan method `$validate->getError()`, dan isi ke dalam variabel `$error`.

Selanjutnya variabel `$error` ditampilkan oleh perulangan `foreach` yang sudah kita rancang pada saat membahas fungsi `validate()` sebelumnya. Berikut hasil kode program kita sejauh ini:



Gambar: Validasi tidak lolos (kiri) dan validasi lolos (kanan)

Silahkan anda coba input beberapa nilai ke dalam form dan lihat apakah semua syarat validasi bisa ditampilkan.

12.8. Mempersiapkan Form Tambah User

Untuk pembahasan kali ini (dan juga materi selanjutnya), saya akan membuat form baru, yakni **form tambah user**. Alasannya agar bisa mengakomodasi syarat validasi lain seperti memastikan inputan password yang harus sama dengan inputan ulangi password, validasi alamat email, serta validasi menggunakan *regular expression*.

Studi kasus dengan form register ini akan membuat class Validate menjadi lebih lengkap.

Baik, berikut kode program yang dipakai untuk membuat form register user:

09.user_form/index.php

```

1  <?php
2  require 'Input.php';
3  require 'Validate.php';
4
5  $error = [];
6
7  if (!empty($_POST)) {
8
9      $validate = new Validate($_POST);
10
11     $username = $validate->setRules('username', 'Username', [
12         'sanitize' => 'string',
13         'required' => true,
14         'min_char' => 4,
15     ]);
16
17     $password = $validate->setRules('password', 'Password', [
18         'sanitize' => 'string',
19         'required' => true,
20         'min_char' => 6,
21     ]);
22
23     $ulangi_password = $validate->setRules('ulangi_password', 'Ulangi password', [
24         'sanitize' => 'string',
25         'required' => true,
26         'min_char' => 6,
27     ]);
28
29     $email = $validate->setRules('email', 'Email', [
30         'sanitize' => 'string',
31         'required' => true,
32     ]);
33
34
35     if($validate->passed()) {
36         echo "Lolos Validasi!";
37     } else {
38         $error = $validate->getError();
39     }
40

```

Case Study: Validate Class

```
41 }
42 ?>
43 <!doctype html>
44 <html lang="id">
45   <head>
46     <meta charset="utf-8">
47     <title>Validation Class</title>
48   </head>
49   <style>
50     .container {
51       margin: 20px auto;
52       width: 500px;
53     }
54     form > div {
55       margin: 15px 0;
56     }
57     label {
58       display:inline-block;
59       width:130px;
60     }
61   </style>
62 <body>
63   <div class="container">
64     <h2>Tambah User</h2>
65     <div class="pesan-error">
66       <ul>
67         <?php
68           foreach ($error as $value) {
69             echo "<li>$value</li>";
70           }
71         ?>
72       </ul>
73     </div>
74     <form method="post">
75       <div>
76         <label for="username">Username</label>
77         <input type="text" name="username"
78           value="<?php if (isset($username)) { echo $username; } ?>">
79       </div>
80       <div>
81         <label for="password">Password</label>
82         <input type="password" name="password">
83       </div>
84       <div>
85         <label for="ulangi_password">Ulangi Password</label>
86         <input type="password" name="ulangi_password">
87       </div>
88       <div>
89         <label for="email">Email</label>
90         <input type="text" name="email"
91           value="<?php if (isset($email)) { echo $email; } ?>">
92       </div>
93       <div>
94         <input type="submit" value="Submit">
```

```
96      </div>
97      </form>
98  </div>
99  </body>
100 </html>
```

Silahkan pelajari sebentar kode program di atas. Kita tetap memakai class **Input** dan **Validate** seperti sebelumnya. Yang membedakan adalah, sekarang terdapat 4 buah inputan form: **username**, **password**, **ulangi_password**, dan **email**.

Batasan validasi sendiri juga menggunakan syarat yang sudah kita pelajari sejauh ini, diantaranya '**sanitize**', '**required**', dan '**min_char**'.

The screenshot shows a browser window with the title 'Validation Class'. The address bar displays 'localhost/belajar_oop_php/bab_12/09.user_form'. The main content is a form titled 'Tambah User'. It lists validation rules:

- Username minimal 4 karakter
- Password tidak boleh kosong
- Ulangi password tidak boleh kosong
- Email tidak boleh kosong

Below the rules are four input fields: 'Username' (value 'joe'), 'Password' (empty), 'Ulangi Password' (empty), and 'Email' (empty). A 'Submit' button is at the bottom.

Gambar: Tampilan pesan error dan isian form tambah user

12.9. Validasi Form Tambah User

Untuk form tambah user, kita akan rancang syarat validasi bary. Penambahan ini bisa dilakukan dengan menulis kondisi case baru ke dalam method **Validate:: setRules()**. Kurang lebih sama seperti yang kita lakukan selama ini.

Validate: matches

Validate **matches** dipakai untuk memeriksa apakah nilai dari sebuah inputan form sudah sama dengan nilai inputan form lain.

Validasi ini cocok dipakai untuk proses register yang butuh 2 kali inputan password. Setelah user mengisi password, dia diminta untuk menulisnya sekali lagi untuk menghindari salah ketik. Jika ternyata kedua inputan ini tidak cocok, maka tampilkan pesan error.

Dalam form kita, inputan password dan ulangi password ini sudah tersedia:

10.user_form_match_password/input.php

```
1 <div>
2   <label for="password">Password</label>
3   <input type="password" name="password">
4 </div>
5 <div>
6   <label for="ulangi_password">Ulangi Password</label>
7 </div>
```

Dan berikut penulisan syarat validasi untuk kedua inputan:

10.user_form_match_password/input.php

```
1 $password = $validate->setRules('password', 'Password', [
2   'sanitize' => 'string',
3   'required' => true,
4   'min_char' => 6,
5 ]);
6
7 $ulangi_password = $validate->setRules('ulangi_password', 'Ulangi password', [
8   'sanitize' => 'string',
9   'required' => true,
10  'min_char' => 6,
11  'matches' => 'password'
12 ]);
```

Perhatikan di baris 11, saya menambah sebuah syarat baru ke dalam inputan `ulangi_password`, berupa `'matches' => 'password'`. Artinya saya ingin inputan ini nilainya di cocok-kan dengan inputan `password`.

Berikut syarat kondisi case untuk membuat validasi `matches` di dalam method `setRules()` class `Validate`:

10.user_form_match_password/Validate.php

```
1 case 'matches' :
2 if ($formValue != $this->_formMethod[$ruleValue]) {
3   $this->_errors[$item] = "$itemLabel tidak sama";
4 }
5 break;
```

Semoga anda masih ingat isi variabel `$formValue`, `$this->_formMethod` dan `$ruleValue` di dalam class `Validate`. Di baris 2 saya membandingkan isi variabel `$formValue` yang berasal dari inputan user dengan nilai syarat validasi.

Pada contoh kita saat ini, kondisi pemeriksaan akan diproses sebagai berikut:

```
1 case 'matches' :
2 if ($_POST['ulangi_password'] != $_POST['password']) {
3   $this->_errors['ulangi_password'] = "Ulangi password tidak sama";
4 }
5 break;
```

Hasilnya, akan tampil pesan error jika inputan password dan ulangi_password tidak sama:

Gambar: Tampilan pesan error karena ulangi password tidak sama

Karena alasan keamanan, inputan password dan ulangi_password tidak perlu di re-populate. User akan diminta untuk mengisi kembali password dan ulangi_password jika form tidak lolos validasi.

Validate: email

Validasi berikutnya adalah untuk memeriksa format email. Yakni apakah isian email sudah sesuai dengan format sebuah email atau tidak. Sebagai contoh, inputan email seharusnya memiliki karakter '@', serta punya alamat domain, seperti @yahoo.com, @gmail.com, dll.

Di buku PHP Uncover, saya menggunakan *regular expression* sederhana untuk memeriksa pola inputan email. Namun kali ini kita akan memakai validasi email khusus bawaan PHP dari fungsi `filter_var()`.

Jika anda masih ingat, `filter_var()` adalah fungsi yang juga kita pakai untuk membuat proses sanitizing. Yup, `filter_var()` adalah sebuah fungsi 'gado-gado' yang bisa dipakai untuk proses sanitizing dan validasi sekaligus. Jenis konstanta parameter-lah yang akan mengatur apa yang harus di lakukan oleh fungsi `filter_var()`.

Untuk membuat proses validasi email, gunakan konstanta `FILTER_VALIDATE_EMAIL` sebagai argument kedua fungsi `filter_var()`. Hasilnya berupa boolean `false` jika format email tidak sesuai, atau mengembalikan string email tersebut jika format sesuai.

Berikut beberapa percobaan fungsi `filter_var()` untuk memeriksa inputan string email:

11.user_form_email/filter_var_func.php

```
1 <?php  
2  
3 $result = filter_var('test@', FILTER_VALIDATE_EMAIL);
```

Case Study: Validate Class

```
4 var_dump($result); // bool(false)
5
6 $result = filter_var('test@co', FILTER_VALIDATE_EMAIL);
7 var_dump($result); // bool(false)
8
9 $result = filter_var('test@co.id$$', FILTER_VALIDATE_EMAIL);
10 var_dump($result); // bool(false)
11
12 $result = filter_var('test@co.id', FILTER_VALIDATE_EMAIL);
13 var_dump($result); // test@co.id
14
15 $result = filter_var('test_4j4@hoho.id', FILTER_VALIDATE_EMAIL);
16 var_dump($result); // test_4j4@hoho.id
17
18 $result = filter_var('a@a.a', FILTER_VALIDATE_EMAIL);
19 var_dump($result); // a@a.a
```

Percobaan di baris 3, 6, dan 9 semuanya mengembalikan nilai **false**. Artinya string 'test@', 'test@co', dan 'test@co.id\$\$' tidak valid untuk sebuah alamat email. Sedangkan string 'test@co.id', 'test_4j4@hoho.id' dan 'a@a.a' lolos sebagai sebuah alamat email. Fungsi inilah yang akan kita pakai untuk proses validasi alamat email di dalam class **Validate**.

Sebelum itu, berikut potongan cara pembuatan syarat validasi:

11.user_form_email/index1.php

```
1 $email = $validate->setRules('email', 'Email', [
2     'sanitize' => 'string',
3     'required' => true,
4     'email' => true,
5 ]);
```

Di baris 5 saya menulis syarat 'email' => true. Ini artinya saya ingin inputan form di cek polanya agar memenuhi kriteria sebuah alamat email. Dan berikut kode program **case** validasi di dalam method **setRules()**:

11.user_form_email/Validate.php

```
1 case 'email' :
2     if ($ruleValue === TRUE && !filter_var($formValue, FILTER_VALIDATE_EMAIL)) {
3         $this->_errors[$item] = "Format $itemLabel tidak sesuai";
4     }
5     break;
```

Di sini saya memakai tanda negasi '!' untuk membalik hasil fungsi **filter_var()**. Sekarang jika pola tidak sesuai akan menghasilkan **true** dan input pesan error ke dalam array **\$this->_errors**.

Berikut percobaannya:

The screenshot shows a web browser window titled "Validation Class". The address bar shows the URL "localhost/belajar_oop_php/bab_12/11.user_for...". The main content is a form titled "Tambah User". Below the title, there is an error message: "• Format Email tidak sesuai". The form has four input fields: "Username" (value: Rissa), "Password" (empty), "Ulangi Password" (empty), and "Email" (value: rissa@gmail). At the bottom of the form is a blue "Submit" button.

Gambar: Tampilan pesan error karena format email tidak sesuai

Hasilnya tampil pesan error karena teks 'rissa@gmail' belum memenuhi syarat dari sebuah alamat email.

Sebagai tambahan, HTML5 sebenarnya sudah memiliki validasi email bawaan dari tag `<input type="email">`. Jika ditulis seperti ini, web browser akan melakukan proses validasi di web browser terlebih dahulu, tanpa perlu dikirim ke PHP. Ini mirip seperti validasi dengan JavaScript.

Akan tetapi validasi di web browser bisa dilewati dengan mudah, misalnya dengan memakai **Inspect Element** dan menukar inputan form menjadi `<input type="text">`. Validasi form tetap harus ditulis di sisi server. Jadikan validasi di sisi client sebagai pelengkap agar lebih *user friendly*.

Validate: url

Karena kita baru saja membahas penggunaan fungsi `filter_var()`, saya ingin membuat 1 lagi validasi menggunakan fungsi ini, yakni validasi alamat **URL** (Uniform Resource Locator).

Tidak jarang di dalam form terdapat inputan yang meminta alamat web, dan fungsi `filter_var()` menyediakan konstanta `FILTER_VALIDATE_URL` untuk men-validasi alamat web. Berikut percobaannya:

12.user_form_url/filter_var_func.php

```
1 <?php
2
3 $result = filter_var('duniailkom', FILTER_VALIDATE_URL);
4 var_dump($result); // bool(false)
5
6 $result = filter_var('duniailkom.com', FILTER_VALIDATE_URL);
```

Case Study: Validate Class

```
7 var_dump($result); // bool(false)
8
9 $result = filter_var('www.duniailkom.com', FILTER_VALIDATE_URL);
10 var_dump($result); // bool(false)
11
12 $result = filter_var('http://duniailkom.com', FILTER_VALIDATE_URL);
13 var_dump($result); // http://duniailkom.com
14
15 $result = filter_var('https://www.duniailkom.com', FILTER_VALIDATE_URL);
16 var_dump($result); // https://www.duniailkom.com
17
18 $result = filter_var('https://www.duniailkom.com?s=php&u=admin',
19 FILTER_VALIDATE_URL);
20 var_dump($result); // 'https://www.duniailkom.com?s=php&u=admin'
```

Terlihat bahwa syarat dari `filter_var()` cukup ketat. Misalnya string 'www.duniailkom.com' dianggap tidak memenuhi syarat sebagai alamat URL, padahal penulisan seperti ini sering kita pakai.

Konstanta `FILTER_VALIDATE_URL` mewajibkan alamat URL memiliki awalan protocol seperti '`http://`' atau '`https://`'. Jadi yang cocok adalah '`https://www.duniailkom.com`', bukan '`www.duniailkom.com`'. Anda bisa mempertimbangkan hal ini sebelum membuat proses validasi dengan fungsi `filter_var()`.

Namun kali ini saya akan menerima syarat tersebut dan membuat tambahan kondisi validasi dengan kode sebagai berikut:

12.user_form_url/Validate.php

```
1 case 'url' :
2     if ($ruleValue === TRUE && !filter_var($formValue, FILTER_VALIDATE_URL)) {
3         $this->_errors[$item] = "Format $itemLabel tidak sesuai";
4     }
5 break;
```

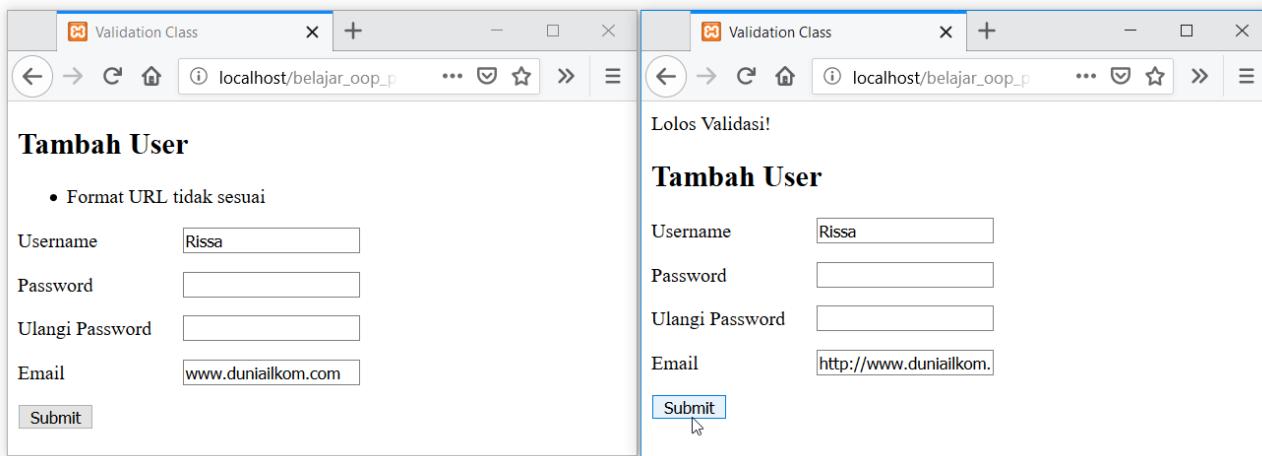
Teknik yang dipakai sama seperti validasi email, cukup tukar kondisi di baris 1 menjadi `case 'url'` dan memakai konstanta `FILTER_VALIDATE_URL` untuk fungsi `filter_var()`.

Sebagai percobaan, saya akan pakai inputan email dengan syarat validasi sebagai berikut:

12.user_form_url/index1.php

```
1 $email = $validate->setRules('email', 'Email', [
2     'sanitize' => 'string',
3     'required' => true,
4     'url' => true,
5 ]);
```

Case Study: Validate Class



Gambar: Tampilan pesan error karena format URL tidak sesuai (kiri), dan lolos validasi (kanan)

Sama seperti email, HTML5 juga sudah memiliki validasi URL bawaan dengan tag `<input type="url">`.

Validate: regexp

Untuk validasi dengan pola yang agak rumit, solusi paling ampuh adalah memakai **regular expression** atau sering disingkat sebagai **regexp** atau **RE**.

Regular expression sangat powerful untuk membuat pemeriksaan pola, tentunya selama kita bisa merangkai pola yang kadang terasa cukup kompleks. Di buku duniaikom sebelumnya saya cukup sering membahas tentang regexp seperti di PHP Uncover, JavaScript Uncover, dan MySQL Uncover. Hampir setiap bahasa pemrograman modern memiliki fitur regexp.

Dengan regexp, kita bisa rancang syarat pola yang agak rumit seperti 'username hanya boleh berisi karakter huruf dan angka saja', atau 'password harus memiliki minimal 1 huruf besar dan 1 karakter angka'.

Untuk memeriksa apakah sebuah string cocok dengan pola regular expression tertentu, PHP menyediakan fungsi `preg_match()`. Berikut contoh penggunaannya:

13.user_form_regexp/preg_match.php

```
1 <?php
2
3 $result = preg_match("/^([A-Za-z]{6,})$/", "aNto");
4 var_dump($result); // int(0)
5
6 $result = preg_match("/^([A-Za-z]{6,})$/", "aNto99");
7 var_dump($result); // int(0)
8
9 $result = preg_match("/^([A-Za-z]{6,})$/", "aNtoni");
10 var_dump($result); // int(1)
11
```

```
12 $result = preg_match("/^A-Za-z]{6,}$/, "Budiansyah");
13 var_dump($result); // int(1)
```

Fungsi `preg_match` mengembalikan angka integer 1 jika string cocok dengan pola, atau 0 jika string tidak cocok dengan pola. Angka 1 dan 0 akan dikonversi menjadi boolean `true` atau `false` ketika dipakai dalam kondisi if.

Dalam contoh di atas, pola regexp `/^A-Za-z]{6,}$/` akan cocok dengan string yang terdiri dari huruf besar atau kecil dan minimal 6 karakter. Di baris 3 string 'aNto' tidak cocok karena hanya terdiri dari 4 karakter, sedangkan di baris 6 string 'aNto99' juga tidak cocok karena mengandung angka.

Untuk form daftar user, saya ingin username memiliki syarat seperti ini, dimana sebuah username hanya bisa diisi huruf saja dan minimal 6 angka. Berikut cara pembuatan syarat validasi tersebut:

13.user_form_regexp/index.php

```
1 $username = $validate->setRules('username', 'Username', [
2   'sanitize' => 'string',
3   'required' => true,
4   'min_char' => 4,
5   'regexp' => "/^A-Za-z]{6,}$/,
6 ]);
```

Tambahan syarat ada di baris 5 untuk pencocokan inputan form dengan pola regexp `"/^A-Za-z]{6,}$/"`. Dan tentu saja kita harus menambah sebuah kondisi `case` ke dalam method `setRules()`:

13.user_form_regexp/Validate.php

```
1 case 'regexp' :
2   if (!preg_match($ruleValue,$formValue)) {
3     $this->_errors[$item] = "Pola $itemLabel tidak sesuai";
4   }
5 break;
```

Di sini saya membuat kondisi `if (!preg_match($ruleValue,$formValue))`, tanda negasi ! dipakai untuk membalik hasil fungsi `preg_match()`. Jika pola string tidak sesuai, hasilnya menjadi `true` dan tambah pesan error ke dalam array `$_error`.

The screenshot shows a web browser window with the title 'Validation Class'. The address bar displays 'localhost/belajar_oop_php/bab_12/13.user_form'. The main content is a form titled 'Tambah User' with the following fields:

- Username: Rissa
- Password: (empty)
- Ulangi Password: (empty)
- Email: rissa@gmail.com

An error message is shown above the form:

- Pola Username tidak sesuai

A 'Submit' button is located at the bottom of the form.

Gambar: Tampilan pesan error karena pola username tidak sesuai

Contoh di atas tampil error karena username 'Rissa' tidak sampai 6 karakter.

Jika diperhatikan kembali, inputan username sebenarnya juga sudah memiliki syarat pembatasan karakter, yakni '`min_char`' => 4. Namun pembatasan ini akan tertimpa oleh syarat pola dari regexp yang harus minimal 6 karakter. Anda bebas apakah ingin menghapus syarat '`min_char`', mengubah isi dari pola regexp, atau tetap memakai keduanya.

12.10. Validasi Form ke Database

Sebagian besar dari kita sudah sering melakukan proses register di sebuah website. Saat mendaftar, terdapat 1 prinsip validasi yang selalu ada, yakni kita tidak bisa memilih username yang sudah didaftarkan oleh user lain. Validasi seperti ini harus melibatkan database karena data username ada di sana.

Dan inilah yang akan kita buat untuk class `Validate`, berupa sebuah mekanisme untuk memeriksa inputan username apakah sudah ada atau belum di dalam database. Jika sudah ada, tampilkan pesan error dan minta user memilih nama lain.

Untungnya, kita sudah menyiapkan ini di dalam class `DB` melalui method `check()`. Sekedar pengingat, berikut definisi method `check()` di dalam class `DB`:

```
1 public function check($tableName, $columnName, $dataValues){  
2     $query = "SELECT {$columnName} FROM {$tableName} WHERE {$columnName} = ? ";  
3     return $this->runQuery($query,[ $dataValues])->rowCount();  
4 }
```

Method `check()` butuh 3 argument: **nama tabel**, **nama kolom** dan **data** yang akan di cari. Method `check()` kemudian mengembalikan jumlah baris yang memenuhi kriteria tersebut, misalnya 1, 2 atau 3. Namun jika data yang dicari tidak ketemu, hasilnya berupa angka 0.

Sebelum merancang kode validasi, kita akan buat tabel **user** di database **ilkoom** terlebih

dahulu. Caranya, silahkan akses file `14.user_form_uniq/generate_tabel_user.php`. Tabel user sebenarnya pernah kita rancang di latihan terakhir bab class DB.

Tabel user memiliki 3 kolom: `username`, `password` dan `email`. Dengan mengakses file di atas, tabel user juga sudah memiliki 1 username bernama 'budianto'. Inilah yang akan menjadi bahan praktik kita

Saya ingin proses pemeriksaan ini tetap masuk ke syarat validasi, tepatnya ke dalam method `setRules()` milik class `Validate`. Berikut cara pemanggilan untuk inputan `username`:

`14.user_form_uniq/index1.php`

```
1 $username = $validate->setRules('username', 'Username', [
2     'sanitize' => 'string',
3     'required' => true,
4     'min_char' => 4,
5     'regexp' => "/^A-Za-z{4,}$/,
6     'unique' => ['user', 'username'],
7 ]);
```

Tambahan validasi ada di baris 6 berupa '`unique`' => `['user', 'username']`. Syarat ini bisa dibaca: Inputan `username` harus unik dengan memeriksa semua data yang ada di kolom 'user' dalam tabel 'username'.

Sebagai contoh lain, jika ditulis '`unique`' => `['nim', 'mahasiswa']`, artinya inputan harus unik dengan memeriksa kolom `nim` di tabel `mahasiswa`.

Berikut kode program untuk membuat validasi di method `setRules()`:

```
1 case 'unique' :
2     require_once 'DB.php';
3     $DB = DB::getInstance();
4     if($DB->check($ruleValue[0],$ruleValue[1],$formValue)){
5         $this->_errors[$item] = "$itemLabel sudah terpakai, silahkan pilih
6                               nama lain";
7     }
8     break;
```

Karena kita butuh class `DB`, maka di baris 2 terdapat perintah untuk proses import file `DB.php`. Saya memakai perintah `require_once` untuk menghindari error jika ternyata di bagian kode program lain sudah ada proses import untuk file `DB.php`. Jika hanya memakai `require` saja, ada kemungkinan bentrok dengan kode lain yang juga meng-import file `DB.php`.

Kemudian di baris 3 terdapat perintah untuk proses instansiasi class `DB`. Hasilnya, variabel `$DB` bertindak sebagai object dari class `DB`.

Di baris 4 saya membuat kondisi `if` dengan hasil pemanggilan method `$DB->check()`. Seperti yang sudah dibahas sebelumnya, method `check()` butuh 3 buah argument. Argument pertama berupa **nama tabel** yang tersimpan di dalam `$ruleValue[0]`. Tambahan angka `[0]` di sini

merujuk ke element pertama dari array \$ruleValue.

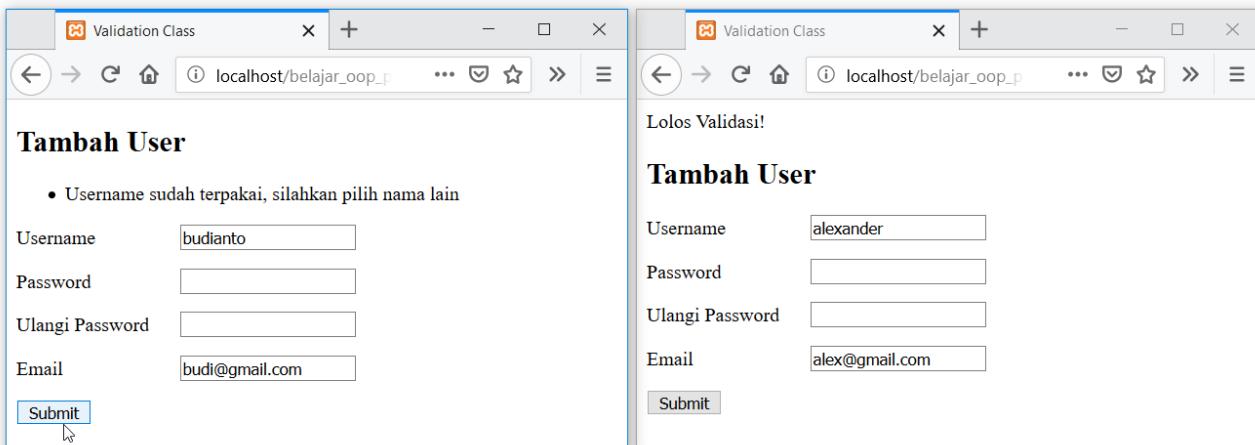
Argument kedua berisi **nama kolom**, yang tersimpan di dalam \$ruleValue[1]. Dan sebagai argument ketiga adalah **hasil inputan form** yang tersimpan di dalam variabel \$formValue.

Sebagai contoh, jika di inputan username saya mengisi 'budianto', maka kondisi di baris 4 akan berbentuk:

```
if($DB->check('username', 'user', 'budianto'))
```

Method \$DB->check() mengembalikan nilai integer positif jika terdapat data di dalam database. Ini kemudian dikonversi menjadi boolean **true** dan input pesan error ke variabel \$_error.

Mari kita coba dengan mengisi username 'budianto' ke dalam inputan form:



Gambar: Tampilan pesan error karena username sudah ada di database (kiri) dan yang belum ada (kanan)

Sip, pesan error validasi sukses tampil.

12.11. Proses Input ke Database

Menyambung pembahasan kita tentang validasi database, saya juga ingin lanjut ke proses insert. Setelah semua syarat validasi terpenuhi, maka data sudah bisa diinput ke database. Menggunakan class DB, ini cukup mudah diproses, cukup jalankan method DB::insert() dengan data yang berasal dari hasil validasi.

Pertama, kita harus import file DB.php di halaman form:

15.user_form_insert/index.php

```
1 <?php
2 require 'Input.php';
3 require 'Validate.php';
4 require 'DB.php';
```

Setelah penulisan syarat validasi, tempatkan kode berikut:

```
15.user_form_insert/index.php

1 if($validate->passed()) {
2
3     // masukkan data user baru ke dalam database
4     $DB = DB::getInstance();
5     $newUser = [
6         'username' => $username,
7         'password' => $password,
8         'email' => $email
9     ];
10    $DB->insert('user', $newUser);
11    echo "Data sudah ditambahkan ke Database!";
12
13 } else {
14     $error = $validate->getError();
15 }
```

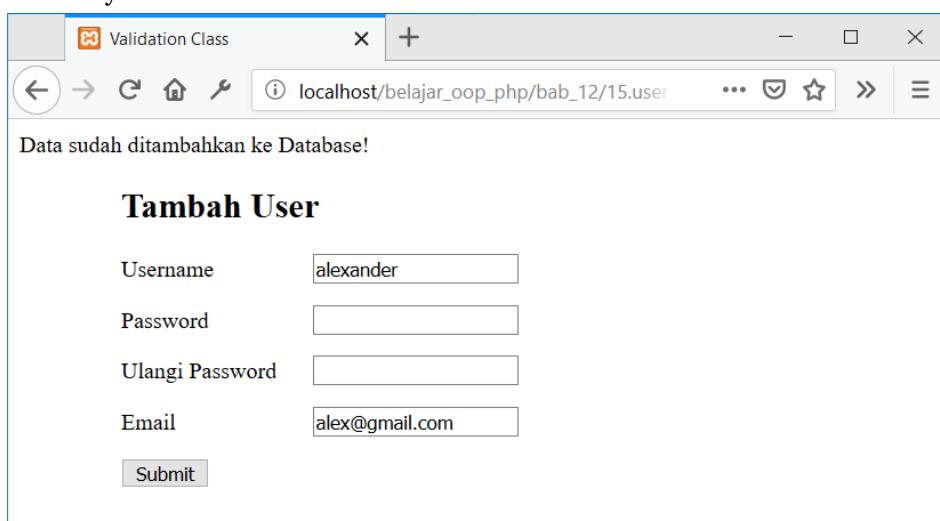
Artinya, jika validasi lolos, yakni saat pemanggilan method `$validate->passed()` menghasilkan nilai true, maka proses seluruh blok kode program antara baris 4 – 11.

Di baris 4 terdapat proses instansiasi class `DB` ke dalam variabel `$DB`. Kemudian di baris 5 saya membuat array `$newUser` yang berisi associative array. Jika anda masih ingat, method `insert()` dari class `DB` butuh associative array untuk semua inputan kolom.

Variabel `$username`, `$password` dan `$email`, semuanya berasal dari hasil fungsi `setRules()`, sehingga bisa langsung kita pakai.

Proses input sendiri dilakukan pada baris 10 dengan mengakses method `$DB->insert('user', $newUser)`. Yang dilanjutkan dengan menampilkan teks 'Data sudah ditambahkan ke Database!', sekedar informasi untuk user bahwa data sudah berhasil masuk ke database.

Berikut percobaannya:



Gambar: Data 'alexander' berhasil diinput ke database

Dalam prakteknya nanti, setelah proses input berhasil kita bisa *redirect* user ke halaman lain. Tapi untuk saat ini tampilan teks di atas sudah mencukupi.

Anda bisa cek isi tabel user dari cmd MySQL Client atau dari phpmyadmin.

	<input type="checkbox"/>	<input type="text"/> Edit	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	username	password	email
	<input type="checkbox"/>	<input type="text"/> Edit	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	alexander	123123	alex@gmail.com
	<input type="checkbox"/>	<input type="text"/> Edit	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	budianto	rahasia	rudianto@gmail.com

Gambar: Data 'alexander' sudah ada di tabel user (dilihat dari phpmyadmin)

Khusus untuk inputan password, saya masih menyimpan data 'mentah'. Sebaiknya teks password di-*hash* terlebih dahulu. Materi ini akan kita bahas dalam bab selanjutnya karena fokus utama kita sekarang hanya proses validasi saja.

12.12. Validasi Checkbox

Mayoritas inputan form memang berupa *textbox*, yakni tag `<input type="text">`. Tapi HTML masih menyediakan bentuk inputan form lain seperti **checkbox**, **radio**, dan **select**. Proses validasi yang sudah kita rancang sebenarnya bisa langsung dipakai untuk jenis inputan ini, karena pada dasarnya yang sampai ke server adalah hasil dari atribut `value`.

Selain itu inputan *checkbox*, *radio*, dan *select* sudah memiliki nilai tetap. User tidak bisa mengubah nilai selain yang sudah disediakan, sehingga telah membantu dalam proses validasi. Namun sebagai pelengkap tidak ada salahnya kita bahas cara validasi inputan form ini, dimulai dari *checkbox*.

Sebagai bahan praktek, saya akan buat form baru dengan kode berikut:

16.checkbox_basic/01.checkbox.php

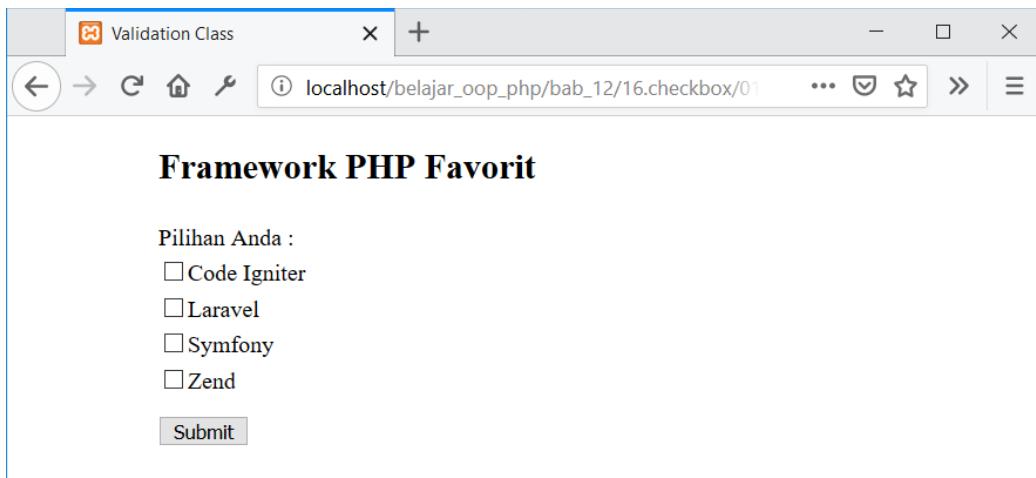
```

1 <form method="post">
2   <div>
3     <label>Pilihan Anda :</label>
4     <div>
5       <label>
6         <input type="checkbox" name="cb_code_igniter"
7           value="Code Igniter">Code Igniter</label>
8       </div>
9       <div>
10      <label><input type="checkbox" name="cb_laravel"
11        value="Laravel">Laravel</label>
12      </div>
13      <div>
14        <label><input type="checkbox" name="cb_symfony"
15          value="Symfony">Symfony</label>
16      </div>

```

Case Study: Validate Class

```
17 <div>
18   <label><input type="checkbox" name="cb_zend"
19     value="Zend">Zend</label>
20 </div>
21 </div>
22 <div>
23   <input type="submit" value="Submit">
24 </div>
25 </form>
```



Gambar: Form pilihan framework

Form ini terdiri dari 4 pilihan checkbox. User bisa memilih satu atau beberapa framework PHP favorit. Masing-masing checkbox memiliki atribut `name= 'cb_code_igniter'`, `name= 'cb_laravel'`, `name= 'cb_symfony'`, dan `name= 'cb Zend'`. Awalan nama 'cb' merupakan singkatan dari `checkbox`, namun ini hanya sekedar nama dan bukan sebuah keharusan.

Validasi pertama kita adalah, saya ingin agar checkbox 'Code Igniter' wajib dipilih. Jika tidak, tampilkan pesan error. Untuk membuatnya kita bisa memakai syarat validasi 'required':

16.checkbox_basic/01.checkbox.php

```
1 <?php
2 require 'Input.php';
3 require 'Validate.php';
4
5 $error = [];
6
7 if (!empty($_POST)) {
8
9   $validate = new Validate($_POST);
10
11  $cb_code_igniter = $validate->setRules('cb_code_igniter', 'Code Igniter', [
12    'required' => true,
13  ]);
14
15  print_r($_POST);
16
```

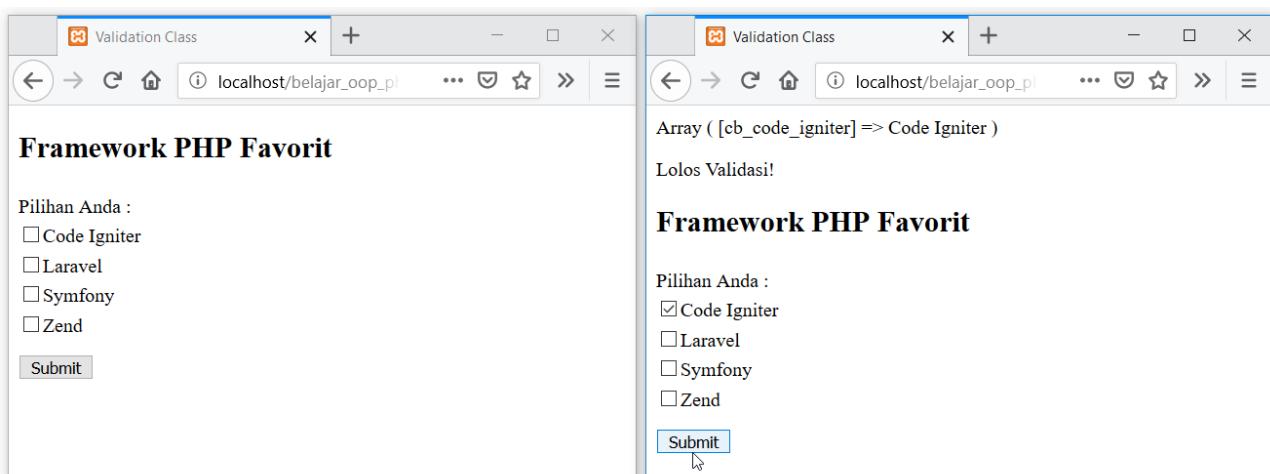
```
17 if($validate->passed()) {  
18     echo "<p> Lolos Validasi!</p>";  
19 } else {  
20     $error = $validate->getError();  
21 }  
22  
23 }
```

Proses validasi yang dipakai kurang lebih sama seperti sebelumnya. Di baris 9 saya membuat object dari class `Validate`, kemudian di baris 11 – 13 terdapat syarat '`required`' => `true` untuk inputan '`cb_code_igniter`'. Artinya inputan ini wajib dipilih.

Tambahan perintah `print_r($_POST)` di baris 15 dipakai untuk memeriksa isi `$_POST`, sekedar memastikan inputan form sudah terkirim.

Baik, mari kita coba jalankan form dan langsung klik tombol submit tanpa memilih inputan apapun. Hasilnya? Ternyata tidak tampil pesan apa-apa!

Lalu jalankan ulang, pilih checkbox '`Code Igniter`' dan submit. Berikut hasilnya:



Gambar: Form di-submit langsung (kiri) dan form di-submit dengan memilih 'Code Igniter'

Pertanyaannya, kenapa validasi '`required`' tidak aktif? Harusnya ketika inputan '`Code Igniter`' tidak dipilih, akan tampil pesan error.

Masalah ini terjadi karena form tidak lolos pemeriksaan kondisi `if(!empty($_POST))` di baris 7.

Selama ini, kondisi `if (!empty($_POST))` tidak bermasalah karena begitu form di submit, HTML akan mengirim nilai inputan textbox meskipun tidak diisi. Maksudnya, jika di dalam sebuah form terdapat tag `<input type="text">`, maka setiap kali form di submit, nilainya tetap sampai ke global variabel `$_POST` meskipun hanya berisi string kosong ''.

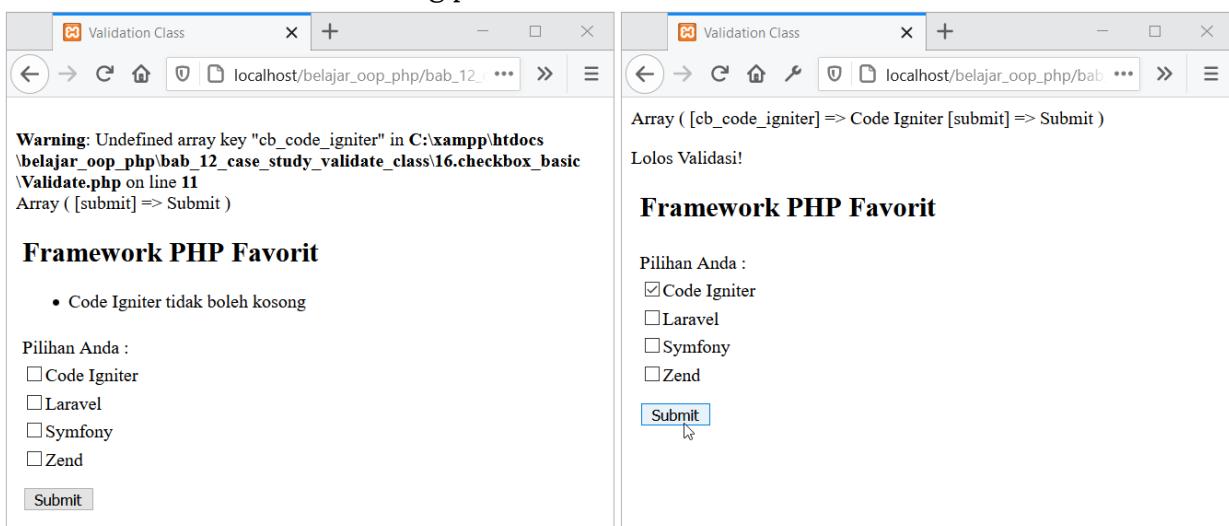
Namun lain halnya dengan tag `<input type="checkbox">`. Jika inputan checkbox tidak dipilih, tidak ada nilai yang dikirim ke `$_POST`. Jika di dalam form hanya terdiri dari checkbox saja dan pada saat proses submit checkbox ini tidak dipilih, maka `$_POST` tidak berisi apa-apa. Inilah yang mengakibatkan kondisi `if (!empty($_POST))` menghasilkan **false**.

Untuk mengatasinya, kita butuh suatu cara agar jika checkbox tidak dipilih, validasi tetap berjalan. Trik sederhana adalah dengan menambah inputan `<input type="hidden" name="dummy">` ke dalam form untuk memaksa `$_POST` berisi suatu nilai. Tag `<input type="hidden">` pada dasarnya mirip seperti `<input type="text">`, hanya saja tidak terlihat di tampilan form.

Atau alternatif lain, tambahkan atribut `name` ke dalam tombol submit:

```
16.checkbox_basic/02.checkbox_submit.php
<input type="submit" value="Submit" name="submit">
```

Sekarang setiap kali tombol submit di tekan, akan selalu dikirim nilai variabel `$_POST['submit']`, hasilnya `if (!empty($_POST))` akan menghasilkan nilai `true`. Dengan tambahan ini, mari kita coba ulang proses submit:



Gambar: Form di-submit langsung (kiri) dan form di-submit dengan memilih 'Code Igniter'

Sekarang ketika form di submit tanpa memilih 'Code Igniter', akan tampil pesan error 'Code Igniter tidak boleh kosong'. Artinya proses validasi sudah berjalan.

Di bagian atas juga tampil teks `Array ([submit] => Submit)`. Ini berasal dari perintah `var_dump($_POST)`, dimana submit merupakan nilai dari tombol `<input type="submit">` yang baru saja kita tambahkan.

Tapi sekarang muncul masalah baru, yakni pesan error `Warning: Undefined array key "cb_code_igniter"` dari class `Validate`. Error ini masih disebabkan karena nilai inputan `<input type="checkbox">` yang tidak terkirim.

Selama ini, kita hanya membuat proses validasi untuk inputan `<input type="text">`. Seperti yang dibahas sebelumnya, meskipun tidak diisi, inputan textbox akan selalu dikirim ke server. Namun tidak dengan `<input type="checkbox">`.

Ketika checkbox `cb_code_igniter` tidak dipilih (tidak di-check), maka variabel

`$_POST['cb_code_igniter']` tidak terdefinisi di dalam class `Validate`. Inilah yang membuat tampil pesan error `Warning: Undefined array key "cb_code_igniter"`.

Sebagai solusi, saya akan modifikasi kode program untuk method `setRules()` di dalam class `Validate`. Sebelumnya proses pengambilan nilai isian form dilakukan dengan kode berikut:

```
$formValue = $this->_formMethod[$item];
```

Sekarang perlu sedikit modifikasi:

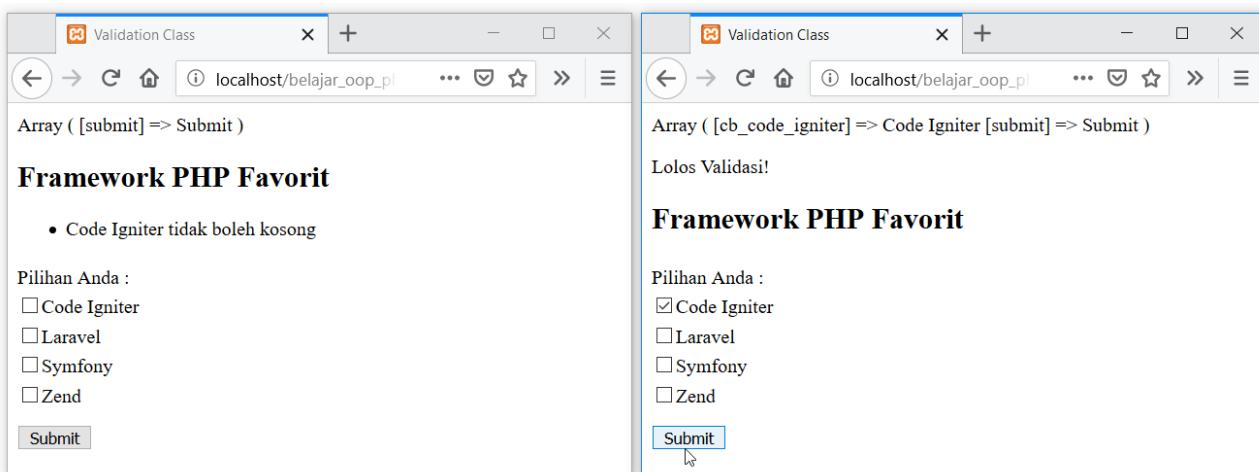
17.checkbox_validate/Validate.php

```
1 if (isset($this->_formMethod[$item])) {  
2     $formValue = trim($this->_formMethod[$item]);  
3 } else {  
4     $formValue = "";  
5 }
```

Dengan kode ini, isi array `$_POST` akan di periksa terlebih dahulu sebelum diambil dengan method `Input::get()`.

Dalam contoh kita, jika `$_POST['cb_code_igniter']` terdefinisi, baru ambil nilainya dan simpan ke dalam `$formValue`. Namun jika tidak terdefinisi, isi variabel `$formValue` dengan string kosong. Pemeriksaan kondisi dilakukan oleh kode program di baris 1, yakni `if (isset($this->_formMethod[$item]))`.

Berikut hasil dari modifikasi kode di atas:



Gambar: Form di-submit langsung (kiri) dan form di-submit dengan memilih 'Code Igniter'

Sip, proses validasi sudah berjalan sebagaimana mestinya. Sekarang jika inputan form tidak sampai ke class `Validate`, inputan itu akan diganti dengan string kosong ''.

Lanjut, kita akan bahas cara re-populate isian checkbox.

Re-populate Checkbox

Agar sebuah checkbox bisa ter-check atau terpilih, butuh tambahan atribut `checked` ke dalam tag tersebut. Sebagai contoh, agar checkbox 'Code Igniter' langsung terpilih begitu halaman di load, bisa memakai kode berikut:

```
<input type="checkbox" name="cb_code_igniter" value="Code Igniter" checked>
```

Tugas kita adalah, bagaimana mencari tau sebuah checkbox sudah terpilih atau tidak.

Ini bisa dilakukan dengan memeriksa apakah nama inputan tersebut ada di dalam global variabel `$_POST` atau tidak. Proses pemeriksaan dilakukan setelah validasi:

18.checkbox_re-populate/index.php

```
1 <?php
2 require 'Input.php';
3 require 'Validate.php';
4
5 $error = [];
6
7 if (!empty($_POST)) {
8     // kode untuk proses validasi di sini
9     // ....
10 }
11
12 $check_code_igniter = Input::get('cb_code_igniter')==='Code Igniter'?
13     'checked' : '';
14 $check_laravel = Input::get('cb_laravel')==='Laravel'? 'checked' : '';
15 $check_symfony = Input::get('cb_symfony')==='Symfony'? 'checked' : '';
16 $check_zend = Input::get('cb_zend')==='Zend'? 'checked' : '';
```

Di baris 12 saya membuat variabel `$check_code_igniter`. Isi variabel ini berasal dari hasil operator ternary. Apabila `Input::get('cb_code_igniter')` berisi teks 'Code Igniter', maka isi variabel `$select_code_igniter` dengan teks 'checked'. Namun jika kondisi tersebut tidak terpenuhi, maka isi string kosong ''.

Kode program di baris 12 merupakan penulisan singkat dari:

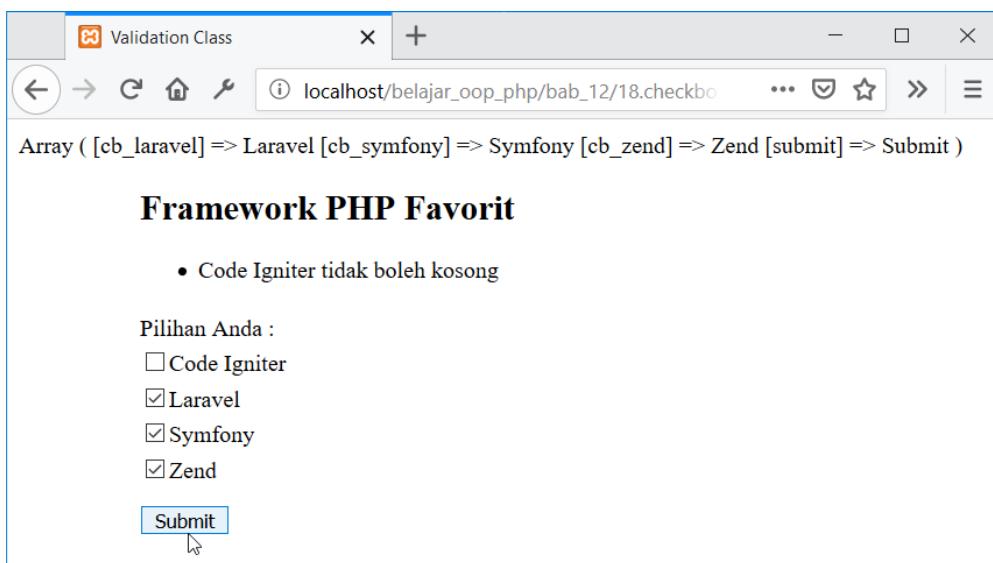
```
1 if (Input::get('cb_code_igniter')==='Code Igniter') {
2     $select_code_igniter = 'checked';
3 } else {
4     $select_code_igniter = '';
5 }
```

Hal yang sama juga perlu dibuat untuk inputan checkbox lain.

Dengan demikian, setiap variabel akan berisi string 'checked' atau string kosong '' tergantung apakah inputan tersebut di pilih atau tidak dari hasil form sebelumnya. Setelah penambahan ini, kita tinggal men-echo isi variabel di dalam setiap inputan checkbox:

18.checkbox_re-populate/index.php

```
1 <form method="post">
2   <div>
3     <label>Pilihan Anda :</label>
4     <div>
5       <label><input type="checkbox" name="cb_code_igniter" value="Code Igniter"
6           <?php echo $check_code_igniter; ?>Code Igniter</label>
7       </div>
8     <div>
9       <label><input type="checkbox" name="cb_laravel" value="Laravel"
10      <?php echo $check_laravel; ?>Laravel</label>
11    </div>
12    <div>
13      <label><input type="checkbox" name="cb_symfony" value="Symfony"
14          <?php echo $check_symfony; ?>Symfony</label>
15    </div>
16    <div>
17      <label><input type="checkbox" name="cb_zend" value="Zend"
18          <?php echo $check_zend; ?>Zend</label>
19    </div>
20  </div>
21  <div>
22    <input type="submit" value="Submit" name="submit">
23  </div>
24 </form>
```



Gambar: Form checkbox sudah menyimpan hasil dari pilihan sebelumnya.

Hasilnya, form re-populate sudah sukses berjalan.

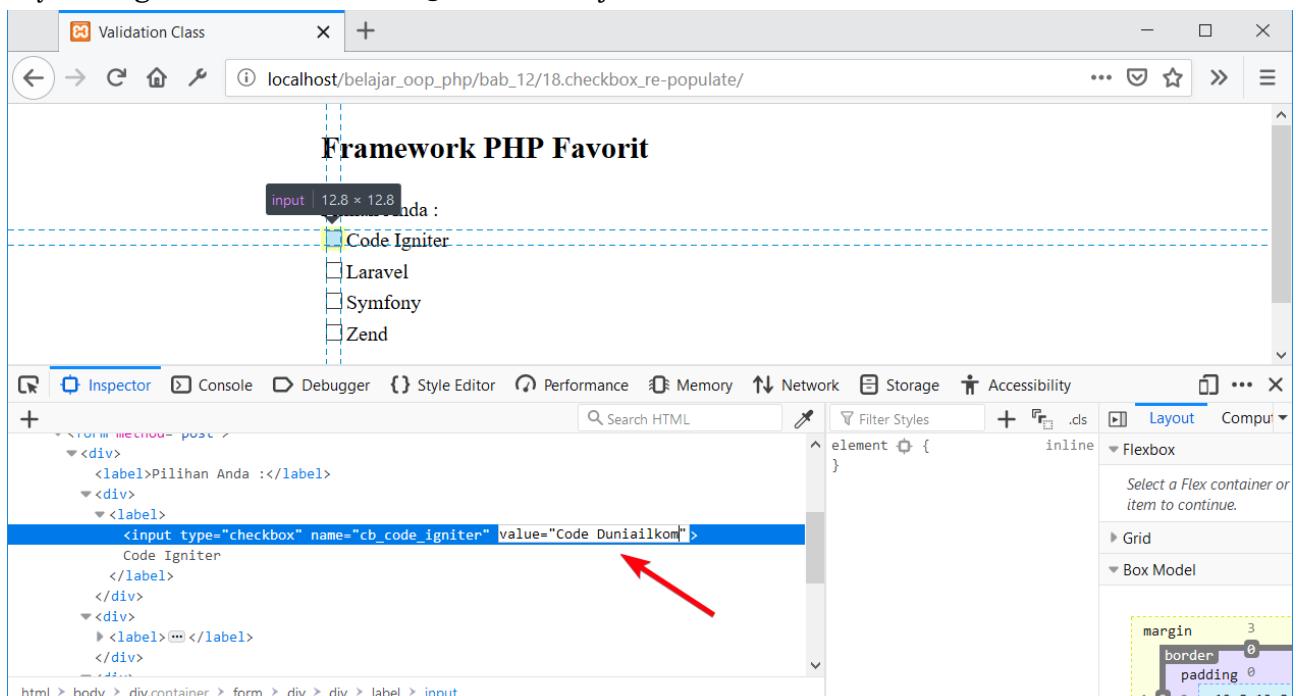
Validasi dengan Regular Expression

Di awal pembahasan tentang checkbox, saya sempat tulis bahwa kita tidak terlalu butuh proses validasi karena isi inputan checkbox tidak bisa diubah. User hanya bisa memilihnya atau tidak.

Well... kenyataannya, ini tidak sepenuhnya benar. Bagi user yang iseng atau berniat jahat, tetap bisa mengubah nilai inputan checkbox dengan nilai lain. Caranya, meng-edit langsung kode HTML yang ada di web browser. Trik ini bisa dilakukan dengan bantuan **Web Developer Tools** atau **Inspect Element** bawaan web browser.

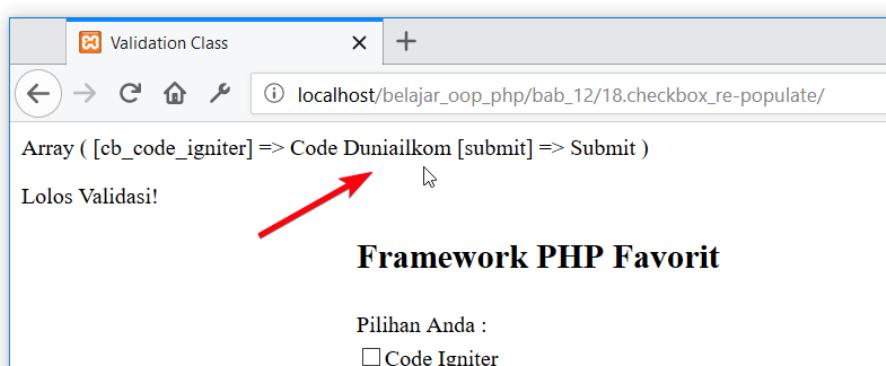
Sebagai praktek, tes jalankan file kita sebelumnya lalu buka Web Developer Tools dengan menekan kombinasi **Ctrl + Shift + I**. Kemudian cari inputan checkbox, dan double klik di bagian atribut **value**.

Cursor akan berubah dan kita bisa menukar nilai yang ada di atribut **value**. Dalam contoh ini, saya mengubah **value="Code Igniter"** menjadi **value="Code DuniaIlkom"**.



Gambar: Ubah nilai atribut value dari checkbox 'Code Igniter'

Setelah nilai atribut **value** diubah, tekan Enter dan tutup Web Developer Tools. Tanpa me-refresh halaman, langsung pilih checkbox 'Code Igniter' dan klik tombol Submit. Berikut hasilnya:



Gambar: Hasil checkbox 'Code Igniter' berubah jadi 'Code DuniaIlkom'

Dapat dilihat bahwa hasil dari `$_POST['cb_code_igniter']` sekarang menjadi '**Code DuniaIkom**'! Artinya saya telah menukar nilai dari inputan checkbox.

Pada dasarnya, semua inputan form bisa diubah dengan cara yang sama. Inilah alasan kenapa validasi di sisi web browser hanya sebagai pelengkap, kita tidak bisa mengandalkannya karena user bisa mengubah semua kode HTML yang ada di web browser. Validasi tetap harus di buat di sisi server menggunakan PHP.

Jadi, bagaimana cara memastikan nilai atribut value tidak bisa diubah oleh user? Jawabnya **tidak bisa**, karena seperti itulah web browser bekerja. Ketika sebuah halaman web di-load, semua file HTML, CSS dan JavaScript akan dikirim ke web browser. Apa yang terjadi di web browser sudah di luar jangkauan kita sebagai programmer PHP. User bisa mengubah kode yang ada di web browsernya sesuka hati. Yang bisa kita lakukan adalah memeriksa inputan form setelah form tersebut di kirim, yakni dari server menggunakan PHP.

Meskipun user bisa men-edit kode HTML, CSS dan JavaScript yang ada, perubahan itu hanya terjadi di web browser yang dia pakai. Kode asli di web server tidak terpengaruh.

Jadi sebagai langkah antisipasi, kita harus pastikan inputan checkbox harus sesuai dengan nilai yang ada di dalam atribut `value`. Untungnya, class `Validate` sudah menyediakan kondisi validasi tersebut, yakni menggunakan `regexp`. Kita bisa memanfaatkan validasi `regexp` dengan membuat pola yang hanya cocok dengan isi dari atribut `value`.

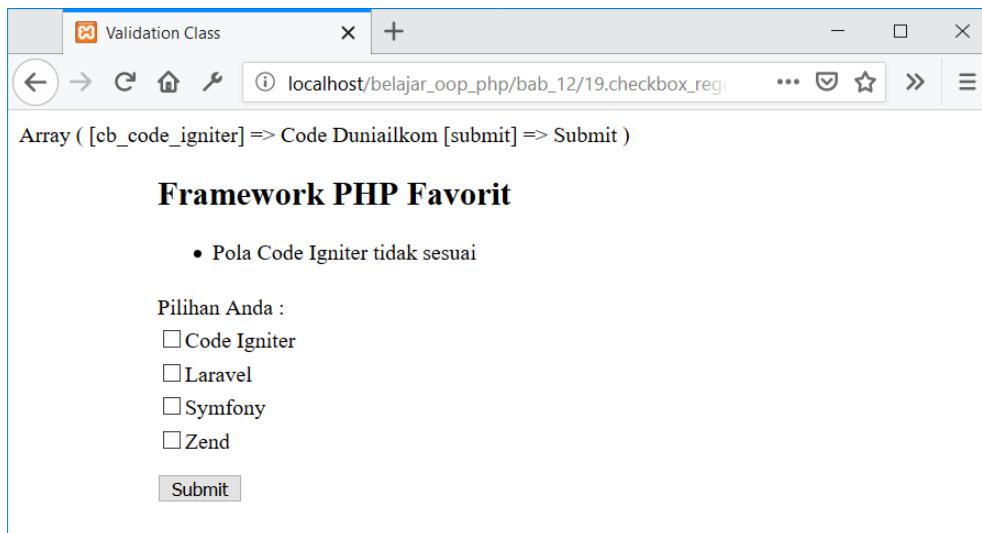
Sebagai contoh, untuk inputan checkbox 'Code Igniter' saya bisa membuat syarat tambahan sebagai berikut:

19.checkbox_regexp/input.php

```
1 $validate = new Validate($_POST);
2
3 $cb_code_igniter = $validate->setRules('cb_code_igniter', 'Code Igniter', [
4     'sanitize' => 'string',
5     'required' => true,
6     'regexp' => '/^Code Igniter$/',
7 ]);
```

Syarat '`regexp` => `'/^Code Igniter$/'`' di baris 6 artinya, inputan dari 'cb_code_igniter' harus berisi string 'Code Igniter', tidak bisa nilai lain.

Dengan tambahan ini, maka ketika ada yang mengubah nilai atribut `value` dari checkbox, akan tampil pesan error sebagai berikut:



Gambar: Tampil pesan error karena hasil inputan checkbox tidak sesuai

Di sini saya kembali menukar nilai atribut value 'Code Igniter' menjadi 'Code DuniaIlkom'. Syarat validasi regexp sekarang bisa menangkap perubahan ini dan menampilkan pesan error bahwa pola tidak sesuai.

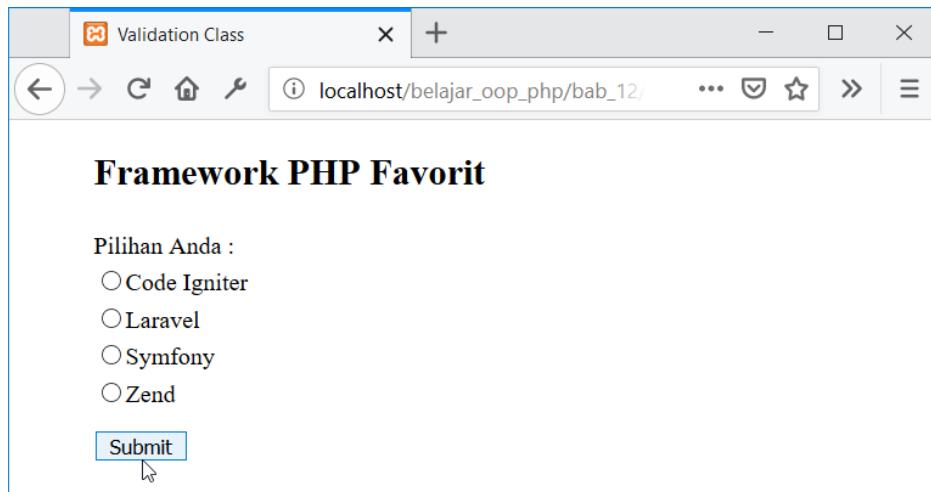
12.13. Validasi Radio

Dengan sedikit modifikasi, teknik yang kita pakai untuk checkbox juga berlaku di tag `<input type="radio">`. Berikut kode HTML untuk pembuatan form:

20.radio/index.php

```
1 <form method="post">
2   <div>
3     <label>Pilihan Anda :</label>
4     <div>
5       <label><input type="radio" name="rd_framework" value="Code Igniter"
6         <?php echo $check_code_igniter; ?>>Code Igniter</label>
7     </div>
8     <div>
9       <label><input type="radio" name="rd_framework" value="Laravel"
10      <?php echo $check_laravel; ?>>Laravel</label>
11    </div>
12    <div>
13      <label><input type="radio" name="rd_framework" value="Symfony"
14        <?php echo $check_symfony; ?>>Symfony</label>
15    </div>
16    <div>
17      <label><input type="radio" name="rd_framework" value="Zend"
18        <?php echo $check_zend; ?>>Zend</label>
19    </div>
20  </div>
21  <div>
22    <input type="submit" value="Submit" name="submit">
```

```
23    </div>
24 </form>
```



Gambar: Form dengan radio

Isian form radio ini sama seperti contoh praktik checkbox, yang berubah hanya nilai atribut `type` dan `name`. Khusus untuk `name`, ke-4 radio button memiliki nama yang sama, yakni '`rd_framework`'. Ketika radio di pilih, nilai yang dikirim berasal dari salah satu 4 inputan radio. Dan berikut kode untuk proses validasi dan re-populate form:

20. radio/index.php

```
1  <?php
2  require 'Input.php';
3  require 'Validate.php';
4
5  $error = [];
6
7  if (!empty($_POST)) {
8
9      $validate = new Validate($_POST);
10
11     $rd_framework = $validate->setRules('rd_framework', 'Framework', [
12         'sanitize' => 'string',
13         'required' => true,
14         'regexp' => '/^Code Igniter|Laravel|Symfony|Zend$/',
15     ]);
16
17     print_r($_POST);
18
19     if($validate->passed()) {
20         echo "<p> Lolos Validasi!</p>";
21     } else {
22         $error = $validate->getError();
23     }
24
25 }
26
```

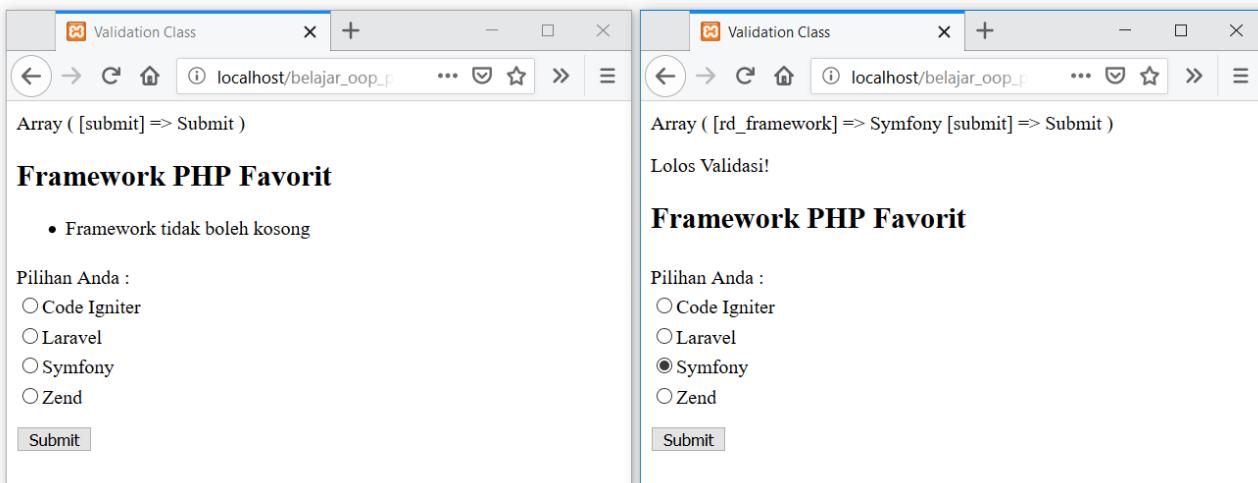
```

27 // Untuk proses re-populate
28
29 $check_code_igniter = Input::get('rd_framework')=='Code Igniter'?
30     'checked' : '';
31 $check_laravel = Input::get('rd_framework')=='Laravel'? 'checked' : '';
32 $check_symfony = Input::get('rd_framework')=='Symfony'? 'checked' : '';
33 $check_zend = Input::get('rd_framework')=='Zend'? 'checked' : '';

```

Perubahan pertama ada di validasi *regexp* pada baris 14. Karena 1 radio button memiliki 4 kemungkinan nilai, maka saya membuat pola *regular expression* dengan logika **OR**. Artinya, nilai inputan `rd_framework` harus salah satu dari Code Igniter, Laravel, Symfony atau Zend, tidak bisa menerima nilai lain.

Kemudian untuk proses re-populate form di baris 29-33 kita hanya perlu mengubah nama inputan form menjadi '`rd_framework`'.



Gambar: Form di-submit langsung (kiri) dan form di-submit dengan memilih radio 'Symfony'

12.14. Validasi Select

Jenis inputan terakhir yang akan kita bahas adalah pilihan dropdown dari tag `<select>`. Di sisi validasi, tidak ada perbedaan dengan inputan form lain, tapi untuk re-populate form kita perlu teknik baru karena inputan select memiliki cara penulisan tersendiri.

Melanjutan contoh sebelumnya, saya akan konversi pilihan framework PHP menjadi menu dropdown:

21.select/index.php

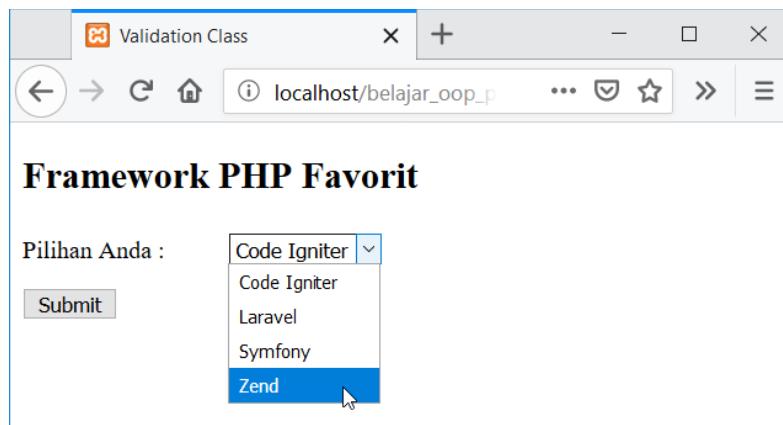
```

1 <form method="post">
2   <div>
3     <label>Pilihan Anda :</label>
4     <select name="sl_framework">
5       <option value="Code Igniter">Code Igniter</option>
6       <option value="Laravel">Laravel</option>

```

Case Study: Validate Class

```
7      <option value="Symfony">Symfony</option>
8      <option value="Zend">Zend</option>
9    </select>
10   </div>
11   <div>
12     <input type="submit" value="Submit" name="submit">
13   </div>
14 </form>
```



Gambar: Form dengan menu dropdown select

Di sini saya membuat tag `<select>` dengan atribut name 'sl_framework'. Ketika form di submit, nilai yang dikirim berasal dari salah satu atribut value milik tag `<option>`.

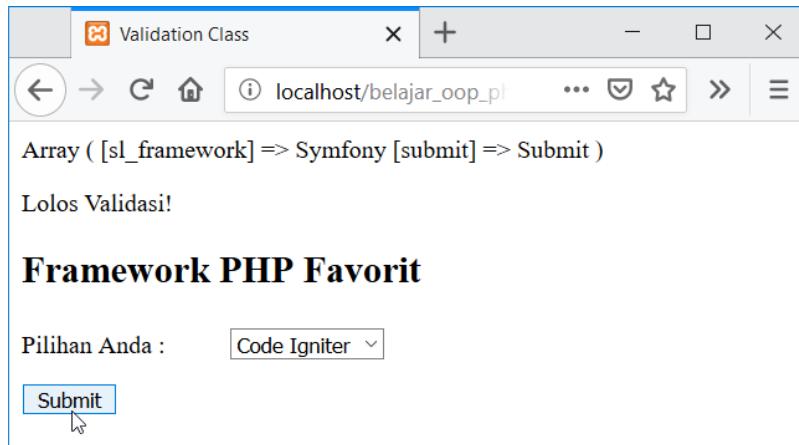
Berikut kode program untuk proses validasi:

21.select/index.php

```
1  <?php
2  require 'Input.php';
3  require 'Validate.php';
4
5  $error = [];
6
7  if (!empty($_POST)) {
8
9    $validate = new Validate($_POST);
10
11   $rd_framework = $validate->setRules('sl_framework', 'Framework', [
12     'sanitize' => 'string',
13     'required' => true,
14     'regexp' => '/^Code Igniter|Laravel|Symfony|Zend$/',
15   ]);
16
17   print_r($_POST);
18
19   if($validate->passed()) {
20     echo "<p> Lolos Validasi!</p>";
21   } else {
22     $error = $validate->getError();
23   }
24 }
```

```
24  
25 }
```

Nyaris tidak ada perbedaan dengan contoh pada inputan radio. Yang berubah hanya di nama inputan form menjadi 'sl_framework'. Berikut tampilan dari validasi ini:



Gambar: Hasil inputan dan validasi menu dropdown select

Dalam contoh kita, kecil kemungkinan inputan 'sl_framework' tidak berisi nilai, karena pilihan untuk mengosongkan inputan memang tidak ada. Begitu halaman di load pertama kali, menu dropdown select langsung terpilih di 'Code Igniter'. Tapi sebagaimana trik *inspect element*, kita tetap harus men-validasi inputan form.

Proses validasi tidak mengalami kendala. Tapi bagaimana dengan proses re-populate? Ini menjadi tantangan tersendiri.

Re-populate Select

Agar inputan tag `<select>` terpilih ketika halaman di load, harus ditambah atribut `selected` ke dalam tag `<option>`. Sebagai contoh, kode berikut akan langsung memilih inputan 'Symfony' begitu halaman di load:

```
1 <select name="sl_framework">  
2   <option value="Code Igniter">Code Igniter</option>  
3   <option value="Laravel">Laravel</option>  
4   <option value="Symfony" selected>Symfony</option>  
5   <option value="Zend">Zend</option>  
6 </select>
```

Sekarang, bagaimana cara agar penambahan atribut `selected` ini bisa dibuat dinamis?

Kita bisa pakai teknik yang sama seperti checkbox dan radio, yakni siapkan sebuah variabel untuk menampung string '`selected`' saat inputan tersebut dipilih oleh user.

Pertama, tempatkan kode berikut setelah proses validasi:

22.select_re-populate/index.php

```
1 <?php
2 require 'Input.php';
3 require 'Validate.php';
4
5 $error = [];
6
7 if (!empty($_POST)) {
8     //.. proses validasi form di sini
9 }
10
11 $select_code_igniter = Input::get('sl_framework')==='Code Igniter'?
12         'selected' : '';
13 $select_laravel = Input::get('sl_framework')==='Laravel'? 'selected' : '';
14 $select_symfony = Input::get('sl_framework')==='Symfony'? 'selected' : '';
15 $select_zend = Input::get('sl_framework')==='Zend'? 'selected' : '';
```

Sedikit berbeda dengan proses re-populate checkbox dan radio, sekarang teks yang diinput berupa 'selected', bukan lagi 'checked'.

Kemudian untuk form bisa ditambah sebagai berikut:

22.select_re-populate/index.php

```
1 <form method="post">
2     <div>
3         <label>Pilihan Anda :</label>
4         <select name="sl_framework">
5             <option value="Code Igniter" <?php echo $select_code_igniter ?>>Code Igniter</option>
6             <option value="Laravel" <?php echo $select_laravel ?>>Laravel</option>
7             <option value="Symfony" <?php echo $select_symfony ?>>Symfony</option>
8             <option value="Zend" <?php echo $select_zend ?>>Zend</option>
9         </select>
10    </div>
11    <div>
12        <input type="submit" value="Submit" name="submit">
13    </div>
14 </form>
```

Hasilnya, setiap kali form dimuat ulang, salah satu variabel ini akan berisi string 'selected'.

Method Input::generateOption

Sebagai pelengkap terakhir class Input, saya ingin membuat method baru yang bernama `generateOption()`. Method ini dipakai untuk meng-generate nilai tag `<option>` untuk dropdown select, lengkap dengan proses re-populate. Method ini akan memudahkan kita membuat inputan tag `<select>`.

Method `generateOption()` butuh 2 buah argument. Argument pertama berupa array berisi string nilai tag `<option>`, serta argument kedua berisi hasil dari inputan form sebelumnya.

Sebagai contoh, untuk membuat menu daftar framework PHP, bisa memakai kode berikut:

```
1 $pilihanFramework = ['Code Igniter', 'Laravel', 'Symfony', 'Zend'];
2 $optionFramework = Input::generateOption($pilihanFramework,
3                                         Input::get('sl_framework'));
```

Dan berikut kode untuk pembuatan method generateOption() di class Input:

23.select_generateOption/Input.php

```
1 public static function generateOption($arr,$selectedValue = "") {
2     $arrOption = [];
3     foreach ($arr as $key => $value) {
4         if ($value==$selectedValue){
5             $arrOption[] = "<option value = \"$value\" selected> $value </option>";
6         } else {
7             $arrOption[] = "<option value = \"$value\"> $value </option>";
8         }
9     }
10    return implode(' ', $arrOption);
11 }
```

Prinsip kerja dari method generateOption() adalah: buat sebuah perulangan foreach untuk setiap array yang ada di argument pertama. Untuk setiap perulangan, generate tag `<option>` dengan nilai yang sama dengan isi array tersebut, lalu simpan kembali ke dalam `$arrOption`.

Selain itu terdapat pemeriksaan apakah nilai yang diproses sama dengan isi argument kedua yang berasal dari inputan. Jika ia, tambah atribut 'selected' ke dalam tag `<option>` tersebut.

Hasilnya, tag `$arrOption` akan berisi array berikut:

```
(  
    [0] => <option value = "Code Igniter"> Code Igniter </option>  
    [1] => <option value = "Laravel" selected> Laravel </option>  
    [2] => <option value = "Symfony"> Symfony </option>  
    [3] => <option value = "Zend"> Zend </option>  
)
```

Agar array bisa dipakai di dalam tag `<select>`, kita akan konversi kembali menjadi string menggunakan fungsi `implode()` di baris 10.

Karena proses pembuatan tag `<option>` sudah diproses oleh method generateOption(), maka ketika menampilkan form cukup dengan perintah berikut:

```
1 <form method="post">
2     <div>
3         <label>Pilihan Anda :</label>
4         <select name="sl_framework">
5             <?php echo $optionFramework; ?>
6         </select>
7     </div>
8     <div>
9         <input type="submit" value="Submit" name="submit">
```

```
10    </div>
11 </form>
```

Secara otomatis tag `<select>` sudah langsung berisi menu daftar framework PHP.

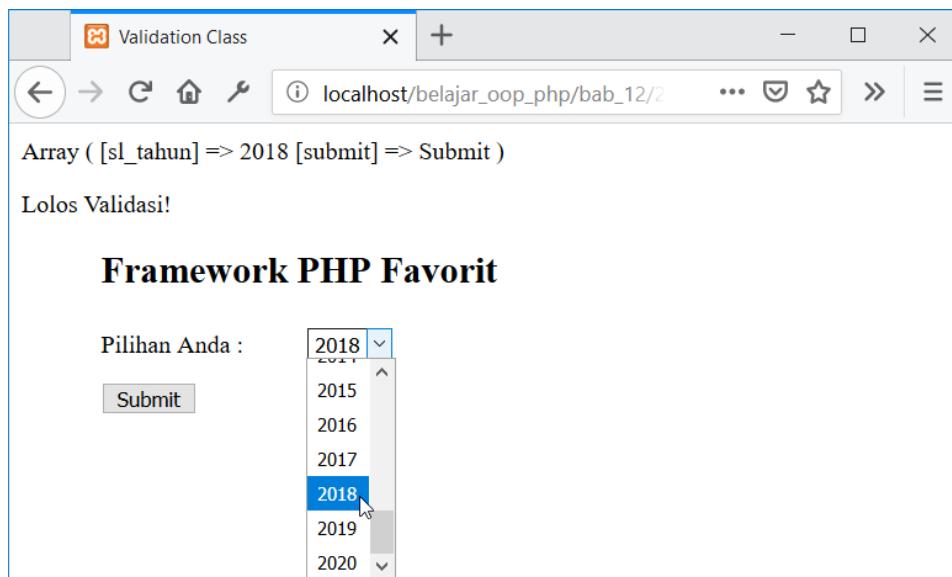
Lebih jauh lagi, method `generateOption()` bisa dipakai untuk membuat daftar menu yang lebih kompleks. Misalnya saya ingin membuat menu pilihan tahun 2000 sampai 2020, yang artinya kita butuh 20 buah tag `<option>`.

Dengan memanfaatkan method `generateOption()`, proses ini jadi lebih mudah:

24.select_generateOption_num/index.php

```
1  <?php
2  require 'Input.php';
3  require 'Validate.php';
4
5  $error = [];
6
7  if (!empty($_POST)) {
8
9      $validate = new Validate($_POST);
10
11     $rd_framework = $validate->setRules('sl_tahun', 'Pilihan Tahun', [
12         'sanitize' => 'string',
13         'required' => true,
14     ]);
15
16     print_r($_POST);
17
18     if($validate->passed()) {
19         echo "<p> Lolos Validasi!</p>";
20     } else {
21         $error = $validate->getError();
22     }
23
24 }
25
26 // generate <option>
27 for ($i=2000;$i<=2020;$i++) {
28     $pilihanTahun[] = $i;
29 }
30
31 $optionTahun = Input::generateOption($pilihanTahun, Input::get('sl_tahun'));
32 ?>
33 <!doctype html>
34 <html lang="id">
35 ...
36     <form method="post">
37         <div>
38             <label>Pilihan Anda :</label>
39             <select name="sl_tahun">
40                 <?php echo $optionTahun; ?>
41             </select>
```

```
42     </div>
43     <div>
44         <input type="submit" value="Submit" name="submit">
45     </div>
46 </form>
47 ...
48 </html>
```



Gambar: Inputan menu tahun yang di generate menggunakan method Input::generateOption()

Dengan method `generateOption()`, kita tidak perlu membuat 20 tag `<option>`, tapi cukup sebuah array yang berisi daftar nama tahun. Inilah yang saya lakukan dengan perulangan for di baris 27-29. Hasilnya, variabel `$pilihanTahun` akan berisi array angka 2000 sampai 2020.

Array `$pilihanTahun` kemudian di input sebagai argument pertama pemanggilan method `generateOption()` di baris 31. Hasil pemanggilan lalu disimpan ke variabel `$optionTahun` dan tinggal di echo di dalam tag `<select>` di baris 40. Method `generateOption()` juga sudah menangani proses re-populate form sehingga nilainya langsung terpilih kembali.

Dalam bab ini kita telah membahas salah satu proses yang cukup kompleks dari setiap pembuatan web, yakni validasi form. Tidak hanya menerapkan prinsip OOP, tapi kita juga membuat sebuah class validasi yang dipakai untuk project-project lain.

Dan inilah yang akan kita bahas sebagai studi kasus bab selanjutnya, yakni membuat sebuah aplikasi CRUD dengan memanfaatkan class `DB`, `Validate`, dan `Input` yang sudah kita rancang.

13. Case Study: Ilkoom Stock Manager

Untuk studi kasus kali ini kita akan masuk ke pembuatan sebuah project utuh. Maksudnya, tidak lagi membuat class atau library seperti 2 bab sebelumnya.

Di sini saya akan merancang sebuah aplikasi **CRUD** (Create, Read, Update dan Delete) dengan memanfaatkan class **DB**, class **Validate** dan class **Input**. Selain itu nantinya juga terdapat beberapa class tambahan lain. Selama pembuatan aplikasi ini, kita akan memakai berbagai teknik pemrograman object, terutama dengan memecah kode program menjadi object-object terpisah.

Agar tidak terlalu kompleks, dalam studi kasus ini saya belum membuat pemisahan struktur dengan teknik **MVC** (Model – View – Controller) sebagaimana yang biasa di temukan di framework PHP. Meskipun begitu, kode yang ada juga tetap "sangat menantang".

Saya menerima cukup banyak request untuk membahas MVC, tapi konsep MVC sendiri juga lumayan rumit. Mengingat ketebalan buku ini yang sudah lebih dari 500 halaman, pilihannya adalah apakah membahas MVC dengan contoh sederhana, atau membahas sebuah project utuh.

Akhirnya saya putuskan untuk membahas project utuh agar kita bisa lihat praktik penggunaan konsep OOP. Untuk materi MVC akan dibahas dalam buku framework PHP, yang salah satunya ada di bab awal buku [Laravel Uncover](#).

13.1. Ilkoom Stock Manager

Agar sedikit lebih 'keren', saya menamakan project ini sebagai **Ilkoom Stock Manager**. Di sini kita akan membuat aplikasi CRUD untuk tabel barang yang selama ini sudah sering menjadi bahan praktik. Setelah itu nantinya juga akan dilengkapi dengan fitur login serta register user.

Dalam project ini kita akan memakai 2 buah tabel, yakni tabel barang dan tabel user. Agar tampilannya lebih menarik, saya juga akan memakai **Bootstrap 5** untuk membuat design tampilan.

Jika sebelumnya anda belum pernah mempelajari Bootstrap, ini tidak lain hanya penambahan beberapa atribut **class** CSS dan tag HTML saja. Bootstrap sendiri tidak mempengaruhi alur logika pemrograman yang kita tulis dengan PHP.

Bahasan materi akan saya bagi menjadi 2 bagian. Pertama, kita akan bahas proses pembuatan aplikasi CRUD tabel barang. Kedua, proses pembuatan CRUD untuk tabel user serta penambahan sistem registrasi user dan fitur login.

Agar lebih rapi, struktur file akan dipecah ke dalam 5 folder: **class**, **css**, **js**, **img** dan **template**.

Folder **class** dipakai untuk menampung seluruh class PHP yang akan (atau telah) kita buat. Diantaranya adalah class `DB.php`, `Input.php`, dan `Validate.php`. Silahkan copy ketiga class ini dari project kita sebelumnya, dan paste ke dalam folder **class**.

Folder **css** dan **js** berisi file-file Bootstrap. Anda bisa ambil langsung file aslinya ke getbootstrap.com, atau bisa juga copy dari contoh file di Google Drive. Isi folder **css** berupa 2 buah file: `bootstrap.css` dan `style.css`. Dan untuk folder **js** berisi 1 file: `bootstrap.bundle.js`.

Folder **img** dipakai untuk menampung gambar yang diperlukan. Dalam project ini kita hanya butuh sebuah gambar `favicon.png`. Ini sekedar pemanis agar tampilan favicon tidak memakai gambar default dari XAMPP.

Terakhir, folder **template** saya siapkan untuk 2 buah file, yakni `header.php` dan `footer.php`. Kedua file ini dipakai agar kita tidak perlu membuat ulang bagian header dan footer untuk setiap halaman. Isi dari keduanya akan di bahas sesaat lagi.

Seluruh file awal tersedia di file `belajar_oop_php.zip` di Google Drive, yakni folder `bab_13/file_awal`.

13.2. Ilkoom Stock Manager: CRUD barang

Dalam bagian pertama, kita akan rancang aplikasi CRUD untuk tabel **barang**. CRUD di sini meliputi proses menampilkan data tabel, input nilai baru, edit, serta menghapus data. Class `DB`, `Input` dan `Validate` sangat membantu dalam proses ini.

Struktur tabel **barang** sama seperti contoh yang kita pakai sebelumnya, yakni terdiri dari 5 kolom: `id_barang`, `nama_barang`, `jumlah_barang`, `harga_barang` dan `tanggal_update`. Jika anda belum memiliki tabel ini atau ingin me-reset ulang isi tabel barang, silahkan akses file `file_awal/db_generate_tabel_barang.php`.

Template: `header.php` dan `footer.php`

Web yang akan kita rancang butuh berbagai file tampilan. Di dalam setiap file ini banyak komponen web yang berulang, terutama di bagian header (tempat menu navigasi), serta bagian footer (tempat link ke file JavaScript). Hampir setiap file tampilan butuh kedua bagian ini dan tidak ada perubahan dari satu file ke file lain.

Daripada melakukan banyak copy paste, saya akan pecah bagian header dan footer menjadi file terpisah, yakni file `header.php` dan `footer.php` di dalam folder **template**. Nantinya file-file

ini akan kita *include* ke file kode program yang membutuhkan.

Berikut kode program untuk membuat file header.php:

ilkoom_stock_barang/template/header.php

```
1  <!doctype html>
2  <html lang="id">
3      <head>
4          <meta charset="utf-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1,
6              shrink-to-fit=no">
7          <title>ILKOOM Inventory</title>
8          <link rel="icon" href="img/favicon.png" type="image/png">
9          <link rel="stylesheet" href="css/bootstrap.css">
10         <link rel="stylesheet" href="css/style.css">
11     </head>
12     <body>
13
14     <!-- NAVBAR -->
15     <nav id="main-navbar" class="navbar navbar-expand-md navbar-dark
16         bg-dark py-0">
17         <div class="container">
18             <a class="navbar-brand" href="#">Hello, Admin</a>
19             <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
20                 data-bs-target="#navbarNav" >
21                 <span class="navbar-toggler-icon"></span>
22             </button>
23
24             <div class="collapse navbar-collapse" id="navbarNav">
25                 <ul class="navbar-nav ms-auto">
26                     <li class="nav-item">
27                         <a class="nav-link p-3 active" href="tampil_barang.php">
28                             Tabel Barang</a>
29                         </li>
30                     </ul>
31                 </div>
32             </div>
33         </nav>
```

Tidak ada kode PHP di file ini. File header.php hanya berisi perintah HTML yang umum kita jumpai di bagian head, antara lain tag <meta> untuk charset dan viewport, tag <title> untuk membuat judul halaman, tag <link> untuk input gambar favicon serta proses import file CSS.

Di baris 15 – 33 terdapat kode untuk pembuatan menu navigasi memanfaatkan komponen navbar bawaan Bootstrap.

Teks "Hello, Admin" di baris 18 merupakan sapaan user default untuk sementara waktu. Setelah sampai ke pembuatan tabel user nanti, sapaan ini akan kita buat dinamis sesuai user yang sedang login.

Dalam halaman ini terdapat 2 perintah import file CSS, yakni bootstrap.css dan style.css di

baris 9 - 10. File `bootstrap.css` merupakan file CSS bawaan Bootstrap, sedangkan file `style.css` berisi sedikit kode CSS tambahan. Berikut isi dari file `style.css`:

ilkoom_stock_barang/css/style.css

```
1  /* Menambah efek border-bottom untuk navbar */
2  #main-navbar .nav-link{
3      border-bottom: 5px solid #343a40;
4      color: #e4e4e4;
5  }
6  #main-navbar .nav-link:hover, #main-navbar a.active {
7      border-bottom: 5px solid #117a8b;
8      color: white;
9  }
10 /* Set warna text-info dan btn-info agar sedikit lebih gelap */
11 .text-info{
12     color: #17a2b8 !important;
13     text-decoration: none;
14 }
15 .text-info:hover{
16     color: #138496 !important;
17 }
18 .btn-info {
19     border-color: #17a2b8;
20     background-color: #17a2b8;
21 }
22 .btn-info:hover {
23     background-color: #138496;
24     border-color: #117a8b;
25 }
26 }
27
28 /*-- Nonaktifkan XXL Breakpoint (agar container tidak terlalu lebar) --*/
29 @media (min-width: 1200px) {
30     .container {
31         max-width: 1140px;
32     }
33 }
```

Kode ini dipakai untuk memberikan efek hover di menu navigasi, mengubah warna class `.text-info` dan `.btn-info`, serta menonaktifkan breakpoint XXL bawaan Booostrap agar tidak terlalu lebar. Jika sebelumnya anda sudah membaca buku [Bootstrap Uncover](#), tentu tidak asing dengan perubahan ini.

Dan berikut kode untuk bagian `footer.php`:

ilkoom_stock_barang/template/footer.php

```
1      <!-- FOOTER -->
2      <footer id="main-footer" class="py-4">
3          <div class="container">
4              <div class="row">
5                  <div class="col-12">
```

```
6      <small>
7      <?php
8          $tanggal = new DateTime('now');
9          echo "Copyright © ".$tanggal->format("Y")." DuniaIlkom";
10         ?>
11         </small>
12     </div>
13     </div>
14   </div>
15 </footer>
16
17   <script src="js/bootstrap.bundle.js"></script>
18 </body>
19 </html>
```

Dalam footer.php terdapat sedikit kode PHP di baris 7 – 10. Class `DateTime()` berguna untuk mengambil tahun dari server. Data tahun kemudian disambung dengan teks 'Copyright', sekedar pemanis tampilan footer.

Di baris 17 terdapat tag `<script>` untuk proses import file JavaScript `bootstrap.bundle.js` yang dipakai Bootstrap.

Kedua file ini, header.php dan footer.php membuat kode kita menjadi lebih efisien. Bagian header dan footer untuk setiap halaman tidak perlu di tulis ulang, cukup memanggil kedua file dengan perintah `include 'template/header.php'` dan `include 'template/footer.php'`.

Autoloading: init.php

Dalam project ini kita butuh mengimport banyak class PHP dengan perintah `include` atau `require`. Namun tidak setiap halaman perlu semua class, misalnya halaman A hanya perlu file `DB.php` saja, halaman B butuh file `Validate.php` saja, dan halaman C perlu file `Validate.php` dan `Input.php`.

Agar pengelolaan import file lebih efisien, saya akan buat sebuah file 'master import' dengan memanfaatkan fitur **autoloading** PHP.

Daripada men-import file satu per satu, cukup import file bernama `init.php`. Di dalam `init.php` terdapat pemanggilan fungsi `spl_autoload_register()` yang akan memproses import file yang diperlukan.

Berikut isi dari file `init.php`:

ilkoom_stock_barang/init.php

```
1 <?php
2 spl_autoload_register(function ($className) {
3     $path = "class/{$className}.php";
4     if (file_exists($path)) {
5         require $path;
6     } else {
7         die ("File $path tidak tersedia");
```

```
8     }
9 );
```

File ini hanya terdiri dari pemanggilan fungsi `spl_autoload_register()`. Jika dalam sebuah file terdapat pemanggilan class yang tidak terdefinisi, fungsi `spl_autoload_register()` akan mencarinya ke dalam folder **class** dan melakukan proses import dengan perintah `require`. Dengan cara ini kita tidak perlu menjalankan perintah `require` untuk setiap file class.

Penjelasan lebih lanjut tentang fungsi `spl_autoload_register()` sudah dibahas pada bab [Autoloading](#).

Menampilkan Tabel Barang: `tampil_barang.php`

Kita akan masuk ke pembuatan halaman pertama, yakni untuk menampilkan tabel barang. File ini saya beri nama `tampil_barang.php`.

Prinsip kerjanya adalah, ketika `tampil_barang.php` di load pertama kali, ambil semua data dari tabel barang dan tampilkan dalam bentuk tabel HTML. Proses pengambilan data ke database sangat terbantu oleh class `DB`, kita cukup menjalankan 1 baris perintah saja:

```
$tabelBarang = $DB->get('barang');
```

Selanjutnya tinggal memproses array yang tersimpan di dalam `$tabelBarang` menjadi tabel HTML.

Di halaman ini juga terdapat sebuah form untuk proses pencarian berdasarkan kolom `nama barang`. Jika user melakukan pencarian dari form ini, halaman `tampil_barang.php` akan di load ulang dengan menyertakan data dari hasil form.

Dengan demikian pada bagian atas halaman `tampil_barang.php` kita butuh pemeriksaan kondisi. Jika terdeteksi ada form yang dikirim (berasal dari form pencarian), maka jalankan method `$DB->getLike()`. Namun jika tidak terdeteksi inputan form, maka jalankan method `$DB->get('barang')` untuk menampilkan semua data barang.

Jika anda masih ingat, method `getLike()` dipakai untuk mengambil data dengan tambahan perintah 'SELECT...LIKE'.

Berikut kode program lengkap dari halaman `tampil_barang.php`:

```
1 <?php
2 // jalankan init.php (untuk session_start dan autoloader)
3 require 'init.php';
4
5 // buat koneksi ke database
6 $DB = DB::getInstance();
7
8 if (!empty($_GET)) {
9     // jika terdeteksi form di submit, tampilkan hasil pencarian
10    $tabelBarang = $DB->getLike('barang', 'nama_barang',
```

```
11                         '%'.Input::get('search')."%");
12 }
13 else {
14     // jika form tidak di submit, ambil semua isi tabel barang
15     $tabelBarang = $DB->get('barang');
16 }
17
18 // include head
19 include 'template/header.php';
20 ?>
21
22 <div class="container">
23     <div class="row">
24         <div class="col-12">
25
26             <!-- Form pencarian -->
27             <div class="py-4 d-flex justify-content-end align-items-center">
28                 <h1 class="h2 me-auto">
29                     <a class="text-info" href="tampil_barang.php">Tabel Barang</a>
30                 </h1>
31                 <a href="tambah_barang.php" class="btn btn-primary">Tambah Barang</a>
32
33             <form class="w-25 ms-4" method="get">
34                 <div class="input-group">
35                     <input type="text" class="form-control" placeholder="Search"
36                         name="search">
37                     <input class="btn btn-outline-secondary" type="submit"
38                         value="Cari">
39                 </div>
40             </form>
41         </div>
42
43             <!-- Tabel barang -->
44             <?php
45                 if (!empty($tabelBarang)) :
46             ?>
47                 <table class="table table-striped align-middle mt-3">
48                     <thead>
49                         <tr>
50                             <th>ID</th>
51                             <th>Nama Barang</th>
52                             <th>Jumlah</th>
53                             <th>Harga (Rp.)</th>
54                             <th>Tanggal Update</th>
55                             <th>Action</th>
56                         </tr>
57                     </thead>
58                     <tbody>
59                         <?php
60                             foreach ($tabelBarang as $barang) {
61                                 echo "<tr>";
62                                 echo "<th>{$barang->id_barang}</th>";
63                                 echo "<td>{$barang->nama_barang}</td>";
64                                 echo "<td>{$barang->jumlah_barang}</td>";
65                                 echo "<td>".number_format($barang->harga_barang, 0, ',', '.')."
```

Case Study: Ilkoom Stock Manager

```

66      "</td>";
67      $tanggal = new DateTime($barang->tanggal_update);
68      echo "<td>".$tanggal->format("d-m-Y H:i")."</td>";
69      echo "<td>";
70          echo "<a href=\"edit_barang.php?id_barang={$barang->id_barang}\"
71              class=\"btn btn-info text-white\">Edit</a> ";
72          echo "<a href=\"hapus_barang.php?id_barang={$barang->id_barang}\"
73              class=\"btn btn-danger\">Hapus</a>";
74      echo "</td>";
75      echo "</tr>";
76  }
77  ?>
78  </tbody>
79  </table>
80
81  <?php
82      endif;
83  ?>
84
85      </div>
86  </div>
87  </div>
88
89 <?php
90 // include footer
91 include 'template/footer.php';
92 ?>
```

ID	Nama Barang	Jumlah	Harga (Rp.)	Tanggal Update	Action
1	TV Samsung 43NU7090 4K	5	5.399.000	04-01-2022 09:59	Edit Hapus
2	Kulkas LG GC-A432HLHU	10	7.600.000	04-01-2022 09:59	Edit Hapus
3	Laptop ASUS ROG GL503GE	7	16.200.000	04-01-2022 09:59	Edit Hapus
4	Printer Epson L220	14	2.099.000	04-01-2022 09:59	Edit Hapus
5	Smartphone Xiaomi Pocophone F1	25	4.750.000	04-01-2022 09:59	Edit Hapus

Copyright © 2022 DuniaIlkom

Gambar: Tampilan halaman `tampil_barang.php`

Kode program untuk halaman `tampil_barang.php` ini memang cukup panjang, tapi akan lebih

panjang jika saja pemrosesan database tidak di tangani oleh class DB.

Di baris 3 terdapat perintah `require 'init.php'`, yakni kode untuk proses *autoload*.

Dengan memanggil file `init.php`, maka ketika kode program menemukan class yang tidak terdefinisi, otomatis akan mencarinya ke dalam folder `class`.

Kemudian di baris 6 terdapat perintah proses instansiasi class DB ke dalam variabel `$DB`. Objek `$DB` inilah yang akan kita pakai untuk mengakses berbagai method dari class DB.

Pada baris ke 8, terdapat kondisi `if (!empty($_GET))`. Kondisi ini dipakai untuk mendeteksi apakah halaman di load setelah form di submit atau tidak. Jika ternyata kondisi `if (!empty($_GET))` menghasilkan nilai `true`, yang artinya halaman di load setelah form di submit, maka isi variabel `$tabelBarang` dengan hasil pemanggilan method `$DB->getLike()`. Dengan perintah ini, variabel `$tabelBarang` akan berisi array data hasil pencarian.

Namun jika ternyata kondisi `if (!empty($_GET))` menghasilkan nilai `false`, yang berarti halaman di load tanpa melalui form, maka isi variabel `$tabelBarang` dengan hasil pemanggilan method `$DB->get('barang')`. Perintah ini akan mengambil semua isi dari tabel barang ke dalam variabel `$tabelBarang`.

Dari mana inputan form ini berasal? Yakni dari sebuah form pencarian tag `<input type="text" name="search">` di baris 35 – 36. Pemanggilan method `$DB->getLike()` di baris 10 bisa dibaca: 'Cari dari tabel barang, dimana kolom `nama_barang` berisi sesuatu yang ada di inputan `$_GET['search']`'.

Di baris 19 terdapat perintah untuk meng-import file `header.php`. Artinya kita sudah mulai masuk ke kode HTML.

Di baris 22 – 24 terdapat kumpulan tag `<div>` dengan berbagai class. Ini adalah class untuk membuat grid system bawaan Bootstrap.

Pada baris 27 – 41 merupakan kode program untuk membuat form pencarian serta tombol **Tambah Barang**. Kembali, saya menggunakan format pembuatan form dari Bootstrap. Namun inti dari kode ini ada di baris 35, berupa tag `<input type="text" name="search">`. Inilah inputan yang dipakai untuk proses pemeriksaan kondisi di baris 8.

Tombol **Tambah Barang** sendiri merupakan sebuah link yang dibuat dari tag ``. Hanya saja link ini tampil dalam bentuk tombol menggunakan class CSS dari Bootstrap. Jika tombol di klik, maka halaman akan pindah ke `tambah_barang.php`. Halaman `tambah_barang.php` sendiri saat ini belum tersedia dan akan kita buat sebentar lagi.

Lanjut di baris 45, terdapat kode PHP untuk memeriksa kondisi `if (!empty($tabelBarang))`. Ini saya lakukan agar ketika tabel barang tidak berisi data apapun, jangan tampilkan tabel.

Namun jika `$tabelBarang` berisi 'sesuatu', lanjut jalankan block kode program di baris 47-79. Di sini lah proses menampilkan tabel barang di buat. Teknik yang dipakai sama seperti dalam bab class DB, yakni menggunakan perulangan `foreach` untuk memproses setiap data yang ada di

dalam \$tabelBarang.

Sedikit tambahan pada baris 70 – 73, saya membuat tombol untuk proses **Edit** dan **Hapus**. Namun tombol ini juga merupakan link dari tag <a> yang di style dengan kode CSS bawaan Bootstrap.

Untuk tombol **Edit** dan **Tambah**, link memiliki tambahan *query string* yang disimpan dalam **id_barang**. Sebagai contoh, data pertama dengan **id_barang = 1** akan menghasilkan tombol **Edit** dengan nilai atribut sebagai berikut:

```
<a href="edit_barang.php?id_barang=1" class="btn btn-info text-white">Edit</a>
```

Kuncinya ada di query string "id_barang=1". Nantinya, nilai inilah yang akan kita pakai ketika merancang kode program untuk proses **edit barang**. Hal yang sama juga dibuat untuk tombol **Hapus** yang akan tampil sebagai berikut:

```
<a href="hapus_barang.php?id_barang=1" class="btn btn-danger">Hapus</a>
```

Query string "id_barang=1" ini akan kita bahas kembali di halaman `edit_barang.php` dan `hapus_barang.php`.

Lanjut di baris 82 terdapat perintah `endif` sebagai penutup block `if` dari baris 45. Terakhir di baris 91 adalah kode program untuk proses input file `footer.php`.

Baik, kita akan lanjut ke halaman `tambah_barang.php`

Menambah Data Barang: `tambah_barang.php`

Kali ini kita akan bahas kode program untuk tambah data barang, yakni proses insert 1 baris baru ke dalam tabel barang. Sebagaimana yang bisa di tebak, halaman ini nantinya memiliki sebuah form sebagai sarana input data. Dan dimana ada form, di situ perlu validasi data.

Seluruh proses validasi dan proses input ke database bisa saja di buat dalam file `tambah_barang.php` ini. Namun saya memutuskan untuk merancang sebuah class baru, yakni class `Barang`.

Class `Barang` nantinya berisi berbagai method yang dipakai untuk memproses data barang, diantaranya proses validasi, proses input barang, edit barang dan hapus barang. File untuk class `Barang` juga akan berada di folder `class` dengan nama `Barang.php`, sehingga bisa ikut ke proses *autoloading* dari file `init.php`.

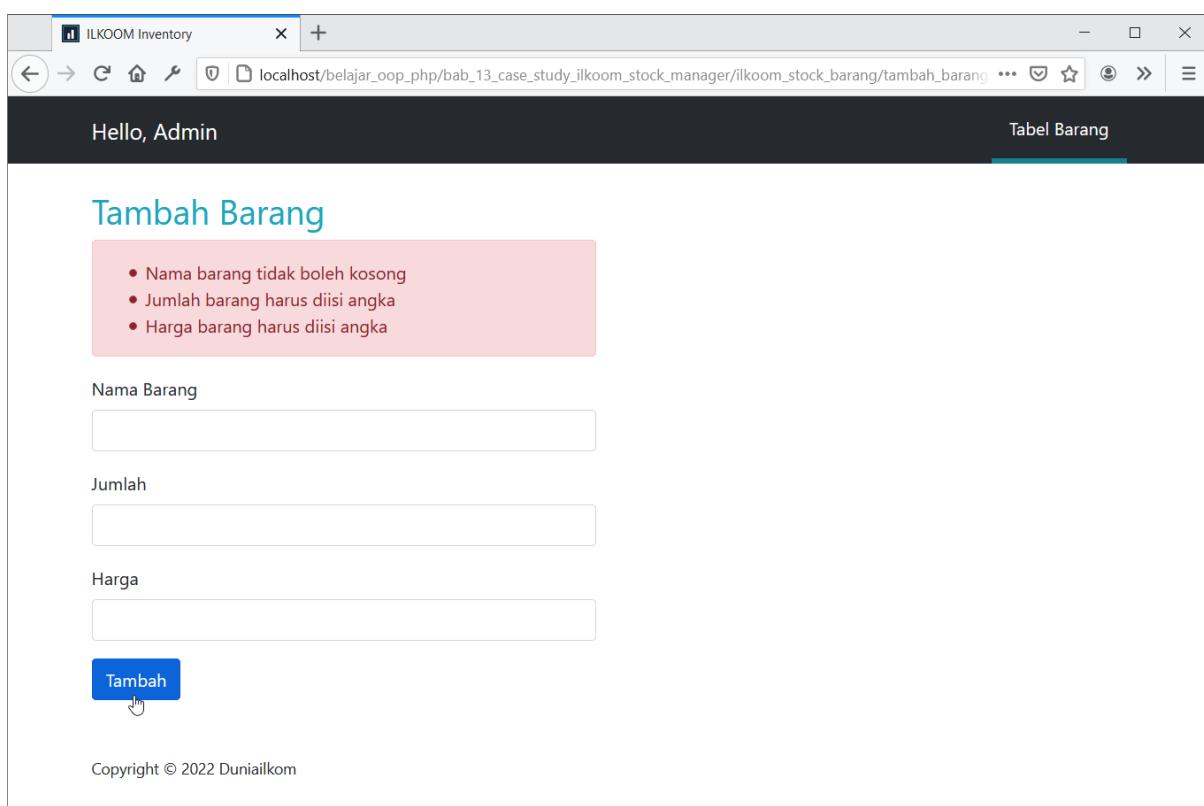
Sebelum masuk ke class `Barang`, mari kita lihat kode lengkap dari halaman `tambah_barang.php`:

ilkoom_stock_barang/tambah_barang.php

```
1  <?php
2  // jalankan init.php (untuk session_start dan autoloader)
3  require 'init.php';
4
```

```
5 // buat object barang yang akan dipakai untuk proses input
6 $barang = new Barang();
7
8 if (!empty($_POST)) {
9     // jika terdeteksi form di submit, jalankan proses validasi
10    $pesanError = $barang->validasi($_POST);
11    if (empty($pesanError)) {
12        // jika tidak ada error, proses insert barang
13        $barang->insert();
14        header('Location:tampil_barang.php');
15    }
16 }
17
18 // include head
19 include 'template/header.php';
20 ?>
21
22 <div class="container">
23     <div class="row">
24         <div class="col-6 py-4">
25             <h1 class="h2 me-auto"><a class="text-info" href="tambah_barang.php">
26                 Tambah Barang</a></h1>
27
28             <?php
29                 // jika ada error, tampilkan pesan error
30                 if (!empty($pesanError)):
31             ?>
32
33             <div id="divPesanError">
34                 <div class="mx-auto">
35                     <div class="alert alert-danger" role="alert">
36                         <ul class="mb-0">
37                             <?php
38                             foreach ($pesanError as $pesan) {
39                                 echo "<li>$pesan</li>";
40                             }
41                         ?>
42                         </ul>
43                     </div>
44                 </div>
45             </div>
46
47             <?php
48                 endif;
49             ?>
50
51             <!-- Form untuk proses insert -->
52             <form method="post">
53                 <div class="mb-3">
54                     <label for="nama_barang" class="form-label">Nama Barang</label>
55                     <input type="text" class="form-control"
56                     name="nama_barang" id="nama_barang"
57                     value="<?php echo $barang->getItem('nama_barang'); ?>">
58                 </div>
59                 <div class="mb-3">
```

```
60      <label for="jumlah_barang" class="form-label">Jumlah</label>
61      <input type="text" class="form-control"
62      name="jumlah_barang" id="jumlah_barang"
63      value=<?php echo $barang->getItem('jumlah_barang'); ?>">
64    </div>
65    <div class="mb-3">
66      <label for="harga_barang" class="form-label">Harga</label>
67      <input type="text" class="form-control"
68      name="harga_barang" id="harga_barang"
69      value=<?php echo $barang->getItem('harga_barang'); ?>">
70    </div>
71    <input type="submit" class="btn btn-primary" value="Tambah">
72  </form>
73
74  </div>
75 </div>
76 </div>
77
78 <?php
79 // include footer
80 include 'template/footer.php';
81 ?>
```



Gambar: Tampilan halaman tambah_barang.php

Halaman di atas baru bisa tampil setelah kita membuat class **Barang** (akan di bahas setelah ini). Tanpa class **Barang**, hasilnya tampil pesan error *File class/Barang.php tidak tersedia.*

Sama seperti halaman `tampil_barang.php`, di baris 3 terdapat perintah `require 'init.php'` untuk proses *autoloading*. Sebenarnya, hampir semua file PHP yang akan kita buat butuh file `init.php` ini di baris awal.

Selanjutnya di baris 6 terdapat perintah untuk proses instansiasi class `Barang`. Kita memang belum membahas isi dari class `Barang` (akan dijelaskan sesaat lagi). Namun kode untuk class `Barang` akan lebih mudah dipelajari jika kita sudah lihat cara pemanggilan method yang ada. Dalam halaman ini akan terdapat 3 buah pemanggilan method milik class `Barang`. Di baris 6 variabel `$barang` akan bertindak sebagai object yang menampung instansiasi class `Barang`.

Di baris 8 terdapat pemeriksaan kondisi `if (!empty($_POST))`. Kondisi `if` dipakai untuk memeriksa apakah terdapat form yang dikirim pada saat halaman di-load. Dalam kasus kita, form tersebut berasal dari form input data barang.

Jika terdeteksi ada form yang dikirim, maka jalankan proses validasi dengan memanggil method `$barang->validasi($_POST)` di baris 10. Method `$barang->validasi()` ini tidak lain berisi pemanggilan ke class `Validate`. Penjelasannya akan dibahas sesaat lagi.

Hasil pemanggilan method `$barang->validasi($_POST)` selanjutnya disimpan ke dalam variabel `$pesanError`. Sesuai namanya, variabel `$pesanError` dipakai untuk menampung pesan error yang terjadi saat proses validasi. Pesan error ini disimpan dalam bentuk array.

Di baris 11 terdapat pemeriksaan kondisi sekali lagi. Kali ini yang diperiksa adalah isi dari variabel `$pesanError`. Jika isinya kosong, yang berarti form lolos validasi, maka kita bisa jalankan method `$barang->insert()`. Method ini akan menginput data baru ke dalam tabel `barang`. Setelah proses input selesai, user di `redirect` ke halaman `tampil_barang.php`.

Setelah kondisi `if`, di baris 19 terdapat perintah `include 'template/header.php'` yang diikuti dengan blok tag `<div>` untuk menampilkan judul "**Tambah Barang**" di baris 25 – 26.

Antara baris 28 – 49 terdapat kode untuk menampilkan pesan error. Teknik yang dipakai sama seperti bahasan bab `Validate`, yakni memakai perulangan `foreach` untuk men-echo isi variabel `$pesanError`. Pesan error akan tampil sebagai sebuah *unordered list* HTML.

Antara baris 52 – 72 merupakan kode HTML untuk pembuatan form. Di sini terdapat 3 inputan form: `nama_barang`, `jumlah_barang`, dan `harga_barang`. Kode HTML yang dipakai terlihat agak rumit karena saya menggunakan komponen form bawaan Bootstrap. Yang menjadi fokus utama kita adalah isi atribut `value` untuk proses re-populate form, di sini saya menggunakan method `$barang->getItem()`.

Sebagai menutup halaman, di baris 77 terdapat perintah `include template/footer.php`.

Dari penjelasan ini terlihat bahwa "kerja" sesungguhnya dilakukan oleh berbagai method dari class `Barang`. Total terdapat 3 buah pemanggilan method, yakni:

```
$barang->validasi();  
$barang->insert();
```

```
$barang->getItem();
```

Kita akan lihat seperti apa isi dari setiap method ini.

Class Barang: Method validasi(), insert() dan getItem()

Class Barang adalah sebuah class yang saya rancang untuk memproses berbagai 'urusan' yang berkaitan dengan data barang. Dengan membuat class terpisah, kode program kita akan lebih rapi dan mudah dikelola. Setiap method yang ada juga bisa dipakai oleh file-file lain, tidak hanya oleh satu halaman saja.

File untuk class Barang berada di dalam folder **class** dengan nama **Barang.php**. Dengan demikian di dalam folder **class** saat ini terdapat 4 file: **DB.php**, **Input.php**, **Validate.php**, serta **Barang.php**.

Berikut kerangka awal dari class Barang:

```
ilkoom_stock_barang/class/Barang.php
```

```
1 <?php
2 class Barang{
3     private $_db = null;
4     private $_formItem = [];
5
6     public function __construct(){
7         $this->_db = DB::getInstance();
8     }
9 }
```

Class barang saya rancang dengan 2 private property: `$_db` dan `$_formItem`. Property `$_db` akan dipakai untuk menampung instansiasi dari class DB, sedangkan property `$_formItem` disiapkan untuk menampung nilai inputan form dari hasil validasi.

Di baris 6-8 terdapat pendefinisian constructor dari class Barang. Constructor ini hanya berisi 1 baris perintah, yakni proses instansiasi class DB ke dalam private property `$this->_db`. Class DB ini akan kita butuhkan untuk proses yang perlu mengakses database.

Tapi tunggu dulu, jika di dalam class Barang kita mengakses class DB, dimana perintah untuk import file **DB.php**?

Ini tidak perlu dilakukan karena class Barang sendiri akan diakses dari file yang sudah menjalankan perintah `require 'init.php'`, seperti dari file **tambah_barang.php**. Dengan demikian, kita tidak perlu melakukan proses import terpisah. Kecuali ada kemungkinan class Barang ini dijalankan oleh file yang tidak mengakses file **init.php**, maka barulah kita perlu meng-import class **DB.php** secara langsung dari dalam class **Barang**.

Berikutnya, kita akan lihat isi dari method pertama class Barang, yakni method **validasi()**:

ilkoom_stock_barang/class/Barang.php

```
1  public function validasi($formMethod){
2      $validate = new Validate($formMethod);
3
4      $this->_formItem['nama_barang'] = $validate->setRules('nama_barang',
5          'Nama barang', [
6              'required' => true,
7              'sanitize' => 'string'
8          ]);
9
10     $this->_formItem['jumlah_barang'] = $validate->setRules('jumlah_barang',
11         'Jumlah barang', [
12             'numeric' => true,
13             'min_value' => 0
14         ]);
15
16     $this->_formItem['harga_barang'] = $validate->setRules('harga_barang',
17         'Harga barang', [
18             'numeric' => true,
19             'min_value' => 0
20         ]);
21
22     if(!$validate->passed()) {
23         return $validate->getError();
24     }
25 }
```

Method `validasi()` butuh sebuah parameter berupa variabel global inputan form, yang disimpan ke dalam `$formMethod`. Variabel global yang dimaksud adalah salah satu dari `$_POST` atau `$_GET`, tergantung dari jenis metode pengiriman form. Dalam contoh kita di halaman `tambah_barang.php`, parameter ini berupa variabel `$_POST`.

Method `validasi()` ini pada dasarnya berisi syarat validasi yang dipakai untuk memanggil method `setRules()` milik class `Validate`. Jadi dalam class `Barang` ini kita juga butuh class `Validate`.

Di baris 2 saya melakukan proses instansiasi class `Validate` ke dalam variabel `$validate`. Class `Validate` sendiri perlu argument berupa variabel global inputan form, yang sebelumnya sudah disimpan dalam parameter `$formMethod`.

Selanjutnya antara baris 4 – 20 terdapat proses menulisan syarat validasi untuk inputan `nama_barang`, `jumlah_barang` dan `harga_barang`. Syarat validasi ini sama persis dengan pembahasan kita di class `Validate`, karena memang akan dipakai untuk memanggil method `setRules()` milik class `Validate`. Hasil dari setiap validasi disimpan ke dalam property `$this->_formItem`.

Property `$this->_formItem` nantinya akan berbentuk array, dimana setiap element akan berisi nilai hasil inputan form yang sudah melewati proses sanitizing dari method `setRules()`.

Sebagai contoh, jika dalam inputan `nama_barang` saya isi string '`Televisi LG<h2>`', maka di

dalam variabel `$this->_formItem['nama_barang']` akan berisi string 'Televisi LG', yakni hasil inputan form yang sudah melewati proses sanitizing oleh method `setRules()`.

Syarat validasi untuk setiap inputan form terbilang standar. Syarat untuk inputan `nama_barang` adalah tidak boleh kosong (harus di isi) dan harus melewati proses sanitizing 'string'.

Sedangkan untuk inputan `jumlah_barang` dan `harga_barang` harus bertipe angka dan harus lebih besar dari 0.

Meskipun saya ingin inputan `jumlah_barang` dan `harga_barang` juga tidak boleh kosong, tapi kita tidak perlu menulis syarat '`required`' => `true`, sebab apabila inputan tidak diisi, tetap tampil pesan error 'Jumlah barang harus diisi angka'.

Di akhir method `validasi()` terdapat kondisi `if` di baris 22. Ini dipakai untuk memeriksa apakah inputan form lolos validasi atau tidak. Jika tidak lolos, maka kembalikan array yang berisi pesan error. Penjelasan tentang method `$validate->passed()` dan `$validate->getError()` sendiri sudah kita bahas dalam bab sebelumnya.

Method kedua untuk class `Barang` adalah `getItem()`, dengan pendefinisian sebagai berikut:

ilkoom_stock_barang/class/Barang.php

```
1 public function getItem($item){  
2     return isset($this->_formItem[$item]) ? $this->_formItem[$item] : '';  
3 }
```

Method `getItem()` saya rancang untuk mengambil isi inputan form yang sudah melewati proses validasi, yakni nilai yang tersimpan dalam `$this->_formItem`. Method `getItem()` ini butuh sebuah parameter berupa nama inputan form yang akan diambil.

Dalam halaman `tambah_barang.php`, method ini akan dipakai untuk proses re-populate isian form. Misalkan saya ingin membuat proses re-populate untuk inputan form `nama_barang`, maka di halaman `tambah_barang.php` perintahnya adalah sebagai berikut:

```
<input type="text" name="nama_barang"  
value="<?php echo $barang->getItem('nama_barang'); ?>">
```

Pemanggilan method `$barang->getItem('nama_barang')` akan mengambil nilai yang ada di dalam `$this->_formItem['nama_barang']`. Isi dari `$this->_formItem['nama_barang']` sendiri berasal dari proses validasi, yakni hasil pemanggilan `$validate->setRules()`.

Namun kita tidak bisa langsung mengembalikan nilai `$this->_formItem[$item]`, karena ada kemungkinan variabel ini tidak terdefinisi sebab proses validasi belum dilakukan.

Oleh karena itu saya akan membuat operasi ternary, yakni jika `$this->_formItem['nama_barang']` terdefinisi (memenuhi syarat `isset()`), maka kembalikan nilai `$this->_formItem['nama_barang']`, namun apabila tidak terdefinisi (yang artinya proses validasi belum dilakukan), maka kembalikan string kosong ''.

Method terakhir dari class `Barang` (untuk saat ini), adalah method `insert()`. Berikut kode programnya:

ilkoom_stock_barang/class/Barang.php

```
1 public function insert(){
2     $newBarang = [
3         'nama_barang' => $this->getItem('nama_barang'),
4         'jumlah_barang' => $this->getItem('jumlah_barang'),
5         'harga_barang' => $this->getItem('harga_barang')
6     ];
7     return $this->_db->insert('barang', $newBarang);
8 }
```

Method `insert()` di dalam class `Barang` tidak lain berupa "perpanjangan tangan" dari method `insert()` kepunyaan class `DB`.

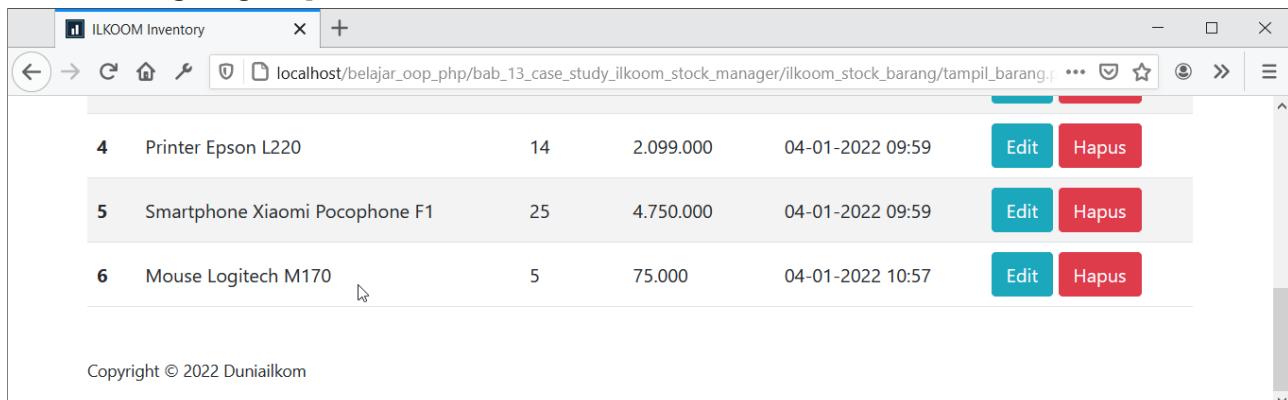
Di baris 2 – 6 saya membuat sebuah associative array `$newBarang`. Isi dari array ini berupa 3 pasangan nama kolom tabel `barang` beserta nilainya. Nilai form diambil menggunakan method `getItem()` yang baru saja kita bahas.

Setelah setiap nama kolom berisi nilai, selanjutnya di baris 7 saya menjalankan method `insert()` milik class `DB`. Perintah yang dipakai adalah `$this->_db->insert('barang', $newBarang)`.

Sekedar pengingat, untuk mengakses nilai private property di dalam sebuah method, kita harus menggunakan tambahan *pseudo variable* `$this`. Misalnya untuk mengakses property `$_db` yang berisi object dari class `DB`, di dalam method `insert()` harus diakses sebagai `$this->_db`, tidak bisa hanya `$_db` saja.

Untuk saat ini kita cukupkan isi class `Barang` dengan 3 method: `validasi()`, `getItem()`, dan `insert()`. Nantinya akan ditambah dengan method baru jika dibutuhkan.

Silahkan tes input data baru menggunakan halaman `tambah_barang.php`. Jika lolos validasi, halaman langsung berpindah ke `tampil_barang.php` dan data baru akan terlihat.



The screenshot shows a web browser window titled "ILKOO Inventory". The address bar displays "localhost/belajar_oop_php/bab_13_case_study_ilkoom_stock_manager/ilkoom_stock_barang/tampil_barang.php". The main content area is a table with the following data:

No	Nama Barang	Jumlah	Harga	Tanggal	Action
4	Printer Epson L220	14	2.099.000	04-01-2022 09:59	<button>Edit</button> <button>Hapus</button>
5	Smartphone Xiaomi Pocophone F1	25	4.750.000	04-01-2022 09:59	<button>Edit</button> <button>Hapus</button>
6	Mouse Logitech M170	5	75.000	04-01-2022 10:57	<button>Edit</button> <button>Hapus</button>

Copyright © 2022 DuniaIlkom

Gambar: Barang baru 'Mouse Logitech M170' berhasil ditambahkan ke dalam database

Mengubah Data Barang: edit_barang.php

Proses berikutnya dalam aplikasi **CRUD** kita adalah **Update**, yakni mengubah isi data tabel barang. Kode yang diperlukan mirip seperti proses tambah barang, hanya saja untuk proses update kita harus ambil terlebih dahulu data sebelumnya dari database.

Sama seperti di halaman `tambah_barang.php`, halaman `edit_barang.php` juga akan memiliki form. Untungnya, karena proses validasi form sudah dibuat terpisah ke dalam class `Barang`, proses validasi untuk form update tinggal memanfaatkan method `validasi()` yang sama. Jika tidak dipisah seperti ini, maka kita harus buat kembali proses validasi untuk form update.

Proses update ke database nantinya juga akan ditangani oleh method dari class `Barang`, yakni method `update()`. Penjelasan dari method `update()` akan kita bahas secara terpisah.

Baik, berikut kode program lengkap dari halaman `edit_barang.php`:

ilkoom_stock_barang/edit_barang.php

```

1 <?php
2 // jalankan init.php (untuk session_start dan autoloader)
3 require 'init.php';
4
5 // halaman tidak bisa diakses langsung, harus ada query string id_barang
6 if(empty(Input::get('id_barang'))) {
7     die ('Maaf halaman ini tidak bisa diakses langsung');
8 }
9
10 // ambil semua data barang yang akan diupdate dari database
11 $barang = new Barang();
12 $barang->generate(Input::get('id_barang'));
13
14 if (!empty($_POST)) {
15     // jika terdeteksi form $_POST di submit, jalankan proses validasi
16     $pesanError = $barang->validasi($_POST);
17     if (empty($pesanError)) {
18         // jika tidak ada error, proses update barang
19         $barang->update($barang->getItem('id_barang'));
20         header('Location:tampil_barang.php');
21     }
22 }
23
24 // include head
25 include 'template/header.php';
26 ?>
27
28 <!doctype html>
29
30 <div class="container">
31     <div class="row">
32         <div class="col-6 py-4">
33             <h1 class="h2 me-auto"><a class="text-info" href="edit_barang.php">
34                 Edit Barang</a></h1>
35

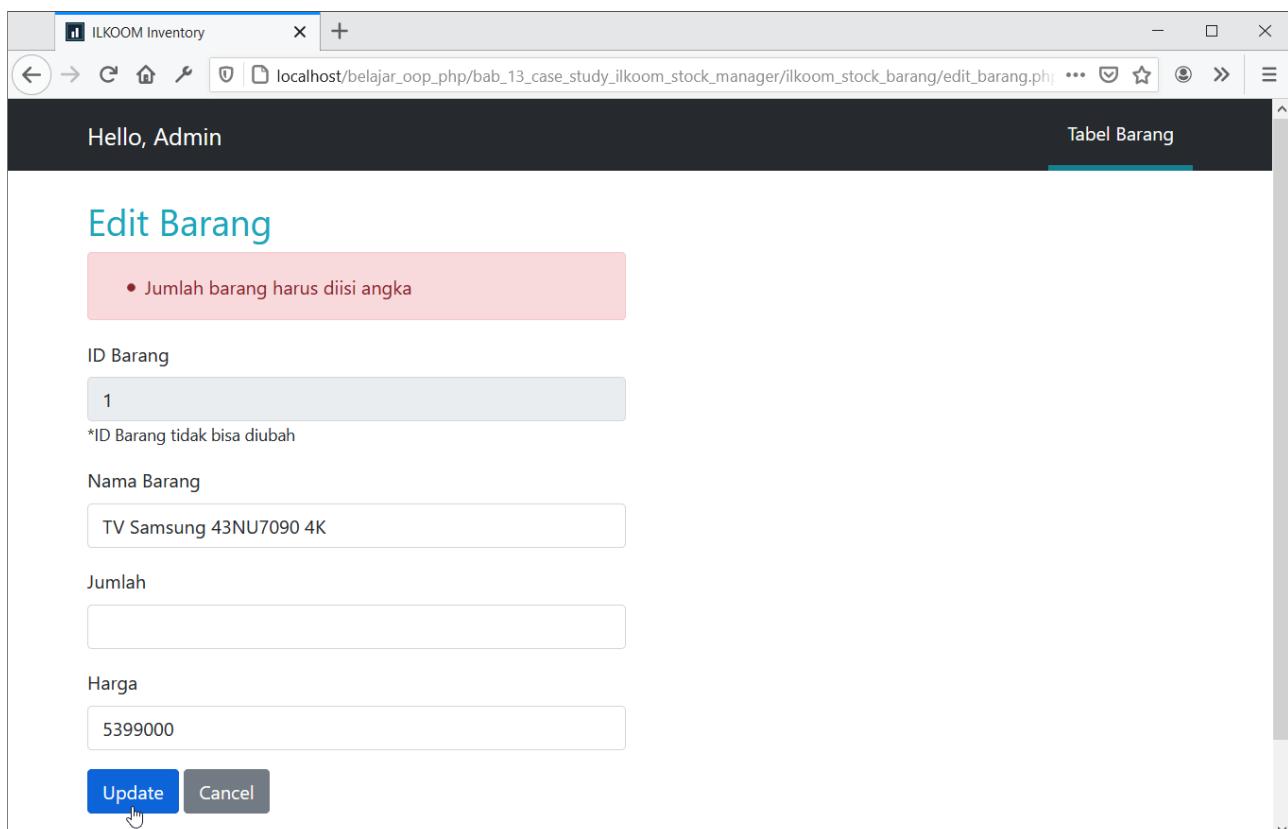
```

```

36 <?php
37     // jika ada error, tampilkan pesan error
38     if (!empty($pesanError)):
39     ?>
40
41     <div id="divPesanError">
42         <div class="mx-auto">
43             <div class="alert alert-danger" role="alert">
44                 <ul class="mb-0">
45                     <?php
46                     foreach ($pesanError as $pesan) {
47                         echo "<li>$pesan</li>";
48                     }
49                 ?>
50                 </ul>
51             </div>
52         </div>
53     </div>
54
55     <?php
56     endif;
57 ?>
58
59     <!-- Form untuk proses update -->
60     <form method="post">
61
62         <div class="mb-3">
63             <label for="id_barang" class="form-label">ID Barang</label>
64             <input type="text" class="form-control" disabled
65                 name="id_barang" id="id_barang"
66                 value="<?php echo $barang->getItem('id_barang'); ?>">
67             <small class="d-block">*ID Barang tidak bisa diubah</small>
68         </div>
69
70         <div class="mb-3">
71             <label for="nama_barang" class="form-label">Nama Barang</label>
72             <input type="text" class="form-control"
73                 name="nama_barang" id="nama_barang"
74                 value="<?php echo $barang->getItem('nama_barang'); ?>">
75         </div>
76
77         <div class="mb-3">
78             <label for="jumlah_barang" class="form-label">Jumlah</label>
79             <input type="text" class="form-control"
80                 name="jumlah_barang" id="jumlah_barang"
81                 value="<?php echo $barang->getItem('jumlah_barang'); ?>">
82         </div>
83
84         <div class="mb-3">
85             <label for="harga_barang" class="form-label">Harga</label>
86             <input type="text" class="form-control"
87                 name="harga_barang" id="harga_barang"
88                 value="<?php echo $barang->getItem('harga_barang'); ?>">
89         </div>
90

```

```
91      <input type="submit" class="btn btn-primary" value="Update">
92      <a href="tampil_barang.php" class="btn btn-secondary">Cancel</a>
93
94  </form>
95
96  </div>
97  </div>
98 </div>
99
100 <?php
101 // include footer
102 include 'template/footer.php';
103 ?>
```



Gambar: Tampilan halaman edit_barang.php

Halaman di atas baru bisa tampil setelah kita menambah method generate() ke dalam class Barang (di bahas sesaat lagi). Tanpa method ini, akan tampil pesan *Fatal error: Uncaught Error: Call to undefined method Barang::generate()*.

Jika anda masih ingat, halaman `edit_barang.php` ini diakses dari tombol link di halaman `tampil_barang.php`. Pada saat tombol "**Edit**" di klik, maka halaman akan berpindah ke `edit_barang.php` beserta sebuah query string yang berisi `id_barang`.

Dalam gambar di atas, bagian URL berbentuk string `http://localhost/belajar_oop_php/bab_13/ilkoom_stock_barang/edit_barang.php?id_barang=1`. Kuncinya ada di bagian

`id_barang=1`, karena inilah informasi yang dipakai untuk proses edit. Dalam contoh kita, barang yang akan di edit adalah yang memiliki nilai `id_barang = 1`.

Jika query string ini diubah, misalnya menjadi `edit_barang.php?id_barang=5`, maka barang yang akan di edit adalah yang memiliki nilai `id_barang = 5`.

Query string adalah sebutan untuk tambahan data di dalam alamat URL. Data ini diawali dengan tanda tanya '?', seperti `?id_barang=1`. Selain ditulis manual, query string umumnya berasal dari inputan form yang dikirim dengan metode **GET**.

Tambahan query string di sebuah URL bisa diambil dari global variabel `$_GET`. Sebagai contoh, jika tertulis `edit_barang.php?id_barang=99`, kita bisa mengambil nilai "99" dengan perintah `$_GET['id_barang']`.

Isi form **Edit Barang** sendiri mirip seperti form di halaman **Tambah Barang** dengan 1 tambahan inputan form berupa nomor `id_barang`. Tapi saya putuskan bahwa nomor `id_barang` ini tidak bisa diubah. Yang bisa di edit hanya inputan nama barang, jumlah barang dan harga barang.

Masuk ke kode program, seperti biasa di baris 3 terdapat perintah `require 'init.php'` untuk proses *autoload*.

Di baris 6 berupa sebuah kondisi `if(empty(Input::get('id_barang')))`. Kondisi ini dipakai untuk memeriksa apakah di bagian URL terdapat query string `id_barang` atau tidak.

Untuk mengambil `id_barang` dari query string, saya memakai method `get()` bawaan class `Input`. Karena mengirim data di dalam query string sama halnya dengan menginput form melalui metode **GET**.

Jika ternyata query string '`id_barang`' tidak ditemukan (kosong), maka kondisi di baris 6 akan menghasilkan nilai **true**. Kondisi ini bisa terjadi jika halaman `edit_barang.php` diakses secara manual, bukan dari tombol di halaman `tampil_barang`. Karena tidak terdapat query string `id_barang`, kita juga tidak tau apa data yang akan di edit, sehingga saya memutuskan untuk menjalankan perintah `die('Maaf halaman ini tidak bisa diakses langsung')`.

Solusi yang lebih baik sebenarnya bisa dengan me-*redirect* user ke halaman khusus yang memberi tahu bahwa halaman `edit_barang.php` tidak bisa diakses langsung. Tapi daripada membuat sebuah halaman baru, saya langsung pakai cara pintas dengan fungsi `die()` saja.

Lanjut, di baris 11 saya membuat perintah untuk instansiasi class `Barang`. Object hasil instansiasi di simpan ke dalam variabel `$barang`.

Di baris 12 terdapat pemanggilan method `$barang->generate()`. Method ini memang belum tersedia di dalam class `Barang` (akan kita bahas sesaat lagi). Method `$barang->generate()` dipakai untuk proses pengambilan data dari database. Method ini butuh sebuah argument berupa data query string `id_barang`.

Sebagai contoh, jika halaman ini di akses sebagai `edit_barang?id_barang=3`, maka kode di baris 12 akan dijalankan sebagai `$barang->generate(3)`. Di dalam method `generate()`, nantinya terdapat sebuah proses pengambilan data ke database barang untuk `id_barang = 3`.

Di baris 14 terdapat kondisi pemeriksaan apakah ada form yang dikirim lewat method `$_POST`. Kondisi ini akan terpenuhi jika form update di bagian HTML nanti sudah diisi dan user men-klik tombol **Update**. Jika ini yang terjadi, maka jalankan proses validasi di baris 16.

Proses validasi dilakukan dengan memanggil method `$barang->validasi($_POST)`, yakni method yang sama dengan proses validasi di halaman `tambah_barang.php`. Inilah salah satu manfaat dari penggunaan konsep OOP yang baik, dimana kita bisa menggunakan method yang sama dari berbagai halaman yang dibutuhkan.

Kemudian di baris 17 ada pemeriksaan isi variabel `$pesanError`. Jika tidak ada error, maka jalankan perintah `$barang->update()`. Secara internal, method `update()` akan menjalankan method `insert()` milik class `DB`. Kita akan lihat rancangan method ini sesaat lagi.

Jika proses update berhasil, user akan di `redirect` ke halaman `tampil_barang.php`, sebagaimana perintah di baris 20.

Selanjutnya sisa kode program kita sangat mirip dengan halaman tambah barang, yakni proses include `header` di baris 25, menampilkan pesan error di baris 41 – 53, kode untuk form di baris 60 – 94, serta proses include `footer` di baris 102.

Yang berbeda hanya berupa tambahan 1 inputan form `id_barang` di baris 64–66. Inputan ini memakai tambahan atribut `disabled` agar tidak bisa diisi (tampil sebagai form berwarna abu-abu).

Class Barang: Method `generate()` dan `update()`

Dalam halaman `edit_barang.php` terdapat pemanggilan 2 buah method baru milik class `Barang`, yakni:

```
$barang->generate()  
$barang->update()
```

Kita akan lihat seperti apa perancangan kedua method ini.

Method `generate()` dipakai untuk mengambil data dari tabel `barang`. Berikut kode program yang dibutuhkan:

ilkoom_stock_barang/class/Barang.php

```
1 public function generate($id_barang){  
2     $result = $this->_db->getWhereOnce('barang',[ 'id_barang' , '=' , $id_barang ]);  
3     foreach ($result as $key => $val) {  
4         $this->_formItem[$key] = $val;  
5     }  
6 }
```

Kodenya hanya 6 baris, tapi banyak proses yang berjalan di "latar belakang".

Pertama, method ini butuh sebuah parameter berupa nilai `id_barang` yang akan diambil. Parameter ini disimpan ke dalam variabel `$id_barang`.

Di baris 2, saya memanggil method `getWhereOnce()` bawaan class `DB`. Jika anda masih ingat, method `getWhereOnce()` dipakai untuk mengambil 1 baris data dari sebuah tabel. Perintah di baris 2 akan menghasilkan query sebagai berikut:

```
SELECT * FROM barang WHERE id_barang = $id_barang
```

Hasil pemanggilan method ini disimpan ke dalam variabel `$result`, yang akan berisi object dari 1 baris data tabel. Ingat, bahwa di dalam method `getWhereOnce()` kita mengambil data tabel dengan method `fetchAll(PDO::FETCH_OBJ)[0]`.

Sebagai contoh, jika method `generate()` dipanggil dengan `generate(1)`, maka varibel `$result` akan berisi data sebagai berikut:

```
stdClass Object (
  [id_barang] => 1
  [nama_barang] => TV Samsung 43NU7090 4K
  [jumlah_barang] => 5 [harga_barang] => 5399000
  [tanggal_update] => 2019-04-10 18:29:30
)
```

Setelah data berhasil diambil, di baris 3 – 5 saya membuat perulangan `foreach`. Tujuannya agar setiap data dari database bisa diinput ke dalam property `$this->_formItem`. Jika anda masih ingat, di dalam method `validasi()` milik class `Barang` saya juga menyimpan hasil proses validasi inputan form ke dalam property `$this->_formItem`.

Menggunakan contoh `id_barang = 1`, maka perulangan di baris 3 – 5 akan memproses data sebagai berikut:

```
$this->_formItem[id_barang] = 1
$this->_formItem[nama_barang] = 'TV Samsung 43NU7090 4K'
$this->_formItem[jumlah_barang] = 5399000
$this->_formItem[tanggal_update] = 2019-04-10 18:29:30
```

Setelah menjalankan method `generate()`, maka variabel `$_formItem` akan berisi semua data untuk 1 baris barang.

Data-data ini bisa diambil menggunakan method `getItem()` milik class `Barang` dan menjadi nilai untuk atribut `value`, seperti contoh inputan form berikut:

```
<input type="text" class="form-control" name="nama_barang" disabled
value="php echo $barang-&gt;getItem('id_barang'); ?&gt;"&gt;</pre
```

Perintah `echo $barang->getItem('id_barang')` akan menampilkan angka 1, yang berasal dari isi `$this->_formItem[id_barang]`.

Method kedua yang akan kita tambah ke dalam class Barang adalah `update()`. Sesuai dengan namanya, method ini dipakai untuk melakukan proses update ke database. Secara internal, method ini akan memanggil method `update()` milik class DB.

Berikut kode yang dibutuhkan:

ilkoom_stock_barang/class/Barang.php

```
1 public function update($id_barang){  
2     $newBarang = [  
3         'nama_barang' => $this->getItem('nama_barang'),  
4         'jumlah_barang' => $this->getItem('jumlah_barang'),  
5         'harga_barang' => $this->getItem('harga_barang')  
6     ];  
7     $this->_db->update('barang', $newBarang, ['id_barang', '=', $id_barang]);  
8 }
```

Method `update()` menggunakan cara yang sama seperti method `insert()` milik class Barang.

Method `update()` butuh sebuah parameter berupa `id_barang` yang akan di update. Kemudian di baris 2 saya membuat associative array `$newBarang` yang berisi nilai untuk 3 kolom tabel. Array `$newBarang` ini selanjutnya diinput ke dalam method `$this->_db->update()`, yang tidak lain adalah pemanggilan dari method `update()` milik class DB.

Hasilnya, data lama yang ada di dalam tabel akan di update dengan nilai baru.

Menghapus Data Barang: hapus_barang.php

Proses terakhir untuk project CRUD tabel barang adalah **Delete**, yang akan ditulis ke dalam file `hapus_barang.php`. Jika di bandingkan dengan proses tambah barang dan edit barang, kode program untuk halaman hapus ini relatif lebih sederhana.

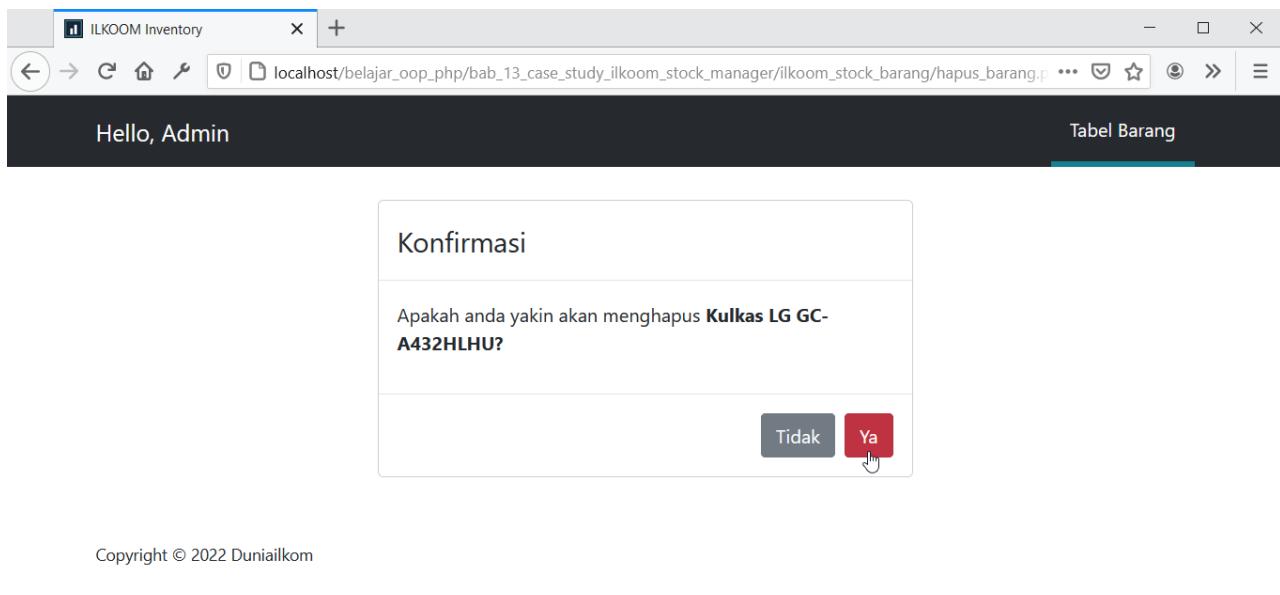
Untuk bisa mengakses file `hapus_barang.php` harus dari halaman `tampil_barang.php`, yakni dengan cara men-klik tombol link "**HAPUS**". Pada saat tombol di klik, akan terkirim query string `id_barang` yang sama seperti untuk halaman `edit_barang.php`. Nilai dari query string inilah yang menjadi patokan data apa yang akan dihapus.

Berikut kode program lengkap dari halaman `hapus_barang.php`:

ilkoom_stock_barang/hapus_barang.php

```
1 <?php  
2 // jalankan init.php (untuk session_start dan autoloader)  
3 require 'init.php';  
4  
5 // halaman tidak bisa diakses langsung, harus ada query string id_barang  
6 if(empty(Input::get('id_barang'))){  
7     die ('Maaf halaman ini tidak bisa diakses langsung');  
8 }  
9  
10 //ambil data barang yang akan dihapus
```

```
11 $barang = new Barang();
12 $barang->generate(Input::get('id_barang'));
13
14 if (!empty($_POST)) {
15     // jika terdeteksi form di submit, hapus barang berdasarkan nilai id_barang
16     $barang->delete(Input::get('id_barang'));
17     header('Location:tampil_barang.php');
18 }
19
20 // include head
21 include 'template/header.php';
22 ?>
23
24 <div class="container">
25     <div class="row">
26         <div class="col-6 mx-auto">
27
28             <!-- Modal Untuk Konfirmasi Hapus -->
29             <div id="modalHapus">
30                 <div class="modal-dialog modal-confirm">
31                     <div class="modal-content">
32                         <div class="modal-header">
33                             <h4 class="modal-title">Konfirmasi</h4>
34                         </div>
35                         <div class="modal-body">
36                             <p> Apakah anda yakin akan menghapus
37                                 <b><?php echo $barang->getItem('nama_barang'); ?>?</b></p>
38                         </div>
39                         <div class="modal-footer">
40                             <a href="tampil_barang.php" class="btn btn-secondary">Tidak</a>
41
42                             <form method="post">
43                                 <input type="hidden" name="id_barang"
44                                     value="<?php echo $barang->getItem('id_barang'); ?>">
45                                 <input type="submit" class="btn btn-danger" value="Ya">
46                             </form>
47
48                         </div>
49                     </div>
50                 </div>
51             </div>
52
53         </div>
54     </div>
55 </div>
56
57 <?php
58 // include footer
59 include 'template/footer.php';
60 ?>
```



Gambar: Tampilan halaman `hapus_barang.php`

Tampilan halaman `hapus_barang.php` ini tidak lain hanya berupa teks konfirmasi. Jika tombol "**Ya**" di klik, maka data tersebut akan dihapus dari database.

Tampilan jendela ini memanfaatkan komponen **Modal** bawaan Bootstrap. Tapi saya memutuskan untuk tidak membuat efek *popup* (kotak yang "melayang" di atas tabel barang), agar kita tidak perlu menggunakan kode JavaScript tambahan.

Isi halaman `hapus_barang.php` diawali dengan perintah `require 'init.php'` di baris 3 untuk proses *autoload*.

Kemudian di baris 6 – 8 terdapat pemeriksaan kondisi apakah query string `id_barang` ada atau tidak. Kode ini sama persis seperti yang ada di halaman update. Jika query string `id_barang` tidak ada, yang artinya halaman ini tidak diakses dari `tampil_barang.php`, maka langsung hentikan kode program dengan fungsi `die()`.

Di baris 11 merupakan perintah untuk proses instansiasi class `Barang`, yang di lanjutkan dengan pemanggilan method `$barang->generate()` di baris 12. Method `generate()` diperlukan untuk mengambil nilai `nama_barang`, karena info yang diterima dari query string hanya `id_barang` saja.

Di baris 14 terdapat pemeriksaan apakah variabel global `$_POST` berisi sesuatu. Jika iya, maka artinya halaman `hapus_barang.php` di load dari sebuah form. Form yang dimaksud nantinya ada di bagian bawah halaman ini.

Isi dari kondisi `if` berupa pemanggilan method `$barang->delete()` di baris 16. Inilah method yang akan melakukan proses hapus dari database. Method ini akan kita tambah ke dalam class `Barang` sesaat lagi. Tapi perhatikan bahwa method `$barang->delete()` dipanggil dengan sebuah argument berupa `id_barang` yang akan dihapus.

Bagian awal kode PHP ditutup dengan perintah `include 'template/header.php'` di baris 21.

Di baris 29 – 51 terdapat kode HTML untuk pembuatan komponen **Modal** bawaan Bootstrap. Komponen modal ini tidak lain berbentuk kotak konfirmasi seperti yang terlihat dari gambar.

Di baris 37 saya memakai sedikit kode PHP untuk menampilkan nama barang. Kita bisa memanggil method `$barang->getItem('nama_barang')` karena sudah didahului oleh method `$barang->generate()` di baris 12. Artinya di dalam class Barang property `$_formItem['nama_barang']` sudah berisi nilai.

Di akhir struktur Modal terdapat 2 buah tombol. Tombol pertama adalah tombol "**Tidak**" yang tidak lain berupa link tag `<a>` menuju halaman `tampil_barang.php`. Tombol ini tidak butuh pemrosesan PHP karena hanya berupa link HTML biasa.

Tombol kedua adalah tombol "**Ya**" yang dipakai untuk proses penghapusan. Tombol ini sebenarnya bisa juga dibuat berbentuk link tag `<a>` dengan penambahan query string untuk data barang `id_barang` yang akan dihapus. Tapi saya memutuskan untuk memakai teknik lain, yakni menggunakan *hidden form*.

Form yang dimaksud ada di baris 42-46 yang terdiri dari 2 buah inputan. Inputan pertama berupa tag `<input type="hidden" name="id_barang">`. Karena *hidden*, inputan form tidak akan terlihat di web browser. Inputan ini saya pakai untuk menyimpan `id_barang` yang telah di konfirmasi akan dihapus. Nilai `id_barang` sendiri di dapat dengan memanggil method `$barang->getItem('id_barang')` di baris 44.

Inputan form kedua berupa tombol `Submit` yang tampil dengan teks '**Ya**'. Ketika tombol '**Ya**' ini di klik, halaman `hapus_barang.php` akan di re-load dengan variabel global `$_POST` sudah berisi sebuah nilai. Inilah yang kemudian di proses ke dalam kondisi `if` di baris 14 untuk menjalankan proses hapus data.

Class Barang: Method `delete()`

Di halaman `hapus_barang.php`, proses penghapusan data ke database di proses oleh method `delete()` milik class Barang. Isi method ini tidak lain hanya meneruskan nilai `id_barang` ke method `delete()` milik class DB.

Berikut kode yang dibutuhkan:

`ilkoom_stock_barang/class/Barang.php`

```
1  public function delete($id_barang){
2      $this->_db->delete('barang',['id_barang','=',$id_barang]);
3  }
```

Method `delete()` butuh sebuah parameter yang berisi `id` dari barang yang akan dihapus. Parameter ini disimpan ke dalam variabel `$id_barang`. Nilai `$id_barang` kemudian diinput lagi ke pemanggilan method `$this->_db->delete()`, yakni method `delete()` bawaan class DB.

Sebagai contoh, jika parameter `$id_barang` diisi dengan angka 3, maka pemanggilan method `delete()` di baris 2 akan memproses query berikut:

```
DELETE FROM barang WHERE id_barang = 3
```

Hasilnya, data dengan `id_barang = 3` akan dihapus dari tabel `barang`.

Halaman `hapus_barang.php` ini melengkapi aplikasi CRUD untuk tabel `barang`. Silahkan anda coba semua fitur yang ada, mulai dari proses insert, edit dan delete. Kode program yang dipakai memang cukup rumit karena kita menggunakan 4 class secara bergantian, yakni class `DB`, class `Validate`, class `Input` dan juga class `Barang`. Khusus di dalam class `Barang`, juga dipakai untuk memanggil ketiga class lain.

Di balik kompleksitas ini, kode program kita juga lebih fleksibel. Di akhir buku PHP Uncover saya juga membuat project CRUD sederhana menggunakan PHP prosedural (tidak memakai konsep OOP). Meskipun semuanya sudah berjalan, tapi kode program yang ada "terlalu melekat" satu sama lain. Misalnya proses validasi langsung ditulis di halaman tersebut yang berakibat cukup susah jika ingin di modifikasi.

Namun untuk project CRUD kita kali ini, sebagian besar kode program memiliki ruang lingkup yang terpisah. Validasi di proses oleh sebuah class khusus dan proses ke database oleh class lain. Sehingga jika kita perlu menambah beberapa inputan form, tidak perlu merombak ulang seluruh kode program.

Berikutnya kita akan "upgrade" aplikasi CRUD Ilkoom Stock Manager ini dengan sebuah konsep *autentikasi*, dimana hanya user yang terdaftar saja yang bisa masuk ke dalam aplikasi kita.

13.3. Ilkoom Stock Manager: Autentikasi User

Maksud dari Autentikasi User adalah kita akan membuat sebuah mekanisme login dan register user. Nantinya, hanya user yang terdaftar saja yang bisa mengakses tabel `barang`. Selain login dan register, kita juga akan merancang tampilan halaman untuk melihat data user, ubah password dan logout.

Proses autentikasi sangat erat kaitannya dengan pemeriksaan password. Seseorang baru bisa diberi hak akses jika username dan password yang diisi di form login sesuai dengan data yang tersimpan di database.

Teknik penyimpanan username dan password ini juga penting untuk dibahas, karena berhubungan dengan keamanan aplikasi kita dan juga data user yang tersimpan. Terlebih lagi mayoritas user menggunakan password yang sama untuk banyak website. Jika password disimpan dalam bentuk "data mentah" maka itu sangat-sangat berbahaya.

Oleh karena itu sebelum masuk ke pembuatan aplikasi, saya ingin membahas sedikit tentang

cara penyimpanan password, terutama teknik *hashing* serta function bawaan PHP untuk proses tersebut.

Password Hashing

Saya yakin sebagai besar dari pembaca buku ini sudah pernah membuat aplikasi PHP sederhana, karena materi OOP PHP sendiri bukan lagi materi dasar, tapi sudah ke materi PHP lanjutan.

Di akhir buku PHP Uncover saya juga membahas studi kasus pembuatan aplikasi CRUD sederhana, dimana terdapat materi tentang cara penyimpanan password menggunakan teknik *hashing md5()* dan *sha1()*.

Di situ ada penjelasan mengenai pentingnya menyimpan teks password bukan dalam bentuk 'data mentah', tapi teks hasil *hashing*. Jika user menginput password 'rahasia', maka kita tidak boleh menyimpan teks tersebut langsung ke dalam database, yang disimpan harus dalam bentuk hashing seperti 'ac43724f16e9241d990427ab7c8f4228'. Tujuannya agar jika (seandainya) aplikasi kita di bobol orang lain, data password user tidak bisa langsung terbaca.

Secara sederhana, **hashing** adalah sebuah teknik mengacak karakter agar tidak bisa dibaca secara langsung. Sebagai contoh, teks 'rahasia' jika di hashing dengan algoritma MD5 akan menjadi string 'ac43724f16e9241d990427ab7c8f4228'.

Fitur lain dari hashing adalah hanya bekerja satu arah. Maksudnya kita tidak bisa mencari tahu apa teks asli dari kode 'ac43724f16e9241d990427ab7c8f4228', setidaknya secara teori hal ini tidak bisa dilakukan.

Sebuah algoritma hashing juga menghasilkan teks dengan panjang tetap, tidak peduli berapa panjang teks asal. Sebagai contoh, algoritma MD5 akan menghasilkan teks hashing sepanjang 32 karakter, meskipun teks asal memiliki panjang 1 karakter atau 10.000 karakter.

Saat ini terdapat puluhan (atau mungkin ratusan) algoritma hashing yang di rancang oleh berbagai ahli bidang komputasi dan matematika. Perbedaan dari satu algoritma dengan algoritma lain adalah dari cara mengacak kata, semakin rumit rumus yang dipakai, semakin bagus algoritma tersebut dan seharusnya juga semakin susah untuk di bobol.

PHP mendukung banyak algoritma hashing. Yang cukup lama beredar adalah function *md5()* dan *sha1()*, yang merupakan implementasi dari algoritma hashing **Message-Digest algortihm 5** (disingkat **MD5**) dan **Secure Hash Algorithm 1** (disingkat **SHA1**).

Berikut contoh penggunaannya:

password_hashing/01.md5_dan_sha1.php

```
1 <?php
2 $password = 'rahasia';
3 $hash_password = md5($password);
4 echo $hash_password; // ac43724f16e9241d990427ab7c8f4228
```

```

5
6 echo '<br>';
7
8 $password = 'rahasia';
9 $hash_password = sha1($password);
10 echo $hash_password; // 829b36babd21be519fa5f9353daf5dbdb796993e

```

Nantinya, isi dari variabel \$hash_password inilah yang kita simpan ke dalam database.

Pada saat proses login, teks hashing ini akan dibandingkan dengan inputan dari user:

password_hashing/02.login_md5.php

```

1 <?php
2 $hash_password = 'ac43724f16e9241d990427ab7c8f4228';
3 $_POST['password'] = 'rahasia';
4
5 if (md5($_POST['password']) === $hash_password) {
6     echo "Password cocok!";
7 } else {
8     echo "Password tidak sesuai";
9 }

```

Hasil kode program:

Password cocok!

Di sini saya menginput string 'rahasia' ke dalam variabel \$_POST['password'], sekedar simulasi bahwa inputan teks ini nantinya akan berasal dari form. Jika hasil fungsi md5(\$_POST['password']) cocok dengan teks hashing yang ada di database, maka artinya user sudah menginput password yang sesuai.

Dengan cara ini, kita bisa membuat proses autentikasi password tanpa menyimpan teks password asli ke dalam database. Yang perlu disimpan adalah teks hasil hashing yang lebih mirip karakter acak.

Namun hashing memiliki satu kelemahan. Meskipun algoritma hashing tidak bisa dibobol dengan cara membalik teks hashing menjadi teks asal, tapi ada trik lain yang bisa dipakai untuk mencari tahu apa teks asli, yakni menggunakan *rainbow table*.

Rainbow table adalah sebutan untuk tabel yang berisi kumpulan pasangan teks asli dan teks hashing, seperti tabel berikut:

Teks Asli	Hasil Hashing MD5
a	0cc175b9c0f1b6a831c399e269772661
aa	4124bc0a9335c27f086f24ba207a4912
aaa	47bce5c74f589f4867dbd57e9ca9f808

Teks Asli	Hasil Hashing MD5
b	92eb5ffee6ae2fec3ad71c777531578f
ba	07159c47ee1b19ae4fb9c40d480856c4
baa	8cdcd79a8dc66aa6c711c9a000b0ac0
...	...

Dengan kecepatan komputer modern, tabel di atas bisa di generate dalam waktu singkat. Terlebih dengan adanya internet, ratusan hingga ribuan komputer bisa saling bekerja sama untuk membuat rainbow table yang berisi puluhan juta baris.

Jika kita memiliki sebuah teks hasil hashing, tinggal melihat apakah teks tersebut ada di tabel atau tidak. Untuk password dengan teks asal yang relatif sederhana seperti 'rahasia', bisa dipastikan sudah ada di dalam rainbow table. Misalnya dengan cara search di google string 'ac43724f16e9241d990427ab7c8f4228', anda bisa langsung temukan bahwa teks tersebut adalah hasil hashing md5 dari 'rahasia'.

Kelemahan dengan rainbow table ini juga berlaku untuk berbagai algoritma hashing yang lebih kompleks, seperti **SHA-256** (Secure Hash Algorithm 256-bit).

Untuk membuat hashing dengan algoritma SHA-256, di PHP kita bisa menggunakan fungsi hash(). Fungsi hash() ini butuh 2 argument, argument pertama berupa string dari nama algoritma yang dipakai dan argument kedua berupa teks asal yang akan dicari nilai hash-nya.

Berikut contoh dari penggunaan fungsi hash() untuk algoritma SHA-256:

password_hashing/03.sha256.php

```

1 <?php
2 $password = 'rahasia';
3 $hash_password = hash('sha256',$password);
4 echo $hash_password;
5 // 541e984103d4099bb8383050c56d511e733d85e6ab889a1c363ced651762eee0

```

Algoritma SHA-256 menghasilkan teks hashing dengan panjang karakter 64 digit.

Fungsi hash() sendiri mendukung banyak algoritma lain. Daftar lengkap dari algoritma hash yang didukung oleh PHP bisa dilihat dengan menjalankan fungsi hash_algos(). Fungsi hash_algos() akan mengembalikan array daftar algoritma hash:

password_hashing/04.hash_algos.php

```

1 <?php
2 echo "<pre>";
3 print_r(hash_algos());
4 echo "</pre>";

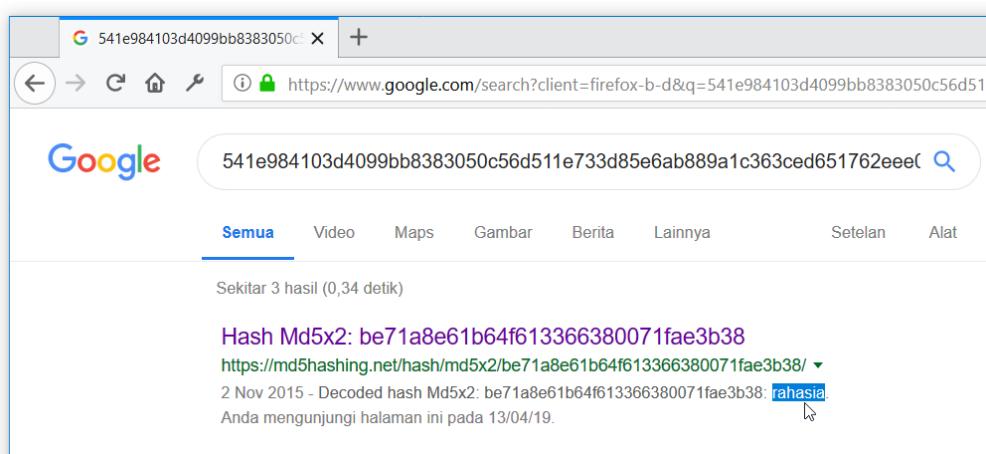
```

Hasil kode program:

```
Array
(
    [0] => md2
    [1] => md4
    [2] => md5
    [3] => sha1
    [4] => sha224
    [5] => sha256
    ...
)
```

Pada saat saja jalankan, total terdapat 51 algoritma hash yang tersedia di PHP.

Kembali ke hasil hash SHA-256 sebelumnya, apakah ini lebih aman dari md5? Berikut hasil pencarian di google:



Gambar: Hasil pencarian teks dari algoritma hash SHA-256

Ternyata teks tersebut tetap ditemukan. Anda bisa mengunjungi web md5hashing.net yang ternyata sudah berisi rainbow table untuk 60 algoritma hash dari teks 'rahasia'.

Jadi, apakah hashing ini tidak berguna?

Tidak juga. **Pertama**, isi dari rainbow table tetap tidak bisa mencari semua kombinasi karakter. Untuk kata-kata yang umum seperti 'rahasia' kemungkinan ada di tabel. Bahkan untuk semua kata yang ada di kamus, besar kemungkinan sudah ada di rainbow table.

Tapi untuk kata yang lebih kompleks dan terdiri dari gabungan angka dan huruf besar, kemungkinan besar tidak ada (atau tepatnya 'belum ada') di rainbow table. Password dengan hashing teks 'rAHAs1a' tidak saya temukan di google, meskipun menggunakan algoritma md5 yang relatif lebih pendek.

Ini pula yang jadi alasan kenapa sewaktu kita mendaftar di sebuah web, kadang diminta untuk memilih password dengan kombinasi angka, huruf besar dan huruf kecil. Kita juga bisa menerapkan teknik yang sama, syarat validasi seperti ini bisa dibuat dengan memanfaatkan pola pemeriksaan *regular expression*.

Kedua, kita bisa memanfaatkan teknik kriptografi yang disebut dengan **salt**. **Salt** adalah karakter atau string tambahan yang dipakai untuk "mengacak" lebih jauh hasil hashing di luar algoritma. Salt ini bisa berbentuk string tambahan yang bersifat tetap atau string acak. Kita akan lihat praktiknya menggunakan string tetap terlebih dahulu.

Penggunaan salt dengan string tetap sebenarnya sangat sederhana, tinggal menambah sebuah string yang sudah dipersiapkan ke dalam teks password. Berikut contohnya:

password_hashing/05.md5_salt.php

```
1 <?php
2 $salt = 'ilkoom';
3 $password = 'rahasia';
4 $hash_password = md5($password.$salt);
5
6 echo $hash_password; // 7a76dee372a15fc60364a2f453dedcb9
```

Di sini saya membuat variabel `$salt` yang berisi string '`ilkoom`'. Fungsi salt di sini hanya sebagai "penyambung" dari password yang akan diinput user. Dalam contoh di atas, string yang di-hash pada baris 4 akan berbentuk `md5('rahasiailkoom')`. Teks '`rahasiailkoom`' ini sudah cukup kompleks karena belum ada di rainbow table.

Untuk proses pemeriksaan, tinggal menyambung kembali hasil inputan user dengan salt:

password_hashing/05.md5_salt.php

```
1 <?php
2 $hash_password = '7a76dee372a15fc60364a2f453dedcb9';
3 $_POST['password'] = 'rahasia';
4
5 $salt = 'ilkoom';
6
7 if (md5($_POST['password'].$salt) === $hash_password) {
8     echo "Password cocok!";
9 } else {
10    echo "Password tidak sesuai";
11 }
```

Hasil kode program:

Password cocok!

Dengan cara ini, teks hasil hash bisa disimpan ke database dengan lebih aman (dibandingkan jika tanpa salt).

Namun bagi yang butuh keamanan tingkat tinggi, ini pun dianggap masih belum mencukupi. Alasannya karena salt yang bersifat tetap, sehingga jika hacker memiliki akses ke kode program, dia bisa mempelajari dan menemukan kalau salt-nya adalah string '`ilkoom`'.

Alternatif lain bisa menggunakan salt yang juga acak, seperti contoh berikut:

password_hashing/07.md5_salt_random.php

```
1 <?php
2 $salt = md5(time());
3 echo $salt;           // 8685446fe26af6022253582ec2900689
4 echo "<br>";
5
6 $password = 'rahasia';
7 $hash_password = md5($password.$salt);
8 echo $hash_password; // b92b786822f91a48fa7d5a2be9095b2e
```

Di baris 2 saya meng-generate salt dengan cara mengambil hasil hash md5 dari fungsi `time()`. Fungsi `time()` sendiri dipakai untuk mengambil waktu sistem server saat ini. Karena hasil dari `time()` selalu berubah-ubah tiap detik, maka hasil hashing-nya juga akan selalu bertukar.

Akibatnya, pemanggilan fungsi hashing di baris 7 juga akan selalu berbeda-beda, meskipun teks password yang diinput sama.

Jika terdapat 2 orang yang menggunakan password 'rahasia', di database akan tersimpan dengan string hash yang berbeda, sehingga relatif lebih susah jika ada yang mencoba melakukan analisis hasil hashing.

Apabila si hacker membaca kode program, dia juga tidak bisa menebak password di database karena salt di generate secara acak. Tidak ada informasi kapan waktu pembuatan hash dilakukan.

Tapi trik ini juga memiliki kelemahan, yakni kita harus ikut menyimpan isi salt ke dalam database. Karena jika tanpa salt, proses pemeriksaan password juga tidak bisa dilakukan. Sebagai contoh, berikut cara pemeriksaan password menggunakan salt yang tersimpan:

password_hashing/08.login_md5_salt_random.php

```
1 <?php
2 $hash_password = 'b92b786822f91a48fa7d5a2be9095b2e';
3 $salt = '8685446fe26af6022253582ec2900689';
4 $_POST['password'] = 'rahasia';
5
6 if (md5($_POST['password'].$salt) === $hash_password) {
7     echo "Password cocok!";
8 } else {
9     echo "Password tidak sesuai";
10 }
```

Hasil kode program:

Password cocok!

Jadi, bagaimana cara terbaik untuk menyimpan password?

Jika password tersebut secara khusus dipakai untuk proses login, maka PHP sudah menyediakan solusinya, yakni menggunakan fungsi `password_hash()` dan `password_verify()`.

Fungsi `password_hash()` akan meng-generate hasil hash sepanjang 60 karakter (ke depannya bisa lebih). Fungsi ini butuh 2 buah argument, berupa string asal yang akan di hash, serta konstanta dari jenis algoritma yang akan dipakai. Berikut contoh penggunaannya:

`password_hashing/09.func_password_hash.php`

```
1 <?php
2 $password = 'rahasia';
3 $hash_password = password_hash($password,PASSWORD_DEFAULT);
4
5 echo $hash_password;
6 // $2y$10$huXIj2DJU1x2a1L0Bby380oGBgYxjeCSKMzvMcD76yhRf6kUjAR8m
```

Sebagai argument kedua dari pemanggilan fungsi `password_hash()` di baris 3, saya menggunakan konstanta `PASSWORD_DEFAULT`. Ini artinya kita menyerahkan kepada PHP untuk memilih algoritma terbaik yang tersedia saat ini.

PHP Manual menyatakan bahwa pilihan algoritma ini bisa berubah di kemudian hari. Inilah alasan kenapa jumlah karakter hash hasil fungsi juga bisa berubah. Saat ini jenis algoritma hashing yang dipakai oleh fungsi `password_hash()` adalah algoritma **Blowfish**.

Nilai lain untuk argument kedua ini adalah `PASSWORD_BCRYPT` yang akan menggunakan algoritma Blowfish. Berikut contoh pemanggilannya:

```
$hash_password = password_hash($password,PASSWORD_BCRYPT);
```

Pilihan lain adalah konstanta `PASSWORD_ARGON2I` dan `PASSWORD_ARGON2ID` untuk menggunakan algoritma hashing **Argon2i** dan **Argon2id**.

Yang cukup unik dari hash yang dihasilkan fungsi `password_hash()` adalah, di dalam string tersebut sudah tersimpan jenis algoritma yang dipakai beserta string salt.

Jika anda jalankan kode program di atas beberapa kali, string hashing yang dihasilkan akan terus berganti, padahal password kita tetap berupa teks 'rahasia'. Ini sebagai pembuktian bahwa `password_hash()` menerapkan salt acak secara internal.

Selain salt, karakter hashing ini juga berisi informasi mengenai algoritma yang dipakai. Untuk algoritma **Blowfish**, 7 karakter pertama hasil hashing diawali dengan `$2y$10$`. Informasi ini akan berguna pada saat pemeriksaan hasil hash.

Sekarang, bagaimana cara memeriksa password hash ini?

Kita tidak bisa menggunakan teknik yang sama seperti sebelumnya, karena hasil hash dari fungsi `password_hash()` akan selalu berubah-ubah. Untuk keperluan ini PHP menyediakan fungsi `password_verify()`.

Fungsi `password_verify()` butuh 2 buah argument, pertama berupa string password, dan kedua berupa serta teks hasil hash (yang biasanya akan di ambil dari database). Fungsi `password_verify()` mengembalikan boolean `true` jika password cocok, atau boolean `false` jika

password tidak cocok. Berikut contoh penggunaannya:

password_hashing/10.func_password_verify.php

```
1 <?php
2 $hash_password = '$2y$10$pcX4UWwJ2UpipTXSgY0Z8u6T/0/bV2fvNmsGdZEJ82aF7obnMO.NS';
3 $_POST['password'] = 'rahasia';
4
5 if (password_verify($_POST['password'], $hash_password)) {
6     echo "Password cocok!";
7 } else {
8     echo "Password tidak sesuai";
9 }
```

Hasil kode program:

Password cocok!

Inilah teknik penyimpanan dan pemeriksaan password yang akan kita pakai.

Sebagai tambahan, karena di dalam hasil hash sudah terdapat jenis algoritma dan salt, maka jika di kemudian hari PHP menggunakan algoritma baru untuk fungsi `password_hash()`, maka fungsi `password_verify()` tetap bisa memprosesnya.

Dalam penjelasan ini saya banyak menggunakan fungsi hash `md5()`. Tapi saat ini algoritma md5 disarankan untuk **tidak dipakai lagi**. Selain karena hasil hashnya cukup pendek, algoritma ini juga "terlalu terkenal" sehingga banyak tool yang tersedia di internet untuk membobolnya, misalkan dari daftar rainbow table.

Dengan kecepatan komputasi yang ada saat ini, panjang string hasil hash yang hanya 32 karakter dianggap sudah tidak cocok lagi. Jika karena sesuatu hal anda tidak bisa menggunakan fungsi `password_hash()`, bisa memilih algoritma hashing lain yang lebih kuat.

Generate Tabel user

Kembali ke project **Ilkoom Stock Manager**, kita akan mulai dengan menyiapkan tabel user. Tabel user ini sebenarnya sudah pernah saya pakai ketika membahas syarat validasi password untuk class `Validate`.

Tabel user terdiri dari 3 kolom: `username`, `password` dan `email`. Untuk membuat tabel user, silahkan akses file `db_generate_tabel_barang_dan_user.php` di folder `ilkoom_stock_user`.

Ketika proses generate tabel user, sudah langsung di input 1 data dengan kode program sebagai berikut:

ilkoom_stock_user/db_generate_tabel_barang_dan_user.php

```
1 <?php
2 //....
3 $passwordAdmin = password_hash('rahasia',PASSWORD_DEFAULT);
4
5 $query = "INSERT INTO user (username, password, email) VALUES
6 ('admin','$passwordAdmin','admin@gmail.com');";
7
8 $mysqli->query($query);
9 //....
```

Artinya di dalam tabel akan terdapat 1 data user dengan kolom username berisi 'admin', kolom password berisi hasil hash dari 'rahasia' dan kolom email berisi 'admin@gmail.com'.

Jika anda lihat langsung isi dari tabel ini (misalnya menggunakan phpMyadmin), bisa terlihat kolom password berisi teks hash seperti '\$2y\$10\$Lb6tCFVZjDAzgI2my1XTDOGie3.fFgzs6EZLyzDWSdXd7V8B8nxg2'. Tapi tentu tidak sama persis karena hasil hash dari `password_hash()` memiliki salt sehingga akan selalu acak. Pada saat proses login nanti, kita akan pakai fungsi `password_verify()` untuk membandingkan hasil hash ini.

Registrasi User: register_user.php

Halaman pertama yang akan kita rancang untuk tabel user adalah proses registrasi. Silahkan buat file baru dengan nama `register_user.php` dan tempatkan di folder yang sama dengan aplikasi kita sebelumnya.

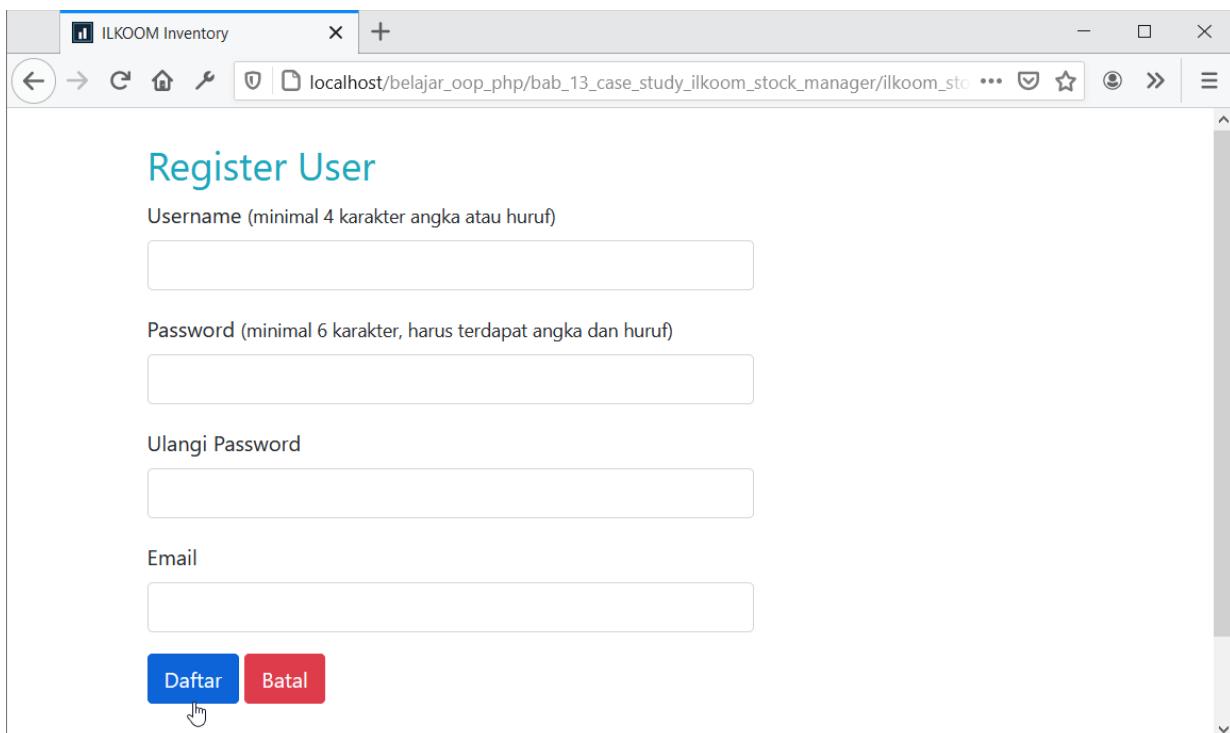
Berikut kode program dari halaman `register_user.php`:

ilkoom_stock_user/register_user.php

```
1 <?php
2 // jalankan init.php (untuk autoloader)
3 require 'init.php';
4
5 // buat object user yang akan dipakai untuk proses input
6 $user = new User();
7
8 if (!empty($_POST)) {
9     // jika terdeteksi form $_POST di submit, jalankan proses validasi
10    $pesanError = $user->validasiInsert($_POST);
11    if (empty($pesanError)) {
12        // jika tidak ada error, proses insert user
13        $user->insert();
14        header('Location:register_berhasil.php');
15    }
16 }
17 ?>
18 <!doctype html>
19 <html lang="id">
20   <head>
21     <meta charset="utf-8">
```

```
22     <meta name="viewport" content="width=device-width, initial-scale=1,
23     shrink-to-fit=no">
24     <title>ILKOOM Inventory</title>
25     <link rel="icon" href="img/favicon.png" type="image/png">
26     <link rel="stylesheet" href="css/bootstrap.css">
27     <link rel="stylesheet" href="css/style.css">
28   </head>
29   <body>
30
31   <div class="container">
32     <div class="row">
33       <div class="col-12 col-md-8 col-lg-6 py-4">
34         <h1 class="h2 me-auto"><a class="text-info" href="register_user.php">
35           Register User</a></h1>
36
37         <?php
38           // jika ada error, tampilkan pesan error
39           if (!empty($pesanError)):
40         ?>
41
42         <div id="divPesanError">
43           <div class="mx-auto">
44             <div class="alert alert-danger" role="alert">
45               <ul class="mb-0">
46                 <?php
47                   foreach ($pesanError as $pesan) {
48                     echo "<li>$pesan</li>";
49                   }
50                 ?>
51                 </ul>
52             </div>
53           </div>
54         </div>
55
56         <?php
57           endif;
58         ?>
59
60         <!-- Form untuk proses insert -->
61         <form method="post">
62
63           <div class="mb-3">
64             <label for="username" class="form-label">Username</label>
65             <small> (minimal 4 karakter angka atau huruf) </small>
66             <input type="text" class="form-control" name="username" id="username"
67             value="<?php echo $user->getItem('username'); ?>">
68           </div>
69
70           <div class="mb-3">
71             <label for="password" class="form-label">Password</label>
72             <small> (minimal 6 karakter, harus terdapat angka dan huruf) </small>
73             <input type="password" class="form-control" name="password"
74             id="password">
75           </div>
76
```

```
77 <div class="mb-3">
78     <label for="ulangi_password" class="form-label">
79         Ulangi Password</label>
80         <input type="password" class="form-control" name="ulangi_password"
81             id="ulangi_password">
82     </div>
83
84     <div class="mb-3">
85         <label for="email" class="form-label">Email</label>
86         <input type="text" class="form-control" name="email" id="email"
87             value=<?php echo $user->getItem('email'); ?>">
88     </div>
89     <input type="submit" class="btn btn-primary" value="Daftar">
90     <a href="login.php" class="btn btn-danger">Batal</a>
91
92 </form>
93
94     </div>
95 </div>
96 </div>
97
98 <?php
99 // include footer
100 include 'template/footer.php';
101 ?>
```



Gambar: Tampilan halaman register_user.php

Halaman di atas baru bisa tampil setelah kita membuat class `User` (akan di bahas setelah ini). Tanpa class `User`, hasilnya berupa pesan `Fatal error: Uncaught Error: Class 'User' not`

found.

Halaman `register_user.php` terdiri dari sebuah form dengan 4 inputan. Saya kembali menggunakan komponen form Bootstrap untuk membuat design tampilan. Di bagian bawah form terdapat 2 buah tombol: **Daftar** dan **Batal**.

Jika tombol **Daftar** di klik, isian form akan di validasi dan jika lolos, data baru akan ditambah ke tabel `user`. Sedangkan jika tombol **Batal** di klik, halaman akan pindah ke `login.php`. Halaman `login.php` sendiri memang belum tersedia dan akan kita buat setelah ini. Nantinya, halaman `register_user.php` juga bisa diakses dari halaman login.

Masuk ke kode program, di baris 3 terdapat perintah `require 'init.php'` untuk proses *autoload*. Ini diperlukan karena dalam halaman `register_user.php` kita akan mengakses beberapa class tambahan yang terdapat di folder `class`.

Di baris 6 saya mengisi variabel `$user` dengan hasil instansiasi dari class `User`. Class `User` ini nantinya berisi berbagai method yang akan "mengurusi" user. Mengenai isi dari class `User` akan kita bahas secara bertahap karena di setiap halaman perlu method yang berbeda-beda.

Di baris 8 terdapat pemeriksaan kondisi `if (!empty($_POST))`. Sampai di sini saya yakin anda sudah bisa membaca pola penulisan kode program yang saya pakai. Sama seperti sebelumnya, kondisi ini akan bernilai **true** jika halaman `register_user.php` di re-load dengan sebuah form. Ini terjadi jika form di bagian bawah sudah diisi dan user men-klik tombol **Daftar**.

Pada saat form di submit, maka perintah di baris 10 – 15 akan dijalankan. Di baris 10 saya memanggil method `$user->validasiInsert($_POST)`. Method ini dipakai untuk proses validasi. Hasil pesan error akan dikirim ke dalam variabel `$pesanError`. Kembali, method ini memang belum tersedia dan akan kita buat setelah ini.

Jika ternyata `$pesanError` tidak berisi nilai, yang artinya semua inputan form lolos validasi, maka jalankan method `$user->insert()` di baris 13. Method ini berfungsi untuk proses input data ke database. Setelah itu, user akan di-redirect ke halaman `register_berhasil.php` melalui fungsi `header()` di baris 14.

Dalam penjelasan ini terdapat banyak pemanggilan method dari class `User`. Semuanya akan kita buat sebentar lagi, termasuk tentang halaman `register_berhasil.php` yang juga belum tersedia.

Lanjut, di baris 18 sampai 29 adalah kode HTML untuk bagian header. Saya tidak memakai file `header.php` karena halaman register ini diakses sebelum proses login.

Di baris 31 – 58 terdapat kode HTML dan sedikit kode PHP untuk menampilkan pesan error. Kode ini sama seperti yang selama ini kita gunakan.

Kemudian di baris 61 – 92 merupakan kode HTML untuk membuat form register. Terdapat 4 buah inputan form: `username`, `password`, `ulangi_password`, dan `email`. Dalam inputan `username`

dan email, saya menambah atribut `value` dengan nilai yang diambil dari method `$user->getItem()`. Method ini dipakai untuk proses re-populate isian form. Ketika terjadi kesalahan validasi maka kedua inputan tidak kembali kosong.

Atas alasan keamanan, inputan `password` dan `ulangi_password` tidak saya tambah atribut `value`. Kedua inputan ini tetap kosong setiap kali terjadi kesalahan validasi.

Terakhir terdapat pemanggilan `include 'template/footer.php'` di baris 100.

Class User: Method `validasiInsert()`, `insert()` dan `getItem()`

Sama seperti saat pembuatan CRUD tabel barang, sebagian besar pemrosesan pada halaman `register_user.php` di handle oleh class terpisah, yakni class **User**.

Class **User** inilah yang akan mengurusi semua hal terkait user, mulai dari proses validasi, pengambilan data dari database, menginput data ke database, update data, hingga pembuatan session untuk proses login.

Isi dari class **User** akan kita buat secara bertahap. Khusus untuk halaman `register_user.php`, terdapat 3 pemanggilan method milik class **User**, yakni `validasiInsert()`, `insert()` dan `getItem()`. Inilah yang akan kita bahas kali ini.

Sebagian besar teknik yang saya pakai dalam class **User** akan sama seperti class **Barang**. Kode program untuk pembuatan class **User** juga ada di folder **class** dengan nama file `User.php`. Silahkan anda buat file ini terlebih dahulu. Berikut struktur dasar dari class **User**:

ilkoom_stock_user/class/User.php

```
1 <?php
2 class User{
3     private $_db = null;
4     private $_formItem = [];
5 }
```

Class **User** hanya memiliki 2 buah private property, yakni `$_db` yang nantinya dipakai untuk menampung object dari class **DB**, serta `$_formItem` yang akan dipakai untuk menampung inputan form hasil proses sanitizing. Class **User** ini saya rancang tidak memiliki constructor.

Kita masuk ke method pertama, yakni `validasiInsert()`:

ilkoom_stock_user/class/User.php

```
1 public function validasiInsert($formMethod){
2     $validate = new Validate($formMethod);
3
4     $this->_formItem['username'] = $validate->setRules('username', 'Username', [
5         'sanitize' => 'string',
6         'required' => true,
7         'min_char' => 4,
8         'regexp' => '/^A-Za-z0-9]+$/',
9     );
10 }
```

```

9      'unique' => ['user', 'username'],
10     ]);
11
12     $this->_formItem['password'] = $validate->setRules('password', 'Password', [
13         'sanitize' => 'string',
14         'required' => true,
15         'min_char' => 6,
16         'regexp' => '/[A-Za-z]+[0-9]+[0-9]+[A-Za-z]/'
17     ]);
18
19     $this->_formItem['ulangi_password'] =
20         $validate->setRules('ulangi_password', 'Ulangi password', [
21             'sanitize' => 'string',
22             'required' => true,
23             'matches' => 'password'
24         ]);
25
26     $this->_formItem['email'] = $validate->setRules('email', 'Email', [
27         'sanitize' => 'string',
28         'required' => true,
29         'email' => true
30     ]);
31
32     if(!$validate->passed()) {
33         return $validate->getError();
34     }
35 }

```

Method `validasiInsert()` dipakai untuk... well, proses validasi insert. Saya menggunakan nama seperti ini karena nantinya juga terdapat proses validasi lain, seperti `validasiLogin()` dan `validasiUbahPassword()`.

Prinsip kerja dari method `validasiInsert()` sangat mirip seperti proses validasi untuk form barang. Yakni berisi kumpulan syarat validasi yang akan dipakai untuk memanggil method `setRules()` milik class `Validate`.

Method `validasiInsert()` butuh sebuah parameter berupa global variable inputan form, yakni salah satu dari `$_POST` atau `$_GET`. Parameter ini disimpan ke dalam variabel `$formMethod` dan menjadi nilai untuk proses instansiasi class `Validate` di baris 2.

Antara baris 4 sampai 30 terdapat syarat validasi untuk ke-4 inputan form. Semua syarat validasi ini sudah kita bahas pada bab sebelumnya. Hasil dari pemanggilan method `setRules()` akan disimpan ke dalam private property `$this->_formItem`. Artinya property `$formItem` akan berisi semua inputan form yang sudah melewati proses sanitizing,

Syarat validasi untuk inputan `username` adalah: harus di sanitize dengan tipe data string, tidak boleh kosong, minimal berisi 4 karakter, memenuhi pola regexp `'/[A-Za-z0-9]+$/'`, serta harus unik dengan cara melihat isi dari tabel `barang` yang ada di database.

Dua syarat terakhir ini perlu sedikit penjelasan. Pola *regular expression* `'/[A-Za-z0-9]+$/'`

artinya, inputan `username` hanya bisa menerima karakter alfanumerik (huruf alphabet + angka), minimal 1 karakter. Namun jumlah karakter ini sudah tertutupi oleh syarat '`min_char`' => 4. Sehingga total digit untuk `username` haruslah minimal 4 karakter.

Dengan syarat *regular expression* ini, `username` tidak bisa diisi karakter selain huruf dan angka. Username dengan nama '`anto$$`' tidak akan lolos validasi karena mengandung karakter dollar '\$'. Begitu juga dengan '`#agung`' yang juga tidak lolos karena memiliki karakter '#'

Syarat selanjutnya, yakni '`unique`' => `['user', 'username']` dipakai untuk memastikan setiap `username` harus unik, tidak boleh ada nama `username` yang sama. Setiap kali `username` di submit, langsung cari ke tabel `user` di database apakah terdapat nama yang sama atau tidak. Jika ada, maka tampilkan pesan error dan minta user memilih nama lain.

Berikutnya untuk isian form `password` memiliki syarat: harus di sanitize dengan tipe data string, tidak boleh kosong, minimal 6 karakter dan memenuhi pola *regular expression* `/[A-Za-z]+[0-9]|[0-9]+[A-Za-z]/`. Pola ini terlihat cukup rumit, fungsinya untuk memastikan `password` memiliki minimal 1 huruf dan 1 angka.

Password seperti '`qwerty`' tidak akan lolos validasi karena tidak memiliki angka, begitu juga dengan password '`643123`'. User harus menggunakan password seperti '`qwerty9`', atau '`qw3rty`'. Intinya harus terdapat minimal 1 angka dan 1 huruf di dalam password.

"Pemaksaan" password seperti ini semata-mata agar data user lebih aman, tapi tidak jarang juga bikin kesal karena user biasanya menggunakan password yang sama untuk berbagai web. Keputusan apakah menggunakan aturan semacam ini perlu anda pertimbangkan terlebih dahulu, karena ada sisi plus dan minusnya.

Untuk inputan `ulangi_password`, syaratnya adalah: melewati proses sanitizing bertipe string, tidak boleh kosong dan harus sama dengan nilai inputan `password`, yakni syarat '`matches`' => '`password`'.

Terakhir, inputan `email` memiliki syarat: melewati proses sanitizing bertipe string, tidak boleh kosong dan harus memenuhi pola email.

Anda bisa coba tes dengan berbagai data untuk menguji syarat validasi yang sudah di rancang. Mulai dari `username` yang memiliki karakter khusus, menginput nama '`admin`' (yang sudah ada di tabel), memilih password yang kurang dari 6 karakter, memilih email yang salah, dst.

The screenshot shows a web browser window titled "ILKOOM Inventory". The URL in the address bar is "localhost/belajar_oop_php/bab_13_case_study_ilkoom_stock_manager/ilkoom_stock_use". The main content is a form titled "Register User". A red box contains validation errors:

- Pola Username tidak sesuai
- Pola Password tidak sesuai
- Ulangi password tidak sama
- Format Email tidak sesuai

The form fields are as follows:

- Username (minimal 4 karakter angka atau huruf): rissa\$
- Password (minimal 6 karakter, harus terdapat angka dan huruf): (displayed as five dots)
- Ulangi Password: (displayed as five dots)
- Email: rissa@gmail

At the bottom are two buttons: "Daftar" (highlighted with a blue background) and "Batal".

Gambar: Tampilan halaman register_user.php dengan pesan error validasi

Baik, kita lanjut ke pembuatan method kedua, yakni `getItem()`:

ilkoom_stock_user/class/User.php

```
1 public function getItem($item){  
2     return isset($this->_formItem[$item]) ? $this->_formItem[$item] : '';  
3 }
```

Pada dasarnya, method ini sama persis dengan method `getItem()` dari class `Barang`. Isinya berupa perintah untuk mengembalikan nilai inputan form yang tersimpan di dalam private property `$this->_formItem`, atau mengembalikan string kosong jika ternyata nilainya belum terdefinisi.

Berikutnya kita akan lihat method `insert()` dari class `User`. Bisa di tebak bahwa fungsi dari method ini adalah untuk menginput nilai form ke database. Berikut kode program yang dibutuhkan:

ilkoom_stock_user/class/User.php

```
1 public function insert(){  
2     $this->_db = DB::getInstance();  
3     $newUser = [  
4         'username' => $this->getItem('username'),  
5         'password' => password_hash($this->getItem('password'),PASSWORD_DEFAULT),
```

```
6     'email' => $this->getItem('email')
7 ];
8     return $this->_db->insert('user', $newUser);
9 }
```

Di baris 2 terdapat perintah instansiasi class DB. Untuk class User ini saya memutuskan membuat object class DB di dalam method yang membutuhkan saja, tidak di dalam constructor sebagaimana yang kita pakai dalam class Barang.

Alasannya adalah karena di class User akan terdapat cukup banyak method yang tidak butuh database. Jadi daripada selalu membuat object class DB setiap kali class User di-instansiasi (jika ditempatkan ke dalam constructor) maka saya memindahkan proses instansiasi class DB ke setiap method yang membutuhkan saja.

Lanjut, di baris 3 saya membuat variabel \$newUser yang berisi associative array untuk setiap nama kolom. Nilai kolom username dan password bisa langsung diambil dengan method getItem().

Namun khusus untuk password, harus dilewaskan terlebih dahulu ke fungsi password_hash(). Karena seperti yang sudah kita bahas, yang disimpan ke database haruslah hasil hash dari password, bukan langsung password asli.

Terakhir, variabel \$newUser dipakai sebagai argument untuk pemanggilan method insert() milik class DB di baris 8.

Itulah 3 buah method diperlukan untuk halaman register_user.php, nantinya class User ini akan kita tambah dengan method lain sesuai dengan kebutuhan setiap halaman.

Pemberitahuan Register: register_berhasil.php

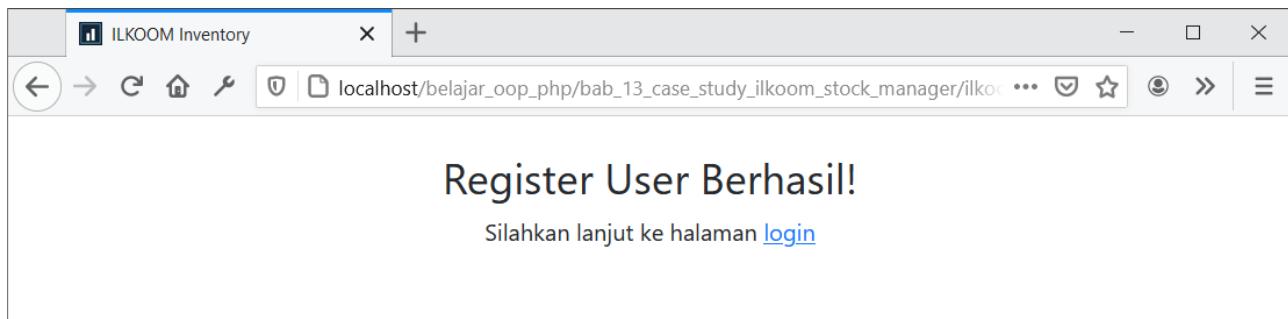
Ketika proses register berhasil, maka user akan di redirect ke register_berhasil.php.

Halaman ini hanya berisi teks pemberitahuan saja:

ilkoom_stock_user/register_berhasil.php

```
1 <!doctype html>
2 <html lang="id">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1,
6       shrink-to-fit=no">
7     <title>ILKOOM Inventory</title>
8     <link rel="icon" href="img/favicon.png" type="image/png">
9     <link rel="stylesheet" href="css/bootstrap.css">
10    <link rel="stylesheet" href="css/style.css">
11  </head>
12  <body>
13
14    <div class="container" class="py-5">
15      <div class="row">
```

```
16 <div class="col-12 py-4 mx-auto">
17   <h1 class="h2 text-center">Register User Berhasil!</h1>
18   <p class="text-center">Silahkan lanjut ke halaman
19     <a href="login.php">login</a>
20   </p>
21 </div>
22 </div>
23 </div>
24
25 </body>
26 </html>
```



Gambar: Tampilan halaman register_berhasil.php

Halaman ini hanya berisi dari kode HTML biasa. Setelah pemberitahuan teks "Register User Berhasil!", terdapat sebuah link menuju halaman login.php. Inilah yang akan kita rancang berikutnya.

Login User: login.php

Halaman login.php berisi inputan form username dan password untuk proses login. Jika kedua data sesuai dengan yang ada di database, maka user bisa lanjut untuk mengelola isi tabel barang. Namun kalau inputan username dan password tidak cocok, tampilkan pesan error.

Berikut kode program lengkap dari login.php:

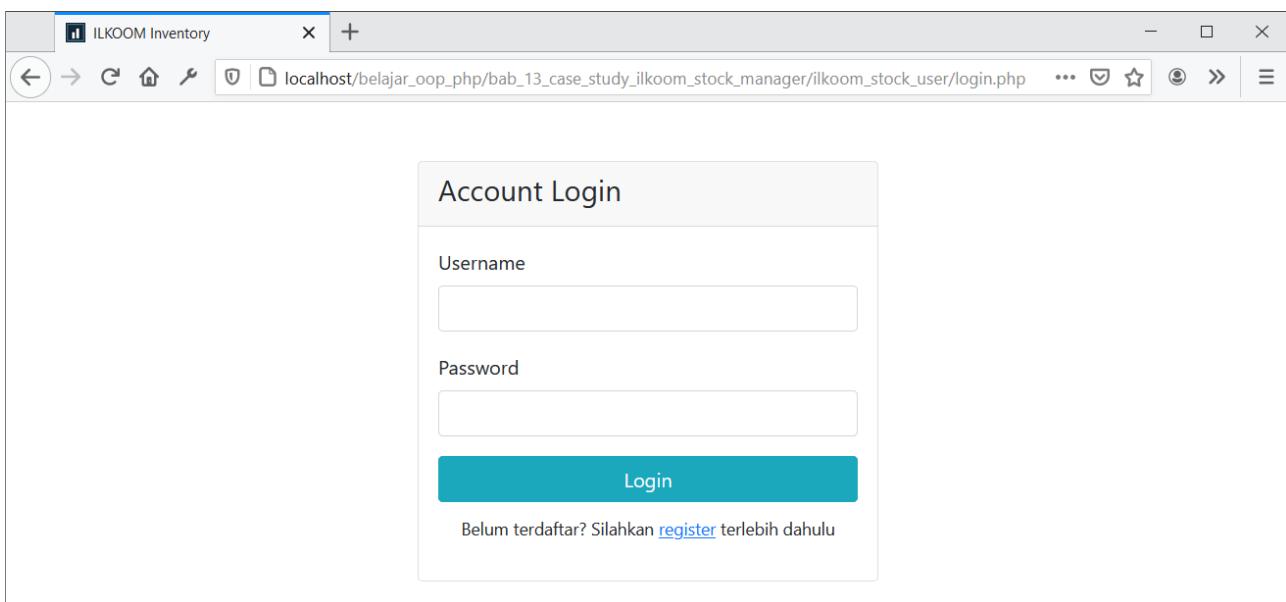
```
1 <?php
2 // jalankan init.php (untuk autoloader)
3 require 'init.php';
4
5 // buat object user yang akan dipakai untuk proses login
6 $user = new User();
7
8 if (!empty($_POST)) {
9   // jika terdeteksi form $_POST di submit, jalankan proses validasi
10  $pesanError = $user->validasiLogin($_POST);
11  if (empty($pesanError)) {
12    // jika tidak ada error, proses login user
13    $user->login();
14  }
15 }
```

```

16 ?>
17 <!doctype html>
18 <html lang="id">
19   <head>
20     <meta charset="utf-8">
21     <meta name="viewport" content="width=device-width, initial-scale=1,
22       shrink-to-fit=no">
23     <title>ILKOOM Inventory</title>
24     <link rel="icon" href="img/favicon.png" type="image/png">
25     <link rel="stylesheet" href="css/bootstrap.css">
26     <link rel="stylesheet" href="css/style.css">
27   </head>
28   <body>
29     <div class="container pt-5">
30
31     <?php
32     if (!empty($pesanError)) :
33     ?>
34
35     <div class="row">
36       <div class="col-10 col-sm-8 col-md-6 col-lg-5 col-xl-4 mx-auto">
37         <div class="alert alert-danger" role="alert">
38           <ul class="mb-0">
39             <?php
40               foreach ($pesanError as $val) {
41                 echo "<li>$val</li>";
42               }
43             ?>
44             </ul>
45           </div>
46         </div>
47     </div>
48
49     <?php
50     endif;
51     ?>
52
53     <div class="row">
54       <div class="col-10 col-sm-8 col-md-6 col-lg-5 col-xl-4 mx-auto">
55         <div class="card">
56           <div class="card-header">
57             <h4>Account Login</h4>
58           </div>
59
60           <div class="card-body">
61             <form method="post" autocomplete="off" >
62               <div class="mb-3">
63                 <label for="username" class="form-label">Username</label>
64                 <input type="username" class="form-control"
65                   name="username" id="username"
66                   value="<?php echo $user->getItem('username'); ?>" >
67               </div>
68               <div class="mb-3">
69                 <label for="password" class="form-label">Password</label>
70                 <input type="password" class="form-control"

```

```
71          name="password" id="password">
72      </div>
73      <div class="d-grid">
74          <input type="submit" class="btn btn-info text-white"
75          value="Login">
76      </div>
77  </form>
78  <p class="mt-2 text-center">
79      <small class="text-center">Belum terdaftar? Silahkan
80          <a href="register_user.php">register</a> terlebih dahulu
81      </small>
82  </p>
83 </div>
84
85      </div>
86  </div>
87 </div>
88
89 </div>
90
91  <script src="js/bootstrap.bundle.js"></script>
92 </body>
93 </html>
```



Gambar: Tampilan halaman login.php

Setelah perintah `require 'init.php'` di baris 3, dilanjutkan dengan pembuatan object dari class `User` di baris 6. Kemudian di baris 8 terdapat pemeriksaan kondisi apakah terdeteksi form di-input atau tidak. Jika iya, maka jalankan proses validasi dengan memanggil method `$user->validasiLogin($_POST)` di baris 10.

Setelah semua syarat validasi terlewati, user bisa login dengan cara memanggil method `$user->login()`. Kedua method ini, `validasiLogin()` dan `login()` akan kita tambah ke dalam class `User` sesaat lagi.

Selanjutnya merupakan kode HTML dengan sedikit perintah PHP. Di baris 17 – 27 terdapat kode HTML untuk pembuatan header, di baris 31 – 51 merupakan kode untuk menampilkan pesan error, serta di baris 60 – 83 terdapat kode untuk pembuatan form. Di dalam form ini saya menambah perintah echo \$user->getItem('username') untuk proses re-populate inputan username.

Di bagian bawah form terdapat teks keterangan: 'Belum terdaftar? Silahkan register terlebih dahulu'. Teks 'register' berbentuk sebuah link ke halaman register_user.php. User yang belum terdaftar bisa memilih untuk mendaftarkan diri terlebih dahulu.

Class User: Method validasiLogin() dan login()

Dalam halaman login.php terdapat 2 kali pemanggilan method baru milik class User, yakni validasiLogin() dan login().

Sesuai dengan namanya, method validasiLogin() berguna untuk proses validasi inputan form login. Berikut kode program yang diperlukan:

ilkoom_stock_user/class/User.php

```

1  public function validasiLogin($formMethod){
2      $validate = new Validate($formMethod);
3
4      $this->_formItem['username'] = $validate->setRules('username', 'Username', [
5          'sanitize' => 'string',
6          'required' => true
7      ]);
8
9      $this->_formItem['password'] = $validate->setRules('password', 'Password', [
10         'sanitize' => 'string',
11         'required' => true
12     ]);
13
14     if(!$validate->passed()) {
15         return $validate->getError();
16     } else {
17         $this->_db = DB::getInstance();
18         $this->_db->select('password');
19         $result = $this->_db->getWhereOnce('user', ['username', '=', 
20             $this->_formItem['username']]);
21
22         if(empty($result) || !password_verify($this->_formItem['password'],
23             $result->password)) {
24             $pesanError[] = 'Maaf, username / password salah';
25             return $pesanError;
26         }
27     }
28 }
```

Sama seperti proses validasi lain, validasi login juga tetap dibuat dengan cara memanggil method setRules() milik class Validate(). Syarat untuk kedua inputan ini hanya harus di

sanitize dengan tipe data string serta tidak boleh kosong.

Hasil validasi kemudian disimpan ke dalam property `$this->_formItem` yang nantinya bisa dipakai untuk proses re-populate form atau untuk mengakses database.

Jika terdapat validasi yang tidak terpenuhi, maka kembalikan pesan error dan method berhenti di baris 15. Namun jika lolos validasi, maka kode program di baris 17 – 26 akan dijalankan.

Setelah lolos validasi, di baris 17 saya mengisi variabel `$this->_db` dengan hasil instansiasi class DB. Kemudian di baris 18 terdapat pemanggilan method `$this->_db->select('password')` yang berfungsi untuk menentukan kolom tabel yang akan diambil. Pengambilan tabel itu sendiri dilakukan dengan method `$this->_db->getWhereOnce()` di baris 19 – 20.

Perintah di baris 18 – 20 akan menjalankan query MySQL sebagai berikut:

```
SELECT password FROM user WHERE username = $this->_formItem['username']
```

Yang bisa dibaca: 'ambil isi kolom password dari tabel user, dengan syarat kolom username harus sama nilainya dengan yang diinput user'.

Dengan demikian, variabel `$result` akan berisi nilai hash dari password username yang tersimpan di database, atau jika username yang dicari tidak ada, variabel `$result` ini akan berisi boolean **false**.

Di baris 22 terdapat pemeriksaan kondisi **if** yang bisa dibaca: "apakah isi variabel `$result` kosong **ATAU** string hash password di database tidak sama dengan hasil hash password dari user?", jika salah satu kondisi ini terpenuhi, yakni username tidak ada di database atau password salah, maka proses validasi gagal dan isi pesan error '*'Maaf, username / password salah'* ke variabel `$pesanError`.

Kode program untuk kondisi perbandingan ini memang cukup rumit, karena saya langsung membuat proses pemeriksaan password ke dalam operasi perbandingan OR.

Kembali sejenak ke halaman `login.php`, jika variabel `$pesanError` tidak berisi apa-apa, maka artinya lolos proses validasi dan user bisa masuk dengan cara memanggil method `login()`. Berikut kode yang diperlukan:

ilkoom_stock_user/class/User.php

```
1 public function login(){
2     $_SESSION["username"] = $this->getItem('username');
3     header("Location: tampil_barang.php");
4 }
```

Isi dari method ini hanya 2 baris. Di baris 2 saya membuat variabel session `$_SESSION["username"]` yang diisi dengan nilai username dari variabel `$this->getItem('username')`.

Sebagai contoh, jika username yang dipakai ketika saat login adalah 'rissa', maka perintah di

baris 2 akan diproses sebagai `$_SESSION["username"] = 'rissa'`. Kemudian user langsung di re-direct ke halaman `tampil_barang.php`.

Proteksi Halaman

Sampai di sini kita sudah membuat mekanisme login, tapi setiap halaman dari tabel barang tetap bisa diakses langsung. Misalnya dengan mengetik di web browser, halaman `tampil_barang.php` tetap bisa diakses tanpa login sekalipun. Kita perlu membuat langkah proteksi sehingga hanya user yang sudah login saja yang bisa mengakses.

Prinsip kerjanya adalah, periksa apakah terdapat session username di variabel `$_SESSION`. Jika ada, maka artinya user sudah login. Jika tidak ada, maka user belum login dan tampilkan pesan error. Dalam contoh ini saya hanya akan me-redirect user ke halaman `login.php` jika session tidak ditemukan.

Untuk membuat sistem seperti ini, kita harus men-edit beberapa file.

Pertama, karena akan mengakses session, maka di bagian atas semua file PHP perlu memanggil fungsi `session_start()`. Fungsi ini dibutuhkan agar kita bisa mengakses variabel `$_SESSION`.

Karena fungsi `session_start()` akan dipanggil dari banyak halaman, saya akan menginputnya ke dalam file `init.php`. Dengan demikian, semua file yang memiliki perintah `require 'init.php'` juga langsung menjalankan fungsi `session_start()`.

Berikut modifikasi dari halaman `init.php`:

ilkoom_stock_user/init.php

```
1 <?php
2 session_start();
3
4 spl_autoload_register(function ($className) {
5     $path = "class/{$className}.php";
6     if (file_exists($path)) {
7         require $path;
8     } else {
9         die ("File $path tidak tersedia");
10    }
11});
```

Tambahannya ada di baris 2, yakni penambahan baris `session_start()`, sedangkan sisa kode program sama persis dengan fungsi `spl_autoload_register()` yang kita pakai pada saat pembuatan CRUD tabel barang.

Langkah proteksi **kedua** adalah dengan membuat pemeriksaan session username di setiap halaman yang hanya bisa diakses setelah login. Halaman tersebut adalah:

- `tampil_barang.php`
- `tambah_barang.php`
- `hapus_barang.php`
- `edit_barang.php`

Untuk ke-4 file ini, tambah kode berikut di bagian paling atas, setelah pemanggilan file `init.php`:

```
1 $user = new User();
2 $user->cekUserSession();
```

Yup, proses pemeriksaan session akan saya tempatkan ke dalam method `cekUserSession()` dari class `User`. Tentang isi dari method ini akan kita bahas sesaat lagi.

Sebagai contoh, berikut kode dari bagian atas file `tampil_barang.php`:

ilkoom_stock_user/tampil_barang.php

```
1 <?php
2 // jalankan init.php (untuk session_start dan autoloader)
3 require 'init.php';
4
5 // cek apakah user sudah Login atau belum
6 $user = new User();
7 $user->cekUserSession();
8
9 // buat koneksi ke database
10 $DB = DB::getInstance();
11
12 ...
13 ...
```

Dengan tambahan perintah di baris 6 – 7, maka method `cekUserSession()` akan selalu dijalankan sebelum kode di baris 8 dst.

Dan berikut isi dari method `cekUserSession()` di dalam class `User`:

ilkoom_stock_user/class/User.php

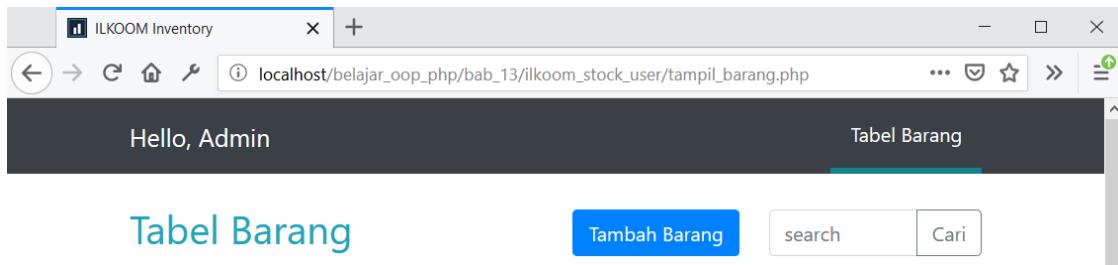
```
1 <?php
2 public function cekUserSession(){
3     if (!isset($_SESSION["username"])) {
4         header("Location: login.php");
5     }
6 }
```

Method ini cukup sederhana, cukup dengan memeriksa apakah variabel `$_SESSION["username"]` terdefinisi atau tidak. Jika tidak, maka re-direct user ke halaman `login.php`.

Alternatifnya, anda juga bisa membuat sebuah file HTML baru yang berisi informasi bahwa user harus login terlebih dahulu, kurang lebih sama seperti halaman `register_berhasil.php`

yang pernah kita buat.

Tambahan **ketiga** adalah membuat menu navigasi baru di bagian atas halaman CRUD tabel barang. Sebelumnya, menu ini hanya berisi teks '**Hello, Admin**' dan '**Tabel Barang**', seperti gambar berikut:



Gambar: Tampilan bagian menu CRUD tabel barang sebelumnya

Saya ingin di sisi kiri atas tidak lagi '**Hello, Admin**', tapi berisi '**Hello <username>**'. Username yang dimaksud adalah user yang login saat ini, misalnya 'Hello, Rissa', atau 'Hello Alex'.

Kemudian di sisi kanan saya ingin menambah menu '**My Profile**' dan '**Logout**'. Menu My Profile dipakai untuk menampilkan data user, dan menu Logout dipakai untuk keluar dari aplikasi. Untungnya, menu ini berada di satu file saja, yakni `header.php`.

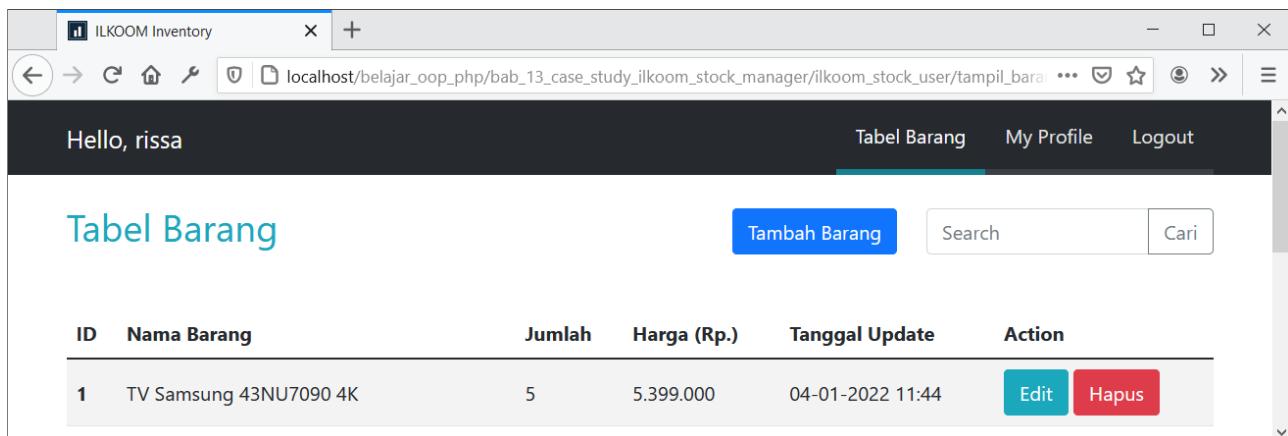
Berikut modifikasi dari file `header.php`:

`ilkoom_stock_user/template/header.php`

```
1  <!doctype html>
2  <html lang="id">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1,
6      shrink-to-fit=no">
7      <title>ILKOOM Inventory</title>
8      <link rel="icon" href="img/favicon.png" type="image/png">
9      <link rel="stylesheet" href="css/bootstrap.css">
10     <link rel="stylesheet" href="css/style.css">
11 </head>
12 <body>
13
14 <!-- NAVBAR -->
15 <nav id="main-navbar" class="navbar navbar-expand-md navbar-dark
16     bg-dark py-0">
17     <div class="container">
18         <span class="navbar-brand">
19             Hello, <?php echo $_SESSION["username"]; ?>
20         </span>
21         <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
22             data-bs-target="#navbarNav" >
23             <span class="navbar-toggler-icon"></span>
24         </button>
25
26         <div class="collapse navbar-collapse" id="navbarNav">
```

```

27 <ul class="navbar-nav ms-auto">
28   <li class="nav-item">
29     <a class="nav-link p-3 <?php echo basename($_SERVER['PHP_SELF'])>
30       == "tampil_barang.php" ? "active" : "" ; ?>" href="tampil_barang.php">
31       Tabel Barang</a>
32   </li>
33   <li class="nav-item">
34     <a class="nav-link p-3 <?php echo basename($_SERVER['PHP_SELF'])>
35       == "profile.php" ? "active" : "" ; ?>" href="profile.php">
36       My Profile</a>
37   </li>
38   <li class="nav-item">
39     <a class="nav-link p-3" href="logout.php">Logout</a>
40   </li>
41 </ul>
42 </div>
43 </div>
44 </nav>
```



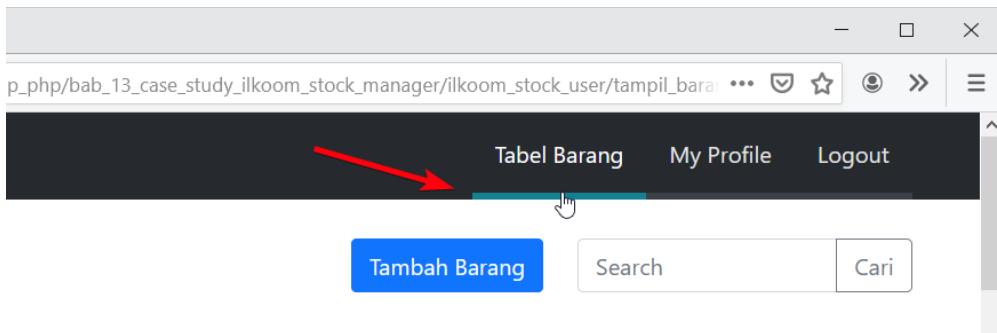
Gambar: Tampilan bagian menu setelah penambahan

Sebelumnya halaman header.php hanya terdiri dari kode HTML, sekarang terdapat penambahan kode PHP agar lebih dinamis.

Untuk membuat pesan 'Hello <username>', cukup mengambil data username ini dari \$_SESSION["username"], yang di proses oleh perintah di baris 18.

Tambahan menu navigasi ada di baris 33- 37 untuk '**My Profile**', dan di baris 38 – 40 untuk menu '**Logout**'. Keduanya dibuat dari tag yang berisi link <a>, ini semua merupakan aturan bawaan dari komponen **navbar** Bootstrap.

Menggunakan kode CSS di file style.css, saya membuat sedikit efek ketika menu ini di hover, yakni ketika cursor mouse berada di atas menu Ketika di hover, akan tampil garis bawah berwarna biru kehijauan.



Gambar: Efek garis bawah dari menu

Efek ini sebenarnya sudah ada sejak pembuatan aplikasi CRUD tabel barang, tapi belum terlihat karena saat itu hanya ada 1 menu, yakni "Tabel Barang".

Saya ingin efek garis bawah ini bisa permanen ketika kita berada di halaman yang sedang terbuka. Misalnya sekarang kita berada di halaman `tampil_barang.php`, maka menu "**Tabel Barang**" akan memiliki efek garis bawah meskipun tidak sedang di hover. Atau jika nanti halaman yang terbuka adalah `profile.php`, maka menu "**My Profile**" - lah yang akan memiliki efek garis bawah.

Di dalam komponen **navbar** Bootstrap, efek seperti ini bisa dibuat dengan cara menambah class `.active` ke dalam tag `<a>` menu.

Jika sebelumnya ditulis sebagai:

```
<a class="nav-link p-3" href="tampil_barang.php">Tabel Barang</a>
```

Maka diubah menjadi:

```
<a class="nav-link p-3 active" href="tampil_barang.php">Tabel Barang</a>
```

Masalahnya, class `.active` ini harus ikut berpindah dari satu menu ke menu lain. Salah satu cara adalah membuat menu yang berbeda untuk setiap halaman. Pada studi kasus di akhir buku Bootstrap Uncover, saya memakai teknik yang sama.

Namun cara manual ini tidak efisien, karena kita harus men-copy paste menu ke setiap halaman. Jika ada perubahan atau penambahan menu baru, maka setiap halaman harus diubah. Apakah bisa membuat 1 menu untuk semua halaman? Bisa, tapi kita butuh sedikit bantuan dari kode PHP.

Menggunakan PHP, kita bisa membuat kondisi logika memanfaatkan konstanta PHP: `$_SERVER['PHP_SELF']`. Konstanta ini akan menghasilkan nama file yang sedang di proses. Namun nama file yang dihasilkan adalah nama path, seperti: `/belajar_oop_php/bab_13/ilkoom_stock_user/tampil_barang.php`. Kita tidak butuh nama ini, yang diperlukan hanya nama file saja, yakni `tampil_barang.php`.

Untuk mengambil potongan nama file dari sebuah path, PHP juga sudah menyediakan fungsi

khusus, yakni `basename()`. Jika file yang terbuka saat ini adalah `tampil_barang.php`, maka hasil dari `basename($_SERVER['PHP_SELF'])` adalah `tampil_barang.php`.

Berikutnya, kita akan membuat kondisi if, yakni jika halaman saat ini sesuai dengan nama menu, maka tambah class `.active`. Kalau tidak sesuai, maka tidak perlu menampilkan apa-apa.

Agar penulisan kondisi if lebih singkat, saya menulisnya menggunakan operator ternary :

```
1 <li class="nav-item">
2   <a class="nav-link p-3 <?php echo basename($_SERVER['PHP_SELF'])>
3   == "tampil_barang.php" ? "active" : "" ; ?>" href="tampil_barang.php">
4   Tabel Barang</a>
5 </li>
6 <li class="nav-item">
7   <a class="nav-link p-3 <?php echo basename($_SERVER['PHP_SELF'])>
8   == "profile.php" ? "active" : "" ; ?>" href="profile.php">
9   My Profile</a>
10 </li>
```

Karena kode PHP ini berada di dalam atribut class, maka otomatis menjadi salah satu dari nama class CSS. Sebagai contoh, jika file yang dibuka saat ini adalah `profile.php`, maka kondisi di baris 2 akan menghasilkan **false**, sehingga hasilnya string kosong "", namun untuk kondisi di baris 7 hasilnya adalah **true**, sehingga hasilnya string 'active', menjadi:

```
1 <li class="nav-item">
2   <a class="nav-link p-3 active" href="profile.php">
3   My Profile</a>
4 </li>
```

Operator ternary ini memang singkat tapi agak susah dipahami bagi yang jarang menggunakannya. Kondisi perbandingan:

```
echo basename($_SERVER['PHP_SELF']) == "tampil_barang.php" ? "active" : "";
```

bisa dikonversi dalam bentuk if else sebagai berikut:

```
if (basename($_SERVER['PHP_SELF']) == "tampil_barang.php"){
    echo "active";
}
else {
    echo "";
}
```

Menggunakan teknik ini, class `active` akan berpindah antar setiap menu tergantung halaman yang dibuka. Dan ini tetap berlaku jika seandainya ada tambahan menu lain. Khusus untuk menu **Logout**, tidak perlu diberi class `active` karena jika menu ini di klik, halaman akan langsung keluar dari aplikasi.

Dengan ini proteksi halaman sudah selesai. Halaman CRUD barang hanya bisa diakses jika user telah login sesuai dengan data yang ada di database.

Logout User: logout.php

Halaman berikutnya yang akan kita buat adalah `logout.php`. Sesuai dengan namanya, halaman ini dipakai untuk proses **logout** dengan cara men-klik menu **Logout** dari sisi kanan menu.

Berikut kode programnya:

ilkoom_stock_user/logout.php

```
1 <?php
2 require 'init.php';
3
4 $user = new User();
5 $user->logout();
```

Isi file ini tidak lain hanya membuat object dari class `User` lalu memanggil method `logout()`. Kesannya memang agak repot, kenapa tidak langsung saja membuat isi dari method `logout`? Ini agar kita disiplin bahwa semua hal yang berkaitan dengan user, harus berada di class `User`.

Class User: Method logout()

Prinsip dari proses logout adalah hapus isi dari `$_SESSION["username"]`. Ini bisa dilakukan dengan beberapa cara, salah satunya menggunakan fungsi `unset()`:

ilkoom_stock_user/class/User.php

```
1 public function logout(){
2     unset($_SESSION["username"]);
3     header("Location: login.php");
4 }
```

Isi dari method `logout()` hanya 2 baris, yakni menghapus variabel `$_SESSION["username"]` dengan fungsi `unset($_SESSION["username"])`, kemudian me-redirect user ke halaman `login.php`.

Dengan demikian ketika menu **Logout** di klik, user akan langsung pindah ke halaman Login. Alternatifnya anda bisa juga membuat halaman khusus yang berisi pesan "Anda telah logout".

Sampai di sini, struktur dasar dari proses authentikasi kita sudah selesai, mulai dari proses register, login hingga logout. Sebagai pemanis terakhir, kita akan rancang satu halaman lagi untuk proses menampilkan data user dan proses ubah password.

Menampilkan Data User: profile.php

Halaman `profile.php` saya rancang untuk menampilkan data user yang saat ini sedang login. Namun data user sendiri tidak banyak, hanya username, password dan email. Oleh karena itu di halaman `profile` ini saya juga menambahkan form untuk proses perubahan password.

Untuk mengubah password, user harus menginput 3 buah password:

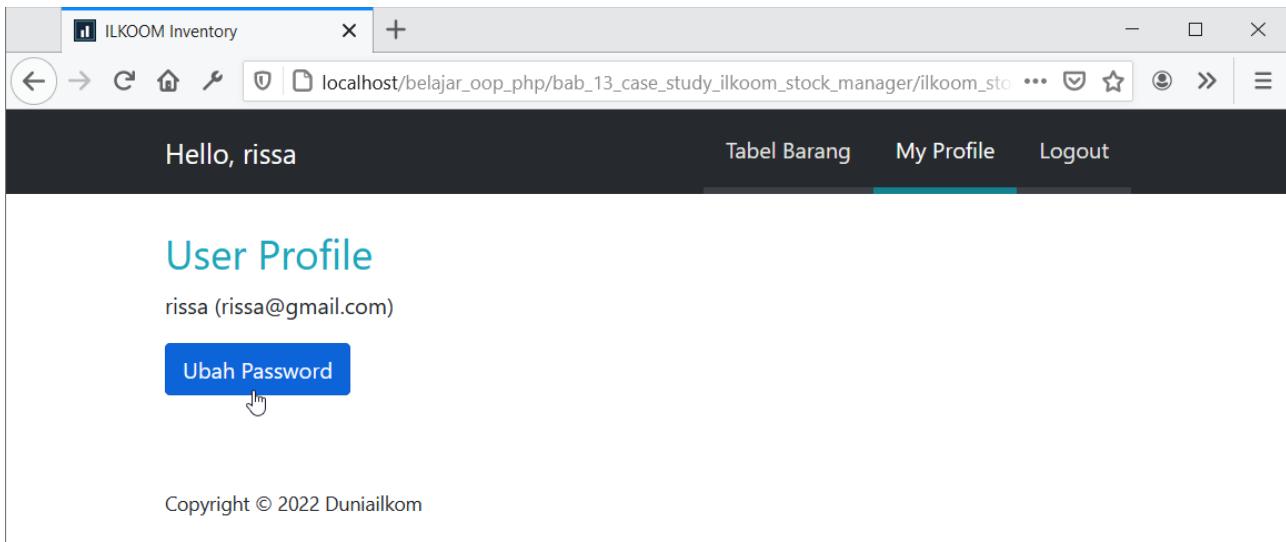
- ✓ Password lama (sesuai dengan yang ada di database)
- ✓ Password baru
- ✓ Ulangi password baru

Berikut kode yang dibutuhkan untuk halaman profile.php:

ilkoom_stock_user/profile.php

```
1 <?php
2 // jalankan init.php (untuk session_start dan autoloader)
3 require 'init.php';
4
5 // cek apakah user sudah login atau belum
6 $user = new User();
7 $user->cekUserSession();
8
9 // ambil semua data user yang akan diupdate dari database
10 $user->generate($_SESSION["username"]);
11
12 if (!empty($_POST)) {
13     // jika terdeteksi form $_POST di submit, jalankan proses validasi
14     $pesanError = $user->validasiUbahPassword($_POST);
15     if (empty($pesanError)) {
16         // jika tidak ada error, proses update password
17         $user->ubahPassword();
18         header('Location:ubah_password_berhasil.php');
19     }
20 }
21
22 // include head
23 include 'template/header.php';
24 ?>
25
26 <div class="container">
27     <div class="row">
28         <div class="col-12 col-sm-10 col-md-8 col-lg-6 py-4">
29             <h1 class="h2 me-auto">
30                 <a class="text-info" href="edit_barang.php">User Profile</a>
31             </h1>
32
33             <?php
34                 // jika ada error, tampilkan pesan error
35                 if (!empty($pesanError)):
36             ?>
37
38             <div id="divPesanError">
39                 <div class="mx-auto">
40                     <div class="alert alert-danger" role="alert">
41                         <ul class="mb-0">
42                             <?php
43                                 foreach ($pesanError as $pesan) {
44                                     echo "<li>$pesan</li>";
45                                 }
46                             ?>
```

```
47      </ul>
48      </div>
49  </div>
50 </div>
51
52 <?php
53   endif;
54 ?>
55
56 <!-- Form untuk proses update -->
57 <p>
58   <?php echo $user->getItem('username')." (".$user->getItem('email').")"; ?>
59 </p>
60
61 <p>
62   <button class="btn btn-primary" type="button" data-bs-toggle="collapse"
63     data-bs-target="#formPassword">Ubah Password</button>
64 </p>
65
66 <form method="post" id="formPassword" class="collapse"
67 <?php if (!empty($_POST)) { echo "show"; }?>">
68
69   <div class="mb-3">
70     <label for="password_lama" class="form-label">Password Lama</label>
71     <input type="password" class="form-control" name="password_lama"
72       id="password_lama">
73   </div>
74
75   <div class="mb-3">
76     <label for="password_baru" class="form-label">Password Baru</label>
77     <small> (minimal 6 karakter, harus terdapat angka dan huruf) </small>
78     <input type="password" class="form-control" name="password_baru"
79       id="password_baru">
80   </div>
81
82   <div class="mb-3">
83     <label for="ulangi_password_baru" class="form-label">
84       Ulangi Password Baru</label>
85     <input type="password" class="form-control" name="ulangi_password_baru"
86       id="ulangi_password_baru">
87   </div>
88
89   <input type="submit" class="btn btn-primary" value="Update">
90
91 </form>
92
93   </div>
94 </div>
95 </div>
96
97 <?php
98 // include footer
99 include 'template/footer.php';
100 ?>
```



Gambar: Tampilan halaman profile.php

Halaman di atas baru bisa tampil setelah membuat method `generate()` untuk class `User` (akan di bahas setelah ini). Tanpa method tersebut, hasilnya berupa pesan *Fatal error*:
`Uncaught Error: Call to undefined method User::generate()`

Ketika dijalankan pertama kali, halaman `profile.php` menampilkan judul **User Profile** yang diikuti nama <`username`>(<`email`>), serta terdapat sebuah tombol "**Ubah Password**".

Saat tombol "Ubah Password" di klik, akan tampil 3 buah inputan form di bagian bawah:

A screenshot of the same web browser window as before, showing the "User Profile" page. The "Ubah Password" button has been clicked, revealing a form with three input fields: "Password Lama", "Password Baru (minimal 6 karakter, harus terdapat angka dan huruf)", and "Ulangi Password Baru". Below the inputs is a blue "Update" button. The rest of the page content, including the header and footer, remains the same.

Gambar: Tampilan halaman profile.php dengan form ubah password

Untuk membuat efek form yang muncul seperti ini, saya memanfaatkan komponen **Collapse** bawaan Bootstrap. Form ini juga akan di-validasi, dan jika lolos, password user diubah dengan password baru.

Masuk ke kode program, seperti biasa di bagian atas terdapat perintah `require 'init.php'`, yang diikuti dengan pembuatan class `User` serta pemeriksaan apakah user sudah login atau belum.

Di baris 10 saya memanggil method `$user->generate($_SESSION["username"])`. Method ini belum ada di dalam class `User` dan akan kita buat sesaat lagi. Fungsinya adalah mengambil semua info tentang user (yang saat ini sedang login) dari database. Kurang lebih sama seperti fungsi method `generate()` milik class `Barang`.

Di baris 12 terdapat pemeriksaan kondisi, jika halaman di load dengan sebuah form, yang artinya user akan menukar password, maka jalankan method `$user->validasiUbahPassword($_POST)` untuk proses validasi.

Hanya jika lolos validasi, method `$user->ubahPassword()` dijalankan dan user akan di redirect ke halaman `ubah_password_berhasil.php`.

Bagian kode PHP ini ditutup dengan pemanggilan `include 'template/header.php'` di baris 23.

Selanjutnya antara baris 33 – 54 merupakan kode program untuk menampilkan pesan kesalahan validasi. Diikuti dengan perintah PHP untuk menampilkan nama username dan email. Data ini diambil dari method `$user->getItem('username')` dan `$user->getItem('email')` di baris 58.

Pada baris 61 – 64 terdapat tag `<button>` yang dipakai untuk menampilkan isian form dibagian bawah. Ketika halaman `profile.php` dijalankan pertama kali, form ini tidak terlihat, hanya ketika tombol **Ubah Password** di klik, barulah form tampil. Ini semua menggunakan komponen collapse bawaan Bootstrap.

Kunci dari komponen collapse ini ada di atribut `data-bs-toggle="collapse"` dan `data-bs-target="#formPassword"`. Ketika tombol ini di klik, sebuah element yang memiliki atribut `id="formPassword"` akan muncul atau tersembunyi. Secara internal, efek seperti ini dibuat menggunakan JavaScript oleh kode program Bootstrap.

Form dengan atribut `id="formPassword"` ada di baris 66 – 91. Isinya terdiri dari kode HTML untuk membuat 3 tag `<input type="password">` dengan nama `password_lama`, `password_baru`, dan `ulangi_password_baru`.

Di baris 67 terdapat sedikit kode PHP yang dipakai untuk menambah class CSS "show" hanya apabila terdeteksi form sudah diinput, yakni jika kondisi `if (!empty($_POST))` bernilai **true**.

Hal ini di perlukan karena secara bawaan dari Bootstrap, komponen collapse (yang dalam contoh kita berbentuk form), akan selalu tersembunyi ketika di-load. Dengan tambahan kondisi ini, form akan tetap terbuka jika terdapat error validasi.

Terakhir terdapat kode `include 'template/footer.php'` di baris 99 untuk membuat bagian footer.

Class User: Method logout()

Di dalam halaman `profile.php` terdapat 3 kali pemanggilan method baru dari class User, yakni:

- `generate()`
- `validasiUbahPassword()`
- `ubahPassword()`

Method `generate()` dipakai untuk mengambil data user dari database. Berikut kode programnya:

`ilkoom_stock_user/class/User.php`

```
1 public function generate($username){  
2     $this->_db = DB::getInstance();  
3     $result = $this->_db->getWhereOnce('user',[ 'username' , '=' , $username ]);  
4     foreach ($result as $key => $val) {  
5         $this->_formItem[$key] = $val;  
6     }  
7 }
```

Method ini memiliki sebuah parameter `$username` berupa data username yang akan diambil dari database. Dalam halaman `profile.php`, data username ini berasal dari session `$_SESSION["username"]`.

Selanjutnya di baris 2 terdapat perintah untuk membuat object dari class DB. Object ini dipakai untuk memanggil method `getWhereOnce('user',['username' , '=' , $username])`. Method ini akan diproses sebagai:

```
SELECT * FROM user WHERE username = $username
```

Hasil dari query ini disimpan ke dalam variabel `$result`. Sampai di sini, variabel `$result` berisi object dari sebuah array.

Sebagai contoh, jika user yang saat ini login adalah 'rissa', maka hasil pemanggilan method `getWhereOnce()` akan menjalankan query:

```
SELECT * FROM user WHERE username = 'rissa'
```

Dan variabel `$result` akan berisi object berikut:

```
object(stdClass)#7 (3) {  
    ["username"]=> string(5) "rissa"  
    ["password"]=> string(60) "$2y$10$mgKuTaW5vCfzFBXgZ1Ht/uM8lI54N1u7i...."  
    ["email"]=> string(15) "rissa@gmail.com"  
}
```

Ketiga nilai ini akan diambil dan diinput ke dalam property `$this->_formItem` menggunakan perulangan `foreach` di baris 4 – 6. Setelah pemanggilan ini, private property `$_formItem` akan berisi data berikut:

```
$_formItem["username"]=> string(5) "rissa"
$_formItem["password"]=> string(60) "$2y$10$mgKuTaW5vCfzFBXgZ1Ht/uM8lI54N1u7i...."
$_formItem["email"]=> string(15) "rissa@gmail.com"
```

Dengan demikian, data dari tabel user sudah berhasil di ambil dari database. Data ini bisa diambil dari class User menggunakan method `getItem()`, seperti yang dipakai dalam halaman `profile.php` di baris 58.

Method berikutnya untuk class User adalah `validasiUbahPassword()`. Sesuai dengan namanya, method ini dipakai untuk membuat syarat validasi dari inputan ubah password. Berikut kode yang diperlukan:

ilkoom_stock_user/class/User.php

```
1  public function validasiUbahPassword($formMethod){
2      $validate = new Validate($formMethod);
3
4      $this->_formItem['password_lama'] = $validate->
5          setRules('password_lama','Password lama',[  

6              'sanitize' => 'string',
7              'required' => true,  

8          ]);
9
10     $this->_formItem['password_baru'] = $validate->
11         setRules('password_baru','Password baru',[  

12             'sanitize' => 'string',
13             'required' => true,
14             'min_char' => 6,
15             'regexp' => '/[A-Za-z]+[0-9]+[0-9]+[A-Za-z]/'  

16         ]);
17
18     $this->_formItem['ulangi_password_baru'] = $validate->
19         setRules('ulangi_password_baru','Ulangi password baru',[  

20             'sanitize' => 'string',
21             'required' => true,
22             'matches' => 'password_baru'  

23         ]);
24
25     if(!$validate->passed()) {
26         return $validate->getError();
27     } else {
28         $this->_db = DB::getInstance();
29         $this->_db->select('password');
30         $result = $this->_db->getWhereOnce('user',[  

31             'username' , '=' ,
32             $this->_formItem['username']]));  

33
34         if(empty($result) || !password_verify($this->_formItem['password_lama'],
35         $result->password)) {
```

```
35     $pesanError[] = 'Maaf, password lama anda tidak sesuai';
36     return $pesanError;
37 }
38 }
39 }
```

Method ini cukup panjang, tapi mayoritas merupakan ulangan dari berbagai proses validasi yang sudah pernah kita buat.

Dari baris 4 – 23 terdapat syarat validasi untuk ketiga inputan password. Pada semua inputan, terdapat syarat harus di sanitize dengan tipe string serta tidak boleh kosong.

Untuk inputan `password_baru`, ditambah dengan syarat minimal 6 karakter dan harus memenuhi pola regular expression `/[A-Za-z]+[0-9]|[0-9]+[A-Za-z]/`. Pola ini sama seperti yang dipakai pada form register user, yakni minimal harus terdapat 1 huruf dan 1 angka. Untuk inputan `ulangi_password_baru`, harus sama isinya dengan inputan `password_baru`.

Jika salah satu syarat validasi tidak terpenuhi, maka method `validasiUbahPassword()` akan selesai di baris 26 dan pesan error tampil di atas inputan form.

User Profile

• Password lama tidak boleh kosong
• Pola Password baru tidak sesuai
• Ulangi password baru tidak sama

rissa (rissa@gmail.com)

Ubah Password

Password Lama

••••••

Password Baru (minimal 6 karakter, harus terdapat angka dan huruf)

••••••

Gambar: Tampilan form ubah password tidak lolos validasi

Apabila syarat validasi terpenuhi, maka kode di baris 28 – 37 akan di jalankan. Kode ini sama persis seperti di method `validasiLogin()`, yakni memeriksa apakah password yang diinput user sama dengan yang ada di database.

Ada 2 kondisi yang harus dipenuhi, yakni username tersebut memang ada di tabel `user`, dan hasil hash `password_lama` harus sama dengan hasil hash yang tersimpan di tabel. Jika salah satu tidak terpenuhi, maka kembali tampilkan pesan error.

Kembali ke halaman `profile.php`, di baris 15 -19 terdapat kondisi seperti ini:

```
1 if (empty($pesanError)) {
2     // jika tidak ada error, proses update password
3     $user->ubahPassword();
4     header('Location:ubah_password_berhasil.php');
5 }
```

Maksudnya, jika semua tahap pemeriksaan sudah terpenuhi, yakni dengan bukti variabel \$pesanError tidak berisi pesan error apapun, maka panggil method ubahPassword() dari class User. Berikut isi dari method tersebut:

ilkoom_stock_user/class/User.php

```
1 public function ubahPassword(){
2     $newUserPassword = [
3         'password' => password_hash($this->getItem('password_baru'),
4                                         PASSWORD_DEFAULT)
5     ];
6     $this->_db->update('user', $newUserPassword, ['username', '=',
7     $this->_formItem['username']]);
8 }
```

Di baris 2 saya membuat variabel \$newUserPassword. Variabel ini berupa associative array yang hanya terdiri dari 1 element, yakni 'password'. Isi element 'password' adalah hasil dari fungsi password_hash() dari inputan password_baru.

Kemudian di baris 6 saya memanggil method update() bawaan class DB dengan \$newUserPassword sebagai data yang akan di update.

Sebagai contoh, jika user rissa ingin menukar password lama dengan password baru 'qwerty1', maka isi dari variabel \$newUserPassword adalah sebagai berikut:

```
['password' => '$2y$10$q/gK1pnAUZk.gdxw10BemeZz0....']
```

Kemudian pemanggilan method update() akan menjalankan query berikut:

```
UPDATE user SET password = '$2y$10$q/gK1pnAUZk.gdxw....' WHERE username = 'rissa'
```

Hasilnya, password untuk user rissa sudah berubah.

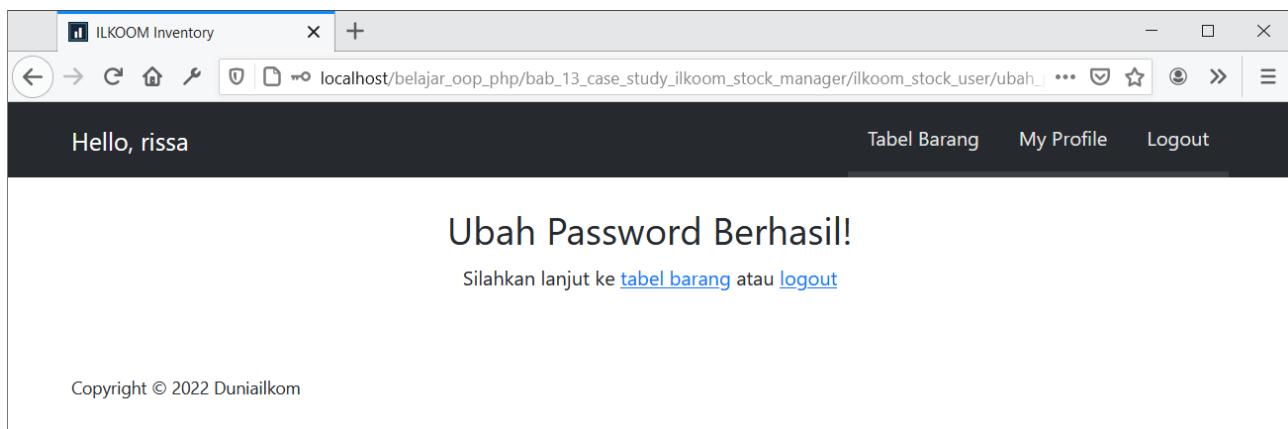
Setelah pemanggilan method ubahPassword(), user akan di redirect ke halaman ubah_password_berhasil.php.

Pemberitahuan Ubah Password: ubah_password_berhasil.php

Halaman ubah_password_berhasil.php tidak lain sekedar pemberitahuan bahwa password user sudah berhasil diubah. Isi dari halaman ini hanya berupa teks saja, dengan kode program sebagai berikut:

ilkoom_stock_user/ubah_password_berhasil.php

```
1 <?php
2 // jalankan init.php (untuk session_start dan autoloader)
3 require 'init.php';
4
5 // include head
6 include 'template/header.php';
7 ?>
8
9 <!-- LOGIN -->
10 <div class="container" class="py-5">
11   <div class="row">
12     <div class="col-12 py-4 mx-auto">
13       <h1 class="h2 text-center">Ubah Password Berhasil!</h1>
14       <p class="text-center">Silahkan lanjut ke <a href="tampil_barang.php">
15         tabel barang</a> atau <a href="logout.php">logout</a></p>
16     </div>
17   </div>
18 </div>
19
20 <?php
21 // include footer
22 include 'template/footer.php';
23 ?>
```



Gambar: Tampilan halaman ubah_password_berhasil.php

Isi halaman ini berupa teks **Ubah Password Berhasil!** Beserta link ke halaman `tampil_barang.php` atau ke `logout.php`.

13.4. Ilkoom Stock Manager: Done!

Dengan selesainya halaman `ubah_password_berhasil.php`, maka aplikasi **Ilkoom Stock Manager** juga sudah selesai.

Pembahasan tentang aplikasi ini sebenarnya sudah dimulai sejak bab tentang class DB dan class Validate, dimana kita merancang sebuah class universal untuk mengakses database serta membuat proses validasi.

Sepanjang pembuatan aplikasi **Ilkoom Stock Manager** saya juga banyak memakai teknik dan trik programming. Tapi ini semua baru sebagian kecil, semoga anda bisa mendapat ide lain dan bisa mengembangkan aplikasi ini lebih jauh lagi.

Sebagai "sentuhan terakhir", saya ingin membuat proses generate database agar lebih *user friendly*. Tidak jarang pembaca buku ingin langsung menjalankan studi kasus bab terakhir sekedar ingin tau seperti apa aplikasi yang akan dipelajari. Dan karena tidak membaca instruksi yang ada di buku, saya sering dihubungi sekedar bertanya bagaimana cara membukanya.

Dengan kode program kita saat ini, database **ilkoom** (beserta tabel **barang** dan **user**) dibuat dengan cara menjalankan file `db_generate_tabel_barang_dan_user.php` secara manual. Maksudnya, file ini harus di ketik di web browser terlebih dahulu. Saya ingin ketika ada yang mencoba mengakses halaman login, aplikasi kita bisa mendeteksi database belum ada dan bertanya apakah user ingin membuatnya terlebih dahulu.

Dari segi keamanan hal ini tentu tidak bagus, karena siapa saja bisa men-generate ulang database setiap saat. Tapi karena kita lebih ke latihan dan bukan membuat project "real world", maka tidak menjadi masalah.

Halaman Welcome: index.php

File `index.php` menjadi pilihan paling pas untuk proses ini. Idenya adalah, jika database tidak terdeteksi maka tampilkan pilihan kepada user apakah ingin men-generate database atau tidak. Jika ternyata database sudah ada, user langsung di re-direct ke halaman **login**.

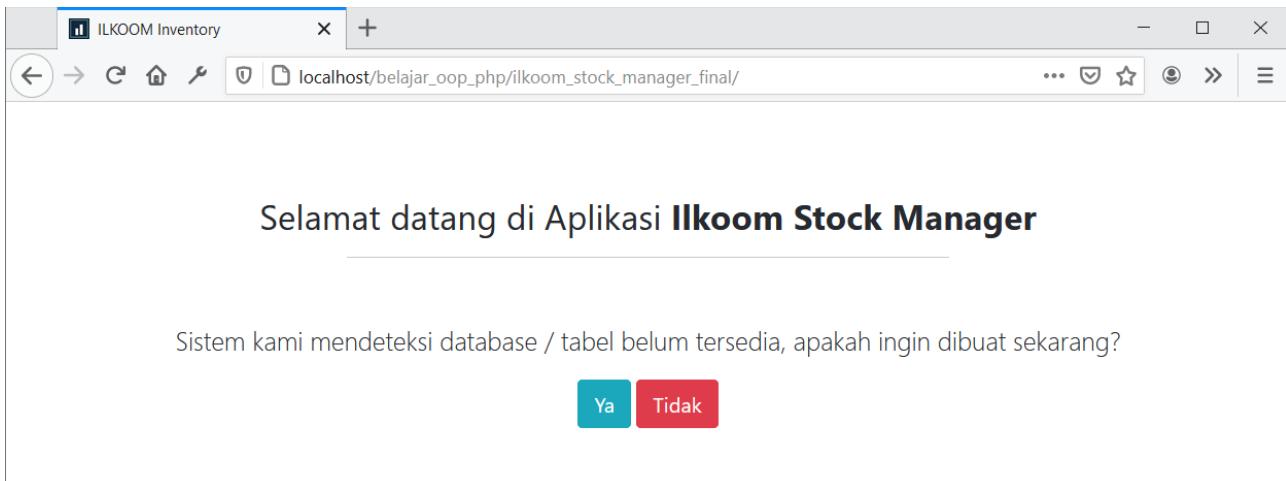
Berikut kode program yang dibutuhkan:

```
ilkoom_stock_manager/index.php

1 <?php
2 mysqli_report(MYSQLI_REPORT_STRICT);
3
4 try {
5     $mysqli = new mysqli("localhost", "root", "");
6
7     // Cek apakah database ilkoom tersedia
8     $mysqli->select_db("ilkoom");
9     if ($mysqli->error){
10         throw new Exception();
11     }
12
13     // Cek apakah tabel barang tersedia
14     $query = "SELECT 1 FROM barang";
15     $mysqli->query($query);
16     if ($mysqli->error){
17         throw new Exception();
18     }
19 }
```

```

20 // Cek apakah tabel user tersedia
21 $query = "SELECT 1 FROM user";
22 $mysqli->query($query);
23 if ($mysqli->error){
24     throw new Exception();
25 }
26
27 // tutup koneksi ke database
28 if (isset($mysqli)) {
29     $mysqli->close();
30 }
31
32 // jika database ilkoom, tabel barang & user ada, redirect ke halaman login
33 header('Location:login.php');
34 }
35 catch (Exception $e) {
36     // kode catch ini akan diproses jika salah satu dari database ilkoom,
37     // tabel barang dan tabel user tidak ada di database.
38 ?>
39
40 <!doctype html>
41 <html lang="id">
42     <head>
43         <meta charset="utf-8">
44         <meta name="viewport" content="width=device-width, initial-scale=1,
45         shrink-to-fit=no">
46         <title>ILKOOM Inventory</title>
47         <link rel="icon" href="img/favicon.png" type="image/png">
48         <link rel="stylesheet" href="css/bootstrap.css">
49         <link rel="stylesheet" href="css/style.css">
50     </head>
51     <body>
52
53     <div class="container" class="py-5">
54         <div class="row">
55             <div class="col-12 py-4 mx-auto text-center">
56                 <h3 class="mt-5">
57                     Selamat datang di Aplikasi <strong>Ilkoom Stock Manager</strong>
58                 </h3>
59                 <hr class="w-50 mx-auto">
60                 <p class="lead mt-5">Sistem kami mendeteksi database /
61                     tabel belum tersedia, apakah ingin dibuat sekarang?</p>
62                 <a href="db_generate_tabel_barang_dan_user.php"
63                     class="btn btn-info text-white">Ya</a>
64                 <a href="#" class="btn btn-danger">Tidak</a>
65             </div>
66         </div>
67     </div>
68
69     <script src="js/bootstrap.bundle.js"></script>
70 </body>
71 </html>
72 <?php
73 // kurung kurawal untuk menutup block catch
74 }
```



Gambar: Tampilan halaman index.php

File ini bernama `index.php`, sehingga akan menjadi halaman default yang tampil pada saat mengakses nama folder `http://localhost/belajar_oop_php/ilkoom_stock_manager/`.

Dalam kode program saya tidak memakai class `DB` tapi mengakses database secara manual menggunakan **mysqli object**. Seluruh pemanggilan mysqli object berada di dalam block **try-catch**.

Setelah membuat koneksi ke database di baris 5, di baris 8 saya menjalankan method `$mysqli->select_db("ilkoom")`. Method ini akan memilih database 'ilkoom' dari daftar database yang ada di MySQL Server. Jika ternyata tidak ditemukan, method `select_db()` menghasilkan pesan error sehingga pemeriksaan kondisi `if ($mysqli->error)` akan bernilai **true**. Jika ini yang terjadi, lempar sebuah exception dengan perintah di baris 10.

Exception yang dilempar hanya sebuah exception kosong, karena yang kita butuhkan hanyalah sebuah cara untuk mengetahui apakah database 'ilkoom' sudah tersedia atau belum.

Ketika sebuah exception dilempar, maka kode program akan langsung lompat ke blok **catch** di baris 35. Dalam blok **catch** ini nantinya terdapat kode HTML untuk memberitahu user kalau database masih belum tersedia.

Pemeriksaan ada atau tidaknya database `ilkom` sebenarnya sudah cukup untuk mendeteksi kalau user baru pertama kali menjalankan aplikasi. Meskipun begitu, saya tetap ingin memastikan bahwa tabel `barang` dan tabel `user` juga sudah ada di dalamnya. Aplikasi kita tidak bisa bekerja tanpa kedua tabel ini.

Oleh karena itu saya menjalankan query "SELECT 1 FROM barang" di baris 14 dan query "SELECT 1 FROM user" di baris 21. Kedua query ini akan menghasilkan error jika salah satu tabel tidak tersedia. Selanjutnya kondisi `if ($mysqli->error)` akan bernilai **true** dan lempar sebuah exception.

Dengan tiga pemeriksaan ini, kita sudah memiliki sebuah mekanisme untuk memastikan apakah salah satu dari database `ilkoom`, tabel `barang` dan tabel `user` sudah ada di database. Jika

salah satunya tidak terpenuhi, program akan langsung lompat ke block **catch**.

Namun jika semua tabel sudah ada di database, kode program akan lanjut dan sampai di baris 33. Di baris ini terdapat perintah `header('Location:login.php')` yang akan me-redirect user ke halaman **Login**.

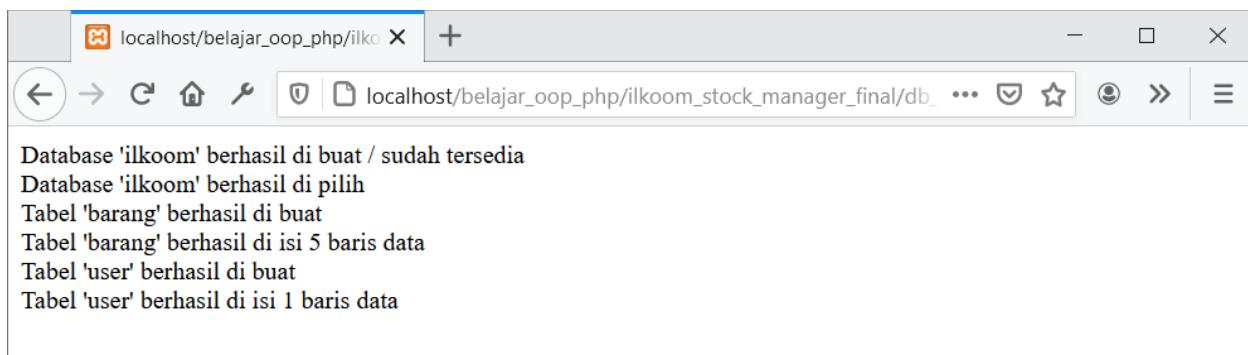
Isi dari block **catch** sendiri hanya terdiri dari kode HTML, yakni berupa pertanyaan '*Sistem kami mendeteksi database / tabel belum tersedia, apakah ingin dibuat sekarang?*', kemudian terdapat 2 buah tombol yang tidak lain merupakan link dari tag `<a>`.

Jika tombol '**Ya**' di klik, user akan pindah ke halaman `db_generate_tabel_barang_dan_user.php`, yakni file PHP yang berisi kode program untuk men-generate database `ilkoom`. Untuk tombol '**Tidak**', tidak berefek apa-apa ketika di klik karena isi dari atribut `href` hanya berupa teks '#'.

Proses Generate Database: `db_generate_tabel_barang_dan_user.php`

Halaman `db_generate_tabel_barang_dan_user.php` terdiri dari kode PHP yang dipakai untuk meng-generate database `ilkoom` beserta tabel `barang` dan `user`.

File ini sudah tersedia sebelumnya dan akan menampilkan teks sederhana mengenai proses urutan pembuatan tabel, dengan hasil sebagai berikut:



Gambar: Tampilan halaman `db_generate_tabel_barang_dan_user.php`

Sekarang saya ingin tambah sedikit kode HTML agar tampilannya jadi lebih menarik:

`ilkoom_stock_manager/db_generate_tabel_barang_dan_user.php`

```
1  <!doctype html>
2  <html lang="id">
3      <head>
4          <meta charset="utf-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1,
6              shrink-to-fit=no">
7          <title>ILKOO Inventory</title>
8          <link rel="icon" href="img/favicon.png" type="image/png">
9          <link rel="stylesheet" href="css/bootstrap.css">
10         <link rel="stylesheet" href="css/style.css">
11     </head>
```

```

12 <body>
13
14 <div class="container" class="py-5">
15   <div class="row">
16     <div class="col-12 py-4 mx-auto text-center">
17       <h3 class="mt-5">Proses Generate Database</h3>
18       <hr class="w-50 mx-auto">
19       <ul>
20
21       <?php
22         mysqli_report(MYSQLI_REPORT_STRICT);
23
24       try {
25         $mysqli = new mysqli("localhost", "root", "");
26
27         // Buat database "ilkoom" (jika belum ada)
28         $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
29         $mysqli->query($query);
30         if ($mysqli->error){
31           throw new Exception($mysqli->error, $mysqli->errno);
32         }
33         else {
34           echo "<li>Database 'ilkoom' berhasil di buat / sudah tersedia</li>";
35         }
36
37         // Pilih database "ilkoom"
38         $mysqli->select_db("ilkoom");
39         if ($mysqli->error){
40           throw new Exception($mysqli->error, $mysqli->errno);
41         }
42         else {
43           echo "<li>Database 'ilkoom' berhasil di pilih</li>";
44         }
45
46         // Hapus tabel "barang" (jika ada)
47         $query = "DROP TABLE IF EXISTS barang";
48         $mysqli->query($query);
49         if ($mysqli->error){
50           throw new Exception($mysqli->error, $mysqli->errno);
51         }
52
53         // Buat tabel "barang"
54         $query = "CREATE TABLE barang (
55           id_barang INT PRIMARY KEY AUTO_INCREMENT,
56           nama_barang VARCHAR(50),
57           jumlah_barang INT,
58           harga_barang DEC,
59           tanggal_update TIMESTAMP
60         )";
61         $mysqli->query($query);
62         if ($mysqli->error){
63           throw new Exception($mysqli->error, $mysqli->errno);
64         }
65         else {
66           echo "<li>Tabel 'barang' berhasil di buat</li>";

```

```

67    };
68
69    // Isi tabel "barang"
70    $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
71    $timestamp = $sekarang->format("Y-m-d H:i:s");
72
73    $query = "INSERT INTO barang
74        (nama_barang, jumlah_barang, harga_barang, tanggal_update) VALUES
75            ('TV Samsung 43NU7090 4K', 5, 5399000, '$timestamp'),
76            ('Kulkas LG GC-A432HLHU', 10, 7600000, '$timestamp'),
77            ('Laptop ASUS ROG GL503GE', 7, 16200000, '$timestamp'),
78            ('Printer Epson L220', 14, 2099000, '$timestamp'),
79            ('Smartphone Xiaomi Pocophone F1', 25, 4750000, '$timestamp')
80        ;";
81    $mysqli->query($query);
82    if ($mysqli->error){
83        throw new Exception($mysqli->error, $mysqli->errno);
84    }
85    else {
86        echo "<li>Tabel 'barang' berhasil di isi ".$mysqli->affected_rows."
87            baris data</li>";
88    }
89
90    // Hapus tabel "user" (jika ada)
91    $query = "DROP TABLE IF EXISTS user";
92    $mysqli->query($query);
93    if ($mysqli->error){
94        throw new Exception($mysqli->error, $mysqli->errno);
95    }
96
97    // Buat tabel "user"
98    $query = "CREATE TABLE user (
99        username VARCHAR(50) PRIMARY KEY,
100       password VARCHAR(255),
101       email VARCHAR(100)
102    )";
103   $mysqli->query($query);
104   if ($mysqli->error){
105       throw new Exception($mysqli->error, $mysqli->errno);
106   }
107   else {
108       echo "<li>Tabel 'user' berhasil di buat</li>";
109   }
110
111   // Isi tabel "user"
112   $passwordAdmin = password_hash('rahasia',PASSWORD_DEFAULT);
113
114   $query = "INSERT INTO user
115       (username, password, email) VALUES
116           ('admin', '$passwordAdmin', 'admin@gmail.com')
117       ;";
118   $mysqli->query($query);
119   if ($mysqli->error){
120       throw new Exception($mysqli->error, $mysqli->errno);
121   }

```

```
122     else {
123         echo "<li>Tabel 'user' berhasil di isi ".$mysqli->affected_rows."  
124             baris data</li>";
125     };
126
127     ?>
128     </ul>
129
130     <hr class="w-50 mx-auto">
131     <p class="lead">Database berhasil dibuat, silahkan <a href="login.php">  
132     Login</a> dengan username: <code>admin</code>, password:  
133     <code>rahasia</code><br>Atau <a href="register_user.php">Register</a>  
134     untuk membuat user baru</p>
135
136     <?php
137     }
138     catch (Exception $e) {
139         echo "<p>Koneksi / Query bermasalah: ".$e->getMessage().  
140             " (".$e->getCode()."")</p>";
141     }
142     finally {
143         if (isset($mysqli)) {
144             $mysqli->close();
145         }
146     }
147     ?>
148     </div>
149     </div>
150     </div>
151
152     <script src="js/bootstrap.bundle.js"></script>
153 </body>
154 </html>
```



Gambar: Tampilan halaman db_generate_tabel_barang_dan_user.php

Dari baris 1 – 19 merupakan kode HTML untuk membuat bagian header, termasuk juga judul "Proses Generate Database" menggunakan tag `<h3>` di baris 17. Kode HTML ini diakhiri dengan tag pembuka `` di baris 19.

Kemudian antara baris 21 – 127 merupakan kode PHP untuk proses pembuatan database dan tabel. Jika setiap query tidak menghasilkan error, maka tampilkan pesan teks menggunakan tag ``, seperti echo "`Tabel 'barang' berhasil dibuat`". Kumpulan tag `` ditutup oleh tag `` di baris 128.

Kemudian di baris 130 – 134 terdapat sedikit kode HTML yang saya pakai untuk membuat teks penjelasan akhir, diantaranya berupa link ke halaman login beserta username default "admin" dan password "rahasia", serta link untuk ke halaman register. User bebas ingin melanjutkan ke halaman yang mana.

Dengan tambahan ini, program kita lebih *user friendly* dan makin mudah dipakai.

Dalam bab ini kita telah membahas tentang pembuatan aplikasi CRUD sederhana: **Ilkoom Stock Manager**. Selama perancangan kode program, kita juga telah menerapkan berbagai teknik pemrograman object seperti membuat class terpisah, serta pemecahan masalah menjadi method-method tertentu. Di luar itu juga banyak teknik pemrograman PHP yang dipakai.

Tidak menutup kemungkinan aplikasi ini bisa dikembangkan lebih lanjut, misalnya menampilkan CRUD untuk beberapa tabel, menambah proses upload gambar profile user, membuat sistem approve sehingga hanya user tertentu saja yang bisa login, dsb.

Terimakasih sudah membeli eBook / buku asli dari DuniaIlkom

Saya menyadari menulis eBook sangat beresiko karena mudah di bajak dan digandakan. Namun ini saya lakukan agar teman-teman (terutama yang di daerah) bisa mendapat materi belajar programming berkualitas dengan harga yang relatif terjangkau.

Proses penulisan buku ini juga tidak sebentar, butuh waktu berbulan-bulan. Mohon kerja sama teman-teman semua untuk tidak menggandakan dan menyebarkan eBook ini.

~ Semoga ilmu yang didapat berkah dan bermanfaat. Terimakasih ~

Penutup OOP PHP Uncover

Tidak terasa, sudah 600 halaman lebih kita membahas konsep pemrograman object untuk PHP. Sedikit banyak semoga anda bisa mendapat gambaran tentang apa itu pemrograman object dan berbagai fitur yang ada di dalamnya.

Di akhir buku kita juga membahas 3 buah studi kasus: pembuatan query builder class DB, pembuatan proses validasi form menggunakan class Validate, serta pembuatan aplikasi CRUD Ilkoom Stock Manager.

Tujuan awal saya merancang buku OOP PHP ini adalah sebagai "jembatan" untuk memasuki materi PHP yang lebih advanced, terutama untuk memakai library dan framework. Hampir semua library dan framework PHP menggunakan konsep pemrograman object.

Jadi, apa selanjutnya?

Jika selama ini anda belajar OOP PHP langsung dari HTML + PHP, maka sudah saatnya pelajari materi dasar lain dari web programming, yakni CSS, MySQL dan JavaScript.

Meskipun telah memutuskan untuk fokus ke back-end, pemahaman materi front-end dasar seperti CSS tetap diperlukan. Sebagai contoh, di studi kasus Ilkoom Stock Manager saya memakai framework CSS **Bootstrap** untuk merancang tampilan. Tanpa ini, design aplikasi kita akan terasa *jadul* dan tidak menarik untuk dilihat.

Materi library JavaScript seperti **jQuery**, **Vue** atau **React** juga bisa jadi alternatif bahan belajar berikutnya, terutama agar bisa memahami konsep **AJAX** (gabungan PHP + JavaScript). Banyak teknik-teknik menarik yang bisa dibuat dengan AJAX.

Dan sebagai tujuan yang paling pas, tentu saja anda bisa lanjut belajar framework PHP. Dua yang paling populer saat ini adalah **Code Igniter** dan **Laravel**. Meskipun ada pilihan framework lain (seperti Symfony, Yii, atau Zend), menurut saya Code Igniter dan Laravel tetap yang paling menonjol. Anda boleh memilih untuk mempelajari salah satu atau keduanya.

Sekedar pengingat, alur belajar pemrograman web yang ideal menurut saya adalah sebagai berikut:

1. Pelajari 5 materi dasar web programming: **HTML**, **CSS**, **PHP**, **MySQL**, dan **JavaScript**.
2. Coba buat project sederhana dengan 5 materi dasar tersebut, minimal aplikasi CRUD.
3. Pelajari **OOP PHP** (pemrograman object), dan buat project sederhana dengan menerapkan OOP PHP beserta semua materi dasar yang sudah dipelajari sebelumnya.

4. Opsional (tidak harus) bisa juga belajar materi tambahan front-end seperti **Bootstrap** dan **jQuery**. Ini juga bisa dipelajari sebelum OOP PHP.
5. Pelajari framework PHP, bisa pilih salah satu antara **Code Igniter** atau **Laravel**.
6. Buat beberapa project menggunakan framework PHP.

Jika sudah sampai ke tahap 6 dan bisa membuat project tanpa harus mengandalkan panduan dari buku / tutorial, menurut saya sudah punya skill yang cukup untuk siap terjun ke dunia kerja

Namun "siap" di sini baru sekedar lolos syarat awal, karena mayoritas lowongan kerja web programming memang mensyaratkan paham salah satu framework PHP.

Saat ini di DuniaIlkom juga sudah tersedia eBook [Laravel Uncover](#) bagi yang ingin lanjut fokus di back-end web programming.

Jangan Berhenti Belajar!

Programming adalah ilmu yang berkembang. Meskipun terasa berat, tapi tidak ada yang namanya "selesai belajar" di dunia IT (terutama programming). Bisa jadi setelah anda menguasai Code Igniter 4, dua tahun berikutnya akan muncul Code Igniter 5. Atau jika saat ini orang masih menggunakan Bootstrap, tahun depan bisa jadi muncul framework CSS yang lebih menarik.

Kunci agar bisa berhasil di dunia IT adalah harus bisa beradaptasi dengan teknologi baru. Dengan dasar yang kuat, seharusnya tidak begitu sulit untuk beralih ke framework baru atau bahkan bahasa pemrograman baru.

Sebagai contoh, jika anda berniat untuk mulai masuk ke pemrograman android, bahasa pemrograman asli (native) di android adalah bahasa **Java**. Bahasa Java merupakan bahasa pemrograman yang sudah menggunakan pemrograman berorientasi objek. Mayoritas konsep OOP yang kita pelajari di buku ini juga bisa diterapkan ke dalam bahasa Java.

Perkembangan dunia JavaScript juga menjadi peluang yang bisa digali. Materi seperti NodeJS, React dan Vue menawarkan tantangan tersendiri. Banyak aplikasi modern yang dibuat dengan teknologi ini.

Namun jika anda memutuskan untuk membeli (dan membaca) semua materi di buku ini, maka saya berpendapat anda ingin mendalami PHP dengan lebih jauh, dan itu tidak salah. Meskipun banyak teknologi baru (terutama dari JavaScript), penggunaan PHP masih mayoritas. Di banyak group dan forum programmer selalu saja ada yang postingan tentang lowongan kerja bagi programmer PHP.

Akhir kata, semoga materi yang ada di buku OOP PHP Uncover ini bisa bermanfaat. Mohon maaf jika ada kata-kata yang salah.

Terima kasih juga atas dukungannya dengan membeli versi asli buku **OOP PHP Uncover** (bukan dari sumber bajakan). Donasi pembelian buku ini menjadi penyemangat saya untuk bisa terus berkarya.

Sampai jumpa di buku DuniaIlkom selanjutnya, semoga ilmu yang di dapat bisa berkah dan bermanfaat :)

Daftar Pustaka

Sepanjang penulisan buku OOP PHP Uncover, saya mengumpulkan bahan dari berbagai sumber. Anda bisa mengunjungi daftar pustaka ini untuk menambah pengetahuan seputar PHP:

- PHP Manual: <http://php.net/manual/en/index.php>
- PHP Delusion: <https://phpdelusions.net>
- Stackoverflow: <https://stackoverflow.com>
- Websitebeaver: <https://websitebeaver.com>
- Mmtuts: <https://www.youtube.com/user/TheCharmefis>
- Traversy Media: <https://www.youtube.com/user/TechGuyWeb>
- Stitcher.io Programming: <https://stitcher.io>