



LEMBAR KERJA MAHASISWA

POLITEKNIK POS INDONESIA

PRODI : D3 SISTEM INFORMASI
TAHUN AKADEMIK : 2020/2021
POKOK BAHASAN : DESIGN PATTERN
BAGIAN 1.2

DOSEN PENGAMPU : MUBASSIRAN, S.SI., M.T.
MATA KULIAH : PEMROGRAMAN BERORIENTASI OBJEK
KODE : LK-5-PBO-MUB-GANJIL-20/21

NAMA MAHASISWA : Muhammad Akbar NPM : 2193013	NILAI
--	-------

N O	INSTRUKSI	HASIL KERJA
1	Buat Folder dengan nama LK-5-PBO-NPM (contoh LK-5-PBO-2193039)	<div><div></div> LK-3-PBO-2193013 <div></div> LK-4-PBO-2193013 <div></div> LK-4-PBO-Latihan-2193013 <div></div> LK-5-PBO-2193013 <div></div> mail-system <div></div> mi <div></div> msb4</div>
	Selamat datang kembali pada pembahasan design pattern. Semoga kamu semakin semangat belajar materi ini. Kali ini kita akan belajar pattern Singleton . Pattern ini banyak diterapkan di berbagai framework, salah satunya adalah CodeIgniter3 . Sebenarnya masalah yang ingin diselesaikan oleh pattern ini adalah instansiasi . Mungkin kita secara sadar/tidak sering menginstansiasi object yang sama di berbagai tempat atau file. Ternyata ini buruk untuk management memory. Jika satu atau	

dua objek mungkin tidak terasa, bagaimana jika banyak?
Nah, disini Singleton pattern mencoba untuk menyelesaikan masalah tersebut. Pola penerapan Singleton sangat dikomendasikan untuk class/object yang **hanya** diperlukan satu kali objek dalam runtime/eksekusi program kita.
Contohnya apa?
koneksi database, logging, dst.
Yang perlu dicatat **tidak semua** objek atau class harus menerapkan Singleton. Dalam praktiknya namanya memprogram OOP kita pasti membutuhkan banyak instansiasi objek.

Supaya lebih paham mari kita lihat kode berikut ini :

```
<?php

// Tanpa
menggunakan prinsip
singleton.

class Database {
    public function
    __construct()
    {
        // Misal
        disini kode koneksi
        ke database.

        // Kalau di
        PHP misalnya saja
        mysqli_connect atau
        PDO connection.
    }
}
```

```

    public function
query($sql)
    {
        // Disini
harusnya adalah
sintaks PHP untuk
melakukan query
baik dengan
mysqli_query atau
PDO

        echo
"Mengeksekusi
\"{$sql}\"
..  


```

Pada kode tersebut kita membuat class Database, tidak ada yang spesial. Method `__construct` ber tugas mengkoneksikan program kita dengan DBMS. Sedangkan method `query` bertugas menjalankan sintaks SQL. Disitu hanya contoh, hanya mencetak string saja. Selanjutnya database kita instansiasi dan melakukan query.

Output kode tersebut adalah :

```

Mengeksekusi
"SELECT * FROM
users" ..

```

<p>Kode tersebut berjalan normal dan tidak ada masalah. Namun coba pikirkan jika program semakin kompleks, kamu harus meng-include kan file Database dimana-mana, pokoknya di semua lini kode yang membutuhkan koneksi database. Dalam satu kali runtime kamu tidak sadar beberapa kali menginstansiasi objek database, bukan hanya database, tapi juga objek lainnya yang sebenarnya hanya dibutuhkan satu kali instansiasi saja. Hal ini dapat menyebabkan kode kamu menjadi tidak efisien dan membuang-buang alokasi memory.</p> <p>Ilustrasi Chaos :</p> <pre>// Menginstansiasi database di file A \$db = new Database; // Menginstansiasi database di file B \$db = new Database; // Menginstansiasi database di file C \$db = new Database; Bahkan di file yang sama ada beberapa instansiasi :)</pre> <p>Lalu bagaimana pattern Singleton untuk menyelesaikan masalah ini? Oke, saatnya kita rombak ke Singleton.</p>	
---	--

Mari kita lihat kode berikut ini :

```
<?php

// Sudah menggunakan
prinsip singleton.

class Database {

    // Bikin wadah untuk
    menampung objek.

    private static
$instance = null;

    public function
__construct()
    {

        // Misal disini kode
koneksi ke database.

        // Kalau di PHP
misalnya saja
mysqli_connect atau
PDO connection.

    }

    // Ini method
pamungkas buat bikin
objek, tidak perlu pakai
new lagi.

    public static function
getInstance()
    {

        // Self sama
kegunaanya dengan
$this, tapi khusus untuk
static property.

        // Disini kita cek
apakah sebelumnya
instance sudah bikin
apa belum untuk cegah
double.

        // Kalau belum
pernah dibikin kita
```

new! kalau sudah
kembalikan yang sudah
ada.

```
    if (self::$instance  
== null) {  
        self::$instance =  
new Database();  
    }  
  
    // Kembalikan.  
    return  
self::$instance;  
}  
  
// Method seperti  
biasanya.  
public function  
query($sql)  
{  
    echo  
"Mengeksekusi  
\"{$sql}\" ..<br/>";  
}  
}  
  
// Menginstansiasi  
database, via  
getInstance, tidak new  
lagi.  
$db =  
Database::getInstance()  
;  
  
// Melakukan query.  
$db->query("SELECT *  
FROM users");  
?>
```

Outputnya sama saja :

Mengeksekusi
`"SELECT * FROM
users" ..`

Tapi cara kerjanya sudah beda jauh. Nah, bisa dilihat pada kode, cuma ada penambahan property private untuk menyimpan hasil instansiasi. Juga method `getInstance` untuk melakukan instansiasi dengan tambahan pengecekan. Pada kode tersebut ketika kita butuh objek Database, dapat dipanggil dengan cara :

```
// Tidak  
menggunakan $db =  
new Database;  
  
$db =  
Database::getInstan  
ce();
```

Dengan begini kita tidak akan pernah membuat instansiasi baru walaupun kode class ini di include dimana mana. Sudah pasti dalam `runtime` hanya akan menggunakan satu objek saja (satu alokasi memory). Kenapa bisa begini?

Cobalah disimak lagi kode berikut ini :

```
// Sudah ada belum?  
if (self::$instance  
== null) {  
    // Kalau blm  
    ada bikin objek  
    baru lalu assign ke  
    property instance.  
    self::$instance  
= new Database();
```

```
}  
  
// Kalau sudah ada  
kembalikan, tidak  
usah buat `new  
Database` lagi.  
return  
self::$instance;
```

Pengecekan pada method `getInstance()` akan menghindari double objek. Dia mengecek badan property pada class Database. Menarik bukan?



Lanjut, kita simulasi panggil beberapa kali dalam satu file, coba modifikasi kode jadi seperti ini :

```
// Kode sebelumnya  
...  
  
// Walaupun berkali  
kali dipanggil  
tidak akan pernah  
membuat alokasi  
memory baru.  
$db1 =  
Database::getInstan  
ce();  
$db2 =  
Database::getInstan  
ce();  
$db3 =  
Database::getInstan  
ce();  
$db4 =  
Database::getInstan  
ce();  
  
// Mengecek dan  
buktikan apakah  
sama?  
if ($db1 === $db2)  
{
```


	<pre>// Kalau tercetak sama, berarti proven. echo '
Sama!
'; }</pre> <pre>// Melakukan query. \$db1->query("SELECT * FROM users"); \$db2->query("SELECT * FROM users"); \$db3->query("SELECT * FROM users"); \$db4->query("SELECT * FROM users");</pre> <p>Objek db1 sampai 4 pada kode diatas tidak akan pernah membuat objek baru. Dia akan memanggil objek yang sudah dibuat sebelumnya. Lalu disitu kita juga membuat kode perbandingan dengan === untuk memastikan objek tersebut sama saja.</p> <p>Outputnya :</p> <pre>Sama! Mengeksekusi "SELECT * FROM users" .. Mengeksekusi "SELECT * FROM users" .. Mengeksekusi "SELECT * FROM users" .. Mengeksekusi "SELECT * FROM users" ..</pre> <p>Begitulah kira-kira penggunaan dari pattern Singleton.</p>	
2	Silahkan coba praktekkan lagi. Coba asah lagi kemampuan	Kasus : Kolom Pencarian

pattern Singleton pada materi diluar modul ini. Bisa mencari kata kunci **Singleton Pattern** pada Google.

```
LK-5-PBO-SINGELTON > apaya.php > Search > inputSearch
1  <?php
2  class Search
3  {
4      private static $instance = null;
5
6      public function __construct()
7      {
8      }
9
10     public static function getInstance()
11     {
12         if (self::$instance == null) {
13             self::$instance = new Search();
14         }
15         return self::$instance;
16     }
17
18     public function inputSearch($search)
19     {
20         echo "Mencari \"{$search}\" ..<br/>";
21     }
22 }
23
24 $s = Search::getInstance();
25
26 $s1 = Search::getInstance();
27 $s2 = Search::getInstance();
28 $s3 = Search::getInstance();
29 $s4 = Search::getInstance();
30
31 $s1->inputSearch("Halaman A");
32 $s2->inputSearch("Halaman B");
33 $s3->inputSearch("Halaman C");
34 $s4->inputSearch("Halaman D");
35
36 // if ($s1 === $s2) {
37 //     echo '<br/>Sama!<br/>';
38 // }
```

		<div>OUTPUT</div> <div>  localhost/lk-5-pbo-singelton/apaya.php</div> <div>Mencari "Halaman A" .. Mencari "Halaman B" .. Mencari "Halaman C" .. Mencari "Halaman D" ..</div>
--	--	--