Pertemuan 9: Static Properties dan Methods

Tujuan Pembelajaran

Setelah mengikuti pertemuan ini, mahasiswa diharapkan dapat:

1. Memahami konsep static properties dan methods dalam OOP
2. Membedakan static members dengan instance members
3. Menggunakan keyword `static` dengan benar
4. Mengakses static members dengan operator `::`
5. Memahami konsep late static binding dengan `static::`
6. Menggunakan static methods untuk utility functions
7. Menerapkan design patterns yang menggunakan static members

Konsep Static dalam OOP

Definisi Static

Static members (properties dan methods) adalah members yang dimiliki oleh class itu sendiri, bukan oleh instance/object tertentu. Static members dapat diakses tanpa perlu membuat instance dari class tersebut.

Karakteristik Static Members

1. **Shared across instances** - Nilai yang sama untuk semua instance
2. **No object context** - Tidak memiliki akses ke `$this`
3. **Class-level scope** - Milik class, bukan object
4. **Memory efficient** - Hanya ada satu copy di memory
5. **Early binding** - Resolved pada compile time (kecuali late static binding)

Kapan Menggunakan Static?

- **Utility functions** - Helper methods yang tidak memerlukan state
- **Constants** - Nilai yang tidak berubah
- **Counters** - Menghitung jumlah instance
- **Factory methods** - Membuat instance dengan cara khusus
- **Configuration** - Settings yang berlaku global
- **Design patterns** - Singleton, Registry, Factory

Static Properties

Sintaks Dasar

```php
class ClassName {
    public static $staticProperty = "value";
    private static $privateStatic = 0;
    protected static $protectedStatic;

    public static function getStaticProperty() {
        return self::$staticProperty;
    }
}

// Mengakses static property
echo ClassName::$staticProperty;
echo ClassName::getStaticProperty();
```

Akses Static Properties

```
class Counter {
    public static $count = 0;

    public function __construct() {
        self::$count++;  // Dari dalam class
    }

    public static function getCount() {
        return self::$count;
    }
}

// Dari luar class
echo Counter::$count;        // Direct access
echo Counter::getCount();    // Via static method

$obj1 = new Counter();       // count = 1
$obj2 = new Counter();       // count = 2
echo Counter::$count;        // Output: 2
```

Visibility pada Static Properties

```
class AccessExample {
    public static $publicStatic = "public";
    private static $privateStatic = "private";
    protected static $protectedStatic = "protected";

    public static function showStatic() {
        echo self::$publicStatic;     // OK
        echo self::$privateStatic;    // OK - dalam class yang sama
        echo self::$protectedStatic;  // OK - dalam class yang sama
    }
}

echo AccessExample::$publicStatic;     // OK
// echo AccessExample::$privateStatic; // Error - tidak bisa diakses
```

Static Methods

Sintaks dan Penggunaan

```
class MathUtils {
    public static function add($a, $b) {
        return $a + $b;
    }

    public static function multiply($a, $b) {
        return $a * $b;
    }

    public static function factorial($n) {
        if ($n <= 1) return 1;
        return $n * self::factorial($n - 1);  // Recursive call
    }
}

// Menggunakan static methods
echo MathUtils::add(5, 3);          // 8
echo MathUtils::multiply(4, 7);     // 28
echo MathUtils::factorial(5);       // 120
```

Batasan Static Methods

```
class Example {
    private $instanceProperty = "instance";
    private static $staticProperty = "static";

    public function instanceMethod() {
        echo $this->instanceProperty;      // OK
        echo self::$staticProperty;        // OK
    }

    public static function staticMethod() {
        // echo $this->instanceProperty;   // Error - $this tidak tersedia
        echo self::$staticProperty;        // OK

        // $this->instanceMethod();        // Error - tidak bisa akses
instance method
        self::anotherStaticMethod();       // OK
    }

    public static function anotherStaticMethod() {
        echo "Another static method";
    }
}
```

Self vs Static (Late Static Binding)

Keyword `self::`

`self::` merujuk ke class dimana code tersebut ditulis (early binding).

```php
class Parent {
    protected static $name = "Parent";

    public static function whoAmI() {
        return self::$name;  // Selalu merujuk ke Parent::$name
    }
}

class Child extends Parent {
    protected static $name = "Child";
}

echo Child::whoAmI();  // Output: "Parent" (bukan "Child")
```

Keyword `static::` (Late Static Binding)

`static::` merujuk ke class yang benar-benar memanggil method tersebut (late binding).

```php
class Parent {
    protected static $name = "Parent";

    public static function whoAmI() {
        return static::$name;  // Merujuk ke class yang memanggil
    }

    public static function selfWhoAmI() {
        return self::$name;    // Selalu Parent
    }
}

class Child extends Parent {
    protected static $name = "Child";
}

echo Child::whoAmI();      // Output: "Child"
echo Child::selfWhoAmI();  // Output: "Parent"
```

Contoh Praktis Late Static Binding

```php
abstract class Model {
    protected static $table;
    public static function getTableName() {
        return static::$table;  // Late static binding
    }

    public static function find($id) {
        $table = static::getTableName();
        return "SELECT * FROM {$table} WHERE id = {$id}";
    }
}

class User extends Model {
    protected static $table = "users";
}

class Product extends Model {
    protected static $table = "products";
}

echo User::find(1);     // SELECT * FROM users WHERE id = 1
echo Product::find(1);  // SELECT * FROM products WHERE id = 1
```

Static dalam Inheritance

Static Property Inheritance

```
class BaseClass {
    protected static $counter = 0;

    public static function increment() {
        static::$counter++;  // Late static binding
    }

    public static function getCounter() {
        return static::$counter;
    }
}

class ClassA extends BaseClass {
    // Inherits $counter, tapi punya copy sendiri
}

class ClassB extends BaseClass {
    // Inherits $counter, tapi punya copy sendiri
}

ClassA::increment();
ClassA::increment();
ClassB::increment();

echo ClassA::getCounter();  // 2
echo ClassB::getCounter();  // 1
echo BaseClass::getCounter(); // 0
```

Static Method Overriding

```
class Parent {
    public static function greet() {
        return "Hello from Parent";
    }

    public static function callGreet() {
        return static::greet();  // Late static binding
    }
}

class Child extends Parent {
    public static function greet() {
        return "Hello from Child";
    }
}

echo Child::greet();      // "Hello from Child"
echo Child::callGreet();  // "Hello from Child" (late binding)
```

Design Patterns dengan Static

1. Singleton Pattern

```
class Database {
    private static $instance = null;
    private $connection;

    private function __construct() {
        // Private constructor mencegah instantiation langsung
        $this->connection = "Database connection established";
    }

    public static function getInstance() {
        if (self::$instance === null) {
            self::$instance = new self();
        }
        return self::$instance;
    }

    public function getConnection() {
        return $this->connection;
    }

    // Mencegah cloning
    private function __clone() {}

    // Mencegah unserialization
    private function __wakeup() {}
}

$db1 = Database::getInstance();
$db2 = Database::getInstance();
var_dump($db1 === $db2); // true - same instance
```

2. Factory Pattern dengan Static

```
class VehicleFactory {
    public static function create($type, $brand, $model) {
        switch (strtolower($type)) {
            case 'car':
                return new Car($brand, $model);
            case 'motorcycle':
                return new Motorcycle($brand, $model);
            case 'truck':
                return new Truck($brand, $model);
            default:
                throw new InvalidArgumentException("Unknown vehicle type:
{$type}");
        }
    }

    public static function createCar($brand, $model) {
        return new Car($brand, $model);
    }

    public static function createMotorcycle($brand, $model) {
        return new Motorcycle($brand, $model);
    }
}

$car = VehicleFactory::create('car', 'Toyota', 'Camry');
$bike = VehicleFactory::createMotorcycle('Honda', 'CBR');
```

3. Registry Pattern

```
class Registry {
    private static $data = [];

    public static function set($key, $value) {
        self::$data[$key] = $value;
    }

    public static function get($key, $default = null) {
        return self::$data[$key] ?? $default;
    }

    public static function has($key) {
        return isset(self::$data[$key]);
    }

    public static function remove($key) {
        unset(self::$data[$key]);
    }

    public static function all() {
        return self::$data;
    }

    public static function clear() {
        self::$data = [];
    }
}

Registry::set('app_name', 'My Application');
Registry::set('version', '1.0.0');
echo Registry::get('app_name');  // My Application
```

Utility Classes dengan Static Methods

String Utilities

```
class StringHelper {
    public static function slugify($text) {
        $text = strtolower($text);
        $text = preg_replace('/[^a-z0-9]+/', '-', $text);
        return trim($text, '-');
    }

    public static function truncate($text, $length, $suffix = '...') {
        if (strlen($text) <= $length) {
            return $text;
        }
        return substr($text, 0, $length - strlen($suffix)) . $suffix;
    }

    public static function camelCase($text) {
        $text = str_replace(['-', '_'], ' ', $text);
        $text = ucwords($text);
        $text = str_replace(' ', '', $text);
        return lcfirst($text);
    }

    public static function randomString($length = 10) {
        $characters =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
        $result = '';
        for ($i = 0; $i < $length; $i++) {
            $result .= $characters[rand(0, strlen($characters) - 1)];
        }
        return $result;
    }
}

echo StringHelper::slugify("Hello World PHP");  // hello-world-php
echo StringHelper::truncate("Long text here", 10);  // Long te...
echo StringHelper::camelCase("hello-world");  // helloWorld
echo StringHelper::randomString(8);  // Random 8-char string
```

Validation Utilities

```
class Validator {
    public static function email($email) {
        return filter_var($email, FILTER_VALIDATE_EMAIL) !== false;
    }

    public static function url($url) {
        return filter_var($url, FILTER_VALIDATE_URL) !== false;
    }

    public static function numeric($value) {
        return is_numeric($value);
    }

    public static function required($value) {
        return !empty(trim($value));
    }

    public static function minLength($value, $min) {
        return strlen($value) >= $min;
    }

    public static function maxLength($value, $max) {
        return strlen($value) <= $max;
    }

    public static function betweenLength($value, $min, $max) {
        $length = strlen($value);
        return $length >= $min && $length <= $max;
    }

    public static function regex($value, $pattern) {
        return preg_match($pattern, $value);
    }
}
```

```
// Usage
var_dump(Validator::email("test@example.com"));  // true
var_dump(Validator::url("https://example.com")); // true
var_dump(Validator::minLength("password", 8));   // false
```

Static Constants

Class Constants

```
class Config {
    const APP_NAME = "My Application";
    const VERSION = "1.0.0";
    const DEBUG = true;

    public const API_URL = "https://api.example.com";  // PHP 7.1+
    private const SECRET_KEY = "secret123";  // PHP 7.1+

    public static function getConfig($key) {
        $constants = [
            'app_name' => self::APP_NAME,
            'version' => self::VERSION,
            'debug' => self::DEBUG,
            'api_url' => self::API_URL,
        ];

        return $constants[$key] ?? null;
    }
}

echo Config::APP_NAME;     // My Application
echo Config::VERSION;      // 1.0.0
echo Config::getConfig('app_name');  // My Application
```

Magic Constants

```
class Example {
    public static function showInfo() {
        echo "Class: " . __CLASS__ . "\n";
        echo "Method: " . __METHOD__ . "\n";
        echo "File: " . __FILE__ . "\n";
        echo "Line: " . __LINE__ . "\n";
    }
}

Example::showInfo();
```

Best Practices

1. Kapan Menggunakan Static

```php
// ✅ Good - Utility functions
class MathHelper {
    public static function percentage($value, $total) {
        return ($value / $total) * 100;
    }
}

// ✅ Good - Factory methods
class UserFactory {
    public static function createAdmin($name, $email) {
        return new User($name, $email, 'admin');
    }
}

// ❌ Bad - Shouldn't be static (needs instance state)
class User {
    private $name;

    // Wrong - needs instance data
    public static function getName() {
        return $this->name;  // Error: $this not available
    }
}
```

2. Naming Conventions

```php
class ApiHelper {
    // Static properties - camelCase dengan static prefix
    private static $defaultHeaders = [];
    private static $timeout = 30;

    // Static methods - camelCase
    public static function makeRequest($url, $data = []) {
        // Implementation
    }

    public static function setDefaultHeaders(array $headers) {
        self::$defaultHeaders = $headers;
    }

    public static function getDefaultTimeout() {
        return self::$timeout;
    }
}
```

3. Error Handling dalam Static Methods

```
class FileHelper {
    public static function readFile($filename) {
        if (!file_exists($filename)) {
            throw new InvalidArgumentException("File not found:
{$filename}");
        }

        $content = file_get_contents($filename);

        if ($content === false) {
            throw new RuntimeException("Failed to read file:
{$filename}");
        }

        return $content;
    }

    public static function writeFile($filename, $content) {
        $result = file_put_contents($filename, $content);

        if ($result === false) {
            throw new RuntimeException("Failed to write file:
{$filename}");
        }

        return $result;
    }
}
```

Testing Static Methods

Unit Testing

```
class CalculatorTest extends PHPUnit\Framework\TestCase {
    public function testAdd() {
        $result = Calculator::add(2, 3);
        $this->assertEquals(5, $result);
    }

    public function testDivideByZero() {
        $this->expectException(InvalidArgumentException::class);
        Calculator::divide(10, 0);
    }
}

class Calculator {
    public static function add($a, $b) {
        return $a + $b;
    }
    public static function divide($a, $b) {
        if ($b == 0) {
            throw new InvalidArgumentException("Division by zero");
        }
        return $a / $b;
    }
}
```

Common Pitfalls

1. Static vs Instance Confusion

```
class Counter {
    private static $staticCount = 0;
    private $instanceCount = 0;

    public function increment() {
        self::$staticCount++;      // Affects all instances
        $this->instanceCount++;    // Affects only this instance
    }

    public static function getStaticCount() {
        return self::$staticCount;
    }

    public function getInstanceCount() {
        return $this->instanceCount;
    }
}
```

2. Late Static Binding Issues

```
class A {
    protected static $name = "A";

    public static function whoAmI() {
        return self::$name;    // Always "A"
    }

    public static function whoAmILate() {
        return static::$name;  // Depends on calling class
    }
}

class B extends A {
    protected static $name = "B";
}
```

```
echo B::whoAmI();      // "A" - early binding
echo B::whoAmILate();  // "B" - late binding
```

Contoh Implementasi

Lihat file `example.php` untuk berbagai contoh implementasi static properties dan methods di PHP.

Latihan

1. Buat class `Counter` dengan static property untuk menghitung total instance
2. Implementasikan Singleton pattern untuk class `Logger`
3. Buat utility class `ArrayHelper` dengan static methods untuk array operations
4. Buat Factory pattern untuk membuat berbagai jenis `Shape` objects

Tugas Rumah

Buat sistem cache sederhana dengan:

- Static properties untuk menyimpan cache data
- Static methods: `set()`, `get()`, `has()`, `delete()`, `clear()`
- Implementasi TTL (Time To Live) untuk expired cache
- Statistics tracking (hit/miss ratio)
- Multiple cache stores (memory, file-based)