Pertemuan 5: Visibility (Public, Private, Protected)

Tujuan Pembelajaran

Setelah mengikuti pertemuan ini, mahasiswa diharapkan dapat:

1. Memahami konsep visibility dalam OOP
2. Membedakan antara public, private, dan protected
3. Mengimplementasikan encapsulation dengan proper visibility
4. Memahami access control dalam inheritance
5. Menggunakan getter dan setter untuk data access
6. Menerapkan information hiding principle
7. Memahami kapan menggunakan masing-masing visibility modifier

Konsep Visibility

Definisi Visibility

Visibility (atau access modifier) adalah mekanisme untuk mengontrol akses terhadap properties dan methods dalam sebuah class. Ini merupakan implementasi dari prinsip encapsulation dalam OOP.

Keuntungan Visibility Control

1. **Encapsulation** - Menyembunyikan detail implementasi internal
2. **Data Protection** - Mencegah akses langsung ke data sensitif
3. **API Control** - Menentukan interface public dari class
4. **Maintainability** - Memudahkan perubahan internal tanpa mempengaruhi code external
5. **Security** - Melindungi data dari modifikasi yang tidak sah

Jenis-jenis Visibility

1. Public

- **Akses:** Dapat diakses dari mana saja (dalam class, luar class, child class)
- **Keyword:** `public`
- **Default:** Jika tidak ada modifier, maka default adalah public
- **Penggunaan:** Untuk API public class, methods yang harus diakses dari luar

```
class Example {
    public $publicProperty = "Dapat diakses dari mana saja";

    public function publicMethod() {
        return "Method ini public";
    }
}
```

2. Private

- **Akses:** Hanya dapat diakses dari dalam class yang sama
- **Keyword:** `private`
- **Inheritance:** Tidak diwariskan ke child class
- **Penggunaan:** Untuk implementation details yang tidak boleh diakses dari luar

```
class Example {
    private $privateProperty = "Hanya bisa diakses dari dalam class";

    private function privateMethod() {
        return "Method ini private";
    }
}
```

3. Protected

- **Akses:** Dapat diakses dari class itu sendiri dan child class
- **Keyword:** `protected`
- **Inheritance:** Diwariskan ke child class
- **Penggunaan:** Untuk properties/methods yang perlu diakses child class tapi tidak dari luar

```
class Parent {
    protected $protectedProperty = "Bisa diakses child class";

    protected function protectedMethod() {
        return "Method ini protected";
    }
}
```

Visibility dalam Inheritance

Aturan Visibility Inheritance

1. **Public** → tetap public di child class
2. **Protected** → tetap protected di child class (bisa diubah ke public)
3. **Private** → tidak diwariskan sama sekali

Overriding Visibility Rules

- Child class tidak boleh membuat visibility lebih restrictive
- Child class boleh membuat visibility lebih permissive

```
class Parent {
    protected function someMethod() {} // Protected
}
```

```
class Child extends Parent {
    public function someMethod() {}    // ✓ OK - lebih permissive
    // private function someMethod() {} // ✗ Error - lebih restrictive
}
```

Encapsulation dengan Getter dan Setter

Prinsip Encapsulation

Encapsulation adalah prinsip OOP yang menyembunyikan detail implementasi internal dan hanya mengekspos interface yang diperlukan.

Implementasi dengan Private Properties

```
class BankAccount {
    private $balance;
    private $accountNumber;

    // Getter
    public function getBalance() {
        return $this->balance;
    }

    // Setter dengan validasi
    public function deposit($amount) {
        if ($amount > 0) {
            $this->balance += $amount;
            return true;
        }
        return false;
    }
}
```

Magic Methods untuk Property Access

```
class Example {
    private $data = [];

    public function __get($property) {
        return $this->data[$property] ?? null;
    }

    public function __set($property, $value) {
        $this->data[$property] = $value;
    }

    public function __isset($property) {
        return isset($this->data[$property]);
```

```
    }
}
```

Static Visibility

Static Properties dan Methods

Static members juga memiliki visibility yang sama:

```php
class Example {
    public static $publicStatic = "Public static";
    private static $privateStatic = "Private static";
    protected static $protectedStatic = "Protected static";

    public static function publicStaticMethod() {
        return self::$privateStatic; // Bisa akses private static
    }
}
```

Best Practices

1. Default ke Private/Protected

```php
class GoodExample {
    private $internalData;      // Private by default
    protected $sharedData;      // Protected jika perlu inheritance
    public $publicInterface;    // Public hanya jika perlu
}
```

2. Use Getters/Setters untuk Validation

```php
class User {
    private $email;

    public function setEmail($email) {
        if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $this->email = $email;
        } else {
            throw new InvalidArgumentException("Invalid email format");
        }
    }

    public function getEmail() {
        return $this->email;
    }
}
```

3. Protect Invariants

```php
class Rectangle {
    private $width;
    private $height;

    public function setWidth($width) {
        if ($width > 0) {
            $this->width = $width;
        } else {
            throw new InvalidArgumentException("Width must be positive");
        }
    }

    public function getArea() {
        return $this->width * $this->height;
    }
}
```

Common Patterns

1. Builder Pattern dengan Fluent Interface

```php
class QueryBuilder {
    private $query = [];

    public function select($fields) {
        $this->query['select'] = $fields;
        return $this;
    }

    private function buildSelect() {
        return "SELECT " . implode(', ', $this->query['select']);
    }
}
```

2. Factory Pattern dengan Private Constructor

```php
class DatabaseConnection {
    private $connection;

    private function __construct($dsn) {
        $this->connection = new PDO($dsn);
    }

    public static function createMysql($host, $database) {
```

```
        return new self("mysql:host=$host;dbname=$database");
    }
}
```

3. Singleton Pattern

```
class Config {
    private static $instance = null;
    private $settings = [];

    private function __construct() {
        // Private constructor
    }

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

Debugging Visibility Issues

ReflectionClass untuk Introspection

```
$reflection = new ReflectionClass('SomeClass');
$properties = $reflection->getProperties();

foreach ($properties as $property) {
    echo $property->getName() . " - ";
    if ($property->isPublic()) echo "Public";
    if ($property->isPrivate()) echo "Private";
    if ($property->isProtected()) echo "Protected";
    echo "\n";
}
```

Common Mistakes

1. Over-exposing Properties

```
// Bad
class User {
    public $password; // Tidak aman!
}
```

```php
// Good
class User {
    private $password;

    public function verifyPassword($input) {
        return password_verify($input, $this->password);
    }
}
```

2. Breaking Encapsulation

```php
// Bad
class BankAccount {
    public $balance; // Bisa diubah langsung dari luar
}

// Good
class BankAccount {
    private $balance;

    public function withdraw($amount) {
        if ($amount <= $this->balance) {
            $this->balance -= $amount;
            return true;
        }
        return false;
    }
}
```

Testing Private/Protected Methods

Using Reflection for Testing

```php
class TestHelper {
    public static function callPrivateMethod($object, $methodName, $args =
[]) {
        $reflection = new ReflectionClass($object);
        $method = $reflection->getMethod($methodName);
        $method->setAccessible(true);
        return $method->invokeArgs($object, $args);
    }
}
```

Contoh Implementasi

Lihat file `example.php` untuk berbagai contoh implementasi visibility modifiers di PHP.

Latihan

1. Buat class `Student` dengan private properties dan public getters/setters
2. Implementasikan validasi di setter methods
3. Buat class `Course` yang menggunakan protected properties untuk inheritance
4. Demonstrasikan perbedaan akses dari dalam class, luar class, dan child class

Tugas Rumah

Buat sistem e-commerce sederhana dengan:

- Class `Product` dengan private properties (id, name, price, stock)
- Proper getter/setter dengan validasi
- Class `ShoppingCart` dengan private items array
- Methods untuk add/remove items dengan proper validation
- Class `DiscountedProduct` extends Product dengan protected discount calculation
- Implementasikan proper encapsulation untuk semua data sensitif