Pertemuan 7: Interface

Tujuan Pembelajaran

Setelah mengikuti pertemuan ini, mahasiswa diharapkan dapat:

1. Memahami konsep interface dalam OOP
2. Membedakan interface dengan abstract class
3. Mengimplementasikan interface dengan keyword `implements`
4. Menggunakan multiple interface implementation
5. Memahami interface inheritance dengan keyword `extends`
6. Menerapkan polymorphism dengan interface
7. Menggunakan interface untuk dependency injection dan loose coupling

Konsep Interface

Definisi Interface

Interface adalah kontrak yang mendefinisikan method-method apa saja yang harus diimplementasikan oleh sebuah class. Interface hanya berisi deklarasi method tanpa implementasi (kecuali PHP 8+ yang mendukung default implementation).

Karakteristik Interface

1. **Tidak dapat diinstansiasi** - Tidak bisa membuat object langsung dari interface
2. **Hanya method signature** - Tidak ada implementasi (kecuali default methods di PHP 8+)
3. **Semua method public** - Visibility hanya public
4. **Tidak ada properties** - Kecuali constants
5. **Multiple implementation** - Class bisa implement multiple interfaces
6. **Interface inheritance** - Interface bisa extend interface lain

Sintaks Dasar

```
interface InterfaceName {
    // Constants (allowed)
    const CONSTANT_NAME = "value";

    // Method signatures (no implementation)
    public function methodName($parameter);
    public function anotherMethod(): string;
}

class ClassName implements InterfaceName {
    public function methodName($parameter) {
        // Implementation required
    }

    public function anotherMethod(): string {
```

```
        // Implementation required
        return "result";
    }
}
```

Interface vs Abstract Class

Perbandingan Detail

Aspek	Interface	Abstract Class
Instantiation	❌ Tidak bisa	❌ Tidak bisa
Method Implementation	❌ Tidak ada (PHP 8+ bisa default)	✅ Bisa concrete methods
Properties	❌ Tidak bisa (hanya constants)	✅ Bisa semua jenis
Constructor	❌ Tidak bisa	✅ Bisa
Multiple Inheritance	✅ Multiple implements	❌ Single extends
Method Visibility	🔒 Public only	✅ Public/Protected/Private
Inheritance Chain	✅ Interface extends interface	✅ Class extends class

Kapan Menggunakan Interface

- **Contract Definition** - Mendefinisikan kontrak yang harus dipenuhi
- **Multiple Capabilities** - Class perlu multiple behaviors
- **Loose Coupling** - Mengurangi dependency antar components
- **Polymorphism** - Treating different objects uniformly
- **Testing** - Memudahkan mocking dan testing
- **Plugin Architecture** - Sistem yang extensible

Kapan Menggunakan Abstract Class

- **Shared Implementation** - Ada code yang bisa dibagi
- **Template Pattern** - Algoritma umum dengan customizable parts
- **IS-A Relationship** - Hubungan hierarkis yang jelas
- **State Management** - Perlu properties untuk menyimpan state

Multiple Interface Implementation

Class dapat implement multiple interfaces:

```php
interface Readable {
    public function read(): string;
}

interface Writable {
    public function write(string $data): bool;
```

```
}

interface Executable {
    public function execute(): mixed;
}

class File implements Readable, Writable, Executable {
    public function read(): string { /* implementation */ }
    public function write(string $data): bool { /* implementation */ }
    public function execute(): mixed { /* implementation */ }
}
```

Interface Inheritance

Interface dapat extend interface lain:

```
interface BasicOperation {
    public function getName(): string;
}

interface AdvancedOperation extends BasicOperation {
    public function getDescription(): string;
    public function getComplexity(): int;
}

class Operation implements AdvancedOperation {
    // Harus implement semua methods dari BasicOperation dan
AdvancedOperation
    public function getName(): string { /* implementation */ }
    public function getDescription(): string { /* implementation */ }
    public function getComplexity(): int { /* implementation */ }
}
```

Interface Constants

Interface dapat berisi constants:

```
interface HttpStatusCode {
    const OK = 200;
    const NOT_FOUND = 404;
    const INTERNAL_SERVER_ERROR = 500;
}

class HttpResponse implements HttpStatusCode {
    public function getStatusCode(): int {
        return self::OK; // Menggunakan constant dari interface
    }
}
```

Polymorphism dengan Interface

Interface memungkinkan polymorphism yang powerful:

```
function processPayment(PaymentProcessor $processor, float $amount) {
    return $processor->processPayment($amount);
}

$creditCard = new CreditCardProcessor();
$bankTransfer = new BankTransferProcessor();
$eWallet = new EWalletProcessor();

processPayment($creditCard, 100.00);   // Polymorphic call
processPayment($bankTransfer, 100.00); // Polymorphic call
processPayment($eWallet, 100.00);      // Polymorphic call
```

Dependency Injection dengan Interface

Interface sangat berguna untuk dependency injection:

```
interface Logger {
    public function log(string $message, string $level = 'info'): void;
}

class FileLogger implements Logger {
    public function log(string $message, string $level = 'info'): void {
        // Log to file
    }
}

class DatabaseLogger implements Logger {
    public function log(string $message, string $level = 'info'): void {
        // Log to database
    }
}

class UserService {
    private Logger $logger;

    public function __construct(Logger $logger) {
        $this->logger = $logger; // Depends on interface, not concrete
class
    }

    public function createUser(array $data): User {
        // Create user logic
        $this->logger->log("User created: " . $data['email']);
        return $user;
    }
}
```

Common Interface Patterns

1. Repository Pattern

```php
interface UserRepository {
    public function find(int $id): ?User;
    public function findByEmail(string $email): ?User;
    public function save(User $user): bool;
    public function delete(int $id): bool;
}

class DatabaseUserRepository implements UserRepository {
    // Database implementation
}

class CacheUserRepository implements UserRepository {
    // Cache implementation
}
```

2. Strategy Pattern

```php
interface SortingStrategy {
    public function sort(array $data): array;
}

class BubbleSort implements SortingStrategy {
    public function sort(array $data): array {
        // Bubble sort implementation
    }
}

class QuickSort implements SortingStrategy {
    public function sort(array $data): array {
        // Quick sort implementation
    }
}
```

3. Observer Pattern

```php
interface Observer {
    public function update(Subject $subject): void;
}

interface Subject {
    public function attach(Observer $observer): void;
    public function detach(Observer $observer): void;
```

```
    public function notify(): void;
}
```

4. Factory Pattern

```
interface VehicleFactory {
    public function createCar(): Car;
    public function createMotorcycle(): Motorcycle;
}

class ToyotaFactory implements VehicleFactory {
    public function createCar(): Car {
        return new ToyotaCar();
    }

    public function createMotorcycle(): Motorcycle {
        return new ToyotaMotorcycle();
    }
}
```

Interface Segregation Principle (ISP)

Interface harus spesifik dan tidak memaksa implementasi method yang tidak diperlukan:

```
// Bad - Fat interface
interface Worker {
    public function work(): void;
    public function eat(): void;
    public function sleep(): void;
}

// Good - Segregated interfaces
interface Workable {
    public function work(): void;
}

interface Eater {
    public function eat(): void;
}

interface Sleeper {
    public function sleep(): void;
}

class Human implements Workable, Eater, Sleeper {
    // Implements all
}

class Robot implements Workable {
```

```
    // Only implements work, doesn't need eat/sleep
}
```

PHP 8+ Interface Features

Default Methods (PHP 8.0+)

```php
interface Timestampable {
    public function getCreatedAt(): DateTime;

    // Default implementation (PHP 8+)
    public function getFormattedCreatedAt(): string {
        return $this->getCreatedAt()->format('Y-m-d H:i:s');
    }
}
```

Mixed Type Hints

```php
interface DataProcessor {
    public function process(mixed $data): mixed;
}
```

Best Practices

1. Interface Naming

```php
// Good naming conventions
interface Readable {}           // Adjective ending in -able
interface PaymentProcessor {}   // Noun describing behavior
interface Logger {}             // Noun describing role
interface UserRepository {}     // Noun describing responsibility
```

2. Small, Focused Interfaces

```php
// Better: Small, focused interfaces
interface Readable {
    public function read(): string;
}

interface Writable {
    public function write(string $data): bool;
}

// Than: Large, monolithic interface
```

```php
interface FileHandler {
    public function read(): string;
    public function write(string $data): bool;
    public function delete(): bool;
    public function copy(string $destination): bool;
    public function move(string $destination): bool;
    // ... many more methods
}
```

3. Use Type Hints

```php
interface MathOperation {
    public function calculate(float $a, float $b): float;
}

class Calculator {
    public function performOperation(MathOperation $operation, float $a,
float $b): float {
        return $operation->calculate($a, $b);
    }
}
```

Testing dengan Interface

Interface memudahkan unit testing:

```php
// Mock implementation for testing
class MockLogger implements Logger {
    private array $logs = [];

    public function log(string $message, string $level = 'info'): void {
        $this->logs[] = ['message' => $message, 'level' => $level];
    }

    public function getLogs(): array {
        return $this->logs;
    }
}

// In test
class UserServiceTest extends PHPUnit\Framework\TestCase {
    public function testCreateUser() {
        $mockLogger = new MockLogger();
        $userService = new UserService($mockLogger);

        $user = $userService->createUser(['email' => 'test@example.com']);

        $logs = $mockLogger->getLogs();
        $this->assertCount(1, $logs);
```

```
        $this->assertEquals('User created: test@example.com', $logs[0]
['message']);
    }
}
```

Interface dalam Framework

Popular frameworks menggunakan interface extensively:

```php
// Laravel-style interfaces
interface Authenticatable {
    public function getAuthIdentifierName(): string;
    public function getAuthIdentifier(): mixed;
    public function getAuthPassword(): string;
}

// Symfony-style interfaces
interface EventDispatcherInterface {
    public function dispatch(object $event, string $eventName = null):
object;
    public function addListener(string $eventName, callable $listener, int
$priority = 0): void;
}
```

Contoh Implementasi

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

Latihan

1. Buat interface `Drawable` dengan method `draw()` dan `getArea()`
2. Implementasikan interface tersebut di class `Circle`, `Rectangle`, `Triangle`
3. Buat interface `Sortable` untuk berbagai algoritma sorting
4. Implementasikan multiple interfaces di satu class

Tugas Rumah

Buat sistem notification dengan:

- Interface `NotificationSender` dengan method `send(string $message, string $recipient)`
- Interface `NotificationFormatter` dengan method `format(array $data): string`
- Implementasi: `EmailSender`, `SMSSender`, `PushNotificationSender`
- Implementasi: `PlainTextFormatter`, `HTMLFormatter`, `JSONFormatter`
- Class `NotificationService` yang menggunakan dependency injection
- Demonstrasikan polymorphism dan multiple interface implementation