

# Pertemuan 6: Abstract Class dan Method

## Tujuan Pembelajaran

Setelah mengikuti pertemuan ini, mahasiswa diharapkan dapat:

1. Memahami konsep abstract class dan abstract method
2. Membedakan abstract class dengan regular class dan interface
3. Mengimplementasikan abstract class dengan proper design
4. Menggunakan abstract method untuk memaksa implementasi di child class
5. Menerapkan template method pattern dengan abstract class
6. Memahami kapan menggunakan abstract class vs interface
7. Mengimplementasikan polymorphism dengan abstract class

## Konsep Abstract Class

### Definisi Abstract Class

Abstract class adalah class yang tidak dapat diinstansiasi secara langsung. Abstract class dirancang untuk menjadi base class yang harus di-extend oleh class lain. Abstract class dapat berisi abstract method (method tanpa implementasi) yang harus diimplementasikan oleh child class.

### Karakteristik Abstract Class

1. **Tidak dapat diinstansiasi** - Tidak bisa membuat object langsung dari abstract class
2. **Dapat berisi abstract method** - Method tanpa implementasi
3. **Dapat berisi concrete method** - Method dengan implementasi lengkap
4. **Dapat memiliki properties** - Regular properties dan static properties
5. **Dapat memiliki constructor** - Constructor dapat dipanggil oleh child class
6. **Harus di-extend** - Untuk digunakan, harus dibuat child class

### Sintaks Dasar

```
abstract class AbstractClass {  
    // Properties (boleh ada)  
    protected $property;  
  
    // Constructor (boleh ada)  
    public function __construct($value) {  
        $this->property = $value;  
    }  
  
    // Concrete method (boleh ada)  
    public function concreteMethod() {  
        return "This is concrete method";  
    }  
  
    // Abstract method (harus diimplementasikan di child class)
```

```
abstract public function abstractMethod();
}
```

## Abstract Method

### Definisi Abstract Method

Abstract method adalah method yang dideklarasikan tanpa implementasi. Abstract method hanya bisa ada di dalam abstract class dan harus diimplementasikan oleh semua child class.

### Aturan Abstract Method

1. **Hanya signature** - Tidak boleh memiliki body/implementasi
2. **Harus diimplementasikan** - Child class wajib mengimplementasikan
3. **Visibility compatible** - Child class tidak boleh lebih restrictive
4. **Parameter compatible** - Signature parameter harus compatible
5. **Return type compatible** - Return type harus compatible (PHP 7.4+)

```
abstract class Shape {
    abstract public function calculateArea(): float;
    abstract public function calculatePerimeter(): float;
    abstract public function draw(): string;
}
```

## Abstract Class vs Interface vs Regular Class

### Perbandingan

Aspek	Abstract Class	Interface	Regular Class
Instantiation	✗ Tidak bisa	✗ Tidak bisa	✓ Bisa
Abstract Methods	✓ Bisa	✓ Semua abstract	✗ Tidak bisa
Concrete Methods	✓ Bisa	✗ Tidak bisa (PHP 8+ bisa)	✓ Bisa
Properties	✓ Bisa	✗ Tidak bisa	✓ Bisa
Constructor	✓ Bisa	✗ Tidak bisa	✓ Bisa
Multiple Inheritance	✗ Single	✓ Multiple	✗ Single
Visibility	✓ Public/Protected/Private	✓ Public only	✓ Semua

### Kapan Menggunakan Abstract Class

- **Shared code** - Ketika ada code yang ingin dibagi ke child classes
- **Template pattern** - Ketika ingin mendefinisikan algoritma umum
- **Partial implementation** - Ketika sebagian functionality sudah jelas
- **IS-A relationship** - Ketika ada hubungan hierarkis yang jelas

## Kapan Menggunakan Interface

- **Contract definition** - Ketika hanya ingin mendefinisikan kontrak
- **Multiple implementation** - Ketika class perlu implement multiple contracts
- **Loose coupling** - Ketika ingin dependency yang lebih loose
- **CAN-DO relationship** - Ketika fokus pada capability, bukan hierarchy

## Template Method Pattern

Template Method Pattern adalah design pattern yang mendefinisikan skeleton algorithm di abstract class, dengan beberapa step yang diimplementasikan oleh subclass.

```
abstract class DataProcessor {  
    // Template method – mendefinisikan algoritma umum  
    public final function process($data) {  
        $validData = $this->validate($data);  
        $transformedData = $this->transform($validData);  
        $result = $this->save($transformedData);  
        return $this->formatResult($result);  
    }  
  
    // Abstract methods – harus diimplementasikan subclass  
    abstract protected function validate($data);  
    abstract protected function transform($data);  
    abstract protected function save($data);  
  
    // Concrete method – implementasi default  
    protected function formatResult($result) {  
        return ['success' => true, 'data' => $result];  
    }  
}
```

## Final Methods dan Classes

### Final Method

Method yang tidak bisa di-override oleh child class.

```
abstract class BaseClass {  
    final public function finalMethod() {  
        // Tidak bisa di-override  
    }  
  
    abstract public function abstractMethod();  
}
```

### Final Class

Class yang tidak bisa di-extend.

```
final class UtilityClass {  
    // Class ini tidak bisa di-extend  
}
```

## Polymorphism dengan Abstract Class

Abstract class memungkinkan polymorphism yang kuat:

```
function processShape(Shape $shape) {  
    echo "Area: " . $shape->calculateArea() . "\n";  
    echo "Perimeter: " . $shape->calculatePerimeter() . "\n";  
    echo $shape->draw() . "\n";  
}  
  
$circle = new Circle(5);  
$rectangle = new Rectangle(4, 6);  
  
processShape($circle); // Polymorphic call  
processShape($rectangle); // Polymorphic call
```

## Best Practices

### 1. Use Abstract Classes for Shared Behavior

```
abstract class Animal {  
    protected $name;  
  
    public function __construct($name) {  
        $this->name = $name;  
    }  
  
    // Shared concrete method  
    public function getName() {  
        return $this->name;  
    }  
  
    // Abstract methods for specific behavior  
    abstract public function makeSound();  
    abstract public function move();  
}
```

### 2. Combine Concrete and Abstract Methods

```
abstract class HttpController {
    // Concrete method – common functionality
    protected function validateRequest($request) {
        // Common validation logic
    }

    protected function sendResponse($data, $statusCode = 200) {
        // Common response logic
    }

    // Abstract method – specific to each controller
    abstract public function handle($request);
}
```

### 3. Use Protected for Inheritance

```
abstract class BaseRepository {
    protected $connection;

    public function __construct($connection) {
        $this->connection = $connection;
    }

    // Template method
    public function find($id) {
        $this->validateId($id);
        return $this->executeFind($id);
    }

    abstract protected function executeFind($id);

    protected function validateId($id) {
        if (!is_numeric($id) || $id <= 0) {
            throw new InvalidArgumentException("Invalid ID");
        }
    }
}
```

## Advanced Concepts

### Abstract Static Methods

PHP tidak mendukung abstract static methods, tetapi bisa disimulasikan:

```
abstract class Model {
    abstract public static function getTableName();

    public static function findAll() {
```

```
    $table = static::getTableName(); // Late static binding
    return "SELECT * FROM {$table}";
}
}
```

## Abstract Properties (Simulated)

```
abstract class BaseModel {
    abstract protected function getRequiredProperties();

    public function validate() {
        $required = $this->getRequiredProperties();
        foreach ($required as $property) {
            if (!isset($this->$property)) {
                throw new Exception("Property {$property} is required");
            }
        }
    }
}
```

## Common Patterns

### 1. Factory Method Pattern

```
abstract class VehicleFactory {
    abstract public function createVehicle($type);

    public function deliverVehicle($type) {
        $vehicle = $this->createVehicle($type);
        $vehicle->prepare();
        return $vehicle;
    }
}
```

### 2. Strategy Pattern Base

```
abstract class SortingStrategy {
    abstract public function sort($data);

    public function benchmark($data) {
        $start = microtime(true);
        $result = $this->sort($data);
        $end = microtime(true);

        return [
            'result' => $result,
```

```
        'time' => $end - $start
    ];
}
}
```

## Error Handling

### Abstract Class Violations

```
// Error: Cannot instantiate abstract class
// $shape = new Shape();

// Error: Class must implement abstract methods
class IncompleteCircle extends Shape {
    // Missing implementation of abstract methods
}
```

## Testing Abstract Classes

### Testing Concrete Methods

```
// Create concrete implementation for testing
class TestableAbstractClass extends AbstractClass {
    public function abstractMethod() {
        return "test implementation";
    }
}

$testObject = new TestableAbstractClass();
// Test concrete methods
```

## Using Mocks

```
$mock = $this->getMockBuilder(AbstractClass::class);
	mock->expects($this->once())
		->method('abstractMethod')
		->willReturn('mocked result');
```

## Contoh Implementasi

Lihat file [example.php](#) untuk berbagai contoh implementasi abstract class dan method di PHP.

## Latihan

1. Buat abstract class `DatabaseConnection` dengan abstract methods untuk connect, query, dan close
2. Implementasikan concrete classes `MySQLConnection` dan `PostgreSQLConnection`
3. Buat abstract class `ReportGenerator` dengan template method pattern
4. Implementasikan `PDFReport` dan `ExcelReport` classes

## Tugas Rumah

Buat sistem payment processing dengan:

- Abstract class `PaymentProcessor` dengan template method `processPayment()`
- Abstract methods: `validatePayment()`, `executePayment()`, `sendNotification()`
- Concrete methods: `logTransaction()`, `handleError()`
- Implementasi: `CreditCardProcessor`, `BankTransferProcessor`, `EWalletProcessor`
- Setiap processor memiliki validasi dan logic yang berbeda
- Demonstrasikan polymorphism dengan array of different processors