# Pertemuan 3: Constructor dan Destructor

## Tujuan Pembelajaran

Setelah mengikuti pertemuan ini, mahasiswa diharapkan dapat:

1. Memahami konsep constructor dan destructor dalam OOP
2. Mengimplementasikan constructor dengan berbagai parameter
3. Menggunakan constructor overloading (simulasi)
4. Memahami kapan dan bagaimana menggunakan destructor
5. Implementasi constructor chaining dan inheritance basics
6. Menerapkan dependency injection melalui constructor

## Constructor

### Definisi Constructor

Constructor adalah method khusus yang secara otomatis dipanggil ketika sebuah object dibuat (instantiated). Constructor digunakan untuk menginisialisasi properti object dan melakukan setup awal yang diperlukan.

### Karakteristik Constructor

- Nama method: `__construct()`
- Dipanggil otomatis saat object dibuat dengan keyword `new`
- Tidak memiliki return value
- Dapat menerima parameter untuk inisialisasi
- Setiap class hanya boleh memiliki satu constructor

### Sintaks Dasar

```
class NamaClass {
    public function __construct($parameter1, $parameter2 = "default") {
        // Kode inisialisasi
        $this->properti1 = $parameter1;
        $this->properti2 = $parameter2;
    }
}
```

### Constructor dengan Parameter

Constructor dapat menerima parameter untuk memberikan nilai awal pada properti object:

```
class Mobil {
    private $merk;
    private $warna;
```

```
    public function __construct($merk, $warna = "putih") {
        $this->merk = $merk;
        $this->warna = $warna;
    }
}

$mobil = new Mobil("Toyota", "merah");
```

### Constructor Overloading (Simulasi)

PHP tidak mendukung constructor overloading secara native, tetapi dapat disimulasikan dengan beberapa teknik:

#### 1. Parameter Opsional

```
public function __construct($param1 = null, $param2 = null, $param3 =
null) {
    if ($param1 !== null) {
        // Logika untuk constructor dengan 1 parameter
    }
    if ($param2 !== null) {
        // Logika untuk constructor dengan 2 parameter
    }
}
```

#### 2. Static Factory Methods

```
class User {
    private $name;
    private $email;

    private function __construct($name, $email) {
        $this->name = $name;
        $this->email = $email;
    }

    public static function createFromArray($data) {
        return new self($data['name'], $data['email']);
    }

    public static function createGuest() {
        return new self("Guest", "guest@example.com");
    }
}
```

## Destructor

### Definisi Destructor

Destructor adalah method khusus yang secara otomatis dipanggil ketika sebuah object dihancurkan atau keluar dari scope. Destructor digunakan untuk cleanup operations seperti menutup file, database connections, atau membebaskan resources.

### Karakteristik Destructor

- Nama method: `__destruct()`
- Dipanggil otomatis saat object dihancurkan
- Tidak dapat menerima parameter
- Tidak memiliki return value
- Dipanggil pada akhir script atau saat `unset()`

### Sintaks Dasar

```
class NamaClass {
    public function __destruct() {
        // Kode cleanup
        echo "Object destroyed";
    }
}
```

### Kapan Destructor Dipanggil

1. Ketika script berakhir
2. Ketika object keluar dari scope
3. Ketika `unset()` dipanggil pada object
4. Ketika object di-overwrite dengan nilai lain

## Constructor dan Destructor dalam Inheritance

### Constructor Inheritance

- Child class mewarisi constructor dari parent class
- Child class dapat override constructor parent
- Gunakan `parent::__construct()` untuk memanggil constructor parent

```
class ParentClass {
    protected $name;

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

class ChildClass extends ParentClass {
    private $age;
```

```
    public function __construct($name, $age) {
        parent::__construct($name); // Memanggil constructor parent
        $this->age = $age;
    }
}
```

## Dependency Injection melalui Constructor

Constructor adalah tempat yang ideal untuk melakukan dependency injection:

```
class DatabaseLogger {
    private $database;
    private $tableName;

    public function __construct(DatabaseConnection $db, $tableName =
"logs") {
        $this->database = $db;
        $this->tableName = $tableName;
    }
}
```

## Best Practices

### Constructor

1. **Keep it Simple**: Constructor harus fokus pada inisialisasi
2. **Validate Parameters**: Lakukan validasi parameter yang diterima
3. **Fail Fast**: Jika ada error, throw exception di constructor
4. **Avoid Heavy Operations**: Hindari operasi berat di constructor
5. **Use Type Hints**: Gunakan type hints untuk parameter

### Destructor

1. **Cleanup Resources**: Gunakan untuk membersihkan resources
2. **Avoid Exceptions**: Jangan throw exception di destructor
3. **Keep it Light**: Destructor harus ringan dan cepat
4. **Log if Necessary**: Gunakan untuk logging jika diperlukan

## Common Patterns

### 1. Factory Pattern dengan Constructor

```
class ConfigManager {
    private $config;

    private function __construct($configFile) {
```

```
        $this->config = parse_ini_file($configFile);
    }

    public static function fromFile($file) {
        return new self($file);
    }

    public static function fromArray($array) {
        $instance = new self();
        $instance->config = $array;
        return $instance;
    }
}
```

### 2. Singleton Pattern

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

    private function __construct() {
        // Private constructor untuk mencegah instantiation
    }

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

## Error Handling

### Constructor Exception

```
class BankAccount {
    private $balance;

    public function __construct($initialBalance) {
        if ($initialBalance < 0) {
            throw new InvalidArgumentException("Saldo awal tidak boleh
negatif");
        }
        $this->balance = $initialBalance;
    }
}
```

## Contoh Implementasi

Lihat file `example.php` untuk berbagai contoh implementasi constructor dan destructor di PHP.

## Latihan

1. Buat class `Karyawan` dengan constructor yang menerima nama, jabatan, dan gaji
2. Implementasikan validasi di constructor untuk memastikan gaji positif
3. Buat destructor yang menampilkan pesan farewell
4. Buat static factory method untuk membuat karyawan dengan jabatan default

## Tugas Rumah

Buat sistem manajemen file dengan:

- Class `FileManager` dengan constructor yang menerima path file
- Validasi di constructor bahwa file exists dan readable
- Destructor yang otomatis menutup file handle jika masih terbuka
- Method untuk read, write, dan append file
- Exception handling yang proper untuk semua operasi file