Jalan Rajawali No. 14, 9 Ilir, Ilir Timur III Palembang 30113
Web: **ukmprogramming.mdp.ac.id** Email: **procom@mdp.ac.id**

OOP (*Object-Oriented-Programming*)

Apa itu OOP?

OOP adalah cara berpikir dalam pemrograman yang berorientasi pada "objek". Bayangkan objek seperti benda di dunia nyata. Setiap objek punya karakteristik (atribut) dan perilaku (method).

Konsep Dasar OOP

- **Class:** *Blueprint* atau cetakan untuk membuat objek. *Class* mendefinisikan atribut dan *method* yang dimiliki oleh objek.
- **Object:** Instance dari *class*. *Object* adalah wujud nyata dari *class*.
- **Atribut:** Karakteristik atau data yang dimiliki oleh objek.
- **Method:** Perilaku atau fungsi yang bisa dilakukan oleh objek.

Contoh

```dart
class Kucing {
  String nama;
  int umur;

  Kucing(this.nama, this.umur);

  void mengeong() {
    print('$nama mengeong...');
  }
}

void main() {
  Kucing kucing1 = Kucing("Milo", 2);
  kucing1.mengeong(); // Output: Milo mengeong...
}
```

Jalan Rajawali No. 14, 9 Ilir, Ilir Timur III Palembang 30113
Web: **ukmprogramming.mdp.ac.id** Email: **procom@mdp.ac.id**

Pilar OOP

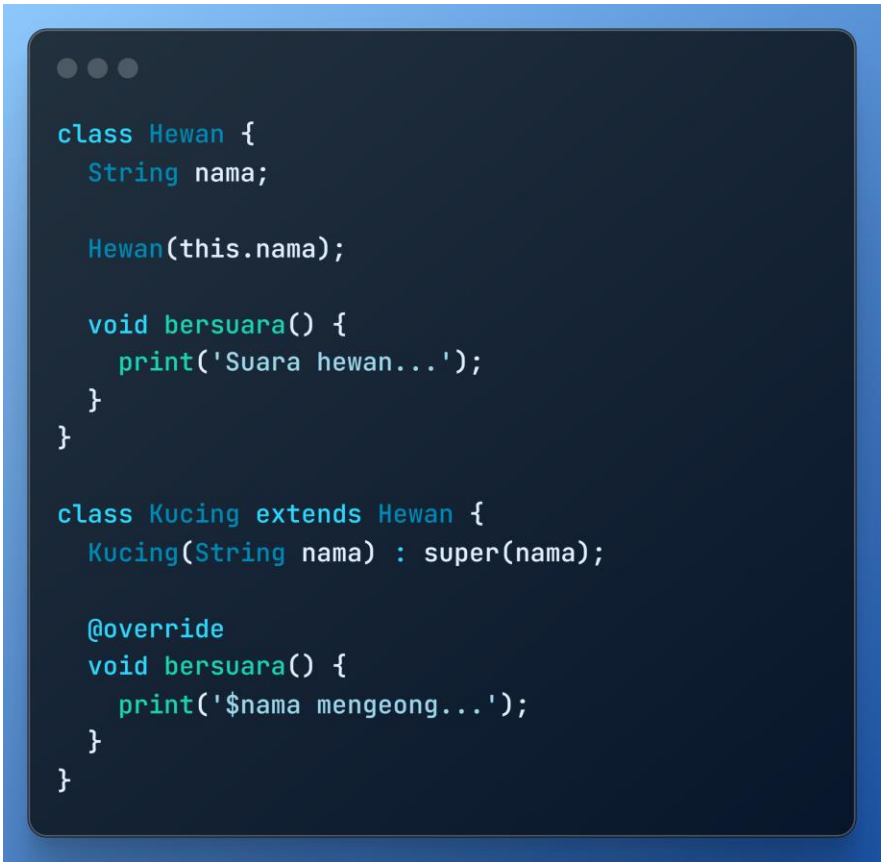
1. Encapsulation

- **Konsep:** Enkapsulasi seperti membungkus data dan *method* ke dalam sebuah kapsul. Tujuannya untuk melindungi data dari akses yang tidak sah dan menjaga kode tetap teratur.
- **Implementasi:** Di Dart, kamu bisa menggunakan *access modifier* (private dan public) untuk mengontrol akses ke atribut dan *method*. Atribut yang dideklarasikan dengan _ (underscore) di awal namanya akan menjadi *private*, artinya hanya bisa diakses dari dalam *class* yang sama.
- **Contoh:**

```dart
class RekeningBank {
  String _namaPemilik;
  double _saldo;

  RekeningBank(this._namaPemilik, this._saldo);

  void setor(double jumlah) {
    _saldo += jumlah;
  }

  void tarik(double jumlah) {
    if (jumlah <= _saldo) {
      _saldo -= jumlah;
    } else {
      print('Saldo tidak cukup');
    }
  }

  double getSaldo() {
    return _saldo;
  }
}
```

2. Inheritance

- **Konsep:** *Inheritance* seperti membuat *class* baru yang mewarisi sifat-sifat dari *class* yang sudah ada. *Class* baru ini disebut *child class* atau *subclass*, sedangkan *class* yang diwarisi disebut *parent class* atau *superclass*.
- **Implementasi:** Di Dart, kamu bisa menggunakan keyword extends untuk membuat *child class*.
- **Contoh:**



```dart
class Hewan {
  String nama;

  Hewan(this.nama);

  void bersuara() {
    print('Suara hewan...');
  }
}

class Kucing extends Hewan {
  Kucing(String nama) : super(nama);

  @override
  void bersuara() {
    print('$nama mengeong...');
  }
}
```

3. Polymorphism

- **Konsep:** Polymorphism artinya "banyak bentuk". Dalam OOP, *polymorphism* memungkinkan objek dari *class* yang berbeda untuk merespon *method* dengan nama yang sama dengan cara yang berbeda-beda.
- **Implementasi:** Di Dart, *polymorphism* bisa diimplementasikan dengan *method overriding* (menimpa *method* dari *parent class*) dan *interface*.
- **Contoh:**

```dart
class Bentuk {
  void gambar() {
    print('Menggambar bentuk...');
  }
}

class Lingkaran extends Bentuk {
  @override
  void gambar() {
    print('Menggambar lingkaran...');
  }
}

class Persegi extends Bentuk {
  @override
  void gambar() {
    print('Menggambar persegi...');
  }
}
```

Latihan

- Buat *class* Mobil dengan atribut merk, warna, dan tahun.
- Tambahkan *method* berjalan() dan berhenti() pada *class* Mobil.
- Buat *class* MobilListrik yang mewarisi dari *class* Mobil dan *override method* berjalan() untuk menampilkan pesan yang spesifik untuk mobil listrik.
- Buat *class* MobilBensin yang juga mewarisi dari *class* Mobil dan *override method* berjalan() untuk menampilkan pesan yang spesifik untuk mobil bensin.
- Di dalam main(), buat objek dari kedua *class* tersebut dan panggil *method* berjalan() untuk melihat perbedaannya.

Tips

- Identifikasi objek dan *class* yang relevan dengan program.
- Definisikan atribut dan *method* yang sesuai dengan karakteristik dan perilaku objek.
- Gunakan *inheritance* untuk membuat kode yang lebih efisien dan terstruktur.
- Terapkan *polymorphism* untuk membuat kode yang lebih fleksibel.

Dengan memahami OOP, kamu bisa membangun aplikasi Flutter yang lebih kompleks dan terstruktur. Selamat belajar!