OOP 🌷

By ladyfanning001

✅ 1. CLASS

Intinya:

Kelas itu kayak blueprint atau rancangan. Di dalamnya ada data (atribut) dan perilaku (method) yang bakal dimiliki objek.

Contoh gampang:

Kelas Mobil punya atribut merk, warna, dan method jalan().

Di dalam class ada :

- **Attribute (atau fields):** Variabel yang menyimpan data
- **Method:** Fungsi-fungsi yang bisa dilakukan oleh objek
- **Constructor:** Method khusus yang otomatis jalan saat bikin objek
- **Access Modifier:** public, private, protected buat ngatur akses data

Modifier	Siapa yang boleh akses	Contoh gampangnya
public	Semua orang, dari mana aja boleh lihat dan pakai	Rumahmu tanpa pagar, semua orang boleh masuk lihat-lihat
private	Cuma diri sendiri (class itu aja) yang boleh akses	Kamar pribadimu, cuma kamu yang boleh masuk
protected	Keluarga sendiri (class anak) atau tetangga deket (satu package)	Rumah punya pagar, tapi keluarga dan tetangga deket boleh masuk
(default) *(gak ditulis apa-apa)*	Tetangga satu komplek (satu package)	Orang luar gak boleh, cuma yang satu lingkungan boleh lihat

- **Static:** Digunakan untuk bikin method/atribut milik class, bukan objek
- **Final:** Supaya nilai nggak bisa diubah

✅ 2. OBJECT

Intinya:

Objek itu barang nyatanya dari kelas. Jadi kalau kelas itu blueprint-nya mobil, nah objeknya itu mobil beneran di dunia nyata.

Contoh:

Dari kelas Mobil, kita buat objek mobil1 yang mereknya “Toyota” warnanya “Merah”.

Di dalam objek:

- Objek “bawa” semua attribute dan method dari class-nya
- Bisa punya nilai atribut berbeda-beda walau dari class yang sama
- Pakai **new** buat bikin objek

Contoh :

```
class Mobil {
    String merk;    // Attribute
    String warna;

    // Constructor
    Mobil(String merk, String warna) {
        this.merk = merk;
        this.warna = warna;
    }

    // Method
    void nyalakan() {
        System.out.println(merk + " dinyalakan");
    }
}

public class Main {
    public static void main(String[] args) {
        // Membuat Object
        Mobil m1 = new Mobil("Toyota", "Merah");
        m1.nyalakan();  // Output: Toyota dinyalakan
    }
}
```

✅ 3. ENCAPSULATION

Intinya:

Ngumpetin data biar lebih aman. Biasanya data dibuat private, terus aksesnya pakai getter & setter.

Contoh gampang:

Kamu gak bisa sembarangan ganti nomor rangka mobil, harus lewat prosedur.

Alat-alat yang dipakai:

- **private** attribute ➔ data disembunyiin
- **getter & setter method** ➔ buat akses data yang disembunyiin
- **Access Modifier (private, public, protected)**

Contoh :

```
class Mahasiswa {
    private String nama;    // disembunyikan

    // Getter
    public String getNama() {
        return nama;
    }

    // Setter
    public void setNama(String nama) {
        this.nama = nama;
    }
}
```

✅ 4. INHERITANCE (Pewarisan)

Intinya:

Kelas anak bisa nurun semua isi dari kelas induk. Jadi gak perlu nulis ulang.

Contoh:

Kelas Mobil mewarisi dari kelas Kendaraan. Semua fungsi Kendaraan otomatis ada di Mobil.

Yang terlibat di dalamnya:

- **Keyword extends** ➔ buat nurunin class lain
- Bisa pakai **super()** ➔ manggil constructor parent
- Bisa override method parent

```
Contoh :

class Kendaraan {
    void jalan() {
        System.out.println("Kendaraan berjalan");
    }
```

```
}

//pake extends
class Mobil extends Kendaraan {
    @Override
    void jalan() {
        System.out.println("Mobil berjalan cepat");
    }
}
```

✅ 5. POLYMORPHISM (Polimorfisme)

Intinya:

Satu nama fungsi bisa dipakai banyak bentuk. Bisa juga, objek anak dianggap sebagai objek induknya.

Contoh:

Method jalan() bisa beda-beda isinya, tergantung objeknya Mobil, Motor, atau Sepeda.

Alat-alat yang terlibat:

- **Method Overloading:** Nama method sama, tapi parameternya beda
- **Method Overriding:** Method di child class nimpain method parent
- **Upcasting:** Object child dianggap sebagai parent-nya

Kegunaan Override: Biar perilaku method sama bisa beda di tiap class anak.

```
Contoh :
class Kendaraan {
    void jalan() {
        System.out.println("Kendaraan berjalan");
    }
}

class Mobil extends Kendaraan {
    @Override
    void jalan() {  // OVERRIDE method jalan dari parent
        System.out.println("Mobil berjalan cepat");
    }
}
Mobil m = new Mobil();
m.jalan();  // Outputnya nanti: Mobil berjalan cepat
```

Method Overloading :

```
class Hitung {
    int tambah(int a, int b) {
        return a + b;
    }

    int tambah(int a, int b, int c) {
        return a + b + c;
    }
}
```

✅ 6. ABSTRACTION (Abstraksi)

Intinya:

Nunjukin yang penting aja, detail yang ribet disembunyiin. Biar gampang dipake orang lain.

Contoh:

Kalau kamu nyetir mobil, tinggal gas atau rem aja. Kamu gak perlu mikirin cara kerja mesin di dalam.

Di dalamnya ada:

- **Abstract Class:** Class yang gak bisa di-new, tapi bisa diwarisin
- **Abstract Method:** Method yang cuma dideklarasikan, isinya nanti di child
- **Interface:** Kontrak yang harus diisi class yang “implements”

Contoh :

```
abstract class Hewan {
    abstract void suara();  // Harus diisi di anak class

    void makan() {
        System.out.println("Hewan sedang makan");
    }
}

class Kucing extends Hewan {
    @Override
    void suara() {
        System.out.println("Meong");
    }
}

public class Main {
    public static void main(String[] args) {
        Kucing k = new Kucing();
        k.makan();   // Ini bisa dipanggil
```

```
        k.suara();    // Ini wajib diisi sama Kucing
    }
}
```

Kalau ada method yang abstract, itu wajib diisi dulu di class turunannya (anaknya), baru bisa jalan.

✅ 7. STATIC & FINAL

Dipakai untuk nilai yang tetap atau milik class, bukan objek.

Kegunaan Static: Bisa diakses tanpa buat objek.

Kegunaan Final: Nilai tetap, gak bisa diubah.

```
Contoh :
class Data {
    static int jumlah = 0;  // Milik class, semua object share ini
    final String kode = "ABC123"; // Nilai tetap, gak bisa diubah

    Data() {
        jumlah++;
    }
}
```

✅ 8. EXCEPTION HANDLING (TRY - CATCH - FINALLY)

Nangkap error biar program gak berhenti.

Kegunaan: Program tetap jalan meski ada error.

```
Contoh :

try {

    int hasil = 10 / 0; // Error karena bagi nol

} catch (ArithmeticException e) {

    System.out.println("Terjadi error: " + e.getMessage());

} finally {

    System.out.println("Program selesai");

}
```