

## MODUL 9

### OVERRIDING dan POLYMORPHISM

#### 9.1 Overriding

- Method Overriding adalah konsep dalam *inheritance* (pewarisan) di mana subclass mendefinisikan ulang (mengubah isi) method yang sudah didefinisikan di superclass.
- Tujuan utamanya adalah agar subclass bisa menetapkan perilaku yang lebih spesifik dibanding superclass.

##### Aturan-aturan Method Overriding:

1. Nama method sama.
2. Parameter harus sama (jumlah dan tipe).
3. Return type sama atau subtype dari method yang dioverride.
4. Akses modifier harus sama atau lebih luas (tidak boleh lebih sempit).
5. Tidak bisa override method yang:
  - Private
  - Static
  - Final
6. Tidak bisa override constructor (constructor  $\neq$  method).

Contoh method override:

```
class Animal {
    public void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    public void sound() {
        System.out.println("Dog barks");
    }
}

public class TestOverride {
    public static void main(String[] args) {
        Animal a = new Dog(); // Polymorphism
        a.sound(); // Output: Dog barks
    }
}
```

Penjelasan:

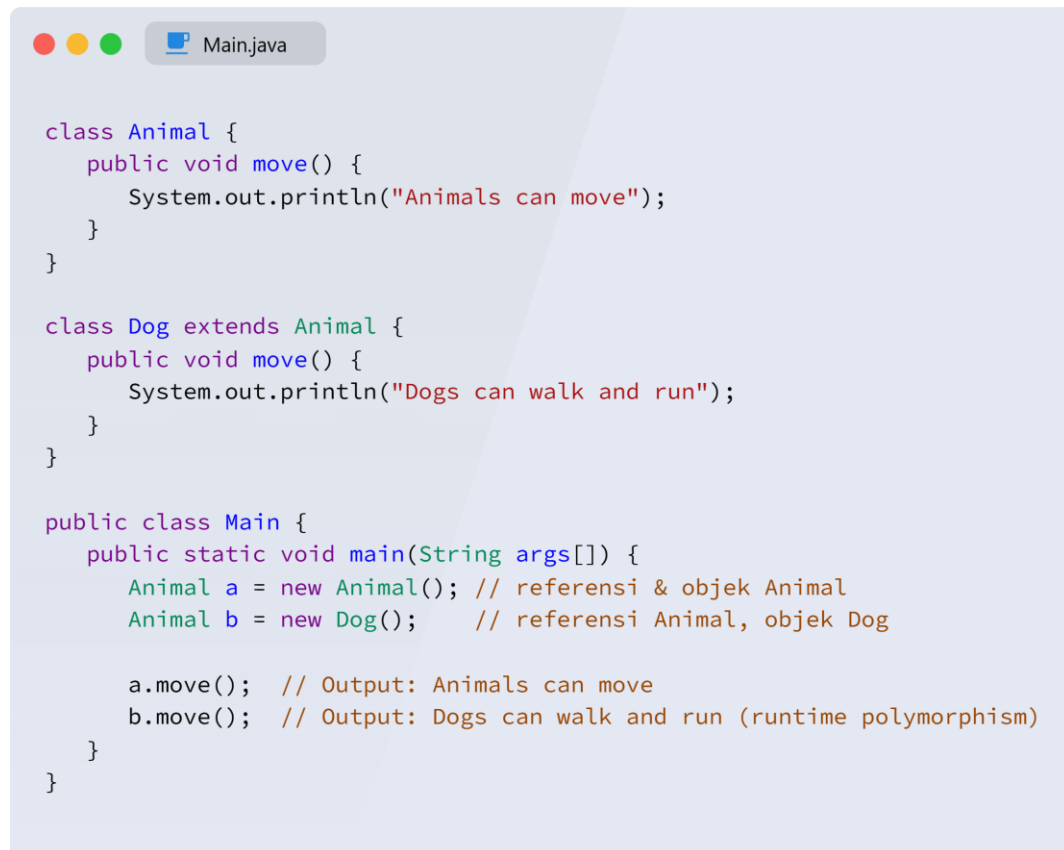
- Method sound() di class Dog mengoverride method sound() di class Animal.

- Meski objek bertipe Animal, karena objek sebenarnya adalah Dog, maka method Dog yang dipanggil.

### Manfaat Method Overriding:

1. Memungkinkan subclass menyesuaikan method dari superclass sesuai kebutuhannya.
2. Mewujudkan polimorfisme saat runtime (runtime polymorphism), yaitu kemampuan objek untuk merespon dengan cara berbeda tergantung tipe aktualnya.

Contoh:



```

class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}

class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
}

public class Main {
    public static void main(String args[]) {
        Animal a = new Animal(); // referensi & objek Animal
        Animal b = new Dog();    // referensi Animal, objek Dog

        a.move(); // Output: Animals can move
        b.move(); // Output: Dogs can walk and run (runtime polymorphism)
    }
}

```

Penjelasan:


- a.move() memanggil method dari Animal karena objeknya Animal.
- b.move() memanggil method dari Dog, walaupun tipe referensinya Animal, karena JVM menggunakan tipe objek saat runtime.

### Overriding VS Overloading

Aspek	Overriding	Overloading
Terjadi pada	Subclass - Superclass	Dalam class yang sama
Parameter	Harus sama	Harus berbeda (jumlah/tipe/urutan)
Return Type	Harus sama (atau subtype)	Bisa berbeda
Inheritance	Diperlukan	Tidak perlu

### Penting:

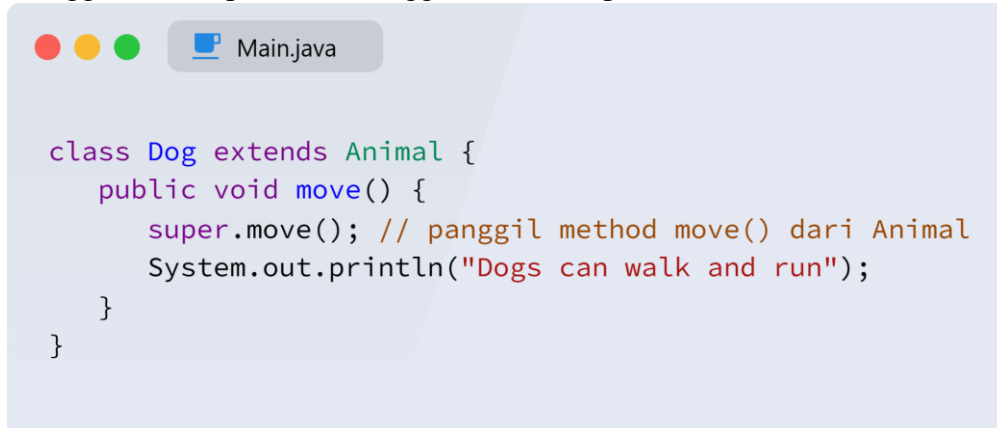
- Gunakan anotasi `@Override` agar compiler bisa membantu mendeteksi kesalahan saat override.
- Contoh kesalahan:



```
class Animal {
    private void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() { // ERROR: tidak bisa override method private
        System.out.println("Dog barks");
    }
}
```

- Menggunakan `super` untuk Panggil Method Superclass:



```
class Dog extends Animal {
    public void move() {
        super.move(); // panggil method move() dari Animal
        System.out.println("Dogs can walk and run");
    }
}
```

Overriding adalah dasar dari runtime polymorphism. Saat method dipanggil pada objek menggunakan referensi superclass, Java menentukan method mana yang akan dipanggil saat runtime, bukan saat kompilasi.

## 9.2 Polimorfisme (Polymorphism)

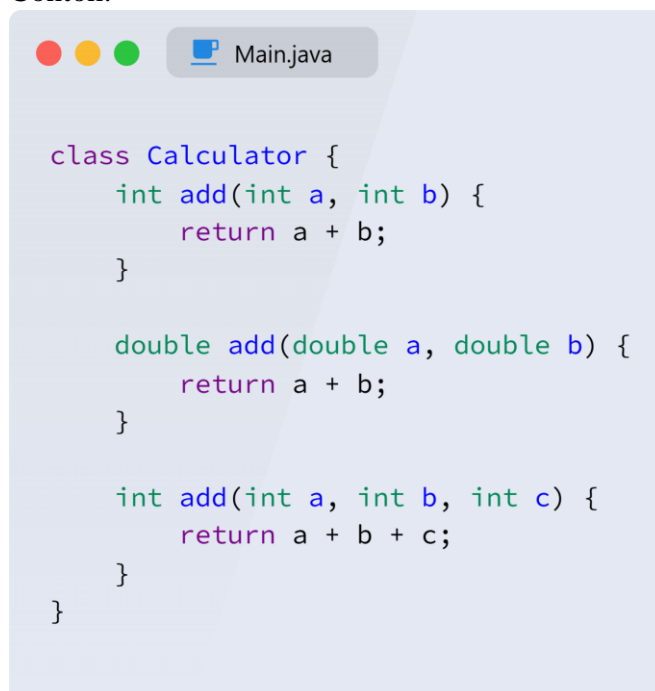
- Polimorfisme berasal dari bahasa Yunani, *Poly* = banyak; *Morph* = bentuk.
- Dalam OOP Java, Polimorfisme adalah kemampuan objek untuk mengambil banyak bentuk, tergantung pada konteksnya.
- Dengan polimorfisme, satu method atau objek dapat digunakan dengan **cara yang berbeda** tergantung pada jenis objek yang memanggilnya.

## Jenis-jenis Polimorfisme di Java:

Jenis	Nama Lain	Kapan Terjadi	Contoh
Compile-time	Static Polymorphism	Saat kompilasi	Method Overloading
Runtime	Dynamic Polymorphism	Saat program dijalankan	Method Overriding

### 1. Compile-Time Polymorphism (Static Polymorphism)

- Terjadi saat method overloading.
- Method Overloading = Dua atau lebih method dalam kelas yang sama, dengan nama sama, tapi parameter berbeda.
- Contoh:



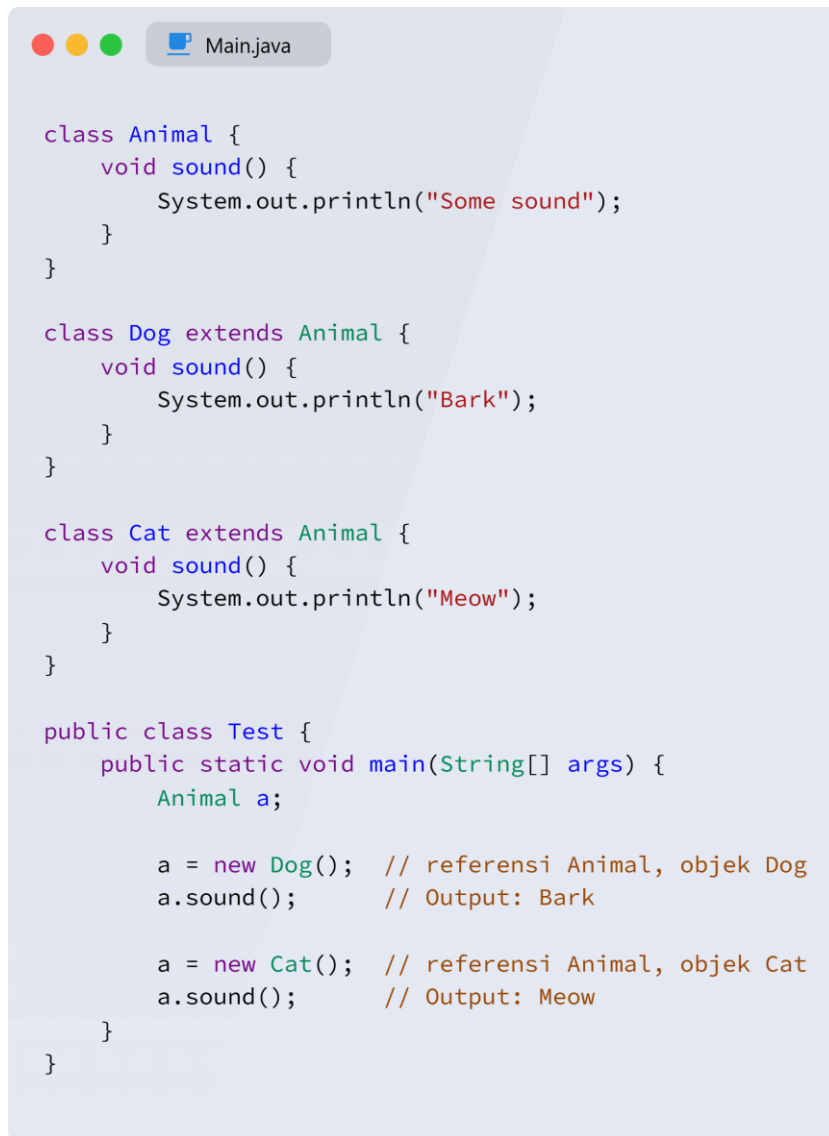
```
class Calculator {  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    double add(double a, double b) {  
        return a + b;  
    }  
  
    int add(int a, int b, int c) {  
        return a + b + c;  
    }  
}
```

- Pemanggilan add() akan ditentukan saat kompilasi berdasarkan parameter yang diberikan.

### 2. Runtime Polymorphism (Dynamic Polymorphism)

- Terjadi saat method overriding.
- Method Overriding = Subclass menerima method dari superclass dengan implementasi sendiri.
- Polimorfisme terjadi saat:
  - Referensi bertipe superclass
  - Objek bertipe subclass

Contoh:



```

class Animal {
    void sound() {
        System.out.println("Some sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}

class Cat extends Animal {
    void sound() {
        System.out.println("Meow");
    }
}

public class Test {
    public static void main(String[] args) {
        Animal a;

        a = new Dog(); // referensi Animal, objek Dog
        a.sound();      // Output: Bark

        a = new Cat(); // referensi Animal, objek Cat
        a.sound();      // Output: Meow
    }
}

```

- JVM akan menentukan method mana yang dipanggil saat runtime berdasarkan objek asli (Dog atau Cat), bukan tipe referensi (Animal).

Tujuan dan Manfaat Polimorfisme:

- Keterbacaan dan pemeliharaan kode jadi lebih baik.
- Fleksibilitas dalam mendesain program.
- Dukungan prinsip OOP seperti inheritance dan abstraction.
- Kode reusable dan scalable.
- Memungkinkan implementasi interface atau abstraksi untuk kelas yang berbeda.

### 9.3 Dynamic Binding & Static Binding

- Binding adalah proses pengaitan antara pemanggilan method (function call) dan implementasinya.
- Binding bisa terjadi saat kompilasi (compile-time) atau saat program berjalan (runtime).
- Static Binding digunakan untuk efisiensi saat kita tahu pasti method yang dipanggil.

- Dynamic Binding memberi fleksibilitas, terutama untuk inheritance dan polymorphism.

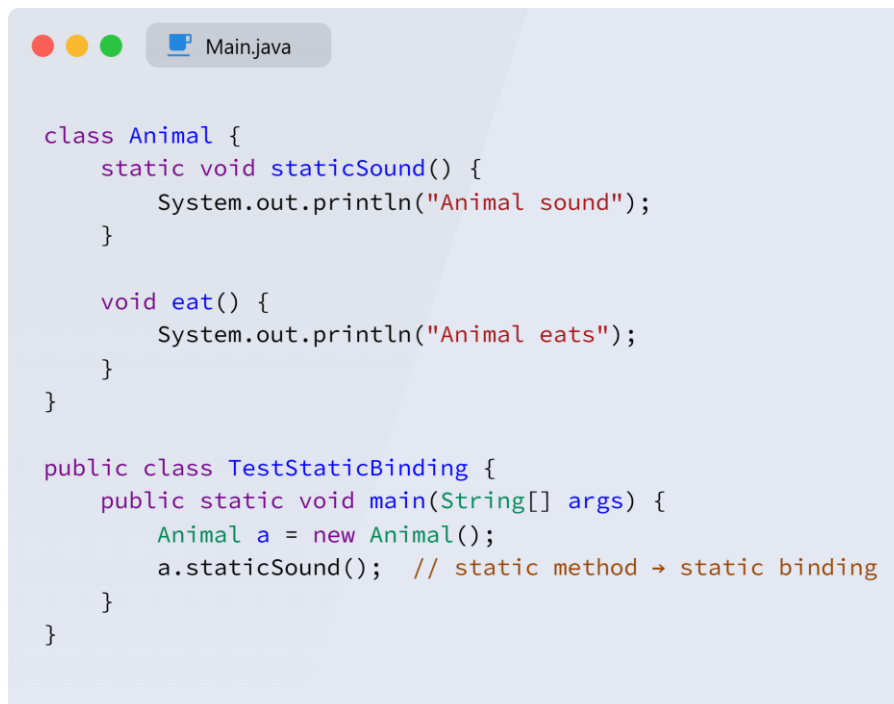
### 1. Static Binding (Early Binding)

Static binding adalah binding yang dilakukan oleh compiler saat kompilasi.

Terjadi ketika:

- Method adalah static
- Method adalah final
- Method adalah private
- Method adalah overloaded (method dengan nama sama tapi parameter berbeda)
- Variabel instance (field)

Contoh:

A screenshot of a Java IDE window titled 'Main.java'. The code defines a class 'Animal' with two methods: 'staticSound()' and 'eat()'. Both methods use 'System.out.println' to output their names. Below the 'Animal' class, there is a 'public class TestStaticBinding' with a 'main' method. In the 'main' method, an instance of 'Animal' is created and 'staticSound()' is called. A comment next to the call reads '// static method → static binding'.

```
class Animal {  
    static void staticSound() {  
        System.out.println("Animal sound");  
    }  
  
    void eat() {  
        System.out.println("Animal eats");  
    }  
}  
  
public class TestStaticBinding {  
    public static void main(String[] args) {  
        Animal a = new Animal();  
        a.staticSound(); // static method → static binding  
    }  
}
```

Penjelasan:

- Di atas, staticSound() akan dipanggil berdasarkan **tipe referensi** Animal, bukan objek.

### 2. Dynamic Binding (Late Binding)

Dynamic binding adalah binding yang dilakukan oleh JVM saat runtime berdasarkan tipe objek asli, bukan tipe referensinya.

Terjadi ketika:

- Method adalah non-static (instance method)
- Method adalah hasil dari overriding

Contoh:

```

class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}

public class TestDynamicBinding {
    public static void main(String[] args) {
        Animal a = new Dog(); // referensi Animal, objek Dog
        a.sound();             // dynamic binding → method milik Dog dipanggil
    }
}

```

Penjelasan:

- Di atas, sound() yang dipanggil adalah milik **class Dog**, walaupun referensinya Animal.

Perbedaan Dynamic Binding vs Static Binding:

Aspek	Static Binding	Dynamic Binding
Waktu binding	Compile-time (saat dikompilasi)	Runtime (saat program berjalan)
Tipe method	static, final, private, overload	non-static, override
Berdasarkan	Tipe referensi	Tipe objek yang sebenarnya
Contoh	Method overloading	Method overriding
Dukungan polimorfisme	Tidak	Ya
Performa	Lebih cepat	Sedikit lebih lambat

## Latihan

Kerjakan soal dibawah ini!

Latihan	
Durasi Pengerjaan	30 menit
Start	
End	

Soal	<p><b>Studi Kasus: Sistem Pembayaran E-Commerce</b></p> <p>Sebuah aplikasi e-commerce memiliki berbagai jenis metode pembayaran. Setiap metode memiliki logika pembayaran yang berbeda-beda tergantung pada:</p> <ol style="list-style-type: none"> <li>1. Jenis metode pembayaran (CreditCard, EWallet, BankTransfer).</li> <li>2. Jumlah pembayaran dan jenis mata uang (IDR, USD, EUR, dll).</li> </ol> <p>Manajemen aplikasi ingin:</p> <ul style="list-style-type: none"> <li>• Menggunakan <b>method overriding</b> untuk memberikan implementasi berbeda bagi setiap metode pembayaran.</li> <li>• Menggunakan <b>method overloading</b> untuk menangani skenario di mana pengguna bisa membayar baik dalam jumlah <i>default</i> (tanpa menyebut mata uang) maupun dengan menyebut mata uang.</li> <li>• Menggunakan <b>polymorphism</b> untuk memproses semua metode pembayaran dalam satu strukturarray/list.</li> </ul> <p><b>Tugas:</b></p> <ol style="list-style-type: none"> <li>1. Buat superclass PaymentMethod dengan: <ul style="list-style-type: none"> <li>◦ Method processPayment(double amount)</li> <li>◦ Method overload processPayment(double amount, String currency)</li> </ul> </li> <li>2. Buat subclass CreditCard, EWallet, dan BankTransfer yang <b>meng-override</b> method processPayment(double amount) dan <b>meng-overload</b> processPayment(double amount, String currency).</li> <li>3. Buat class Main untuk menguji semua fitur dengan <b>polymorphism</b>.</li> </ol>
Link Pengumpulan	

### Posttest

Kerjakan soal dibawah ini!

Pengerjaan Posttest	
Durasi Pengerjaan	30 menit
Start	
End	
Soal	<p><b>Studi Kasus: Sistem Manajemen Karyawan Perusahaan Teknologi</b></p> <p>Sebuah perusahaan teknologi memiliki berbagai jenis karyawan dengan tugas dan gaji yang berbeda tergantung pada perannya:</p> <ul style="list-style-type: none"> <li>• <b>SoftwareEngineer</b></li> </ul>



	<ul style="list-style-type: none"> <li>• <b>DataScientist</b></li> <li>• <b>Intern</b></li> </ul> <p>Perusahaan ingin:</p> <ol style="list-style-type: none"> <li>1. Menyediakan method calculateSalary() untuk menghitung gaji bulanan dasar tiap karyawan.</li> <li>2. Menyediakan method overload calculateSalary(boolean withBonus) yang menghitung gaji ditambah bonus jika parameter withBonus bernilai true.</li> <li>3. Menampilkan data setiap karyawan menggunakan polymorphism.</li> <li>4.</li> </ol> <p><b>Spesifikasi:</b></p> <ul style="list-style-type: none"> <li>• Superclass Employee: <ul style="list-style-type: none"> <li>◦ Method: calculateSalary()</li> <li>◦ Overload: calculateSalary(boolean withBonus)</li> </ul> </li> <li>• Subclass SoftwareEngineer, DataScientist, dan Intern: <ul style="list-style-type: none"> <li>◦ Meng-override method calculateSalary()</li> <li>◦ Meng-overload method calculateSalary(boolean withBonus)</li> </ul> </li> <li>• Class Main: <ul style="list-style-type: none"> <li>◦ Menyimpan array/list Employee[]</li> <li>◦ Memanggil method calculateSalary() secara polymorphic</li> <li>◦ Memanggil calculateSalary(true) secara langsung dari objek (karena overload tidak didukung langsung oleh polymorphism)</li> </ul> </li> </ul>
Link Pengumpulan	

### Take Home Task

Kerjakan soal dibawah ini!

Tugas	
Durasi Pengerjaan	6 hari
Start	Kamis
End	
Template laporan	<a href="#">contoh template laporan tht</a>
Soal	<p><b>Sistem Pemesanan Tiket Transportasi</b></p> <p>Sebuah perusahaan transportasi ingin membuat sistem sederhana untuk <b>pemesanan tiket</b>. Perusahaan menyediakan beberapa jenis transportasi, seperti:</p> <ul style="list-style-type: none"> <li>• <b>Bus</b></li> <li>• <b>Kereta</b></li> <li>• <b>Pesawat</b></li> </ul> <p>Setiap transportasi memiliki informasi umum seperti:</p>

- Nama transportasi
- Jumlah kursi
- Tujuan

Selain itu, tiap jenis transportasi memiliki cara perhitungan harga tiket yang berbeda-beda, tergantung jenis transportasi dan kelas layanan (Ekonomi, Bisnis, atau VIP).

#### Tugas Anda:

1. **Buatlah kelas induk Transportasi** yang memiliki properti:
  - nama, jumlahKursi, tujuan
  - Gunakan **enkapsulasi** (private + getter & setter)
  - Method `hitungHargaTiket()` dengan nilai default 100.000
2. **Buat subclass** dari Transportasi, yaitu:
  - Bus
  - Kereta
  - Pesawat

Masing-masing **meng-override** method `hitungHargaTiket()` sesuai aturan:

- Bus: Tambah 10% dari default
  - Kereta: Tambah 20% dari default
  - Pesawat: Tambah 50% dari default
3. Tambahkan **method overload** **`hitungHargaTiket(String kelasLayanan)`** pada masing-masing subclass. Aturan tambahan harga:
    - Ekonomi: +0%
    - Bisnis: +25%
    - VIP: +50%
  4. Buat kelas Main untuk:
    - Membuat object dari ketiga jenis transportasi
    - Menampilkan informasi masing-masing
    - Menampilkan hasil hitung harga tiket **tanpa dan dengan parameter (overloading)**
    - Gunakan konsep **polimorfisme** untuk menyimpan semua object dalam array `Transportasi[]` dan panggil `hitungHargaTiket()` secara dinamis.

#### Contoh output:

```
Bus ke Bandung - Harga tiket (default): 110000.0
Bus ke Bandung - Harga tiket (Bisnis): 137500.0

Kereta ke Surabaya - Harga tiket (default): 120000.0
Kereta ke Surabaya - Harga tiket (VIP): 180000.0

Pesawat ke Medan - Harga tiket (default): 150000.0
Pesawat ke Medan - Harga tiket (Ekonomi): 150000.0
```

Link Pengumpulan	
------------------	--