

Laporan Praktikum 4 Dasar Pemrograman Java Lanjutan

Pemrograman Berorientasi Objek

Nama : Cania Nabilatul Adawah
NIM : 2403102
Kelas : D3TI2C
Mata Kuliah : Pemrograman Berorientasi Objek

1. Encapsulation (Enkapsulasi)

Penjelasan Konsep

Encapsulation adalah salah satu dari empat pilar utama dalam Pemrograman Berorientasi Objek (PBO). Konsep ini mengacu pada pembungkusan (menggabungkan) data (variabel) dan metode (fungsi) yang beroperasi pada data tersebut menjadi satu unit, yaitu Class.

Tujuan utama Enkapsulasi adalah Data Hiding (Penyembunyian Data), yang dicapai dengan:

1. Mendeklarasikan variabel (field/atribut) di dalam class sebagai private (menggunakan access modifier private). Ini mencegah akses langsung dari luar class.
2. Menyediakan metode akses publik, yaitu Getter (untuk mengambil/membaca nilai data) dan Setter (untuk mengubah/menetapkan nilai data).

Dengan menggunakan metode setter, kita dapat menambahkan logika validasi atau kontrol lain sebelum data diubah, sehingga integritas data objek tetap terjaga.

Access Modifier:

- a. public: Ini adalah modifier yang paling tidak ketat. Anggota (member) yang dideklarasikan sebagai public dapat diakses dari mana saja; baik dari dalam class itu sendiri, dari package yang sama, dari subclass di package lain, maupun dari class mana pun di luar package (World).
- b. protected: Modifier ini memberikan akses yang lebih luas daripada default, tetapi tetap lebih terbatas daripada public. Anggota protected dapat diakses oleh class itu sendiri, semua class dalam package yang sama, dan semua subclass (child class), meskipun subclass tersebut berada di package yang berbeda.
- c. Default (Tidak Ada Modifier): Ketika tidak ada access modifier yang dituliskan, Java secara otomatis memberikan akses Default (sering disebut Package-Private). Akses ini memungkinkan anggota diakses oleh class itu sendiri dan semua class lain dalam package yang sama. Class di luar package tersebut, termasuk subclass di package lain, tidak dapat mengaksesnya.
- d. private: Ini adalah modifier yang paling ketat. Anggota yang dideklarasikan sebagai private hanya dapat diakses dari dalam class itu sendiri. Akses dari package yang

sama, subclass di package lain, maupun class di luar package sama sekali tidak diperbolehkan. Modifier ini adalah kunci utama dalam implementasi prinsip Encapsulation (Data Hiding).

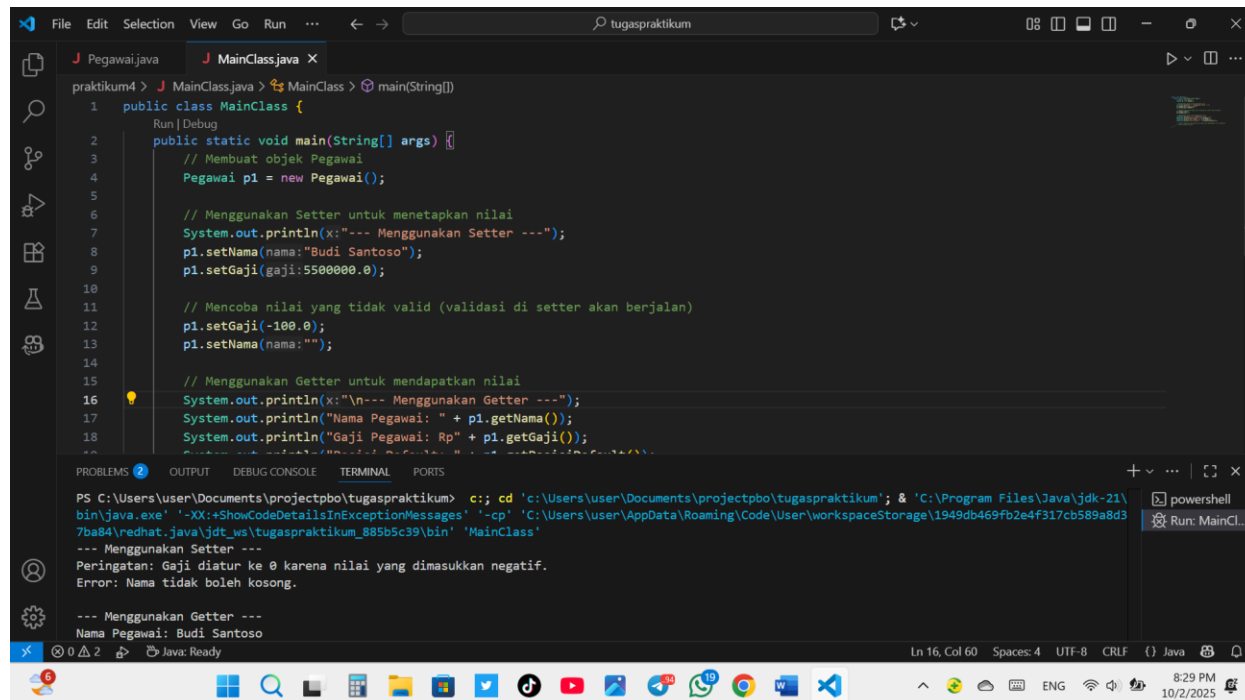
Non-Access Modifier:

- a. **final**: Keyword ini berfungsi untuk menetapkan kekekalan (immutability). Ketika diterapkan pada variabel, menjadikannya konstanta yang nilainya tidak dapat diubah setelah inisialisasi. Jika diterapkan pada metode, ia mencegah subclass untuk menimpanya (override). Jika diterapkan pada class, ia mencegah class tersebut untuk diwariskan atau diperluas (extend).
- b. **static**: Modifier ini mengubah kepemilikan anggota class dari objek (instance) menjadi milik class itu sendiri. Variabel static (class variable) memiliki satu salinan nilai yang dibagi oleh semua objek dari class tersebut. Metode static dapat dipanggil langsung menggunakan nama class tanpa perlu membuat objek (instance), misalnya `Math.random()`.
- c. **abstract**: Digunakan untuk menetapkan bahwa suatu class atau metode bersifat tidak lengkap dan membutuhkan implementasi di tempat lain. Metode abstract hanya berupa deklarasi tanpa body dan wajib diimplementasikan (override) oleh subclass non-abstract. Class abstract adalah class yang mengandung setidaknya satu metode abstract dan tidak dapat di-instantiate (dibuat objeknya secara langsung).
- d. **synchronized**: Modifier ini penting dalam lingkungan multithreading untuk mengamankan sumber daya bersama. Ketika diterapkan pada metode, ia menjamin bahwa pada satu waktu, hanya satu thread yang dapat mengeksekusi metode tersebut pada objek tertentu. Ini adalah mekanisme kunci untuk mencegah inkonsistensi data (thread safety).
- e. **volatile**: Modifier ini digunakan pada variabel untuk mengatasi masalah visibilitas di lingkungan multithreading. Ia menjamin bahwa nilai variabel yang diubah oleh satu thread akan segera dibaca dari memori utama dan tidak menggunakan salinan yang mungkin usang di cache lokal thread lainnya.
- f. **transient**: Modifier ini diterapkan pada variabel untuk menandakan bahwa variabel tersebut tidak perlu disimpan (diserialisasi) ketika objek ditulis ke dalam stream data (misalnya ke file atau saat dikirim melalui jaringan). Ini sering digunakan untuk melindungi data sensitif (seperti kata sandi) atau data yang dapat dihitung ulang.

Contoh Program Encapsulation:

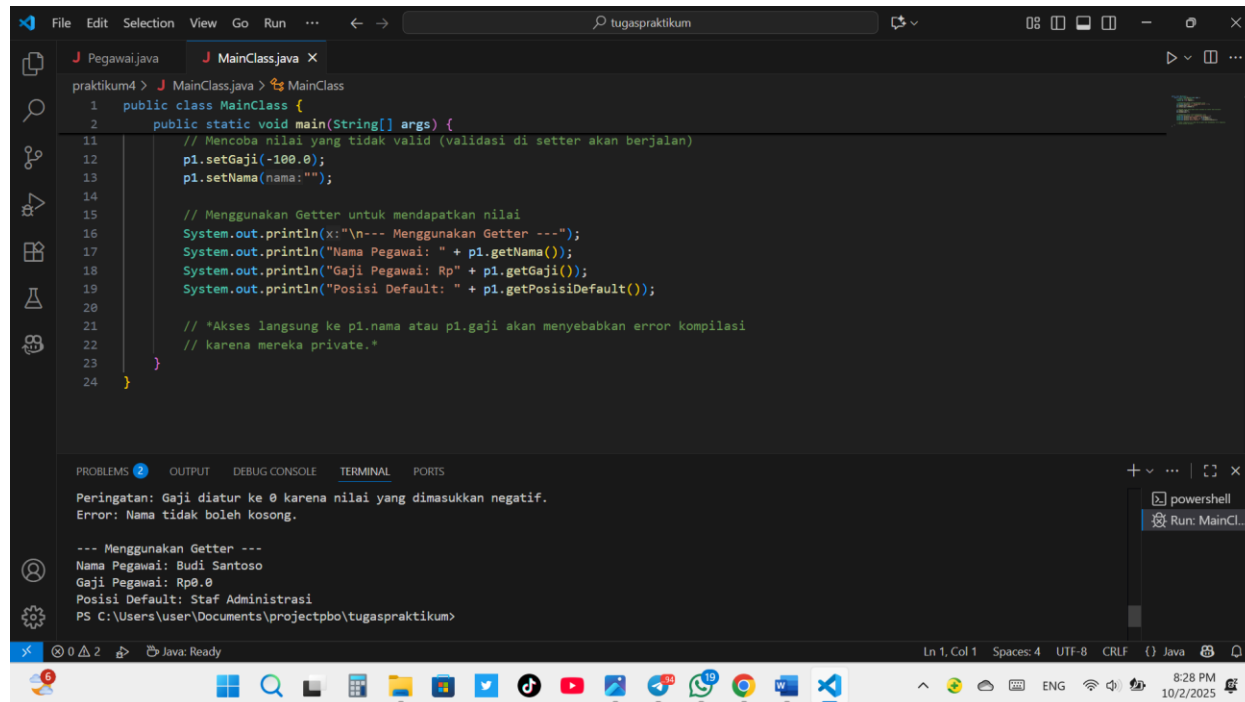
```
praktikum4 > J Pegawai.java > Pegawai
1 public class Pegawai {
2
3     // 1. Variabel (field) dideklarasikan private (Data Hiding)
4     private String nama;
5     private double gaji;
6
7     // Non-Access Modifier: final untuk konstanta
8     private final String POSISI_DEFAULT = "Staf Administrasi";
9
10    // 2. Setter Method: Untuk mengubah/menetapkan nilai data
11    public void setNama(String nama) {
12        // Logika kontrol/validasi (contoh: tidak boleh kosong)
13        if (nama != null && !nama.trim().isEmpty()) {
14            this.nama = nama;
15        } else {
16            System.out.println(x:"Error: Nama tidak boleh kosong.");
17        }
18    }
19
20    public void setGaji(double gaji) {
21        // Logika kontrol/validasi (contoh: gaji harus positif)
22        if (gaji >= 0) {
23            this.gaji = gaji;
24        } else {
25            this.gaji = 0; // Set ke 0 jika nilai negatif
26            System.out.println(x:"Peringatan: Gaji diatur ke 0 karena nilai yang dimasukkan negatif.");
27        }
28    }
29 }
```

```
praktikum4 > J Pegawai.java > Pegawai > setGaji(double)
1 public class Pegawai {
2     public void setGaji(double gaji) {
3
4     } else {
5         this.gaji = 0; // Set ke 0 jika nilai negatif
6         System.out.println(x:"Peringatan: Gaji diatur ke 0 karena nilai yang dimasukkan negatif.");
7     }
8 }
9
10 // 3. Getter Method: Untuk mengambil/membaca nilai data
11 public String getNama() {
12     return nama;
13 }
14
15 public double getGaji() {
16     return gaji;
17 }
18
19 // Getter untuk konstanta
20 public String getPosisiDefault() {
21     return POSISI_DEFAULT;
22 }
23 }
```



```
praktikum4 > J MainClass.java > MainClass > main(String[])
1 public class MainClass {
2     public static void main(String[] args) {
3         // Membuat objek Pegawai
4         Pegawai p1 = new Pegawai();
5
6         // Menggunakan Setter untuk menetapkan nilai
7         System.out.println(x:"--- Menggunakan Setter ---");
8         p1.setNama(nama:"Budi Santoso");
9         p1.setGaji(gaji:5500000.0);
10
11        // Mencoba nilai yang tidak valid (validasi di setter akan berjalan)
12        p1.setGaji(-100.0);
13        p1.setNama(nama:"");
14
15        // Menggunakan Getter untuk mendapatkan nilai
16        System.out.println(x:"\n--- Menggunakan Getter ---");
17        System.out.println("Nama Pegawai: " + p1.getNama());
18        System.out.println("Gaji Pegawai: Rp" + p1.getGaji());
19    }
20 }

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\User\Documents\projectpbo\tugaspraktikum> c:: cd 'C:\Users\User\Documents\projectpbo\tugaspraktikum'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\User\AppData\Roaming\Code\User\workspaceStorage\1949db469fb2e4f317cb589a8d37ba84\redhat.java\jdt_ws\tugaspraktikum_885b5c39\bin' 'MainClass'
--- Menggunakan Setter ---
Peringatan: Gaji diatur ke 0 karena nilai yang dimasukkan negatif.
Error: Nama tidak boleh kosong.
--- Menggunakan Getter ---
Nama Pegawai: Budi Santoso
```



```
praktikum4 > J MainClass.java > MainClass
1 public class MainClass {
2     public static void main(String[] args) {
3         // Membuat objek Pegawai
4         Pegawai p1 = new Pegawai();
5
6         // Menggunakan Setter untuk menetapkan nilai
7         System.out.println(x:"--- Menggunakan Setter ---");
8         p1.setNama(nama:"Budi Santoso");
9         p1.setGaji(gaji:5500000.0);
10
11        // Mencoba nilai yang tidak valid (validasi di setter akan berjalan)
12        p1.setGaji(-100.0);
13        p1.setNama(nama:"");
14
15        // Menggunakan Getter untuk mendapatkan nilai
16        System.out.println(x:"\n--- Menggunakan Getter ---");
17        System.out.println("Nama Pegawai: " + p1.getNama());
18        System.out.println("Gaji Pegawai: Rp" + p1.getGaji());
19        System.out.println("Posisi Default: " + p1.getPosisiDefault());
20
21        // *Akses langsung ke p1.nama atau p1.gaji akan menyebabkan error kompilasi
22        // karena mereka private.*
23    }
24 }

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Peringatan: Gaji diatur ke 0 karena nilai yang dimasukkan negatif.
Error: Nama tidak boleh kosong.
--- Menggunakan Getter ---
Nama Pegawai: Budi Santoso
Gaji Pegawai: Rp0.0
Posisi Default: Staf Administrasi
PS C:\Users\User\Documents\projectpbo\tugaspraktikum>
```

2. Inheritance (Pewarisan)

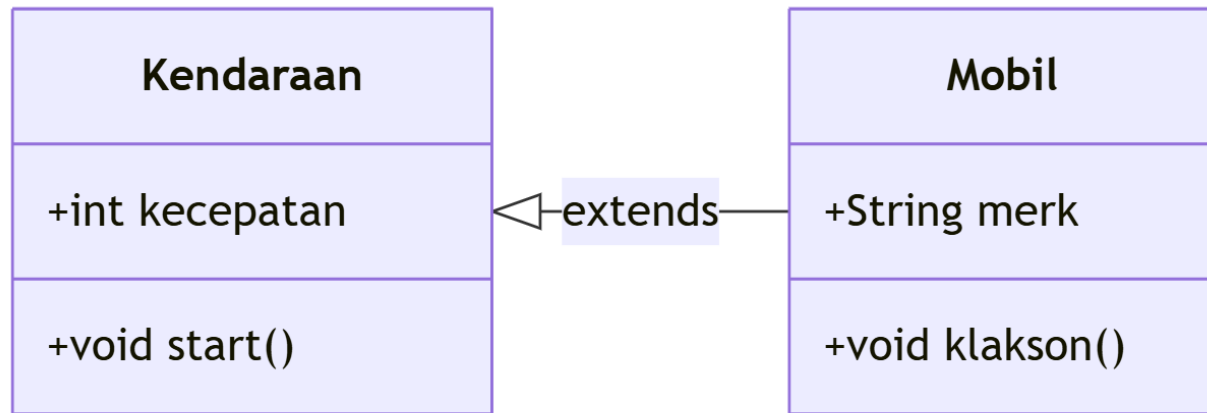
Inheritance adalah mekanisme di mana satu class (disebut subclass atau child class) dapat mewarisi properti (variabel) dan perilaku (metode) dari class lain (disebut superclass atau parent class). Inheritance mempromosikan Code Reusability (Penggunaan Ulang Kode) dan merepresentasikan hubungan "IS-A" (adalah sebuah). Dalam Java, kata kunci `extends` digunakan untuk melakukan inheritance antar class.

a. Single Inheritance (Pewarisan Tunggal)

Single Inheritance adalah bentuk pewarisan yang paling sederhana, di mana satu subclass (anak) mewarisi properti dan metode dari satu superclass (induk).

Hubungan: ClassB adalah turunan dari ClassA.

Notasi UML: ClassA <|-- ClassB.



The screenshot shows an IDE with the following Java code in `Kendaraan.java`:

```
1 class Kendaraan {
2     protected String jenis = "Mobil";
3
4     public void start() {
5         System.out.println(jenis + " telah dinyalakan.");
6     }
7
8     public void stop() {
9         System.out.println(jenis + " telah dimatikan.");
10    }
11 }
12
13 class Mobil extends Kendaraan {
14     public String merk;
15
16     public Mobil(String merk) {
17         // Mengubah nilai variabel 'jenis' dari superclass
18         this.jenis = "Mobil Sedan";
19         this.merk = merk;
20     }
21 }
```

The terminal output shows the execution of the program:

```
PS C:\Users\user\Documents\projectpbo\tugaspraktikum> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\user\AppData\Roaming\Code\User\workspaceStorage\1949db469fb2e4f317cb589a8d37ba84\redhat.java\jdt_ws\tugaspraktikum_885b5c39\bin' 'Mobil'
--- Single Inheritance ---
Toyota berbunyi 'Tin Tin!'
Jenis Kendaraan: Mobil Sedan
Mobil Sedan telah dinyalakan.
Mobil Sedan telah dimatikan.
PS C:\Users\user\Documents\projectpbo\tugaspraktikum>
```

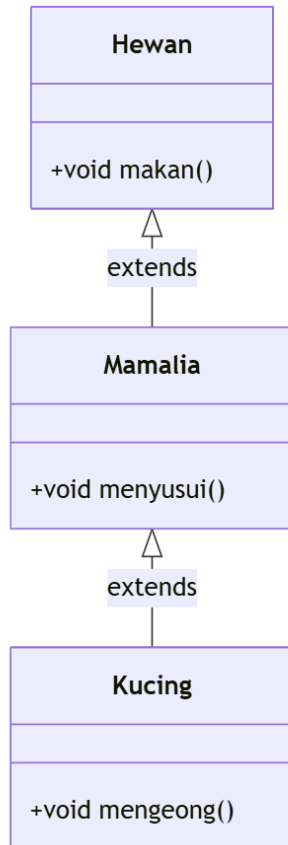
```
File Edit Selection View Go Run ...  
praktikum4 > J Kendaraan.java > Mobil  
13 class Mobil extends Kendaraan {  
16     public Mobil(String merk) {  
18         this.jenis = "Mobil Sedan";  
19         this.merk = merk;  
20     }  
21  
22     public void klakson() {  
23         System.out.println(merk + " berbunyi 'Tin Tin!'");  
24     }  
25  
26     // Metode utama untuk menjalankan  
27     public static void main(String[] args) {  
28         Mobil mobilSaya = new Mobil(merk:"Toyota");  
29  
30         System.out.println("Single Inheritance");  
31  
32         // Memanggil metode dari subclass  
33         mobilSaya.klakson();  
34  
35         // Memanggil properti dan metode yang diwarisi dari Kendaraan  
36         System.out.println("Jenis Kendaraan: " + mobilSaya.jenis);  
37         mobilSaya.start();  
38         mobilSaya.stop();  
39     }  
40 }
```

b. Multilevel Inheritance (Pewarisan Bertingkat)

Multilevel Inheritance terjadi ketika sebuah class mewarisi dari class lain, dan class turunan tersebut kemudian menjadi superclass untuk class ketiga, membentuk sebuah rantai.

Hubungan: $\text{ClassC} \rightarrow \text{ClassB} \rightarrow \text{ClassA}$.

Notasi UML: $\text{ClassA} <|-- \text{ClassB}, \text{ClassB} <|-- \text{ClassC}$.



The screenshot shows an IDE with the following Java code:

```
praktikum4 > J Hewan.java > Hewan
1 // File: Hewan.java (Superclass / Level 1)
2 class Hewan {
3     public void makan() {
4         System.out.println(x:"Hewan sedang makan.");
5     }
6 }
7
8 // File: Mamalia.java (Subclass dari Hewan / Superclass untuk Kucing / Level 2)
9 class Mamalia extends Hewan {
10     public void menyusui() {
11         System.out.println(x:"Mamalia dapat menyusui.");
12     }
13 }
14
15 // File: Kucing.java (Subclass dari Mamalia / Level 3)
16 class Kucing extends Mamalia {
17     public void mengeong() {
18         System.out.println(x:"Kucing mengeong 'Meow!'");
19     }
20 }
21
22 // Metode utama untuk menjalankan
23 Run | Debug
24 public static void main(String[] args) {
25     Kucing myCat = new Kucing();
26
27     System.out.println(x:"--- Multilevel Inheritance ---");
28
29     // Memanggil metode dari Kucing (Level 3)
30     myCat.mengeong();
31 }
```

The screenshot shows an IDE with the following content:

```
praktikum4 > J Hewan.java > Hewan
16 class Kucing extends Mamalia {
    Run | Debug
22 public static void main(String[] args) {
23     Kucing myCat = new Kucing();
24
25     System.out.println(x:"--- Multilevel Inheritance ---");
26
27     // Memanggil metode dari Kucing (Level 3)
28     myCat.mengeong();
29
30     // Memanggil metode dari Mamalia (Level 2 - Parent)
31     myCat.menyusui();
32
33     // Memanggil metode dari Hewan (Level 1 - Grandparent)
34     myCat.makan();
35 }
36 }
```

TERMINAL

```
PS C:\Users\user\Documents\projectpbo\tugaspraktikum> c;; cd 'c:\Users\user\Documents\projectpbo\tugaspraktikum'; & 'C
:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\user\AppData\Roamin
g\Code\User\workspaceStorage\1949db469fb2e4f317cb589a8d37ba84\redhat.java\jdt_ws\tugaspraktikum_885b5c39\bin' 'Kucing'

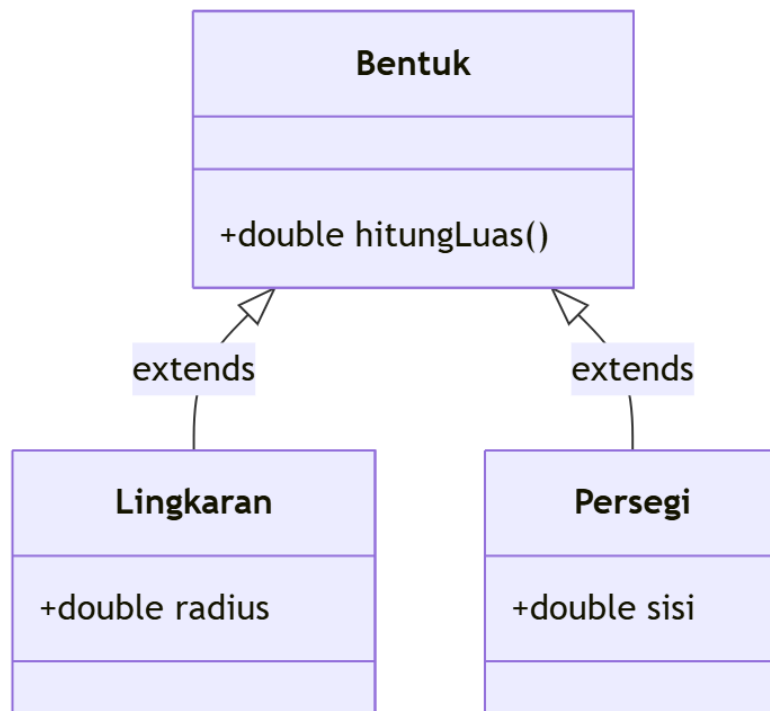
--- Multilevel Inheritance ---
Kucing mengeong 'Meow!'
Mamalia dapat menyusui.
Hewan sedang makan.
PS C:\Users\user\Documents\projectpbo\tugaspraktikum>
```

c. Hierarchical Inheritance (Pewarisan Hierarki)

Hierarchical Inheritance terjadi ketika satu superclass diwarisi oleh lebih dari satu subclass. Semua subclass berbagi properti dan metode dari superclass yang sama.

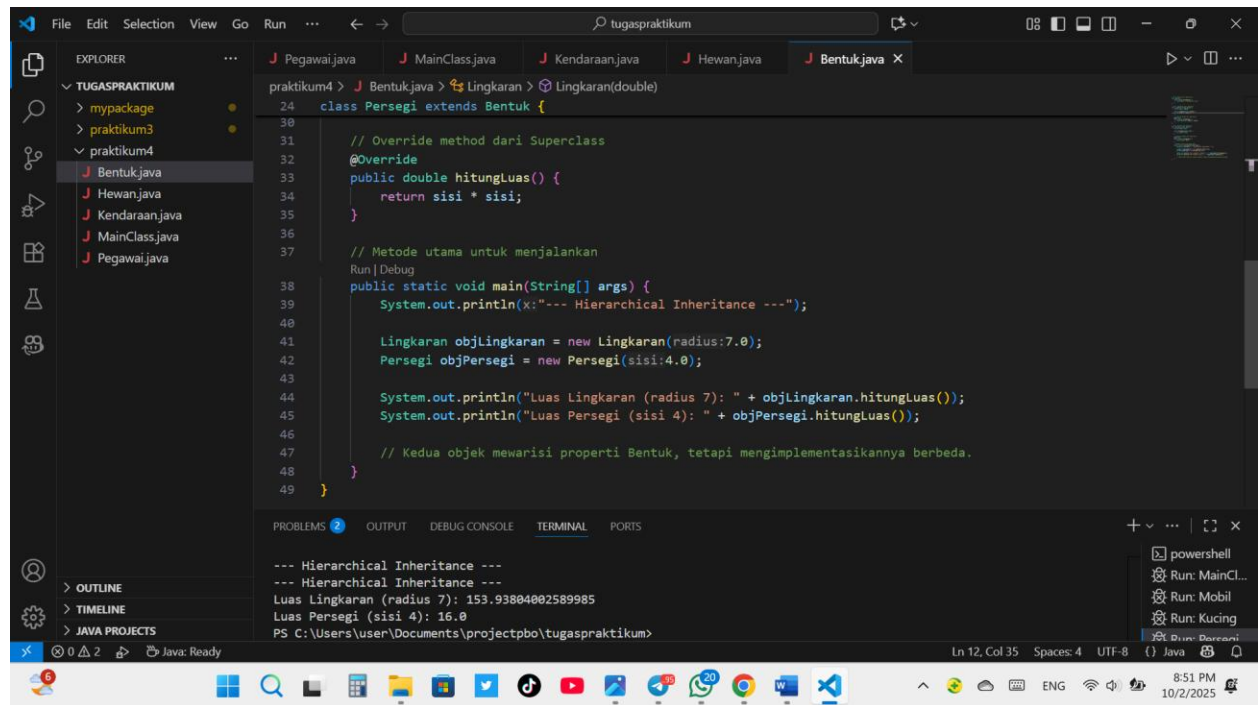
Hubungan: ClassB → ClassA, ClassC → ClassA.

Notasi UML: ClassA <|-- ClassB, ClassA <|-- ClassC.



The screenshot shows an IDE with the following code:

```
praktikum4 > J Bentuk.java > Persegi
1 // File: Bentuk.java (Superclass)
2 class Bentuk {
3     public double hitungLuas() {
4         return 0.0; // Implementasi dasar
5     }
6 }
7
8 // File: Lingkaran.java (Subclass)
9 class Lingkaran extends Bentuk {
10     private double radius;
11
12     public Lingkaran(double radius) {
13         this.radius = radius;
14     }
15
16     // Override method dari Superclass
17     @Override
18     public double hitungLuas() {
19         return Math.PI * radius * radius;
20     }
21 }
22
23 // File: Persegi.java (Subclass)
24 class Persegi extends Bentuk {
25     private double sisi;
26
27     public Persegi(double sisi) {
28         this.sisi = sisi;
29     }
30 }
```



```
praktikum4 > J Bentuk.java > Lingkaran > Lingkaran(double)
24 class Persegi extends Bentuk {
25
26     // Override method dari Superclass
27     @Override
28     public double hitungLuas() {
29         return sisi * sisi;
30     }
31
32     // Metode utama untuk menjalankan
33     public static void main(String[] args) {
34         System.out.println(x:"--- Hierarchical Inheritance ---");
35
36         Lingkaran objLingkaran = new Lingkaran(radius:7.0);
37         Persegi objPersegi = new Persegi(sisi:4.0);
38
39         System.out.println("Luas Lingkaran (radius 7): " + objLingkaran.hitungLuas());
40         System.out.println("Luas Persegi (sisi 4): " + objPersegi.hitungLuas());
41
42         // Kedua objek mewarisi properti Bentuk, tetapi mengimplementasikannya berbeda.
43     }
44 }
```

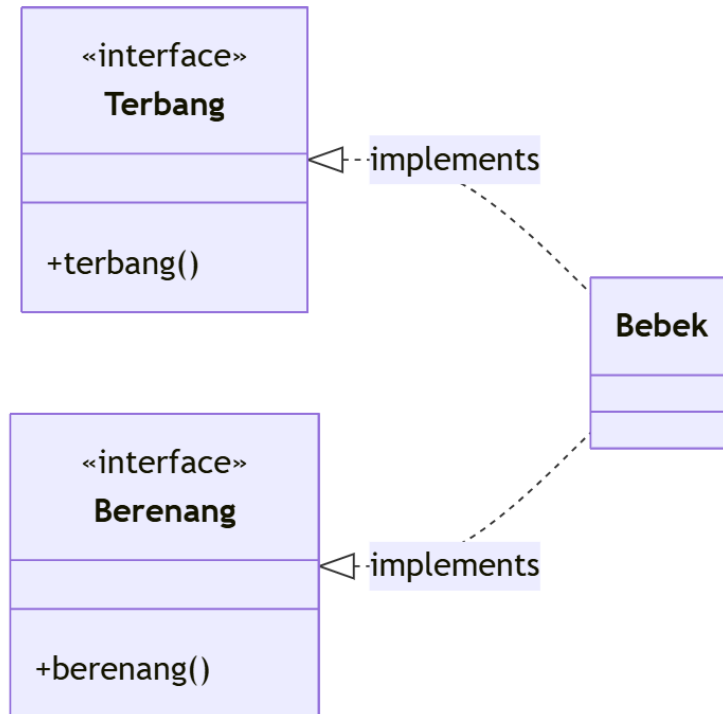
```
--- Hierarchical Inheritance ---
--- Hierarchical Inheritance ---
Luas Lingkaran (radius 7): 153.93804002589985
Luas Persegi (sisi 4): 16.0
PS C:\Users\user\Documents\projectpbo\tugaspraktikum>
```

d. Multiple Inheritance (Pewarisan Ganda)

Multiple Inheritance adalah kondisi di mana satu subclass mewarisi properti dan metode dari dua atau lebih superclass.

Hubungan: ClassC mewarisi dari ClassA dan ClassB.

Notasi UML: InterfaceA <|.. ClassC, InterfaceB <|.. ClassC.



```
praktikum4 > J Terbang.java > Bebek > main(String[])
1 // File: Terbang.java (Interface 1)
2 interface Terbang {
3     void terbang();
4 }
5
6 // File: Berenang.java (Interface 2)
7 interface Berenang {
8     void berenang();
9 }
10
11 // File: Bebek.java (Class mengimplementasikan kedua Interface)
12 class Bebek implements Terbang, Berenang {
13     @Override
14     public void terbang() {
15         System.out.println(x:"Bebek bisa terbang rendah.");
16     }
17
18     @Override
19     public void berenang() {
20         System.out.println(x:"Bebek bisa berenang di air.");
21     }
22
23     // Metode utama untuk menjalankan
24     Run | Debug
25     public static void main(String[] args) {
26         Bebek donald = new Bebek();
27
28         System.out.println(x:"--- Multiple Inheritance (via Interface) ---");
29
30         // Menanggil method dari Terbang dan Berenang
31     }
```

The screenshot shows the source code of the **Bebek** class in an IDE. The code defines two interfaces, **Terbang** and **Berenang**, and a class **Bebek** that implements both. The **Bebek** class overrides the `terbang()` and `berenang()` methods. The `main` method is also shown, creating an instance of **Bebek** and printing a message about multiple inheritance via interfaces. The IDE's Explorer and Outline panels are visible on the left, showing the project structure.

```

praktikum4 > J Terbang.java > Bebek
12 class Bebek implements Terbang, Berenang {
23     // Metode utama untuk menjalankan
Run | Debug
24     public static void main(String[] args) {
25         Bebek donald = new Bebek();
26
27         System.out.println(x:"--- Multiple Inheritance (via Interface) ---");
28
29         // Memanggil perilaku dari Interface Terbang
30         donald.terbang();
31
32         // Memanggil perilaku dari Interface Berenang
33         donald.berenang();
34     }
35 }

```

```

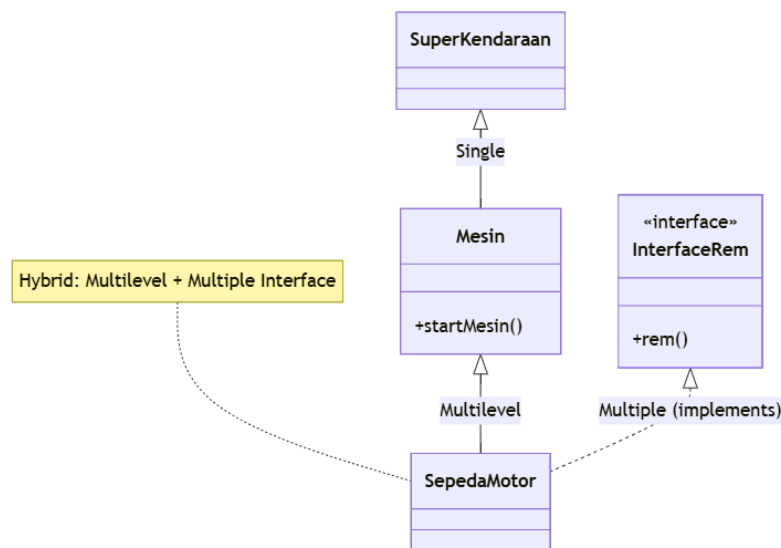
PS C:\Users\user\Documents\projectpbo\tugaspraktikum> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\user\AppData\Roaming\Code\User\workspaceStorage\1949db469fb2e4f317cb589a8d37ba84\redhat.java\jdt_ws\tugaspraktikum_885b5c39\bin' 'Bebek'
kum_885b5c39\bin' 'Bebek'
--- Multiple Inheritance (via Interface) ---
Bebek bisa terbang rendah.
Bebek bisa berenang di air.
PS C:\Users\user\Documents\projectpbo\tugaspraktikum>

```

e. Hybrid Inheritance (Pewarisan Hibrid)

Hybrid Inheritance adalah kombinasi dari dua atau lebih jenis inheritance (misalnya, kombinasi Multilevel dan Hierarchical). Karena batasan Java, Hybrid Inheritance sering kali melibatkan campuran Class (extends) dan Interface (implements).

Contoh: ClassD mewarisi properti dari ClassB (Multilevel) dan mengimplementasikan InterfaceX (Multiple).



```
1 // Interface (Multiple part)
2 interface InterfaceRem {
3     void rem();
4 }
5
6 // File: SuperKendaraan.java (Level 1)
7 class SuperKendaraan {
8     public void infoUmum() {
9         System.out.println(x:"Ini adalah kendaraan bermesin.");
10    }
11 }
12
13 // File: Mesin.java (Level 2 - Multilevel part)
14 class Mesin extends SuperKendaraan {
15     public void startMesin() {
16         System.out.println(x:"Mesin dihidupkan.");
17     }
18 }
19
20 // File: SepedaMotor.java (Level 3 - Hybrid)
21 class SepedaMotor extends Mesin implements InterfaceRem {
22     @Override
23     public void rem() {
24         System.out.println(x:"Sepeda motor mengerem.");
25     }
26
27     public void jalan() {
28         System.out.println(x:"Sepeda motor berjalan.");
29     }
30 }
```

```
21 class SepedaMotor extends Mesin implements InterfaceRem {
22     // Metode utama untuk menjalankan
23     public static void main(String[] args) {
24         SepedaMotor motor = new SepedaMotor();
25
26         System.out.println(x:"--- Hybrid Inheritance ---");
27
28         // Perilaku dari SuperKendaraan (Level 1 - Multilevel)
29         motor.infoUmum();
30
31         // Perilaku dari Mesin (Level 2 - Multilevel)
32         motor.startMesin();
33
34         // Perilaku dari SepedaMotor (Metode sendiri)
35         motor.jalan();
36
37         // Perilaku dari InterfaceRem (Multiple/Hybrid)
38         motor.rem();
39     }
40 }
```

--- Hybrid Inheritance ---
Ini adalah kendaraan bermesin.
Mesin dihidupkan.
Sepeda motor berjalan.
Sepeda motor mengerem.
PS C:\Users\user\Documents\projectpbo\tugaspraktikum>

Referensi :

1. Sugandi, Z. A. W., Nugraha, Y. A., Anam, S. N., & Darmayanti, I. (2022). Implementasi Konsep Pemrograman Berorientasi Objek Dalam Aplikasi Pembukuan Keuangan

- Penjual Jus Buah Menggunakan Bahasa Pemrograman Java. Jurnal IT CIDA, 8(1), 1–9.
2. Ilham, N. A. (2019). Implementasi Konsep Pemrograman Berorientasi Objek Pada Aplikasi Sistem Parkir Menggunakan Bahasa Pemrograman Java. Jurnal Edukasi Elektro, 3(2).
 3. Akil, I. (Tahun Tidak Ditemukan). Pemrograman Berorientasi Object Dengan Java Tingkat Dasar. Modul Ajar. Diakses dari Repositori Universitas BSI.
 4. Retnoningsih, E., Shadiq, J., & Oscar, D. (2017). Pembelajaran Pemrograman Berorientasi Objek (Object Oriented Programming) Berbasis Project Based Learning. INFORMATICS FOR EDUCATORS AND PROFESSIONALS, 2(1), 95–104.

Link : <https://github.com/caniaye/PraktikumPBO/>