

LAPORAN PRAKTIKUM

Studi Kasus



Disusun Oleh:
Dzakir Tsabit Asy Syafiq (241511071)
Jurusan Teknik Komputer dan Informatika

Program Studi D-3 Teknik Informatika
Politeknik Negeri Bandung
09/10/2025

STUDI KASUS 1 – ANOTHER TYPE OF EMPLOYEE (COMMISSION)

1. Analisis Masalah

Studi kasus ini meminta kita untuk menambahkan satu jenis karyawan baru ke dalam hierarki kelas yang sudah ada. Karyawan baru ini adalah seorang pekerja per jam yang juga mendapatkan komisi berdasarkan total penjualan.

Kelas baru ini akan dinamai Commission dan harus memenuhi persyaratan berikut:

- Mewarisi (extends) kelas Hourly.
- Memiliki dua variabel instance tambahan: totalSales (tipe double) untuk total penjualan dan commissionRate (tipe double) untuk persentase komisi.
- Memiliki constructor yang menerima 6 parameter (5 parameter untuk constructor Hourly dan 1 untuk commissionRate). Constructor ini harus memanggil constructor kelas induknya (super()).
- Memiliki metode addSales(double totalSales) untuk menambahkan jumlah penjualan.
- Meng-override metode pay() untuk menghitung total gaji (gaji per jam + komisi). Setelah pembayaran, total penjualan harus di-reset menjadi 0.
- Meng-override metode toString() untuk menampilkan informasi tambahan, yaitu total penjualan.

2. Implementasi Kode

Berikut adalah implementasi kode yang diperlukan untuk menyelesaikan studi kasus ini.

A. Kelas Commission.java

Ini adalah kelas baru yang dibuat sesuai dengan spesifikasi. Kelas ini mewarisi Hourly dan menambahkan fungsionalitas untuk menghitung komisi.

```
// Nama File: Commission.java
```

```
public class Commission extends Hourly {  
    private double totalSales;    // Total penjualan yang dibuat  
    private double commissionRate; // Tingkat komisi (dalam desimal, misal 0.20 untuk 20%)  
  
    /**  
     * Constructor untuk membuat objek Commission.  
     * Memanggil constructor dari kelas induk (Hourly) dan menginisialisasi variabel komisi.  
     */  
}
```

```

public Commission(String eName, String eAddress, String ePhone,
                  String socSecNumber, double rate, double commRate) {
    // Memanggil constructor kelas induk (Hourly)
    super(eName, eAddress, ePhone, socSecNumber, rate);

    // Mengatur tingkat komisi
    this.commissionRate = commRate;
    this.totalSales = 0; // Inisialisasi total penjualan awal adalah 0
}

/**
 * Metode untuk menambahkan total penjualan.
 */
public void addSales(double totalSales) {
    this.totalSales += totalSales;
}

/**
 * Meng-override metode pay() dari kelas induk.
 * Menghitung total gaji = (gaji per jam) + (gaji komisi).
 */
@Override
public double pay() {
    // 1. Hitung gaji per jam dengan memanggil metode pay() dari kelas induk (Hourly)
    double hourlyPayment = super.pay();

    // 2. Hitung gaji dari komisi
    double commissionPayment = totalSales * commissionRate;

    // 3. Reset total penjualan kembali ke 0 setelah dihitung
    totalSales = 0;

    // 4. Kembalikan total pembayaran
    return hourlyPayment + commissionPayment;
}

/**
 * Meng-override metode toString() untuk menambahkan informasi total penjualan.
 */

```

```

@Override
public String toString() {
    // Memanggil toString() dari kelas induk (Hourly) untuk mendapatkan info dasar
    String result = super.toString();

    // Menambahkan informasi total penjualan
    result += "\nTotal Sales: " + totalSales;

    return result;
}
}

```

Pertanyaan: The pay method must call the pay method of the parent class to compute the pay for hours worked then add to that the pay from commission on sales. (See the pay method in the Executive class.) The total sales should be set back to 0 (note: you don't need to set the hoursWorked back to 0—why not?)

Jawaban: Alasannya terletak pada **prinsip pewarisan dan pembagian tanggung jawab (separation of concerns)**.

1. **Tanggung Jawab Hourly:** Variabel hoursWorked adalah milik kelas Hourly. Metode pay() di dalam kelas Hourly sudah dirancang untuk melakukan dua hal: menghitung pembayaran berdasarkan payRate * hoursWorked dan kemudian **me-reset hoursWorked menjadi 0**.
2. **Panggilan super.pay():** Di dalam metode pay() kelas Commission, baris double hourlyPayment = super.pay(); dieksekusi. Panggilan super.pay() ini mengeksekusi seluruh kode yang ada di dalam metode pay() milik kelas induknya (Hourly). Artinya, proses penghitungan gaji per jam dan reset hoursWorked sudah **otomatis dilakukan** oleh kelas Hourly.
3. **Tanggung Jawab Commission:** Kelas Commission hanya bertanggung jawab atas variabel dan logika yang ia definisikan sendiri, yaitu totalSales dan commissionRate. Oleh karena itu, setelah komisi dihitung, adalah tugas metode pay() di Commission untuk me-reset totalSales ke 0.

Singkatnya, hoursWorked di-reset oleh Hourly.pay() dan totalSales di-reset oleh Commission.pay(). Ini adalah contoh desain yang baik di mana setiap kelas hanya mengelola state (variabel) miliknya sendiri

4. Latihan dan Hasil

Setelah memodifikasi Staff.java, program akan dijalankan dan menghasilkan output yang mencakup dua karyawan baru.

Perhitungan Gaji Karyawan Komisi

Berikut adalah rincian perhitungan manual untuk memverifikasi hasil program:

1. Karyawan 1 (Budi)

- Tarif per jam: \$6.25
- Jam kerja: 35
- Total penjualan: \$400
- Tingkat komisi: 20% (0.20)
- **Perhitungan:**
 - Gaji Per Jam: $35 \times \$6.25 = \218.75
 - Gaji Komisi: $\$400 \times 0.20 = \80.00
 - **Total Pembayaran:** $\$218.75 + \$80.00 = \$298.75$

2. Karyawan 2 (Siti)

- Tarif per jam: \$9.75
- Jam kerja: 40
- Total penjualan: \$950
- Tingkat komisi: 15% (0.15)
- **Perhitungan:**
 - Gaji Per Jam: $40 \times \$9.75 = \390.00
 - Gaji Komisi: $\$950 \times 0.15 = \142.50
 - **Total Pembayaran:** $\$390.00 + \$142.50 = \$532.50$

Ouput :

```

----- PAYDAY REPORT -----
Name: Sam
Address: 123 Main Line
Phone: 555-0469
Social Security Number: 123-45-6789
Paid: 2923.07
-----
Name: Carla
Address: 456 Off Line
Phone: 555-0101
Social Security Number: 987-65-4321
Paid: 1246.15
-----
Name: Woody
Address: 789 Off Rocker
Phone: 555-0000
Social Security Number: 010-20-3040
Paid: 1169.23
-----
Name: Diane
Address: 678 Fifth Ave.
Phone: 555-0690
Social Security Number: 958-47-3625
Current hours: 40
Paid: 422.0
-----
Name: Norm
Address: 987 Suds Blvd.
Phone: 555-8374
Thanks!
-----
Name: Cliff
Address: 321 Duds Lane
Phone: 555-7282
Thanks!
-----

```

Output Baru di bawah nya ada (potongan dari atas):

```

-----
Name: Budi
Address: Jl. Merdeka No. 10
Phone: 0812345678
Social Security Number: 111-22-333
Current hours: 35
Total Sales: 400.0
Paid: 298.75
-----
Name: Siti
Address: Jl. Kemerdekaan No. 20
Phone: 0898765432
Social Security Number: 444-55-666
Current hours: 40
Total Sales: 950.0
Paid: 532.5
-----

```

Secara keseluruhan, kode ini adalah simulasi sederhana dari sistem manajemen karyawan yang tujuan utamanya adalah untuk mendemonstrasikan **empat pilar utama Pemrograman Berorientasi Objek (PBO)**.

1. Pewarisan (Inheritance)

Apa itu? Inheritance adalah mekanisme di mana sebuah kelas baru (anak) dapat mewarisi properti (variabel) dan perilaku (metode) dari kelas yang sudah ada (induk). Ini memungkinkan penggunaan kembali kode dan menciptakan hierarki kelas yang logis.

Dalam Kode: Hierarki pewarisan sangat jelas terlihat

```
// StaffMember adalah kelas dasar (induk paling atas)
abstract public class StaffMember { ... } [cite: 112]

// Employee mewarisi dari StaffMember
public class Employee extends StaffMember { ... } [cite: 181]

// Hourly mewarisi dari Employee
public class Hourly extends Employee { ... } [cite: 265]

// Commission mewarisi dari Hourly
public class Commission extends Hourly { ... }
```

- Kelas Commission **mewarisi** dari Hourly.
- Hourly **mewarisi** dari Employee.
- Employee **mewarisi** dari StaffMember.

Ini berarti, sebuah objek Commission secara otomatis memiliki semua yang dimiliki Hourly, Employee, dan StaffMember. Contohnya, ia memiliki variabel name dan address dari StaffMember, variabel socialSecurityNumber dari Employee, dan metode addHours() dari Hourly. Konsep ini mencegah kita menulis ulang kode yang sama berulang kali.

2. Polimorfisme (Polymorphism)

Apa itu? Secara harfiah berarti "banyak bentuk". Dalam PBO, ini adalah kemampuan suatu objek untuk diperlakukan sebagai objek dari tipe induknya. Manfaat terbesarnya adalah kita dapat memanggil metode yang sama pada objek-objek yang berbeda tipe, dan masing-masing akan memberikan respons yang sesuai dengan tipenya.

Dalam Kode: Polimorfisme adalah dari file Staff.java

```
public class Staff {
    StaffMember[] staffList; [cite: 61]
    // ... (constructor di mana staffList diisi dengan berbagai jenis karyawan)

    // Metode inilah yang menunjukkan polimorfisme
    public void payday () {
```

```

double amount;

for (int count = 0; count < staffList.length; count++) {
    System.out.println (staffList[count]);

    // Panggilan polimorfik:
    // Java secara otomatis memilih metode pay() yang benar
    // sesuai tipe objek (Executive, Hourly, Commission, dll.)
    amount = staffList[count].pay(); [cite: 91]

    if (amount == 0.0) [cite: 92, 93]
        System.out.println ("Thanks!"); [cite: 94]
    else
        System.out.println ("Paid: " + amount); [cite: 96]

    System.out.println ("-----");
}
}
}

```

- Array `staffList` dideklarasikan sebagai `StaffMember[]`, yang merupakan tipe paling dasar (induk).
- Namun, array ini diisi dengan berbagai jenis objek turunan: `Executive`, `Employee`, `Hourly`, dan `Commission` yang baru dibuat.
- Di dalam loop `payday()`, ada satu baris kunci: `staffList[count].pay();`.

Ketika baris ini dieksekusi, Java secara otomatis tahu versi `pay()` mana yang harus dijalankan. Jika `staffList[count]` adalah objek `Hourly`, maka `pay()` dari kelas `Hourly` yang dijalankan. Jika itu adalah objek `Commission`, maka `pay()` dari kelas `Commission` yang dijalankan. Ini membuat kode menjadi sangat fleksibel dan mudah dikelola.

3. Penimpaan Metode (Method Overriding)

Apa itu? Ini adalah kemampuan kelas anak untuk menyediakan implementasi yang berbeda (lebih spesifik) untuk sebuah metode yang sudah didefinisikan di kelas induknya.

Dalam Kode: Kelas `Commission` melakukan *overriding* pada dua metode penting dari kelas induknya:

```

// Dari kelas Hourly.java (Induk)
// =====
public class Hourly extends Employee {
    private int hoursWorked;

```



```

// Metode pay() asli di kelas induk
public double pay() {
    double payment = payRate * hoursWorked; [cite: 293]
    hoursWorked = 0; [cite: 292]
    return payment; [cite: 294]
}

// Metode toString() asli di kelas induk
public String toString() {
    String result = super.toString(); [cite: 302]
    result += "\nCurrent hours: " + hoursWorked; [cite: 306]
    return result; [cite: 307]
}
// ...
}

// Dari kelas Commission.java (Anak)
// =====
public class Commission extends Hourly {
    private double totalSales;
    private double commissionRate;

    // Metode pay() yang di-override
    @Override
    public double pay() {
        // Memanggil pay() dari Hourly, lalu menambahkan logika komisi
        double hourlyPayment = super.pay(); // Memanggil kode dari induk
        double commissionPayment = totalSales * commissionRate;
        totalSales = 0;
        return hourlyPayment + commissionPayment;
    }

    // Metode toString() yang di-override
    @Override
    public String toString() {
        // Memanggil toString() dari Hourly, lalu menambahkan info sales
        String result = super.toString(); // Memanggil kode dari induk

```

```

        result += "\nTotal Sales: " + totalSales;
        return result;
    }
    // ...
}

```

- **public double pay():** Kelas Hourly sudah punya metode pay() untuk menghitung gaji berdasarkan jam. Kelas Commission menyimpannya dengan menambahkan fungsionalitas baru: ia memanggil metode pay() milik induknya menggunakan super.pay() untuk mendapatkan gaji per jam, lalu menambahkan perhitungan komisi di atasnya.
- **public String toString():** Metode ini juga di-override untuk menambahkan informasi "Total Sales" ke dalam detail karyawan, selain informasi dasar yang sudah disediakan oleh toString() milik kelas Hourly.

4. Abstraksi (Abstraction)

Apa itu? Abstraksi berarti menyembunyikan detail implementasi yang kompleks dan hanya menampilkan fungsionalitas yang penting bagi pengguna. Salah satu caranya adalah melalui abstract class.

Dalam Kode: StaffMember adalah sebuah **kelas abstrak**

```

abstract public class StaffMember { [cite: 112]
    protected String name; [cite: 114]
    protected String address; [cite: 115]
    protected String phone; [cite: 116]

    // Constructor
    public StaffMember (String eName, String eAddress, String ePhone) { [cite: 119]
        name = eName; [cite: 123]
        address = eAddress; [cite: 124]
        phone = ePhone; [cite: 125, 126]
    }

    // Metode toString() biasa
    public String toString() { [cite: 132]
        String result = "Name: " + name + "\n"; [cite: 134]
        result += "Address: " + address + "\n"; [cite: 136]
        result += "Phone: " + phone; [cite: 137, 138]
        return result; [cite: 139]
    }
}

```

```
// Metode abstract:
// Tidak memiliki isi dan harus di-override oleh kelas anak.
public abstract double pay(); [cite: 145]
}
```

- Kita tidak bisa membuat objek langsung dari StaffMember (misalnya, new StaffMember(...) akan error).
- Ia memiliki metode public abstract double pay();. Ini adalah sebuah "kontrak" yang memaksa setiap kelas turunan (seperti Employee atau Volunteer) untuk *pasti* memiliki implementasi metode pay() mereka sendiri. Ini menjamin bahwa setiap jenis anggota staf dapat "dibayar", bahkan jika pembayarannya nol seperti pada Volunteer.

Kesimpulan Akhir

Kode ini secara efektif menunjukkan bagaimana pilar-pilar PBO bekerja sama untuk menciptakan sebuah sistem yang **terstruktur, fleksibel, dan mudah diperluas**. Dengan menggunakan **abstraksi** (StaffMember) sebagai dasar, **pewarisan** digunakan untuk menciptakan berbagai jenis karyawan. **Polimorfisme** memungkinkan semua jenis karyawan tersebut dikelola dalam satu daftar dan diperlakukan secara seragam, sementara **method overriding** memungkinkan setiap jenis karyawan memiliki perilaku khusus sesuai kebutuhannya

STUDI KASUS 2 – PAINTING SHAPES

Analisis Masalah

Studi kasus ini meminta kita untuk membangun sebuah sistem kecil berbasis Java untuk menghitung kebutuhan cat dalam mengecat berbagai bentuk objek. Tugas utamanya adalah mengimplementasikan konsep **Pewarisan (Inheritance)** dan **Polimorfisme** dengan membuat sebuah hierarki kelas bentuk (Shapes).

Tugas-tugas yang harus diselesaikan adalah:

1. Membuat kelas abstrak Shape sebagai kelas induk.
2. Membuat kelas turunan Rectangle dan Cylinder yang mewarisi Shape.
3. Memperbaiki logika perhitungan di dalam kelas Paint.
4. Melengkapi program utama di PaintThings.java untuk membuat objek-objek bentuk dan menghitung kebutuhan cat untuk masing-masing.

Implementasi Kode

Berikut adalah kode lengkap untuk semua file yang perlu dibuat atau dimodifikasi agar program dapat berjalan sesuai permintaan.

1. Shape.java (File Baru)

Ini adalah kelas abstrak yang menjadi dasar bagi semua bentuk lainnya

// File: Shape.java

```
public abstract class Shape {  
    // Variabel instance untuk nama bentuk  
    private String shapeName; [cite: 7]  
  
    /**  
     * Constructor untuk mengatur nama bentuk.  
     * Dipanggil oleh kelas anak menggunakan super().  
     */  
    public Shape(String name) {  
        this.shapeName = name;  
    }  
  
    /**  
     * Metode abstrak untuk menghitung luas.  
     * Harus diimplementasikan oleh setiap kelas anak.  
     */  
}
```

```

    public abstract double area(); [cite: 7]

    /**
     * Mengembalikan nama bentuk sebagai String.
     */
    public String toString() {
        return shapeName; [cite: 8]
    }
}

```

2. Rectangle.java (File Baru)

Kelas turunan dari Shape untuk merepresentasikan bentuk persegi panjang.

// File: Rectangle.java

```

public class Rectangle extends Shape {
    private double length;
    private double width;

    /**
     * Constructor: Mengatur nama, panjang, dan lebar persegi panjang.
     */
    public Rectangle(double l, double w) {
        super("Rectangle"); // Memanggil constructor kelas induk
        this.length = l;
        this.width = w;
    }

    /**
     * Menghitung dan mengembalikan luas persegi panjang.
     * Rumus: panjang * lebar
     */
    @Override
    public double area() {
        return length * width; [cite: 12]
    }

    /**
     * Mengembalikan informasi persegi panjang sebagai String.
     */
}

```

```

@Override

public String toString() {
    return super.toString() + " of length " + length + " and width " + width;
}
}

```

3. Cylinder.java (File Baru)

Kelas turunan dari Shape untuk merepresentasikan bentuk silinder

// File: Cylinder.java

```

public class Cylinder extends Shape {
    private double radius;
    private double height;

    /**
     * Constructor: Mengatur nama, radius, dan tinggi silinder.
     */
    public Cylinder(double r, double h) {
        super("Cylinder"); // Memanggil constructor kelas induk
        this.radius = r;
        this.height = h;
    }

    /**
     * Menghitung dan mengembalikan luas permukaan silinder.
     * Rumus sesuai instruksi:  $PI * radius^2 * tinggi$ 
     */
    @Override
    public double area() {
        return Math.PI * radius * radius * height; [cite: 13]
    }

    /**
     * Mengembalikan informasi silinder sebagai String.
     */
    @Override
    public String toString() {
        return super.toString() + " of radius " + radius + " and height " + height;
    }
}

```

```
}
```

4. Paint.java (File yang Dimodifikasi)

Kelas ini diperbaiki untuk menghitung jumlah cat yang dibutuhkan dengan benar

```
// File: Paint.java
```

```
public class Paint {  
    private double coverage; // jumlah kaki persegi per galon [cite: 73]  
  
    /**  
     * Constructor: Mengatur objek cat.  
     */  
    public Paint(double c) {  
        coverage = c; [cite: 79]  
    }  
  
    /**  
     * Mengembalikan jumlah cat (dalam galon) yang dibutuhkan  
     * untuk mengecat bentuk yang diberikan sebagai parameter.  
     */  
    public double amount(Shape s) {  
        System.out.println("Computing amount for " + s);  
        // PERBAIKAN: Menghitung jumlah cat dengan benar  
        // Rumus: luas / cakupan  
        return s.area() / coverage; [cite: 17]  
    }  
}
```

5. PaintThings.java (File yang Dimodifikasi)

Program utama yang dilengkapi untuk membuat objek dan menghitung kebutuhan cat

```
// File: PaintThings.java
```

```
import java.text.DecimalFormat;  
  
public class PaintThings {  
    /**  
     * Membuat beberapa objek bentuk dan sebuah objek Paint  
     * lalu mencetak jumlah cat yang dibutuhkan untuk mengecat setiap bentuk.  
     */  
    public static void main(String[] args) {
```

```

    final double COVERAGE = 350; [cite: 108]
    Paint paint = new Paint(COVERAGE); [cite: 109]

    Rectangle deck;
    Sphere bigBall;
    Cylinder tank;

    double deckAmt, ballAmt, tankAmt;

    // Instansiasi tiga bentuk untuk dicat [cite: 114]
    deck = new Rectangle(20, 35); [cite: 21]
    bigBall = new Sphere(15); [cite: 21]
    tank = new Cylinder(10, 30); [cite: 21]

    // Hitung jumlah cat yang dibutuhkan untuk setiap bentuk [cite: 115]
    deckAmt = paint.amount(deck); [cite: 22]
    ballAmt = paint.amount(bigBall); [cite: 22]
    tankAmt = paint.amount(tank); [cite: 22]

    // Cetak jumlah cat untuk setiap bentuk [cite: 116]
    DecimalFormat fmt = new DecimalFormat("0.#"); [cite: 117]
    System.out.println("\nNumber of gallons of paint needed...");
    System.out.println("Deck " + fmt.format(deckAmt)); [cite: 120]
    System.out.println("Big Ball " + fmt.format(ballAmt)); [cite: 121]
    System.out.println("Tank " + fmt.format(tankAmt)); [cite: 122]
}
}

```

6. Sphere.java (File dari Studi Kasus)

File ini sudah disediakan dalam studi kasus dan tidak perlu diubah

```

// File: Sphere.java
public class Sphere extends Shape {
    private double radius; //radius in feet [cite: 34]

    // Constructor: Sets up the sphere. [cite: 36]
    public Sphere (double r) {
        super ("Sphere"); [cite: 40]
        radius = r;
    }
}

```



```

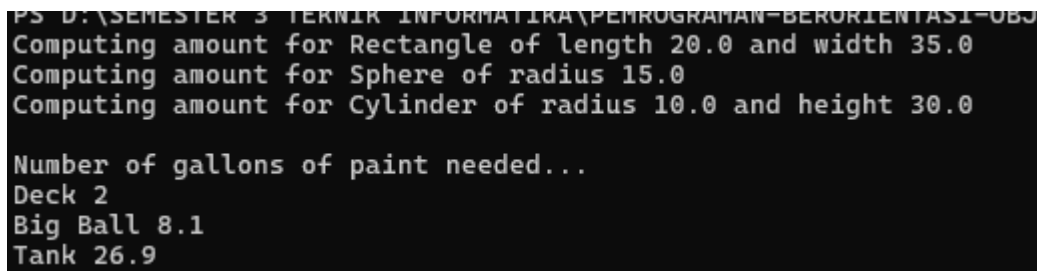
// Returns the surface area of the sphere. [cite: 44]
public double area() {
    return 4*Math.PI*radius*radius; [cite: 48]
}

// Returns the sphere as a String. [cite: 54]
public String toString() {
    return super.toString() + " of radius " + radius;
}
}

```

Catatan Penting: Instruksi pada PDF memberikan rumus $\text{PI} \times \text{radius}^2 \times \text{height}$ untuk "area" silinder. Perlu diketahui bahwa ini sebenarnya adalah rumus untuk **volume**. Rumus luas permukaan total silinder yang benar adalah $2 \times \text{PI} \times \text{radius} \times \text{height} + 2 \times \text{PI} \times \text{radius}^2$. Namun, untuk memenuhi permintaan studi kasus, kode di atas menggunakan rumus yang diberikan

Output yang dihasilkan:



```

PS D:\SEMESTER 3 TEKNIK INFORMATIKA\PEMROGRAMAN-BERORIENTASI-OBJEK>
Computing amount for Rectangle of length 20.0 and width 35.0
Computing amount for Sphere of radius 15.0
Computing amount for Cylinder of radius 10.0 and height 30.0

Number of gallons of paint needed...
Deck 2
Big Ball 8.1
Tank 26.9

```

Kode dalam studi kasus ini secara efektif mendemonstrasikan bagaimana **empat pilar utama Pemrograman Berorientasi Objek (PBO)** digunakan untuk membangun sebuah program yang fleksibel dan terstruktur dengan baik. Tujuannya adalah menghitung kebutuhan cat untuk berbagai bentuk geometris.

1. Abstraksi (Abstraction)

Apa itu? Abstraksi adalah konsep menyembunyikan detail implementasi yang rumit dan hanya menampilkan fungsionalitas esensial. Ini dicapai melalui kelas abstrak (abstract class) yang berfungsi sebagai "cetak biru" atau kontrak untuk kelas-kelas turunannya.

Dalam Kode: Konsep ini diwakili oleh kelas Shape.

- Kelas Shape dideklarasikan sebagai abstract class, yang berarti objek new Shape() tidak bisa dibuat secara langsung.
- Di dalamnya, terdapat metode public abstract double area();. Metode ini tidak memiliki isi dan berfungsi sebagai aturan yang **memaksa** setiap kelas turunan (seperti Sphere, Rectangle, dan Cylinder) untuk menyediakan implementasi atau rumus perhitungan area() versi mereka sendiri.

2. Pewarisan (Inheritance)

Apa itu? Pewarisan memungkinkan sebuah kelas baru (anak) untuk mengambil alih properti dan metode dari kelas yang sudah ada (induk). Hal ini mendorong penggunaan kembali kode (*code reusability*).

Dalam Kode: Semua kelas bentuk mewarisi sifat dasar dari kelas Shape.

- public class Sphere extends Shape
- public class Rectangle extends Shape
- public class Cylinder extends Shape

Karena pewarisan ini, setiap kelas bentuk (Sphere, Rectangle, Cylinder) secara otomatis memiliki variabel shapeName dan metode dasar toString() tanpa perlu menuliskannya ulang. Mereka hanya perlu memanggil super("NamaBentuk") di dalam *constructor* mereka untuk menginisialisasi properti warisan tersebut.

3. Polimorfisme (Polymorphism)

Apa itu? Polimorfisme (berarti "banyak bentuk") adalah kemampuan untuk memproses objek dari kelas yang berbeda seolah-olah mereka adalah objek dari kelas induk yang sama. Ini membuat kode menjadi sangat fleksibel.

Dalam Kode: Polimorfisme adalah kunci dari fungsionalitas kelas Paint.

- Metode public double amount(Shape s) dirancang untuk menerima parameter bertipe Shape.
- Karena polimorfisme, metode ini dapat menerima objek *apapun* yang merupakan turunan dari Shape, baik itu Sphere, Rectangle, maupun Cylinder.
- Di dalam PaintThings.java, saat paint.amount(deck), paint.amount(bigBall), dan paint.amount(tank) dipanggil, satu metode yang sama digunakan untuk menangani berbagai jenis objek. Di dalam metode amount, baris s.area() secara otomatis akan memanggil versi area() yang benar sesuai dengan tipe objek s yang sedang diproses.

4. Penimpaan Metode (Method Overriding)

Apa itu? Ini adalah kemampuan kelas anak untuk menyediakan implementasi spesifik dari sebuah metode yang sudah ada di kelas induknya.

Dalam Kode: Konsep ini paling jelas terlihat pada implementasi metode area() dan toString().

- **Metode area():** Setiap kelas turunan **wajib** meng-override metode area() yang abstrak dari kelas Shape. Sphere mengisinya dengan rumus luas bola, Rectangle dengan rumus luas persegi panjang, dan Cylinder dengan rumus yang telah ditentukan.

- **Metode toString():** Setiap kelas bentuk juga meng-override metode toString(). Mereka memanggil super.toString() untuk mendapatkan nama bentuk dari kelas induk, lalu menambahkan detail spesifik mereka sendiri seperti "of radius..." atau "of length and width...".

Kesimpulan Akhir

Studi kasus ini menunjukkan alur kerja PBO yang ideal: **Abstraksi** mendefinisikan sebuah "kontrak" umum untuk semua bentuk. **Pewarisan** digunakan untuk membuat bentuk-bentuk spesifik yang mematuhi kontrak tersebut. **Method Overriding** memungkinkan setiap bentuk memiliki perilaku (rumus) yang unik. Terakhir, **Polimorfisme** menyatukan semuanya, memungkinkan kode lain (seperti kelas Paint) untuk berinteraksi dengan semua bentuk tersebut secara seragam dan elegan.

STUDI KASUS 3 – POLYMORPHIC SORTING

Analisis Masalah

Studi kasus ini berfokus pada penggunaan sebuah kelas Sorting generik yang dapat mengurutkan berbagai jenis objek, selama objek tersebut mengimplementasikan antarmuka (interface) Comparable. Tugas-tugasnya meliputi:

1. Memperbaiki program pengurutan angka (Numbers.java) dengan memahami perbedaan antara tipe data primitif dan objek.
2. Membuat program serupa untuk mengurutkan String (Strings.java).
3. Memodifikasi algoritma insertionSort agar mengurutkan secara menurun (*descending*).
4. Melengkapi logika perbandingan kustom di kelas Salesperson berdasarkan penjualan dan nama.
5. Menguji semua fungsionalitas menggunakan program WeeklySales.java.

Implementasi & Penjelasan Kode

Berikut adalah kode yang telah diperbaiki dan dibuat untuk menyelesaikan semua bagian dari studi kasus.

Bagian 1 & 2: Memperbaiki Numbers.java

Masalah: Kode Numbers.java asli menggunakan `int[]`, yang merupakan tipe data primitif. Kelas Sorting mengharapkan `Comparable[]`, yaitu sebuah array objek yang dapat dibandingkan. Tipe `int` bukanlah objek dan tidak mengimplementasikan `Comparable`, sehingga menyebabkan error kompilasi.

Solusi: Mengganti tipe array dari `int[]` menjadi `Integer[]`. `Integer` adalah kelas pembungkus (*wrapper class*) untuk `int` yang merupakan objek dan sudah mengimplementasikan `Comparable`.

// File: Numbers.java (Sudah diperbaiki)

```
import java.util.Scanner;
```

```
public class Numbers {
```

```
    public static void main(String[] args) {
```

```
        // PERUBAHAN: Menggunakan Integer[] bukan int[]
```

```
        Integer[] intList;
```

```
        int size;
```

```
        Scanner scan = new Scanner(System.in);
```

```
        System.out.print("\nBerapa banyak integer yang akan diurutkan? ");
```

```
        size = scan.nextInt();
```

```

        intList = new Integer[size];

        System.out.println("\nMasukkan angka-angka...");
        for (int i = 0; i < size; i++) {
            intList[i] = scan.nextInt(); // Autoboxing mengubah int menjadi Integer
        }

        // Memanggil insertionSort untuk Bagian 4
        Sorting.insertionSort(intList);

        System.out.println("\nAngka-angka setelah diurutkan (menurun):");
        for (int i = 0; i < size; i++) {
            System.out.print(intList[i] + " ");
        }
        System.out.println();
    }
}

```

Bagian 3: Membuat Strings.java

Program ini dibuat dengan menyalin Numbers.java yang sudah diperbaiki dan mengubah tipe data dari Integer menjadi String, serta metode input dari scan.nextInt() menjadi scan.next().

// File: Strings.java (File Baru)

```

import java.util.Scanner;

public class Strings {
    public static void main(String[] args) {
        String[] stringList;
        int size;
        Scanner scan = new Scanner(System.in);

        System.out.print("\nBerapa banyak string yang akan diurutkan? ");
        size = scan.nextInt();
        stringList = new String[size];

        System.out.println("\nMasukkan string...");
        for (int i = 0; i < size; i++) {
            stringList[i] = scan.next();
        }
    }
}

```

```

        // Memanggil insertionSort untuk Bagian 4
        Sorting.insertionSort(stringList);

        System.out.println("\nString setelah diurutkan (menurun):");
        for (int i = 0; i < size; i++) {
            System.out.print(stringList[i] + " ");
        }
        System.out.println();
    }
}

```

Bagian 4: Mengurutkan Menurun (Sorting.java)

Masalah: Modifikasi insertionSort agar mengurutkan secara menurun.

Solusi: Logika pengurutan ditentukan oleh hasil compareTo. Untuk urutan menaik (*ascending*), kondisi `key.compareTo(list[position-1]) < 0` digunakan. Untuk mengubahnya menjadi menurun (*descending*), kondisi ini dibalik menjadi `key.compareTo(list[position-1]) > 0`

```

// File: Sorting.java (insertionSort dimodifikasi)
public class Sorting {
    // ... (metode selectionSort tetap sama)

    public static void insertionSort(Comparable[] list) {
        for (int index = 1; index < list.length; index++) {
            Comparable key = list[index];
            int position = index;

            // PERUBAHAN: Ubah '< 0' menjadi '> 0' untuk urutan menurun
            while (position > 0 && key.compareTo(list[position - 1]) > 0) {
                list[position] = list[position - 1];
                position--;
            }
            list[position] = key;
        }
    }
}

```

Bagian 5: Melengkapi Salesperson.java

Masalah: Melengkapi metode compareTo untuk mengurutkan berdasarkan total penjualan (menurun), dan jika sama, berdasarkan nama (abjad).

Solusi: Logika compareTo diimplementasikan sebagai berikut:

1. Bandingkan totalSales. Jika berbeda, kembalikan hasilnya.
2. Jika penjualan sama, bandingkan lastName. Jika berbeda, kembalikan hasilnya.
3. Jika lastName juga sama, bandingkan firstName

```
// File: Salesperson.java (compareTo dilengkapi)
public class Salesperson implements Comparable {
    private String firstName, lastName;
    private int totalSales;

    // ... (constructor dan metode lain tetap sama)

    @Override
    public int compareTo(Object other) {
        int result;
        Salesperson otherSalesperson = (Salesperson) other;

        // Bandingkan berdasarkan total penjualan
        if (this.totalSales < otherSalesperson.getSales()) {
            result = -1;
        } else if (this.totalSales > otherSalesperson.getSales()) {
            result = 1;
        } else {
            // Jika penjualan sama, bandingkan berdasarkan nama belakang
            result = this.lastName.compareTo(otherSalesperson.getLastName());

            // Jika nama belakang sama, bandingkan berdasarkan nama depan
            if (result == 0) {
                result = this.firstName.compareTo(otherSalesperson.getFirstName());
            }
        }
        return result;
    }
    // ... (getters tetap sama)
}
```

Bagian 6 & 7: Menguji dengan WeeklySales.java

Versi ini memperbaiki *loop* dan menjalankan pengujian seperti yang diminta di Bagian 6

```
// File: WeeklySales.java (Sudah diperbaiki)
public class WeeklySales {
```

```

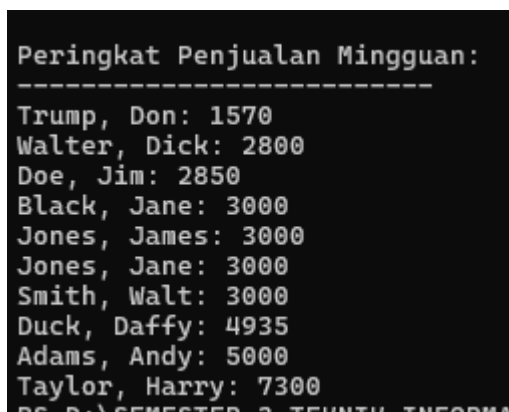
public static void main(String[] args) {
    Salesperson[] salesStaff = new Salesperson[10];
    salesStaff[0] = new Salesperson("Jane", "Jones", 3000);
    salesStaff[1] = new Salesperson("Daffy", "Duck", 4935);
    salesStaff[2] = new Salesperson("James", "Jones", 3000);
    salesStaff[3] = new Salesperson("Dick", "Walter", 2800);
    salesStaff[4] = new Salesperson("Don", "Trump", 1570);
    salesStaff[5] = new Salesperson("Jane", "Black", 3000);
    salesStaff[6] = new Salesperson("Harry", "Taylor", 7300);
    salesStaff[7] = new Salesperson("Andy", "Adams", 5000);
    salesStaff[8] = new Salesperson("Jim", "Doe", 2850);
    salesStaff[9] = new Salesperson("Walt", "Smith", 3000);

    Sorting.insertionSort(salesStaff);

    System.out.println("\nPeringkat Penjualan Mingguan:");
    System.out.println("-----");
    // PERBAIKAN: Loop for-each yang benar
    for (Salesperson s : salesStaff) {
        System.out.println(s);
    }
}
}

```

OutPut yang dihasilkan:



```

Peringkat Penjualan Mingguan:
-----
Trump, Don: 1570
Walter, Dick: 2800
Doe, Jim: 2850
Black, Jane: 3000
Jones, James: 3000
Jones, Jane: 3000
Smith, Walt: 3000
Duck, Daffy: 4935
Adams, Andy: 5000
Taylor, Harry: 7300

```

Kesimpulan:

Studi kasus ini secara efektif menunjukkan kekuatan polimorfisme dalam algoritma pengurutan. Sebuah metode sorting tunggal dapat mengurutkan berbagai tipe objek (Integer, String, Salesperson) tanpa mengetahui detail internalnya. Kuncinya terletak pada antarmuka Comparable yang menyediakan "kontrak" standar (compareTo) bagi objek-objek tersebut

untuk dapat dibandingkan satu sama lain, sehingga memungkinkan logika pengurutan yang generik dan dapat digunakan kembali.