

**LAPORAN PRAKTIKUM**  
**PEMROGRAMAN BERORIENTASI OBJEK**  
**CSF308 - CR001**



Dosen Pengampu:

Arief Ichwani, ST, MT

Disusun Oleh:

Arditya Adjie Rosandi

NIM: 20230801274

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**FAKULTAS ILMU KOMPUTER**  
**UNIVERSITAS ESA UNGGUL**  
**TAHUN AKADEMIK 2024 – 2025**

## DAFTAR ISI

<b>LAPORAN PRAKTIKUM 1.....</b>	<b>2</b>
<b>1.1    Penjelasan Kode Pertemuan 3 .....</b>	<b>2</b>
<b>LAPORAN PRAKTIKUM 2.....</b>	<b>9</b>
<b>2.1    Penjelasan Kode Pertemuan 4 .....</b>	<b>9</b>
<b>LAPORAN PRAKTIKUM 3.....</b>	<b>15</b>
<b>3.1    Penjelasan Kode Pertemuan 5 .....</b>	<b>15</b>
<b>LAPORAN PRAKTIKUM 4.....</b>	<b>21</b>
<b>4.1    Penjelasan Kode Pertemuan 6 .....</b>	<b>21</b>
<b>LAPORAN PRAKTIKUM 5.....</b>	<b>31</b>
<b>5.1    Penjelasan Kode Pertemuan 7 .....</b>	<b>31</b>
<b>LAPORAN PRAKTIKUM 6.....</b>	<b>51</b>
<b>6.1    Penjelasan Kode Pertemuan 8 .....</b>	<b>51</b>
<b>LAPORAN PRAKTIKUM 7.....</b>	<b>69</b>
<b>7.1    Penjelasan Kode Pertemuan 9 .....</b>	<b>69</b>
<b>LAPORAN PRAKTIKUM 8.....</b>	<b>77</b>
<b>8.1    Penjelasan Kode Pertemuan 10 .....</b>	<b>77</b>
<b>LAPORAN PRAKTIKUM 9.....</b>	<b>96</b>
<b>9.1    Penjelasan Kode Pertemuan 11 .....</b>	<b>96</b>
<b>LAPORAN PRAKTIKUM 10.....</b>	<b>106</b>
<b>10.1   Penjelasan Kode Pertemuan 12 .....</b>	<b>106</b>
<b>LAPORAN PRAKTIKUM 11.....</b>	<b>123</b>
<b>11.1   Penjelasan Kode Pertemuan 13 .....</b>	<b>123</b>

# LAPORAN PRAKTIKUM 1

## 1.1 Penjelasan Kode Pertemuan 3

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

**Import Statement:** Mengimpor kelas Scanner dari paket java.util, yang digunakan untuk membaca input dari pengguna melalui konsol.

```
java
public class Kalkulator {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
```

**Deklarasi Kelas:** Mendefinisikan kelas Kalkulator.

**Method main:** Titik masuk program. Semua eksekusi dimulai dari sini.

**Membuat Objek Scanner:** Membuat objek input dari kelas Scanner untuk membaca input dari pengguna.

```
java
while (true) {
    // Clear console
    System.out.print("\033[H\033[2J");
    System.out.print("\nNAMA : ARDITYA ADJIE ROSANDI");
    System.out.print("\nNIM : 20230801274");
```

**Loop Tanpa Henti:** Menggunakan while (true) untuk membuat loop yang akan terus berulang hingga pengguna memilih untuk keluar.

**Membersihkan Konsol:** Menggunakan escape sequence `\033[H\033[2J` untuk membersihkan layar konsol (berfungsi di banyak terminal).

**Menampilkan Informasi Pengguna:** Menampilkan nama dan NIM pengguna.

```
java
```

□ Salin kode

```
int menu, pilihan;  
double a, b;  
double hasil;
```

**Deklarasi Variabel:** Mendeklarasikan variabel menu dan pilihan sebagai tipe int, serta a, b, dan hasil sebagai tipe double. Variabel ini digunakan untuk menyimpan input pengguna dan hasil kalkulasi.

```
java
```

□ Salin kode

```
System.out.println("\nKALKULATOR BY GAMMAURA");  
System.out.println("\n1. KALKULATOR A");  
System.out.println("2. KALKULATOR B");  
System.out.println("\n0. KELUAR");
```

**Menampilkan Menu:** Menampilkan menu utama kalkulator, termasuk pilihan untuk Kalkulator A, Kalkulator B, dan opsi untuk keluar dari program.

```
java
```

□ Salin kode

```
System.out.print("\nMASUKKAN PILIHAN : ");  
menu = input.nextInt();
```

**Membaca Pilihan Menu:** Meminta pengguna untuk memasukkan pilihan menu dan menyimpannya di variabel menu

```
java
```

□ Salin kode

```
if (menu == 1) {  
    System.out.print("\033[H\033[2J");  
    System.out.println("\nKALKULATOR A");  
    System.out.println("\n1. PENJUMLAHAN");  
    System.out.println("2. PENGURANGAN");  
    System.out.println("3. PERKALIAN");  
    System.out.println("4. PEMBAGIAN");
```

**Memeriksa Pilihan Menu:** Jika pengguna memilih menu 1, program akan membersihkan konsol dan menampilkan submenu untuk Kalkulator A, dengan pilihan untuk operasi matematika dasar.

```
java
```

□ Salin kode

```
System.out.print("\nMASUKKAN PILIHAN : ");
pilihan = input.nextInt();
```

**Membaca Pilihan Operasi:** Meminta pengguna untuk memilih operasi (penjumlahan, pengurangan, perkalian, atau pembagian) dan menyimpannya di variabel pilihan.

```
java
```

□ Salin kode

```
// Mengambil dua angka
System.out.print("\nMASUKKAN ANGKA PERTAMA : ");
a = input.nextDouble();
System.out.print("MASUKKAN ANGKA KEDUA : ");
b = input.nextDouble();
```

**Membaca Angka:** Meminta pengguna untuk memasukkan dua angka yang akan digunakan dalam perhitungan dan menyimpannya di variabel a dan b.

```
java
```

□ Salin kode

```
switch (pilihan) {
    case 1:
        hasil = a + b;
        System.out.println("\nHASIL PENJUMLAHAN = " + hasil);
        break;
```

**Pernyataan Switch:** Menggunakan switch untuk memeriksa nilai pilihan.

**Operasi Penjumlahan:** Jika pilihan adalah 1, menghitung penjumlahan dan mencetak hasilnya.

```
java
```

□ Salin kode

```
case 2:
    hasil = a - b;
    System.out.println("\nHASIL PENGURANGAN = " + hasil);
    break;
```

**Operasi Pengurangan:** Jika pilihan adalah 2, menghitung pengurangan dan mencetak hasilnya.

```
java
```

Salin kode

```
case 3:  
    hasil = a * b;  
    System.out.println("\nHASIL PERKALIAN = " + hasil);  
    break;
```

**Operasi Perkalian:** Jika pilihan adalah 3, menghitung perkalian dan mencetak hasilnya.

```
java
```

Salin kode

```
case 4:  
    if (b == 0) {  
        System.out.println("\nHASIL PEMBAGIAN = TAK HINGGA");  
    } else {  
        hasil = a / b;  
        System.out.println("\nHASIL PEMBAGIAN = " + hasil);  
    }  
    break;
```

**Operasi Pembagian:** Jika pilihan adalah 4, program memeriksa apakah angka kedua b sama dengan 0. Jika ya, mencetak pesan kesalahan. Jika tidak, menghitung pembagian dan mencetak hasilnya.

```
java
```

Salin kode

```
default:  
    System.out.println("PILIHAN TIDAK VALID!");
```

**Default Case:** Jika pilihan tidak valid, mencetak pesan kesalahan.

```
java
```

Salin kode

```
// Menunggu pengguna untuk melanjutkan  
System.out.print("\nKETIK APA SAJA UNTUK MELANJUTKAN : ");  
input.nextLine(); // Membersihkan buffer  
input.nextLine(); // Menunggu input dari pengguna
```

**Menunggu Input Pengguna:** Meminta pengguna untuk mengetikkan sesuatu sebelum melanjutkan ke langkah berikutnya.

```
java
```

Salin kode

```
else if (menu == 2) {  
    System.out.print("\033[H\033[2J");  
    System.out.println("\nKALKULATOR B");  
    System.out.print("\nMASUKKAN ANGKA PERTAMA : ");  
    double angka1 = input.nextDouble();
```

**Pilihan Menu 2:** Jika pengguna memilih menu 2, membersihkan konsol dan meminta angka pertama untuk Kalkulator B.

```
java
```

Salin kode

```
// Menampilkan opsi operator  
System.out.println("\n(TAMBAH GUNAKAN => (+));  
System.out.println("(KURANG GUNAKAN => (-));  
System.out.println("(KALI GUNAKAN => (*));  
System.out.println("(BAGI GUNAKAN => (/));  
System.out.print("\nMASUKKAN OPERATOR : ");  
char operator = input.next().charAt(0);
```

**Menampilkan Operator:** Menampilkan opsi untuk operator yang bisa digunakan dalam Kalkulator B dan meminta pengguna untuk memasukkan operator.

```
java
```

Salin kode

```
System.out.print("\nMASUKKAN ANGKA KEDUA : ");  
double angka2 = input.nextDouble();
```

**Membaca Angka Kedua:** Meminta pengguna untuk memasukkan angka kedua untuk operasi.

```
java
```

Salin kode

```
switch (operator) {  
    case '+':  
        hasil = angka1 + angka2;  
        System.out.println("\nHASIL PENJUMLAHAN : " + hasil);  
        break;
```

**Pernyataan Switch untuk Operator:** Memeriksa operator yang dimasukkan pengguna dan melakukan operasi yang sesuai.

**Operasi Penjumlahan:** Jika operator adalah +, menghitung hasil penjumlahan dan mencetak hasilnya.

```
java
case '+':
    hasil = angka1 + angka2;
    System.out.println("\nHASIL PENJUMLAHAN : " + hasil);
    break;
```

**Operasi Pengurangan:** Jika operator adalah -, menghitung hasil pengurangan dan mencetak hasilnya.

```
java
case '-':
    hasil = angka1 - angka2;
    System.out.println("\nHASIL PENGURANGAN : " + hasil);
    break;
```

**Operasi Perkalian:** Jika operator adalah \*, menghitung hasil perkalian dan mencetak hasilnya.

```
java
case '*':
    hasil = angka1 * angka2;
    System.out.println("\nHASIL PERKALIAN : " + hasil);
    break;
```

**Operasi Pembagian:** Jika operator adalah /, program memeriksa apakah angka kedua angka2 sama dengan 0. Jika ya, mencetak pesan kesalahan. Jika tidak, menghitung pembagian dan mencetak hasilnya.

```
java
```

□ Salin kode

```
default:  
    System.out.println("OPERATOR TIDAK TERSEDIA");
```

**Default Case:** Jika operator yang dimasukkan tidak valid, mencetak pesan kesalahan.

```
java
```

□ Salin kode

```
if (menu == 0) {  
    System.out.println("\nPROGRAM SELESAI, TERIMAKASIH >_<");  
    break;  
}
```

**Keluar dari Program:** Jika pengguna memilih 0, mencetak pesan terima kasih dan keluar dari loop.

```
java
```

□ Salin kode

```
else {  
    System.out.println("PILIHAN TIDAK TERSEDIA");  
}
```

**Pesan Kesalahan:** Jika pilihan menu tidak valid, mencetak pesan kesalahan.

```
java
```

□ Salin kode

```
input.close();
```

**Menutup Scanner:** Menutup objek Scanner untuk membebaskan sumber daya.

## Kesimpulan:

Program kalkulator ini dirancang untuk melakukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian. Terdapat dua mode: Kalkulator A, di mana pengguna memilih operasi setelah memasukkan dua angka, dan Kalkulator B, yang langsung meminta operator setelah angka pertama dan kedua. Program menangani pembagian dengan nol secara aman, mencegah kesalahan dengan memberikan pesan peringatan. Antarmuka yang jelas dan *loop* interaktif memungkinkan pengguna menjalankan kalkulator hingga memilih keluar. Pengelolaan sumber daya juga dilakukan dengan baik melalui penutupan objek Scanner.

## LAPORAN PRAKTIKUM 2

### 2.1 Penjelasan Kode Pertemuan 4

```
java
package Pertemuan4.Tugas;
import java.util.Scanner;
```

**package Pertemuan4.Tugas:** Menunjukkan bahwa file ini termasuk dalam paket Pertemuan4.Tugas. Paket dalam Java digunakan untuk mengelompokkan kelas-kelas yang terkait.

**import java.util.Scanner:** Mengimpor kelas Scanner dari pustaka java.util, yang digunakan untuk membaca input dari pengguna melalui konsol.

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

**public class TiketKereta:** Mendeklarasikan kelas publik TiketKereta, yang merupakan container untuk semua kode dan berfungsi sebagai program utama.

**public static void main(String[] args):** Metode utama yang dieksekusi ketika program dijalankan. Ini adalah titik masuk dari aplikasi Java.

```
java
Scanner scanner = new Scanner(System.in);
int Menu;
String Back;
String Pilihan;
```

**Scanner scanner:** Membuat objek Scanner untuk membaca input dari konsol.

**int Menu:** Variabel integer untuk menyimpan pilihan menu yang diinput oleh pengguna.

**String Back:** Variabel sementara untuk membaca input ketika pengguna ingin kembali ke menu utama setelah menyelesaikan pilihan (walaupun tidak digunakan secara efektif dalam kode ini).

**String Pilihan:** Variabel untuk menyimpan pilihan pengguna apakah ingin melakukan pemesanan tiket lagi atau tidak (Y atau N).

```
java

while (true) {
    // Konten loop
}
```

**while (true):** Sebuah loop yang akan terus berjalan sampai ada perintah break yang menghentikan eksekusinya. Loop ini digunakan untuk menampilkan menu utama berulang-ulang sampai pengguna memutuskan untuk keluar dari program.

```
java

System.out.print("\033[H\033[2J");
```

Ini adalah perintah untuk "membersihkan" konsol, bergantung pada sistem operasi. Secara teknis, ini adalah escape sequence yang berfungsi pada terminal tertentu untuk membersihkan layar, tetapi mungkin tidak berfungsi pada semua platform.

```
java

System.out.println("|| 1. MENAMPILKAN DAFTAR TIKET      ||");
System.out.println("|| 2. MENAMPILKAN JADWAL KEBERANGKATAN ||");
System.out.println("|| 3. PEMESANAN TIKET SECARA ONLINE   ||");
System.out.println("|| 0. KELUAR DARI PROGRAM           ||");
```

Program menampilkan menu utama dengan empat pilihan:

- **1:** Menampilkan daftar tiket kereta.
- **2:** Menampilkan jadwal keberangkatan kereta.
- **3:** Memesan tiket secara online.
- **0:** Keluar dari program.

```
java

System.out.print("\nMASUKKAN PILIHAN : ");
Menu = scanner.nextInt();
```

Program meminta input dari pengguna untuk memilih salah satu dari menu yang ditampilkan, dan input tersebut disimpan dalam variabel Menu.

```
java                                         Salin kode

if (Menu == 1) {
    System.out.print("\033[H\033[2J");
    System.out.println("|| 1. EKONOMI    (RP 100.000) ||");
    System.out.println("|| 2. BISNIS     (RP 200.000) ||");
    System.out.println("|| 3. EKSEKUTIF  (RP 300.000) ||");
    Back = scanner.next(); // Kembali ke menu utama
    continue;
}
```

Jika pengguna memilih **1**, program akan menampilkan daftar kelas tiket kereta yang tersedia beserta harganya (Ekonomi, Bisnis, dan Eksekutif).

Setelah menampilkan daftar, program akan meminta input pengguna (Back) untuk kembali ke menu utama dengan menggunakan perintah continue untuk mengulangi loop.

```
java                                         Salin kode

if (Menu == 2) {
    System.out.print("\033[H\033[2J");
    System.out.println("|| KELAS EKONOMI : G-1010 = 06.00 WIB, TIBA 12.00 WIB ||");
    // Jadwal keberangkatan lain ...
    Back = scanner.next(); // Kembali ke menu utama
    continue;
}
```

Jika pengguna memilih **2**, program akan menampilkan jadwal keberangkatan untuk berbagai kelas tiket (Ekonomi, Bisnis, dan Eksekutif), dengan waktu keberangkatan dan tiba yang jelas.

Setelah menampilkan jadwal, pengguna diminta untuk memasukkan input apa pun untuk kembali ke menu utama.

```
java
```

Salin kode

```
if (Menu == 3) {  
    int Ekonomi = 100000;  
    int Bisnis = 200000;  
    int Eksekutif = 300000;  
    int JenisTiket, JumlahEkonomi = 0, JumlahBisnis = 0, JumlahEksekutif = 0;  
    int Total = 0, Diskon = 0;  
    String Nama;
```

Jika pengguna memilih 3, program memasuki sub-menu pemesanan tiket. Pengguna diminta untuk memasukkan nama dan jenis tiket yang ingin dipesan.

Ada tiga jenis tiket yang dapat dipesan:

- **Ekonomi:** Rp 100.000
- **Bisnis:** Rp 200.000
- **Eksekutif:** Rp 300.000

```
java
```

Salin kode

```
System.out.print("\nMASUKKAN NAMA PEMESAN : ");  
Nama = scanner.next();  
while (true) {  
    System.out.print("\nMASUKKAN JENIS TIKET (1/2/3) : ");  
    JenisTiket = scanner.nextInt();  
    // Meminta jumlah tiket sesuai jenis yang dipilih  
    System.out.print("MASUKKAN JUMLAH : ");  
    Jumlah = scanner.nextInt();  
    // Menambah jumlah tiket sesuai jenis yang dipilih  
    if (JenisTiket == 1) JumlahEkonomi += Jumlah;  
    if (JenisTiket == 2) JumlahBisnis += Jumlah;  
    if (JenisTiket == 3) JumlahEksekutif += Jumlah;  
  
    System.out.print("\nAPAKAH INGIN MEMESAN TIKET LAGI? (Y/N) : ");  
    Pilihan = scanner.next();  
    if (Pilihan.equals("N")) break; // Keluar dari loop pemesanan  
}
```

Pengguna diminta untuk memasukkan nama dan jenis tiket yang ingin dipesan. Pengguna juga bisa memilih lebih dari satu tiket.

Setelah setiap pemesanan, pengguna dapat memilih apakah ingin memesan tiket lagi atau tidak (Y untuk ya, N untuk tidak).

```
java □ Salin kode
Total = (Ekonomi * JumlahEkonomi) + (Bisnis * JumlahBisnis) + (Eksekutif * JumlahEksekutif
if (Total >= 1000000) {
    Diskon = Total * 90 / 100;
    System.out.println("DISKON 10%, BAYAR RP " + Diskon);
} else if (Total >= 500000) {
    Diskon = Total * 95 / 100;
    System.out.println("DISKON 5%, BAYAR RP " + Diskon);
} else if (Total >= 250000) {
    Diskon = Total * 98 / 100;
    System.out.println("DISKON 2%, BAYAR RP " + Diskon);
} else {
    System.out.println("TOTAL RP " + Total);
}
```

Program menghitung total biaya berdasarkan tiket yang dipesan dan menerapkan diskon:

- **10% diskon** jika total lebih dari Rp 1.000.000.
- **5% diskon** jika total lebih dari Rp 500.000.
- **2% diskon** jika total lebih dari Rp 250.000.

```
java □ Salin kode
JumlahEkonomi = 0;
JumlahBisnis = 0;
JumlahEksekutif = 0;
```

Setelah transaksi selesai, jumlah tiket direset agar pemesanan baru dapat dimulai dari nol.

```
java □ Salin kode
if (Menu == 0) {
    System.out.print("TERIMAKASIH ...");
    break;
}
```

Jika pengguna memilih **0**, program akan keluar dari loop dan menampilkan pesan "TERIMAKASIH", lalu program berhenti.

```
java
```

□ Salin kode

```
scanner.close();
```

Setelah selesai digunakan, objek scanner ditutup untuk mencegah kebocoran sumber daya.

### Kesimpulan:

Program **TiketKereta** adalah simulasi sederhana sistem pemesanan tiket kereta yang memungkinkan pengguna melihat daftar tiket, jadwal keberangkatan, dan melakukan pemesanan tiket dengan perhitungan diskon berdasarkan total biaya. Diskon diberikan secara progresif: 10% untuk total  $> \text{Rp } 1.000.000$ , 5% untuk total  $> \text{Rp } 500.000$ , dan 2% untuk total  $> \text{Rp } 250.000$ . Program menggunakan menu interaktif dalam *loop* sehingga pengguna dapat terus berinteraksi hingga memilih keluar. Input pengguna dikelola dengan kelas Scanner, dan setelah transaksi selesai, program mereset jumlah tiket yang dipesan untuk memulai pemesanan baru. Sederhana, efisien, dan ramah pengguna, program ini cocok untuk pemahaman dasar sistem berbasis menu interaktif.

# LAPORAN PRAKTIKUM 3

## 3.1 Penjelasan Kode Pertemuan 5

### Penjelasan Code Bilangan.java:

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

Scanner adalah class di Java yang digunakan untuk mengambil input dari pengguna. Program ini mengimpor Scanner dari library java.util untuk memungkinkan input dari pengguna melalui keyboard.

```
java
public class Bilangan {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
```

Program ini mendeklarasikan sebuah kelas Bilangan dan di dalamnya terdapat metode main, yang merupakan titik awal eksekusi program Java.

Sebuah objek Scanner bernama input dideklarasikan, meskipun dalam kode ini objek tersebut tidak digunakan.

```
java
int A;
Scanner scan = new Scanner(System.in);
System.out.print("MASUKKAN SEBUAH ANGKA : ");
A = scan.nextInt();
```

int A; mendeklarasikan variabel A yang akan menyimpan angka yang diinput oleh pengguna. Sebuah objek Scanner bernama scan diciptakan untuk menerima input dari pengguna.

Program meminta pengguna untuk memasukkan angka dengan perintah System.out.print(), kemudian scan.nextInt() digunakan untuk mengambil input angka dari pengguna dan menyimpannya ke variabel A

```
java  
  
if (A > 0) {  
    System.out.println(A + " ADALAH ANGKA POSITIF.");  
}  
  
if (A < 0) {  
    System.out.println(A + " ADALAH ANGKA NEGATIF.");  
}  
  
if (A == 0) {  
    System.out.println(A + " ADALAH ANGKA NOL.");  
}
```

Program menggunakan struktur if untuk mengecek nilai dari variabel A:

- Jika  $A > 0$ , program akan mencetak bahwa bilangan tersebut positif.
- Jika  $A < 0$ , program akan mencetak bahwa bilangan tersebut negatif.
- Jika  $A == 0$ , program akan mencetak bahwa bilangan tersebut nol.

```
java  
  
scan.close();
```

**scan.close()** menutup objek Scanner setelah input tidak diperlukan lagi untuk membebaskan sumber daya yang digunakan.

```
yaml  
  
MASUKKAN SEBUAH ANGKA : 5  
5 ADALAH ANGKA POSITIF.
```

Jika pengguna memasukkan angka 5

```
diff
```

□ Salin kode

MASUKKAN SEBUAH ANGKA : -3

-3 ADALAH ANGKA NEGATIF.

Jika pengguna memasukkan angka -3

```
yaml
```

□ Salin kode

MASUKKAN SEBUAH ANGKA : 0

0 ADALAH ANGKA NOL.

Jika pengguna memasukkan angka 0

### Penjelasan Code LoopingFor.java:

```
java
```

□ Salin kode

```
int i;  
  
for (i = 1; i <= 5; i++) {  
    System.out.println(i + " ");  
}
```

Variabel **i** dideklarasikan sebagai **counter** untuk loop.

**Loop for** dimulai dari  $i = 1$  dan akan berulang sampai nilai  $i$  mencapai **5**. Setiap kali loop berjalan, nilai  $i$  akan bertambah 1 ( $i++$ ).

Di dalam loop, nilai  $i$  akan dicetak ke terminal menggunakan `System.out.println(i + " ");`. Setiap angka akan dicetak pada baris yang berbeda, karena `System.out.println()` secara otomatis menambahkan **newline** di akhir output.

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

Pada **iterasi pertama**, nilai  $i$  adalah 1, dan program mencetak "1 ".

Pada **iterasi kedua**, nilai i menjadi 2, dan program mencetak "2".

Hal ini berlanjut hingga iterasi terakhir di mana nilai i mencapai 5, kemudian loop berhenti.

### Penjelasan Code InputNol.java:

```
java
while(true) {
```

**while(true)** menciptakan loop tanpa batas yang akan terus berjalan sampai diakhiri secara manual, yaitu ketika pengguna memasukkan angka nol (seperti yang dijelaskan di bawah).

```
java
int B;
Scanner scan = new Scanner(System.in);
System.out.print("MASUKKAN SEBUAH ANGKA : ");
B = scan.nextInt();
```

Variabel **B** digunakan untuk menyimpan input angka dari pengguna.

**Scanner scan** digunakan untuk menerima input dari pengguna.

Program meminta pengguna untuk memasukkan angka dengan perintah **System.out.print()**.

```
java
if (B == 0) {
    break;
} else {
    continue;
}
```

Jika pengguna memasukkan angka 0, program akan mengeksekusi perintah **break**, yang menyebabkan keluar dari loop dan mengakhiri program.

Jika pengguna memasukkan angka selain 0, program akan menjalankan **continue**, yang menyebabkan loop berulang kembali dari awal, sehingga program kembali meminta input.

```
yaml
```

□ Salin kode

```
MASUKKAN SEBUAH ANGKA : 5
```

Karena angka yang dimasukkan bukan nol, program meminta input lagi.

```
yaml
```

□ Salin kode

```
MASUKKAN SEBUAH ANGKA : 0
```

Karena angka yang dimasukkan adalah 0, program keluar dari loop dan berhenti.

### Penjelasan Code JumlahGenap.java:

```
java
```

□ Salin kode

```
int Jumlah = 0;  
int i;
```

Variabel **Jumlah** digunakan untuk menyimpan hasil penjumlahan total.

Variabel **i** adalah variabel yang digunakan sebagai counter dalam loop for.

```
java
```

□ Salin kode

```
for (i=1; i<=100; i++){  
  
    if (i%2==0){  
        Jumlah += i * 2;  
    }  
}
```

**Loop for:** Program menggunakan loop for yang berjalan dari angka 1 hingga 100.

### Kondisi bilangan genap:

Di dalam loop, program memeriksa apakah i adalah bilangan genap dengan  $i \% 2 == 0$ .

- Jika i adalah bilangan genap, program menggandakan nilai i (mengalikan dengan 2) dan menambahkannya ke variabel Jumlah.
- Proses ini berarti setiap bilangan genap (misalnya 2, 4, 6, dst.) digandakan terlebih dahulu sebelum ditambahkan ke total.

```
System.out.println("JUMLAH : " + jumlah);
```

Setelah loop selesai, program menampilkan hasil total penjumlahan dengan teks "JUMLAH : ".

### Kesimpulan:

Keempat program menunjukkan penerapan konsep dasar pemrograman Java. **Bilangan.java** menentukan apakah bilangan input adalah positif, negatif, atau nol menggunakan logika if. **LoopingFor.java** mencetak angka 1 hingga 5 menggunakan loop for. **InputNol.java** menjalankan loop tak terbatas yang berhenti saat pengguna memasukkan angka nol, memanfaatkan break dan continue. **JumlahGenap.java** menghitung total penjumlahan bilangan genap dari 1 hingga 100, dengan menggandakan setiap bilangan genap sebelum menambahkannya ke total. Semua program menggunakan antarmuka sederhana dan pengelolaan sumber daya yang baik.

# LAPORAN PRAKTIKUM 4

## 4.1 Penjelasan Kode Pertemuan 6

### Program Mobil

```
java                                     Salin kode

class Mobil {
    string aktifitas;
    string warna;
    int kecepatan;
```

Mendefinisikan kelas **Mobil** dengan tiga atribut:

- aktifitas: sebuah String yang digunakan untuk menyimpan status aktifitas mobil, misalnya "PARKIR".
- warna: sebuah String untuk menyimpan warna mobil.
- kecepatan: sebuah int untuk menyimpan kecepatan mobil.

```
java                                     Salin kode

public Mobil (String aktifitas, String warna, int kecepatan){
    this.aktifitas = aktifitas;
    this.warna = warna;
    this.kecepatan = kecepatan;
}
```

Mendefinisikan konstruktor dengan parameter aktifitas, warna, dan kecepatan. Konstruktor ini digunakan untuk menginisialisasi objek Mobil dengan nilai-nilai tertentu ketika objek dibuat.

- this.aktifitas = aktifitas; -> Menggunakan kata kunci this untuk membedakan antara atribut kelas dan parameter metode. Atribut aktifitas dari objek diset dengan nilai dari parameter aktifitas.
- this.warna = warna; -> Atribut warna diset dengan nilai dari parameter warna.
- this.kecepatan = kecepatan; -> Atribut kecepatan diset dengan nilai dari parameter kecepatan.

```
java
```

Salin kode

```
public Mobil() {  
}
```

Mendefinisikan konstruktor tanpa parameter. Ini adalah konstruktor default yang memungkinkan pembuatan objek Mobil tanpa memberikan nilai awal untuk atribut.

```
java
```

Salin kode

```
void cekKecepatan() {  
    if (kecepatan == 0)  
        aktifitas = "PARKIR";  
}
```

Mendefinisikan metode cekKecepatan yang memeriksa nilai kecepatan:

- Jika kecepatan bernilai 0, maka aktifitas diatur menjadi "PARKIR".
- Metode ini digunakan untuk memastikan bahwa jika mobil tidak bergerak (kecepatan = 0), maka status aktifitas mobil adalah "PARKIR".

```
java
```

Salin kode

```
void cetakAtribut() {  
    System.out.println("AKTIFITAS = " + aktifitas);  
    System.out.println("WARNA = " + warna);  
    System.out.println("KECEPATAN = " + kecepatan);  
}
```

Mendefinisikan metode cetakAtribut untuk mencetak nilai dari aktifitas, warna, dan kecepatan ke layar.

- System.out.println("AKTIFITAS = " + aktifitas); -> Mencetak nilai aktifitas mobil.
- System.out.println("WARNA = " + warna); -> Mencetak nilai warna mobil.
- System.out.println("KECEPATAN = " + kecepatan); -> Mencetak nilai kecepatan mobil.

```
java
```

□ Salin kode

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

Mendefinisikan kelas **ClassMobil** yang memiliki metode utama main. Metode main adalah titik masuk program, yang akan dieksekusi ketika program dijalankan.

```
java
```

□ Salin kode

```
Mobil mobilku = new Mobil();
```

Membuat objek mobilku dari kelas Mobil menggunakan konstruktor tanpa parameter. Ini berarti mobilku dibuat dengan atribut aktifitas, warna, dan kecepatan yang belum diinisialisasi (atau null dan 0 sebagai nilai default).

```
java
```

□ Salin kode

```
mobilku.kecepatan = 0;  
mobilku.warna = "MERAH";
```

Mengatur nilai atribut kecepatan dan warna untuk objek mobilku:

- kecepatan diatur ke 0.
- warna diatur ke "MERAH".

```
java
```

□ Salin kode

```
mobilku.cekKecepatan();
```

Memanggil metode cekKecepatan() pada objek mobilku. Karena kecepatan mobil diatur ke 0, maka metode ini akan mengubah aktifitas mobil menjadi "PARKIR".

```
java
```

□ Salin kode

```
        mobilku.cetakAtribut();  
    }  
}
```

Memanggil metode cetakAtribut() pada objek mobilku, yang mencetak nilai dari aktifitas, warna, dan kecepatan ke layar.

```
makefile
```

```
AKTIFITAS = PARKIR
WARNA = MERAH
KECEPATAN = 0
```

Salin kode

Karena kecepatan diatur ke 0, metode cekKecepatan() mengubah aktifitas menjadi "PARKIR". Metode cetakAtribut() kemudian mencetak nilai-nilai dari atribut mobilku.

## Program BintangFilm

```
java
```

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

Salin kode

Mendefinisikan kelas **ClassBintangFilm** yang berisi metode utama main. Metode main adalah titik masuk program, yang akan dieksekusi saat program dijalankan.

```
java
```

```
BintangFilm siA = new BintangFilm("BUDI", true);
BintangFilm siB = new BintangFilm("HANI", false);
```

Salin kode

Membuat dua objek BintangFilm:

- siA dengan nama "BUDI" dan pria bernilai true, artinya siA adalah seorang pria.
- siB dengan nama "HANI" dan pria bernilai false, artinya siB adalah seorang wanita.

```
java
```

```
System.out.println("siA => " + siA.perolehNama() + ", " + siA.perolehJenisKelamin()
System.out.println("siB => " + siB.perolehNama() + ", " + siB.perolehJenisKelamin()
}
}
```

Salin kode

Menampilkan atribut dari objek siA dan siB menggunakan metode perolehNama dan perolehJenisKelamin.

- siA.perolehNama() mengembalikan nama "BUDI", dan siA.perolehJenisKelamin() mengembalikan "PRIA".
- siB.perolehNama() mengembalikan nama "HANI", dan siB.perolehJenisKelamin() mengembalikan "WANITA".

```
makefile
Salin kode

siA => BUDI, PRIA
siB => HANI, WANITA
```

Pada **output** ini:

- siA memiliki nama "BUDI" dan jenis kelamin "PRIA".
- siB memiliki nama "HANI" dan jenis kelamin "WANITA".

## Program Calculator

```
java
Salin kode

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

Kelas **ProgramCalculator** berisi metode utama main, yang merupakan titik masuk program.

```
java
Salin kode

Calculator calc = new Calculator();
```

Membuat objek Calculator bernama **calc**.

```
java
Salin kode

calc.isiOperan1(10.0);
calc.isiOperan2(4.0);
```

Mengatur operan1 dengan nilai 10.0 dan operan2 dengan nilai 4.0 menggunakan metode isiOperan1 dan isiOperan2.

```
java                                     Salin kode

    System.out.println("Operan1 + Operan2 = " + calc.tambah());
    System.out.println("Operan1 - Operan2 = " + calc.kurang());
    System.out.println("Operan1 * Operan2 = " + calc.kali());
    System.out.println("Operan1 / Operan2 = " + calc.bagi());
    System.out.println("Operan1 ^ Operan2 = " + calc.pangkat());
}

}
```

Mencetak hasil operasi matematika antara operan1 dan operan2 menggunakan metode yang telah dibuat (tambah, kurang, kali, bagi, dan pangkat).

```
                                     Salin kode

Operan1 + Operan2 = 14.0
Operan1 - Operan2 = 6.0
Operan1 * Operan2 = 40.0
Operan1 / Operan2 = 2.5
Operan1 ^ Operan2 = 10000.0
```

**Output** ini menunjukkan hasil operasi matematika yang dilakukan antara operan1 dan operan2.

## Program BangunDatar

```
java                                     Salin kode

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

Kelas ProgramBangunDatar berisi metode main, yang merupakan titik masuk program. Program ini akan meminta pengguna untuk memilih jenis bangun datar yang luasnya akan dihitung.

```
java                                     Salin kode

Scanner input = new Scanner(system.in);
```

Membuat objek Scanner untuk menerima input dari pengguna.

```
java                                     Salin kode

    System.out.println("PILIH BANGUN DATAR UNTUK MENGHITUNG LUAS");
    System.out.println("\n1. LUAS PERSEGI");
    System.out.println("2. LUAS PERSEGI PANJANG");
    System.out.println("3. LUAS LINGKARAN");
    System.out.println("4. LUAS SEGITIGA");
    System.out.print("\nMASUKKAN PILIHAN : ");
    int pilihan = input.nextInt();
```

Menampilkan daftar pilihan bangun datar dan meminta input pengguna untuk memilih salah satu.

```
java                                     Salin kode

    BangunDatar(double sisi) {
        this.sisi = sisi;
    }

    double luasPersegi() {
        return sisi * sisi;
    }
```

Konstruktor ini digunakan untuk inisialisasi persegi dengan mengisi atribut sisi dengan nilai yang diberikan sebagai parameter.

```
java
```

 Salin kode

```
BangunDatar(double panjang, double lebar) {  
    this.panjang = panjang;  
    this.lebar = lebar;  
}
```

```
java
```

 Salin kode

```
double luasPersegiPanjang() {  
    return panjang * lebar;  
}
```

Konstruktor ini digunakan untuk inisialisasi persegi panjang dengan mengisi atribut panjang dan lebar dengan nilai yang diberikan sebagai parameter.

```
java
```

 Salin kode

```
BangunDatar(double jariJari, boolean isLingkaran) {  
    this.jariJari = jariJari;  
}
```

```
java
```

 Salin kode

```
double luasLingkaran() {  
    return Math.PI * jariJari * jariJari;  
}
```

Konstruktor ini digunakan untuk inisialisasi lingkaran dengan mengisi atribut jariJari. Parameter tambahan isLingkaran hanya berfungsi sebagai penanda dan tidak digunakan dalam logika program.

```
java
```

□ Salin kode

```
BangunDatar(double alas, double tinggi, boolean isSegitiga) {  
    this.alas = alas;  
    this.tinggi = tinggi;  
}
```

```
java
```

□ Salin kode

```
double luasSegitiga() {  
    return 0.5 * alas * tinggi;  
}
```

Konstruktor ini digunakan untuk inisialisasi segitiga dengan mengisi atribut alas dan tinggi. Parameter tambahan isSegitiga hanya berfungsi sebagai penanda dan tidak digunakan dalam logika program.

```
java
```

□ Salin kode

```
switch (pilihan) {
```

Menggunakan struktur switch untuk memproses pilihan pengguna dan menghitung luas bangun datar sesuai pilihan.

#### Pilihan:

- **Pilihan 1 (Persegi):**
  - Meminta pengguna untuk memasukkan panjang sisi persegi, lalu menghitung dan menampilkan luas persegi.
- **Pilihan 2 (Persegi Panjang):**
  - Meminta pengguna memasukkan panjang dan lebar persegi panjang, lalu menghitung dan menampilkan luas persegi panjang.
- **Pilihan 3 (Lingkaran):**
  - Meminta pengguna memasukkan jari-jari lingkaran, lalu menghitung dan menampilkan luas lingkaran.
- **Pilihan 4 (Segitiga):**
  - Meminta pengguna memasukkan panjang alas dan tinggi segitiga, lalu menghitung dan menampilkan luas segitiga.

- **Pilihan Tidak Valid:**

- Menampilkan pesan jika pilihan yang dimasukkan tidak valid.

```
yaml
LUAS PERSEGI : 25.0
```

A screenshot of a Java code editor window. The title bar says "yaml". The main area contains the following Java code:

```
LUAS PERSEGI : 25.0
```

The code consists of a single line of output text.

Salin kode

Jika pengguna memilih opsi 1 dan memasukkan sisi persegi 5

### Kesimpulan:

Keempat program menunjukkan implementasi konsep OOP dan operasi dasar Java. Mobil menggunakan atribut, konstruktor, dan metode untuk menentukan status, warna, dan kecepatan mobil serta mencetak atributnya. BintangFilm mendemonstrasikan penggunaan objek untuk menyimpan dan menampilkan data nama dan jenis kelamin aktor. Calculator menunjukkan operasi aritmatika dasar melalui metode untuk penjumlahan, pengurangan, perkalian, pembagian, dan pangkat. BangunDatar menghitung luas berbagai bangun datar (persegi, persegi panjang, lingkaran, dan segitiga) berdasarkan pilihan pengguna, dengan validasi input melalui switch-case. Semua program mencerminkan pemanfaatan konsep kelas, objek, metode, dan struktur kontrol dalam Java.

# LAPORAN PRAKTIKUM 5

## 5.1 Penjelasan Kode Pertemuan 7

### Program MhsTester

```
java                                     Salin kode

package Pertemuan7;

class Mahasiswa {
    // Atribut dan metode akan dijelaskan di bawah ini
}
```

**Package Declaration:** package Pertemuan7; menunjukkan bahwa kelas Mahasiswa berada dalam package Pertemuan7.

**Class Declaration:** class Mahasiswa mendefinisikan sebuah kelas bernama Mahasiswa. Tidak ada modifier akses yang diberikan, sehingga secara default kelas ini memiliki akses *package-private*, artinya hanya bisa diakses oleh kelas lain dalam package yang sama

```
java                                     Salin kode

public String nama;                      // Public: bisa diakses dari mana saja
protected int usia;                      // Protected: bisa diakses dari dalam package dan subclass
private String jurusan;                  // Private: hanya bisa diakses dalam kelas ini
```

#### public String nama:

- **Modifier Akses:** public
- **Deskripsi:** Atribut nama dapat diakses dari mana saja, baik dari dalam kelas Mahasiswa, kelas lain dalam package Pertemuan7, maupun dari luar package.

#### protected int usia:

- **Modifier Akses:** protected
- **Deskripsi:** Atribut usia dapat diakses dari dalam package Pertemuan7 dan oleh subclass yang mungkin meng-extend kelas Mahasiswa di package lain.

#### private String jurusan:

- **Modifier Akses:** private

- **Deskripsi:** Atribut jurusan hanya dapat diakses dari dalam kelas Mahasiswa itu sendiri. Tidak bisa diakses langsung dari luar kelas, termasuk kelas dalam package yang sama.

```
java                                         Salin kode

// Constructor
public Mahasiswa(String nama, int usia, String jurusan) {
    this.nama = nama;
    this.usia = usia;
    this.jurusan = jurusan;
}
```

**Fungsi:** Konstruktor ini digunakan untuk menginisialisasi objek Mahasiswa dengan nilai-nilai awal untuk atribut nama, usia, dan jurusan.

#### Parameter:

- String nama: Nama mahasiswa.
- int usia: Usia mahasiswa.
- String jurusan: Jurusan mahasiswa.

**Penggunaan this:** Kata kunci this digunakan untuk merujuk pada atribut kelas saat terjadi konflik nama antara parameter dan atribut.

```
java                                         Salin kode

// Getter untuk atribut private jurusan
public String getJurusan() {
    return jurusan;
}

// Setter untuk atribut private jurusan
public void setJurusan(String jurusanBaru) {
    this.jurusan = jurusanBaru;
}
```

#### Getter (getJurusan):

- **Fungsi:** Mengembalikan nilai atribut jurusan.
- **Aksesibilitas:** public, sehingga dapat diakses dari luar kelas untuk mendapatkan nilai jurusan.

### Setter (setJurusan):

- **Fungsi:** Mengubah nilai atribut jurusan.
- **Parameter:** String jurusanBaru adalah nilai baru yang akan diassign ke atribut jurusan.
- **Aksesibilitas:** public, memungkinkan perubahan nilai jurusan dari luar kelas.

```
java                                     Salin kode

// Metode untuk menampilkan informasi
public void tampilanInfo() {
    System.out.println("NAMA : " + nama);
    System.out.println("USIA : " + usia);
    System.out.println("JURUSAN : " + jurusan);
}
```

**Fungsi:** Menampilkan semua informasi mahasiswa (nama, usia, dan jurusan) ke konsol.

**Aksesibilitas:** public, sehingga dapat dipanggil dari luar kelas untuk menampilkan informasi mahasiswa.

```
java                                     Salin kode

package Pertemuan7;

public class MhsTester {
    public static void main(String[] args) {
        // Isi metode main akan dijelaskan di bawah
    }
}
```

**Package Declaration:** package Pertemuan7; menunjukkan bahwa kelas MhsTester juga berada dalam package Pertemuan7.

**Class Declaration:** public class MhsTester mendefinisikan kelas MhsTester yang bersifat public, sehingga dapat diakses dari luar package jika diperlukan.

**Main Method:** public static void main(String[] args) adalah titik masuk program yang akan dieksekusi saat program dijalankan.

```
java
```

□ Salin kode

```
Mahasiswa mahasiswa1 = new Mahasiswa("ANDI", 21, "TEKNIK INFORMATIKA");
```

**Fungsi:** Membuat instansi (objek) Mahasiswa dengan nama "ANDI", usia 21, dan jurusan "TEKNIK INFORMATIKA".

**Proses:**

- Memanggil konstruktor Mahasiswa dengan parameter yang diberikan.
- Atribut nama, usia, dan jurusan diinisialisasi sesuai dengan parameter.

```
java
```

□ Salin kode

```
System.out.println("NAMA MAHASISWA : " + mahasiswa1.nama); // Output: ANDI
```

**Fungsi:** Mencetak nilai atribut nama dari objek mahasiswa1.

**Aksesibilitas:** Atribut nama bersifat public, sehingga dapat diakses langsung dari luar kelas.

```
java
```

□ Salin kode

```
System.out.println("USIA MAHASISWA : " + mahasiswa1.usia); // Output: 21
```

**Fungsi:** Mencetak nilai atribut usia dari objek mahasiswa1.

**Aksesibilitas:** Atribut usia bersifat protected dan dapat diakses karena kelas MhsTester berada dalam package yang sama (Pertemuan7).

```
java
```

□ Salin kode

```
System.out.println("JURUSAN MAHASISWA : " + mahasiswa1.getJurusan()); // Output: TEKNIK IN
```

**Fungsi:** Mencetak nilai atribut jurusan menggunakan metode getJurusan().

**Aksesibilitas:** Atribut jurusan bersifat private, sehingga tidak dapat diakses langsung. Metode getJurusan() (yang bersifat public) digunakan untuk mendapatkan nilai tersebut.

```
java
```

□ Salin kode

```
mahasiswa1.setJurusan("SISTEM INFORMASI");
System.out.println("JURUSAN MAHASISWA SETELAH DIUBAH : " + mahasiswa1.getJurusan()); // Output
```

### Fungsi:

- Mengubah nilai atribut jurusan dari "TEKNIK INFORMATIKA" menjadi "SISTEM INFORMASI" menggunakan metode setJurusan().
- Mencetak nilai jurusan yang baru untuk memastikan perubahan berhasil.

**Aksesibilitas:** Metode setJurusan() bersifat public, memungkinkan modifikasi atribut jurusan dari luar kelas.

```
java
```

□ Salin kode

```
mahasiswa1.tampilkanInfo();
```

**Fungsi:** Memanggil metode tampilkanInfo() yang mencetak semua informasi mahasiswa (nama, usia, dan jurusan) ke konsol.

### Program Mobil

```
java
```

□ Salin kode

```
package Pertemuan7;

public class Mobil {
    // Atribut dan metode akan dijelaskan di bawah ini
}
```

**Package Declaration:** package Pertemuan7; menunjukkan bahwa kelas Mobil berada dalam package Pertemuan7.

**Class Declaration:** public class Mobil mendefinisikan kelas Mobil sebagai kelas publik yang dapat diakses dari luar package.

```
java
```

□ Salin kode

```
public String merk;
protected int tahunProduksi;
private double harga;
```

**public String merk:** Atribut merk dapat diakses dari mana saja karena bersifat public.

**protected int tahunProduksi:** Atribut tahunProduksi dapat diakses dalam package yang sama dan oleh subclass di package lain karena bersifat protected.

**private double harga:** Atribut harga bersifat private, hanya bisa diakses dari dalam kelas Mobil.

```
java                                     Salin kode

public Mobil(String merk, int tahunProduksi, double harga) {
    this.merk = merk;
    this.tahunProduksi = tahunProduksi;
    this.harga = harga;
}
```

**Fungsi:** Konstruktor ini menginisialisasi objek Mobil dengan merk, tahunProduksi, dan harga.

**Parameter:**

- String merk: Merek mobil.
- int tahunProduksi: Tahun produksi mobil.
- double harga: Harga mobil.

```
java                                     Salin kode

public double getHarga() {
    return harga;
}
```

**Fungsi:** Mengembalikan nilai harga.

**Aksesibilitas:** Bersifat public, sehingga dapat diakses dari luar kelas Mobil.

```
java                                     Salin kode

public void setHarga(double hargaBaru) {
    if (hargaBaru > 0) {
        this.harga = hargaBaru;
    } else {
        System.out.println("HARGA HARUS LEBIH BESAR DARI 0");
    }
}
```

- **Fungsi:** Mengubah nilai harga jika hargaBaru lebih besar dari 0. Jika tidak, pesan kesalahan ditampilkan.
- **Parameter:**
  - o double hargaBaru: Harga baru yang ingin diset.
- **Aksesibilitas:** Bersifat public.

```
java                                     Salin kode

public void tampilkanInfoMobil() {
    System.out.println("MERK : " + merk);
    System.out.println("TAHUN PRODUKSI : " + tahunProduksi);
    System.out.println("HARGA : " + harga);
}
```

**Fungsi:** Menampilkan semua informasi tentang mobil (merk, tahunProduksi, dan harga) ke konsol.

**Aksesibilitas:** Bersifat public.

```
java                                     Salin kode

Mobil mobil1 = new Mobil("TOYOTA", 2022, 30000000);
```

Membuat instansi Mobil dengan merk "TOYOTA", tahunProduksi 2022, dan harga 30,000,000.

```
java
```

□ Salin kode

```
System.out.println("MERK MOBIL : " + mobil1.merk);
System.out.println("TAHUN PRODUKSI MOBIL : " + mobil1.tahunProduksi);
System.out.println("HARGA MOBIL : " + mobil1.getHarga());
```

Mengakses merk dan tahunProduksi secara langsung karena bersifat public dan protected, dan mengakses harga melalui metode getHarga() karena harga bersifat private.

```
java
```

□ Salin kode

```
mobil1.setHarga(35000000);
System.out.println("HARGA MOBIL SETELAH DIUBAH : " + mobil1.getHarga());
```

Mengubah harga menjadi 35,000,000 menggunakan setHarga().

```
java
```

□ Salin kode

```
mobil1.tampilkanInfoMobil();
```

Memanggil metode tampilkanInfoMobil() untuk menampilkan semua informasi mobil.

## Program Nilai 1

```
java
```

□ Salin kode

```
static class Nilai {
    // Atribut
    private double quis;
    private double uts;
    private double uas;
}
```

Kelas Nilai adalah inner class yang bersifat static, dengan atribut quis, uts, dan uas, masing-masing bertipe double dan bersifat private. Ini berarti atribut-atribut ini hanya bisa diakses melalui metode getter dan setter yang disediakan.

```
public void setQuis(double quis) {
    this.quis = quis;
}

public void setUTS(double uts) {
    this.uts = uts;
}

public void setUAS(double uas) {
    this.uas = uas;
}

// Getter untuk atribut
public double getQuis() {
    return quis;
}

public double getUTS() {
    return uts;
}

public double getUAS() {
    return uas;
}
```

**Setter:** Mengatur nilai untuk atribut quis, uts, dan uas.

**Getter:** Mengambil nilai dari masing-masing atribut.

```
java
```

```
public double getNA() {
    return (0.2 * quis) + (0.3 * uts) + (0.5 * uas);
}
```

**Fungsi:** Menghitung nilai akhir (NA) berdasarkan bobot:

- quis: 20%
- uts: 30%
- uas: 50%

**Return Value:** Mengembalikan hasil perhitungan NA sebagai double.

```
java
```

Salin kode

```
public char getIndex() {  
    double na = getNA();  
    if (na >= 80 && na <= 100) {  
        return 'A';  
    } else if (na >= 68 && na < 80) {  
        return 'B';  
    } else if (na >= 56 && na < 68) {  
        return 'C';  
    } else if (na >= 45 && na < 56) {  
        return 'D';  
    } else {  
        return 'E';  
    }  
}
```

**Fungsi:** Menentukan indeks nilai berdasarkan nilai akhir (NA).

- A: 80 - 100
- B: 68 - 79
- C: 56 - 67
- D: 45 - 55
- E: Di bawah 45

**Return Value:** Mengembalikan karakter yang merepresentasikan indeks nilai.

```
java                                         ⌂ Salin kode

public String getKeterangan() {
    switch (getIndex()) {
        case 'A':
            return "SANGAT BAIK";
        case 'B':
            return "BAIK";
        case 'C':
            return "CUKUP";
        case 'D':
            return "KURANG";
        case 'E':
            return "SANGAT KURANG";
        default:
            return "TIDAK ADA";
    }
}
```

**Fungsi:** Memberikan keterangan berdasarkan indeks nilai:

- A -> "SANGAT BAIK"
- B -> "BAIK"
- C -> "CUKUP"
- D -> "KURANG"
- E -> "SANGAT KURANG"

**Return Value:** Mengembalikan string keterangan yang sesuai.

```
java                                     Salin kode

public static void main(String[] args) {

    System.out.print("\033[H\033[2J"); // Membersihkan layar terminal

    Nilai n = new Nilai();           // Membuat objek Nilai
    n.setQuis(60);                  // Mengatur nilai quis
    n.setUTS(80);                   // Mengatur nilai UTS
    n.setUAS(75);                   // Mengatur nilai UAS

    System.out.println("QUIS      : " + n.getQuis());
    System.out.println("UTS       : " + n.getUTS());
    System.out.println("UAS      : " + n.getUAS());
    System.out.println("NILAI AKHIR : " + n.getNA());
    System.out.println("INDEX     : " + n.getIndex());
    System.out.println("KETERANGAN : " + n.getKeterangan());
}

}
```

**Fungsi:** Metode main adalah titik masuk untuk menjalankan program. Kode ini:

1. Menghapus layar terminal (hanya berfungsi di terminal tertentu).
2. Membuat objek Nilai.
3. Menetapkan nilai quis, uts, dan uas menggunakan metode setter.
4. Menampilkan nilai quis, uts, uas, nilai akhir, indeks, dan keterangan ke layar.

## Program Nilai 2

```
java                                     Salin kode

private double quiz;
private double UTS;
private double UAS;
```

Atribut quiz, UTS, dan UAS disimpan sebagai variabel private bertipe double, yang berarti nilai-nilai ini hanya bisa diakses melalui metode getter dan setter di dalam kelas ini.

```

public void setQuiz(double quiz) {
    if (quiz >= 0 && quiz <= 100) {
        this.quiz = quiz;
    } else {
        System.out.println("NILAI QUIZ HARUS ANTARA 0 DAN 100");
    }
}

public void setUTS(double UTS) {
    if (UTS >= 0 && UTS <= 100) {
        this.UTS = UTS;
    } else {
        System.out.println("NILAI UTS HARUS ANTARA 0 DAN 100");
    }
}

public void setUAS(double UAS) {
    if (UAS >= 0 && UAS <= 100) {
        this.UAS = UAS;
    } else {
        System.out.println("NILAI UAS HARUS ANTARA 0 DAN 100");
    }
}

```

 Salin kode



**Fungsi:** Metode ini digunakan untuk menetapkan nilai quiz, UTS, dan UAS dengan validasi.

**Validasi:** Jika nilai yang diberikan berada di luar rentang 0 - 100, maka pesan error akan ditampilkan, dan nilai tidak akan disimpan.

```

java

public double getQuiz() {
    return quiz;
}

public double getUTS() {
    return UTS;
}

public double getUAS() {
    return UAS;
}

```

 Salin kode

**Fungsi:** Metode ini digunakan untuk mengakses nilai quiz, UTS, dan UAS dari luar kelas.

**Return Value:** Mengembalikan nilai dari masing-masing atribut.

```
java                                     Salin kode

public double getNA() {
    return 0.20 * quiz + 0.30 * UTS + 0.50 * UAS;
}
```

- **Fungsi:** Menghitung nilai akhir berdasarkan bobot nilai:
  - QUIZ: 20%
  - UTS: 30%
  - UAS: 50%
- **Return Value:** Mengembalikan hasil perhitungan nilai akhir (NA) sebagai double.

```
java                                     Salin kode

public class NilaiTester2 {
    public static void main(String[] args) {

        System.out.print("\033[H\033[2J"); // Membersihkan layar terminal

        Nilai n = new Nilai(); // Membuat objek Nilai

        n.setQuiz(90);          // Menetapkan nilai quiz menjadi 90
        n.setUTS(70);           // Menetapkan nilai UTS menjadi 70
        n.setUAS(150);          // Menetapkan nilai UAS menjadi 150 (akan ditolak karena me

        System.out.println("NILAI QUIZ : " + n.getQuiz());
        System.out.println("NILAI UTS  : " + n.getUTS());
        System.out.println("NILAI UAS  : " + n.getUAS());
        System.out.println("NILAI AKHIR : " + n.getNA()); // Menampilkan Nilai Akhir (NA)
    }
}
```

**Membuat Objek Nilai:** Nilai n = new Nilai(); membuat objek Nilai bernama n.

**Set Nilai dengan Validasi:**

- n.setQuiz(90); menetapkan nilai kuis menjadi 90 (dalam rentang yang valid).
- n.setUTS(70); menetapkan nilai UTS menjadi 70 (dalam rentang yang valid).
- n.setUAS(150); mencoba menetapkan nilai UAS menjadi 150. Karena nilai ini melebihi 100, setter akan menampilkan pesan "NILAI UAS HARUS ANTARA 0 DAN 100" dan nilai tidak akan disimpan.

**Menampilkan Nilai dan Nilai Akhir:** Menampilkan nilai quiz, UTS, UAS, dan nilai akhir yang dihitung berdasarkan bobot.

## Program Siswa

```
java □ Salin kode
class Siswa {
    private String nama;           // Private: hanya bisa diakses dalam kelas ini
    private int nilaiUjian;        // Private: hanya bisa diakses dalam kelas ini
```

nama dan nilaiUjian adalah atribut private dari kelas Siswa, yang berarti mereka tidak dapat diakses langsung dari luar kelas ini. nama menyimpan nama siswa, sedangkan nilaiUjian menyimpan nilai ujian siswa.

```
java □ Salin kode
public Siswa(String nama, int nilaiUjian) {
    this.nama = nama;
    setNilaiUjian(nilaiUjian); // Gunakan setter untuk validasi nilai awal
}
```

Constructor ini digunakan untuk membuat objek Siswa dengan nama dan nilaiUjian tertentu.

this.nama = nama; menginisialisasi nama siswa.

setNilaiUjian(nilaiUjian); menggunakan setter untuk mengatur nilai ujian dan memvalidasi nilai awal yang diberikan.

```
java □ Salin kode

    public String getNama() {
        return nama;
    }

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

    public int getNilaiUjian() {
        return nilaiUjian;
    }

    public void setNilaiUjian(int nilaiUjian) {
        if (nilaiUjian >= 0 && nilaiUjian <= 100) {
            this.nilaiUjian = nilaiUjian;
        } else {
            System.out.println("\nNILAI HARUS ANTARA 0 DAN 100");
        }
    }
}
```

getNama() mengembalikan nilai nama.

setNama(String nama) menetapkan nilai baru

untuk nama. getNilaiUjian() mengembalikan nilai  
nilaiUjian.

setNilaiUjian(int nilaiUjian) mengatur nilai nilaiUjian dengan validasi. Jika nilai berada  
di luar rentang 0-100, pesan kesalahan akan ditampilkan dan nilai tidak diubah.

```
java □ Salin kode

    public void tampilanInfo() {
        System.out.println("NAMA SISWA : " + nama);
        System.out.println("NILAI UJIAN : " + nilaiUjian);
    }
}
```

Metode ini menampilkan nama dan nilaiUjian dari siswa.

```
java
```

□ Salin kode

```
public class SiswaTester {  
    public static void main(String[] args) {  
  
        System.out.print("\033[H\033[2J");
```

SiswaTester adalah kelas utama untuk menjalankan dan menguji fungsi kelas Siswa.

System.out.print("\033[H\033[2J"); membersihkan layar konsol (ini hanya berfungsi di beberapa terminal).

```
java
```

□ Salin kode

```
Siswa siswa1 = new Siswa("ANDI", 85);  
siswa1.tampilkanInfo();
```

siswa1 adalah objek dari kelas Siswa yang diinisialisasi dengan nama "ANDI" dan nilaiUjian 85. tampilkanInfo() menampilkan informasi awal siswa1.

```
java
```

□ Salin kode

```
siswa1.setNama("BUDI");  
siswa1.setNilaiUjian(95);  
  
System.out.println("\nSETELAH DIUBAH : ");  
siswa1.tampilkanInfo();
```

setNama("BUDI") mengubah nama menjadi "BUDI".

setNilaiUjian(95) mengubah nilaiUjian menjadi 95.

Setelah atribut diubah, tampilkanInfo() dipanggil untuk menampilkan informasi terbaru.

## Program Waktu

```
java  
  
public class Waktu {  
    private int menitWaktu;
```

menitWaktu adalah atribut private yang menyimpan jumlah waktu dalam satuan menit.

```
java  
  
public Waktu(int menitWaktu) {  
    this.menitWaktu = menitWaktu;  
}
```

- Konstruktor ini menerima parameter menitWaktu dan menetapkannya ke atribut kelas menitWaktu.
- Misalnya, jika objek dibuat dengan new Waktu(125), menitWaktu akan diinisialisasi dengan nilai 125 menit.

```
java  
  
public int getJam() {  
    return menitWaktu / 60;  
}
```

getJam() mengembalikan jumlah jam dengan membagi menitWaktu dengan 60.

Misalnya, jika menitWaktu adalah 125, hasilnya adalah 2 (jam).

```
java  
  
public int getMenit() {  
    return menitWaktu % 60;  
}
```

getMenit() mengembalikan sisa menit setelah dihitung jam.

Misalnya, jika menitWaktu adalah 125, hasilnya adalah 5 (menit).

```
java
```

Salin kode

```
public void setJam(int j) {  
    menitWaktu = (j * 60) + (menitWaktu % 60);  
}
```

setJam(int j) mengatur bagian jam dengan mengalikan j dengan 60, kemudian menambahkan sisa menit dari menitWaktu.

Jika kita memanggil setJam(3) ketika menitWaktu adalah 125, hasilnya adalah  $(3 * 60) + 5 = 185$ .

```
java
```

Salin kode

```
public void setMenit(int m) {  
    menitWaktu = ((menitWaktu / 60) * 60) + m;  
}
```

setMenit(int m) mengatur bagian menit dengan mengganti bagian menit dari menitWaktu tanpa mempengaruhi bagian jam.

Jika kita memanggil setMenit(30) ketika menitWaktu adalah 185, hasilnya adalah  $180 + 30 = 210$ .

```
java
```

Salin kode

```
public int getTotalMenit() {  
    return menitWaktu;  
}
```

getTotalMenit() mengembalikan nilai menitWaktu secara keseluruhan.

```
java
```

Salin kode

```
public static void main(String[] args) {  
  
    System.out.print("\u001B[H\u001B[2J");  
  
    Waktu waktu = new Waktu(125); // 125 menit dari jam 00:00
```

Kode ini membersihkan layar konsol dan membuat objek Waktu dengan nilai 125 menit, atau 2 jam dan 5 menit dari jam 00:00.

```
java
```

□ Salin kode

```
System.out.println("JAM : " + waktu.getJam()); // Output: 2  
System.out.println("MENIT : " + waktu.getMenit()); // Output: 5
```

Menampilkan jam (2) dan menit (5) dari menitWaktu (125 menit).

```
java
```

□ Salin kode

```
waktu.setJam(3); // Mengatur jam menjadi 3, menit tetap 5  
System.out.println("SETELAH SET JAM(3) - TOTAL MENIT : " + waktu.getTotalMenit());
```

setJam(3) mengubah menitWaktu menjadi 185 (3 jam 5 menit).

getTotalMenit() kemudian menampilkan nilai 185.

```
java
```

□ Salin kode

```
waktu.setMenit(30); // Mengatur menit menjadi 30, jam tetap 3  
System.out.println("SETELAH SET MENIT(30) - TOTAL MENIT : " + waktu.getTotalMenit()  
}
```

setMenit(30) mengubah menitWaktu menjadi 210 (3 jam 30 menit).

getTotalMenit() kemudian menampilkan nilai 210.

## Kesimpulan:

Masing-masing program dirancang untuk mengimplementasikan prinsip-prinsip dasar pemrograman berorientasi objek (OOP) seperti enkapsulasi, pewarisan, dan pengelolaan akses data menggunakan modifier akses (public, protected, dan private). Program-program tersebut memperlihatkan penggunaan atribut, konstruktor, getter, setter, dan metode untuk mengatur serta mengakses data secara aman dan terstruktur. Misalnya, pada program Mahasiswa dan Mobil, atribut private hanya bisa diakses melalui metode getter dan setter, memastikan validasi data berjalan dengan baik. Program Nilai dan Siswa menunjukkan bagaimana validasi dapat diterapkan dalam setter untuk menjaga konsistensi data. Program Waktu memperlihatkan penggunaan metode getter dan setter untuk manipulasi atribut secara logis, seperti konversi menit ke jam dan sebaliknya. Secara keseluruhan, program-program ini menekankan pentingnya pengelolaan data yang terstruktur, modularitas kode, dan penerapan prinsip OOP untuk menciptakan kode yang mudah dipelihara, fleksibel, dan aman.

# LAPORAN PRAKTIKUM 6

## 6.1 Penjelasan Kode Pertemuan 8

### Bagian 1: Bangun Datar

=> Parent Class

```
public class BangunDatar{  
  
    float luas(){  
        System.out.print("\033[H\033[2J");  
        System.out.println("MENGHITUNG LUAS BANGUN DATAR.");  
        return 0;  
    }  
  
    float keliling(){  
        System.out.println("MENGHITUNG KELILING BANGUN DATAR");  
        System.out.println();  
        return 0;  
    }  
}
```

Kelas ini merupakan kelas dasar (parent class) untuk berbagai bangun datar. Ia menyediakan metode umum untuk menghitung luas dan keliling, yang dapat di-override oleh kelas turunan.

#### Metode luas():

Menampilkan pesan "MENGHITUNG LUAS BANGUN DATAR" ke konsol.

Membersihkan layar sebelum mencetak pesan menggunakan escape sequence \033[H\033[2J.

Mengembalikan nilai default 0 sebagai placeholder.

#### Metode keliling():

Menampilkan pesan "MENGHITUNG KELILING BANGUN DATAR" ke konsol.

Mengembalikan nilai default 0 sebagai placeholder.

Kelas ini dirancang sebagai template, dengan logika perhitungan spesifik yang akan diimplementasikan di kelas turunan.

## => Child Class

PersegiPanjang (turunan dari BangunDatar).

```
class Persegi extends BangunDatar {
    float sisi;

    @Override
    float luas() {
        // Menghitung luas persegi dan mengembalikan hasilnya sebagai float
        float hasil = sisi * sisi;
        System.out.println("LUAS PERSEGI: " + hasil);
        return hasil;
    }

    @Override
    float keliling() {
        // Menghitung keli float sisi dan mengembalikan hasilnya sebagai float
        float hasil = 4 * sisi;
        System.out.println("KELILING PERSEGI : " + hasil);
        System.out.println();
        return hasil;
    }
}
```

Kelas ini merupakan turunan (**child class**) dari BangunDatar. Ia meng-override metode luas dan keliling untuk menghitung luas dan keliling persegi berdasarkan nilai properti sisi.

### Properti sisi:

Menyimpan panjang sisi persegi.

### Override luas():

Menghitung luas menggunakan rumus sisi \* sisi.

Menampilkan hasil perhitungan.

### Override keliling():

Menghitung keliling menggunakan rumus 4 \* sisi.

Menampilkan hasil perhitungan.

## PersegiPanjang (turunan dari BangunDatar).

```
class PersegiPanjang extends BangunDatar {
    float panjang, lebar;

    @Override
    float luas() {
        // Menghitung luas persegi panjang dan mengembalikan hasilnya sebagai float
        float hasil = panjang * lebar;
        System.out.println("LUAS PERSEGI PANJANG : " + hasil);
        return hasil;
    }

    @Override
    float keliling() {
        // Menghitung keliling persegi panjang dan mengembalikan hasilnya sebagai float
        float hasil = 2 * (panjang + lebar);
        System.out.println("KELILING PERSEGI PANJANG : " + hasil);
        System.out.println();
        return hasil;
    }
}
```

Kelas ini merupakan turunan (child class) dari BangunDatar. Ia meng-override metode luas dan keliling untuk menghitung luas dan keliling persegi panjang berdasarkan nilai properti panjang dan lebar.

### Properti panjang dan lebar:

Menyimpan panjang dan lebar persegi panjang.

### Override luas():

Menghitung luas menggunakan rumus panjang \* lebar.

Menampilkan hasil perhitungan.

### Override keliling():

Menghitung keliling menggunakan rumus  $2 * (\text{panjang} + \text{lebar})$ .

Menampilkan hasil perhitungan.

## Segitiga (turunan dari BangunDatar).

```
class Segitiga extends BangunDatar {
    float alas, tinggi;

    @Override
    float luas() {
        // Menghitung luas segitiga dan mengembalikan hasilnya sebagai float
        float hasil = 0.5f * alas * tinggi;
        System.out.println("LUAS SEGITIGA: " + hasil);
        return hasil;
    }

    @Override
    float keliling() {
        // Keliling segitiga belum didefinisikan dengan lengkap
        System.out.println("KELILING SEGITIGA: BELUM DIDEFINISIKAN (BUTUH SISI-SISI LAIN");
        System.out.println();
        return 0;
    }
}
```

Kelas ini merupakan turunan (child class) dari BangunDatar. Ia meng-override metode luas dan keliling untuk menghitung luas segitiga dan menampilkan pesan terkait keliling segitiga.

### Properti alas dan tinggi:

Menyimpan panjang alas dan tinggi segitiga.

### Override luas():

Menghitung luas menggunakan rumus  $0.5 * \text{alas} * \text{tinggi}$ .

Menampilkan hasil perhitungan.

### Override keliling():

Menampilkan pesan bahwa keliling segitiga belum didefinisikan karena membutuhkan informasi sisi-sisi lain.

Mengembalikan nilai 0 sebagai placeholder.

## Lingkaran (turunan dari BangunDatar).

```
class Lingkaran extends BangunDatar {
    float r;
    final float PI = 3.14f;

    @Override
    float luas() {
        // Menghitung luas lingkaran dan mengembalikan hasilnya sebagai float
        float hasil = PI * r * r;
        System.out.println("LUAS LINGKARAN : " + hasil);
        return hasil;
    }

    @Override
    float keliling() {
        // Menghitung keliling lingkaran dan mengembalikan hasilnya sebagai float
        float hasil = 2 * PI * r;
        System.out.println("KELILING LINGKARAN : " + hasil);
        System.out.println();
        return hasil;
    }
}
```

Kelas ini merupakan turunan (child class) dari BangunDatar. Ia meng-override metode luas dan keliling untuk menghitung luas dan keliling lingkaran berdasarkan jari-jari.

### Properti r (jari-jari):

Menyimpan panjang jari-jari lingkaran.

PI adalah konstanta dengan nilai 3.14.

### Override luas():

Menghitung luas menggunakan rumus PI \* r \* r.

Menampilkan hasil perhitungan.

### Override keliling():

Menghitung keliling menggunakan rumus 2 \* PI \* r.

Menampilkan hasil perhitungan.

## => Main Class

```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Membuat objek Bangun Datar
        BangunDatar bangunDatar = new BangunDatar();

        // Membuat objek Persegi dan mengisi nilai properti
        Persegi persegi = new Persegi();
        persegi.sisi = 2;

        // Membuat objek Lingkaran dan mengisi nilai properti
        Lingkaran lingkaran = new Lingkaran();
        lingkaran.r = 22;

        // Membuat objek Persegi Panjang dan mengisi nilai properti
        PersegiPanjang persegiPanjang = new PersegiPanjang();
        persegiPanjang.panjang = 8;
        persegiPanjang.lebar = 4;

        // Membuat objek Segitiga dan mengisi nilai properti
        Segitiga mSegitiga = new Segitiga();
        mSegitiga.alas = 12;
        mSegitiga.tinggi = 8;

        // Memanggil method Luas dan Keliling
        bangunDatar.luas();
        bangunDatar.keliling();

        persegi.luas();
        persegi.keliling();

        lingkaran.luas();
        lingkaran.keliling();

        persegiPanjang.luas();
        persegiPanjang.keliling();

        mSegitiga.luas();
        mSegitiga.keliling();
    }
}
```

Kelas ini merupakan kelas utama untuk menguji fungsi dari berbagai objek bangun datar seperti persegi, lingkaran, persegi panjang, dan segitiga.

### Pembuatan Objek:

Membuat objek BangunDatar sebagai instance umum.

Membuat objek Persegi dan mengisi nilai properti sisi.

Membuat objek Lingkaran dan mengisi nilai properti r (jari-jari).

Membuat objek PersegiPanjang dan mengisi nilai properti panjang dan lebar.

Membuat objek Segitiga dan mengisi nilai properti alas dan tinggi.

### Pemanggilan Metode:

Memanggil metode luas() dan keliling() dari setiap objek untuk menghitung dan menampilkan hasil perhitungan masing-masing bangun datar.

Metode di setiap objek akan menjalankan implementasi spesifik yang telah di-override dari kelas BangunDatar.

### => Output

```
MENGHITUNG LUAS BANGUN DATAR.  
MENGHITUNG KELILING BANGUN DATAR  
  
LUAS PERSEGI: 4.0  
KELILING PERSEGI : 8.0  
  
LUAS LINGKARAN : 1519.76  
KELILING LINGKARAN : 138.16  
  
LUAS PERSEGI PANJANG : 32.0  
KELILING PERSEGI PANJANG : 24.0  
  
LUAS SEGITIGA: 48.0  
KELILING SEGITIGA: BELUM DIDEFINISIKAN (BUTUH SISI-SISI LAIN)
```

### Bagian 2: Hewan

#### => Parent Class

```
public class Hewan {  
    protected String suara;  
  
    public Hewan(){  
        this.suara = "SUARA";  
    }  
  
    public void cetak(){  
        System.out.println("HEWAN BERSUARA" +suara);  
    }  
}
```

Kelas ini merupakan kelas dasar (parent class) untuk objek hewan yang menyediakan properti dan metode umum.

### **Properti suara:**

Menyimpan suara hewan, dengan tipe data String.

Properti ini bersifat protected, yang memungkinkan akses dari kelas ini dan kelas turunan.

### **Konstruktor Hewan():**

Menginisialisasi properti suara dengan nilai default "SUARA". Ini berarti setiap objek Hewan yang dibuat akan memiliki suara dengan nilai "SUARA", kecuali kelas turunan memberikan nilai yang berbeda.

### **Metode cetak():**

Menampilkan pesan "HEWAN BERSUARA" diikuti dengan nilai dari properti suara.

Pesan dicetak menggunakan System.out.println(), yang menunjukkan output ke stream error.

```
class Mamalia extends Hewan {  
    protected String berkembangbiak;  
  
    public Mamalia(){  
        this.berkembangbiak = "BERANAK";  
    }  
  
    public void cetak(){  
        System.out.println("MAMALIA BERKEMBANG BIAK" +berkembangbiak);  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Hewan yang menambahkan properti dan metode terkait mamalia.

### **Properti berkembangbiak:**

Menyimpan cara berkembang biak mamalia, dengan tipe data String.

Properti ini diinisialisasi dengan nilai default "BERANAK", yang berarti mamalia berkembang biak dengan melahirkan.

### **Konstruktor Mamalia():**

Menginisialisasi properti berkembangbiak dengan nilai "BERANAK", menunjukkan cara berkembang biak mamalia.

### **Override cetak():**

Meng-override metode cetak() dari kelas Hewan untuk menampilkan pesan yang lebih spesifik, yaitu "MAMALIA BERKEMBANG BIAK" diikuti dengan nilai properti berkembangbiak.

Pesan ini dicetak menggunakan System.out.println().

```
class Sapi extends Mamalia {
    private String nama;
    private String umur;

    public Sapi(){
        suara = "MOOOOOHHH";
        berkembangbiak = "BERANAK";
        this.nama = "AVRI";
        this.umur = "1";
    }

    public void cetak(){
        System.out.println("DATA HEWAN MAMALIA SAPI");
        System.out.println();
        System.out.println("SUARA : " + suara);
        System.out.println("BERKEMBANG BIAK : " + berkembangbiak);
        System.out.println("NAMA : " + nama);
        System.out.println("UMUR : " + umur + "TAHUN");
    }
}
```

Kelas ini merupakan turunan (child class) dari Mamalia yang lebih spesifik menggambarkan objek Sapi.

### **Properti nama dan umur:**

nama menyimpan nama sapi, dengan tipe data String.

umur menyimpan umur sapi, dengan tipe data String.

### **Konstruktor Sapi():**

Menginisialisasi properti suara dari kelas Hewan dengan nilai "MOOOOOHHH", yang menggambarkan suara sapi.

Menginisialisasi properti berkembangbiak dari kelas Mamalia dengan nilai "BERANAK", yang menunjukkan bahwa sapi berkembang biak dengan cara melahirkan.

Menginisialisasi properti nama dengan nilai "AVRI" dan umur dengan nilai "1", yang menunjukkan nama dan umur sapi tersebut.

### **Override cetak():**

Meng-override metode cetak() dari kelas Mamalia untuk menampilkan informasi yang lebih spesifik mengenai sapi.

Informasi yang ditampilkan meliputi suara, cara berkembang biak, nama, dan umur sapi.

Setiap informasi ditampilkan menggunakan System.out.println() dengan format yang jelas.

Secara keseluruhan, kelas Sapi memperluas kelas Mamalia dengan menambahkan atribut nama dan umur, serta mengubah implementasi metode cetak() untuk menampilkan data yang lebih spesifik mengenai sapi.

```
class Kambing extends Mamalia {  
    private String nama;  
    private String umur;  
  
    public Kambing(){  
        suara = "MBEEEEEKKK";  
        berkembangbiak = "BERANAK";  
        this.nama = "TEGAR";  
        this.umur = "2";  
    }  
  
    public void cetak(){  
        System.out.println("DATA HEWAN MAMALIA KAMBING");  
        System.out.println("SUARA : " + suara);  
        System.out.println("BERKEMBANG BIAK : " + berkembangbiak);  
        System.out.println("NAMA : " + nama);  
        System.out.println("UMUR : " + umur + "TAHUN");  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Mamalia yang lebih spesifik menggambarkan objek Kambing.

### **Properti nama dan umur:**

nama menyimpan nama kambing, dengan tipe data String.

umur menyimpan umur kambing, dengan tipe data String.

### **Konstruktor Kambing():**

Menginisialisasi properti suara dari kelas Hewan dengan nilai "MBEEEEEKKK", yang menggambarkan suara kambing.

Menginisialisasi properti berkembangbiak dari kelas Mamalia dengan nilai "BERANAK", yang menunjukkan bahwa kambing berkembang biak dengan cara melahirkan.

Menginisialisasi properti nama dengan nilai "TEGAR" dan umur dengan nilai "2", yang menunjukkan nama dan umur kambing tersebut.

### Override cetak():

Meng-override metode cetak() dari kelas Mamalia untuk menampilkan informasi yang lebih spesifik mengenai kambing.

Informasi yang ditampilkan meliputi suara, cara berkembang biak, nama, dan umur kambing.

Setiap informasi ditampilkan menggunakan System.out.println() dengan format yang jelas.

Secara keseluruhan, kelas Kambing memperluas kelas Mamalia dengan menambahkan atribut nama dan umur, serta mengubah implementasi metode cetak() untuk menampilkan data yang lebih spesifik mengenai kambing.

```
class Unggas extends Hewan {  
    protected String berkembangBiak;  
  
    public Unggas(){  
        this.berkembangBiak = "BERTELUR";  
    }  
  
    public void cetak(){  
        System.out.println("BERTELUR");  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Hewan yang menggambarkan objek Unggas (seperti burung atau ayam).

### Properti berkembangBiak:

Menyimpan cara berkembang biak unggas, dengan tipe data String.

Properti ini diinisialisasi dengan nilai default "BERTELUR", yang menunjukkan bahwa unggas berkembang biak dengan cara bertelur.

### Konstruktor Unggas():

Menginisialisasi properti berkembangBiak dengan nilai "BERTELUR", yang menunjukkan cara berkembang biak unggas.

### **Override cetak():**

Meng-override metode cetak() dari kelas Hewan untuk menampilkan informasi yang lebih spesifik mengenai unggas.

Dalam metode ini, hanya mencetak informasi mengenai cara berkembang biak unggas, yaitu "BERTELUR", menggunakan System.out.println().

Secara keseluruhan, kelas Unggas memperluas kelas Hewan dengan menambahkan properti khusus yang menggambarkan cara berkembang biak unggas dan meng-override metode cetak() untuk menampilkan informasi ini.

```
class Burung extends Unggas {  
    private String jenis;  
    private String ciri;  
  
    public Burung (){  
        suara = "MBEEKURR";  
        berkembangBiak = "BERTELUR";  
        this.jenis = "MERPATI";  
        this.ciri = "BULU PUTIH";  
    }  
  
    public void cetak(){  
        System.out.println("DATA HEWAN UNGGAS BURUNG");  
        System.out.println("SUARA : " + suara);  
        System.out.println("BERKEMBANG BIAK : " + berkembangBiak);  
        System.out.println("JENIS : " + jenis);  
        System.out.println("CIRI : " + ciri);  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Unggas yang menggambarkan objek Burung lebih spesifik.

### **Properti jenis dan ciri:**

jenis menyimpan jenis burung, dengan tipe data String.

ciri menyimpan ciri khas burung, dengan tipe data String.

### **Konstruktor Burung():**

Menginisialisasi properti suara dari kelas Hewan dengan nilai "MBEEKURR", yang menggambarkan suara burung.

Menginisialisasi properti berkembangBiak dari kelas Unggas dengan nilai "BERTELUR", yang menunjukkan bahwa burung berkembang biak dengan cara bertelur.

Menginisialisasi properti jenis dengan nilai "MERPATI" dan ciri dengan nilai "BULU PUTIH", yang menggambarkan jenis burung dan ciri fisiknya.

### **Override cetak():**

Meng-override metode cetak() dari kelas Unggas untuk menampilkan informasi yang lebih spesifik mengenai burung.

Informasi yang ditampilkan meliputi suara, cara berkembang biak, jenis, dan ciri khas burung.

Setiap informasi ditampilkan menggunakan System.out.println().

Secara keseluruhan, kelas Burung memperluas kelas Unggas dengan menambahkan atribut jenis dan ciri, serta mengubah implementasi metode cetak() untuk menampilkan data yang lebih spesifik mengenai burung.

```
class Ayam extends Unggas {  
    private String jenis;  
    private String ciri;  
  
    public Ayam(){  
        suara = "POKK POKK PETOKK";  
        berkembangBiak = "BERTELUR";  
        this.jenis = "AYAM KAMPUNG";  
        this.ciri = "BULU BERCORAK PUTIH HITAM";  
    }  
  
    public void cetak(){  
        System.out.println("DATA HEWAN UNGGAS AYAM");  
        System.out.println("SUARA : " + suara);  
        System.out.println("BERKEMBANG BIAK : " + berkembangBiak);  
        System.out.println("JENIS : " + jenis);  
        System.out.println("CIRI : " + ciri);  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Unggas yang menggambarkan objek Ayam.

### **Properti jenis dan ciri:**

jenis menyimpan jenis ayam, dengan tipe data String.

ciri menyimpan ciri khas ayam, dengan tipe data String.

### **Konstruktor Ayam():**

Menginisialisasi properti suara dari kelas Hewan dengan nilai "POKK POKK PETOKK", yang menggambarkan suara ayam.

Menginisialisasi properti berkembangBiak dari kelas Unggas dengan nilai "BERTELUR", yang menunjukkan bahwa ayam berkembang biak dengan cara bertelur.

Menginisialisasi properti jenis dengan nilai "AYAM KAMPUNG" dan ciri dengan nilai "BULU BERCORAK PUTIH HITAM", yang menggambarkan jenis ayam dan ciri fisiknya.

### Override cetak():

Meng-override metode cetak() dari kelas Unggas untuk menampilkan informasi yang lebih spesifik mengenai ayam.

Informasi yang ditampilkan meliputi suara, cara berkembang biak, jenis, dan ciri khas ayam.

Setiap informasi ditampilkan menggunakan System.out.println().

Secara keseluruhan, kelas Ayam memperluas kelas Unggas dengan menambahkan atribut jenis dan ciri, serta mengubah implementasi metode cetak() untuk menampilkan data yang lebih spesifik mengenai ayam.

```
class Ikan extends Hewan {  
    protected String napas;  
  
    public Ikan(){  
        this.napas = "INSANG";  
    }  
  
    public void cetak(){  
        System.out.println("INSANG");  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Hewan yang menggambarkan objek Ikan.

### Properti napas:

napas menyimpan cara ikan bernapas, dengan tipe data String.

Properti ini diinisialisasi dengan nilai default "INSANG", yang menunjukkan bahwa ikan bernapas menggunakan insang.

### Konstruktor Ikan():

Menginisialisasi properti napas dengan nilai "INSANG", yang menggambarkan cara ikan bernapas.

### **Override cetak():**

Meng-override metode cetak() dari kelas Hewan untuk menampilkan informasi yang lebih spesifik mengenai ikan.

Dalam metode ini, hanya mencetak informasi mengenai cara bernapas ikan, yaitu "INSANG", menggunakan System.out.println().

Secara keseluruhan, kelas Ikan memperluas kelas Hewan dengan menambahkan properti khusus yang menggambarkan cara ikan bernapas dan meng-override metode cetak() untuk menampilkan informasi ini.

```
class Gurami extends Ikan {
    private String Ciri;
    private String Berat;

    public Gurami(){
        suara = "(SUARA HATI)";
        napas = "INSANG";
        this.Ciri = "BERWARNA HITAM";
        this.Berat = "2";
    }

    public void cetak(){
        System.out.println("DATA HEWAN IKAN GURAMI");
        System.out.println("SUARA : " + suara);
        System.out.println("BERNAFAS : " + napas);
        System.out.println("CIRI : " + Ciri);
        System.out.println("BERAT : " + Berat + "KG");
    }
}
```

Kelas ini merupakan turunan (child class) dari Ikan yang lebih spesifik menggambarkan objek Gurami.

### **Properti Ciri dan Berat:**

Ciri menyimpan ciri fisik ikan gurami, dengan tipe data String.

Berat menyimpan berat ikan gurami, dengan tipe data String.

### **Konstruktor Gurami():**

Menginisialisasi properti suara dari kelas Hewan dengan nilai "(SUARA HATI)", yang memberikan representasi suara gurami (meskipun ini lebih bersifat metaforis, karena ikan tidak bersuara seperti hewan lainnya).

Menginisialisasi properti napas dari kelas Ikan dengan nilai "INSANG", yang menunjukkan bahwa gurami bernapas dengan insang.

Menginisialisasi properti Ciri dengan nilai "BERWARNA HITAM", menggambarkan ciri fisik ikan gurami.

Menginisialisasi properti Berat dengan nilai "2", yang menggambarkan berat ikan gurami dalam kilogram.

### Override cetak():

Meng-override metode cetak() dari kelas Ikan untuk menampilkan informasi yang lebih spesifik mengenai ikan gurami.

Informasi yang ditampilkan meliputi suara, cara bernapas, ciri fisik, dan berat ikan gurami.

Setiap informasi ditampilkan menggunakan System.out.println().

Secara keseluruhan, kelas Gurami memperluas kelas Ikan dengan menambahkan atribut Ciri dan Berat, serta mengubah implementasi metode cetak() untuk menampilkan data yang lebih spesifik mengenai ikan gurami.

```
class Lele extends Ikan {  
    private String Ciri;  
    private String Berat;  
  
    public Lele(){  
        suara = "---";  
        napas = "INSANG";  
        this.Ciri = "BERWARNA HITAM KEPUTIHAN";  
        this.Berat = "3";  
    }  
    public void cetak(){  
        System.out.println("DATA HEWAN IKAN LELE");  
        System.out.println("SUARA : " + suara);  
        System.out.println("BERNAFAS : " + napas);  
        System.out.println("CIRI : " + Ciri);  
        System.out.println("BERAT : " + Berat + "KG");  
    }  
}
```

Kelas ini merupakan turunan (child class) dari Ikan yang menggambarkan objek Lele.

### Properti Ciri dan Berat:

Ciri menyimpan ciri fisik ikan lele, dengan tipe data String.

Berat menyimpan berat ikan lele, dengan tipe data String.

### **Konstruktor Lele():**

Menginisialisasi properti suara dari kelas Hewan dengan nilai " --- ", yang menggambarkan bahwa ikan lele tidak memiliki suara yang terdengar (diwakili dengan tanda --- ).

Menginisialisasi properti napas dari kelas Ikan dengan nilai "INSANG", yang menunjukkan bahwa lele bernapas dengan insang.

Menginisialisasi properti Ciri dengan nilai "BERWARNA HITAM KEPUTIHAN", menggambarkan ciri fisik ikan lele.

Menginisialisasi properti Berat dengan nilai "3", yang menggambarkan berat ikan lele dalam kilogram.

### **Override cetak():**

Meng-override metode cetak() dari kelas Ikan untuk menampilkan informasi yang lebih spesifik mengenai ikan lele.

Informasi yang ditampilkan meliputi suara, cara bernapas, ciri fisik, dan berat ikan lele.

Setiap informasi ditampilkan menggunakan System.out.println().

Secara keseluruhan, kelas Lele memperluas kelas Ikan dengan menambahkan atribut Ciri dan Berat, serta mengubah implementasi metode cetak() untuk menampilkan data yang lebih spesifik mengenai ikan lele.

## => Main Class

```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        System.out.print("\033[H\033[2J");

        Sapi sp1 = new Sapi();
        sp1.cetak();
        System.out.println("");

        Kambing kmb1 = new Kambing();
        kmb1.cetak();
        System.out.println("");

        Burung brg1 = new Burung();
        brg1.cetak();
        System.out.println("");

        Ayam aym1 = new Ayam();
        aym1.cetak();
        System.out.println("");

        Gurami grm1 = new Gurami();
        grm1.cetak();
        System.out.println("");

        Lele l11 = new Lele();
        l11.cetak();
        System.out.println("");
    }
}
```

Kelas Main ini digunakan untuk menjalankan program dan menguji semua kelas yang telah didefinisikan sebelumnya, seperti Sapi, Kambing, Burung, Ayam, Gurami, dan Lele.

### Instansiasi objek dan pemanggilan metode cetak():

Untuk setiap kelas (seperti Sapi, Kambing, Burung, dll.), objek baru dibuat, dan metode cetak() dipanggil untuk menampilkan informasi terkait masing-masing objek.

**Sapi:** Sapi sp1 = new Sapi();

Memanggil konstruktor dari kelas Sapi, yang kemudian menjalankan metode cetak() untuk menampilkan data tentang sapi.

**Kambing:** Kambing kmb1 = new Kambing();

Sama halnya, objek Kambing dibuat dan cetak() dipanggil untuk menampilkan informasi kambing.

**Burung:** Burung brg1 = new Burung();

Objek Burung dibuat dan informasi terkait burung ditampilkan.

**Ayam:** Ayam aym1 = new Ayam();

Objek Ayam dibuat dan informasi ayam ditampilkan.

**Gurami:** Gurami grm1 = new Gurami();

Objek Gurami dibuat dan informasi ikan gurami ditampilkan.

**Lele:** Lele ll1 = new Lele();

Objek Lele dibuat dan informasi ikan lele ditampilkan.

### **Kesimpulan:**

Kesimpulannya, program ini dirancang untuk mengimplementasikan konsep pewarisan dan polimorfisme dalam pemrograman berorientasi objek. Bagian pertama menggambarkan bagaimana kelas induk **BangunDatar** digunakan sebagai template untuk menghitung luas dan keliling berbagai bangun datar, seperti persegi, persegi panjang, segitiga, dan lingkaran, dengan metode yang di-override sesuai kebutuhan spesifik masing-masing bentuk. Bagian kedua mendemonstrasikan hierarki pewarisan pada objek hewan, dimulai dari kelas induk **Hewan** hingga turunan yang lebih spesifik seperti **Mamalia**, **Unggas**, dan **Ikan**, yang selanjutnya diperluas dengan detail unik pada objek seperti **Sapi**, **Kambing**, **Burung**, **Ayam**, **Gurami**, dan **Lele**. Setiap turunan meng-override metode untuk menampilkan informasi yang lebih detail sesuai sifat, suara, cara berkembang biak, dan karakteristik masing-masing. Program ini berhasil menunjukkan bagaimana konsep pewarisan dapat digunakan untuk meningkatkan fleksibilitas dan modularitas kode, sementara polimorfisme memungkinkan setiap objek memberikan perilaku yang spesifik dalam kerangka yang seragam. Hal ini menjadikan kode lebih terstruktur, efisien, dan mudah diperluas.

## LAPORAN PRAKTIKUM 7

### 7.1 Penjelasan Kode Pertemuan 9

#### Bagian 1: Hewan

```
// Superclass
class Animal {
    // Properti
    String name;

    // Constructor
    public Animal(String name) {
        this.name = name;
    }

    // Metode
    public void speak() {
        System.out.println("SUARA HEWAN");
    }
}
```

**Superclass:** Animal adalah superclass yang memiliki properti name dan metode speak(). Metode speak() akan mencetak "SUARA HEWAN" ke layar.

```
// Subclass
class Dog extends Animal {

    // Constructor
    public Dog(String name) {
        super(name); // Memanggil constructor dari superclass
    }

    // Override metode speak
    @Override
    public void speak()
    {
        System.out.println(name + " BILANG GUKGUKGUK");
    }
}
```

**Subclass:** Dog adalah subclass yang extends Animal. Subclass ini memiliki constructor yang memanggil constructor dari superclass menggunakan kata kunci super.

**Override metode:** Subclass Dog override metode speak() dari superclass Animal.

Metode speak() pada subclass Dog akan mencetak "DIKDOG BILANG GUKGUKGUK" ke layar.

```
public class HewanTester {
    Run | Debug
    public static void main(String[] args)
    {
        System.out.print("\033[H\033[2J");
        Dog dog = new Dog(name:"DIKDOG");
        dog.speak();
    }
}
```

**Penggunaan:** Pada class HewanTester, objek Dog dibuat dengan nama "DIKDOG" dan metode speak() dipanggil. Karena metode speak() telah di-override pada subclass Dog, maka yang akan dicetak adalah "DIKDOG BILANG GUKGUKGUK".

## Bagian 2: Kendaraan

```
class Vehicle {
    String brand;

    // Constructor
    public Vehicle(String brand) {
        this.brand = brand;
    }

    // Metode untuk menampilkan informasi kendaraan
    public void displayInfo() {
        System.out.println("BRAND : " + brand);
    }
}
```

**Superclass:** Vehicle adalah superclass yang memiliki properti brand dan metode displayInfo() untuk menampilkan informasi kendaraan.

```

// Subclass
class Car extends Vehicle {
    int doors;

    // Constructor
    public Car(String brand, int doors) {
        super(brand);
        this.doors = doors;
    }

    // Overloading metode displayInfo
    public void displayInfo(String type)
    {
        super.displayInfo();
        System.out.println("TIPE : " + type);
        System.out.println("PINTU : " + doors);
    }
}

```

**Subclass:** Car adalah subclass yang extends Vehicle. Subclass ini memiliki properti doors dan constructor yang memanggil constructor dari superclass menggunakan kata kunci super.

**Overloading metode:** Subclass Car memiliki metode displayInfo() yang overload metode displayInfo() dari superclass Vehicle. Metode displayInfo() pada subclass Car memiliki parameter tambahan type dan menampilkan informasi tambahan tentang kendaraan, seperti tipe dan jumlah pintu.

```

public class KendaraanTester {
    Run | Debug
    public static void main(String[] args)
    {
        System.out.print("\033[H\033[2J");

        Car car = new Car(brand:"TOYOTA", doors:4);
        car.displayInfo(type:"SEDAN");
    }
}

```

**Penggunaan:** Pada class KendaraanTester, objek Car dibuat dengan brand "TOYOTA" dan jumlah pintu 4. Metode displayInfo() dipanggil dengan parameter "SEDAN" untuk menampilkan informasi tentang kendaraan.

### Bagian 3: Pegawai

```
public class Pegawai {  
    String nama;  
    int id_pegawai;  
    String gaji;  
  
    public void menampilkan(){  
        System.out.println("MENAMPILKAN NAMA, ID PEGAWAI, GAJI DAN TUGAS.");  
        System.out.println("-----");  
    }  
}
```

**Kelas Pegawai:** Kelas Pegawai memiliki beberapa atribut (properti) yang merepresentasikan informasi tentang pegawai, yaitu:

nama: nama pegawai

id\_pegawai: ID pegawai

gaji: gaji pegawai

**Metode menampilkan:** Kelas Pegawai memiliki metode menampilkan() yang digunakan untuk menampilkan informasi tentang pegawai. Namun, pada contoh ini, metode menampilkan() hanya menampilkan teks yang menyatakan bahwa informasi tentang pegawai akan ditampilkan, tetapi tidak benar-benar menampilkan informasi tentang pegawai.

```
public class Kasir extends Pegawai {  
  
    @Override  
    public void menampilkan()  
    {  
        System.out.println("NAMA : " + nama);  
        System.out.println("ID PEGAWAI : " + id_pegawai);  
        System.out.println("GAJI : " + gaji);  
    }  
  
    public void tugas() {  
        System.out.println("TUGAS : MELAKUKAN TRANSAKSI DENGAN PEMBELI.");  
        System.out.println();  
    }  
}
```

**Kelas Kasir:** Kelas Kasir adalah subclass dari kelas Pegawai. Artinya, kelas Kasir memiliki semua atribut dan metode yang dimiliki oleh kelas Pegawai.

**Override metode menampilkan:** Kelas Kasir override metode menampilkan() dari kelas Pegawai. Metode menampilkan() pada kelas Kasir digunakan untuk menampilkan informasi tentang kasir, seperti nama, ID pegawai, dan gaji.

**Metode tugas:** Kelas Kasir memiliki metode tugas() yang digunakan untuk menampilkan tugas yang dilakukan oleh kasir, yaitu melakukan transaksi dengan pembeli.

```

public class Koki extends Pegawai {

    @Override
    public void menampilkan()
    {
        System.out.println("NAMA : " + nama);
        System.out.println("ID PEGAWAI : " + id_pegawai);
        System.out.println("GAJI : " + gaji);
    }

    public void tugas() {
        System.out.println("TUGAS : MEMASAK MAKANAN DAN MEMBUAT MINUMAN.");
        System.out.println();
    }
}

```

**Kelas Koki:** Kelas Koki adalah subclass dari kelas Pegawai. Artinya, kelas Koki memiliki semua atribut dan metode yang dimiliki oleh kelas Pegawai.

**Override metode menampilkan:** Kelas Koki override metode menampilkan() dari kelas Pegawai. Metode menampilkan() pada kelas Koki digunakan untuk menampilkan informasi tentang koki, seperti nama, ID pegawai, dan gaji.

**Metode tugas:** Kelas Koki memiliki metode tugas() yang digunakan untuk menampilkan tugas yang dilakukan oleh koki, yaitu memasak makanan dan membuat minuman.

```

public class Manager extends Pegawai {

    @Override
    public void menampilkan()
    {
        System.out.println("NAMA : " + nama);
        System.out.println("ID PEGAWAI : " + id_pegawai);
        System.out.println("GAJI : " + gaji);
    }

    public void tugas() {
        System.out.println("TUGAS : MELAKUKAN MANAJEMEN UNTUK FRANCHISE.");
        System.out.println();
    }
}

```

**Kelas Manager:** Kelas Manager adalah subclass dari kelas Pegawai. Artinya, kelas Manager memiliki semua atribut dan metode yang dimiliki oleh kelas Pegawai.

**Override metode menampilkan:** Kelas Manager override metode menampilkan() dari kelas Pegawai. Metode menampilkan() pada kelas Manager digunakan untuk menampilkan informasi tentang manager, seperti nama, ID pegawai, dan gaji.

**Metode tugas:** Kelas Manager memiliki metode tugas() yang digunakan untuk menampilkan tugas yang dilakukan oleh manager, yaitu melakukan manajemen untuk franchise.

```

public class Pelayan extends Pegawai {

    @Override
    public void menampilkan()
    {
        System.out.println("NAMA : " + nama);
        System.out.println("ID PEGAWAI : " + id_pegawai);
        System.out.println("GAJI : " + gaji);
    }

    public void tugas() {
        System.out.println("Loading... MELAYANI DAN MENYAJIKAN PESANAN PEMBELI.");
        System.out.println();
    }
}

```

**Kelas Pelayan:** Kelas Pelayan adalah subclass dari kelas Pegawai. Artinya, kelas Pelayan memiliki semua atribut dan metode yang dimiliki oleh kelas Pegawai.

**Override metode menampilkan:** Kelas Pelayan override metode menampilkan() dari kelas Pegawai. Metode menampilkan() pada kelas Pelayan digunakan untuk menampilkan informasi tentang pelayan, seperti nama, ID pegawai, dan gaji.

**Metode tugas:** Kelas Pelayan memiliki metode tugas() yang digunakan untuk menampilkan tugas yang dilakukan oleh pelayan, yaitu melayani dan menyajikan pesanan pembeli.

```

public class Satpam extends Pegawai {

    public void menampilkan()
    {
        System.out.println("NAMA : " + nama);
        System.out.println("ID PEGAWAI : " + id_pegawai);
        System.out.println("GAJI : " + gaji);
    }

    public void tugas() {
        System.out.println("TUGAS : MENJAGA KEAMANAN DI DALAM DAN DI LUAR FRANCHISE.");
        System.out.println();
    }
}

```

**Kelas Satpam:** Kelas Satpam adalah subclass dari kelas Pegawai. Artinya, kelas Satpam memiliki semua atribut dan metode yang dimiliki oleh kelas Pegawai.

**Metode menampilkan:** Kelas Satpam memiliki metode menampilkan() yang digunakan untuk menampilkan informasi tentang satpam, seperti nama, ID pegawai, dan gaji.

**Metode tugas:** Kelas Satpam memiliki metode tugas() yang digunakan untuk menampilkan tugas yang dilakukan oleh satpam, yaitu menjaga keamanan di dalam dan di luar franchise.

```

public class Main {
    Run | Debug
    public static void main(String[] args) {
        System.out.print("\033[H\033[2J");

        // Membuat objek berdasarkan class
        Manager manager = new Manager();
        Kasir kasir = new Kasir();
        Koki koki = new Koki();
        Pelayan pelayan = new Pelayan();
        Satpam satpam = new Satpam();

        // Memasukkan nilai variabel menggunakan objek
        manager.nama = "SYIFA";
        manager.id_pegawai = 1;
        manager.gaji = "7 JUTA";

        kasir.nama = "ALDI";
        kasir.id_pegawai = 2;
        kasir.gaji = "3 JUTA";

        koki.nama = "REZA";
        koki.id_pegawai = 3;
        koki.gaji = "2 JUTA";

        pelayan.nama = "DEAN";
        pelayan.id_pegawai = 4;
        pelayan.gaji = "1,2 JUTA";

        satpam.nama = "DIKI";
        satpam.id_pegawai = 5;
        satpam.gaji = "3 JUTA";

        // Menjalankan fungsi pada superclass

        manager.menampilkan();
        manager.tugas();

        kasir.menampilkan();
        kasir.tugas();

        koki.menampilkan();
        koki.tugas();

        pelayan.menampilkan();
        pelayan.tugas();

        satpam.menampilkan();
        satpam.tugas();
    }
}

```

**Membuat objek:** Kode tersebut membuat objek-objek berdasarkan kelas Manager, Kasir, Koki, Pelayan, dan Satpam.

**Memasukkan nilai variabel:** Kode tersebut memasukkan nilai variabel ke dalam objek-objek yang telah dibuat.

**Menjalankan fungsi:** Kode tersebut menjalankan fungsi menampilkan() dan tugas() pada setiap objek untuk menampilkan informasi tentang pegawai dan tugas yang dilakukan oleh mereka.

### Output:

```
NAMA : SYIFA  
ID PEGAWAI : 1  
GAJI : 7 JUTA  
TUGAS : MELAKUKAN MANAJEMEN UNTUK FRANCHISE.  
  
NAMA : ALDI  
ID PEGAWAI : 2  
GAJI : 3 JUTA  
TUGAS : MELAKUKAN TRANSAKSI DENGAN PEMBELI.  
  
NAMA : REZA  
ID PEGAWAI : 3  
GAJI : 2 JUTA  
TUGAS : MEMASAK MAKANAN DAN MEMBUAT MINUMAN.  
  
NAMA : DEAN  
ID PEGAWAI : 4  
GAJI : 1,2 JUTA  
TUGAS : MELAYANI DAN MENYAJIKAN PESANAN PEMBELI.  
  
NAMA : DIKI  
ID PEGAWAI : 5  
GAJI : 3 JUTA  
TUGAS : MENJAGA KEAMANAN DI DALAM DAN DI LUAR FRANCHISE.
```

### Kesimpulan:

Kesimpulannya, implementasi pemrograman berorientasi objek ini menunjukkan penggunaan pewarisan, overriding, dan overloading untuk menciptakan struktur kelas yang efisien. Pada bagian pertama, subclass **Dog** meng-override metode dari **Animal**. Pada bagian kedua, **Car** me-overload metode **displayInfo** dari **Vehicle**. Pada bagian ketiga, berbagai subclass **Pegawai** seperti **Manager** dan **Kasir** meng-override metode untuk menampilkan informasi dan tugas spesifik masing-masing pegawai. Ini menunjukkan fleksibilitas dan modularitas dalam pemrograman berorientasi objek.

## LAPORAN PRAKTIKUM 8

### 8.1 Penjelasan Kode Pertemuan 10

#### Polymorfisme Dinamis

=> Abstract

```
package Pertemuan10.PolymorfismeDinamis.Abstract;

public class Pegawai {
    private String NIP;
    private String nama;

    // Constructor untuk inisialisasi nama dan NIP
    public Pegawai(String nama, String NIP) {
        this.nama = nama;
        this.NIP = NIP;
    }

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

    // Getter untuk NIP
    public String getNIP() {
        return NIP;
    }

    // Method untuk mengirim email
    public void kirimEmail(String to, String subjek, String isi) {
        System.out.println(getNama() + " Kirim email ke : " + to);
        System.out.println("Dengan Subjek : " + subjek);
        System.out.println("Dengan Isi : " + isi);
    }
}
```

```
package Pertemuan10.PolymorfismeDinamis.Abstract;

public class Staff extends Pegawai {
    private String bagian;
    public Staff(String nama, String NIP, String bagian){
        super(nama,NIP);
        setBagian(bagian);
    }

    public void setBagian(String namabagian){
        bagian=namabagian;
    }

    public String getBagian(){
        return bagian;
    }
}
```

```

package Pertemuan10.PolimorfismeDinamis.Abstract;

public class StaffTester {
    Run | Debug
    public static void main(String[] args) {
        Staff s = new Staff(nama:"Januar", NIP:"1234", bagian:"Keuangan");
        s.kirimEmail(to:"a@test.com", subjek:"info test", isi:"isi email");
        System.out.println("NIP : " + s.getNIP() + "\n" + "Nama : " +
        s.getNama() + "\n" + "Bagian : " + s.getBagian());
    }
}

```

## Struktur Program

Program ini terdiri dari **tiga kelas**:

1. **Kelas Pegawai** (Superclass)
2. **Kelas Staff** (Subclass, turunan dari Pegawai)
3. **Kelas StaffTester** (Kelas utama untuk menjalankan program)

### 1. Kelas Pegawai

Kelas Pegawai berperan sebagai **kelas induk** yang menyimpan data umum dari pegawai.

#### Atribut

- private String NIP → Menyimpan Nomor Induk Pegawai.
- private String nama → Menyimpan nama pegawai.

#### Constructor

public Pegawai(String nama, String NIP)

- **Tujuan:** Menginisialisasi atribut nama dan NIP saat objek Pegawai dibuat.

#### Metode

1. **public String getNama()**
  - Mengembalikan nilai nama.
2. **public String getNIP()**
  - Mengembalikan nilai NIP.
3. **public void kirimEmail(String to, String subjek, String isi)**
  - Menampilkan simulasi pengiriman email ke layar konsol dengan format:
  - [Nama Pegawai] Kirim email ke : [tujuan email]
  - Dengan Subjek : [subjek email]
  - Dengan Isi : [isi email]

### 2. Kelas Staff

Kelas Staff adalah **subkelas** dari kelas Pegawai. Kelas ini menambahkan atribut **bagian** yang merepresentasikan bagian/departemen tempat staff bekerja.

## Atribut

- private String bagian → Menyimpan informasi bagian/departemen tempat bekerja.

## Constructor

```
public Staff(String nama, String NIP, String bagian)
```

- **Tujuan:** Menginisialisasi atribut nama, NIP, dan bagian.
- **Implementasi:**
  - Memanggil constructor superclass (super(nama, NIP)) untuk mengatur nama dan NIP.
  - Mengatur atribut bagian melalui metode setBagian().

## Metode

### 1. public void setBagian(String namabagian)

- **Tujuan:** Mengatur nilai atribut bagian.

### 2. public String getBagian()

- **Tujuan:** Mengembalikan nilai atribut bagian.

### 3. Kelas StaffTester

Kelas StaffTester digunakan untuk **menguji** fungsionalitas kelas Staff. Program ini membuat objek Staff dan memanggil metode yang dimiliki oleh kelas Pegawai dan Staff.

## Metode main

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

- **Tujuan:** Metode utama yang menjalankan program.

## Langkah-langkah Implementasi

### 1. Membuat Objek Staff

```
Staff s = new Staff("Januar", "1234", "Keuangan");
```

- **Penjelasan:**
  - Membuat objek Staff dengan parameter:
    - nama: "Januar"
    - NIP: "1234"
    - bagian: "Keuangan"
  - Constructor Staff memanggil constructor dari superclass Pegawai untuk mengatur atribut nama dan NIP.

## 2. Memanggil Metode kirimEmail()

```
s.kirimEmail("a@test.com", "info test", "isi email");
```

- **Penjelasan:**
  - Memanggil metode kirimEmail() yang diwarisi dari kelas Pegawai.
  - Menampilkan simulasi pengiriman email ke layar konsol.
  - **Output:**
    - Januar Kirim email ke : a@test.com
    - Dengan Subjek : info test
    - Dengan Isi : isi email

## 3. Menampilkan Informasi Staff

```
System.out.println("NIP : " + s.getNIP() + "\n" +
  "Nama : " + s.getNama() + "\n" +
  "Bagian : " + s.getBagian());
```

### Penjelasan:

- Memanggil metode getNIP(), getNama(), dan getBagian() untuk mengambil nilai atribut dari objek Staff.
- Menampilkan informasi ke layar konsol.
- **Output:**
  - NIP : 1234
  - Nama : Januar
  - Bagian : Keuangan

## Kesimpulan Program

1. Kelas Pegawai sebagai superclass menyimpan atribut dan metode umum yang dimiliki oleh semua pegawai.

2. Kelas Staff memperluas fungsionalitas Pegawai dengan menambahkan atribut khusus **bagian**.
3. Kelas StaffTester digunakan untuk membuat objek Staff, menguji metode pewarisan, dan menampilkan hasilnya.

### **Output Lengkap Program:**

Januar Kirim email ke : a@test.com

Dengan Subjek : info test

Dengan Isi : isi email

NIP : 1234

Nama : Januar

Bagian : Keuangan

### **=> Final**

```
package Pertemuan10.PolymorfismeDinamis.Final;

public final class MyMath {
    // Atribut
    public final double pi = 3.1416;

    // Method
    public final double luasLingkaran(double r) {
        return pi * r * r;
    }

    public final double kelilingLingkaran(double r) {
        return 2 * pi * r;
    }

    public final double sin(double derajat){
        return Math.sin(Math.toRadians(derajat));
    }

    public final double cos(double derajat){
        return Math.cos(Math.toRadians(derajat));
    }

    public final double tan(double derajat){
        return Math.tan(Math.toRadians(derajat));
    }

    public final double pangkat(double a, double b){
        return Math.pow(a, b);
    }
}
```

```

package Pertemuan10.PolimorfismeDinamis.Final;

public class Main
{
    Run | Debug
    public static void main( String[] args )
    {
        // Instance object
        MyMath mm = new MyMath();
        System.out.println("Luas Lingkaran = " + mm.luasLingkaran(r:5));
        System.out.println("Keliling Lingkaran = " + mm.kelilingLingkaran(r:5));
        System.out.println("Sin 30 derajat = " + mm.sin(derajat:30));
        System.out.println("Cos 30 derajat = " + mm.cos(derajat:30));
        System.out.println("Tan 30 derajat = " + mm.tan(derajat:30));
        System.out.println("2^8 = " + mm.pangkat(a:2, b:8));
    }
}

```

## Struktur Program

Program ini terdiri dari **dua kelas**:

- 1. Kelas MyMath**
- 2. Kelas Main**

### 1. Kelas MyMath

Kelas **MyMath** adalah kelas yang bersifat **final**, yang artinya kelas ini **tidak dapat diturunkan** (diwarisi) oleh kelas lain.

### Atribut

#### 1. public final double pi

- **Tujuan:** Menyimpan nilai konstanta  $\pi$  (pi).
- **Tipe:** double
- **Aksesibilitas:** public (dapat diakses di mana saja).
- **Final:** Nilainya **tidak dapat diubah** setelah diinisialisasi.

### Metode

Semua metode dalam kelas MyMath diberi modifier **final**. Artinya, metode ini **tidak dapat dioverride** (ditimpak) di kelas turunannya (jika memungkinkan).

#### 1. public final double luasLingkaran(double r)

- **Tujuan:** Menghitung luas lingkaran.
- **Parameter:**
  - double r: Jari-jari lingkaran.
- **Rumus:**  $\pi \times r \times r$
- **Return:** Nilai luas lingkaran bertipe double.

## 2. **public final double kelilingLingkaran(double r)**

- **Tujuan:** Menghitung keliling lingkaran.
- **Parameter:**
  - double r: Jari-jari lingkaran.
- **Rumus:**  $2 \times \pi \times r^2$
- **Return:** Nilai keliling lingkaran bertipe double.

## 3. **public final double sin(double derajat)**

- **Tujuan:** Menghitung nilai sinus dari sudut tertentu.
- **Parameter:**
  - double derajat: Sudut dalam satuan derajat.
- **Implementasi:**
  - Konversi derajat ke radian menggunakan Math.toRadians(derajat).
  - Hitung sinus menggunakan Math.sin().
- **Return:** Nilai sinus bertipe double.

## 4. **public final double cos(double derajat)**

- **Tujuan:** Menghitung nilai cosinus dari sudut tertentu.
- **Parameter:**
  - double derajat: Sudut dalam satuan derajat.
- **Return:** Nilai cosinus bertipe double.

## 5. **public final double tan(double derajat)**

- **Tujuan:** Menghitung nilai tangen dari sudut tertentu.
- **Parameter:**
  - double derajat: Sudut dalam satuan derajat.
- **Return:** Nilai tangen bertipe double.

## 6. **public final double pangkat(double a, double b)**

- **Tujuan:** Menghitung nilai pangkat.
- **Parameter:**
  - double a: Bilangan dasar (base).
  - double b: Pangkat (eksponen).
- **Implementasi:** Menggunakan metode Math.pow(a, b).
- **Return:** Hasil perpangkatan bertipe double.

## **2. Kelas Main**

Kelas Main berisi metode main untuk menguji fungsionalitas kelas **MyMath**.

### **Metode main**

public static void main(String[] args)

**Tujuan:** Metode utama yang digunakan untuk menjalankan program.

### **Implementasi:**

Membuat objek MyMath bernama mm.

Memanggil metode-metode dalam kelas MyMath untuk melakukan perhitungan.

Menampilkan hasil perhitungan ke layar konsol.

### **Output Lengkap Program**

Luas Lingkaran = 78.54

Keliling Lingkaran = 31.416

Sin 30 derajat = 0.5

Cos 30 derajat = 0.8660254037844386

Tan 30 derajat = 0.5773502691896257

$2^8 = 256.0$

## => Gambar

```
package Pertemuan10.PolymorfismeDinamis.Gambar;

public class Bentuk {
    protected void gambar() {
        System.out.println("superclass -> Menggambar");
    }

    protected void hapus() {
        System.out.println("superclass -> Menghapus Gambar");
    }
}
```

```
package Pertemuan10.PolymorfismeDinamis.Gambar;

public class Elips extends Bentuk {
    protected void gambar() {
        System.out.println("subclass -> Menggambar Elips");
    }

    protected void hapus() {
        System.out.println("subclass -> Menghapus Gambar Elips");
    }
}
```

```
package Pertemuan10.PolymorfismeDinamis.Gambar;

public class Lingkaran extends Bentuk {
    protected void gambar() {
        System.out.println("subclass -> Menggambar Lingkaran");
    }

    protected void hapus() {
        System.out.println("subclass -> Menghapus Gambar Lingkaran");
    }
}
```

```
package Pertemuan10.PolymorfismeDinamis.Gambar;

public class Segitiga extends Bentuk {
    protected void gambar() {
        System.out.println("subclass -> Menggambar Segitiga");
    }

    protected void hapus() {
        System.out.println("subclass -> Menghapus Gambar Segitiga");
    }
}
```

```

package Pertemuan10.PolimorfismeDinamis.Gambar;

public class Cetakgambar extends Bentuk {
    private void tampil(Bentuk[] obj) {
        // Polimorfisme
        // Memanggil method yang sama yaitu method gambar() dan hapus()
        // pada masing-masing class
        for (int i = 0; i < obj.length; i++) {
            obj[i].gambar();
            obj[i].hapus();
            System.out.println();
        }
    }

    Run | Debug
    public static void main(String[] args) {
        // Array objek dari berbagai subclass Bentuk
        Bentuk[] obj = {
            new Lingkaran(),
            new Elips(),
            new Segitiga()
        };

        Cetakgambar cetak = new Cetakgambar();

        // Menampilkan method gambar() & hapus() pada class Bentuk (superclass)
        System.out.println("Memanggil method gambar() dan hapus() pada superclass dan subclass:");
        cetak.tampil(obj);
    }
}

```

## Kelas Bentuk (Superclass)

Kelas ini mendefinisikan dua metode:

**gambar()**: Menampilkan pesan default yang menyatakan "superclass -> Menggambar".

**hapus()**: Menampilkan pesan default yang menyatakan "superclass -> Menghapus Gambar".

Metode-metode ini nantinya akan di-*override* oleh kelas-kelas yang mewarisi kelas Bentuk.

## Kelas Lingkaran, Elips, dan Segitiga (Subclass)

Ketiga kelas ini adalah **subclass** dari kelas Bentuk, dan mereka masing-masing meng-override metode gambar() dan hapus() untuk memberikan implementasi yang lebih spesifik sesuai dengan jenis objek mereka.

**Lingkaran** mengimplementasikan metode gambar() untuk menampilkan pesan "Menggambar Lingkaran" dan hapus() untuk menampilkan pesan "Menghapus Gambar Lingkaran".

**Elips** mengimplementasikan metode gambar() untuk menampilkan pesan "Menggambar Elips" dan hapus() untuk menampilkan pesan "Menghapus Gambar Elips".

**Segitiga** mengimplementasikan metode gambar() untuk menampilkan pesan "Menggambar Segitiga" dan hapus() untuk menampilkan pesan "Menghapus Gambar Segitiga".

## Kelas Cetakgambar (Main Program)

Di kelas Cetakgambar, terdapat metode **tampil()** yang menerima array objek bertipe Bentuk. Dalam metode ini, program melakukan iterasi untuk memanggil metode gambar() dan hapus() pada setiap objek dalam array.

Meskipun referensi yang digunakan adalah Bentuk[], yang dipanggil adalah implementasi spesifik dari metode gambar() dan hapus() sesuai dengan jenis objek yang ada (misalnya, Lingkaran, Elips, atau Segitiga). Ini adalah contoh dari **Polimorfisme Dinamis**, di mana metode yang sama (gambar() dan hapus()) akan berperilaku berbeda tergantung pada jenis objek yang memanggilnya.

## Array Objek Bentuk[]:

Dalam program, dibuat array objek yang berisi objek-objek dari kelas-kelas Lingkaran, Elips, dan Segitiga. Ketiga objek tersebut semuanya memiliki tipe referensi Bentuk[], tetapi masing-masing objek ini akan menjalankan implementasi metode yang sesuai dari kelas masing-masing.

## Pemanggilan Metode:

Metode **gambar()** dan **hapus()** dipanggil pada setiap objek dalam array, dan meskipun objek-objek tersebut bertipe Bentuk, mereka menjalankan metode yang *di-override* sesuai dengan jenis objek spesifiknya. Ini menunjukkan **Polimorfisme Dinamis** dalam aksi.

## Output Program

Saat program dijalankan, output yang dihasilkan adalah:

Memanggil method gambar() dan hapus() pada superclass dan subclass: subclass -> Menggambar Lingkaran subclass -> Menghapus Gambar Lingkaran

subclass -> Menggambar Elips subclass -> Menghapus Gambar Elips

subclass -> Menggambar Segitiga subclass -> Menghapus Gambar Segitiga

## => Interface

```
package Pertemuan10.PolimorfismeDinamis.Interface;

public interface Bidang2D {
    double getKeliling();
    double getLuas();
}
```

```
package Pertemuan10.PolymorfismeDinamis.Interface;

public class BujurSangkar implements Bidang2D {
    public double sisi;

    public double getKeliling() {
        return 4*sisi;
    }

    public double getLuas() {
        return sisi*sisi;
    }
}
```

```
package Pertemuan10.PolymorfismeDinamis.Interface;

public class Lingkaran implements Bidang2D {
    public double radius;

    public double getKeliling() {
        return 2*Math.PI*radius;
    }

    public double getLuas() {
        return Math.PI*radius*radius;
    }
}
```

```
package Pertemuan10.PolymorfismeDinamis.Interface;

public class PersegiPanjang implements Bidang2D {
    public double panjang;
    public double lebar;

    public double getKeliling() {
        return 2*(panjang + lebar);
    }

    public double getLuas() {
        return panjang*lebar;
    }
}
```

## Interface Bidang2D

Interface Bidang2D mendeklarasikan dua metode:

**getKeliling()**: Untuk menghitung keliling bidang 2D.

**getLuas()**: Untuk menghitung luas bidang 2D.

Kedua metode ini tidak memiliki implementasi di dalam interface. Setiap kelas yang

mengimplementasikan interface ini harus menyediakan implementasi spesifik untuk metode-metode tersebut.

### **Kelas BujurSangkar, Lingkaran, dan PersegiPanjang**

Ketiga kelas ini mengimplementasikan interface Bidang2D, yang berarti mereka harus mengimplementasikan kedua metode yang dideklarasikan di dalam interface, yaitu getKeliling() dan getLuas().

#### **BujurSangkar:**

**getKeliling()**: Menghitung keliling bujur sangkar dengan rumus  $4 * \text{sisi}$ .

**getLuas()**: Menghitung luas bujur sangkar dengan rumus  $\text{sisi} * \text{sisi}$ .

#### **Lingkaran:**

**getKeliling()**: Menghitung keliling lingkaran dengan rumus  $2 * \text{Math.PI} * \text{radius}$ .

**getLuas()**: Menghitung luas lingkaran dengan rumus  $\text{Math.PI} * \text{radius} * \text{radius}$ .

#### **PersegiPanjang:**

**getKeliling()**: Menghitung keliling persegi panjang dengan rumus  $2 * (\text{panjang} + \text{lebar})$ .

**getLuas()**: Menghitung luas persegi panjang dengan rumus  $\text{panjang} * \text{lebar}$ .

### **Polimorfisme Dinamis**

Meskipun ketiga kelas ini memiliki tipe yang berbeda (BujurSangkar, Lingkaran, PersegiPanjang), mereka semua mengimplementasikan interface yang sama, yaitu Bidang2D. Ini memungkinkan kita untuk menggunakan **Polimorfisme Dinamis**, di mana objek-objek dari kelas-kelas ini bisa diperlakukan sebagai objek bertipe Bidang2D.

Hal ini memungkinkan kita untuk memanggil metode getKeliling() dan getLuas() pada objek bertipe Bidang2D, tanpa peduli objek spesifik yang digunakan. Yang dieksekusi adalah implementasi metode sesuai dengan jenis objek yang ada (misalnya, Lingkaran, BujurSangkar, atau PersegiPanjang).

```

package Pertemuan10.PolymorfismeDinamis.Static;

public final class MyMath {
    // Atribut
    public static final double pi = 3.14;

    // Method
    public static double kelilingLingkaran(double r) {
        return 2 * pi * r;
    }

    public static double luasLingkaran(double r) {
        return pi * r * r;
    }

    public static double sin(double derajat){
        return Math.sin(Math.toRadians(derajat));
    }

    public static double cos(double derajat){
        return Math.cos(Math.toRadians(derajat));
    }

    public static double tan(double derajat){
        return Math.tan(Math.toRadians(derajat));
    }

    public static double pangkat(double bilangan, int pangkat){
        return Math.pow(bilangan, pangkat);
    }

    public static double getPi() {
        return pi;
    }
}

```

```

package Pertemuan10.PolymorfismeDinamis.Static;

public class Main
{
    Run | Debug
    public static void main( String[] args )
    {
        // Tidak perlu instansiasi objek
        System.out.println("Besar PI adalah " + MyMath.pi);
        System.out.println("Keliling lingkaran dengan jari-jari 5 adalah " + MyMath.kelilingLingkaran(r:5));
        System.out.println("Luas lingkaran dengan jari-jari 5 adalah " + MyMath.luasLingkaran(r:5));
        System.out.println("Sinus 30 derajat adalah " + MyMath.sin(derajat:30));
        System.out.println("Cosinus 30 derajat adalah " + MyMath.cos(derajat:30));
        System.out.println("Tangens 30 derajat adalah " + MyMath.tan(derajat:30));
        System.out.println("5 pangkat 3 adalah " + MyMath.pangkat(bilangan:5, pangkat:3));
    }
}

```

## Kelas MyMath

Kelas MyMath berisi beberapa metode statis dan satu atribut statis yang digunakan untuk melakukan perhitungan matematis seperti menghitung keliling dan luas lingkaran, menghitung nilai trigonometri, dan menghitung pangkat.

### **Atribut Statis pi:**

Atribut ini menyimpan nilai konstanta  $\pi$  (pi), yang digunakan dalam perhitungan keliling dan luas lingkaran.

Atribut ini adalah static, yang berarti nilainya dapat diakses langsung menggunakan nama kelas (MyMath.pi), tanpa perlu membuat objek dari kelas MyMath.

### **Metode Statis:**

Semua metode dalam kelas MyMath didefinisikan sebagai metode statis (menggunakan kata kunci static).

Metode ini dapat diakses langsung menggunakan nama kelas tanpa membuat objek dari kelas MyMath. Beberapa metode yang ada antara lain:

**kelilingLingkaran(double r):** Menghitung keliling lingkaran berdasarkan jari-jari r dengan rumus  $2 * \pi * r$ .

**luasLingkaran(double r):** Menghitung luas lingkaran berdasarkan jari-jari r dengan rumus  $\pi * r * r$ .

**sin(double derajat):** Menghitung sinus dari sudut dalam derajat.

**cos(double derajat):** Menghitung kosinus dari sudut dalam derajat.

**tan(double derajat):** Menghitung tangen dari sudut dalam derajat.

**pangkat(double bilangan, int pangkat):** Menghitung nilai bilangan yang dipangkatkan dengan pangkat yang diberikan.

**getPi():** Mengembalikan nilai konstanta  $\pi$  (pi).

### **Kelas Main**

Kelas Main adalah kelas yang menjalankan program. Di sini, semua metode statis dari kelas MyMath digunakan secara langsung melalui nama kelas tanpa perlu membuat objek dari MyMath.

### **Pemanggilan Metode Statis:**

Dalam Main, Anda dapat melihat bahwa semua metode dipanggil langsung menggunakan nama kelas MyMath, misalnya MyMath.kelilingLingkaran(5) untuk menghitung keliling lingkaran dengan jari-jari 5, atau MyMath.sin(30) untuk menghitung nilai sinus dari sudut 30 derajat.

Tidak ada kebutuhan untuk membuat objek MyMath karena metode-metode tersebut adalah statis.

### Keuntungan Metode Statis

**Tidak Perlu Membuat Objek:** Metode statis dapat diakses langsung melalui nama kelas tanpa perlu membuat instance (objek) dari kelas tersebut. Ini menghemat memori dan membuat kode lebih ringkas jika tidak membutuhkan status objek.

**Penggunaan Bersama (Shared Usage):** Atribut dan metode statis dimiliki bersama oleh semua objek dari kelas tersebut. Dalam hal ini, nilai pi digunakan di seluruh kelas tanpa perlu membuat objek MyMath.

**Cocok untuk Fungsi yang Tidak Bergantung pada Status Objek:** Metode-metode dalam kelas MyMath adalah fungsi yang tidak bergantung pada status internal objek. Mereka hanya membutuhkan parameter yang diberikan untuk melakukan perhitungan.

### Contoh Output Program

Saat program dijalankan, output yang dihasilkan adalah sebagai berikut:

Besar PI adalah 3.14

Keliling lingkaran dengan jari-jari 5 adalah 31.400000000000002

Luas lingkaran dengan jari-jari 5 adalah 78.5

Sinus 30 derajat adalah 0.4999999999999994

Cosinus 30 derajat adalah 0.8660254037844387

Tangens 30 derajat adalah 0.5773502691896257

5 pangkat 3 adalah 125.0

Output ini mencerminkan perhitungan-perhitungan yang dilakukan oleh metode-metode statis dalam kelas MyMath.

## Polymorfisme Statis

```
package Pertemuan10.PolymorfismeStatis;

public class Lingkaran {
    //Method menghitung luas dengan jari-jari
    float luas(float r){
        return (float) (Math.PI * r * r);
    }

    //Method menghitung luas dengan diameter
    double luas (double d){
        return (double) (1/4 *Math.PI *d*d);
    }
}
```

```
package Pertemuan10.PolymorfismeStatis;

public class Overloading {
    public void Tampil() {
        System.out.println("I love Java");
    }

    public void Tampil(int i) {
        System.out.println("Method dengan 1 parameter = " + i);
    }

    public void Tampil(int i, int j) {
        System.out.println("Method dengan 2 parameter = " + i + " & " + j);
    }

    public void Tampil(String str) {
        System.out.println(str);
    }

    Run | Pertemuan10.PolymorfismeStatis.Overloading
    publ
        Overloading objek = new Overloading();
        objek.Tampil();
        objek.Tampil(i:8);
        objek.Tampil(i:6, j:7);
        objek.Tampil(str:"Hello world");
    }
}
```

```
package Pertemuan10.PolymorfismeStatis;

class Polymorph {
    public int tambah(int x, int y) {
        return x+y;
    }
    public String tambah(String x, String y) {
        return x+ " " +y;
    }
}

public class PolymorphTester {
    Run | Debug
    public static void main(String[] args) {
        Polymorph p=new Polymorph();
        System.out.println("2 + 3 = " +p.tambah(x:2, y:3));
        System.out.println("\\"2\\" + \"3\" = " +p.tambah(x:"2", y:"3"));
    }
}
```

## Kelas Lingkaran

Kelas Lingkaran memiliki dua metode dengan nama yang sama, yaitu **luas()**, tetapi dengan parameter yang berbeda:

**luas(float r)**: Menghitung luas lingkaran berdasarkan jari-jari r menggunakan rumus  $\pi * r^2$ . Parameter r bertipe float.

**luas(double d)**: Menghitung luas lingkaran berdasarkan diameter d menggunakan rumus  $(1/4) * \pi * d^2$ . Parameter d bertipe double.

**Konsep yang diterapkan**: Ini adalah contoh **Overloading**, di mana dua metode dengan nama yang sama digunakan untuk operasi yang serupa, tetapi dengan parameter yang berbeda (tipe parameter yang berbeda, yaitu float dan double).

## Kelas Overloading

Kelas ini menunjukkan **Overloading** metode Tampil() yang memiliki empat variasi:

**Tampil()**: Metode tanpa parameter, hanya mencetak "I love Java".

**Tampil(int i)**: Metode yang menerima satu parameter bertipe int, mencetak nilai parameter tersebut.

**Tampil(int i, int j)**: Metode yang menerima dua parameter bertipe int, mencetak keduanya.

**Tampil(String str)**: Metode yang menerima satu parameter bertipe String, mencetak nilai parameter tersebut.

**Konsep yang diterapkan**: Ini adalah contoh **Overloading** dalam Java, di mana metode yang memiliki nama yang sama dapat memiliki parameter yang berbeda (baik jumlah maupun tipe parameter), memungkinkan pemanggilan metode yang berbeda bergantung pada parameter yang diberikan.

**Penggunaan**: objek.Tampil(); // Memanggil metode tanpa parameter objek.Tampil(8); // Memanggil metode dengan satu parameter (int) objek.Tampil(6, 7); // Memanggil metode dengan dua parameter (int) objek.Tampil("Hello world"); // Memanggil metode dengan satu parameter (String)

## Kelas Polymorph

Kelas Polymorph memiliki dua metode dengan nama yang sama, yaitu **tambah()**, tetapi dengan tipe parameter yang berbeda:

**tambah(int x, int y)**: Menambahkan dua bilangan bulat (int).

**tambah(String x, String y)**: Menggabungkan dua string.

**Konsep yang diterapkan:** Ini adalah contoh lain dari **Overloading**. Meski nama metode sama, Java membedakan keduanya berdasarkan tipe parameter yang diterima (int vs. String).

#### Penggunaan:

```
Polymorph p = new Polymorph();
```

```
System.out.println("2 + 3 = " + p.tambah(2, 3)); // Memanggil metode dengan parameter int
```

```
System.out.println("\\"2\\" + \\"3\\" = " + p.tambah("2", "3")); // Memanggil metode dengan parameter String
```

#### Kesimpulan tentang Polimorfisme Statis dan Overloading

**Polimorfisme Statis:** Dalam hal ini, polimorfisme diterapkan melalui **Overloading**, yaitu memiliki beberapa metode dengan nama yang sama tetapi dengan parameter yang berbeda. Pemilihan metode yang tepat dilakukan pada saat kompilasi berdasarkan tipe dan jumlah parameter.

**Overloading** memungkinkan satu nama metode digunakan untuk berbagai tindakan yang terkait, tetapi berbeda dalam cara mereka diimplementasikan. Overloading meningkatkan keterbacaan dan fleksibilitas kode, karena Anda tidak perlu menggunakan nama metode yang berbeda untuk operasi yang serupa.

# LAPORAN PRAKTIKUM 9

## 9.1 Penjelasan Kode Pertemuan 11

### Operasi Bilangan 1

```
package Pertemuan11.OperasiBilangan1;

public class OperasiBilangan {
    protected double a, b, c;

    protected void set_A(double a) { this.a = a; }
    protected void set_B(double b) { this.b = b; }
    protected void set_C(double c) { this.c = c; }
    protected double get_A() { return a; }
    protected double get_B() { return b; }
    protected double get_C() { return c; }

    protected void tampil() {
        System.out.println("HASIL OPERASI : " + c);
    }
}
```

```
package Pertemuan11.OperasiPembagian;

public class OperasiPembagian extends OperasiBilangan {

    @Override
    protected void tampil() {
        if (b != 0) {
            c = a / b;
            System.out.println("PEMBAGIAN : " + a + " / " + b + " = " + c);
        } else {
            System.out.println("TIDAK DAPAT MEMBAGI DENGAN NOL.");
        }
    }
}
```

```
package Pertemuan11.OperasiBilangan1;

public class OperasiPengurangan extends OperasiBilangan {

    @Override
    protected void tampil() {
        c = a - b;
        System.out.println("PENGURANGAN : " + a + " - " + b + " = " + c );
    }
}
```

```
package Pertemuan11.OperasiBilangan1;

public class OperasiPenjumlahan extends OperasiBilangan {

    @Override
    protected void tampil() {
        c = a + b;
        System.out.println("PENJUMLAHAN : " + a + " + " + b + " = " + c );
    }
}
```

```

package Pertemuan11.OperasiBilangan1;

public class OperasiPerkalian extends OperasiBilangan {

    @Override
    protected void tampil() {
        c = a * b;
        System.out.println("PERKALIAN : " + a + " * " + b + " = " + c );
    }
}

package Pertemuan11.OperasiBilangan1;

public class PrintOperasi {
    public void cetakSemua(OperasiBilangan[] OB, double a, double b) {
        for (OperasiBilangan operasi : OB) {
            operasi.set_A(a);
            operasi.set_B(b);
            operasi.tampil();
        }
    }
}

package Pertemuan11.OperasiBilangan1;

public class Main {
    Run|Debug
    public static void main(String[] args) {

        System.out.print("\033[H\033[2J");
        System.out.println("OPERASI ARITMATIKA JAVA\n");

        // Nilai A dan B
        double a = 10.5;
        double b = 0.5;

        // Array Operasi
        OperasiBilangan[] operasiArray = {
            new OperasiPenjumlahan(),
            new OperasiPengurangan(),
            new OperasiPerkalian(),
            new OperasiPembagian()
        };

        // Cetak semua operasi
        PrintOperasi cetak = new PrintOperasi();
        cetak.cetakSemua(operasiArray, a, b);
    }
}

```

## 1. Kelas OperasiBilangan (Superclass)

Kelas ini adalah kelas dasar yang menyediakan atribut **a**, **b**, dan **c**, yang digunakan untuk menyimpan dua bilangan input (a dan b), serta hasil operasi yang disimpan dalam c.

Ada juga beberapa **metode setter** dan **getter** untuk mengatur dan mengambil nilai dari atribut a, b, dan c.

Metode **tampil()** digunakan untuk menampilkan hasil operasi, tetapi di kelas ini, hanya mencetak hasil yang disimpan dalam c tanpa melakukan perhitungan apapun. Metode ini akan dioVERRIDE pada subclass.

## 2. Kelas OperasiPembagian (Subclass)

Kelas ini mewarisi dari OperasiBilangan dan mengimplementasikan metode **tampil()** untuk menghitung pembagian antara a dan b.

**Periksa pembagiannya:** Sebelum melakukan pembagian, program memeriksa apakah b tidak sama dengan nol. Jika  $b == 0$ , maka akan mencetak pesan error ("TIDAK DAPAT MEMBAGI DENGAN NOL.") untuk mencegah kesalahan pembagian oleh nol.

Jika  $b != 0$ , maka hasil pembagian disimpan dalam c dan ditampilkan.

## 3. Kelas OperasiPengurangan (Subclass)

Kelas ini juga mewarisi dari OperasiBilangan dan mengimplementasikan metode **tampil()** untuk menghitung pengurangan antara a dan b.

Hasil pengurangan disimpan dalam c dan dicetak dalam format "PENGURANGAN : a - b = c".

## 4. Kelas OperasiPenjumlahan (Subclass)

Kelas ini juga mewarisi dari OperasiBilangan dan mengimplementasikan metode **tampil()** untuk menghitung penjumlahan antara a dan b.

Hasil penjumlahan disimpan dalam c dan ditampilkan dalam format "PENJUMLAHAN : a + b = c".

## 5. Kelas OperasiPerkalian (Subclass)

Kelas ini mewarisi dari OperasiBilangan dan mengimplementasikan metode **tampil()** untuk menghitung perkalian antara a dan b.

Hasil perkalian disimpan dalam c dan dicetak dalam format "PERKALIAN : a \* b = c".

## 6. Kelas Main

- Kelas Main adalah titik masuk program. Di sini, beberapa hal dilakukan:

Nilai untuk a dan b diinisialisasi (masing-masing dengan nilai 10.5 dan 0.5).

Array **operasiArray** didefinisikan, yang berisi objek dari kelas OperasiPenjumlahan, OperasiPengurangan, OperasiPerkalian, dan OperasiPembagian. Ini menciptakan polimorfisme, karena meskipun semua objek memiliki metode **tampil()**, implementasi masing-masing berbeda tergantung pada kelasnya.

**Instansi kelas PrintOperasi** digunakan untuk mencetak hasil operasi. Kelas ini (meskipun tidak diberikan dalam kode) tampaknya memiliki metode **cetakSemua()** yang

akan melakukan iterasi terhadap array operasiArray dan menjalankan metode tampil() pada masing-masing objek dalam array, dengan nilai a dan b yang diberikan.

### Penjelasan Polimorfisme Dinamis

**Polimorfisme dinamis** tercermin dalam penggunaan array **OperasiBilangan[]** yang menyimpan objek dari kelas OperasiPenjumlahan, OperasiPengurangan, OperasiPerkalian, dan OperasiPembagian.

Meskipun array tersebut hanya menyimpan objek dari superclass OperasiBilangan, ketika metode tampil() dipanggil, Java secara otomatis memanggil implementasi yang sesuai dari masing-masing subclass (berdasarkan objek yang sesungguhnya, bukan tipe variabel). Ini adalah contoh polimorfisme dinamis, di mana pemanggilan metode yang tepat ditentukan saat runtime berdasarkan tipe objek yang ada.

### Output

OPERASI ARTIMATIKA JAVA

PEMBAGIAN :  $10.5 / 0.5 = 21.0$

PENGURANGAN :  $10.5 - 0.5 = 10.0$

PERKALIAN :  $10.5 * 0.5 = 5.25$

PENJUMLAHAN :  $10.5 + 0.5 = 11.0$

## Operasi Bilangan 2

=> Abstract

```
package Pertemuan11.OperasiBilangan2.Abstract;

public abstract class OperasiBilanganAbs {
    protected double a, b, c;

    public abstract void set_A(double a);
    public abstract void set_B(double b);
    public abstract void set_C();
    public abstract double get_A();
    public abstract double get_B();
    public abstract double get_C();
    public abstract void tampil();
}
```

=> Final

```
package Pertemuan11.OperasiBilangan2.Final;

import Pertemuan11.OperasiBilangan2.Abstract.OperasiBilanganAbs;

public final class OperasiBilanganAbsCetak {

    public void cetakSemua(OperasiBilanganAbs[] OB, double a, double b) {
        for (OperasiBilanganAbs operasi : OB) {
            operasi.set_A(a);
            operasi.set_B(b);
            operasi.set_C();
            operasi.tampil();
        }
    }
}
```

=> Polimorfisme

```
import Pertemuan11.OperasiBilangan2.Abstract.OperasiBilanganAbs;

public class OperasiPembagian extends OperasiBilanganAbs {

    @Override
    public void set_A(double a) {
        this.a = a;
    }

    @Override
    public void set_B(double b) {
        this.b = b;
    }

    @Override
    public void set_C() {
        if (b != 0) {
            this.c = this.a / this.b;
        } else {
            System.out.println("TIDAK DAPAT MEMBAGI DENGAN NOL.");
            this.c = 0;
        }
    }

    @Override
    public double get_A() {
        return this.a;
    }

    @Override
    public double get_B() {
        return this.b;
    }

    @Override
    public double get_C() {
        return this.c;
    }

    @Override
    public void tampil() {
        System.out.println("HASIL PEMBAGIAN : " + this.get_C());
    }
}
```

```

import Pertemuan11.OperasiBilangan2.Abstract.OperasiBilanganAbs;

public class OperasiPengurangan extends OperasiBilanganAbs {

    @Override
    public void set_A(double a) {
        this.a = a;
    }

    @Override
    public void set_B(double b) {
        this.b = b;
    }

    @Override
    public void set_C() {
        this.c = this.a - this.b;
    }

    @Override
    public double get_A() {
        return this.a;
    }

    @Override
    public double get_B() {
        return this.b;
    }

    @Override
    public double get_C() {
        return this.c;
    }

    @Override
    public void tampil() {
        System.out.println("HASIL PENGURANGAN : " + this.get_C());
    }
}

```

```

import Pertemuan11.OperasiBilangan2.Abstract.OperasiBilanganAbs;

public class OperasiPenjumlahan extends OperasiBilanganAbs {

    @Override
    public void set_A(double a) {
        this.a = a;
    }

    @Override
    public void set_B(double b) {
        this.b = b;
    }

    @Override
    public void set_C() {
        this.c = this.a + this.b;
    }

    @Override
    public double get_A() {
        return this.a;
    }

    @Override
    public double get_B() {
        return this.b;
    }

    @Override
    public double get_C() {
        return this.c;
    }

    @Override
    public void tampil() {
        System.out.println("HASIL PENJUMLAHAN : " + this.get_C());
    }
}

```

```

import Pertemuan11.OperasiBilangan2.Abstract.OperasiBilanganAbs;

public class OperasiPerkalian extends OperasiBilanganAbs {

    @Override
    public void set_A(double a) {
        this.a = a;
    }

    @Override
    public void set_B(double b) {
        this.b = b;
    }

    @Override
    public void set_C() {
        this.c = this.a * this.b;
    }

    @Override
    public double get_A() {
        return this.a;
    }

    @Override
    public double get_B() {
        return this.b;
    }

    @Override
    public double get_C() {
        return this.c;
    }

    @Override
    public void tampil() {
        System.out.println("HASIL PERKALIAN : " + this.get_C());
    }
}

```

```

public class Main {
    Run | Debug
    public static void main(String[] args) {

        System.out.print("\033[H\033[2J");
        System.out.println("OPERASI ARITMATIKA JAVA\n");
        double A = 6.5, B = 0.5;

        // Array untuk polimorfisme
        OperasiBilanganAbs[] operasiBilangan = {
            new OperasiPenjumlahan(),
            new OperasiPengurangan(),
            new OperasiPerkalian(),
            new OperasiPembagian()
        };

        // Final class untuk mencetak
        OperasiBilanganAbsCetak cetak = new OperasiBilanganAbsCetak();
        cetak.cetakSemua(operasiBilangan, A, B);
    }
}

```

## **1. Kelas OperasiBilanganAbs**

### **Deskripsi:**

OperasiBilanganAbs adalah kelas abstrak yang mendefinisikan struktur dasar untuk operasi matematika. Kelas ini memiliki atribut a, b, dan c, yang mewakili dua angka yang akan dihitung (a dan b), serta hasil perhitungan (c). Kelas ini juga mendeklarasikan beberapa metode abstrak yang harus diimplementasikan oleh kelas turunan:

- set\_A(double a) untuk menetapkan nilai a.
  - set\_B(double b) untuk menetapkan nilai b.
  - set\_C() untuk menetapkan hasil perhitungan.
  - get\_A(), get\_B(), dan get\_C() untuk mendapatkan nilai a, b, dan hasilnya (c).
- tampil() untuk menampilkan hasil perhitungan.

## **2. Kelas OperasiPenjumlahan**

### **Deskripsi:**

OperasiPenjumlahan adalah kelas turunan dari OperasiBilanganAbs yang mengimplementasikan metode set\_A(), set\_B(), dan set\_C() untuk operasi penjumlahan.

- set\_A(double a) menetapkan nilai a.
- set\_B(double b) menetapkan nilai b.
- set\_C() menghitung hasil penjumlahan antara a dan b.

tampil() menampilkan hasil penjumlahan.

## **3. Kelas OperasiPengurangan**

### **Deskripsi:**

OperasiPengurangan adalah kelas turunan dari OperasiBilanganAbs yang mengimplementasikan metode set\_A(), set\_B(), dan set\_C() untuk operasi pengurangan.

- set\_A(double a) menetapkan nilai a.
- set\_B(double b) menetapkan nilai b.
- set\_C() menghitung hasil pengurangan antara a dan b.

tampil() menampilkan hasil pengurangan.

## **4. Kelas OperasiPerkalian**

### **Deskripsi:**

OperasiPerkalian adalah kelas turunan dari OperasiBilanganAbs yang mengimplementasikan metode set\_A(), set\_B(), dan set\_C() untuk operasi perkalian.

- set\_A(double a) menetapkan nilai a.

- set\_B(double b) menetapkan nilai b.
- set\_C() menghitung hasil perkalian antara a dan b.

tampil() menampilkan hasil perkalian.

## 5. Kelas OperasiPembagian

### Deskripsi:

OperasiPembagian adalah kelas turunan dari OperasiBilanganAbs yang mengimplementasikan metode set\_A(), set\_B(), dan set\_C() untuk operasi pembagian.

- set\_A(double a) menetapkan nilai a.
- set\_B(double b) menetapkan nilai b.
- set\_C() memeriksa apakah b tidak sama dengan nol, kemudian menghitung hasil pembagian antara a dan b. Jika b adalah nol, program akan mencetak pesan kesalahan dan hasilnya diset ke 0.

tampil() menampilkan hasil pembagian.

## 6. Kelas OperasiBilanganAbsCetak

### Deskripsi:

OperasiBilanganAbsCetak adalah kelas yang bertanggung jawab untuk mencetak hasil operasi.

- cetakSemua(OperasiBilanganAbs[] OB, double a, double b) menerima array objek dari kelas yang mengimplementasikan OperasiBilanganAbs dan dua nilai angka a dan b. Metode ini akan memanggil set\_A(), set\_B(), set\_C(), dan tampil() pada setiap objek untuk menghitung dan menampilkan hasil operasi.

## 7. Kelas Main

### Deskripsi:

Di dalam kelas Main, program memulai dengan mendeklarasikan dua variabel angka A dan B dengan nilai 6.5 dan 0.5. Kemudian, array objek dari kelas OperasiPenjumlahan, OperasiPengurangan, OperasiPerkalian, dan OperasiPembagian dibuat untuk melakukan operasi matematika menggunakan polimorfisme.

Kelas OperasiBilanganAbsCetak digunakan untuk mencetak hasil operasi melalui metode cetakSemua().

Program kemudian memanggil metode ini untuk menghitung dan menampilkan hasil operasi penjumlahan, pengurangan, perkalian, dan pembagian.

**Output:**

**HASIL PENJUMLAHAN:**

Penjumlahan  $6.5 + 0.5$  menghasilkan 7.0.

**HASIL PENGURANGAN:**

Pengurangan  $6.5 - 0.5$  menghasilkan 6.0.

**HASIL PERKALIAN:**

Perkalian  $6.5 * 0.5$  menghasilkan 3.25.

**HASIL PEMBAGIAN:**

Pembagian  $6.5 / 0.5$  menghasilkan 13.0. (Jika pembagi B adalah nol, maka akan menampilkan pesan kesalahan.)

# LAPORAN PRAKTIKUM 10

## 10.1 Penjelasan Kode Pertemuan 12

```
class Dosen {  
    private String nik;  
    private String nama;  
  
    public Dosen(String nik, String nama) {  
        this.nik = nik;  
        this.nama = nama;  
    }  
  
    public String getNik() {  
        return nik;  
    }  
  
    public String getNama() {  
        return nama;  
    }  
  
    public void view() {  
        System.out.println("NIK : " + nik);  
        System.out.println("NAMA : " + nama);  
    }  
}
```

```
class Rektor extends Dosen {  
    private int tahunMasuk;  
  
    public Rektor(String nik, String nama, int tahunMasuk) {  
        super(nik, nama);  
        this.tahunMasuk = tahunMasuk;  
    }  
  
    public int getTahunMasuk() {  
        return tahunMasuk;  
    }  
  
    public void viewRektor() {  
        view();  
        System.out.println("TAHUN MASUK : " + tahunMasuk);  
    }  
}
```

```

class Dekan extends Dosen {
    private String fakultas;

    public Dekan(String nik, String nama, String fakultas) {
        super(nik, nama);
        this.fakultas = fakultas;
    }

    public String getFakultas() {
        return fakultas;
    }

    public void viewDekan() {
        view();
        System.out.println("FAKULTAS : " + fakultas);
    }
}

```

```

public class Main {
    Run | Debug
    public static void main(String[] args) {

        System.out.print("\033[H\033[2J");
        // Object Dosen
        Dosen dosen = new Dosen(nik:"101", nama:"ARDITYA ADJIE ROSANDI");
        System.out.println("==|| DATA DOSEN ||==");
        dosen.view();
        System.out.println();

        // Object Rektor
        Rektor rektor = new Rektor(nik:"102", nama:"DHICKY HARYADI SUPRIYONO", tahunMasuk:2022);
        System.out.println("==|| DATA REKTOR ||==");
        rektor.viewRektor();
        System.out.println();

        // Object Dekan
        Dekan dekan = new Dekan(nik:"103", nama:"AVRIAN RIZKY MAULANA", fakultas:"ILMU KOMPUTER");
        System.out.println("==|| DATA DEKAN ||==");
        dekan.viewDekan();
    }
}

```

## Nomor 1: Program Inheritance

Kode ini menggunakan konsep **inheritance** (pewarisan) dalam pemrograman berorientasi objek (OOP) untuk mewariskan properti dan metode dari satu kelas ke kelas lainnya:

### 1. Class Dosen:

- Merupakan **superclass** yang memiliki atribut nik dan nama.
- Constructor digunakan untuk menginisialisasi nik dan nama.
- Metode getNik() dan getNama() mengembalikan nilai atribut.
- Metode view() digunakan untuk menampilkan informasi Dosen.

### 2. Class Rektor:

- Kelas ini mewarisi Dosen menggunakan **extends**.
- Menambahkan atribut baru tahunMasuk.

- Menambahkan metode getTahunMasuk() dan viewRektor() untuk menampilkan data tambahan.

### 3. Class Dekan:

- Juga mewarisi Dosen dan menambahkan atribut fakultas.
- Constructor menginisialisasi atribut tambahan fakultas.
- Metode getFakultas() dan viewDekan() untuk menampilkan informasi Dekan.

### Soal No 2

```
interface Transportasi {
    void tampil();
    void setData();
    int getId();
}
```

```
class Gojek implements Transportasi {
    protected int harga;
    protected int id;

    public Gojek() {
        this.harga = 0;
        this.id = 0;
    }

    @Override
    public void tampil() {
        System.out.println("GOJEK - HARGA : " + harga + ", ID: " + id);
    }

    @Override
    public void setData() {
        this.harga = 20000; // Contoh data
        this.id = 123;
    }

    @Override
    public int getId() {
        return id;
    }
}
```

```

class Bayar extends Gojek {
    private int jarak;
    private int total;
    private String nama;

    public Bayar() {
        super();
        this.jarak = 10; // Contoh jarak
        this.nama = "PENGGUNA GOJEK";
        this.total = jarak * 2000; // Perhitungan total biaya
    }

    public Bayar(int id) {
        this();
        this.id = id;
        System.out.println("BAYAR DENGAN ID : " + this.id);
    }

    @Override
    public void tampil() {
        System.out.println("NAMA : " + nama);
        System.out.println("JARAK : " + jarak + " KM");
        System.out.println("TOTAL BAYAR : RP." + total);
        System.out.println("ID GOJEK : " + id);
    }

    @Override
    public void setData() {
        super.setData();
        this.total = jarak * 2000;
        System.out.println("DATA BAYAR SUDAH DISET.");
    }

    @Override
    public int getId() {
        return id;
    }
}

```

```

public class Main {
    Run | Debug
    public static void main(String[] args) {

        System.out.print("\033[H\033[2J");

        System.out.println("==|| GOJEK ||==");
        Gojek gojek = new Gojek();
        gojek.setData();
        gojek.tampil();

        System.out.println("\n==|| BAYAR ||==");
        Bayar bayar = new Bayar(id:456);
        bayar.setData();
        bayar.tampil();
    }
}

```

## **Nomor 2: Program Inheritance dan Interface**

Kode ini menggabungkan konsep **inheritance** dan **interface** untuk memperluas fungsionalitas:

### **1. Interface Transportasi:**

- Merupakan kontrak yang berisi metode abstrak seperti tampil(), setData(), dan getId().
- Semua kelas yang mengimplementasi interface ini wajib mendefinisikan metode tersebut.

### **2. Class Gojek:**

- Merupakan **superclass** yang mengimplementasi Transportasi.
  - Menyediakan metode setData() untuk menginisialisasi atribut harga dan id.
  - Metode tampil() digunakan untuk menampilkan data harga dan id.
  - Metode getId() mengembalikan nilai id.

### **3. Subclass Bayar:**

- Mewarisi Gojek dan menambahkan atribut jarak, total, dan nama.
  - Constructor menghitung total biaya berdasarkan jarak.
- Metode setData() dan tampil() diperbarui untuk menampilkan informasi pembayaran.

### Soal No 3

```
interface Phone {
    int MAX_VOLUME = 100;
    int MIN_VOLUME = 0;

    void powerOn();
    void powerOff();
    void volumeUp();
    void volumeDown();
}

class Xiaomi implements Phone {
    private int volume;
    private boolean isPowerOn;

    public Xiaomi() {
        this.volume = 50; // Default volume
        this.isPowerOn = false;
    }

    @Override
    public void powerOn() {
        isPowerOn = true;
        System.out.println("XIAOMI IS ON.");
    }

    @Override
    public void powerOff() {
        isPowerOn = false;
        System.out.println("XIAOMI IS OFF.");
    }

    @Override
    public void volumeUp() {
        if (isPowerOn) {
            if (volume < MAX_VOLUME) {
                volume++;
                System.out.println("XIAOMI VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MAX.");
            }
        }
    }

    @Override
    public void volumeDown() {
        if (isPowerOn) {
            if (volume > MIN_VOLUME) {
                volume--;
                System.out.println("XIAOMI VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MIN.");
            }
        }
    }
}
```

```
class iPhone implements Phone {
    private int volume;
    private boolean isPowerOn;

    public iPhone() {
        this.volume = 50; // Default volume
        this.isPowerOn = false;
    }

    @Override
    public void powerOn() {
        isPowerOn = true;
        System.out.println("IPHONE IS ON.");
    }

    @Override
    public void powerOff() {
        isPowerOn = false;
        System.out.println("IPHONE IS OFF.");
    }

    @Override
    public void volumeUp() {
        if (isPowerOn) {
            if (volume < MAX_VOLUME) {
                volume++;
                System.out.println("IPHONE VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MAX.");
            }
        }
    }

    @Override
    public void volumeDown() {
        if (isPowerOn) {
            if (volume > MIN_VOLUME) {
                volume--;
                System.out.println("IPHONE VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MIN.");
            }
        }
    }
}
```

```
class Samsung implements Phone {
    private int volume;
    private boolean isPowerOn;

    public Samsung() {
        this.volume = 50; // Default volume
        this.isPowerOn = false;
    }

    @Override
    public void powerOn() {
        isPowerOn = true;
        System.out.println("SAMSUNG IS ON.");
    }

    @Override
    public void powerOff() {
        isPowerOn = false;
        System.out.println("SAMSUNG IS OFF.");
    }

    @Override
    public void volumeUp() {
        if (isPowerOn) {
            if (volume < MAX_VOLUME) {
                volume++;
                System.out.println("SAMSUNG VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MAX.");
            }
        }
    }

    @Override
    public void volumeDown() {
        if (isPowerOn) {
            if (volume > MIN_VOLUME) {
                volume--;
                System.out.println("SAMSUNG VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT .");
            }
        }
    }
}
```

```
class Oppo implements Phone {
    private int volume;
    private boolean isPowerOn;

    public Oppo() {
        this.volume = 50; // Default volume
        this.isPowerOn = false;
    }

    @Override
    public void powerOn() {
        isPowerOn = true;
        System.out.println("OPPO IS ON.");
    }

    @Override
    public void powerOff() {
        isPowerOn = false;
        System.out.println("OPPO IS OFF.");
    }

    @Override
    public void volumeUp() {
        if (isPowerOn) {
            if (volume < MAX_VOLUME) {
                volume++;
                System.out.println("OPPO VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MAX.");
            }
        }
    }

    @Override
    public void volumeDown() {
        if (isPowerOn) {
            if (volume > MIN_VOLUME) {
                volume--;
                System.out.println("OPPO VOLUME : " + volume);
            } else {
                System.out.println("VOLUME IS AT MIN.");
            }
        }
    }
}
```

```

class PhoneUser {
    private Phone phone;

    public PhoneUser(Phone phone) {
        this.phone = phone;
    }

    public void turnOnThePhone() {
        phone.powerOn();
    }

    public void turnOffThePhone() {
        phone.powerOff();
    }

    public void makePhoneLouder() {
        phone.volumeUp();
    }

    public void makePhoneSilent() {
        phone.volumeDown();
    }
}

```

```

public class Main {
    Run | Debug
    public static void main(String[] args) {
        System.out.print("\033[H\033[2J");

        Phone[] phones = {new Xiaomi(), new iPhone(), new Samsung(), new Oppo()};
        String[] brands = {"XIAOMI", "IPHONE", "SAMSUNG", "OPPO"};

        for (int i = 0; i < phones.length; i++) {
            System.out.println("\n==|| " + brands[i] + " ||==");
            PhoneUser user = new PhoneUser(phones[i]);

            user.turnOnThePhone();
            user.makePhoneLouder();
            user.makePhoneLouder();
            user.makePhoneSilent();
            user.turnOffThePhone();
        }
    }
}

```

### Nomer 3: Program Polimorfisme dan Overriding

Program ini menerapkan **konsep polimorfisme** dan **method overriding** dalam pemrograman berorientasi objek (OOP) menggunakan Java. Berikut penjelasannya:

### Konsep Polimorfisme

- Polimorfisme memungkinkan sebuah objek untuk memiliki **banyak bentuk**.
- Polimorfisme terjadi ketika sebuah metode dalam **kelas induk** di-**override** oleh **kelas anak**.
- Pemanggilan metode yang sama akan menghasilkan output yang berbeda bergantung pada objek yang digunakan.

#### 1. Superclass:

- Sebuah **kelas induk** mendefinisikan metode yang nantinya akan di-**override** oleh kelas turunannya.

#### 2. Subclass:

- Beberapa **kelas anak** yang mewarisi kelas induk dan mengimplementasikan ulang metode (overriding).

#### 3. Polimorfisme:

- Kelas utama (main) menggunakan **referensi kelas induk** untuk memanggil metode di kelas anak.
- Pemanggilan metode akan mengeksekusi versi metode yang **di-override** oleh kelas anak.

### Soal No 4

```
interface Movement {  
    void move();  
}  
  
interface Flying {  
    void fly();  
}
```

```
// Membuat class Animal
abstract class Animal {
    private String nama;
    private String sifat;
    private int ukuran;

    public Animal() {}

    public Animal(String nama, int ukuran) {
        this.nama = nama;
        this.ukuran = ukuran;
    }

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

    public String getNama() {
        return nama;
    }

    public void setUkuran(int ukuran) {
        this.ukuran = ukuran;
    }

    public int getUkuran() {
        return ukuran;
    }

    public void setSifat(String sifat) {
        this.sifat = sifat;
    }

    public String getSifat() {
        return sifat;
    }
}
```

```
class Mamalia extends Animal implements Movement {
    private String jalan;
    private String jenisMamalia;
    private boolean bisaJalan;
    private int jumlahKaki;

    public Mamalia() {}

    public Mamalia(String nama) {
        super.setNama(nama);
    }

    public void setJalan(String jalan) {
        this.jalan = jalan;
    }

    public String getJalan() {
        return jalan;
    }

    public void setBisaJalan(boolean bisaJalan) {
        this.bisaJalan = bisaJalan;
    }

    public boolean getBisaJalan() {
        return bisaJalan;
    }

    public void setJumlahKaki(int jumlahKaki) {
        this.jumlahKaki = jumlahKaki;
    }

    public int getJumlahKaki() {
        return jumlahKaki;
    }

    public void setJenisMamalia(String jenisMamalia) {
        this.jenisMamalia = jenisMamalia;
    }

    public String getJenisMamalia() {
        return jenisMamalia;
    }

    @Override
    public void move() {
        System.out.println("MAMALIA BERJALAN DENGAN " + jumlahKaki + " KAKI DI " + jalan + ".");
    }
}
```

```
class Aves extends Animal implements Flying {
    private String jenisAves;
    private boolean bisaTerbang;

    public Aves() {}

    public Aves(String nama, int ukuran) {
        super(nama, ukuran);
    }

    public void setBisaTerbang(boolean bisaTerbang) {
        this.bisaTerbang = bisaTerbang;
    }

    public boolean getBisaTerbang() {
        return bisaTerbang;
    }

    public void setJenisAves(String jenisAves) {
        this.jenisAves = jenisAves;
    }

    public String getJenisAves() {
        return jenisAves;
    }

    @Override
    public void fly() {
        if (bisaTerbang) {
            System.out.println("AVES INI BISA TERBANG.");
        } else {
            System.out.println("AVES INI TIDAK BISA TERBANG.");
        }
    }
}
```

```
class Ayam extends Aves {
    private String jenisAyam;
    private boolean bisaDiadu;

    public Ayam() {}

    public Ayam(String nama, int ukuran) {
        super(nama, ukuran);
    }

    public void setJenisAyam(String jenisAyam) {
        this.jenisAyam = jenisAyam;
    }

    public String getJenisAyam() {
        return jenisAyam;
    }

    public void setBisaDiadu(boolean bisaDiadu) {
        this.bisaDiadu = bisaDiadu;
    }

    public boolean getBisaDiadu() {
        return bisaDiadu;
    }

    @Override
    public final void fly() {
        System.out.println("AYAM BIASANYA TIDAK BISA TERBANG JAUH.");
    }
}
```

```
class Merpati extends Aves {
    public Merpati() {}

    public Merpati(String nama, int ukuran) {
        super(nama, ukuran);
    }

    @Override
    public void fly() {
        System.out.println("MERPATI TERBANG TINGGI DAN JAUH.");
    }
}
```

```

public class Main {
    Run | Debug
    public static void main(String[] args) {
        System.out.print("\033[H\033[2J");

        // Mamalia Object
        Mamalia mamalia = new Mamalia(nama:"SINGA");
        mamalia.setJumlahKaki(jumlahKaki:4);
        mamalia.setBisaJalan(bisaJalan:true);
        mamalia.setJalan(jalan:"PADANG RUMPUT");
        System.out.println("NAMA MAMALIA : " + mamalia.getNama());
        mamalia.move();

        // Ayam Object
        Ayam ayam = new Ayam(nama:"AYAM JAGO", ukuran:3);
        ayam.setJenisAyam(jenisAyam:"AYAM ADUAN");
        ayam.setBisaDiadu(bisaDiadu:true);
        System.out.println("\nNAMA AVES : " + ayam.getNama());
        System.out.println("JENIS AVES : " + ayam.getJenisAyam());
        ayam.fly();

        // Merpati Object
        Merpati merpati = new Merpati(nama:"MERPATI POS", ukuran:2);
        System.out.println("\nNAMA AVES : " + merpati.getNama());
        merpati.fly();
    }
}

```

#### Nomer 4: Program Encapsulation dengan Konstruktor

Program ini menerapkan **encapsulation** menggunakan **getter dan setter**, serta menambahkan penggunaan **konstruktor**.

##### Konsep Encapsulation:

- Encapsulation adalah proses **menyembunyikan data** (atribut) di dalam kelas agar tidak dapat diakses langsung dari luar kelas.
- Data diakses dan dimodifikasi melalui **getter** dan **setter**.

##### 1. Kelas Utama:

- Kelas ini berisi atribut-atribut yang dibuat dengan **akses modifier private**.
- Atribut hanya bisa diakses melalui **getter** dan **setter**.

##### 2. Konstruktor:

- Konstruktor digunakan untuk menginisialisasi nilai atribut pada saat objek dibuat.
- Konstruktor dapat berupa **parameterized constructor** (dengan parameter) atau **default constructor**.

##### 3. Keuntungan Encapsulation:

- **Keamanan:** Data tidak dapat diakses atau dimodifikasi secara langsung.

- **Fleksibilitas:** Mengontrol bagaimana data diakses atau diubah melalui metode getter dan setter.
- **Kemudahan Pemeliharaan:** Jika terjadi perubahan atribut, hanya metode setter dan getter yang perlu diperbarui.

# LAPORAN PRAKTIKUM 11

## 11.1 Penjelasan Kode Pertemuan 13

### Linked List

```
package Pertemuan13.Tugas;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class TugasLinkedList {
    Run|Debug
    public static void main(String[] args) {
        // Membuat dua objek list
        List<String> warna = new ArrayList<>();
        List<String> warnaDihapus = new LinkedList<>();

        // Mengisi list warna
        warna.add("MAGENTA");
        warna.add("RED");
        warna.add("WHITE");
        warna.add("BLUE");
        warna.add("CYAN");

        // Mengisi warnaDihapus dengan beberapa elemen yang sama
        warnaDihapus.add("RED");
        warnaDihapus.add("WHITE");

        // Hapus data dari warna yang terdapat pada warnaDihapus
        warna.removeAll(warnaDihapus);

        // Tampilkan hasil
        System.out.println("Warna: " + warna);
    }
}
```

Penggunaan koleksi ArrayList dan LinkedList di Java untuk mengelola data dalam bentuk list. Pada awalnya, dua list dibuat, yaitu warna menggunakan ArrayList untuk menyimpan data warna dan warnaDihapus menggunakan LinkedList untuk menyimpan daftar warna yang akan dihapus. List warna diisi dengan lima elemen warna: "MAGENTA", "RED", "WHITE", "BLUE", dan "CYAN", sedangkan list warnaDihapus diisi dengan warna "RED" dan "WHITE", yang juga terdapat dalam list warna. Selanjutnya, metode removeAll dipanggil pada list warna dengan argumen warnaDihapus. Metode ini secara otomatis menghapus elemen-elemen dalam warna yang juga terdapat pada warnaDihapus. Akibatnya, elemen "RED" dan "WHITE" dihapus dari list warna. Terakhir, elemen yang tersisa dalam list warna, yaitu "MAGENTA", "BLUE", dan "CYAN", ditampilkan melalui perintah System.out.println. Kode ini menunjukkan cara manipulasi data dalam list menggunakan operasi penghapusan berbasis elemen tertentu.

```

package Pertemuan13.Tugas;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

class Mahasiswa implements Comparable<Mahasiswa> {
    private String nrp;
    private String nama;
    private int nilai;

    public Mahasiswa(String nrp, String nama, int nilai) {
        this.nrp = nrp;
        this.nama = nama;
        this.nilai = nilai;
    }

    public int getNilai() {
        return nilai;
    }

    @Override
    public String toString() {
        return "NRP: " + nrp + ", Nama: " + nama + ", Nilai: " + nilai;
    }

    @Override
    public int compareTo(Mahasiswa m) {
        return Integer.compare(this.nilai, m.nilai);
    }
}

```

```

public class TugasPengurutanMahasiswa {
    Run | Debug
    public static void main(String[] args) {
        List<Mahasiswa> mahasiswaList = new ArrayList<>();
        Random rand = new Random();

        // Menambahkan 10 data mahasiswa
        mahasiswaList.add(new Mahasiswa(nrp:"202308001", nama:"Alice", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308002", nama:"Bob", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308003", nama:"Charlie", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308004", nama:"Diana", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308005", nama:"Eve", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308006", nama:"Frank", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308007", nama:"Grace", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308008", nama:"Hank", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308009", nama:"Ivy", rand.nextInt(41) + 60));
        mahasiswaList.add(new Mahasiswa(nrp:"202308010", nama:"Jack", rand.nextInt(41) + 60));

        // Tampilkan data sebelum diurutkan
        System.out.println("Data Mahasiswa sebelum diurutkan:");
        for (Mahasiswa m : mahasiswaList) {
            System.out.println(m);
        }

        // Urutkan data berdasarkan nilai
        Collections.sort(mahasiswaList);

        // Tampilkan data setelah diurutkan
        System.out.println("\nData Mahasiswa setelah diurutkan berdasarkan nilai:");
        for (Mahasiswa m : mahasiswaList) {
            System.out.println(m);
        }
    }
}

```

Program ini memanfaatkan kelas Mahasiswa, yang merepresentasikan informasi mahasiswa dengan atribut nrp (nomor registrasi mahasiswa), nama, dan nilai. Kelas ini mengimplementasikan antarmuka Comparable untuk memungkinkan pengurutan data berdasarkan atribut nilai. Metode compareTo pada kelas Mahasiswa membandingkan objek mahasiswa berdasarkan nilai mereka dengan menggunakan Integer.compare. Selain itu, metode toString diimplementasikan untuk menampilkan informasi mahasiswa dalam format yang mudah dibaca. Dalam kelas utama

TugasPengurutanMahasiswa, sebuah list mahasiswaList dideklarasikan menggunakan ArrayList dan diisi dengan sepuluh objek Mahasiswa. Setiap mahasiswa diberikan nilai acak antara 60 hingga 100 menggunakan kelas Random. Setelah itu, program menampilkan data mahasiswa sebelum diurutkan. Selanjutnya, metode Collections.sort digunakan untuk mengurutkan list berdasarkan nilai mahasiswa secara ascending, memanfaatkan implementasi compareTo. Akhirnya, data mahasiswa setelah diurutkan ditampilkan kembali. Program ini menunjukkan bagaimana cara menggunakan koleksi dan antarmuka Comparable untuk melakukan operasi pengurutan pada data.

### **Ucapan Terima Kasih**

Saya mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas terselesaiannya laporan praktikum ini. Terima kasih kepada dosen pembimbing, asisten laboratorium, serta semua pihak yang telah memberikan bimbingan dan dukungan. Kami berharap laporan ini dapat bermanfaat dan terbuka untuk kritik serta saran yang membangun.