

# JobSheet - Week 1 - Fundamental Programming Structures in Java

**Nama: Iyantona Ariesta Putra**

**NIM: 251524106**

**Kelas: 1D**

**Repo GitHub: <https://github.com/Ikanapa/TP>**

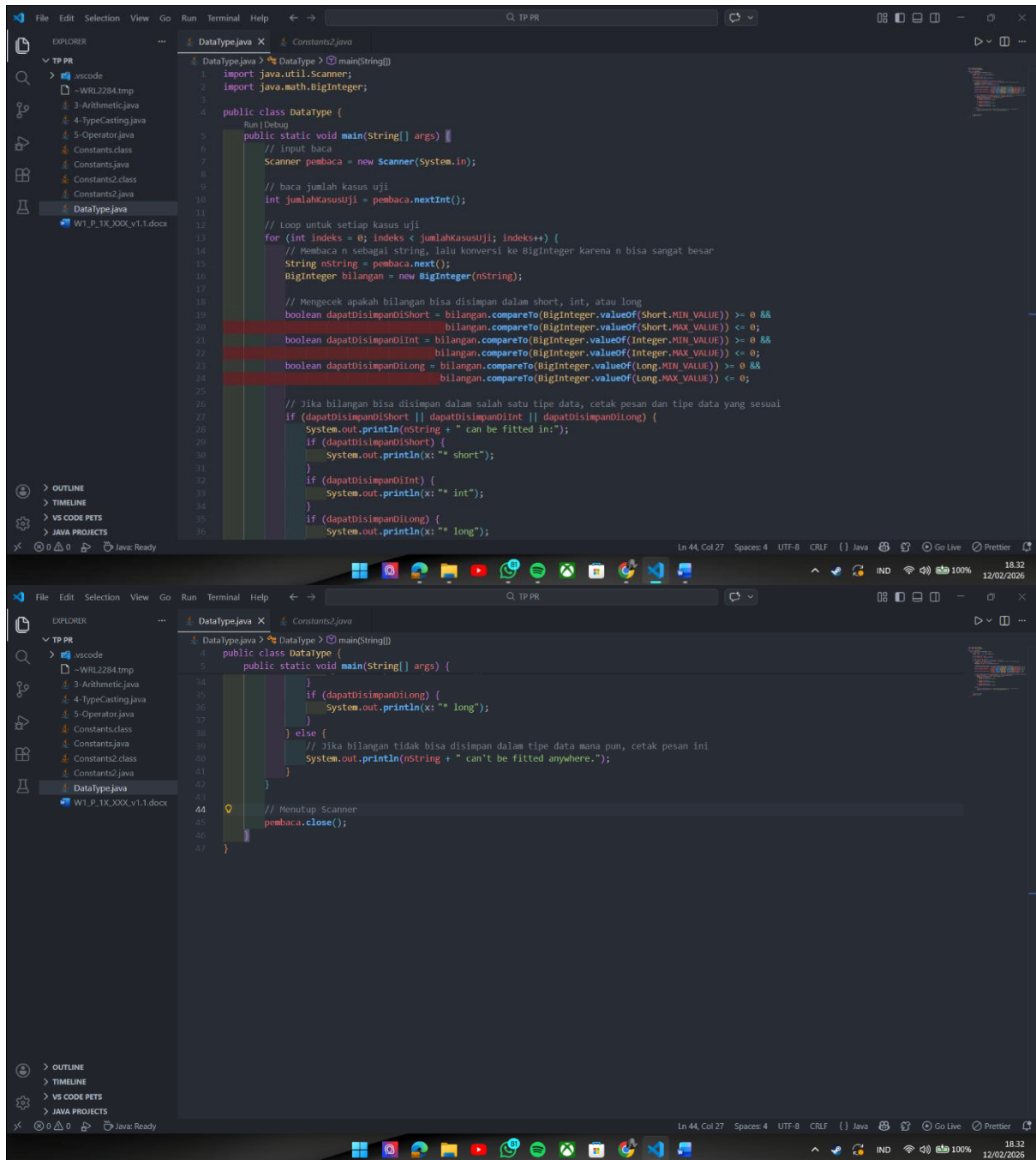
## Instruksi Pengerjaan:

1. Kerjakan 5 soal di bawah ini dengan melengkapi setiap kolom jawaban yang disediakan pada jobsheet ini.
2. Jawaban setiap soal mencakup source code, screenshot hasil dari program yang ditampilkan full screen termasuk taskbar (tambahkan beberapa screenshot jika diperlukan), penjelasan permasalahan dan solusi yang dihadapi, nama teman yang membantu memecahkan masalah (opsional).
3. Dikumpulkan pada Assignment Classroom sesuai dengan deadline yang tertera pada assignment tersebut.
4. Format penamaan file jobsheet: W1\_P\_<Kelas 1X>\_<3 Digit\_NIM\_Terakhir>.docx/pdf. Contoh: W1\_P\_1B\_001.docx/pdf.
5. Buatlah satu file java yang mengandung jawaban dalam bentuk source untuk satu jawaban yang dapat langsung dieksekusi. Contoh penamaan: 1-DataTypes.java, 2-Variables.java, 3-Arithmetic.java, 4-TypeCasting.java, dan 5-Operator.java.
6. Submit semua jawaban dalam bentuk file java pada repository GitHub masing-masing.

## No. 1 Data Types

Soal Praktikum
<p>Java memiliki 8 tipe data primitif; char, boolean, byte, short, int, long, float, dan double.</p> <p>Untuk praktikum ini, kita akan Latihan dengan tipe data primitif yang digunakan untuk menyimpan nilai bilangan bulat, yaitu byte, short, int, dan long.</p> <ul style="list-style-type: none"><li>• A byte is an 8-bit signed integer.</li><li>• A short is a 16-bit signed integer.</li><li>• An int is a 32-bit signed integer.</li><li>• A long is a 64-bit signed integer.</li></ul> <p>Dengan diberikan sebuah bilangan bulat masukan, Anda harus menentukan tipe data primitif mana yang mampu menyimpan masukan tersebut dengan benar.</p>





## Penjelasan Permasalahan dan Solusi

### PERMASALAHAN

Bilangan 'n' bisa lebih besar dari 'Long.MAX\_VALUE' atau lebih kecil dari 'Long.MIN\_VALUE', sehingga kita tidak bisa menggunakan tipe data primitif langsung. Kita perlu menggunakan `BigInteger` untuk menangani nilai besar.

### SOLUSI

1. Membaca Input: Gunakan `Scanner` untuk membaca T terlebih dahulu. Kemudian, untuk setiap kasus uji, baca n sebagai string (karena bisa sangat besar), lalu konversi ke `BigInteger` untuk perbandingan yang aman.
2. Pengecekan Range: Bandingkan n dengan batas minimum dan maksimum dari masing-masing tipe data

## Nama Teman Hal yang Dibantu (Opsional)

## No. 2 Variables

### Soal Praktikum

Perhatikan dua bagian program di bawah ini.

#### Bagian 1:

```
public class Constants {  
    public static void main(String[] args) {  
        final double CM_PER_INCH = 2.54; double paperWidth = 8.5;  
        double paperHeight = 11;  
        System.out.println("Paper size in centimeters: " + paperWidth  
* CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);  
    }  
}
```

#### Bagian 2:

```
public class Constants2 {  
    public static final double CM_PER_INCH = 2.54; public static void  
main(String[] args) {  
        double paperWidth = 8.5; double paperHeight = 11;  
        System.out.println("Paper size in centimeters: " + paperWidth  
* CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);  
    }  
}
```

Dari 2 contoh baris program diatas, jawablah pertanyaan dibawah ini:

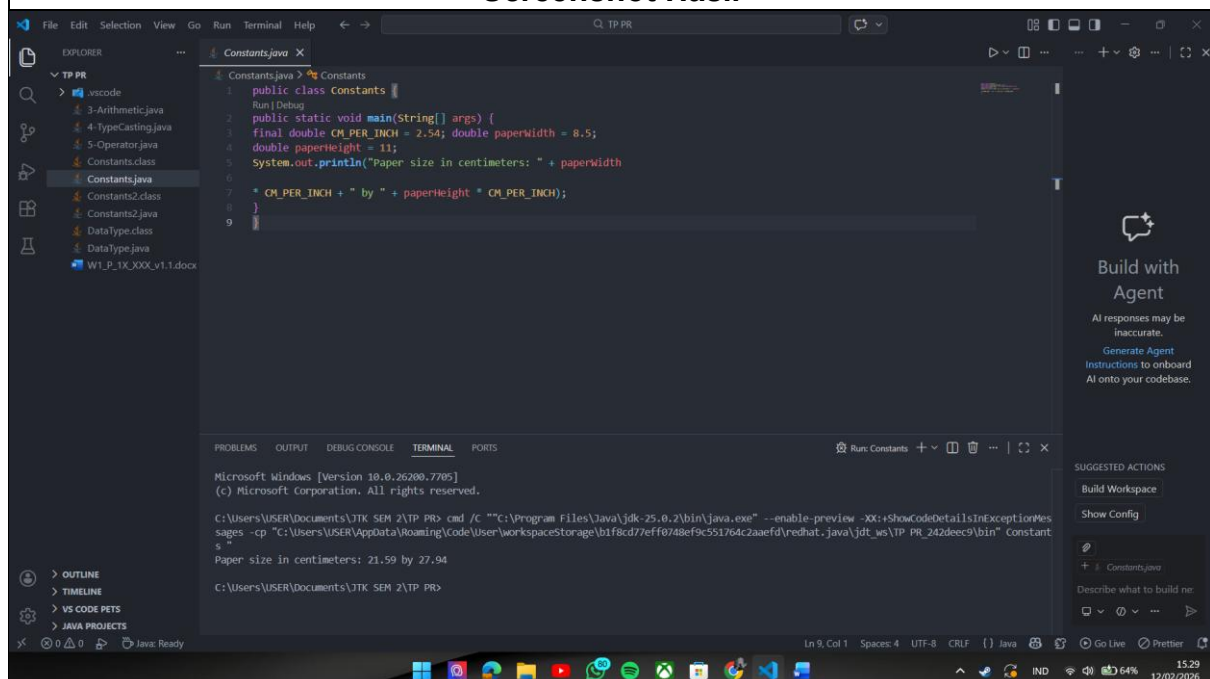
1. Bagaimana output dari masing masing class Constants dan Constants2?
2. Apa perbedaan penggunaan final double dengan public static final double?

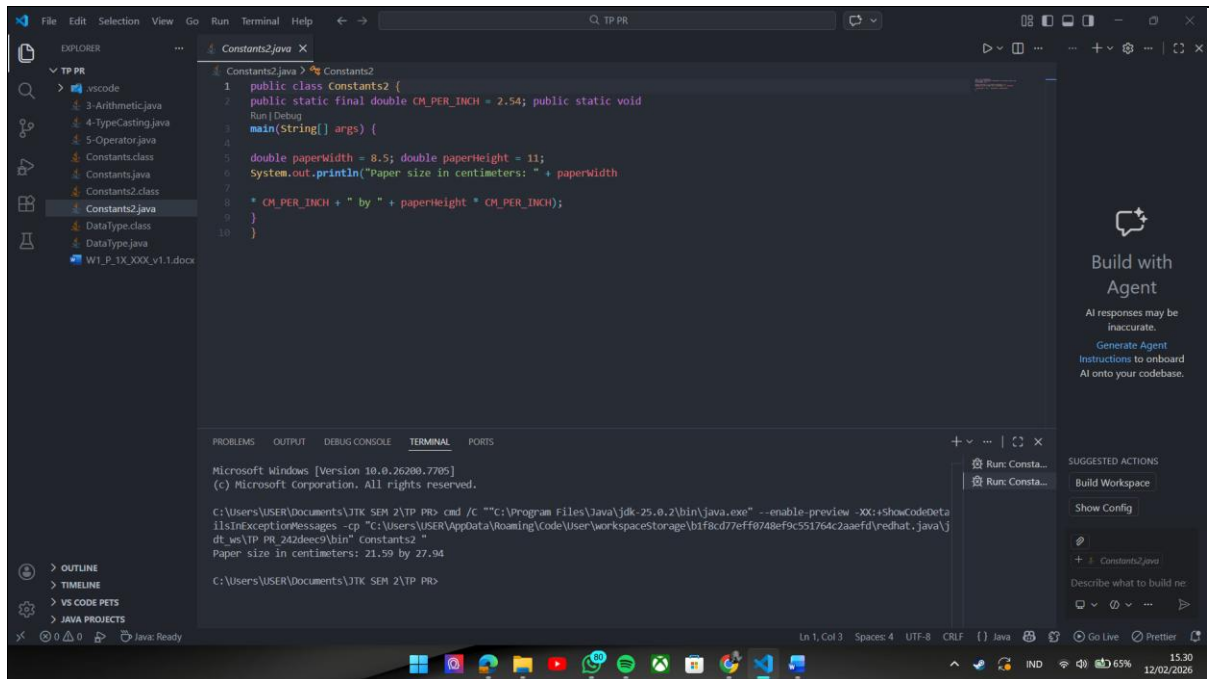
### Source Code

[TP/Constants2.java at master · Ikanapa/TP](#)

[TP/Constants.java at master · Ikanapa/TP](#)

### Screenshot Hasil





## Penjelasan Permasalahan dan Solusi

### 1. Output dari masing masing class :

- Class Constants outputnya adalah paper size in centimeters: 21.59 by 27.94
- Class Constants2 outputnya adalah paper size in centimeters: 21.59 by 27.94

Penjelasan : Kode program “Constants” menghitung lebar kertas (8.5 inci  $\times$  2.54 cm/inch = 21.59 cm) dan tinggi (11 inci  $\times$  2.54 cm/inch = 27.94 cm), lalu mencetak hasilnya.

### 2. Perbedaan penggunaan final double dengan public static final double :

- Final double : Mendeklarasikan variabel lokal yang tidak bisa diubah setelah inisialisasi dan tidak bisa diakses dari luar metode atau kelas lain.
- Public static final double : Mendeklarasikan konstanta kelas yang bersifat publik yang bisa diakses tanpa membuat objek kelas (statis) dan final, namun bisa diakses dari mana saja dalam program.

## PERMASALAHAN

Pada kode program Constants, ‘CM\_PER\_INCH’ dideklarasikan sebagai variabel lokal di dalam metode ‘main’. Yang berarti ini tidak bisa digunakan diluar metode tsb, apabila kode diperluas dan misal ingin menambahkan metode lain dalam kelas atau kelas lain. Kita harus mendeklarasikan ulang, yang menyebabkan duplikasi kode yang tidak efektif.

Pada kedua program juga memiliki masalah umum, yakni tidak adanya validasi input atau penanganan error yang bisa menjadi masalah jika nilai seperti ‘paperWidth’ atau ‘paperHeight’ berasal dari input pengguna.

## SOLUSI

1. Gunakan public static final untuk konstanta global
2. Buat validasi

## Nama Teman dan Hal yang Dibantu (Opsional)

## No. 3 Arithmetic - Math Class

### Soal Praktikum

Perhatikan bagian program di bawah ini.

```
Class FloatingPoint {
    public static void main(String[] args) {
        double x = 92.98;
        int nx = (int) Math.round(x);
    }
}
```

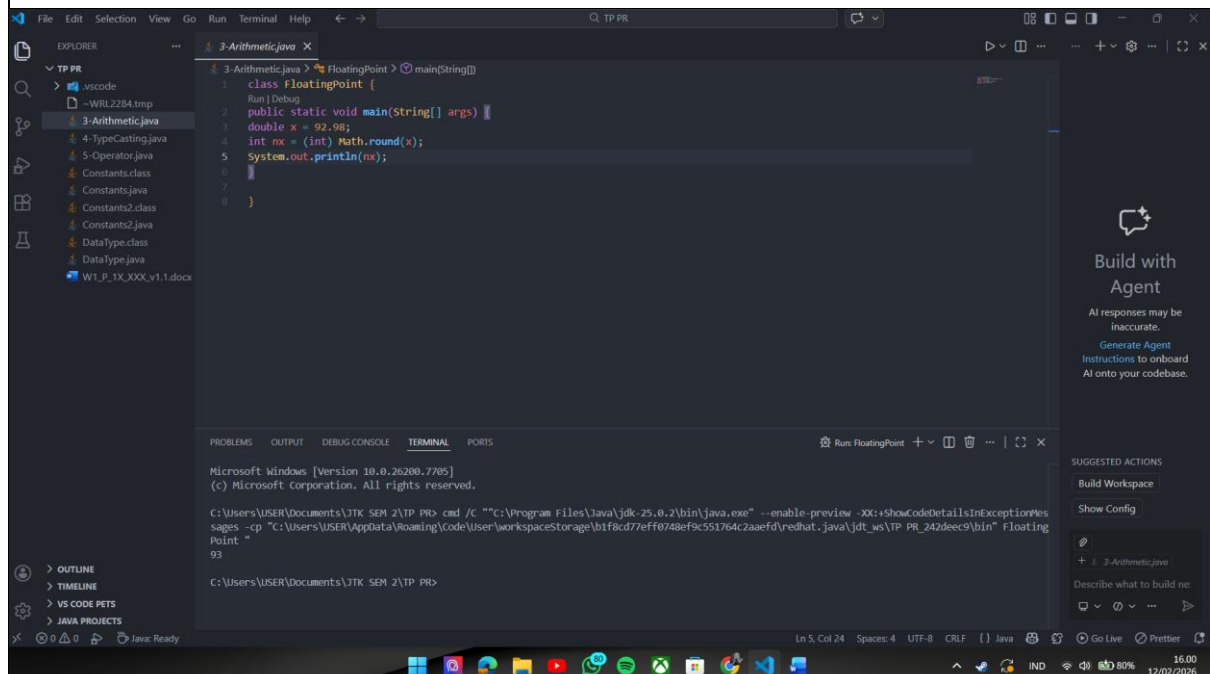
}  
**Math** Class berisi bermacam-macam fungsi matematika seperti pada contoh diatas pada penggunaan round(x), terdapat beberapa pertanyaan yang perlu untuk dijelaskan:

1. Pada kasus berikut jelaskan nilai nx setelah digunakan **Math.round(x)**!
2. Kenapa dibutuhkan cast (int) dalam penggunaan **Math.round(x)**?

### Source Code

[TP/3-Arithmetic.java at master · Ikanapa/TP](#)

### Screenshot Hasil



### Penjelasan Permasalahan dan Solusi

1. Nilai nx setelah digunakan Math.round(x) :
  - Dalam kode 'double x = 92.98; int nx = (int) Math.round(x);', nilai nx adalah '93'  
Penjelasan : metode 'Math.round(double x)' membulatkan nilai 'x' ke bilangan bulat terdekat. Karena x=92.98, bagian desimalnya 0,98 sehingga dibulatkan ke atas menjadi 93.0 (dalam bentuk 'long').  
Kemudian, cast '(int)' mengubahnya menjadi 'int' tanpa kehilangan nilai.
2. Kenapa dibutuhkan cast (int) dalam penggunaan Math.round (x) :  
Karena metode 'Math.round(double x)' mengembalikan nilai bertipe 'long', bukan 'int'. Dan Cast '(int)' dibutuhkan untuk mengubah 'long' menjadi 'int', karena variabel 'nx' dideklarasikan sebagai 'int'. Tanpa cast, akan terjadi error kompilasi karena ketidakcocokan tipe data. Namun, cast ini aman hanya jika nilai 'long' berada dalam rentang 'int'.

#### PERMASALAHAN

1. Risiko overflow apabila x sangat besar yang bisa menghasilkan long yang melebihi rentang 'int'
2. Tidak ada validasi

#### SOLUSI

1. Periksa rentang sebelum cast atau pastikan nilai long dari Math.round() dalam rentang int untuk menghindari overflow

### Nama Teman dan Hal yang Dibantu (Opsional)

## No. 4 Type Casting/ Data Type Conversion

### Soal Praktikum

Perhatikan baris program dibawah ini:

```
class ConvertDataType {
    static short methodOne(long l) {
        int i = (int) l; return (short)i;
    }

    public static void main(String[] args) {
        double d = 10.25; float f = (float) d;
        byte b = (byte) methodOne((long) f); System.out.println(b);
    }
}
```

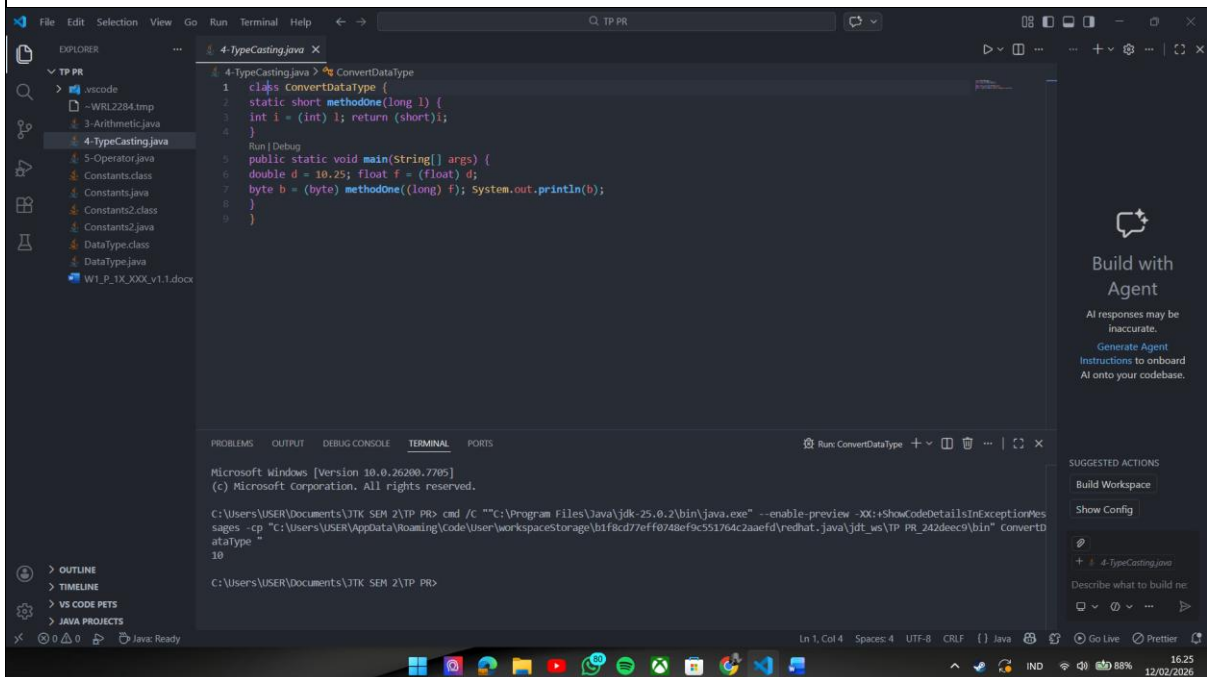
Program berikut melakukan convert tipe data yang berukuran besar ke kecil (long -> int -> short) dan (double -> float -> byte).

1. Jelaskan output nilai dari variable b.
2. Jelaskan apa yang berubah dari variable d menjadi variable b setelah dilakukan cast?

### Source Code

[TP/4-TypeCasting.java at master · lkanapa/TP](#)

### Screenshot Hasil



### Penjelasan Permasalahan dan Solusi

1. Output nilai variabel b :

Nilai berubah dari 10.25 (desimal) menjadi 10 (integer bulat) karena casting dari float ke long memotong bagian desimal

Nilai awal : 'd = 10.25'

Konversi ke float : 'f = (float) d' jadi 'f' lalu menjadi '10.25f'

Konversi ke long : '(long) f' jadi Casting 'float' ke 'long' memotong bagian desimal, sehingga menjadi '10L'

Panggilan methodOne(10L) :

'int i = (int) 10L' jadi 'i = 10'

'return (short) i' jadi 'short' (nilai 10 masih dalam rentang, jadi kembali 10)

Konversi ke byte : 'byte b = (byte) 10' jadi byte (nilai 10 masih valid , jadi b = 10 )

2. Perubahan dari variable d ke variable b :

Nilai berubah dari 10.25 (desimal) menjadi 10 (integer bulat) karena casting dari float ke long memotong bagian desimal.

#### PERMASALAHAN

1. Resiko overflow : Jika nilai 'd' lebih besar (misalnya 1000.25), casting ke byte bisa menghasilkan nilai negatif atau salah
2. Tidak ada pengecekan



## SOLUSI

1. Pengecekan rentang sebelum casting

**Nama Teman dan Hal yang Dibantu (Opsional)**

## No. 5 Operator

### Soal Praktikum

Perhatikan bagian program di bawah ini.

```
class OperatorChallenge {
    public static void main(String[] args) {
        int a = 5;
        int b = 10;

        boolean result = (++a * 2 > b) && (b++ % 3 == 1);

        System.out.println("Hasil Boolean: " + result);
        System.out.println("Nilai a: " + a);
        System.out.println("Nilai b: " + b);
    }
}
```

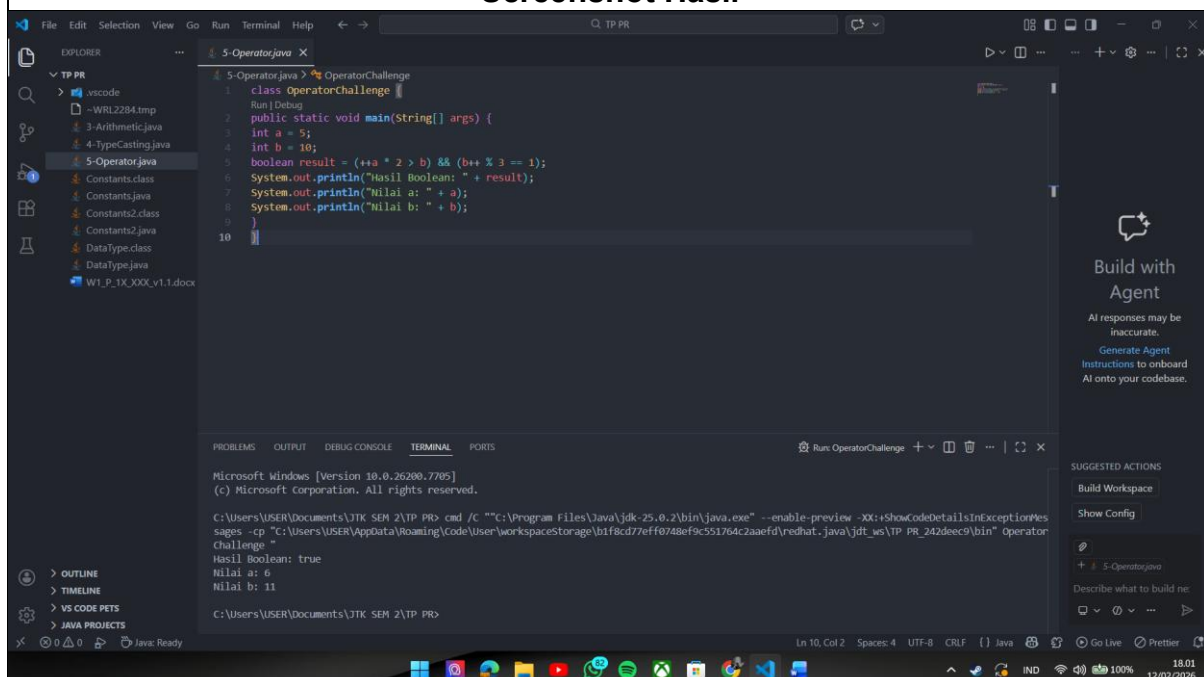
### Pertanyaan Analisis:

1. **Analisis Langkah Demi Langkah:** Jelaskan urutan eksekusi pada baris `boolean result`. Mana yang dijalankan lebih dulu antara `++a` dan perkalian `*`?
2. **Short-Circuit Logic:** Jika bagian pertama `(++a * 2 > b)` bernilai `false`, apakah bagian kedua `(b++ % 3 == 1)` akan tetap dieksekusi oleh Java? Jelaskan dampaknya pada nilai akhir variabel `b`.
3. **Output:** Berapakah nilai akhir dari `result`, `a`, dan `b`?

### Source Code

[TP/5-Operator.java at master · Ikanapa/TP](#)

### Screenshot Hasil



### Penjelasan Permasalahan dan Solusi



1. Urutan eksekusi boolean result :

Inisialisasi awal:  $a = 5$ ,  $b = 10$

Bagian kiri:  $(++a * 2 > b)$  :Nilai  $a$  ditingkatkan menjadi 6 lalu digunakan dalam ekspresi. Setelah  $++a$ , dilakukan perkalian  $6 * 2 = 12$ . Perbandingan  $12 > 10 \rightarrow \text{true}$

Bagian kanan ( $b++ \% 3 == 1$ ) (dieksekusi karena bagian kiri true : Nilai  $b$  (10) digunakan terlebih dahulu dalam ekspresi, lalu  $b$  ditingkatkan menjadi 11. Modulo  $10 \% 3 = 1$ . Perbandingan  $1 == 1 = \text{true}$

Hasil ekspresi :  $\text{true} \ \&\& \ \text{true} = \text{true}$ .

Dalam bagian kiri,  $++a$  dieksekusi lebih dulu (karena pre-increment memodifikasi nilai sebelum digunakan), diikuti oleh perkalian, lalu perbandingan.

2. Short-Circuit Logic: Dampak pada Bagian Kedua

Apakah bagian kedua dieksekusi jika bagian pertama false? Tidak. Operator ' $\&\&$ ' menggunakan short-circuit evaluation: Jika bagian kiri  $(++a * 2 > b)$  bernilai false, Java langsung menghentikan evaluasi dan menetapkan result = false tanpa mengeksekusi bagian kanan ( $b++ \% 3 == 1$ ). Dampaknya, Jika bagian kiri false (misalnya, jika  $a$  awalnya lebih kecil sehingga  $++a * 2$  tidak  $> b$ ), maka  $b++$  tidak dieksekusi, sehingga  $b$  tetap 10 (tidak bertambah menjadi 11). Ini berarti nilai akhir  $b$  bergantung pada hasil bagian kiri

3. Output: Nilai Akhir dari result,  $a$ , dan  $b$

- result: true
- $a$ : 6
- $b$ : 11

**Nama Teman dan Hal yang Dibantu (Opsional)**