Week 1 - Fundamental Programming Structures in Java

Nama : Alia Ardani
NIM : 251524035
Kelas : 1B
Repo GitHub : https://github.com/vssixla/Teknik_Pemrograman_2026

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

Java memiliki 8 tipe data primitif; char, boolean, byte, short, int, long, float, dan double.

Untuk praktikum ini, kita akan Latihan dengan tipe data primitif yang digunakan untuk menyimpan nilai bilangan bulat, yaitu byte, short, int, dan long.

- A byte is an 8-bit signed integer.
- A short is a 16-bit signed integer.
- An int is a 32-bit signed integer.
- A long is a 64-bit signed integer.

Dengan diberikan sebuah bilangan bulat masukan, Anda harus menentukan tipe data primitif mana yang mampu menyimpan masukan tersebut dengan benar.

Input Format

Baris pertama berisi bilangan bulat, T, yang menunjukkan jumlah kasus uji. Setiap kasus uji, T, terdiri dari satu baris dengan bilangan bulat, n, yang nilainya bisa sangat besar atau sangat kecil.

Output Format

Untuk setiap variabel masukan n dan tipe data primitif yang sesuai, Anda harus menentukan apakah tipe data primitif yang diberikan mampu menyimpannya. Jika ya, maka cetak:

```
N can be fitted in:
* datatype
```

Jika terdapat lebih dari satu tipe data yang sesuai, cetak masing-masing pada barisnya sendiri dan urutkan berdasarkan ukurannya (misalnya: **byte < short < int < long**).
Jika angka tersebut tidak dapat disimpan dalam salah satu dari empat tipe data primitif yang disebutkan di atas, cetak baris berikut:

```
N can’t be fitted anywhere
```

Sample Input:

```
5
-150
150000
1500000000
21333333333333333333333333333333333
-1000
```

Sample Output:

```
-150 can be fitted in:
  *  short
  *  int
  *  long
150000 can be fitted in:
  *  int
  *  long
1500000000 can be fitted in:
  *  int
  *  long
```

```
213333333333333333333333333333333333 can't be fitted anywhere.
```

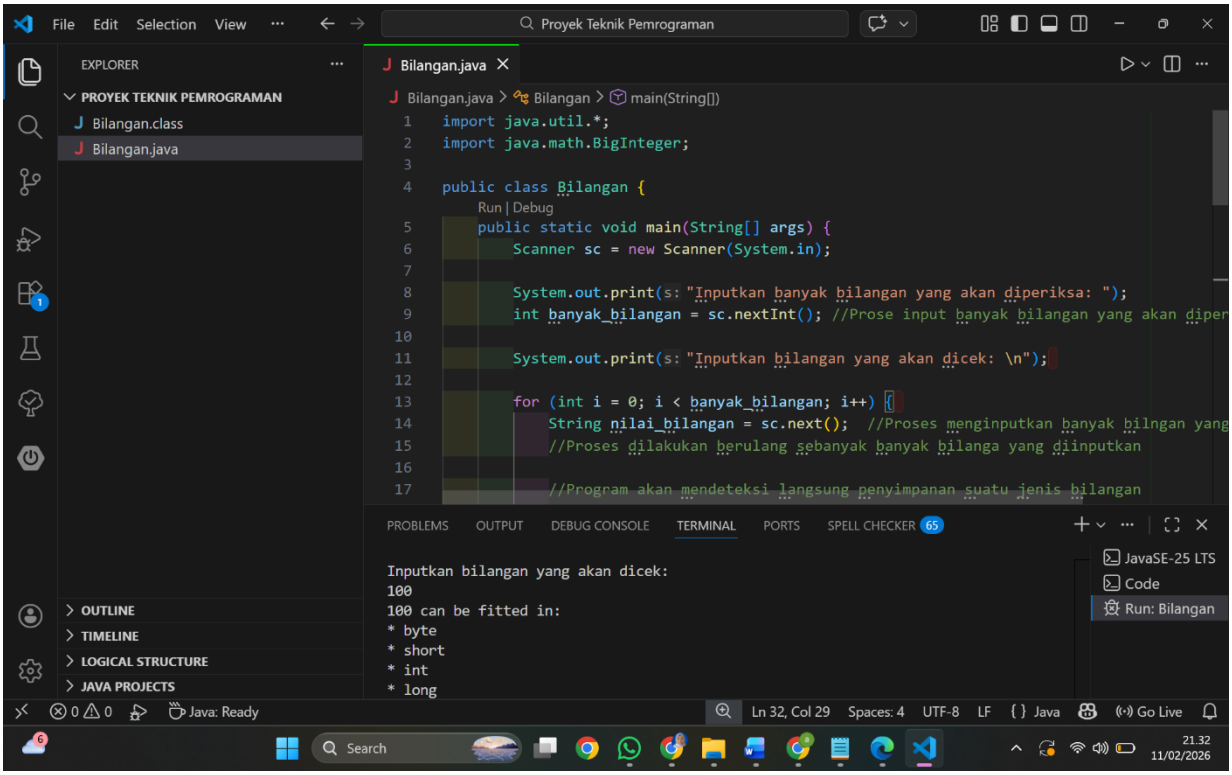
Explanation:
Angka 150 dapat disimpan dalam tipe data short, int, atau long. Angka 213333333333333333333333333333333333 sangat besar dan berada di luar rentang nilai yang diizinkan untuk tipe data primitif yang dibahas dalam masalah ini.

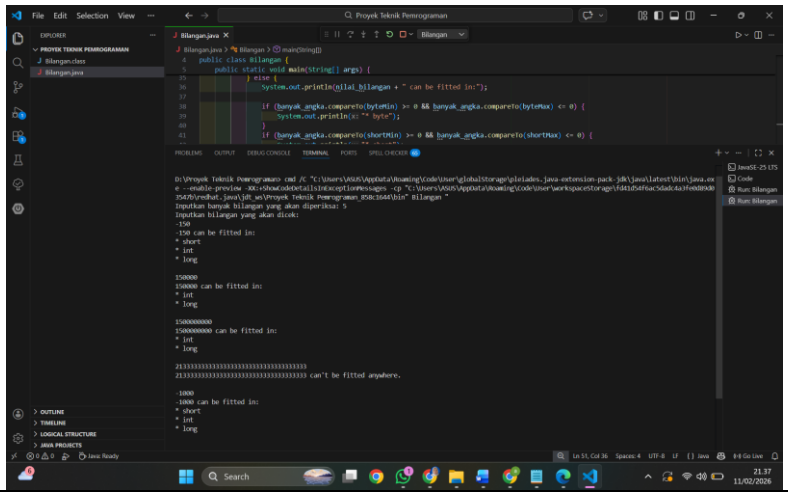
Source Code

```
import java.util.*;
import java.math.BigInteger;

public class Bilangan {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Inputkan banyak bilangan yang akan diperiksa: ");
    int banyak_bilangan = sc.nextInt(); //Prose input banyak bilangan yang akan diperiksa

    System.out.print("Inputkan bilangan yang akan dicek: \n");

    for (int i = 0; i < banyak_bilangan; i++) {
      String nilai_bilangan = sc.next();  //Proses menginputkan banyak bilngan yang akan dicek
      //Proses dilakukan berulang sebanyak banyak bilanga yang diinputkan

      //Program akan mendeteksi langsung penyimpanan suatu jenis bilangan
      BigInteger banyak_angka = new BigInteger(nilai_bilangan);

      BigInteger byteMin = BigInteger.valueOf(Byte.MIN_VALUE);
      BigInteger byteMax = BigInteger.valueOf(Byte.MAX_VALUE);

      BigInteger shortMin = BigInteger.valueOf(Short.MIN_VALUE);
      BigInteger shortMax = BigInteger.valueOf(Short.MAX_VALUE);

      BigInteger intMin = BigInteger.valueOf(Integer.MIN_VALUE);
      BigInteger intMax = BigInteger.valueOf(Integer.MAX_VALUE);

      BigInteger longMin = BigInteger.valueOf(Long.MIN_VALUE);
      BigInteger longMax = BigInteger.valueOf(Long.MAX_VALUE);

      //Proses pemeriksaan bilangan sekaligus output yang akan keluar
      if (banyak_angka.compareTo(longMin) < 0 || banyak_angka.compareTo(longMax) > 0) {
        System.out.println(nilai_bilangan + " can't be fitted anywhere.");
      } else {
        System.out.println(nilai_bilangan + " can be fitted in:");

        if (banyak_angka.compareTo(byteMin) >= 0 && banyak_angka.compareTo(byteMax) <= 0) {
          System.out.println("* byte");
        }
        if (banyak_angka.compareTo(shortMin) >= 0 && banyak_angka.compareTo(shortMax) <= 0) {
```

```
            System.out.println("* short");
        }
        if (banyak_angka.compareTo(intMin) >= 0 && banyak_angka.compareTo(intMax) <= 0) {
            System.out.println("* int");
        }
        if (banyak_angka.compareTo(longMin) >= 0 && banyak_angka.compareTo(longMax) <= 0) {
            System.out.println("* long");
        }
      }
      System.out.print("\n");
    }
    sc.close();
  }
}
```

Screenshot Hasil





Penjelasan Permasalahan dan Solusi

Analisis Permasalahan

Dalam bahasa pemrograman Java, terdapat delapan tipe data *primitive* yang salah satunya digunakan untuk menyimpan objek bilangan bulat, *yaitu byte, short, int, dan long*. Setiap tipe data ini memiliki kapasitas penyimpanan yang berbeda, yang ditentukan oleh jumlah bit yang digunakan.

Diketahui:

- *byte* menggunakan 8 bit sehingga dapat menyimpan bilangan dari -128 hingga 127
- *short* menggunakan 16 bit dengan rentang -32.768 hingga 32.767
- *int* menggunakan 32 bit dengan rentang -2.147.483.648 hingga 2.147.483.647
- *long* menggunakan 64 bit dengan rentang -9.223.372.036.854.775.808 hingga 9.223.372.036.854.775.807

Pemasalahan ini muncul ketika akan menginputkan suatu bilangan bulat, yang mana nilainya bisa sangat besar ataupun sangat kecil, sehingga tidak semua tipe data mampu menyimpan nilai angka tersebut. Jika nilai angka yang diberikan melebihi kapasitas tipe data yang digunakan, maka akan terjadi *overflow* atau program gagal memproses angka dengan benar.

Analisis Solusi

Untuk mengatasi masalah ini, diperlukan sebuah mekanisme untuk menentukan tipe data primitif terkecil yang dapat menampung angka input dengan tepat, atau menentukan jika angka tersebut berada di luar kapasitas semua tipe data yang tersedia.

Solusi untuk permasalahan ini dapat dilakukan dengan membaca setiap angka masukan sebagai *string* terlebih dahulu, kemudian dikonversi menjadi *BigInteger* (memproses angka yang sangat besar tanpa terjadi *overflow*). Jika angka tersebut masuk ke dalam rentang sebuah tipe data, maka tipe data tersebut dicatat sebagai tipe yang dapat menyimpan angka tersebut. Proses ini diulang untuk semua tipe data agar dapat menampilkan semua tipe yang mampu menampung angka tersebut, diurutkan dari ukuran terkecil hingga terbesar. Apabila angka masukan tidak termasuk dalam rentang tipe data manapun, maka program akan menampilkan pesan bahwa angka tersebut "*can't be fitted anywhere*".

Dengan pendekatan ini, program mampu menangani berbagai kasus angka, baik yang kecil maupun yang sangat besar, dan memberikan output yang sesuai dengan format yang diminta.

Nama Teman Hal yang Dibantu (Opsional)

Kemal Ardian – Bantu memberikan gambaran dalam repositori

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

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);
  }
}
```

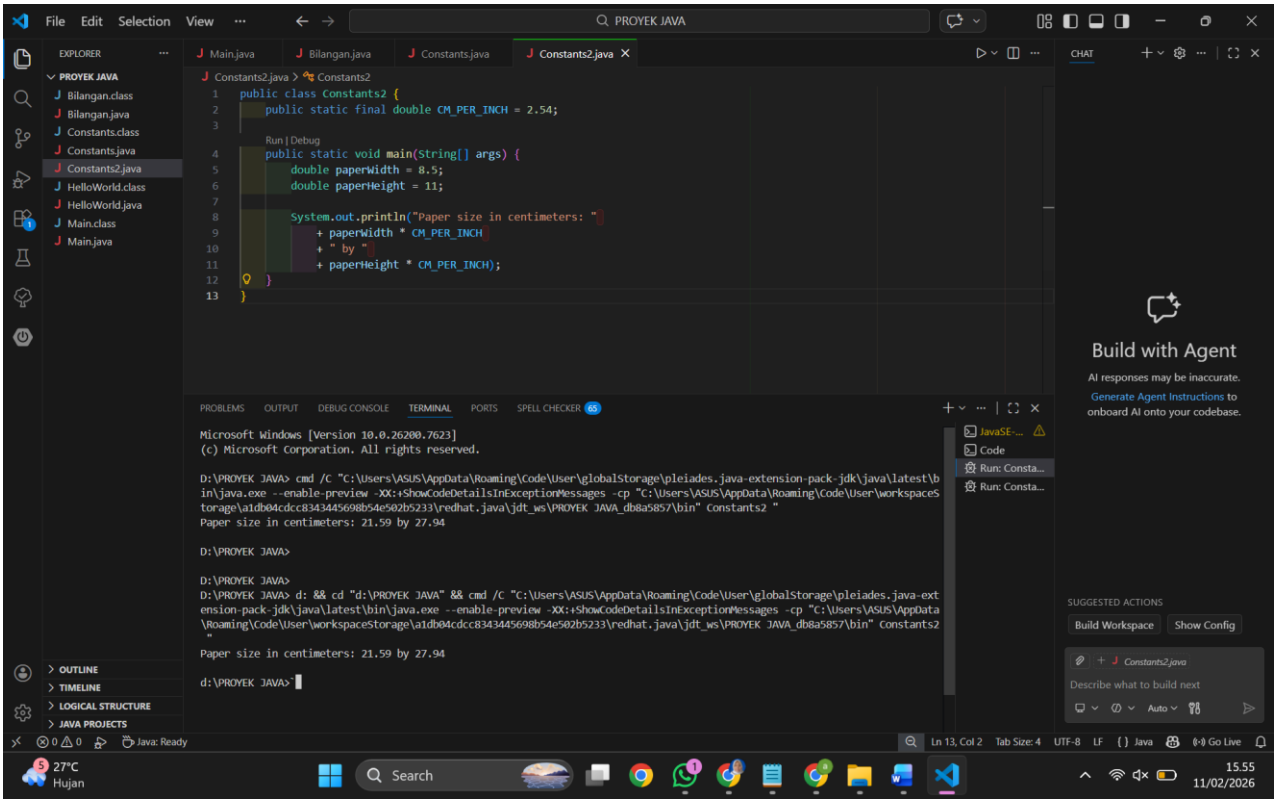
Bagian 2 :

```
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);
    }
}
```

Screenshot Hasil

Constants.java (Paragraph Pertama)
Constants2.java (Paragraph Kedua)



Penjelasan Permasalahan dan Solusi

Analisis Permasalahan

Permasalahan dalam soal ini berfokus pada penggunaan konstanta dalam program Bahasa Java. Pada soal menampilkan dua potongan program yang menghasilkan *output* yang sama, yaitu "Paper size in centimeters: 21.59 by 27.94", tetapi menggunakan penyusunan kode yang berbeda. Hal ini tentu akan menimbulkan kebingungan.

Kedua program melakukan operasi perhitungan yang sama, yaitu mengubah ukuran kertas dari *inci* ke *centimeter* menggunakan konstanta. Karena nilai konstanta dan rumus yang digunakan sama, hal ini yang membuat kedua program tersebut menghasilkan output yang sama. Kedua program tersebut, memiliki perbedaan cara dalam mendeklarasikan konstanta.

Pada program bagian pertama (Constants) menggunakan *final double* di dalam metode main. Hal ini membuat konstanta hanya dapat digunakan dalam lingkup method tersebut, dan jika

program dikembangkan menjadi lebih besar maka konstanta tersebut tidak dapat langsung digunakan.

Sedangkan pada program bagian kedua (Constants2) konstanta dideklarasikan menggunakan *public static final double*. Hal ini menunjukkan bahwa konstanta tersebut bersifat milik class, dapat diakses tanpa membuat objek, dan dapat digunakan oleh class lain.

Jawaban Pertanyaan

1. Bagaimana output dari masing masing class Constants dan Constants2?
   Output dari kedua program tersebut sama, yaitu Paper size in centimeters: 21.59 by 27.94

2. Apa perbedaan penggunaan final double dengan public static final double?

Perbedaan penggunaan final double dan public static final double terdapat dalam lingkup akses, kepemilikan variabel, serta cara penggunaannya dalam program. Pada *final double*, variabel digunakan sebagai konstanta yang nilainya tidak dapat diubah setelah diinisialisasi. Namun, jika dideklarasikan di dalam *method*, konstanta tersebut hanya berlaku pada *method* tersebut saja. Artinya, konstanta tidak bisa digunakan di method lain maupun di *class* lain.

Sedangkan *public static final double* digunakan untuk membuat konstanta yang bersifat global pada *level class*. *Keyword* public memungkinkan konstanta diakses oleh *class* lain, *static* menandakan bahwa konstanta tersebut milik class dan tidak perlu membuat objek untuk menggunakannya, serta final memastikan nilainya tidak dapat diubah. Dengan demikian, konstanta yang dideklarasikan menggunakan *public static fina*l dapat digunakan di berbagai bagian program secara konsisten.

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);
      }
}
```

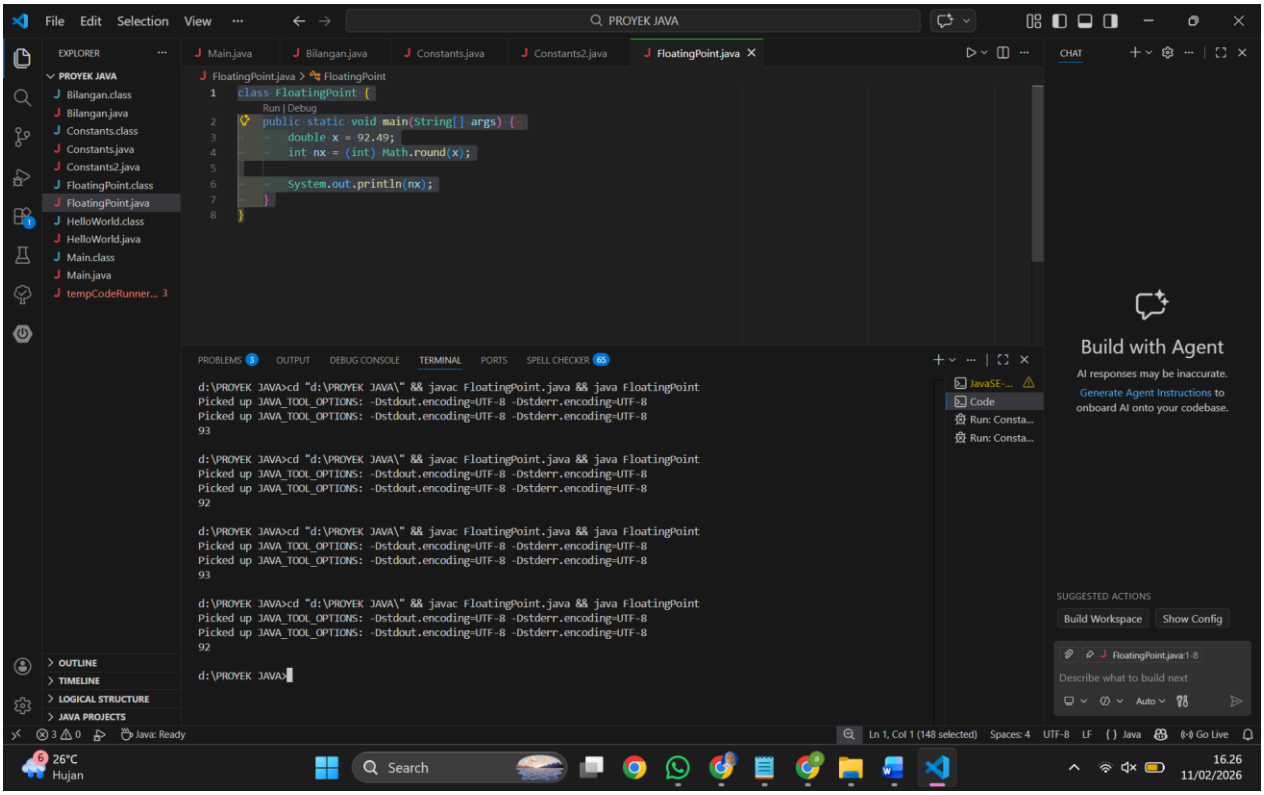
Math Class berisi bermacam-macam fungsi matematika seperti pada contoh diatas pada penggunaan round(x), terdapat beberapa pertanyaan yang perlu untuk dijelaskan:

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

Source Code

```
class FloatingPoint {
  public static void main(String[] args) {
    double x = 92.49;
    int nx = (int) Math.round(x);

    System.out.println(nx);  //Menambahkan sendiri untuk mengetahui hasilnya
  }
}
```

Screenshot Hasil



Penjelasan Permasalahan dan Solusi

Analisis Permasalahan

Pada program diatas menggunakan fungsi matematika, yaitu Math.round(), dan ketika program tersebut di *run*, program akan membulatkan sebuah bilangan.

Jawaban Pertanyaan

1. Pada kasus berikut jelaskan nilai nx setelah digunakan **Math.round(x)**!

Dalam soal, permasalahannya berkaitan dengan pembulatan nilai x, x memiliki nilai 92.98 yang merupakan tipe bilangan *double*. Ketika fungsi Math.round(x) dijalankan, nilai tersebut akan dibulatkan ke bilangan bulat terdekat, karena nilai desimal 0.98 lebih besar dari 0.5, maka nilai akan dibulatkan ke atas. Oleh karena itu, hasil dari Math.round(92.98) adalah 93, nilai ini kemudian disimpan ke dalam variabel nx, sehingga nilai akhir nx adalah 93.

2. Kenapa dibutuhkan cast (int) dalam penggunaan **Math.round(x)**?

Fungsi Math.round() memiliki tipe pengembalian yang berbeda tergantung pada tipe data inputnya. Jika sebuah input bertipe *double*, maka hasil Math.round() akan bertipe *long*. Sementara itu, variabel nx pada program bertipe int. Karena Java tidak secara otomatis mengubah tipe data long menjadi *int* (berpotensi menyebabkan kehilangan data), maka diperlukan *casting* secara manual menggunakan *(int)*. *Casting* ini bertujuan untuk mengubah hasil pembulatan dari tipe *long* menjadi *int* agar sesuai dengan tipe data variabel nx.

Dalam program ini saya mencoba mengubah objek dari variable x menjadi bilangan yang memiliki *decimal* berbeda:

- Ketika x saya ubah menjadi 92.23, nilai x akan membulat ke bawah menjadi 92.
- Ketika x saya ubah menjadi 92.50, nilai x akan membulat ke atas menjadi 93
- Ketika x saya ubah menjadi 92.49, nilai x akan membulat ke atas menjadi 93

Hal ini sesuai dengan aturan pembulatan matematika, jika suatu bilangan *decimal* kurang dari 0.5 maka bilangan tersebut akan membulat kebawah dan jika suatu bilangan desimal lebih atau sama dengan 0.5 makan akan membulat ke atas.

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);
	}
}
```

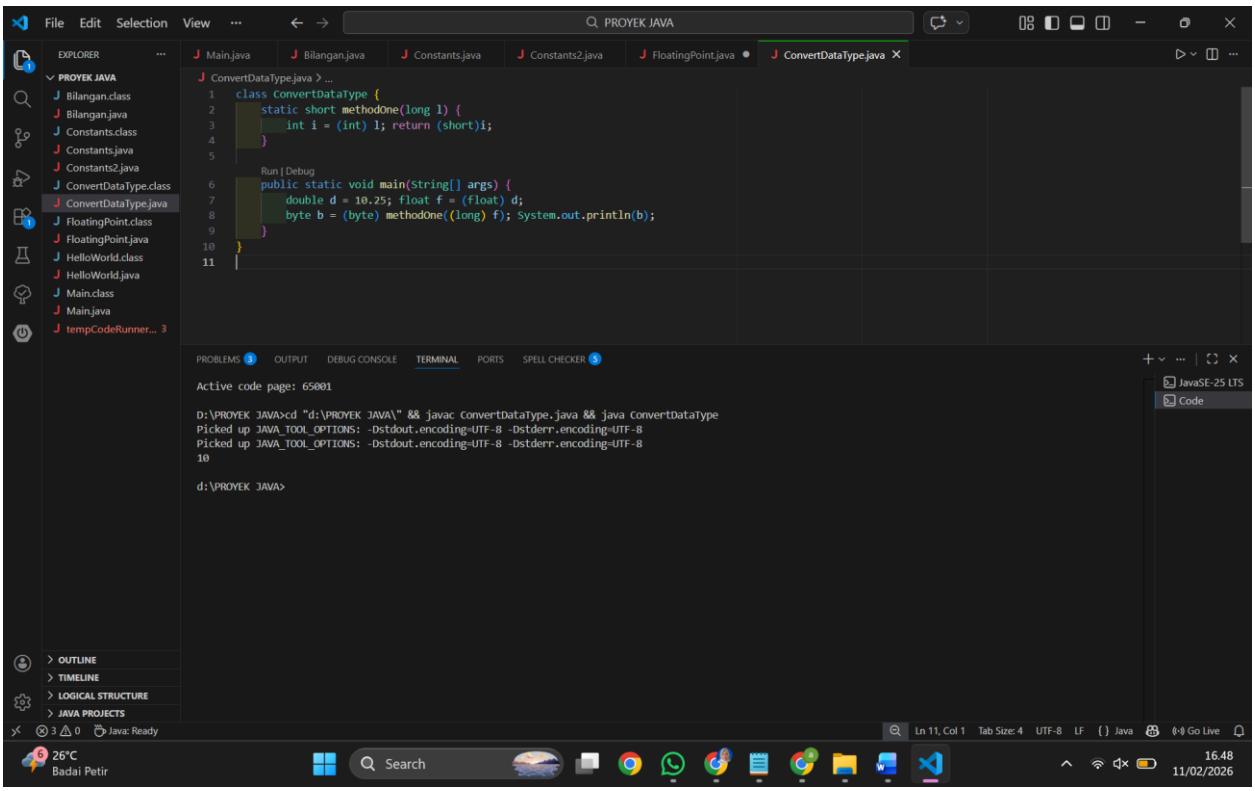
Program berikut melakukan convert tipe data yang berukuran besar ke kecil (long -> int -> short) dan (double -> float -> byte).

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

Source Code

```
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);
  }
}
```

Screenshot Hasil



Penjelasan Permasalahan dan Solusi

Analisis Permasalahan

Pada program diatas merupakan program yang melakukan konversi tipe data dari yang berukuran besar (*double*) menjadi ukuran yang lebih kecil (*byte*). Program ini menggunakan *casting eksplisit* yang memaksa perubahan pada tipe dayang berbeda ukuran. Dari program diatas, dapat dilihat nilai awal dari variable d berubah dan disimpan di dalam variable b.

Jawaban Pertanyaan

1. Jelaskan output nilai dari variable b.
   Output dari nilai variable b adalah 10.

   Pada program ini, nilai awal variabel d adalah 10.25 dengan tipe data double. Nilai ini kemudian melalui beberapa tahap konversi tipe data, dimulai dari :
   - Perubahan dalam nilai d menjadi nilai yang disimpan f
     Awalnya nilai d sebesar 10.25 di cast menjadi *float* dan disimpan ke variable f. Dalam proses ini, nilai f tetap 10.25 karena masih berada dalam rentang representasi *float*.
   - Perubahan dalam nilai f menjadi nilai yang dikirim ke method (long)
     Nilai f berubah menjadi tipe data long saat dipanggil ke method methodOne((long) f). Pada proses casting dari float ke long, bagian angka di belakang koma akan dihilangkan (dipotong, bukan dibulatkan). Oleh karena itu, nilai 10.25 akan berubah menjadi 10.
   - Perubahan nilai long menjadi int di dalam method
     Di dalam methodOne, nilai bertipe long tersebut kemudian diubah menjadi tipe data int menggunakan (int) l. Karena nilai 10 masih berada dalam rentang tipe int, maka nilainya tidak berubah dan tetap menjadi 10.
   - Perubahan nilai int menjadi short
     Nilai int tersebut diubah kembali menjadi tipe data short. Sama seperti sebelumnya, karena nilai 10 masih berada dalam rentang tipe short, maka nilai tetap 10.
   - Perubahan nilai short menjadi byte
     Nilai hasil dari method kemudian di-cast lagi menjadi tipe data byte saat disimpan ke variabel b. Karena tipe data byte memiliki rentang −128 sampai 127, nilai 10 masih aman sehingga tidak berubah.

2. Jelaskan apa yang berubah dari variable d menjadi variable b setelah dilakukan cast?
   Perubahan yang terjadi dari variabel d ke variabel b adalah hilangnya bagian desimal dan penyempitan rentang data (karena berubahnya jenis bilangan). Variabel d bertipe double memiliki kemampuan menyimpan angka desimal dengan presisi tinggi. Namun, selama proses casting bertahap, terutama saat program mengonversi dari float ke long, bagian desimal **10.25** dihilangkan menjadi **10**.

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);
    }
}
```

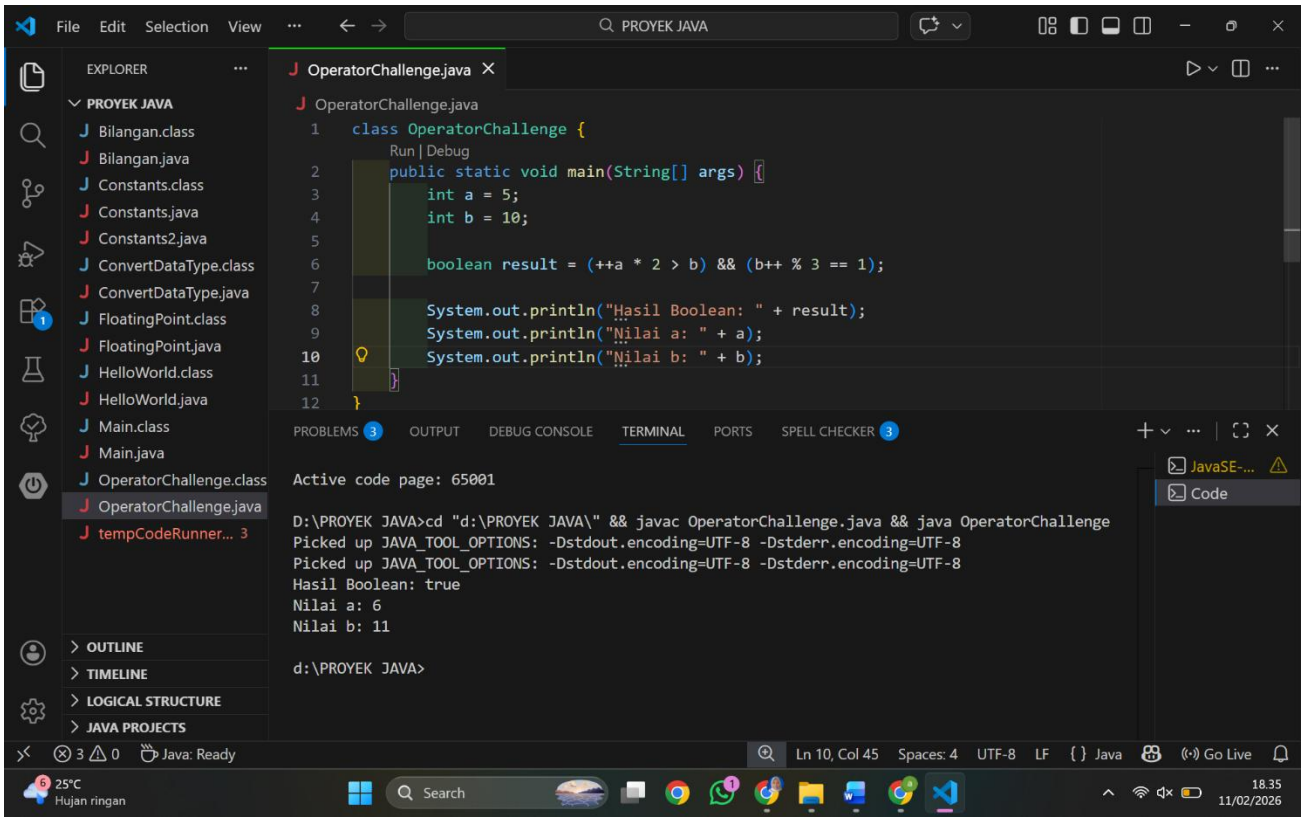
Pertanyaan Analisis:

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

```
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);
  }
}
```

Screenshot Hasil



Penjelasan Permasalahan dan Solusi

Analisis Permasalahan

Program ini merupakan program yang menguji penggunaan operator *increment*, operator aritmatika, operator perbandingan, dan operator logika dalam satu permasalahan. Program ini akan menunjukkan bagaimana urutan prioritas operator mempengaruhi proses perhitungan serta bagaimana *short-circuit* pada operator logika dapat mempengaruhi perubahan nilai suatu variabel.

Jawaban Pertanyaan

1. Analisis Langkah Demi Langkah: Jelaskan urutan eksekusi pada baris `boolean result`. Mana yang dijalankan lebih dulu antara `++a` dan perkalian `*`?

Pada baris keenam yang merupakan operator logika, urutan eksekusi ditentukan oleh prioritas operator (operator *precedence*) di Java. Dalam program ini, yang terjadi pertama adalah ++a, bukan perkalian. Hal ini karena operator increment prefix (++) memiliki kelas lebih tinggi dibanding operator perkalian (*). *Prefix increment* bekerja dengan cara menaikkan nilai variabel terlebih dahulu baru digunakan dalam perhitungan.

- Awalnya nilai a = 5.
- Saat ++a dijalankan, a menjadi 6.
- Kemudian baru dilakukan perkalian 6 * 2 = 12.
- Setelah itu dilakukan perbandingan antara 12 dengan 10, 12 > 10 yang hasilnya true.

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`.

   Jika bagian pertama (++a * 2 > b) menghasilkan *false*, maka bagian kedua (b++ % 3 == 1) tidak akan jalan sama sekali. Hal ini terjadi, karena operator && menggunakan konsep *short-circuit evaluation*, yaitu jika suatu bagian logika salah maka hasil kesulurahan pasti salah, sehingga Java tidak perlu mengecek bagian kanan lagi.

   Dalam matematika-pun jika ada operasi "&" dan salah satunya salah maka yang lainnya salah, karena sistemnya terikat. Sehingga dalam kasus ini, nilai b tidak akan mengalami *increment*, jadi nilai b tetap seperti semula.

3. Output: Berapakah nilai akhir dari `result`, `a`, dan `b`?

   Nilai akhir dari program tersebut berdasarkan nilai awal a = 5 dan b = 10 adalah a: 6 dan b: 11.

   Pada bagian kiri terjadi;
   - ++a, sehingga a menjadi 6
   - 6 × 2 = 12
   - 12 > 10, nilainya true

     Karena true maka lanjut ke bagian kanan.

   Pada bagian kanan terjadi;
   - b++, sehingga nilai b dipakai dulu = 10
   - 10 % 3 = 1
   - 1 == 1, nilainya true
   - Karena hasilnya true, maka nilai b bertam satu sehingga b menjadi 11

   Proses terakhir;
   - result = true && true → true
   - a = 6
   - b = 11

Nama Teman dan Hal yang Dibantu (Opsional)
-