### MODUL 8

### *MULTI INHERITANCE*

1. **Tujuan**
2. Mahasiswa memahami dan bisa mengimplementasikan konsep penurunan sifat dari beberapa class induk dengan menggunakan *Interface*.
3. Memahami jenis *Multi Inheritance*
4. **Dasar Teori**

**B.1 *Abstract Method***

*Abstract Method* adalah sebuah ‘*method* dasar’ yang harus diimplementasikan ulang di dalam *class* anak (*child class*). *Abstract method* ditulis tanpa isi dari *method*, melainkan hanya ‘*signature*’-nya saja. *Signature* dari sebuah *method* adalah bagian *method* yang terdiri dari nama method dan parameternya (jika ada). *Abstract method* merupakan sebuah *method* yang dideklarasikan dengan menambahkan *keyword* abstract pada deklarasinya, dan tanpa ada implementasi dari *method* tersebut. Dalam arti, hanya pendeklarasian saja, tanpa tanda sepasang kurung kurawal. Tetapi diakhiri dengan tanda titik koma(;), seperti contoh sederhana berikut ini :

|  |
| --- |
| abstract void setName();  abstract void setMakanan(); |

Kode program diatas artinya : Mendeklarasikan *abstract method* dengan nama setName dan setMakanan. Jadi intinya method abstrak itu adalah sebuah method yang tidak tahu mau kita apakan nantinya, sebuah *class* dan *method* abstrak dibuat oleh seorang *programmer* sebagai acuan atau gambaran dari program yang ingin mereka buat.

**B.2 *Class Abstract***

Penggunaan *class* *abstract* dan *interface* pada java, merupakan konsep dari pemrograman berorientasikan objek (OOP), pada java, *abstract* dan *interface* fungsi dan kegunaannya yang hampir sama, kedua class tersebut digunakan untuk di implementasikan pada *class* turunannya. Didalam *class* abstract dan *interface* terdapat *method*-*method* yang tidak mempunyai *body*, *method* tersebut akan kita implementasikan pada *class* turunannya seperti membuat sebuah pernyataan atau *statement*.

*Abstract Class* adalah sebuah *class* yang tidak bisa di-instansiasi (tidak bisa dibuat menjadi objek) dan berperan sebagai ‘kerangka dasar’ bagi *class* turunannya. Di dalam *abstract* *class* umumnya akan memiliki *abstract* *method*. Kelas Abstrak adalah kelas yang mengandung satu *method* abstrak atau lebih, kelas abstrak tersebut digunakan hanya untuk membuat sebuah *method* yang tanpa ada implementasinya secara langsung. Abstract class digunakan di dalam *inheritance* (pewarisan *class*) untuk ‘memaksakan’ implementasi *method* yang sama bagi seluruh *class* yang diturunkan dari *abstract* *class*. *Abstract* *class* digunakan untuk membuat struktur logika penurunan di dalam pemrograman objek. Konsep *abstract* *class* dan *abstract* *method* akan lebih mudah dipahami dengan menggunakan contoh. Misalkan kita ingin membuat *class* yang terdiri dari berbagai jenis komputer seperti *class* laptop, class PC, class netbook, dan lain-lain. Seluruh *class* ini tentunya memiliki sifat-sifat komputer, seperti memiliki spesifikasi, memiliki *processor*, dan membutuhkan listrik. Dalam implementasinya, kita bisa membuat seluruh class ini diturunkan dari *class* komputer.

**B.3 *Interface***

*Interface* memiliki pengertian dan fungsi yang hampir sama dengan *class Abstract*, walalupun fungsi dari keduanya sama, tetapi ada beberapa perbedaan yang perlu diketahui, berikut ini adalah tabel yang menjelaskan perbedaan diantara *abstract* dan interface:

|  |  |
| --- | --- |
| ***Abstract*** | ***Interface*** |
| Bisa berisi *abstract* dan *non*-*abstract* *method*. | Hanya boleh berisi *abstract* *method*. |
| Kita harus menuliskan sendiri modifiernya. | Kita tidak perlu menulis *public* *abstract* di depan nama *method*. Karena secara *implisit*, *modifier* untuk *method* di *interface* adalah *public* dan *abstract*. |
| Bisa mendeklarasikan *constant* dan *instance* *variable*. | Hanya bisa mendeklarasikan *constant*. Secara implisit *variable* yang dideklarasikan di *interface* bersifat *public*, *static* dan *final*. |
| *Method* boleh bersifat *static*. | *Method* tidak boleh bersifat *static*. |
| *Method* boleh bersifat *final*. | *Method* tidak boleh bersifat *final*. |
| Suatu *abstact* *class* hanya bisa meng-*extend* satu *abstract* *class* lainnya | Suatu *interface* bisa meng-*extend* satu atau lebih *interface* lainnya. |
| *Abstract* *class* hanya bisa meng-*extend* satu *abstract* c*l*ass dan meng-*implement* beberapa interface | Suatu *interface* hanya bisa meng-*extend* *interface* lainnya dan tidak bisa meng-implement *class* atau int |

***B.4 Multiple Inheritance***

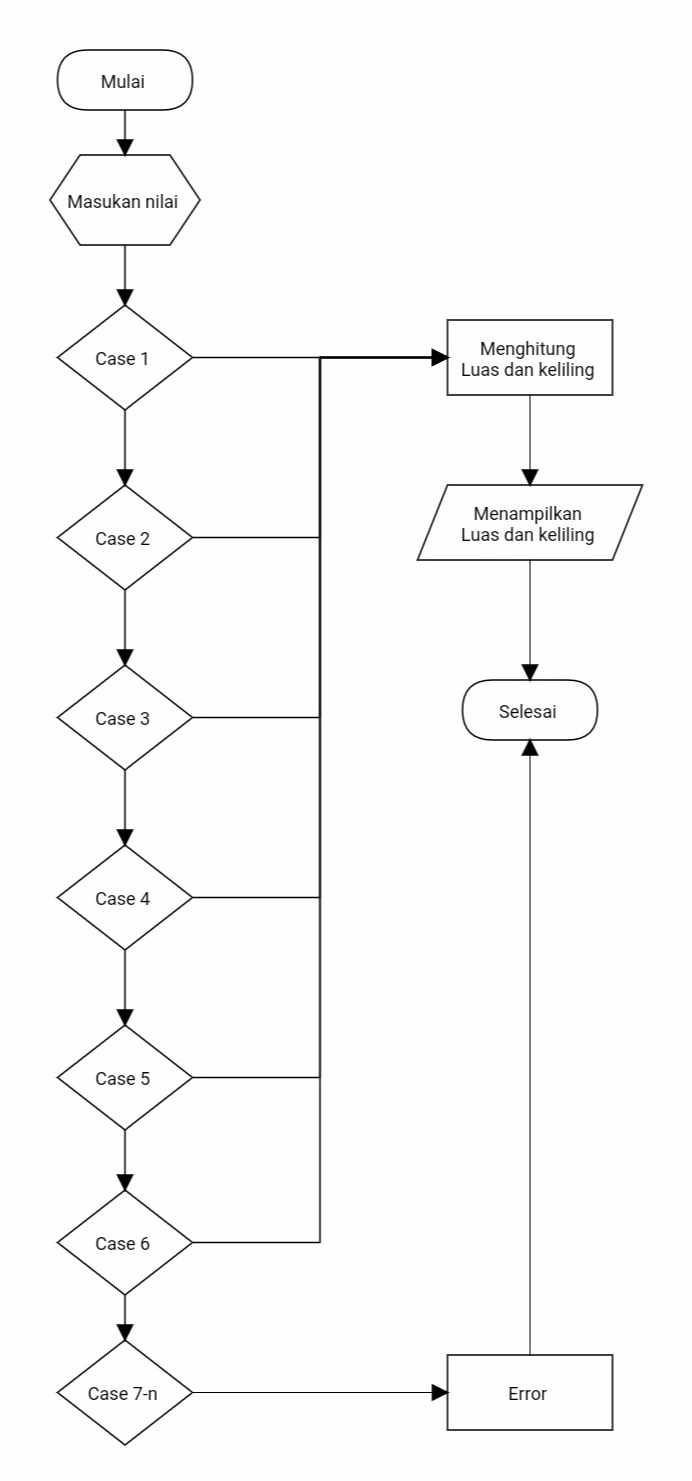
*Multiple* *inheritance* adalah fitur dari konsep berorientasi *object*, dimana sebuah *class* dapat mewariskan propertinya kepada lebih dari satu *parent* *class*. Permasalahan yang terjadi ketika terdapat *method* dengan *signature* sama dalam kedua *super* *class* dan *subclass* tersebut. Ketika *method* dipanggil maka kompilator tidak dapat menentukan *method* *class* mana yang akan dipanggil, bahkan ketika memanggil *method* *class* mana yang harus diproritaskan terlebih dahulu.

Satu-satunya cara untuk mengimplementasikan *multiple* *inheritance* pada Java adalah dengan mengimplementasikan banyak *interface* dalam sebuah *class*. Pada Java, satu *class* dapat mengimplementasikan dua atau lebih *interface*, dimana hal ini tidak akan menyebabkan ambiguitas karena semua *method* yang dideklarasikan dalam *interface* diimplementasikan pada *class*

1. **Studi Kasus**

Buatlah aplikasi kalkulator untuk menghitung luas dan keliling bangun datar yang terdiri dari persegi, persegi panjang, segitiga, lingkaran, trapesium dan jajar genjang. Program dibuat dengan dinamis dengan menggunakan menu seperti switch case atau if else.

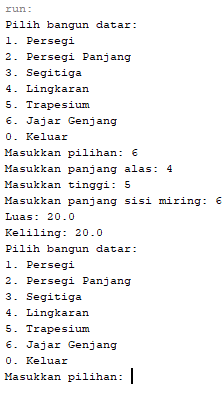
1. **Data Hasil**
2. *Flowchart*



1. *Script*

|  |
| --- |
| package kalkulatormenghitungluas;  import java.util.Scanner;  public class Kalkulatormenghitungluas{      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          while (true) {              System.out.println("Pilih bangun datar:");              System.out.println("1. Persegi");              System.out.println("2. Persegi Panjang");              System.out.println("3. Segitiga");              System.out.println("4. Lingkaran");              System.out.println("5. Trapesium");              System.out.println("6. Jajar Genjang");              System.out.println("0. Keluar");              System.out.print("Masukkan pilihan: ");              int choice = scanner.nextInt();              switch (choice) {                  case 1:                      hitungPersegi();                      break;                  case 2:                      hitungPersegiPanjang();                      break;                  case 3:                      hitungSegitiga();                      break;                  case 4:                      hitungLingkaran();                      break;                  case 5:                      hitungTrapesium();                      break;                  case 6:                      hitungJajarGenjang();                      break;                  case 0:                      System.out.println("Terima kasih, program selesai.");                      System.exit(0);                  default:                      System.out.println("Pilihan tidak valid, coba lagi.");              }          }      }      private static void hitungPersegi() {          Scanner scanner = new Scanner(System.in);          System.out.print("Masukkan sisi persegi: ");          double sisi = scanner.nextDouble();          double luas = sisi \* sisi;          double keliling = 4 \* sisi;          System.out.println("Luas: " + luas);          System.out.println("Keliling: " + keliling);      }      private static void hitungPersegiPanjang() {          Scanner scanner = new Scanner(System.in);          System.out.print("Masukkan panjang: ");          double panjang = scanner.nextDouble();          System.out.print("Masukkan lebar: ");          double lebar = scanner.nextDouble();          double luas = panjang \* lebar;          double keliling = 2 \* (panjang + lebar);          System.out.println("Luas: " + luas);          System.out.println("Keliling: " + keliling);      }      private static void hitungSegitiga() {          Scanner scanner = new Scanner(System.in);          System.out.print("Masukkan alas: ");          double alas = scanner.nextDouble();          System.out.print("Masukkan tinggi: ");          double tinggi = scanner.nextDouble();          double luas = 0.5 \* alas \* tinggi;          double keliling = alas + tinggi + Math.sqrt(alas \* alas + tinggi \* tinggi);          System.out.println("Luas: " + luas);          System.out.println("Keliling: " + keliling);      }      private static void hitungLingkaran() {          Scanner scanner = new Scanner(System.in);          System.out.print("Masukkan jari-jari lingkaran: ");          double jariJari = scanner.nextDouble();          double luas = Math.PI \* jariJari \* jariJari;          double keliling = 2 \* Math.PI \* jariJari;          System.out.println("Luas: " + luas);          System.out.println("Keliling: " + keliling);      }      private static void hitungTrapesium() {          Scanner scanner = new Scanner(System.in);          System.out.print("Masukkan panjang sisi atas: ");          double sisiAtas = scanner.nextDouble();          System.out.print("Masukkan panjang sisi bawah: ");          double sisiBawah = scanner.nextDouble();          System.out.print("Masukkan tinggi: ");          double tinggi = scanner.nextDouble();  System.out.print("Masukkan panjang sisi miring: ");          double sisiMiring = scanner.nextDouble();          double luas = 0.5 \* (sisiAtas + sisiBawah) \* tinggi;          double keliling = sisiAtas + sisiBawah + tinggi + sisiMiring;          System.out.println("Luas: " + luas);          System.out.println("Keliling: " + keliling);      }      private static void hitungJajarGenjang() {          Scanner scanner = new Scanner(System.in);          System.out.print("Masukkan panjang alas: ");          double alas = scanner.nextDouble();          System.out.print("Masukkan tinggi: ");          double tinggi = scanner.nextDouble();          System.out.print("Masukkan panjang sisi miring: ");          double sisiMiring = scanner.nextDouble();          double luas = alas \* tinggi;          double keliling = 2 \* (alas + sisiMiring);          System.out.println("Luas: " + luas);          System.out.println("Keliling: " + keliling);      }  } |

1. Hasil *running*



**DAFTAR PUSTAKA**

Anonim. 2023. Modul Praktikum Pemrograman Berorientasi Objek. Laboratorium Jaringan dan Komputer. Jurusan Teknik Elektro. Fakultas Teknik. Universitas Mataram.

Budiarto, Raden. 2021. *“Pemrograman Berorientasi Objek Teori dan Implementasi dengan Java”.* Jakarta: RBH.

Sutopo, A. H. dan Masya, Fajar. 2005. “*Pemrograman Berorientasi Objek dengan Java*”. Yogyakarta: Graha Ilmu.

Yoevestian, Whindy. 2007. “*Java 5 Tiger”.* Jakarta: PT Elex Media Komputindo.