# MODUL V
# Pewarisan Jamak (Multiple Inheritance)

## I. TUJUAN

a. Mengerti prinsip Pewarisan Jamak dan pemakaiannya dalam membentuk suatu kelas dalam bahasa C++ dan Java

b. Mengerti tentang *interface* dan implementasinya

## II. DASAR TEORI

### a. Multiple Inheritance

Pada modul sebelumnya kita telah membahas mengenai pewarisan tunggal (*inheritance*), baik pewarisan satu tingkat maupun lebih. Pada pewarisan tunggal setiap kelas anak mewarisi data maupun *method* dari satu kelas induk. ***Multiple Inheritance*** memungkinkan suatu kelas mewarisi data maupun *method* lebih dari satu kelas induk.

Konstruktor kelas turunan memanggil semua konstruktor kelas induk untuk menginisialisasi data member kelas-kelas induk. Pada saat objek kelas anak diciptakan dengan urutan penulisan:

**class** *Anak*: **public** *Induk1*, **public** *Induk2*

maka urutan eksekusi konstruktor adalah:

- *Induk1*
- *Induk2*
- *Anak*

  Sedangkan urutan eksekusi destruktor adalah kebalikan dari konstruktor yaitu:
- *Anak*
- *Induk2*
- *Induk1*

Ada kemungkinan bahwa pada pewarisan jamak terdapat dua buah fungsi anggota pada kelas dasar dengan nama sama. Jika kedua fungsi anggota ini diwariskan ke kelas anak maka dapat menimbulkan kerancuan (ambigu).

### b. Interface

*Multiple Inheritance* merupakan topik kompleks yang dibahas secara detail pada pemrograman C++. Pada pemrograman Java, solusi atas prinsip pewarisan jamak (*Multiple Inheritance*) adalah dengan menggunakan *interface*. *Interface* adalah kumpulan *method* yang hanya memuat deklarasi dan struktur *method*, tanpa detail implementasinya.*interface* digunakan bila ingin mengaplikasikan suatu *method* yang spesifik, yaitu tidak diperoleh dari proses pewarisan kelas. *Interface* bersifat disisipkan (*embeded*) pada program, dan *programmer* diberi keleluasaan untuk merancang dan mendefinisikan sendiri detail prosesnya.

Komponen penyusun *interface* adalah sebagai berikut:

| Komponen Penyusun | Interface |
|---|---|
| **Tipe data / variabel** | **Hanya boleh berupa konstanta** |
| ***Method*** | **Hanya berupa *signature*, *method* yang terdapat dalam *interface* adalah *method* abstrak** |
| **Sintaks *method*** | **Tidak perlu membubuhkan *modifier* abstract pada semua *method* di dalam kelas** |

Bila suatu kelas ingin menggunakan *interface*, maka pada deklarasi kelas tersebut ditambahkan *keyword* **implements** dan nama *interfacenya*. Selanjutnya, *method* dari *interface* tersebut harus disisipkan dan didefinisikan secara detail pada kelas. Yang perlu digarisbawahi adalah bahwa *multiple inheritance* dalam C++ **tidak sama** dengan *interface* dalam Java.

## III. GUIDED

### a. Multiple Inheritance dalam C++

**PaketBuku.cpp**

```
#include<iostream.h>
#include<string.h>
#include <conio.h>
class Buku
{
     private :
           char judul[35];
           char pengarang[25];
           int  jumlah;
     public:
     void inisialisasiBuku(char *jdl,char *pngarang,int jmlh)
     {
           strcpy(judul, jdl);
           strcpy(pengarang, pngarang);
           jumlah = jmlh;
     }
     void infoBuku()
     {
           cout<<" Judul  :"<<judul<<endl;
           cout<<" Pengarang :"<<pengarang<<endl;
           cout<<" Jumlah buku :"<<jumlah<<endl;
     }
};

int main ()
{
    Buku novel,fiksi;
    novel. inisialisasiBuku
    ("Meriam Benteng   navarone","Alistair  Maclean",12);
    fiksi. inisialisasiBuku
    ("Jurassic park","Michael Crichton",3);
    novel. infoBuku();
    fiksi. infoBuku();
    getch();
    return 0;
}
```

**b. Multiple Inheritance dalam Java → Project DemoPaket.java**

**Buku.java**

```
/**Kelas Buku*/
class Buku
{
  String judul,pengarang;
  long hargaBuku;
  public Buku(String judul,String pengarang,long hargaBuku)
  {
       this.judul=judul;
       this.pengarang=pengarang;
       this.hargaBuku=hargaBuku;
  }
  public void cetakBuku()
  {
      System.out.println("\nJudul     : "+judul);
      System.out.println("Pengarang : "+pengarang);
      System.out.println("Harga Buku: Rp "+hargaBuku);
      System.out.println();
  }
}
```

**InterfaceCD.java**

```
/**Interface CD*/
interface InterfaceCD
{
  void cetakCD();
  long getHargaCD();
}
```

**CD.java**

```
/**Kelas CD*/
class CD
{
   String ukuran;
   long hargaCD;
   public CD(String ukuran,long hargaCD)
   {
      this.ukuran=ukuran;
      this.hargaCD=hargaCD;
   }
   public long getHargaCD()
   {
      return hargaCD;
   }
   public void cetakCD()
   {
      System.out.println("Ukuran CD : "+ukuran);
      System.out.println("Harga CD  : Rp "+hargaCD);
      System.out.println();
   }
}
```

**ChildCD.java**

```
/**kelas ChildCD merupakan turunan dari CD dan
mengimplementasikan interface InterfaceCD*/
class ChildCD extends CD implements InterfaceCD
{
  public ChildCD(String ukuran,long hargaCD)
  {
    super(ukuran,hargaCD);
  }
}
```

**Paket.java**

```
/**kelas Paket merupakan turunan dari Buku dan mengimplementasikan
interface InterfaceCD*/
class Paket extends Buku implements InterfaceCD
{
   long hargaPaket;
   InterfaceCD interfaceCD;
   public Paket(String judul,String pengarang,long hargaBuku,String
   ukuran,long hargaCD)
   {
      super(judul,pengarang,hargaBuku);
      interfaceCD = new ChildCD(ukuran,hargaCD);
   }
   public void hitungHargaPaket()
   {
      hargaPaket=super.hargaBuku + getHargaCD();
   }
   public void cetakCD()
   {
      interfaceCD.cetakCD();
   }
   public long getHargaCD()
   {
      return (interfaceCD.getHargaCD());
   }
   public void cetakPaket()
   {
      super.cetakBuku();
      cetakCD();
      System.out.println("Harga paket Buku dan CD: Rp "+ hargaPaket
      +"\n");
    }
}
```

**Main.java**

```
class DemoPaket
{
  public static void main(String args[])
  {
    Paket a=new Paket("Pemrograman Berorientasi Objek",
    "Benyamin Langgu Sinaga",60000,"700 MB",50000);
    a.hitungHargaPaket();
    a.cetakPaket();
  }
}
```

**Untuk dibahas dalam Laporan :**

"*Multiple Interface Inheritance* BUKAN *Multiple Implementation Inheritance*"

Apakah Anda setuju dengan pernyataan di atas? Jika iya, jelaskan mengapa dan apa perbedaannya! Dan jika tidak, berikan alasannya!

**Contoh lain (boleh dicoba di luar praktikum)**

**DemoInterface.java**

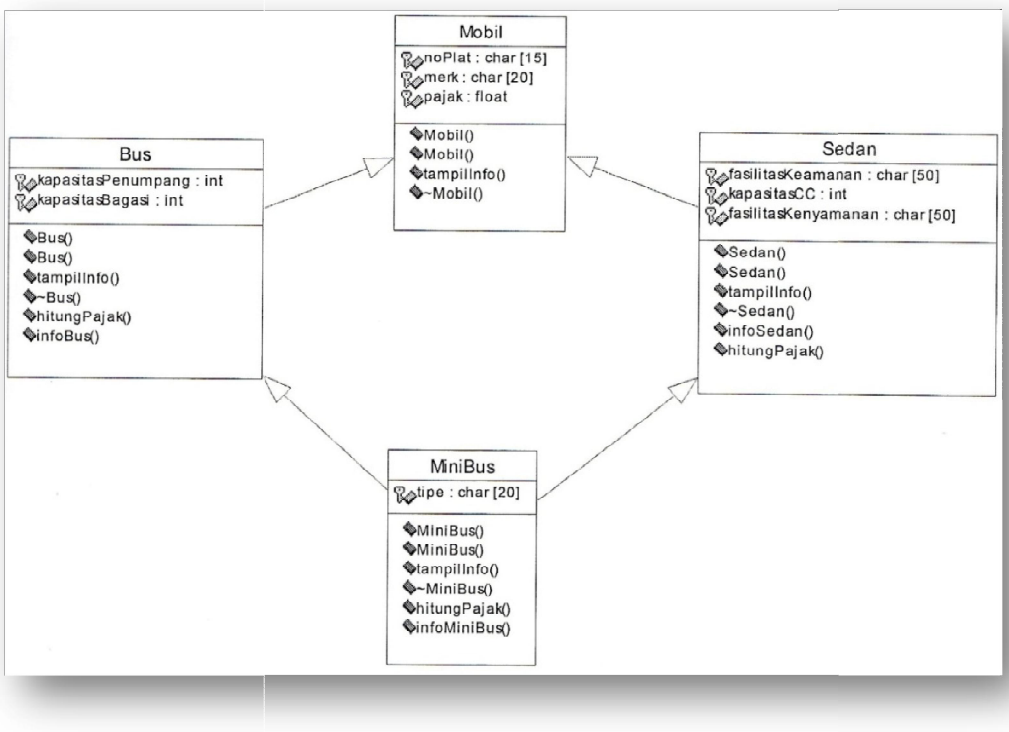
```
/**Deklarasi Interface*/
interface Konstanta
{
  double KONST_PI = 3.14;
  String KONST_SATUAN_LUAS = " cm. persegi ";
  String KONST_SATUAN_PANJANG = " cm.";
}

public class DemoInterface implements Konstanta
{
  /**Main Method*/
  public static void main(String args[])
  {
    System.out.println("\nPROGRAM DEMO INTERFACE");
    System.out.println("-------------------------\n");

    double radius = 10;
    System.out.println("Radius Lingkaran  : "+ radius);
    System.out.println("Luas Lingkaran    : "+
    (KONST_PI*radius*radius)+KONST_SATUAN_LUAS);
    System.out.println("Keliling Lingkaran: "+
    (2*KONST_PI*radius)+Konstanta.KONST_SATUAN_PANJANG);
    System.out.println();
  }
}
```

## IV. UNGUIDED

Perhatikanlah Object Relationship Diagrams pada gambar dibawah ini. Anda diminta untuk mengimplementasikan kasus Pewarisan Jamak pada diagram tersebut.




1. Buatlah kelas `Mobil` dengan ketentuan sebagai berikut (**20%**) :
   a. memiliki atribut : `noPlat`, `merk`, `pajak`
   b. memiliki *default* konstruktor dan konstruktor buatan
   c. memiliki *method* `tampilInfo()`, untuk menampilkan informasi `Mobil`
2. Buatlaah kelas `Bus` dengan ketentuan sebagai berikut (**20%**) :
   a. turunan dari kelas `Mobil`
   b. memiliki atribut : `kapasitasPenumpang`, `kapasitasBagasi`
   c. memiliki *default* konstruktor dan konstruktor buatan
   d. memiliki *method* `infoBus()`, untuk menampilkan informasi `Bus` (`kapasitasPenumpang`, `kapaitasBagasi`)
   e. memiliki *method* `tampilInfo()`, untuk menampilkan informasi keseluruhan dari `Bus` (memanggil *method* `tampilInfo ()` dari kelas `Mobil` dan *method* `infoBus()`)
   f. memiliki *method* `float hitungPajak()`, untuk mengembalikan perhitungan besar pajak dengan rumus : `pajak + (pajak * kapasitaspenumpang * kapasitasBagasi * 0.00005)`
3. Buatlah kelas `Sedan` dengan ketentuan sebagai berikut (**20%**) :
   a. turunan dari kelas `Mobil`
   b. memiliki *atribut* : `fasilitasKeamanan`, `kapasitasCC`, `fasilitasKenyamanan`
   c. memiliki *default* konstruktor, dan konstruktor bentukan

d. memiliki *method* `infoSedan()`, untuk menampilkan informasi `Sedan` (`fasilitasKeamanan, kapasitasCC, fasilitasKenyamanan`)

e. memiliki *method* `tampilInfo()`, untuk menampilkan informasi keseluruhan dari `Bus` (memanggil *method* `tampilInfo()` dari kelas `Mobil` dan *method* `infoSedan()`)

f. memiliki *method* `float hitungPajak()`, untuk mengembalikan perhitungan besar pajak dengan rumus : `pajak + (pajak + (pajak * kapasitasCC * 0.00005)`

4. Buatlah kelas `MiniBus` dengan ketentuan sebagai berikut (**40%**) :

   a. turunan dari kelas `Sedan`, `Bus`

   b. memiliki *atribut* : `tipe`

   c. memiliki *default* konstruktor dan konstruktor bentukan

   d. memiliki *method* `infoMiniBus()`, untuk menampilkan informasi `MiniBus` (jika `tipe` adalah **"Pribadi"** maka tampilkan "`Tipe MiniBus : Pribadi, digunakan sebagai kendaraan pribadi`", sedangkan jika `tipe` adalah **"Wagon"** maka tampilkan "`Tipe MiniBus : Wagon digunakan sebagai kendaraan angkut/travel`")

   e. memiliki *method* `tampilInfo()`, untuk menampilkan informasi keseluruhan dari `miniBus` (memanggil *method* `tampilInfo()` dari kelas `Sedan`, *method* `infoBus()` dari kelas `Bus`, dan *method* `infoMiniBus ()` )

   f. memiliki *method* `float hitungPajak()`, untuk mengembalikan perhitungan besar pajak dengan rumus :

      - Jika `tipe` adalah **"Pribadi"**, maka rumusnya : (`Sedan::hitungPajak () * 0.05) + (Bus::hitungPajak()*0.03`)
      - Jika `tipe` adalah **"Wagon"**, maka rumusnya : (`Sedan::hitungPajak() * 0.03)+(Bus::hitungPajak()*0.05`)

**==========[ Selamat Berlatih ]==========**