MODUL VI
Polimorfisme

I. TUJUAN

a. Mengerti prinsip polimorfisme dalam bahasa C++ dan Java
b. Mengerti tentang prinsip polimorfisme dan pemakaiannya dalam membentuk suatu kelas

II. DASAR TEORI

a. Polimorfisme

Polimorfisme dapat berarti "mempunyai banyak bentuk" sehingga dapat disimpulkan, **Polimorfisme** adalah kemampuan untuk meminta objek yang berbeda untuk melaksanakan tugas yang sama dan membuat objek tahu bagaimana untuk mencapainya dengan caranya sendiri. Polimorfisme menunjukkan kemampuan untuk menangani dua atau lebih bentuk obyek yang berlainan saat eksekusi berlangsung.

Polimorfisme dapat diilustrasikan sebagai berikut, perhatikanlah penggunaan kata "mentah" dalam beberapa kalimat. "Sayuran itu masih **mentah**, belum dimasak", "Pukulan petinju itu berhasil di**mentah**kan oleh lawannya", "Gagasan ini masih **mentah** sehingga perlu di bahas kembali". Kata "mentah" pada contoh di atas dapat diaplikasikan pada berbagai objek dan dapat di-interpretasikan ke dalam beberapa makna.

1. Redefinisi fungsi

- Kelas Pelajar dan Pekerja mewarisi Manusia
- Fungsi cetakInfo() diwariskan ke kelas anak

  Di definisi ulang untuk menampilkan atribut NIM (Pelajar) dan NPP (kelas Pekerja).

2. Fungsi Virtual

- Objek kelas turunan diperlakukan sebagai objek kelas dasar
- Sekilas hampir sama dengan mekanisme redefinisi fungsi

Fungsi virtual merupakan dasar dari polimorfisme. Suatu fungsi anggota yang dibuat sebagai fungsi virtual perlu dideklarasikan ulang pada setiap kelas turunan. Bentuk pendeklarasiannya harus sama, baik nilai balikan maupun argument-argumennya. Namun keyword virtual tidak harus disertakan pada kelas turunannya.

Fungsi virtual di bagi menjadi 2 jenis yaitu :

a) Fungsi virtual biasa

Untuk fungsi virtual biasa ini, hampir mirip dengan redefinisi fungsi. Untuk lebih jelasnya dapat di lihat pada contoh program saat praktikum.

```
// deklarasi fungsi virtual
virtual void keterangan()
// atau fungsi yang memiliki nilai balikan seperti
virtual int hitung_nilai(int nil)
```

b) Fungsi virtual murni (pure virtual)

Hasil dari fungsi pure virtual ini adalah **kelas abstrak**. Pada kelas fungsi ini hanya dideklarasikan saja dan tidak memiliki definisian. Selanjutnya pada kelas turunan baru dideklarasikan kembali sebagai virtual biasa dan memiliki definisian fungsinya.

```
// deklarasi fungsi pure virtual
virtual void keterangan()=0;
// atau fungsi yang memiliki nilai balikan seperti
virtual int hitung_nilai(int nil)=0;
```

b. Abstract Class

Di dalam Java, class yang memiliki method tanpa definisi atau method kosong dapat kita jadikan kelas abstrak. Pendefinisian kelas abstrak biasanya digunakan untuk polimorfisme di mana subclass dari Class Abstrak yang mendefinisikan method kosong dari Class Abstrak. Class abstrak **tidak bisa** di-instantiate (di-create menjadi objek), tetapi **bisa** me-refer objek kongkrit yang Classnya diturunkan dari dirinya.

Contoh kelas abstrak yang di-instantiate :

```
public abstract class A
{       //codes
}
public class B
{       public static void main(String args[])
        {
                A objek = new A();          //invalid
        }
}
```

Untuk contoh kelas abstrak yang me-refer objek kongkrit yang kelasnya diturunkan dari kelas abstrak tersebut, bisa di lihat pada program saat demo.

c. Abstract Method

Abstract Method adalah method yang didefinisikan dengan keyword **abstract**. Di dalam sebuah kelas abstrak, kita dapat membuat *abstract method*. Abstract method adalah method yang tidak ada implementasinya. Implementasi dari semua abstract method yang dimiliki kelas abstrak dilakukan di kelas konkrit yang di turunkan dari kelas abstrak tersebut. Method Abstrak ini hanya dapat di miliki oleh kelas abstrak. Sebuah kelas konkrit tidak dapat memiliki method abstrak.

Contoh Implementasi method abstrak :

```
public abstract class A
{       public abstract void F();
}
public class B extends A
{       public void F()
        {       //Implementasi sesungguhnya

        }
}
```

III. GUIDED

1. Polimorfisme dalam C++

Demo1.cpp

```
#include <iostream.h>
#include <conio.h>
class Makhluk
{
    public :
    //deklarasi fungsi virtual
    virtual void keterangan()
    {
        cout<<"keterangan() pada kelas Makhluk"<<endl;
    }
};
class Mamalia:public Makhluk
{
    public:
    void keterangan()
    {
        cout<<"keterangan() pada kelas Mamalia"<<endl;
    }
};
class Sapi:public Mamalia
{
    void keterangan()
    {
        cout<<"keterangan() pada kelas Sapi"<<endl;
    }
};
int main()
{
    //definisi objek mamalia
    Mamalia mamalia;
    //definisi objek sapi
    Sapi sapi;
    //definisi pointer ke objek berkelas Makhluk
    Makhluk *binatang;
    //menunjuk ke objek mamalia
    binatang=&mamalia;
    binatang->keterangan();
    cout<<"----------------------"<<endl;
    //menunjuk ke objek sapi
    binatang=&sapi;
    binatang->keterangan();
    getch();
    return 0;
}
```

Demo2.cpp

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class Keluarga
{
    protected:
       char nama[20];
    public:
    //konstruktor Keluarga
    Keluarga(char Nama[20])
    {
        strcpy(nama,Nama);
    }
    //destruktor Keluarga
    virtual ~Keluarga()
    {
        cout<<"Destruktor di kelas Keluarga.."<<endl;
        delete[] nama;
    }
    //deklarasi fungsi virtual murni
    virtual void info()=0;
};
class Keturunan : public Keluarga
{
    private :
       char nama_depan[15];

    public :
       Keturunan(char Nama_depan[15], char Nama_kel[20]) :
       Keluarga(Nama_kel)
    {
        strcpy(nama_depan,Nama_depan);
    }

    ~Keturunan()
    {
        cout<<"Destruktor di Keturunan.."<<endl;
        delete [] nama_depan;
    }

    void info()
    {
        cout<<nama_depan<<' '<<nama<<endl;
    }
};
int main()
{
    Keluarga *anak1 = new Keturunan("Umar","Khatab");
    anak1->info();
    Keluarga *anak2 = new Keturunan("Udin","Pambudi");
    anak2->info();
    delete anak1;
    delete anak2;
    getch();
    return 0;
}
```

2. Polimorfisme dalam Java

2.1 Nama-nama Binatang

Binatang.java

```
public abstract class Binatang
{
   Binatang(String jenis)
   {
      this.jenis=jenis;
   }
   protected   abstract   void   suara();    //Deklarasi,tidak
   didefisikan
   public String toString()
   {
      return "Seekor "+jenis;
   }
   private String jenis;
}
```

Burung.java

```
public class Burung extends Binatang
{
   Burung(String nama)
   {
      super("Burung");
      this.nama=nama;
   }
   public void suara()
   {
      System.out.println("berkicau");
   }
   public String toString()
   {
      return super.toString()+" "+nama;
   }
   private String nama;
}
```

Kucing.java

```
public class Kucing extends Binatang
{
   Kucing(String nama)
   {
      super("Kucing");
      this.nama=nama;
   }
   public void suara()
   {
      System.out.println("mengeong");
   }
   public String toString()
   {
      return super.toString()+" "+nama;
   }
   private String nama;
}
```

Anjing.java

```
public class Anjing extends Binatang
{
   Anjing(String nama)
   {
      super("Anjing");
      this.nama=nama;
   }
   public void suara()
   {
      System.out.println("menggonggong");
   }
   public String toString()
   {
      return super.toString()+" "+nama;
   }
   private String nama;
}
```

Kambing.java

```
public class Kambing extends Binatang
{
   Kambing(String nama)
   {
      super("Kambing");
      this.nama=nama;
   }
   public void suara()
   {
      System.out.println("mengembik");
   }
   public String toString()
   {
      return super.toString()+" "+nama;
   }
   private String nama;
}
```

CobaPolimorpic.java

```
import java.util.Random;
public class CobaPolimorpic
{
   public static void main(String []args)
   {
            Binatang[] peliharaanku={new Burung("Kakak Tua"),
                                     new Kambing("Etawa"),
                                     new Anjing("Kintamani"),
                                     new Kucing("Anggora")};
            Binatang kesayangan;
            Random pilihan=new Random();
            //memilih secara acak
kesayangan=peliharaanku[pilihan.nextInt(peliharaanku.length);
   //mengacak bilangan dari 0 sampai length-1 atau (i-1).
   System.out.println("Binatang Kesayangan anda :
   "+kesayangan);
   System.out.print("Suaranya : ");
   kesayangan.suara();
   }
}
```

2.2 Pegawai

Direktur.java

```
public final class Direktur extends Pegawai
{
private double gajiDirektur;
private double dividenSaham;
//Konstruktor Kelas Direktur
public Direktur(String nama, double gaji, double dividen)
{
     super(nama); //Memanggil konstruktor kelas Pegawai
     setGajiDirektur(gaji);
     setDividen(dividen);
}
public void setGajiDirektur(double gaji) //Mengeset gaji direktur
{
     if(gaji>0)
           gajiDirektur=gaji;
     else
           gajiDirektur=0;
}
//Mengeset hasil pembagian dividen keuntungan saham
public void setDividen(double dividen)
{
    if(dividen>0)
           dividenSaham=dividen;
     else
           dividenSaham=0;
}
public String nama()//Method yang mengembalikan nama
{
     return super.namaPegawai();
}
public String jabatan()//Method yang mengembalikan jabatan
{
     return "Direktur";
}
//Method yang mengembalikan besar gaji direktur
public double gajiPerBulan()
{
     return gajiDirektur;
}
//Method yang mengembalikan besar dividen saham
public double labaDividen()
{
     return dividenSaham;
}
//Pengimplementasian / Pendefinisian method abstract dari kelas Pegawai
//Method ini mengembalikan besar gaji direktur
public double income()
{
     return(gajiDirektur+dividenSaham);
}
}
```

Pegawai.java

```
public abstract class Pegawai
{
   private String namaPeg;
   //konstruktor
   public Pegawai(String nama)
   {
      namaPeg=nama;
   }
   //method (get) untuk mengembalikan nama pegawai
   public String namaPegawai()
   {
      return namaPeg;
   }
   //Method abstrak ini diwariskan ke semua kelas yang
   diturunkan dari kelas abstrak ini
   public abstract double income();
}
```

Test.java

```
import java.text.DecimalFormat;
public class Test
{
   /**Main Method*/
   public static void main( String args[] )
   {
      Pegawai pgw;
      //Membuat objek referensi dari kelas abstrak //Pegawai
      String output = "";
      Direktur d = new Direktur("Wahyu", 12000000.00, 7500000.00);
      DecimalFormat digitPresisi = new DecimalFormat("0.00");
      pgw = d;
      /*objek referensi dari kelas abstrak pegawai (pgw) merefer objek
      dari kelas Direktur (d) yang diturunkan dari kelas abstrak
      pegawai */
      System.out.println("\nDEMO INHERITANS, ENKAPSULASI,POLIMORFI");
      System.out.println("-----------------------------------\n");
      // Mencetak informasi Direktur ke console
      System.out.println("Nama    : " + d.namaPegawai() + "\n" +
      "Jabatan : " + d.jabatan() + "\n" + "Gaji    : " +
      digitPresisi.format(d.gajiPerBulan()) + "\n" +
      "Dividen : " + digitPresisi.format(d.labaDividen())+ "\n" +
      "Total     : " + digitPresisi.format(d.income()) + "\n");
      System.exit( 0 );
   }
}
```

2.3 Ekspersi Wajah

EkspresiWajah.java

```
class EkspresiWajah
{
   public String respons()
   {
      return("Lihat Wajahku ini");
   }
}
```

Gembira.java

```
class Gembira extends EkspresiWajah
{
   public String respons()
   {
      return("Ha..ha..saya lagi senang =)");
   }
}
```

Sedih.java

```
class Sedih extends EkspresiWajah
{
   public String respons()
   {
      return("Hiks..hiks.. =(");
   }
}
```

Ekspresi.java

```
public class Ekspresi
{
   public static void main(String args[])
   {
      System.out.println("DEMO POLIMORFISME");
      System.out.println("========================");
      EkspresiWajah objEkspresi=new EkspresiWajah();
      Gembira objGembira=new Gembira();
      Sedih objSedih=new Sedih();
      EkspresiWajah[] ekspresi=new EkspresiWajah[3];
      ekspresi[0]=objEkspresi;
      ekspresi[1]=objGembira;
      ekspresi[2]=objSedih;
      System.out.println("Ekspresi[0]:"+ekspresi[0].respons());
      System.out.println("Ekspresi[1]:"+ekspresi[1].respons());
      System.out.println("Ekspresi[2]:"+ekspresi[2].respons());
   }
}
```

IV. UNGUIDED

Dalam sebuah perusahaan, terdapat banyak *employess*, diantaranya *Salaried Employee*, *Commission Employee,* dan *Project Planner*.

1. Setiap *Employee* tersebut memiliki fungsi untuk menghitung gaji dan cetak informasi
2. *Salaried Employee* memiliki atribut nama, nip, upah mingguan
3. *Commission Employee* memiliki atribut nama, nip, gaji pokok, komisi dan total penjualan
4. *Project Planner* memiliki atribut nama, nip, gaji pokok, komisi dan total hasil proyek

Setiap kelas harus memiliki konstruktor, fungsi untuk menghitung gaji setiap pegawai, dan fungsi untuk menampilkan informasi yang dimiliki setiap jenis *employee*. Fungsi hitung gajinya adalah :

- *Salaried Employee* : gaji = upah mingguan
- *Commission Employee* : gaji = gaji pokok + (komisi * total penjualan)
- *Project Planner* : gaji = gaji pokok + (komisi * total hasil proyek) - pajak. Pajak dihitung 5 % dari gaji pokok.

Buatlah program tersebut dalam bahasa Java menggunakan konsep polymorphism. Setiap kelas dibuat dalam file yang berbeda

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