INHERITANCE DAN POLYMORFISME

1.1 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. **Mengerti** dan **memahami inheritance** secara **mendalam**.
2. **Memahami** konsep **inheritance** dan **polymofisme** serta dapat **mengimplementasikan** dalam suatu **program.**

1.2 Pendahuluan

Pada dasarnya kita sebagai manusia sudah terbiasa untuk melihat objek yang berada di sekitar kita tersusun secara hierarki berdasarkan class-nya masing-masing. Dari sini kemudian timbul suatu konsep tentang pewarisan yang merupakan suatu **proses dimana** suatu **class diturunkan** dari **class lainnya** sehingga ia **mendapatkkan ciri** atau **sifat dari class** tersebut.

Dari hirarki di atas dapat dilihat bahwa, **semakin ke bawah**, **class** akan semakin **bersifat spesifik**. Misalnya, sebuah Class Pria memiliki seluruh sifat yang dimiliki oleh manusia, demikian halnya juga Wanita.

Penurunan sifat tersebut dalam Bahasa Pemrograman Java **disebut** dengan **Inheritance** yaitu satu dalam Pilar Dasar OOP (Object Oriented Programing), yang dalam implementasinya merupakan sebuah hubungan **"adalah bagian dari"** atau ***"is a relationship"*** object yang **diinherit** (diturunkan).

Latar belakang diperlukannya suatu **inheritance** dalam pemrograman java adalah untuk **menghindari duplikasi object** baik itu **field, variable** maupun **method** yang sebenarnya merupakan **object** yang bisa **diturunkan** dari hanya sebuah class. Jadi **inheritance bukan** sebuah **Class** yang **diturunkan** oleh sebuah **Literial**, tapi lebih menunjukkan ke **hubungan** antar **object** itu sendiri.

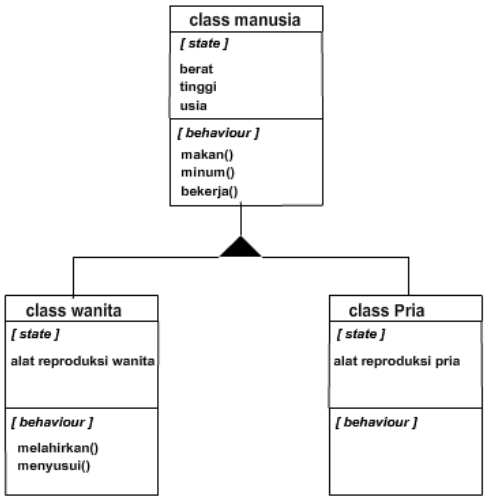
Sedangkan suatu **kemampuan** dari sebuah **object** untuk membolehkan **mengambil** beberapa **bentuk** yang **berbeda** agar **tidak** terjadi **duplikasi object** kita kenal sebagai **polymorphism.**

Antara **penurununan sifat** (inheritance) maupun **polymorphism** merupakan **konsep** yang **memungkinkan** digunakannya suatu **interface** yang **sama** untuk **memerintah objek** agar melakukan **aksi** atau **tindakan** yang **mungkin** secara **prinsip sama** namun secara **proses berbeda**. Dalam konsep yang lebih umum sering kali polymorphism disebut dalam istilah tersebut.

1.3 Inheritance

Inheritance merupakan suatu **cara** untuk **menurunkan** suatu **class** yang lebih **umum menjadi** suatu class yang lebih **spesifik**. **Inheritance** adalah salah satu **ciri utama** suatu bahasa program **yang berorientasi** pada **objek**, dan Java pasti menggunakannya. Java mendukung **inheritance** dengan memungkinkan **satu kelas** untuk **bersatu** dengan **kelas lainnya** ke **dalam deklarasinya**. Dalam inheritance terdapat dua istilah yang sering digunakan. Kelas yang menurunkan disebut kelas dasar (base class/super class), sedangkan kelas yang diturunkan disebut kelas turunan (derived class/sub class). **Karakteristik** pada **super class** akan **dimiliki** juga oleh **subclassnya**.

Manusia adalah **superclass** dari **pria** dan **wanita**. **Pria** dan **wanita mewarisi state** dan **behaviour** class **manusia**.



Gambar 0-1 Ilustrasi pada class manusia

Contoh pewarisan dalam program java:

```
class A
{
    int x;
    int y;

    void tampilXY()
    {
        System.out.println("nilai x: "+x+" nilai y: "+y);
```

```
    }
}

class B extends A
{
    int z;
    void jumlahXY()
    {
        System.out.println("jumlah: "+(x+y+z));
    }
}

public class DemoInheritance
{
    public static void main(String[] args)     {
    A superclass=new A();
    B subclass=new B();
    System.out.println("superclass :");

    superclass.x=3;
    superclass.y=4;
    superclass.tampilXY();
    System.out.println("subclass :");

//member superclass dapat diakses dari subclassnya
    subclass.x=1;
    subclass.y=2;
    subclass.tampilXY();

//member tambahan hanya ada di subclass
    subclass.z=5;
    subclass.jumlahXY();
    }
}
```

1.4 Polymorfisme

Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk. Dalam PBO, konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah obyek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda.

Dalam konsep yang lebih umum sering kali polymorphism disebut dalam istilah satu interface banyak aksi. Contoh yang konkrit dalam dunia nyata yaitu mobil.

Mobil yang ada dipasaran terdiri atas berbagai tipe dan berbagai merk, namun semuanya memiliki interface kemudi yang sama, seperti: stir, tongkat transmisi, pedal gas dan rem. Jika seseorang dapat mengemudikan satu jenis mobil saja dari satu merk tertentu, maka orang itu akan dapat mengemudikan hamper semua jenis mobil yang ada, karena semua mobil tersebut menggunakan interface yang sama.

Interface yang **sama tidak berarti** cara **kerjanya juga sama**. Misal pedal gas, jika **ditekan** maka **kecepatan mobil** akan **meningkat,** tapi proses peningkatan kecepatan ini berbeda-beda untuk setiap jenis mobil. Dengan **OOP** maka dalam **melakukan pemecahan** suatu **masalah** kita **tidak melihat bagaimana cara menyelesaikan** suatu **masalah** tersebut (terstruktur) tetapi **obyek-obyek** apa yang dapat **melakukan pemecahan masalah** tersebut.

Sebagai contoh anggap kita memiliki departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager ingin memperoleh data dari bag administrasi maka manager tersebut dapat menyuruh petugas bag administrasi untuk mengambilnya. Pada kasus tersebut, manager tidak harus mengetahui bagaimana cara mengambil data tetapi manager bisa mendapatkan data tersebut melalui obyek petugas administrasi.

Jadi untuk **menyelesaikan suatu masalah** dengan **kolaborasi** antar **obyek-obyek** yang ada **karena setiap obyek memiliki deskripsi tugasnya** sendiri.

1.4.1 Overloading method

Di dalam java Anda diperbolehkan **membuat dua** atau **lebih method** dengan **nama** yang **sama** dalam **satu class**, tetapi **jumlah** dan **tipe data argumen** masing – masing method **haruslah berbeda** satu dengan yang lainnya. Hal itu yang dinamakan dengan **Overloading Method**.Contoh program:

```
class Lagu {
    String pencipta;
    String judul;

    void IsiParam(String param1) {
        judul    = param1;
        pencipta = "Tidak dikenal";
    }

    void IsiParam(String param1,String param2) {
       judul     = param1;
       pencipta  = param2;
    }

    void CetakKeLayar() {
        System.out.println("Judul : " + judul + ", pencipta : " + pencipta);
    }
}
class DemoOver2 {
public static void main(String[] args) {
       Lagu d,e;
       d = new Lagu();
       e = new Lagu();

       d.IsiParam("Lagu 1");
       e.IsiParam("kepastian yang kutunggu","GiGi");


       d.CetakKeLayar();
       e.CetakKeLayar();
    }
}
```

Pada contoh program di atas, kita melakukan **overloading** terhadap **method IsiParam()**. Di mana **IsiParam()** yang pertama **menerima satu** buah **parameter** bertipe **String** dan **IsiParam()** yang **kedua menerima dua** buah parameter bertipe **String**. Sehinnga inilah hasil eksekusi program yang didapat:

```
Judul : Lagu 1, pencipta : Tidak dikenal
Judul : kepastian yang kutunggu: GiGi
```

1.4.2 Overriding method

Di dalam java, jika dalam suatu **subclass** Anda **mendefinisikan** sebuah **method** yang **sama dengan** yang dimiliki oleh **superclass**, maka method yang Anda buat dalam subclass tersebut dikatakan meng-**override superclass**-nya. Sehingga jika Anda mencoba **memanggil** method tersebut **dari** instance **subclass** yang Anda buat, maka **method** milik **subclass**-lah yang akan **dipanggil**, **bukan** lagi **method** milik **superclass**-nya.Contoh program:

```
class A {
    public void tampilkanKeLayar() {
        System.out.println("Method milik class A dipanggil...");
    }
}
class B extends A {
    public void tampilkanKeLayar() {
        super.tampilkanKeLayar(); // method milik superclas dipanggil
        System.out.println("Method milik class B dipanggil...");
    }
}
class DemoInheritance {
    public static void main(String[] args) {
        B subOb = new B();
        subOb.tampilkanKeLayar();
```

```
    }
}
```

Output program diatas:

```
Method milik class A dipanggil...
Method milik class B dipanggil...
```

Terlihat pada contoh di atas, **class *B*** yang merupakan **turunan** dari **class *A***, meng-**override** method ***tampilkanKeLayar(),*** sehingga pada waktu kita **memanggil method** tersebut dari variable instance class ***B*** **(variable *subOb*),** yang terpanggil adalah **method** yang sama yang ada pada **class *B*.**

1.5 Referensi Variabel Casting

Terdapat 2 tipe variabel reference yaitu **Downcasting** dan **Upcasting.**

Downcasting: apabila terdapat **variabel reference** yang **mengacu** pada **subtipe objek**, maka dapat **dimasukkan** ke dalam **subtipe variabel reference**. Dengan kata lain **downcasting** hanya **dapat dilakukan** bila antara **type** dari **object** dan **type** dari **variabel reference** masih **dalam** sebuah **inheritance.** Bila ada polymorphisme sebagai berikut:

```
A ref = new B();

// Dimana B adalah subclass dari A, maka dari variabel ref kita hanya akan dapat meng-invoke (menjalankan) method-method yang dideklarasikan di kelas A. Bila kita ingin meng-invoke method yang hanya terdapat di kelas B, maka kita tidak dapat secara langsung memanggil method yang hanya ada di B seperti sebagai berikut:

ref.methodDiB();     //COMPILE TIME ERROR !!!!
```

Potongan program di atas akan menghasilkan compile time error sbb:

```
Cannot find symbol
```

Agar dapat **meng-invoke method** yang ada di **kelas B**, kita dapat menggunakan **casting** seperti sebagai berikut:

```
//cara 1
B b = (B) ref;     //CASTING !!!!
b.methodDiB();

//Atau :

//cara 2
((B)ref).methodDiB();   //CASTING !!!!
```

Contoh :

```
class A
{
    void methodDiA()
    {
        System.out.println("A.methodDiA()");
    }
}
class B extends A
{
    void methodDiB()
    {
        System.out.println("B.methodDiB()");
    }
}
class Polymorphisme01
{
    public static void main(String[] args)
    {
        A ref = new B();

        //cara 1
        B b = (B) ref;
        b.methodDiB();

        //cara 2
        ((B)ref).methodDiB();
```

```
        }
}
```

Program di atas akan menghasilkan :

```
B.methodDiB()
B.methodDiB()
```

Downcasting hanya dapat dilakukan bila **antara type** dari **object** dan **type** dari **variabel reference** masih dalam sebuah **inheritance**. Bila tidak, maka akan terjadi compile time error (inconvertible types). Contoh:

```
class A {}
class B extends A {}
class C {}

class Test
{
    void lakukanSesuatu()
    {
        A ab = new B();
        B b = (B) ab;   //Downcast berhasil
        C c = (C) ab;   //Downcast GAGAL
    }
}
```

Compile time error yang dihasilkan adalah :

```
inconvertible types
```

Ada kemungkinan **downcast tidak error** saat **compile time**, akan tetapi **melemparkan** **`ClassCastException`**. Hal ini terjadi karena **type** dari **object** yang **diassign** ke suatu **variabel reference** adalah **superclass** dari **type** dari **variabel references** tersebut. Contoh:

```
class A {}
class B extends A {}

class Test
{
    public static void main(String[] args)
    {
        Test t = new Test();
        t.lakukanSesuatu();
    }
    void lakukanSesuatu()
    {
        A a = new A();
        /*
         *Statement di bawah dapat dicompile, akan tetapi
         *saat dijalankan akan melemparkan exception
         *ClassCastException !!!!
         */
        B b = (B) a;
    }
}
/*
Saat dicompile berhasil !!! (dengan javac)
Saat dijalankan (dengan java) akan menghasilkan :
java.lang.NoClassDefFoundError: inheritance/VarRefCast03
Exception in thread "main"
Java Result: 1
*/
```

Kesimpulan untuk downcast adalah :

1. Downcast **pasti explisit.**
2. Downcast akan **berhasil** saat **compile time** bila antara **object** dengan **variabel reference** masih dalam **1** buah **jalur inheritance**.
3. Downcast akan **gagal** saat **runtime** (melemparkan exception) bila ternyata **type** dari **object** yang **diassign ke variabel reference** ternyata adalah **superclass** dari **variabel reference** itu.

Upcasting: **mengisikan variabel reference** ke **variabel reference lainnya** dengan **tipe superclassnya**.

Upcasting bisa dilakukan dengan dua cara :

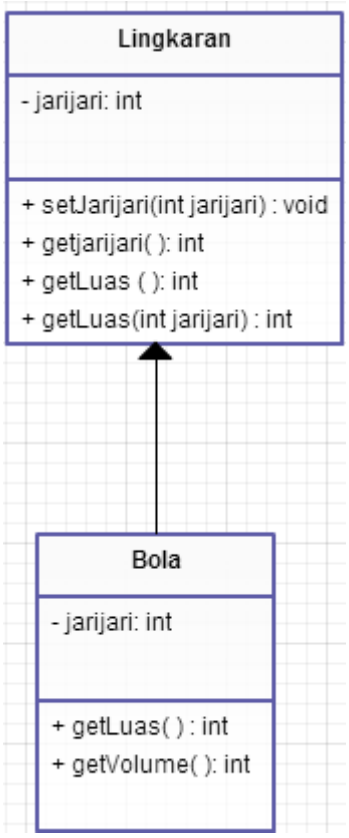
1. **Implisit upcasting** (otomatis)
2. **Eksplisit upcasting**

```
Contoh : [1]
Program 08

class A {}
class B extends A {}

class VarRefCast01
{
    public static void main(String[] args)
    {
        B b = new B();
        //implisit upcasting
        //dari variabel reference ke variabel reference lainnya
        A ab1 = b;
        //implisit upcasting
        //dari suatu object ke variabel reference
        A ab2 = new B();
        //explisit upcasting
        //dari variabel reference ke variabel reference lainnya
        A ab3 = (A) b;
        //explisit upcasting
        //dari object kel variabel reference
        A ab4 = (A) new B();
    }
}
```

Perhatikan kelas diagram disamping. Jawablah pertanyaan dibawah ini!



1. Sebutkan method mana yang melakukan override?

2. Sebutkan method mana yang melakukan overload?

3. Buatlah syntax yang mengaplikasikan kelas diagram tersebut.