IF34348 - PEMROGRAMAN LANJUT

OBJECT ORIENTED PROGRAMMING (OOP)

06

Oleh : Andri Heryandi, M.T.

DEFINISI

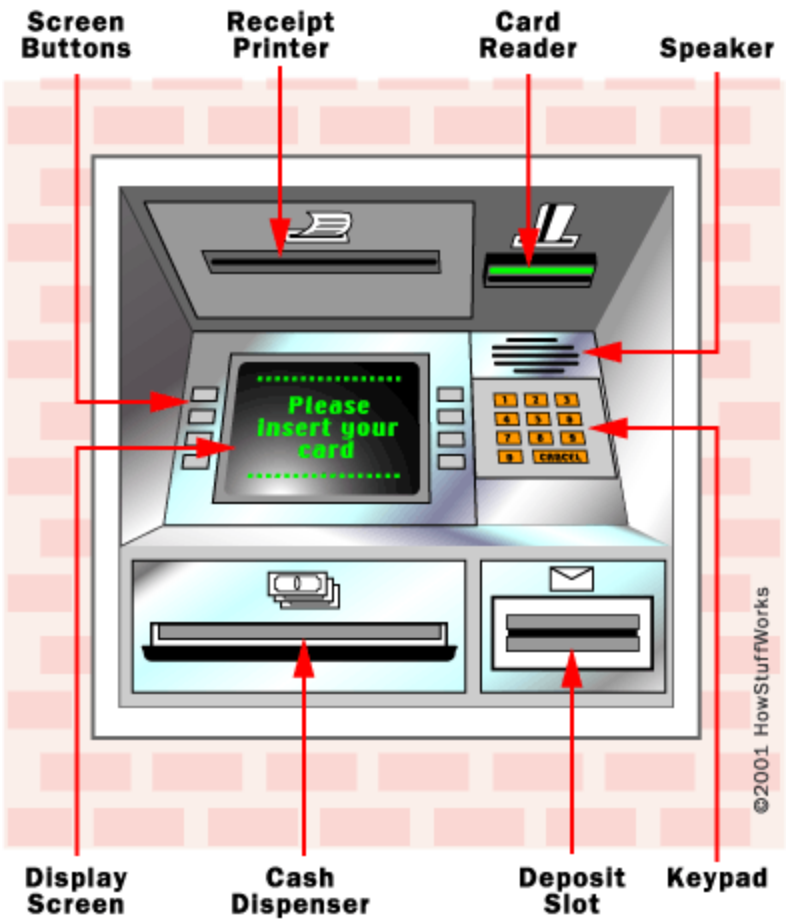
- **Object Oriented Programming (OOP) / Pemrograman Berbasis Objek (PBO)** adalah konsep pemrograman yang menerapkan konsep objek, dimana objek terdiri dari atribut (informasi-informasi mengenai objek) dan method (prosedur/proses) yang bisa dilakukan oleh objek tersebut.
- Software/Perangkat lunak terdiri dari objek-objek yang saling berinteraksi.

CONTOH

- Contoh :
  - Object : Manusia
    - Attribut : Nama, tinggi, umur, berat badan dll
    - Method : Makan, Minum, Berjalan, Bekerja
  - Object : Windows
    - Attribut : Left, Top, Width, Height, BackgroundColor,
    - Method : OnClick, OnClose, OnDoubleClick
  - Object : Keluarga
    - Attribut : ayah, Ibu, Anak[]
    - Method : TambahAnak,

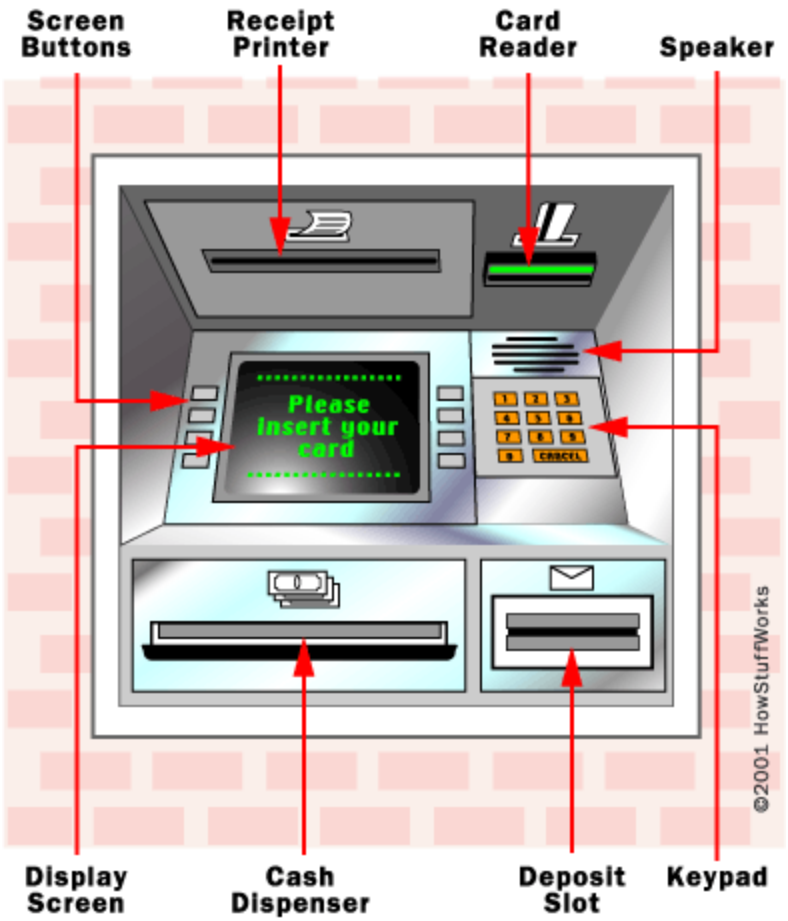
CONTOH

- Perhatikan mesin ATM
- Sebuah mesin ATM terdiri dari elemen-elemen berikut :
  - Display Screen (Layar)
  - Screen Button (Tombol)
  - Receipt Printer
  - Card Reader
  - Speaker
  - Cash Dispenser
  - Deposit Slot
  - Keypad
- Di dalam konsep OOP, setiap setiap elemen tersebut berinteraksi dengan mengirimkan pesan (message) tertentu.



CONTOH

- Interaksi ketika penekanan tombol Ambil Uang Rp. 200.000
  - Speaker mengeluarkan bunyi beep.
  - CashDispenser mengeluarkan uang 200.000
  - Jika uang diambil, maka Receipt Printer mencetak faktur,
  - Jika uang tidak diambil, Uang dimasukkan kembali ke Cash Dispenser,
  - Layar kembali ke menu Utama.



FITUR-FITUR OOP

- **Encapsulation**

  Penggabungan antara data (attribut) dengan prosedure (method) yang mengolahnya.

- **Inheritance**

  Penurunan sifat (attribut dan method) dari Class Parent (SuperClass) ke Class Child (SubClass). Ini menandakan bahwa OOP mendukung konsep code reuse dimana data-data yang ada di class parent bisa di kenal di kelas child.

- **Polymorphism**

  Sebuah kemampuan dari sebuah objek untuk bekerja dalam berbagai bentuk. Penggunaan umum polymorphism biasanya digunakan ketika sebuah reference dari class parent digunakan untuk mengacu ke class child.

CLASS DAN OBJECT

- Class

  Class adalah cetak biru/prototipe/pendefinisian dari suatu benda. Didalam class-lah attribut dan method suatu object didefinisikan.

  Contoh : Manusia, Window

- Object

  Object adalah bentuk instance/nyata/real/hidup dari sebuah class.

  Contoh :

  - Shelly:Manusia (Object Shelly mempunyai Class Manusia)
  - Form1:Window (Object Form1 mempunyai class Window)

CLASS DAN OBJECT

- Setiap object pasti memiliki class (sebagai templatenya)
- Setiap object harus diinstansiasi/dihidupkan terlebih dahulu sebelum digunakan. Instansiasi sebuah objek dapat dilakukan dengan keyword new. Contoh berikut :

```
NamaClass NamaObject;
NamaObject=new NamaClass(parameter_konstruktornya);
```

- Untuk mengakses attribut atau method suatu object, gunakan tanda titik setelah nama objeknya.

```
Kucing catty=new Kucing("Catty");
catty.warna="putih";
catty.jalan();
```

Mengakses atribut/method

CONTOH PENGGUNAAN CLASS

- String di java sebenarnya adalah sebuah Class.
- Method-method yang ada di class String
  - charAt(index) : Mereturnkan huruf pada posisi ke-index. Index 0 menunjukkan huruf pertama.
  - equals(string_lain) : Mereturnkan true jika isi string sama dengan isi string_lain (case sensitive).
  - equalsIgnoreCase(string_lain) : Mereturnkan true jika isi string sama dengan string lain dengan mode perbandingan case insensitive.
  - length() : Mereturnkan berapa banyak huruf dalam string.
  - toUpperCase() : Mereturnkan string yang berisi bentuk kapital dari stringnya.
  - toLowerCase() : Mereturnkan string yang berisi bentuk huruf kecil dari stringnya.

CONTOH PENGGUNAAN CLASS


```
public class TestString {
    public static void main(String[] args) {
        String nama="Universitas Komputer Indonesia";
        System.out.println("ISI STRING : "+nama);
        System.out.println("Panjang    : "+nama.length());
        System.out.println("Upper Case : "+nama.toUpperCase());
        System.out.println("Lower Case : "+nama.toLowerCase());
        System.out.println("=UNIKOM    : "+nama.equals("UNIKOM"));
    }
}
```

```
ISI STRING : Universitas Komputer Indonesia
Panjang    : 30
Upper Case : UNIVERSITAS KOMPUTER INDONESIA
Lower Case : universitas komputer indonesia
=UNIKOM    : false
```

MEMBUAT CLASS SEDERHANA

- **Sintak pembuatan class sederhana**

```
class NamaKelas{
    tipe_data nama_atribut;
    tipe_data nama_atribut;
    nama_kelas nama_object;

    NamaKelas(parameter){
       … // isi konstruktor
    }
    void nama_method(parameter){
        … // isi method berbentuk procedure
    }
    tipe_data nama_method_function(paramter){
       … // isi method berbentuk function
       return ….;
    }
}
```

Daftar Attribut

Daftar Method (Procedure/Function /Constructor)

CONTOH CLASS SEDERHANA

- Buatlah sebuah class bernama Titik yang digunakan untuk menyimpan sebuah titik koordinat.
  - Setiap titik mempunyai atribut posisi X dan attribut posisi Y.
  - Class ini harus dapat melakukan hal berikut :
    - Memberikan nilai default (X=0, Y=0) ketika X dan Y belum didefinisikan.
    - Mengisi Nilai X
    - Mengisi Nilai Y
    - Mengisi Nilai X dan Y (sekaligus)
    - Menampilkan nilai X dan Y
    - Pindah ke Koordinat Lain (berdasarkan jarak atau ke titik tertentu)
    - Menghitung Jarak Ke Titik Lain

CONTOH CLASS SEDERHANA


```
class Titik {
        double x;
        double y;
        Titik(){
                x=0;
                y=0;
        }
        Titik(double x1,double y1){
                x=x1;
                y=y1;
        }
        void tampil(){
                System.out.println("("+x+","+y+")");
        }
        void pindah(double x1, double y1){
                x=x1;
                y=y1;
        }
```

CONTOH CLASS SEDERHANA


```
void pindah(Titik t){
        x=t.x;
        y=t.y;
}
void isiX(double x1){
        x=x1;
}
void isiY(double y1){
        y=y1;
}
void isiXY(double x1,double y1){
        x=x1;
        y=y1;
}
```

CONTOH CLASS SEDERHANA


```
        double jarakKe(double x1, double y1){
                double jarak;
                jarak=Math.pow(Math.pow(x-x1,2)+Math.pow(y-
y1,2),0.5);
                return jarak;
        }
        double jarakKe(Titik t2){
                double jarak;
                jarak=Math.pow(Math.pow(x-t2.x,2)+Math.pow(y-
t2.y,2),0.5);
                return jarak;
        }
}
```

Math.pow(x,y) = x^y

MEMBUAT KELAS TESTER

- Kelas tester digunakan sebagai class untuk melakukan uji coba terhadap class yang telah dibuat.
- Sebaiknya kelas tester dibuat terpisah dari file classnya.

MEMBUAT KELAS TESTER


```
public class TitikTester {
  public static void main(String[] args) {
       Titik t1,t2;
       t1=new Titik();// Tanpa parameter x=0, y=0
       t2=new Titik(9,4);// x=9, y=4
       System.out.print("T1 : ");
       t1.tampil();
       System.out.print("T2 : ");
       t2.tampil();
       t1.pindah(5,1);
       System.out.print("Setelah pindah T1 : ");
       t1.tampil();
       double jarak;
       jarak=t1.jarakKe(t2);
       System.out.println("Jarak dari T1 ke T2 : "+jarak);
  }
}
```

```
T1 : (0.0,0.0)
T2 : (9.0,4.0)
Setelah pindah T1 : (5.0,1.0)
Jarak dari T1 ke T2 : 5.0
```

CLASS CALCULATOR

- Buatlah class Calculator
  - Attribut :
    - Operan1 bertipe double
    - Operan2 bertipe double
  - Method :
    - isiOperan1(double x) : Mengisi atribut operan1 dengan nilai x
    - isiOperan2(double x) : Mengisi atribut operan2 dengan nilai x
    - tambah() : Mereturnkan nilai Operan1 + Operan2
    - kurang() : Mereturnkan nilai Operan1 – Operan2
    - kali() : Mereturnkan nilai Operan1 * Operan2
    - bagi() : Mereturnkan nilai Operan1 / Operan2
    - pangkat() : Mereturnkan nilai Operan1^Operan2
- Kelas tersebut harus bisa dijalankan dengan menggunakan Class CalculatorTester (di slide berikutnya)

CLASS CALCULATORTESTER


```
public class CalculatorTester {
      public static void main(String[] args) {
            Calculator c=new Calculator();
            c.isiOperan1(7);
            c.isiOperan2(5);
            System.out.println("Tambah  : "+c.tambah());
            System.out.println("Kurang  : "+c.kurang());
            System.out.println("Kali    : "+c.kali());
            System.out.println("Bagi    : "+c.bagi());
            System.out.println("Pangkat : "+c.pangkat());
      }
}
```

```
Tambah  : 12.0
Kurang  : 2.0
Kali    : 35.0
Bagi    : 1.4
Pangkat : 16807.0
```

CLASS NILAI

- Buatlah class Nilai
  - Attribut :
    - Quis bertipe double
    - UTS bertipe double
    - UAS bertipe double
  - Method :
    - setQuis(double x) : Mengisi nilai quis
    - setUTS(double x) : Mengisi nilai UTS
    - setUAS(double x) : Mengisi nilai UAS
    - getNA() : Mereturnkan nilai akhir berupa double
    - getIndex(): Mereturnkan index berupa char
    - getKeterangan() : Mereturnkan keterangan berupa String

- Rumus NilaiAkhir

NA = 20% QUIS + 30%*UTS + 50% * UAS

- Aturan Index
  - NA 80..100 Index='A'
  - NA 68..80 Index='B'
  - NA 56..68 Index='C'
  - NA 45..56 Index='D'
  - NA 0..45 Index='E'
- Aturan Keterangan
  - Index='A' : Sangat Baik
  - Index='B' : Baik
  - Index='C' : Cukup
  - Index='D' : Kurang
  - Index='E' : Sangat Kurang

CLASS NILAITESTER


```
public class NilaiTester {
  public static void main(String[] args) {
        Nilai n=new Nilai();
        n.setQuis(60);
        n.setUTS(80);
        n.setUAS(75);
        System.out.println("Quis        : "+n.Quis);
        System.out.println("UTS         : "+n.UTS);
        System.out.println("UAS         : "+n.UAS);
        System.out.println("NA          : "+n.getNA());
        System.out.println("Index       : "+n.getIndex());
        System.out.println("Keterangan  : "+n.getKeterangan());
  }
}
```

```
Quis        : 60.0
UTS         : 80.0
UAS         : 75.0
NA          : 73.5
Index       : B
Keterangan  : Baik
```

PART II

06

PACKAGE

- Package adalah sekumpulan class/interface/enumerasi yang berelasi.
- Package menyediakan mekanisme untuk mengatur *class* dan *interface* dalam jumlah banyak dan menghindari konflik pada penamaan
- 2 buah class yang mempunyai nama yang sama (tetapi isinya beda) dapat digunakan sekaligus jika berada di masing-masing package.
- Jika anda membuat class tanpa dimasukkan ke dalam package, maka secara default akan dianggap sebagai unnamed package (package tanpa nama). Package tanpa nama tidak bisa diimport oleh class dari package yang mempunyai nama.

MEMBUAT PACKAGE

- Jika anda ingin sebuah class berada dalam suatu package, maka didalam file classnya tambahkan baris berikut

  `package <namapackage>`

- Penamaan package sebaiknya menggunakan huruf kecil semua.
- Nama package harus disesuaikan dengan nama folder tempat menyimpan class tersebut.

```
package mylib;
class Titik {
        double x;
        double y;
}
```

ATURAN PENAMAAN PACKAGE

- Nama package sebaiknya ditulis dengan huruf kecil.
- Nama package harus sama dengan nama folder.
- Suatu file kode sumber hanya boleh menuliskan 1 statement package.
- Perusahaan besar pengembang software biasanya memberikan nama class sesuai dengan nama domainnya. Contoh :
  - Facebook : package com.facebook.android
  - Twitter : package com.twitter.interop
  - Google maps : package com.google.android.maps

IMPORT PACKAGE

- Import package berarti anda ingin menggunakan member/isi dari sebuah package.
- Import suatu package dilakukan dengan keyword import.

```
import namapackage.namaclass; // import 1 class
```

atau

```
import namapackage.*; // import seluruh isi package.
```

- Contoh :

```
import mylib.Titik;
```

Atau

```
import mylib.*;
```

- Secara default, jika anda tidak mengimport satu package-pun maka sebenarnya anda mengimport package java.lang.* yang contoh class yang ada di dalamnya adalah class String dan Math. Inilah yang membuat anda tidak usah import package apa pun ketika akan menggunakan class String atau Math.

IMPORT PACKAGE

- Jika aplikasi yang anda buat menggunakan 2 buah class yang namanya sama tetapi berpada package yang berbeda maka sebagai pembeda anda harus menuliskan nama packagenya.
  - Contoh berikut akan menampilkan cara pembuatan 2 objek yang mempunyai nama class yang sama.

```
package1.Titik t1=new package1.Titik();
package2.Titik t2=new package2.Titik();
```

ACCESS MODIFIER

- Pengaturan akses terbagi menjadi 2 level,
  - Di level kelas

    Pengaturan akses di level kelas terdiri dari 2 cara yaitu
    1. Tanpa menuliskan keyword apa pun (default/package-private)
    2. Menuliskan keyword `public`
  - Di level member

    Pengaturan akses di level member terdiri dari 4 cara yaitu :
    1. Tanpa menulis keyword apa pun (default/package-private)
    2. Menuliskan keyword `public`
    3. Menuliskan keyword `private`
    4. Menuliskan keyword `protected`

PENGATURAN AKSES DI LEVEL KELAS

- Pengaturan akses di level kelas hanya memiliki 2 kemungkinan yaitu :
  - Default/Package-Private :

    Jika sebuah class dibuat dengan menggunakan akses ini maka class tersebut hanya akan dikenal di class-class pada package yang sama saja.
  - Public

    Jika sebuah class dibuat dengan menggunakan akses public maka class tersebut dapat dikenal di mana pun, baik dari dalam package yang sama atau dari luar package (world).

PENGATURAN AKSES DI LEVEL KELAS

- Akses : Default/Package-Private

```
// Nama File : package1/Titik.java
package package1;
class Titik {
        public Titik(){
                System.out.println("Titik di package 1");
        }
}
```

Kedua kelas berada pada package yang sama

√

```
// Nama File : package1/Tester.java
package package1;
class Tester {
        public static void main(String[] args) {
                Titik t1=new Titik();
        }
}
```

PENGATURAN AKSES DI LEVEL KELAS

- Akses : Default/Package-Private

```
// Nama File : package1/Titik.java
package package1;
class Titik {
        public Titik(){
                System.out.println("Titik di package 1");
        }
}
```

Class Titik berada di package bernama package1, Class Tester tidak memiliki package

X

```
// Nama File : Tester.java
import package1.Titik;
class Tester {
        public static void main(String[] args) {
                Titik t1=new Titik();
        }
}
```

PENGATURAN AKSES DI LEVEL KELAS

- Akses : public

```
// Nama File : package1/Titik.java
package package1;
public class Titik {
        public Titik(){
                System.out.println("Titik di package 1");
        }
}
```

Kedua kelas berada pada package yang sama

√

```
// Nama File : package1/Tester.java
package package1;
class Tester {
        public static void main(String[] args) {
                Titik t1=new Titik();
        }
}
```

PENGATURAN AKSES DI LEVEL KELAS

- Akses : public

```
// Nama File : package1/Titik.java
package package1;
public class Titik {
        public Titik(){
                System.out.println("Titik di package 1");
        }
}
```

Class Titik berada di package bernama package1, Class Tester tidak memiliki package

√

```
// Nama File : Tester.java
import package1.Titik;
class Tester {
        public static void main(String[] args) {
                Titik t1=new Titik();
        }
}
```

PENGATURAN AKSES DI LEVEL MEMBER

- Pengaturan akses di level member memiliki 4 kemungkinan yaitu :
  - Public

    Jika member memiliki access modifier ini, maka member ini bisa diakses dari mana saja (world) yaitu dari class sendiri atau dari class-class lain (walaupun berada di package yang berbeda). Member ini dikenal di subclassnya
  - Protected

    Jika member memiliki access modifier ini, maka member ini bisa diakses dari class sendiri atau dari class-class lain yang se-package. Member ini bisa diakses di subclassnya (class turunannya)
  - Default/Package-Private :

    Jika member memiliki access modifier ini, maka member ini hanya bisa diakses dari class sendiri dan dari class-class pada package yang sama. Member ini tidak dikenal di subclassnya (class turunan).
  - Private

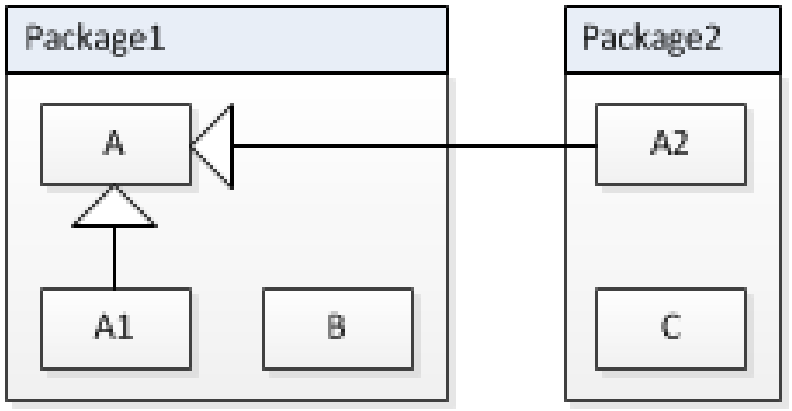
    Jika member memiliki access modifier ini, maka member ini bisa diakses dari class sendiri saja. Member ini tidak bisa diakses dari class lain walaupun se-package ataupun dari subclassnya (class turunannya)

PENGATURAN AKSES DI LEVEL MEMBER

Access Levels				
Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	T
tanpa modifier	Y	Y	T	T
private	Y	T	T	T

Keterangan :
Y : Bisa diakses
T : Tidak bisa diakses

PENGATURAN AKSES DI LEVEL MEMBER



- Visibilitas member-member class A dari class lain.

Modifier Member A	A	A1	B	A2	C
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	Y	T
tanpa modifier	Y	Y	Y	T	T
private	Y	T	T	T	T

ACCESS MODIFIER

Kenapa harus memikirkan Access Modifier?

Alasannya adalah

- Membatasi hak akses. Ini dilakukan agar sebuah member tidak sembarangan dibaca atau diisi
- Menyembunyikan informasi. Pengguna class tidak harus tahu apa yang ada/terjadi di dalam class (Information Hiding)

CONTOH KASUS ACCESS MODIFIER

- Buatlah sebuah class bernama Nilai yang akan menampung nilai Quis, UTS, UAS. Nilai yang dapat diterima adalah antara 0 sampai 100. Setiap nilai dapat diubah nilainya. Class ini harus dapat diakses dari class mana pun.

```
public class Nilai {
        public double Quis;
        public double UTS;
        public double UAS;
        public double getNA(){
                return 0.20*Quis+0.30*UTS+0.5*UAS;
        }
}
```

CONTOH KASUS ACCESS MODIFIER

- Contoh pemanggilan class Nilai

```
public class NilaiTester {
        public static void main(String[] args) {
                Nilai n=new Nilai();
                n.Quis=90;
                n.UTS=70;
                n.UAS=150;
                System.out.println("NA : "+n.getNA());
        }
}
```

- Jika aplikasi di atas dieksekusi maka menghasilkan nilai akhir sebesar 114.0. Apakah ini legal???????. Hal ini terjadi karena atribut UAS bersifat public sehingga bisa diisi secara bebas dari luas class.

CONTOH KASUS ACCESS MODIFIER

- Solusinya adalah dengan membuat atribut Quis, UTS dan UAS tidak bisa diisi secara langsung (misalnya dengan private atau protected.

```
public class Nilai {
        private double Quis;
        private double UTS;
        private double UAS;
        public double getNA(){
                return 0.20*Quis+0.30*UTS+0.5*UAS;
        }
}
```

Tidak, dengan hanya mengubah public menjadi private, maka pengisian nilai Quis, UTS dan UAS tidak bisa dilakukan. Solusinya adalah dengan

CONTOH KASUS ACCESS MODIFIER

- Solusinya adalah dengan membuat atribut Quis, UTS dan UAS tidak bisa diisi secara langsung (misalnya dengan private atau protected.

```
public class Nilai {
        private double Quis;
        private double UTS;
        private double UAS;
        public double getNA(){
                return 0.20*Quis+0.30*UTS+0.5*UAS;
        }
}
```

- Cukup? Tidak. Karena atribut Quis, UTS dan UAS tidak bisa diakses dari mana pun.

CONTOH KASUS ACCESS MODIFIER (METHOD SETTER DAN GETTER)

- Karena atribut Quis, UTS, dan UAS mempunyai akses private, maka class harus menyediakan suatu cara agar bisa mengakses (mengisi/membaca) nilai atribut tersebut.
- Solusinya adalah method setter dan getter.
  - Method getter adalah method yang digunakan sebagai perantara untuk **mengambil** nilai atribut yang tidak bisa diakses (karena private atau protected). Method getter biasanya berupa function yang mereturnkan tipe data sesuai atribut yang diambil.
  - Method setter adalah method yang digunakan sebagai perantara untuk **mengisi** nilai atribut yang tidak bisa diakses (karena private atau protected). Method setter biasanya berupa procedure (void function) yang mempunyai parameter input yang bertipe data sama dengan tipe data atributnya.
- Method setter dan getter seharusnya mempunyai access modifier public (karena harus bisa diakses oleh semua class).

CONTOH KASUS ACCESS MODIFIER (METHOD SETTER DAN GETTER)

- **Method setter dan getter untuk Quis, UTS, UAS.**

```
public class Nilai {
      private double Quis;
      private double UTS;
      private double UAS;
      public void setQuis(double x){
            Quis=x;
      }
      public void setUTS(double x){
            UTS=x;
      }
      public void setUAS(double x){
            UAS=x;
      }
```

Function setter

CONTOH KASUS ACCESS MODIFIER (METHOD SETTER DAN GETTER)

- Method setter dan getter untuk Quis, UTS, UAS.

```
        public double getQuis(){
                return Quis;
        }
        public double getUTS(){
                return UTS;
        }
        public double getUAS(){
                return UAS;
        }
        public double getNA(){
                return 0.20*Quis+0.30*UTS+0.5*UAS;
        }
}
```

Function getter

CONTOH KASUS ACCESS MODIFIER (METHOD SETTER DAN GETTER)

- Contoh pemanggilan class Nilai berubah, karena pengisian atribut dilakukan melalui function.

```
public class NilaiTester {
        public static void main(String[] args) {
                Nilai n=new Nilai();
                n.setQuis(90);
                n.setUTS(70);
                n.setUAS(150);
                System.out.println("NA : "+n.getNA());
        }
}
```

- Beres? Belum, ternyata walau pun atribut telah diberi akses private, ternyata nilai atribut bisa diberi nilai yang tidak seharusnya. Padahal nilai hanya boleh antara 0 s.d 100.

CONTOH KASUS ACCESS MODIFIER (METHOD SETTER DAN GETTER)

- Solusinya adalah dengan membuat validasi pada method getter. Nilai atribut hanya akan berubah jika nilai yang diinput valid.
- Perubahan method getter.

```
public void setQuis(double x){
        if(x>=0 && x<=100)
                Quis=x;
}
public void setUTS(double x){
        if(x>=0 && x<=100)
                UTS=x;
}
public void setUAS(double x){
        if(x>=0 && x<=100)
                UAS=x;
}
```

CONTOH KASUS ACCESS MODIFIER (METHOD SETTER DAN GETTER)

- Contoh pemanggilan class Nilai setelah ada validasi di method getter

```
public class NilaiTester {
        public static void main(String[] args) {
                Nilai n=new Nilai();
                n.setQuis(90);
                n.setUTS(70);
                n.setUAS(150);
                System.out.println("NA : "+n.getNA());
        }
}
```

- Pada perintah n.setUAS(150), pengisian UAS menjadi 150 tidak dilakukan karena nilainya tidak valid.
- Jika dieksekusi akan menghasilkan NA : 39.0 karena atribut UAS bernilai 0

LATIHAN

- Buatlah suatu class bernama Waktu dengan spesifikasi class adalah :
  - Attribut
    - menitWaktu bertipe integer. menitWaktu akan berisi jumlah menit sejak jam 00.00.
  - Method
    - getJam() : Mengambil bagian jam dari menitWaktu.
    - getMenit() : Mengambil bagian menit dari menitWaktu.
    - setJam(int j) : Menambahkan j*60 menit ke menitWaktu tanpa mempengaruhi bagian menitnya.
    - setMenit(int m) : Menambahkan m menit ke menitWaktu tanpa mempengaruhi bagian jamnya.
    - getTotalMenit() : Mengambil nilai dari menitWaktu

LATIHAN

- Method (lanjutan)
  - tambahJam(int j) : Menambahkan j*60 ke menitWaktu
  - tambahMenit(int m) : Menambahkan m ke menitWaktu
  - tambahWaktu(int j, int m) : Menambahkan j*60+m ke menitWaktu
  - tampilWaktu() : Menampilkan jam dan menit.

CLASS WAKTUTESTER


```
public class WaktuTester {
  public static void main(String[] args) {
    Waktu w=new Waktu();
    System.out.println("1. Total Menit : "+w.getTotalMenit());
    w.setJam(10);
    System.out.println("2. Total Menit : "+w.getTotalMenit());
    w.setMenit(57);
    System.out.println("3. Total Menit : "+w.getTotalMenit());
    w.tampilWaktu();
    w.tambahJam(7);
    System.out.println("4. JAM : "+w.getJam()+" MENIT : "+w.getMenit());
    w.tambahMenit(40);
    System.out.println("5. JAM : "+w.getJam()+" MENIT : "+w.getMenit());
  }
}
```

```
1. Total Menit : 0
2. Total Menit : 600
3. Total Menit : 657
Waktu : 10:57
4. JAM : 17 MENIT : 57
5. JAM : 18 MENIT : 37
```

PART III

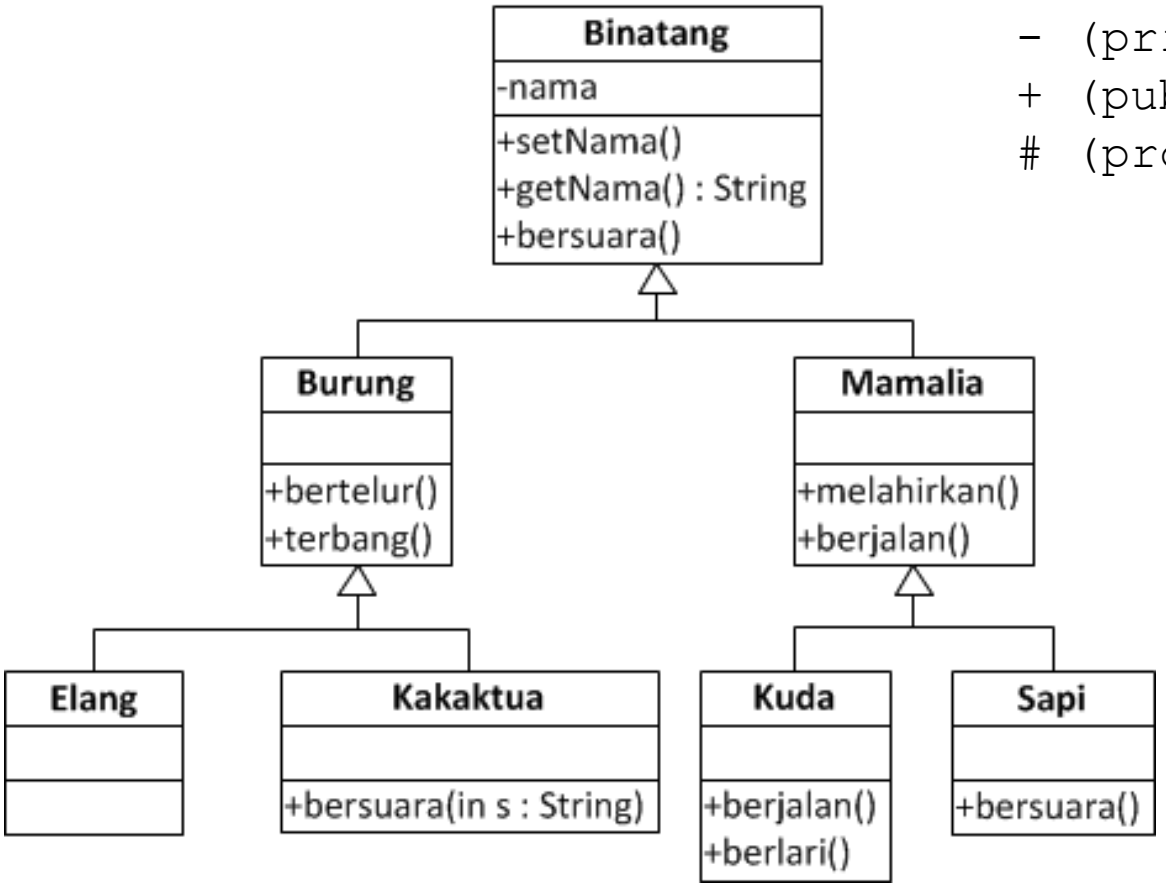
06

INHERITANCE (PEWARISAN)

- Inheritance adalah salah satu fitur utama dari OOP.
- Inheritance adalah proses dimana sebuah class mendapatkan properti (atribut + method) dari class lain.
- Dengan inheritance, sebuah class dapat dikembangkan menjadi class baru yang lebih lengkap/baik.
- Class yang diturunkan disebut sebagai superclass (*base class* atau *parent class)* dan class yang menjadi turunan disebut sebagai subclass(*derived class*, *extended class*, atau *child class)*.
- Inheritance dilakukan dengan keyword **extends**.
- Class paling atas adalah class bernama **Object**.

INHERITANCE (PEWARISAN)

- Contoh :




```
-  (private)
+  (public)
#  (protected)
```

INHERITANCE (PEWARISAN)

- Class Binatang

```
package pewarisan;

public class Binatang {
        private String nama;
        public void setNama(String n){
                nama=n;
        }
        public String getNama(){
                return nama;
        }
        public void bersuara(){
                System.out.println(getNama()+" sedang bersuara.");
        }
}
```

Setiap Binatang mempunyai **kesamaan**, yaitu mempunyai nama dan dapat bersuara.

INHERITANCE (PEWARISAN)

- Class BinatangTester

```
package pewarisan;

public class BinatangTester {
        public static void main(String[] args) {
                Binatang a;
                a=new Binatang();
                a.setNama("kuda");
                a.bersuara();
        }
}
```

Tampilan hasil run :

```
kuda sedang bersuara.
```

INHERITANCE (PEWARISAN)

- Class Burung

```
package pewarisan;

public class Burung extends Binatang {
    public void bertelur(){
        System.out.println(getNama()+" sedang bertelur");
    }
    public void terbang(){
        System.out.println(getNama()+" sedang terbang");
    }
}
```

Burung adalah Binatang (mempunyai nama dan bersuara) yang dapat bertelur dan terbang.
Method bertelur dan terbang adalah method tambahan yang dapat dilakukan oleh class Burung.

INHERITANCE (PEWARISAN)

- Class BurungTester

```
package pewarisan;

public class BurungTester {
        public static void main(String[] args) {
                Burung b;
                b=new Burung();
                b.setNama("Pipit");// diturunkan dari Binatang
                b.bersuara();      // diturunkan dari Binatang
                b.bertelur();      // ditambahkan di Burung
                b.terbang();       // ditambahkan di Burung
        }
}
```

Hasil Run :

```
Pipit sedang bersuara.
Pipit sedang bertelur
Pipit sedang terbang
```

INHERITANCE (PEWARISAN)

- Class Mamalia

```
package pewarisan;

public class Mamalia extends Binatang {
        public void melahirkan(){
                System.out.println(getNama()+" sedang melahirkan.");
        }
        public void berjalan(){
                System.out.println(getNama()+" sedang berjalan");
        }
}
```

Mamalia adalah Binatang (mempunyai nama dan bersuara) yang dapat melahirkan dan berjalan. Method melahirkan dan berjalan adalah method tambahan yang dapat dilakukan oleh class Mamalia.

INHERITANCE (PEWARISAN)

- Class MamaliaTester

```
package pewarisan;

public class MamaliaTester {
        public static void main(String[] args) {
                Mamalia m;
                m=new Mamalia();
                m.setNama("kuda"); // diturunkan dari Binatang
                m.bersuara();      // diturunkan dari Binatang
                m.berjalan();      // ditambahkan di Mamalia
                m.melahirkan();    // ditambahkan di Mamalia
        }
}
```

Hasil Run :

kuda sedang bersuara.
kuda sedang berjalan
kuda sedang melahirkan.

INHERITANCE (PEWARISAN)

- Class Elang

```
package pewarisan;

public class Elang extends Burung {

}
```

Elang adalah Burung. Asumsikan bahwa Elang tidak mempunyai perbedaan dari burung secara umum (general) [hanya punya nama dan bersuara].

INHERITANCE (PEWARISAN)

- Class ElangTester

```
package pewarisan;

public class ElangTester {
        public static void main(String[] args) {
                Elang e=new Elang();
                e.setNama("Elang");
                e.bersuara(); // Diturunkan dari Binatang
                e.bertelur(); // Diturunkan dari Burung
                e.terbang();  // Diturunkan dari Burung
        }
}
```

Hasil Run :

Elang sedang bersuara.
Elang sedang bertelur
Elang sedang terbang

INHERITANCE (PEWARISAN)

- Class Kakaktua

```
package pewarisan;

public class Kakaktua extends Burung {
        public void bersuara(String s){
                System.out.println(getNama()+" bersuara : "+s);
        }
}
```

Kakaktua adalah Burung. Tetapi cara bersuara burung kakaktua dapat meniru ucapan manusia. Oleh karena itu method bersuara yang diturunkan dari superclassnya harus dioverride(diganti) sehingga menghasilkan operasi/perilaku yang berbeda dari superclassnya.

INHERITANCE (PEWARISAN)

- Class KakaktuaTester

```
package pewarisan;

public class KakaktuaTester {
        public static void main(String[] args) {
                Kakaktua k=new Kakaktua();
                k.setNama("Kakaktua"); // diturunkan dari Binatang
                k.bertelur();          // diturunkan dari Burung
                k.terbang();           // diturunkan dari Burung
                k.bersuara();          // diturunkan dari Binatang
                k.bersuara("Selamat Datang ");// override di Kakaktua
        }
}
```

Hasil Run :

Kakaktua sedang bertelur
Kakaktua sedang terbang
Kakaktua sedang bersuara.
Kakaktua bersuara : Selamat Datang

INHERITANCE (PEWARISAN)

- Class Kuda

```
package pewarisan;

public class Kuda extends Mamalia {
        public void berjalan(){
                System.out.println(getNama()+
                        " berjalan : Tuk tik tak tik tuk tik tak");
        }
        public void berlari(){
                System.out.println(getNama()+" sedang berlari ");
        }
}
```

Kuda adalah Mamalia yang dapat berjalan dan berlari. Bedanya ketika kuda berjalan mengeluarkan suara “Tuk tik tak tik tuk tik tak”, dan kelebihan lain dari kuda adalah dapat berlari.

INHERITANCE (PEWARISAN)

- Class KudaTester

```
package pewarisan;

public class KudaTester {
        public static void main(String[] args) {
                Kuda k=new Kuda();
                k.setNama("Si Hitam");// diturunkan dari Binatang
                k.bersuara();  // diturunkan dari Binatang
                k.melahirkan();        // diturunkan dari Mamalia
                k.berjalan();  // override di Kuda
                k.berlari();   // tambahan di Kuda
        }
}
```

Hasil Run :

```
Si Hitam sedang bersuara.
Si Hitam sedang melahirkan.
Si Hitam berjalan : Tuk tik tak tik tuk tik tak
Si Hitam sedang berlari
```

INHERITANCE (PEWARISAN)

- Class Sapi

```
package pewarisan;

public class Sapi extends Mamalia {
        public void bersuara(){
            System.out.println(getNama()+" bersuara : Mooooooooooo");
        }
}
```

Sapi adalah Mamalia (biasa) yang kalau bersuara mengeluarkan suara “Mooooooooooo”. Karena itu maka method bersuara di class Sapi akan mengoverride fungsi bersuara dari class Binatang.

INHERITANCE (PEWARISAN)

- Class SapiTester

```
package pewarisan;

public class SapiTester {
        public static void main(String[] args) {
                Sapi s=new Sapi();
                s.setNama("Si Mooo");
                s.berjalan();
                s.melahirkan();
                s.bersuara();
        }
}
```

Hasil Run :

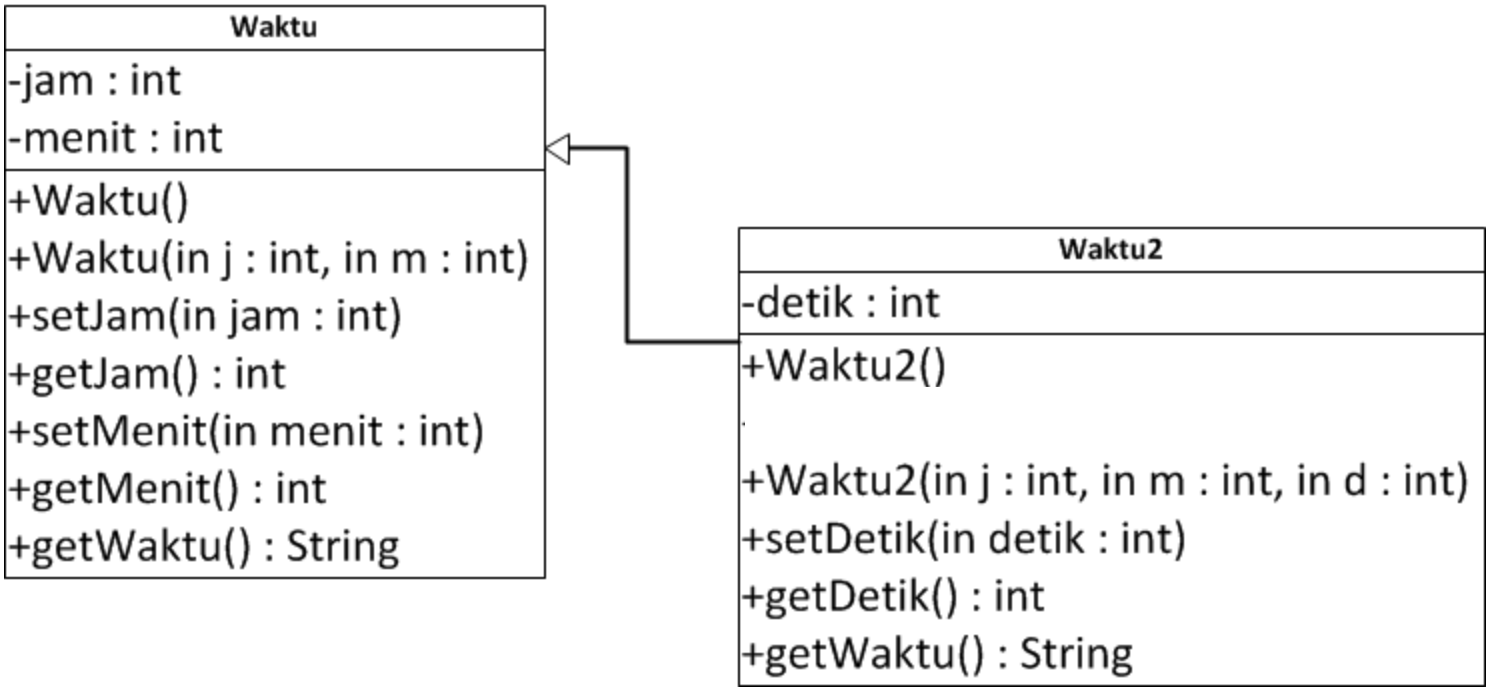
```
Si Mooo sedang berjalan
Si Mooo sedang melahirkan.
Si Mooo bersuara : Mooooooooo
```

OVERRIDE CONSTRUCTOR

- Constructor tidak diturunkan ke subclass, tetapi constructor dapat dipanggil dari subclass dengan menggunakan keyword **super**.
- Keyword super tidak hanya digunakan untuk memanggil constuctor tetapi dapat juga digunakan untuk memanggil method yang ada di superclass.
- Cara pemanggilannya
  - Untuk constructor : super(), atau super(daftar_parameter)
  - Untuk method : super.namamethhod(daftar_parameter)

OVERRIDE CONSTRUCTOR

- Contoh :
  - Class Waktu2 merupakan turunan dari class Waktu yang mempunyai atribut baru bernama detik.



OVERRIDE CONSTRUCTOR

- Class Waktu

```
package pewarisan;

public class Waktu {
        private int jam;
        private int menit;

        public Waktu(){
                jam=0;
                menit=0;
        }
        public Waktu(int j, int m){
                jam=j;
                menit=m;
        }
```

Class Waktu mempunyai 2 buah constructor yaitu :

1. Constructor tanpa parameter
2. Constructor dengan parameter j (jam) dan m (menit).

OVERRIDE CONSTRUCTOR

- Class Waktu

```
    public void setJam(int jam){
            this.jam=jam;
    }
    public int getJam(){
            return jam;
    }
    public void setMenit(int menit){
            this.menit=menit;
    }
    public String getWaktu(){
            return jam+":"+menit;
    }
}
```

Perhatikan pada method setJam. Method ini memiliki parameter input bernama jam. Class waktu juga mempunyai atribut bernama jam. Untuk membedakan jam dari parameter dan jam milik atribut class, gunakan keyword this.
Keyword this menyatakan "milik class ini".

Pernyataan "**`this.jam=jam`**" berarti atribut jam milik class ini diisi dengan jam dari parameter.

Begitu pula untuk method setMenit.

OVERRIDE CONSTRUCTOR

- Class WaktuTester

```
package pewarisan;

public class WaktuTester {
    public static void main(String[] args) {
        Waktu w1, w2;
        w1=new Waktu(); // memanggil constructor tanpa parameter
        System.out.println("W1 : "+w1.getWaktu());
        w2=new Waktu(5,7);// memanggil constructor dengan parameter
        System.out.println("W2 : "+w2.getWaktu());
    }
}
```

Hasil Run :

```
W1 : 0:0
W2 : 5:7
```

OVERRIDE CONSTRUCTOR

- Class Waktu2

```
package pewarisan;

public class Waktu2 extends Waktu {
  private int detik;
  public Waktu2(){
     super();// memanggil constructor class super tanpa parameter
     detik=0;
  }
  public Waktu2(int j, int m, int d){
     super(j,m); // memanggil construktor class super yang menggunakan
parameter
     detik=d;
  }
```

OVERRIDE CONSTRUCTOR

- Class Waktu2Tester

```
  public void setDetik(int detik){
     this.detik=detik;
  }
  public int getDetik(){
     return this.detik; // sama saja dengan return detik;
  }
  public String getWaktu(){
     return super.getWaktu()+":"+detik;
     // atau return getJam()+":"+getMenit()+":"+detik;
  }
}
```

OVERRIDE CONSTRUCTOR

- Class Waktu2

```
package pewarisan;

public class Waktu2Tester {

        public static void main(String[] args) {
                Waktu2 w1,w2;
                w1=new Waktu2();
                System.out.println("W1 : "+w1.getWaktu());
                w2=new Waktu2(10,12,20);
                System.out.println("W2 : "+w2.getWaktu());
        }
}
```

Hasil Run :

```
W1 : 0:0:0
W2 : 10:12:20
```

POLYMORPH

- Polymorph (mempunyai banyak bentuk).
- Jenis-jenis polymorph bisa terjadi adalah :
  - Polymorph Ad Hoc
  - Polymorph Subtyping

POLYMORPH

Polymorph Ad Hoc terjadi ketika memiliki sebuah method yang akan mempunyai operasi yang berbeda ketika dipanggil dengan tipe parameter yang berbeda.

```
package pewarisan;

public class Polymorph {
    public int tambah(int x, int y){
                return x+y;
    }
    public String tambah(String x, String y){
        return x+" "+y;
    }
}
```

Class Polymorph mempunyai 2 buah method yang namanya sama yaitu “tambah”, tetapi dibedakan dengan parameternya.

POLYMORPH

- Class PolymorphTester

```
package pewarisan;

public class PolymorphTester {

  public static void main(String[] args) {
        Polymorph p=new Polymorph();
        System.out.println("2 + 3 = "+p.tambah(2, 3));
        System.out.println("\"2\" + \"3\" = "+p.tambah("2", "3"));
  }
}
```

Hasil Run :

```
2 + 3 = 5
"2" + "3" = 2 3
```

POLYMORPH

Polymorph Subtyping terjadi sebuah referensi dari sebuah class diakses dengan menggunakan referensi subclassnya.

```
package pewarisan;

public class PolySubtype {
  public static void main(String[] args) {
        Binatang elang,kakaktua,kuda,sapi;
        elang=new Elang();elang.setNama("Elang");
        kakaktua=new Kakaktua();kakaktua.setNama("Kakaktua");
        kuda=new Kuda();kuda.setNama("Si Pony");
        sapi=new Sapi();sapi.setNama("Si Mooo");
        elang.bersuara();
        kakaktua.bersuara();
        kuda.bersuara();
        sapi.bersuara();
  }
}
```

Seluruh objek menggunakan superclass. Tetapi tetap akan mengeksekusi method masing-masing class.

Hasil Run :

```
Elang sedang bersuara.
Kakaktua sedang bersuara.
Si Pony sedang bersuara.
Si Mooo bersuara : Mooooooooo
```

POLYMORPH

Contoh Lain :

```
package pewarisan;
public class PolySubtype2 {
        static void bicara(Binatang b){
                b.bersuara();
        }
        public static void main(String[] args) {
                Elang e=new Elang();e.setNama("Hunter");
                Kakaktua k=new Kakaktua();k.setNama("Oces");
                Kuda kd=new Kuda();kd.setNama("Blacky");
                Sapi s=new Sapi();s.setNama("Moooo");
                bicara(e);
                bicara(k);
                bicara(kd);
                bicara(s);

        }
```

Method bicara membutuhkan parameter dengan class Binatang.

Pemanggilan method Bicara menggunakan parameter subclass dari Binatang.

Hasil Run :

```
Hunter sedang bersuara.
Oces sedang bersuara.
Blacky sedang bersuara.
Moooo bersuara : Mooooooooo
```