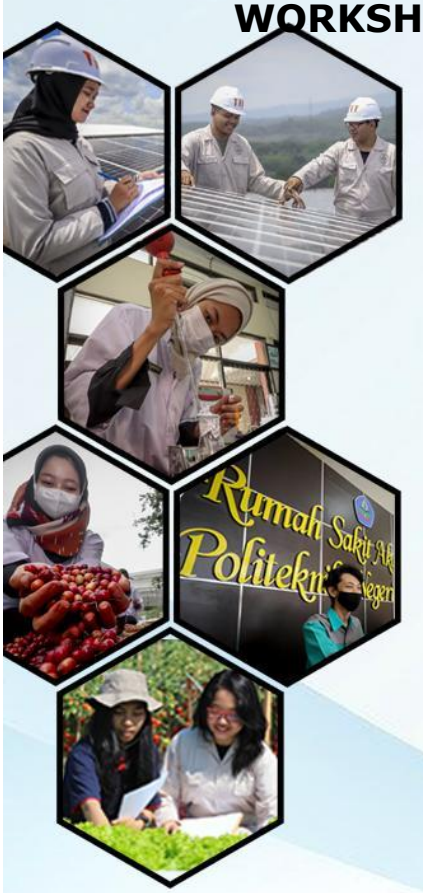
BUKU KERJA PRAKTIK MAHASISWA

(BKPM)

WORKSHOP SISTEM INFORMASI BERBASIS DESKTOP

TIF120707

SEMESTER 2



OLEH :

Bety Etikasari, S.Pd., M.Pd.

Prawidya Destarianto, S.Kom., MT.

Syamsul Arifin, S.Kom, M.Cs.

Zilvanhisna Emka Fitri, ST. MT.

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI JEMBER

TAHUN 2023

Acara 17, 18, 19, 20 Inheritance, Polymorphism, Abstract Class

Materi Pembelajaran : Inheritance, Polymorphism, Abstract Class
Acara Praktikum/Praktik : Minggu 5 / 17, 18, 19, dan 20
Tempat : Laboratorium Jurusan Teknologi Informasi
Alokasi Waktu : 400 menit

a. Capaian Pembelajaran Mata Kuliah (CPMK)

1. Mahasiswa mampu memahami konsep inheritance dan mempraktikkan penerapannya dalam program
2. Mahasiswa mampu memahami konsep polymorphism dan mempraktikkan penerapannya dalam program
3. Mahasiswa mampu memahami konsep abstract class dan mempraktikkan penerapannya dalam program

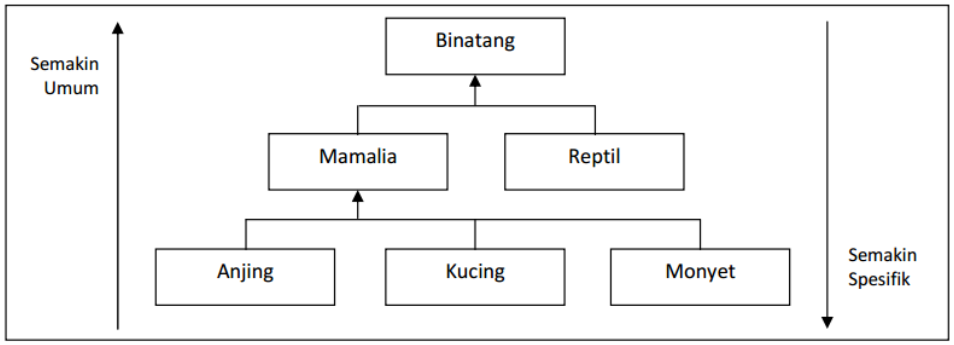
b. Indikator Penilaian

1. Ketepatan dalam memahami konsep inheritance dan mempraktikkan penerapannya dalam program
2. Ketepatan dalam memahami konsep polymorphism dan mempraktikkan penerapannya dalam program
3. Ketepatan dalam memahami konsep abstract class dan mempraktikkan penerapannya dalam program

c. Dasar Teori

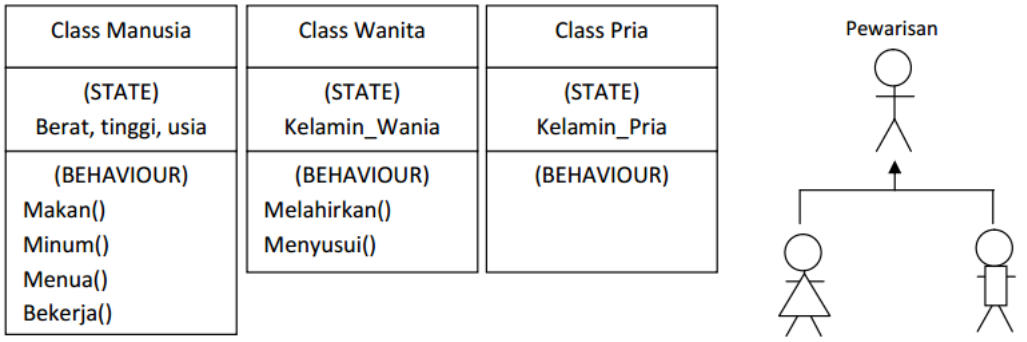
1. Dasar Pewarisan

Sebagai manusia, kita sebenarnya terbiasa untuk melihat objek di sekitar kita tersusun secara hierarki berdasarkan class-nya masing-masing. Dari sini kemudian timbul suatu konsep tentang pewarisan yang merupakan suatu proses di mana suatu class diturunkan dari class lainnya sehingga ia mendapatkan ciri atau sifat dari class tersebut. Perhatikan contoh hierarki class berikut ini :



Gambar 1. Contoh Hierarki class

Dari hierarki di atas, dapat dilihat bahwa semakin ke bawah, class akan bersifat semakin spesifik dan semakin ke atas hierarki, class akan bersifat semakin umum. Class **Mamalia** memiliki seluruh sifat yang dimiliki oleh class **Binatang**, demikian juga halnya class **Anjing**, **Kucing**, dan **Monyet** memiliki seluruh sifat yang diturunkan dari class **Mamalia**. Dengan konsep ini, karakteristik yang dimiliki class Binatang cukup didefinisikan di dalam class **Binatang** saja. Class **Mamalia** tidak perlu didefinisikan ulang apa yang telah dimiliki oleh class **Binatang**, karena sebagai class tutunannya, ia akan mendapakan karakteristik dari class **Binatang** secara otomatis. Demikian juga dengan class **Anjing**, **Kucing**, dan **Monyet**, hanya perlu mendefinisikan karakteristik yang spesifik dimiliki oleh class-nya masing-masing. Dengan memanfaatkan konsep pewarisan ini dalam pemrograman, Anda hanya perlu mendefinisikan karakteristik yang spesifik dimiliki ole suatu class, sementara semua karakteristik yang lebih umum akan didapatkan dari class dari mana ia diturunkan.



Gambar 2. Class dan Objek

Gambar di atas menunjukkan sifat dan hubungan antara class dan objek secara lebih konkrit. Class **Manusia** merupakan class tertinggi, di mana semua state dan behaviour dari class tersebut juga dimiliki oleh class **Wanita** dan **Pria** yang merupakan turunannya. Class **Wanita** dan **Pria** sendiri memiliki sekumpulan state dan behaviour yang spesifik hanya memiliki oleh class mereka masing-masing. Dari gambar di atas juga terlihat perbedaan antara class dan objek, dimana class digunakan untuk menggambarkan suatu model atau spesifikasi dari suatu entitas, sedangkan objek merupakan kenyataan, realisasi, atau perwujudan dari class itu sendiri. Pada dasarnya, kita melakukan pewarisan untuk pembuat class baru (class turunan/subclass) yang masih memiliki sifat atau spesifikasi dari class mana ia diturunkan (superclass). Untuk melakukan ini, Java menyediakan keyword extends yang dapat dipakai pada waktu kita mendeklarasikan class.

2. Eksekusi Konstruktor

Oleh karena subclass memiliki semua yang dimiliki oleh superclass, maka subclass juga memiliki konstruktor yang dimiliki oleh superclass-nya. Anda juga telah mempelajari bahwa class di dalam Java dibentuk secara hierarki dari bentuk yang lebih umum ke bentuk yang lebih khusus. Oleh karena itu, eksekusi konstruktor juga mengikuti pola ini.

3. Method Overriding

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 encoba memanggil method tersebut dari instance subclass yang Anda buat, maka method milik subclass-lah yang akan dipanggil, bukan lagi method milik super-class.

4. Class Abstrak

Terkadang ada saatnya Anda ingin membuat suatu class di mana tidak semua method yang Anda deklarasikan di dalamnya memiliki implementasi. Class ini merupakan superclass yang dibuat sebagai dasar untuk kemudian digunakan oleh subclass, di mana implementasi dari method yang tidak dibuat di dalam superclass-nya tersebut harus diimplementasikan oleh masing-masing. Dengan demikian super-class tersebut dapat memastikan bahwa superclass-nya pasti akan memiliki method yang telah ditentukan sebelumnya. Class yang demikian ini disebut dengan **class abstrak**.

Beberapa aturan tentang class abstrak yang perlu Anda perhatikan:

1) Method yang tidak memiliki implementasi pada suatu class harus dideklarasikan sebagai abstrak dengan menggunakan keyword **abstract**.
2) Anda tidak dapat membuat instance dari class abstrak. Umumnya agar dapat digunakan, class abstrak tersebut harus diturunkan terlebih dahulu, di mana class hasil turunannya tersebut harus meng-override semua method abstrak dari superclass-nya dan membuat implementasinya (sehingga method tersebut bukan lagi merupakan method abstrak) baru kemudian subclass ini dapat dibuat instance-nya.

3) Jika ada satu saja method (baik yang didapat dari superclass-nya sebagai hasil dari turunan maupun yang dideklarasikan di dalam class itu sendiri) yang tidak di -implementasikan, maka class tersebut juga harus dideklarasikan sebagai abstrak dengan menggunakan keyword **abstract**.

5. Polymorphism

Polymorphism berasal dari bahasa Yunani yang berarti “banyak bentuk”. Dalam OOP, konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah suatu objek agar melakukan suatu aksi atau tindakan yang mungkin secara prinsip sama tetapi secara proses berbeda. Dalam konsep yang lebih umum seringkali polymorphism disebut dalam istilah : satu interface, banyak aksi. Contoh yang konkrit dalam dunia nyata.

Mobil yang ada di pasaran terdiri atas berbagai tipe dan merek, namun semuanya memiliki interface kemudi yang sama, seperti : stir kemudi, tongkat transmisi, pedal gas, dan rem. Jika Anda dapat mengemudikan satu jenis mobil saja dari suatu merk tertentu, Anda akan

dapat mengemudikan hampir semua jenis mobil yang ada, karena semua mobil tersebut menggunakan interface yang sama. Harus diperhatikan di sini bahwa interface yang sama tidak berarti cara kerjanya juga sama. Anda mengetahui jika Anda menekan pedal gas, maka mobil akan melaju lebih cepat. Tetapi bagaimana proses ini terjadi, dapat berbeda-beda untuk setiap jenis mobil.

Dengan menggunakan interface yang sama Anda akan mendapatkan hasil yang sama, tetapi bagaimana proses yang terjadi dari aksi (menekan pedal gas) menjadi hasil (menambah kecepatan) mungkin saja berbeda, dalam kasus mobil di atas, hal ini bergantung pada teknologi dan bahan bakar yang digunakan.

d. Alat dan Bahan

1. Kertas HVS A4
2. Spidol
3. Bolpoin
4. Stickynote kecil
5. Netbeans
6. GitHub
7. PC/Laptop
8. Koneksi Internet
9. Web Browser

e. Prosedur Kerja

a. Dasar Pewarisan

Perhatikan dan coba contoh sederhana berikut ini:

```
2   package Pewarisan;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class A {
8       int x;
9       int y;
10
11      void TampilkanNilaiXY(){
12          System.out.println("Nilai X : " + x +", y : " + y);
13      }
14  }
```

```
2   package Pewarisan;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class B extends A{ // Mendeklarasikan Class B yang diturunkan
8       int z;                // dari Class A
9
10      void TampilkanJumlah(){
11          // subclass dapat mengakses member dari superclass
12          System.out.println("Jumlah :" + (x+y+z));
13      }
14  }
```

```
2   package Pewarisan;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class Demo {
8       public static void main(String[] args) {
9           A superOb = new A();
10          B subOb = new B();
11
12          System.out.println("SuperClass");
13          superOb.x = 10;
14          superOb.y = 20;
15          superOb.TampilkanNilaiXY();
16
17          System.out.println("SubClass");
18          subOb.x = 5;
19          subOb.y = 4;
20          subOb.TampilkanNilaiXY();
21
22          //member tambahan yang hanya ada pada subclass
23          subOb.z = 50;
24          subOb.TampilkanJumlah();
25      }
26  }
```

Output:

```
Output - OOP-TI_UMJ (run)
run:
SuperClass
Nilai X : 10, y : 20
SubClass
Nilai X : 5, y : 4
Jumlah :59
BUILD SUCCESSFUL (total time: 1 second)
```

Pada contoh di atas, **class B** diturunkan dari **class A**, dengan demikian **class B** juga memiliki apa yang dimiliki oleh **class A**. Oleh karena pada contoh di atas kita

tidak menggunakan access specifier pada **class A**, maka **class B** dapat mengakses semua member dari **class A**.

Manfaat Pewarisan :

Class B memiliki semua yang dimiliki oleh **class A**. Oleh karena itu pada **class B** cukup ditambahkan bagian yang memang merupakan spesifik milik dari **class B** itu sendiri. Dengan demikian prinsip "reuse of code" telah kita terapkan walaupun dalam bentuk yang paling sederhana.

Contoh:

```
/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Orang {
    private String nama;
    private double tinggi;
    private double berat;

    public Orang (String nama, double tinggi, double berat){
        this.nama = nama;
        this.tinggi = tinggi;
        this.berat = berat;
    }

    public String toString(){
        return ("Nama : "+nama+"\nTinggi :"+ tinggi + "\nBerat : "+berat);
    }
}
```

```
/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Pelajar extends Orang{
    private String nim;
    private String asalSekolah;
    private double nilai; // range : 0-30

    public Pelajar (String nama, double tinggi, double berat, String nim,
            String sekolah, double nilai){
        super(nama, tinggi, berat);
        this.nim = nim;
        asalSekolah = sekolah;
        this.nilai = nilai;
    }
    public String toString(){
        return (super.toString()+"\nNIM :"+nim+"\nSekolah :"+asalSekolah+
                "\nBilai :"+nilai);
    }
}
```

```
/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Test {
    public static void main(String[] args) {
        Pelajar mahasiswa = new Pelajar("Lukman", 172, 75,
                "1010651098", "UM JEMBER", 98.8);
        System.out.println(mahasiswa.toString());
    }
}
```

Cek outputnya

b. Konstruktor

Perhatikan dan coba contoh berikut ini untuk lebih jelasnya :

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class A {
    A(){
        System.out.println("Konstruktor class A dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class B extends A{
    B(){
        System.out.println("Konstruktor class B dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class C extends B{
    C(){
        System.out.println("Konstruktor class C dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Demo {
    public static void main(String[] args) {
        C subOb = new C();
    }
}
```

Dari contoh di atas terlihat bahwa konstruktor dieksekusi dari konstruktor milik superclass milik superclass tertinggi, baru kemudian yang terakhir adalah konstruktor dari class itu sendiri.

Perlu Anda perhatikan bahwa sebenarnya dalam setiap konstruktor dari suatu class, kode pertama yang ada di sana adalah kode yang memanggil konstruktor dari superclass-nya. Itulah sebabnya mengapa konstruktor selalu dieksekusi secara berurutan mulai dari konstruktor milik superclass tertinggi. Jika Anda tidak menuliskan secara eksplisit kode untuk memanggil konstruktor dari superclass, maka

Java akan membuatkan kode tersebut bagi Anda secara otomatis, di mana kode tersebut akan memanggil konstruktor default dari superclass-nya.

Anda dapat saja menuliskan secara eksplisit kode untuk memanggil konstruktor superclass, di mana keyword yang digunakan sama sepert jika kita ingin mengakses member dari superclass, yaitu super. Bedanya di sini, keyword ini diperlakukan seolah -olah ia adalah method. Secara umum bentuk pemakaiannya adalah sebagai berikut:

super(daftar-parameter);

Daftar-parameter di sini harus sesuai dengan daftar-parameter milik konstruktor superclass-nya. Dengan keyword ini, Anda dapat menentukan sendiri konstruktor mana yang dimiliki oleh superclass yang akan diseksekusi (jika superclass memiliki lebih dari satu konstruktor) dengan mencocokkan jmalh dan tipe data dari parameternya. Maka jika Anda memanggil konstruktor default, Anda akan menulis kode berikut : super().

Seperti yang telah dijelaskan sebelumnya, jika Anda tidak menuliskan secara eksplisit konstruktor superclass mana yang ingin Anda eksekusi di dalam konstruktor subclass-nya, maka konstruktor default dari superclass-lah yang akan di eksekusi. Dengan demikian Class A, B, dan C pada contoh program di atas sebenarnya secara otomatis akan dituliskan oleh Java seperti berikut ini :

```
2   package Konstruktor;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class A {
8       A(){ // super(); Tambahan kode secara otomatis oleh Java untuk memanggil
9            //          Konstruktor dari SuperClassnya
10
11          super(); // Konstruktor default dari class
12                   // Object dipanggil
13          System.out.println("Konstruktor class A dieksekusi ...");
14      }
15  }
```

```
2   package Konstruktor;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class B extends A{
8       B(){ // super(); Tambahan kode secara otomatis oleh Java untuk memanggil
9            //          Konstruktor dari SuperClassnya
10
11          super(); // Konstruktor default dari class A
12                   // Object dipanggil
13          System.out.println("Konstruktor class B dieksekusi ...");
14      }
15  }
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class C extends B{
    C(){ // super(); Tambahan kode secara otomatis oleh Java untuk memanggil
         //          Konstruktor dari SuperClassnya
        super(); // Konstruktor default dari class B
                 // Object dipanggil
        System.out.println("Konstruktor class C dieksekusi ...");
    }
}
```

Kesalahan umum yang sering terjadi di sini adalah, disadari atau tidak, class yang Anda buat mencoba memanggil konstruktor superclass yang sebenarnya tidak ada. Perhatikan contoh berikut ini :

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class A {
    A(String param1){ //Konstruktor untuk class A sudah didefinisikan sehingga
                      //Java tidak membuat konstruktor default untuk class ini.

        super(); // Konstruktor default dari class
                 // Object dipanggil
        System.out.println("Konstruktor class A dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class B extends A{
    B(){
        // Pemanggilan konstruktor superclass dengan keyword super tidak
        // dilakukan sehingga Java akan menambahkan secara otomatis kode untuk
        // memanggil konstruktor default milik superclassnya disini.
        System.out.println("Konstruktor class B dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Demo {
    public static void main(String[] args) {
        B subOb = new B();
    }
}
```

Output:

```
Output - OOP-TI_UMJ (run)
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - Erroneous sy
        at Konstruktor.B.<init>(B.java:8)
        at Konstruktor.Demo.main(Demo.java:9)
C:\Users\LScom\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Jika Anda mencoba mengompilasi program di atas, Anda akan mendapatkan pesna kesalahan. Apakah Anda tahu di mana letak kesalahan tersebut ? Perhatikan **class A**. Konstruktor untuk **class A** sudah kita definisikan, karena itu Java tidak akan membuat konstruktor default (konstruktor tanpa parameter) untuk **class A** ini. Sekarang perhatikan konstruktor dari **class B** yang merupakan turunan dari **class A**. Di sini tidak ada kode yang secara eksplisit menentukan konstruktor milik superclass mana yang ingin dieksekusi. Oleh karena itu Java akan secara otomatis menambahkan kode untuk mengeksekusi konstruktor default milik superclass (dalam hal ini adalah **class A**). Namun ingatlah, karena konstruktor untuk **class A** sudah didefinisikan, maka **class A** tidak memiliki konstruktor default. Dengan demikian Anda akan mendapatkan kesalahan pada waktu hendak mengompilasi program tersebut. Agar program di atas dapat dikompilasi, ada dua cara yang dapat Anda lakukan:

Pertama, tambahkan bentuk konstruktor default pada class A, seperti berikut:

```
1
2   package Konstruktor;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
@   public class A {
8       A(){} //Konstruktor default, diperlukan hanya agar program dapat
9             //dikompilasi.
10      A(String param1){
11          System.out.println("Konstruktor class A dengan Parameter " + param1 + "dieksekusi ...");
12      }
13  }
```

Kedua, jika Anda tidka ingin menambahkan konstruktor default seperti di atas, Anda dapat mengubah konstruktor dari **class B** supaya secara eksplisit memanggil konstruktor dari **class A** yang ada, Contohnya :

```
1
2   package Konstruktor;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
@   public class B extends A{
8       B(){
9           super("test");
10          System.out.println("Konstruktor class B dieksekusi ...");
11      }
12  }
```

Jika Anda memilih bentuk kedua maka program di atas menjadi :

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class A {
    A(String param1){
        System.out.println("Konstruktor class A dengan Parameter " + param1 + "dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class B extends A{
    B(){
        super("test"); // tambahan
        System.out.println("Konstruktor class B dieksekusi ...");
    }
}
```

```java
package Konstruktor;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Demo {
    public static void main(String[] args) {
        B subOb = new B();
    }
}
```

Output:

```
Output - OOP-TI_UMJ (run)
run:
Konstruktor class A dengan Parameter testdieksekusi ...
Konstruktor class B dieksekusi ...
BUILD SUCCESSFUL (total time: 0 seconds)
```

c. Method Overriding

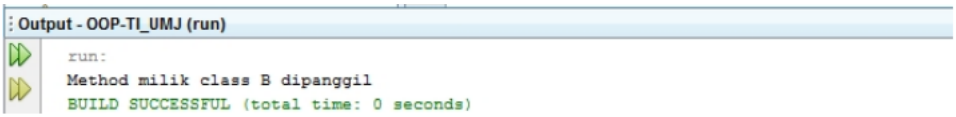
Perhatikan dan coba contoh berikut ini:

```java
package Overriding;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class A {
    public void tampilkanKeLayar(){
        System.out.println("Method milik class A dipanggil");
    }
}
```

```java
package Overriding;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class B extends A{
    public void tampilkanKeLayar(){
        System.out.println("Method milik class B dipanggil");
    }
}
```

```
1
2   package Overriding;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class Demo {
8       public static void main(String[] args) {
9           B subOb = new B();
10          subOb.tampilkanKeLayar();
11      }
12  }
```

Output:



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 variabel instance class B (variabel subOb), yang terpanggil adalah method yang sama yang ada di class B. Sebenarnya Anda tetap dapat memanggil method milik superclass yang telah di-override tersebut dari dalam subclass-nya dengan menggunakan keyword super, dimana penggunaannya mirip jika Anda hendak mengakses property milik dari superclass yang diakses dari subclass (lihat penjelaskan penggunaan keyword super yang telah dijelaskan sebelumnya). Contoh program :

```java

package Overriding;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class A {
    public void tampilkanKeLayar(){
        System.out.println("Method milik class A dipanggil");
    }
}
```

```java

package Overriding;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class B extends A{
    public void tampilkanKeLayar(){
        super.tampilkanKeLayar(); // Memanggil method milik dari superclassnya.
        System.out.println("Method milik class B dipanggil");
    }
}
```

```java

package Overriding;

/**
 * @author Lukman Hakim, S.Kom., M.Kom
 */
public class Demo {
    public static void main(String[] args) {
        B subOb = new B();
        subOb.tampilkanKeLayar();
    }
}
```

Output:

```
Output - OOP-TI_UMJ (run)
run:
Method milik class A dipanggil
Method milik class B dipanggil
BUILD SUCCESSFUL (total time: 0 seconds)
```

d. Abstract Class

Perhatikan dan coba contoh berikut ini:

```
2   package Abstract;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public abstract class A { // Deklarasi Class
8       abstract public void method2(); // Deklarasi method abstrak.
9       // method konkrit yang memiliki implementasi
10      public void method1(){
11          System.out.println("Method konkrit dari class A");
12      }
13  }
```

```
2   package Abstract;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class B extends A{
8       public void method2(){ // Method absrtak di-override dan dibuat
9
10          System.out.println("Method abstract yang sudah menjadi konkrit dalam class B");
11      }
12  }
```

```
2   package Abstract;
3
4   /**
5    * @author Lukman Hakim, S.Kom., M.Kom
6    */
7   public class Demo {
8       public static void main(String[] args) {
9           B ob = new B();
10          ob.method1();
11          ob.method2();
12      }
13  }
```

Output:

```
Output - OOP-TI_UMJ (run)
run:
Method konkrit dari class A
Method abstract yang sudah menjadi konkrit dalam class B
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tugas

1. Ketikkan code program berikut dan jelaskan outputnya

```
import java.awt.*;
import java.applet.*;

public class DrawShapes extends Applet  {
        Font font;
   Color redColor;
   Color blueColor;
   Color backgroundColor;
   public void init() {
      //The Font is Arial size, 18 and is italicized
     font = new Font("Arial",Font.ITALIC,18);

     //Some colors are predefined in the Color class
      redColor = Color.red;
      backgroundColor = Color.orange;

      //Colors can be created using Red, Green, Blue values
      blueColor = new Color(0,0,122);

      //Set the background Color of the applet
      setBackground(backgroundColor);
  }
     public void stop() {
     }
     /**
      * This method paints the shapes to the screen
      */
     public void paint(Graphics graph) {
        //Set font
         graph.setFont(font);
         //Create a title
        graph.drawString("Draw Shapes",90,20);

        //Set the color for the first shape
        graph.setColor(blueColor);

        // Draw a rectangle using drawRect(int x, int y, int width, int height)
        graph.drawRect(120,120,120,120);

        // This will fill a rectangle
        graph.fillRect(115,115,90,90);

        //Set the color for the next shape
        graph.setColor(redColor);

        //Draw a circle using  drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
        graph.fillArc(110,110,50,50,0,360);

        //Set color for next shape
        graph.setColor(Color.CYAN);

        //Draw another rectangle
        graph.drawRect(50,50,50,50);

        // This will fill a rectangle
        graph.fillRect(50,50,60,60);
     }
}
```

2. Ketikkan code program berikut dan jelaskan outputnya

```
class A
{
    void callthis() {
        System.out.println("Inside Class A's Method!");
    }
}

class B extends A
{
    void callthis() {
        System.out.println("Inside Class B's Method!");
    }
}

class C extends A
{
    void callthis() {
        System.out.println("Inside Class C's Method!");
    }
}

class DynamicDispatch {
    public static void main(String args[]) {
        A a = new A();
        B b = new B();
        C c = new C();
        A ref;

        ref = b;
        ref.callthis();

        ref = c;
        ref.callthis();

        ref = a;
        ref.callthis();
    }
}
```

3. Ketikkan code program berikut dan jelaskan outputnya

```
public class Animal {

    public void makeNoise() {

        System.out.println("talk");

    }
```

```
}
public class Dog extends Animal {
	public void makeNoise() {
		System.out.println("Bark");
	}
}
```

Apabila ditambah code ini maka:

```
Animal animal = new Animal();
animal.makeNoise();
Dog dog = new Dog();
dog.makeNoise();
Animal animaldog = new Dog();
animaldog.makeNoise();

if (animal instanceof Animal)
	System.out.println("animal is Animal");
if (dog instanceof Animal)
	System.out.println("dog is Animal");
if (animaldog instanceof Animal)
	System.out.println("animaldog is Animal");
if (animal instanceof Dog)
	System.out.println("animal is Dog");
```

f. Hasil dan Pembahasan

Laporan hasil praktik dan tugas beserta penjelasannya

g. Rubrik Penilaian

No	INDIKATOR KINERJA	Bobot (%)	Penilaian	Nilai Akhir
1	Memberikan penjelasan dan analisa secara terstruktur yang disertai contoh implementasi Inheritance, Polymorphism, Abstract Class	30		
2	Memberikan bukti referensi dari jawaban	15		
3	Mempertanggung jawabkan hasil pekerjaan dengan presentasi	25		
4	Kerapian dalam menulis (bahasa dan struktur penulisan)	10		
5	Ketepatan waktu mengumpulkan	20		
	Total	**100**		