



# Modul Praktikum Pemrograman Berorientasi Objek

Jurusan Ilmu Komputer  
Fakultas Matematika dan Ilmu Pengetahuan Alam  
**Universitas Lampung**

**2022**  
Semester Ganjil

## Modul 10. Inheritance dan Polymorphism

Inheritance atau pewarisan dapat didefinisikan sebagai proses di mana satu kelas memperoleh properti (attribute dan method) dari kelas yang lain.

### Capaian Pembelajaran

1. Mahasiswa mampu mengimplementasikan konsep inheritance pada class
2. Mahasiswa mampu membuat constructor pada class turunan
3. Mahasiswa mampu mengimplementasikan konsep polymorphism

### Materi

#### Inheritance/Pewarisan

Inheritance atau pewarisan dapat didefinisikan sebagai proses di mana satu kelas memperoleh properti (attribute dan method) dari kelas yang lain. Kelas yang mewarisi sifat-sifat kelas lain disebut subclass (kelas turunan, kelas anak) dan kelas yang mewarisi sifat-sifatnya disebut superclass (kelas dasar, kelas induk).

Ide pewarisan sebenarnya sederhana, yaitu ketika kita ingin membuat kelas baru dan sudah ada kelas yang menyertakan beberapa kode yang kita inginkan, kita bisa mendapatkan kelas baru dari kelas yang telah ada tersebut. Dalam melakukan ini, kita dapat menggunakan kembali field dan method kelas yang ada tanpa harus menulis ulang.

Subclass mewarisi semua anggota (field dan method) dari superclass-nya. Konstruktor tidak termasuk sebagai anggota, jadi mereka tidak diwarisi oleh subclass, tetapi konstruktor superclass dapat dipanggil dari subclass.

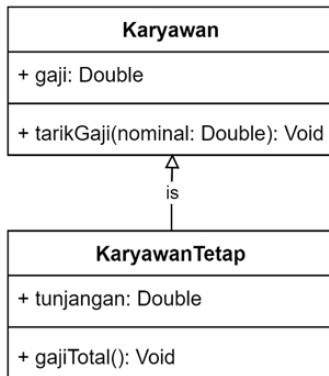
#### Keyword Extends

Extends adalah keyword yang digunakan untuk mewarisi properti dari kelas. Sintaks dari extends adalah:

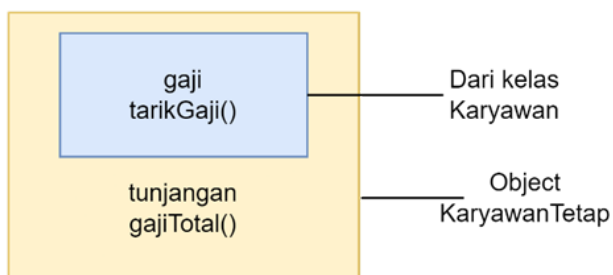
```
class Super {  
    ...  
    ...  
}
```

```
class Sub extends Super {
    ...
    ...
}
```

### Ilustrasi Inheritance



Misalkan terdapat 2 kelas, dimana **Karyawan** bertindak sebagai superclass dan **KaryawanTetap** bertindak sebagai subclass. **KaryawanTetap** mewarisi properti dari **Karyawan**, yaitu gaji dan juga method `tarikGaji`, serta memiliki properti tambahan yang hanya dimiliki oleh **KaryawanTetap**, yaitu tunjangan dan method `gajiTotal`. Tidak semua karyawan memiliki tunjangan dan method `gajiTotal`, hanya **KaryawanTetap** saja.



### Keyword Super

Keyword `super` mirip dengan keyword `this`. Keyword ini digunakan dalam skenario berikut:

1. Untuk membedakan member dari superclass dan member dari subclass, jika mereka memiliki nama yang sama

```
class Super {
    int num = 10;

    public void print() {
```

```

        System.out.println("Ini superclass");
    }
}

class Sub extends Super {
    int num = 6;

    public void print() {
        System.out.println("Ini subclass");
    }

    public void proveIt() {
        Sub sub = new Sub();

        sub.print();
        super.print();

        System.out.println("Sub num: " + sub.num);
        System.out.println("Super num: " + super.num);
    }
}

public class Playground {
    public static void main(String[] args) {
        Sub subClass = new Sub();

        subClass.proveIt();
    }
}

```

Output:

```

Ini subclass
Ini superclass
Sub num: 6
Super num: 10

```

## 2. Memanggil constructor superclass

Telah disebutkan diatas bahwa subclass tidak dapat langsung memanggil constructor superclassnya, oleh karenanya dapat diatasi dengan keyword `super`

```
class Super {
    int num;

    Super(int _num) {
        this.num = _num;
    }

    public void printNum() {
        System.out.println(num);
    }
}

class Sub extends Super {
    String nama;
    Sub(int _num, String _nama) {
        super(_num); //Constructor superclass
        this.nama = _nama;
    }

    public void proveIt() {
        System.out.println("Sub num: " + this.num);
        System.out.println("Sub nama: " + nama);
    }
}


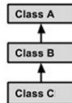
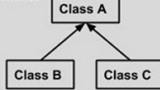
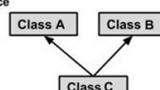
public class Playground {
    public static void main(String[] args) {
        Sub subClass = new Sub(10, "Ana");

        subClass.proveIt();
    }
}
```

Output:

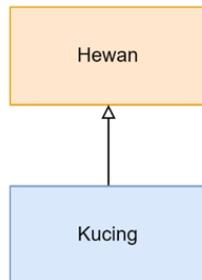
```
Sub num: 10
Sub nama: Ana
```

## Type-Type Inheritance

<b>Single Inheritance</b>  <pre> graph BT     B[Class B] --&gt; A[Class A]         </pre>	<pre> public class A {     ..... } public class B extends A {     ..... }         </pre>
<b>Multi Level Inheritance</b>  <pre> graph BT     C[Class C] --&gt; B[Class B]     B --&gt; A[Class A]         </pre>	<pre> public class A { ..... } public class B extends A { ..... } public class C extends B { ..... }         </pre>
<b>Hierarchical Inheritance</b>  <pre> graph BT     B[Class B] --&gt; A[Class A]     C[Class C] --&gt; A         </pre>	<pre> public class A { ..... } public class B extends A { ..... } public class C extends A { ..... }         </pre>
<b>Multiple Inheritance</b>  <pre> graph BT     A[Class A] --&gt; C[Class C]     B[Class B] --&gt; C         </pre>	<pre> public class A { ..... } public class B { ..... } public class C extends A,B {     ..... } // Java does not support multiple Inheritance         </pre>

### 1. Single Inheritance

Dalam single inheritance, subclass mewarisi fitur dari satu superclass.



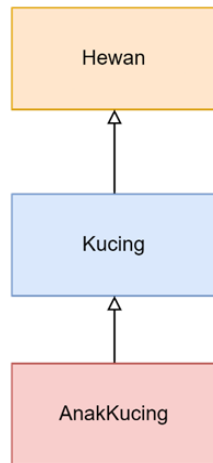
```

class Hewan {
    public void makan() {
        System.out.println("Makan lur");
    }
}

class Kucing extends Hewan {
    public void mengeong() {
        System.out.println("Ngeong");
    }
}
        
```

### 2. Multilevel Inheritance

Dalam multilevel inheritance, sebuah kelas turunan akan mewarisi sebuah kelas dasar dan begitu juga kelas turunan tersebut juga berperan sebagai kelas dasar bagi kelas lainnya.



```

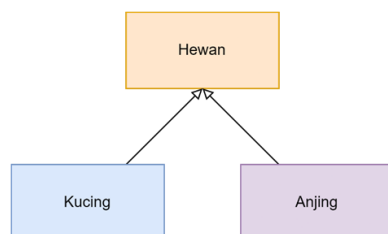
class Hewan {
    public void makan() {
        System.out.println("Makan lur");
    }
}

class Kucing extends Hewan {
    public void mengeong() {
        System.out.println("Ngeong");
    }
}

class AnakKucing extends Kucing {
    public void menyusui() {
        System.out.println("Haus nih");
    }
}
  
```

### 3. Hierarchical Inheritance

Dalam hierarchical inheritance, satu kelas berfungsi sebagai superclass (kelas dasar) untuk lebih dari satu subkelas.



```

class Hewan {
    public void makan() {
  
```

```

        System.out.println("Makan lur");
    }
}

class Kucing extends Hewan {
    public void mengeong() {
        System.out.println("Ngeong");
    }
}

class Anjing extends Hewan {
    public void menggonggong() {
        System.out.println("Gukguk");
    }
}

```

## Polymorphism

Polymorphism berarti banyak bentuk dalam bahasa Yunani. Dalam pemrograman berorientasi objek, istilah ini digunakan untuk memerintah objek untuk melakukan tindakan yang secara prinsip sama, namun secara proses berbeda.

Salah satu contoh nyatanya adalah Hewan yang terdiri atas berbagai jenis. Setiap hewan tentunya memiliki interface untuk gerak yang secara umum sama, yaitu kaki. Walaupun sama-sama memiliki kaki, namun jumlah kaki yang dimiliki dan kecepatan geraknya tentu berbeda antara jenis hewan yang satu dengan yang lain. Konsep interface yang sama, namun dengan proses dan hasil yang berbeda inilah yang dinamakan sebagai Polymorphism.

## Virtual Method Invocation

Virtual Method Invocation adalah suatu hal yang penting dalam konsep Polymorphism. Syarat terjadinya VMI adalah sebelumnya sudah terjadi Polymorphism. Pada saat objek yang dibuat memanggil overridden method pada parent class, compiler akan menjalankan invocation (pemanggilan) pada method overriding pada subclass.



```

class Parent {
    public int x = 5;

    public void info() {
        System.out.println("Ini parent");
    }
}

class Child extends Parent {
    public int x = 10;

    public void info() {
        System.out.println("Ini child");
    }
}

public class App {
    public static void main(String[] args) throws Exception {
        Parent tes = new Child();
        System.out.println("Nilai x = " + tes.x);
        tes.info();
    }
}

```

Pada contoh diatas, terjadi hal-hal berikut :

- Obyek tes mempunyai method sesuai dengan runtime type
- Ketika compile time, tes adalah Parent
- Ketika runtime, tes adalah Child
- Sehingga :
  - tes hanya bisa mengakses variabel milik Parent
  - tes hanya bisa mengakses method milik Child

### Polymorphic Argument

Polymorphic arguments adalah tipe data suatu argumen pada suatu method yang bisa menerima suatu nilai bertipe subclass-nya. Polymorphic arguments diperlukan, bertujuan untuk mengefisienkan pembuatan program. Misalnya suatu kelas memiliki banyak subclass, maka perlu didefinisikan semua method yang menangani behaviour dari masing-masing subclass. Dengan adanya

polymorphic arguments cukup mendefinisikan satu method saja yang bisa digunakan untuk menangani behaviour semua kelas.

```
class Parent {
    public void info() {
        System.out.println("Ini parent");
    }
}

class Child extends Parent {
    public void info() {
        System.out.println("Ini child");
    }
}

class ChildDua extends Parent {
    public void info() {
        System.out.println("Ini child kedua");
    }
}

public class App {
    public static void main(String[] args) throws Exception {
        getInfo(new Parent());
        getInfo(new Child());
        getInfo(new ChildDua());
    }

    public static void getInfo(Parent parent) {
        parent.info();
    }
}
```

### Operator instanceof

Pernyataan instanceof sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments. Pemakaian instanceof seringkali diikuti dengan casting objek dari tipe parameter ke tipe asal. Tanpa melakukan casting objek, maka nilai yang akan kita pakai setelah proses instanceof masih bertipe parent class-nya, sehingga jika tipe tersebut perlu dipakai maka harus di casting terlebih dahulu ke tipe subclass-nya.

## Overriding

Overriding adalah suatu keadaan dimana subclass memodifikasi tingkah laku yang diwarisi dari super class. Tujuannya untuk menspesifikan tingkah laku dari subclass tersebut. Overriding dilakukan dengan cara mendeklarasikan kembali method milik superclass di dalam subclass. Deklarasi method pada subclass harus sama dengan yang terdapat pada di super class, yaitu kesamaan pada nama, return type, dan daftar parameter (jumlah, tipe, dan urutan). Method pada parent class disebut overridden method dan pada subclass disebut overriding method.

```
class Parent {
    public void info() {
        System.out.println("Ini parent");
    }
}

class Child extends Parent {
    public void info() {
        System.out.println("Ini child");
    }
}

class ChildDua extends Parent {
    public void info() {
        System.out.println("Ini child kedua");
    }
}
```

Pada contoh diatas, fungsi info() akan di-override pada class Child. Fungsi info() pada Parent disebut sebagai overridden method, dan fungsi info() pada Child disebut sebagai overriding method.

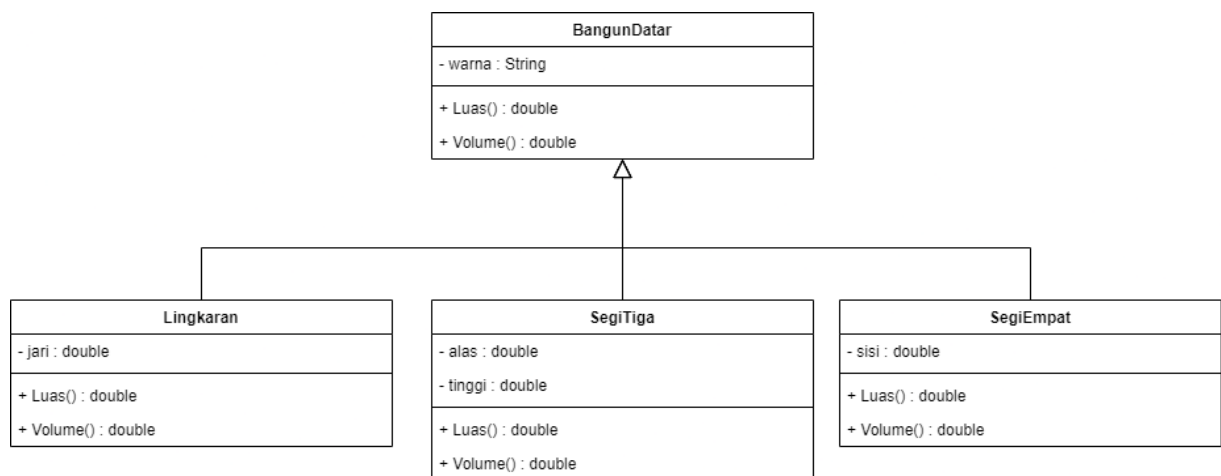
## Langkah Praktikum

### Membuat Project

1. Buka Netbeans
2. Buat project baru dengan nama project **Praktikum10\_nama**.

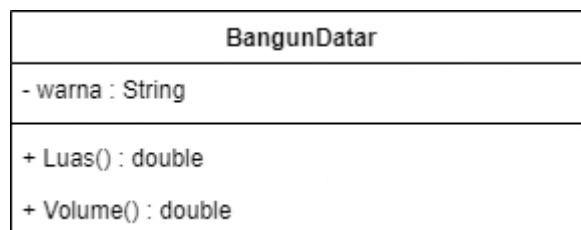
### Studi Kasus

Anda akan diminta untuk membuat beberapa class yang saling terhubung dengan konsep Inheritance dan Polymorphism. Class yang akan dibuat dapat dilihat pada class diagram berikut:



### Membuat Class Utama

1. Buatlah sebuah class **BangunDatar** yang akan menjadi parent class dengan atribut dan method sesuai class diagram



2. Buat Constructor, setter, dan getter untuk class tersebut.
3. Method luas dan keliling akan mengembalikan nilai **double**.

4. Isi method luas dan keliling pada class ini dengan return 0.

```
public class BangunDatar {
    private String warna;

    public BangunDatar(String warna) {
        this.warna = warna;
    }

    public BangunDatar() {
    }

    public String getWarna() {
        return warna;
    }

    public void setWarna(String warna) {
        this.warna = warna;
    }

    public double luas(){
        return 0;
    }

    public double keliling(){
        return 0;
    }
}
```

### Membuat Child Class

1. Buatlah class Lingkaran yang merupakan turunan dari class BangunDatar sesuai dengan class diagram berikut :

Lingkaran
- jari : double
+ Luas() : double
+ Volume() : double

2. Anda dapat menggunakan kata kunci **extends** untuk membuat turunan class.

```
public class Lingkaran extends BangunDatar {  
  
}
```

3. Untuk membuat constructor pada child class diperlukan kata kunci **super**, untuk mengisi atribut yang ada pada parent class.

```
public Lingkaran(String warna, double jari) {  
    super(warna);  
    this.jari = jari;  
}
```

4. Buat juga setter dan getter untuk atribut jari pada class Lingkaran.
5. Anda perlu mendefinisikan kembali isi dari method Luas dan Keliling sesuai dengan rumus luas dan keliling lingkaran.
6. Anda dapat menambahkan kata kunci **Override** di atas masing-masing method yang di override dari parent class nya.

```
@Override  
public double luas() {  
    return Math.PI * this.jari * this.jari;  
}  
  
@Override  
public double keliling() {  
    return 2 * Math.PI * this.jari;  
}
```

7. Sehingga secara keseluruhan class Lingkaran akan terlihat seperti di bawah ini

```
public class Lingkaran extends BangunDatar {
    private double jari;

    public Lingkaran(String warna, double jari) {
        super(warna);
        this.jari = jari;
    }

    public Lingkaran(){}

    @Override
    public double luas() {
        return Math.PI * this.jari * this.jari;
    }

    @Override
    public double keliling() {
        return 2 * Math.PI * this.jari;
    }

    public double getJari() {
        return jari;
    }

    public void setJari(double jari) {
        this.jari = jari;
    }
}
```

8. Lakukan hal yang sama pada class SegiTiga dan SegiEmpat sesuai dengan class diagram berikut:

SegiTiga	SegiEmpat
- alas : double	- sisi : double
- tinggi : double	
+ Luas() : double	+ Luas() : double
+ Volume() : double	+ Volume() : double

### Membuat Objek pada Classs Utama

- Selanjutnya buatlah objek dari setiap class dan cetak beberapa atribut yang ada pada masing-masing objek.

2. Sebagai contoh seperti pada class utama berikut:

```
BangunDatar bd = new BangunDatar("Hitam");
System.out.println("Warna bangun datar: " + bd.getWarna());
System.out.println();

Lingkaran lingkaran = new Lingkaran("Merah", 26.0);
System.out.println("Luas Lingkaran: " + lingkaran.luas());
System.out.println("Keliling Lingkaran: " + lingkaran.keliling());
System.out.println();

SegiTiga segiTiga = new SegiTiga("Kuning", 15, 7);
System.out.println("Luas Segi Tiga: " + segiTiga.luas());
System.out.println("Keliling Segi Tiga: " + segiTiga.keliling());
System.out.println();

SegiEmpat segiEmpat = new SegiEmpat("Hijau", 10);
System.out.println("Luas Segi Empat: " + segiEmpat.luas());
System.out.println("Keliling Segi Empat: " + segiEmpat.keliling());
System.out.println();
```

## Polymorphism

1. Ubahlah deklarasi objek pada class utama sebelumnya.
2. Ubah deklarasi class Lingkaran, SegiTiga, dan SegiEmpat menjadi BangunDatar.

```
BangunDatar bd = new BangunDatar("Hitam");
System.out.println("Warna bangun datar: " + bd.getWarna());
System.out.println();

BangunDatar lingkaran = new Lingkaran("Merah", 26.0);
System.out.println("Luas Lingkaran: " + lingkaran.luas());
System.out.println("Keliling Lingkaran: " + lingkaran.keliling());
System.out.println();

BangunDatar segiTiga = new SegiTiga("Kuning", 15, 7);
System.out.println("Luas Segi Tiga: " + segiTiga.luas());
System.out.println("Keliling Segi Tiga: " + segiTiga.keliling());
System.out.println();

BangunDatar segiEmpat = new SegiEmpat("Hijau", 10);
System.out.println("Luas Segi Empat: " + segiEmpat.luas());
System.out.println("Keliling Segi Empat: " + segiEmpat.keliling());
System.out.println();
```