



PEMROGRAMAN BERORIENTASI OBJEK

KELAS XI SMK

Teori dan Praktik
Enkapsulasi dan Pewarisan

MUHAMMAD FARID ARY NUGROHO
DILA UMNIA SORAYA, S.PD., M.PD.

PEMROGRAMAN BERORIENTASI OBJEK

JURUSAN REKAYASA PERANGKAT LUNAK KELAS XI



Penyusun

Muhammad Farid Ary Nugroho

Pembimbing

Dila Umnia Soraya, S.PD., M.PD.

UNIVERSITAS NEGERI MALANG

DAFTAR ISI

DAFTAR ISI	II
PETUNJUK PENGGUNAAN	III
Enkapsulasi	1
Definisi Enkapsulasi	3
Penyembunyian data dan pengaksesan data.	4
Modifier	7
Get dan Set (Getter dan Setter)	8
Video Materi Enkapsulasi	11
Kesimpulan	12
Tugas	13
Inheritance dan Polymorphism	14
Definisi Inheritance	16
Penerapan keyword extends, super, dan this	28
Polymorphism	33
Video materi	40
Kesimpulan	41
Tugas	42
Daftar Pustaka	43

PETUNJUK PENGGUNAAN

Berikut merupakan panduan lengkap untuk pemanfaatan aplikasi yang diharapkan dapat membantu dalam memahami fitur-fitur utama dan cara pengoperasian untuk mendapat pengalaman terbaik.

Fungsi toolbar :



Halaman Sebelumnya



Halaman Selanjutnya



Halaman Pertama



Halaman Terakhir



Memperbesar atau memperkecil tampilan



Menampilkan thumbnail halaman



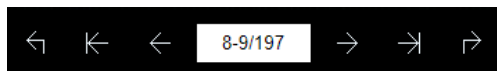
Menampilkan Heading Navigation (Table Of Content)



Mengaktifkan atau menonaktifkan autoflip



Mengatur volume suara (on / off sound effect dan backsound)



Toolbar Halaman khusus Windows dan Mac



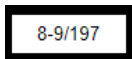
Backward



First



Previous



Jump To



Next



Last



Forward



Cara Penggunaan khusus mobile (IOS dan Android)



Navigasi halaman khusus mobile (IOS dan Android)



Mengunduh Flipbook dengan format .pdf



Menampilkan E-Modul secara Full-Screen

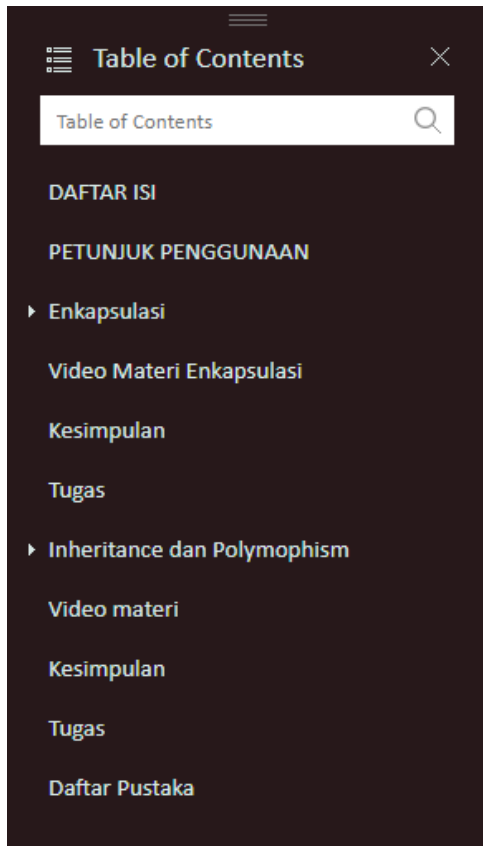


Untuk membagikan melalui E-mail



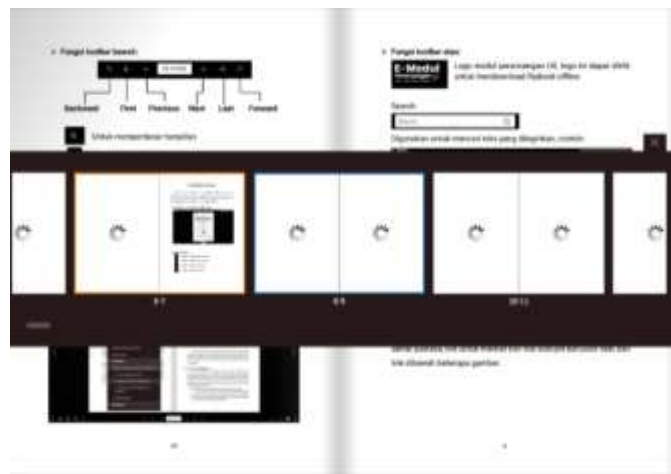
Untuk menandai teks pada Flipbook

Tampilan Heading Navigation



Tampilan thumbnail halaman

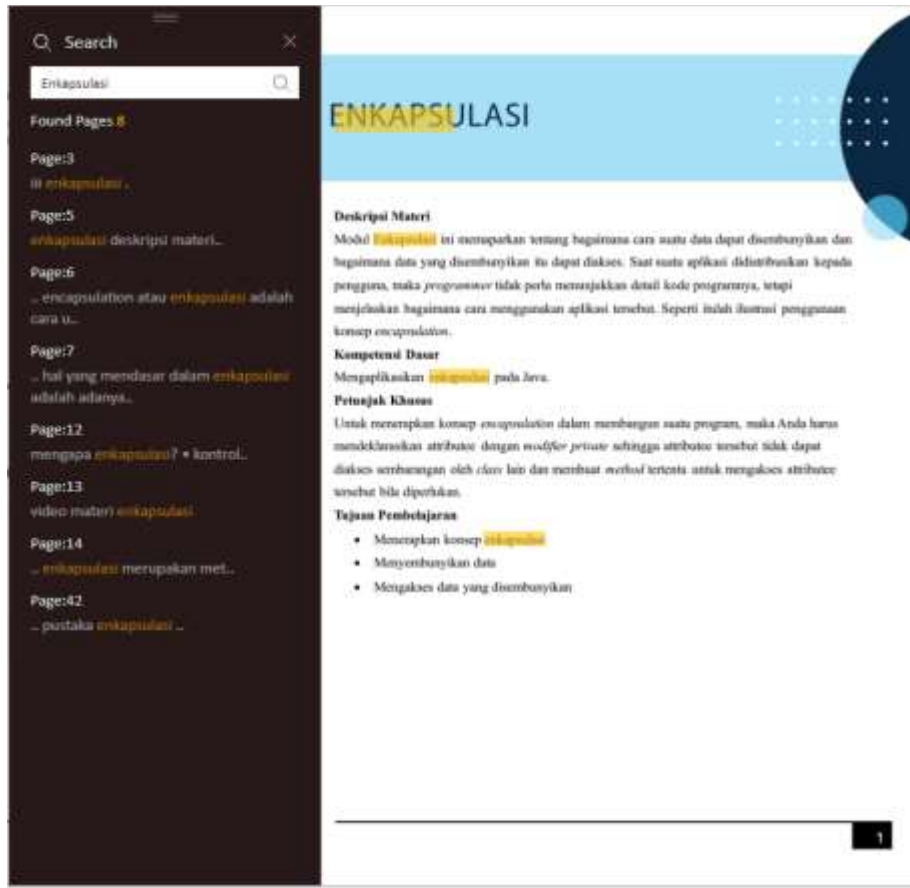
Tampilan pada mobile (IOS dan Android) dan Tampilan pada Desktop (Windows dan Mac)



Fungsi Search tool



Menampilkan kata kunci yang ingin dicari pada Flipbook



Fungsi Icon logo pada Flipbook



Icon logo Flipbook PBO Kelas XI ini dapat di klik untuk akses modul secara lengkap via Google Drives

ENKAPSULASI

Deskripsi Materi

Modul Enkapsulasi ini memaparkan tentang bagaimana cara suatu data dapat disembunyikan dan bagaimana data yang disembunyikan itu dapat diakses. Saat suatu aplikasi didistribusikan kepada pengguna, maka programmer tidak perlu menunjukkan detail kode programnya, tetapi menjelaskan bagaimana cara menggunakan aplikasi tersebut. Seperti itulah ilustrasi penggunaan konsep encapsulation.

Kompetensi Dasar

Mengaplikasikan enkapsulasi pada Java.

Petunjuk Khusus

Untuk menerapkan konsep encapsulation dalam membangun suatu program, maka Anda harus mendeklarasikan attributee dengan modifier private sehingga attributee tersebut tidak dapat diakses sembarangan oleh class lain dan membuat method tertentu untuk mengakses attributee tersebut bila diperlukan.

Tujuan Pembelajaran

- Menerapkan konsep enkapsulasi
- Menyembunyikan data
- Mengakses data yang disembunyikan

Peta Konsep

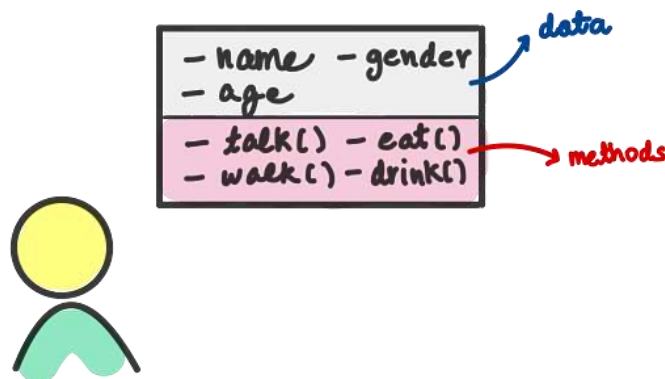
1. ENKAPSULASI Pada Pemrograman Berorientasi Objek (Java)

- Definisi Enkapsulasi
- Penyembunyian Dan Akses Data
 - Akses dan Penyembunyian Data Dengan Modifier
 - Contoh Penggunaan Modifier
- Modifier
 - Pembagian Modifier
- Get dan Set
 - Contoh Penggunaan Getter and Setter

Dasar Teori

1. Definisi Enkapsulasi

Encapsulation atau enkapsulasi adalah cara untuk menyembunyikan informasi detail dari suatu class. Ada dua jenis method yang digunakan dalam enkapsulasi, yaitu mutator method dan accessor method. Mutator method digunakan untuk mengubah atau memberi nilai variabel pada class, baik berupa instance maupun static, sedangkan accessor method digunakan untuk membaca nilai variabel pada class, baik berupa instance maupun static. Paling umum digunakan dalam bidang pemrograman berorientasi objek, enkapsulasi mengacu pada pengemasan data dan fungsi yang mewakili entitas yang dapat diwujudkan (dunia nyata) ke dalam lingkup yang dapat diprogram (biasanya kelas/objek). Data (variabel) di sini mewakili atribut dan properti yang menentukan unit, sedangkan fungsi melambangkan kemungkinan perilaku dan operasinya. Jadi, dengan kata lain, enkapsulasi adalah pengikatan data dan metode ke dalam satu entitas untuk mendefinisikan dirinya sendiri dan ruang lingkup operasinya. Contoh paling umum dari unit tersebut adalah pengaturan kelas/objek. Misalnya, kita merangkum aspek-aspek manusia menjadi entitas yang dapat diprogram. Dalam hal ini, data akan membedakan satu manusia dari manusia lainnya, dan metodenya akan menentukan kemungkinan operasi (perilaku/tindakan).



Pengikatan ini memberi makna pada entitas dan secara semantik menyerupai aspek objek dunia nyata yang coba dimodelkan.

2. Penyembunyian data dan pengaksesan data.

Dua hal yang mendasar dalam enkapsulasi adalah adanya data yang disembunyikan dan bagaimana cara untuk mengakses data tersebut. Information hiding adalah proses menyembunyikan informasi dari suatu class sehingga informasi tersebut tidak dapat diakses dari luar dengan cara memberikan modifier “private” ketika mendeklarasikan attribute atau method. Interface to access data adalah cara melakukan perubahan terhadap attribute yang telah disembunyikan. Caranya yaitu dengan membuat interface berupa method untuk menginisialisasi atau merubah nilai dari attribute tersebut.

```
class MyClass {  
    public int myPublicVar = 20;  
    protected int myProtectedVar = 10;  
    private int myPrivateVar = 30;  
}
```

Selanjutnya, buat kelas lain yang akan menampung metode utama. Nantinya juga akan mewarisi kelas sebelumnya di sini agar dapat bekerja dengan variabel yang dilindungi.

```
public class HelloWorld extends MyClass {  
    public static void main (String []args){  
        MyClass myObject = new MyClass(); // membuat objek class  
        System.out.println(myObject.myPublicVar); //print variabel  
public  
        System.out.println(myObject.myProtectedVar); // print variabel  
protected  
    }  
}
```

Maka keluaran yang dihasilkan adalah :

20

10

Ini berfungsi dengan baik karena myPublicVar bersifat publik. Variabel yang dilindungi (myProtectedVar) juga akan muncul karena kita mewarisi kelas lain (dan kedua kelas berada dalam paket yang sama). Sekarang mari kita lihat bagaimana keadaan berubah ketika kita mengakses variabel privat kita secara langsung (myPrivateVar).

```
public class HelloWorld extends MyClass { // extending MyClass -
- to be able to access protected variables
    public static void main(String []args){
        MyClass myObject = new MyClass();
        System.out.println(myObject.myPrivateVar); //
trying to directly access private variable -- will result in
error
    }
}
```

Keluaran :

```
Error: myPrivateVar has private access in MyClass
        System.out.println(myObject.myPrivateVar); //
trying to directly access private variable -- will result in
error
^1 error
```

Seperti yang diharapkan, ini gagal. Hal ini karena hanya elemen kelas itu sendiri yang dapat mengakses variabel private. Oleh karena itu, saatnya untuk menambahkan fungsi pengambil ke MyClass (seperti yang kita lakukan sebelumnya) dan memanggilnya melalui objek (myObject) untuk mendapatkan nilai variabel private.

```
class MyClass {
    public int myPublicVar = 20;
    protected int myProtectedVar = 10;
    private int myPrivateVar = 30;

    public int getMyPrivateVar () { // added (public) getter
function to access private var
        return this.myPrivateVar;
    }
}
```

Perhatikan konvensi penamaan di sini — fungsi pengambil untuk `myPrivateVar` menjadi `getMyPrivateVar` .

Sekarang di main class, kita dapat memanggil fungsi ini (karena bersifat publik) dan membaca variabel private, seperti yang ditunjukkan di bawah ini :

```
public class HelloWorld extends MyClass {  
  
    public static void main(String []args){  
        MyClass myObject = new MyClass();  
        System.out.println(myObject.getMyPrivateVar()); //  
        accessing private variable through getter function  
    }  
}
```

Keluaran :

30

Perhatikan bagaimana implementasi ini hanya memungkinkan kita membaca variabel dan karena itu masih mencegah modifikasi. Untuk penerapan keyword `extends` akan dijelaskan pada bab selanjutnya.

3. Modifier

Sekarang, Anda sudah cukup familiar dengan keyword "public" yang muncul di hampir semua contoh. Keyword public adalah pengubah akses, yang berarti bahwa keyword ini digunakan untuk mengatur tingkat akses untuk class, attribute, methods, dan constructor.

Pembagian Modifier (pengubah) ke dalam dua kelompok:

Access Modifier - mengontrol tingkat akses

Non-Access Modifier - tidak mengontrol tingkat akses, tetapi menyediakan fungsionalitas lain

Access Modifier (pengubah akses)

Untuk class, bisa menggunakan "public" atau "default"

Public : adalah class yang dapat diakses oleh semua class lainnya

Default : Class hanya dapat diakses oleh class dalam paket yang sama. Ini digunakan ketika Anda tidak menentukan pengubah.

Untuk attribute, metod dan constructor, dapat menggunakan salah satu dari berikut ini :

Public : kode dapat diakses untuk semua class

Private : kode hanya dapat diakses melalui deklarasi class

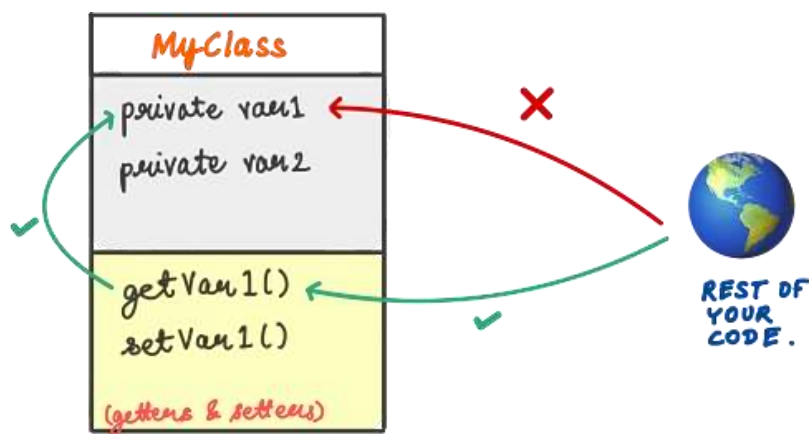
Default : kode hanya dapat diakses pada paket yang sama. Ini terjadi apabila tidak mencantumkan modifier

Protected : kode dapat diakses pada paket dan subklas yang sama

4. Get dan Set (Getter dan Setter)

variabel private hanya dapat diakses di dalam class yang sama (class luar tidak memiliki akses ke sana). Namun, dimungkinkan untuk mengaksesnya jika menyediakan methods "get" and "set" public. Methods "get" mengembalikan nilai variabel, dan methods "set" menetapkan nilainya.

Sintaks keduanya dimulai dengan "get" atau "set", diikuti dengan nama variabel, dengan huruf pertama dalam huruf besar.



Getter dapat mengizinkan (membaca) akses ke variabel private.

```
public class Person {  
    private String name; // private = restricted access  
  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```

Penjelasan :

Methods "get" mengembalikan nilai nama variabel. Methods "set" mengambil parameter (newName) dan menugaskannya ke variabel name. Keyword "this" digunakan untuk merujuk pada objek saat ini. Namun, karena variabel name dideklarasikan sebagai private, kita tidak dapat mengaksesnya dari luar class ini.

Contoh :

```
public class Main {  
    public static void main(String[] args) {  
        Person myObj = new Person();  
        myObj.name = "John"; // error  
        System.out.println(myObj.name); // error  
    }  
}
```

Jika variabel tersebut dideklarasikan sebagai public, kita akan mendapatkan keluaran sebagai berikut :

```
John
```

Namun, ketika mencoba mengakses variabel private, maka mendapatkan kesalahan

```
MyClass.java:4: error: name has private access in Person  
    myObj.name = "John";  
        ^  
MyClass.java:5: error: name has private access in Person  
    System.out.println(myObj.name);  
                        ^  
2 errors
```

Sebagai gantinya, kita menggunakan methods getName() dan setName() untuk mengakses dan memperbarui variabel :

```
public class Main {  
    public static void main(String[] args) {  
        Person myObj = new Person();  
        myObj.setName("John"); // Set the value of the name variable to "John"  
        System.out.println(myObj.getName());  
    }  
}  
  
// Outputs "John"
```

Mengapa Enkapsulasi?

- Kontrol yang lebih baik terhadap class attribute dan method.
- Class attribute dapat dibuat **read-only** (jika Anda hanya menggunakan methods get), atau **write-only** (jika Anda hanya menggunakan methods set).
- Fleksibel: programmer dapat mengubah satu bagian dari kode tanpa mempengaruhi bagian lainnya.
- Peningkatan keamanan data.
- Hal ini dapat mencegah akses yang tidak diinginkan ke data sensitif dan menyembunyikan informasi melalui pengubah akses sekaligus mengurangi perubahan yang dilakukan manusia secara keliru.
- Karena perubahan kode bersifat independen, enkapsulasi membuat pemeliharaan kode lebih mudah dikelola.





VIDEO MATERI



Kesimpulan

Enkapsulasi merupakan metode menyembunyikan data – data kelas suatu pemrograman ke dalam suatu unit tertentu. Ini bertujuan untuk memberikan proteksi terhadap data agar tidak diakses oleh pihak luar tanpa izin.

Cara menerapkan Enkapsulasi dalam bahasa pemrograman terbilang sangat mudah. Programmer hanya perlu mengidentifikasi data dan fungsi, lalu membuat kelas dengan metode private dan public.

Gunakan metode private untuk menyimpan data yang tidak bisa diakses oleh pihak lain. Lalu gunakan metode public untuk mengakses dan mengubah data kelas.

Menggunakan Enkapsulasi akan memberikan keuntungan. Pertama pengguna dapat memodifikasi data implementasi internal kelas tanpa mempengaruhi baris kode lainnya.

Kedua, pengguna dapat mengontrol dengan fleksibel terhadap akses ke data atau atribut kelas. Selain itu juga dapat mengatur aturan tertentu untuk melakukan modifikasi data saat membuat kelas.

Tugas

```
class Buku
{
    private String judulBuku="Konsep Dasar Pemrograman Java";
    private String pengarangBuku="Patrick Naughton";
    private int stokBuku=27;
    private int hargaBuku=75000;

    public void setPembeli(String nama)
    {
        namaPembeli=nama;
    }

    public void setAlamat(String alamat)
    {
        alamatPembeli=alamat;
    }

    public void setPembelian(int pembelian)
    {
        banyakPembelian=pembelian;
    }

    public void hitungBayar()
    {
        bayarBuku=hargaBuku*banyakPembelian;
    }

    public void hitungSisa()
    {
        sisaBuku=stokBuku-banyakPembelian;
    }
}
```

Lengkapi kode program di atas untuk membuat sebuah program berbasis console di bidang pertokoan! Terapkan penggunaan konsep encapsulation! Buatlah sebuah class lagi yang berisi method main untuk menjalankan program tersebut! Buatlah agar program dapat menerima input dari user! Lakukan kompilasi dan eksekusi program kemudian tunjukkan hasilnya !

INHERITANCE dan POLYMORPHISM

Deskripsi materi

Modul Inheritance ini memaparkan tentang bagaimana cara suatu class dapat mewariskan attributnya kepada class lain sehingga class lain juga dapat menggunakan attribut tersebut. Hal ini dilakukan untuk menghindari pengulangan penulisan kode program yang sama di class yang berbeda. Di sini kembali menerapkan konsep polymorphism dengan melakukan overriding terhadap method, yaitu dengan menulis kembali method dalam superclass pada subclass untuk melakukan modifikasi data.

Petunjuk Khusus

Untuk menerapkan konsep inheritance dalam membangun suatu program, maka Anda harus membuat setidaknya dua class, yaitu satu class sebagai superclass dan satu class lainnya sebagai subclass yang ditandai dengan penyertaan keyword `extends` dan nama superclass. Sebuah superclass dapat diturunkan kepada lebih dari satu subclass, tetapi satu subclass hanya bisa menerima pewarisan dari satu superclass saja.

Tujuan Pembelajaran

- Menerapkan konsep inheritance.
- Membuat superclass dan subclass.
- Menerapkan keyword `extends`.
- Menerapkan keyword `super` dan `this`.
- Memanggil method dari superclass.
- Memanggil constructor method dari superclass.
- Membuat overriding method dari superclass.

Peta Konsep

2. INHERITANCE dan POLYMORPHISM

→ Definisi Inheritance

- Penerapan Superclass (Parent) dan Subclass (Child)
- Contoh Inheritance
- Penggunaan Keyword Final

→ Penerapan Keyword Pada Inheritance

→ Polymorphism

- Pembagian Polymorphism
- Overloading
- Overriding
- Implementasi Pada Interface

Dasar Teori

1. Definisi Inheritance

Inheritance atau pewarisan merupakan konsep dalam pemrograman berorientasi objek yang digunakan untuk membuat suatu class berdasarkan class yang sudah ada sehingga memungkinkan suatu class mewarisi semua method dan variabel dari class yang sudah ada. Bila programmer ingin membatasi hak akses method dan variabel, maka dapat digunakan modifier `private` dan `protected`. Salah satu bentuk hubungannya adalah inheritance (pewarisan). Hubungan ini seperti hubungan keluarga antara orang tua dan anak. Sebuah class di Java, bisa memiliki satu atau lebih keturunan atau class anak. Class anak akan memiliki warisan properti dan method dari class ibu.

Ada 2 istilah yang sering digunakan dalam penerapan konsep inheritance, yaitu superclass dan subclass.

- Superclass (parent) digunakan untuk menyatakan class induk yang akan diwariskan atau diturunkan
- Subclass (child) merupakan sebutan untuk class yang menjadi turunan dari superclass.

Untuk mewarisi dari sebuah class, digunakan kunci `extends`. Pada contoh dibawah ini, class Car (subclass) mewarisi attribute dan methods dari class Car (superclass) :

- Kode program class Parent :

```
//superclass
class Parent
{
    private int a=3; //variabel dengan modifier private
    protected int b=5; //variabel dengan modifier protected

    protected void ParentMethod() //method dengan
modifier protected
    {
        System.out.println("\nMethod di dalam class Parent");
        System.out.println("Nilai a = "+a); //menampilkan
nilai variabel a
    }
}
```

- Kode program class Child :

```
//subclass
class Child extends Parent
{
    public void ChildMethod()
    {
        System.out.println("\nMethod di dalam class Child");
        System.out.println("Nilai b = "+b); //menampilkan nilai
variabel b
        ParentMethod(); //memanggil method dari superclass
    }
}
```

- Kode program class CallMethod :

```
public class CallMethod
{
    public static void main(String[] args)
    {

System.out.println("\n*****
*****");

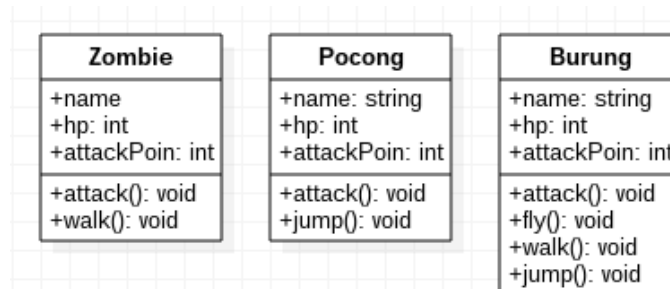
        System.out.println("\n\tMEMANGGIL METHOD PADA
SUPERCLASS");

System.out.println("\n*****
*****");

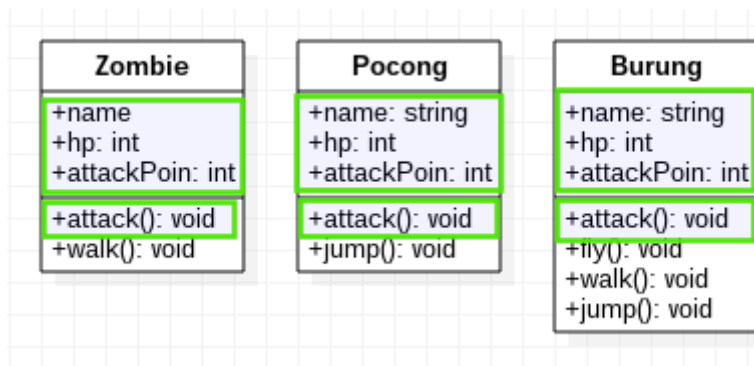
        Child call = new Child(); //membuat objek dari subclass
        call.ChildMethod(); //memanggil method dari subclass

System.out.println("\n*****
*****");
    }
}
```

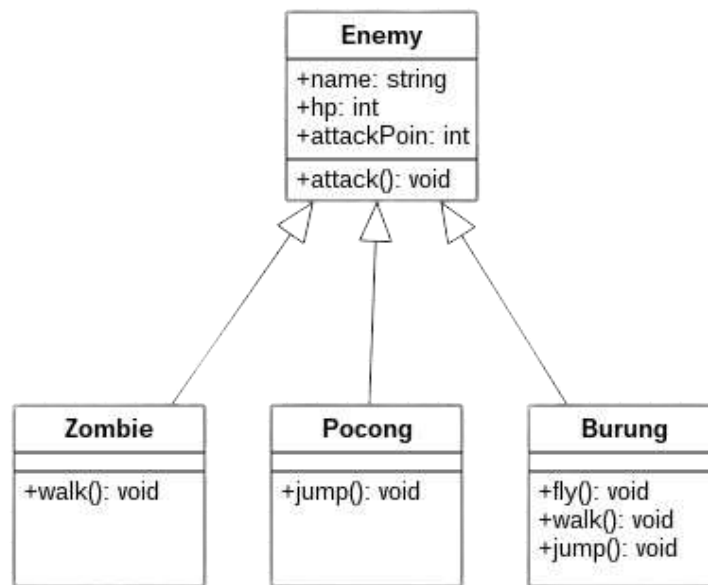
Contoh penggunaan Inheritance dengan perumpamaan dalam game dengan perilaku berbeda



Jika dilihat secara seksama, maka gambar tersebut memiliki beberapa kesamaan



Setelah menggunakan inheritance, maka akan menjadi seperti berikut



Class "Enemy" adalah class induk yang memiliki anak "Zombie", "Pocong", dan "Burung". Apapun property yang ada pada class parent, akan dimiliki juga oleh class child. Berikut adalah contoh pada superclass "Enemy" :

Pada file Enemy.java :

```
class Enemy {
    String name;
    int hp;
    int attackPoin;

    void attack(){
        System.out.println("Serang!");
    }
}
```

Pada class child akan menggunakan keyword “extends” untuk menyatakan class turunan dari “Enemy”.

Pada file Zombie.java :

```
class Zombie extends Enemy {  
    void walk(){  
        System.out.println("Zombie jalan-jalan");  
    }  
}
```

Pada file Pocong.java :

```
class Pocong extends Enemy {  
    void jump(){  
        System.out.println("loncat-loncat!");  
    }  
}
```

Pada file Burung.java :

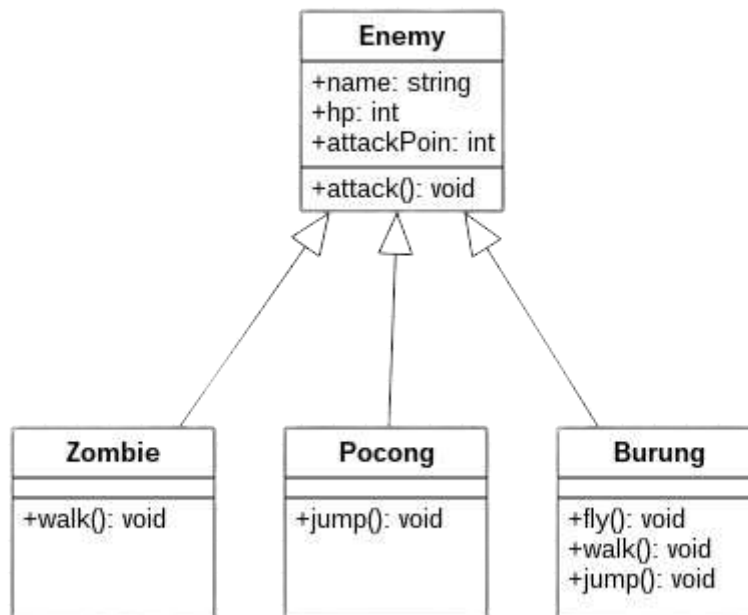
```
class Burung extends Enemy {  
    void walk(){  
        System.out.println("Burung berjalan");  
    }  
    void jump(){  
        System.out.println("Burung loncat-loncat");  
    }  
    void fly(){  
        System.out.println("Burung Terbang...");  
    }  
}
```

Lalu bila ingin membuat objek dari class tersebut maka dapat dibuat seperti berikut:

```
Enemy monster = new Enemy();  
Zombie zumbi = new Zombie();  
Pocong hantuPocong = new Pocong();  
Burung garuda = new Burung();
```

Contoh lain Inheritance adalah berikut :

Program yang akan dibuat agar berfungsi untuk menghitung luas dan keliling bangun datar. Berikut merupakan bentuk class diagram :



Maka dari diagram tersebut dapat dibuat classnya. Pertama buat package dengan nama "inheritance".

Membuat class Bangun datar

```
package inheritance;

public class BangunDatar {

    float luas(){
        System.out.println("Menghitung luas bangun datar");
        return 0;
    }

    float keliling(){
        System.out.println("Menghitung keliling bangun datar");
        return 0;
    }

}
```

Membuat class Persegi

```
package inheritance;

public class Persegi extends BangunDatar {
    float sisi;
}
```

Membuat class Lingkaran

```
package inheritance;

public class Lingkaran extends BangunDatar{

    // jari-jari lingkaran
    float r;

}
```

Membuat class Persegi panjang

```
package inheritance;

public class PersegiPanjang extends BangunDatar {
    float panjang;
    float lebar;
}
```

Membuat class Segitiga

```
package inheritance;

public class Segitiga extends BangunDatar {

    float alas;
    float tinggi;

}
```

Lalu buat main class

package inheritance;

```
public class Main {
    public static void main(String[] args) {

        // membuat objek bangun datar
        BangunDatar bangunDatar = new BangunDatar();

        // membuat objek persegi dan mengisi nilai properti
        Persegi persegi = new Persegi();
        persegi.sisi = 2;

        // membuat objek Lingkaran dan mengisi nilai properti
        Lingkaran lingkaran = new Lingkaran();
        lingkaran.r = 22;

        // membuat objek Persegi Panjang dan mengisi nilai
        properti
        PersegiPanjang persegiPanjang = new PersegiPanjang();
        persegiPanjang.panjang = 8;
        persegiPanjang.lebar = 4;

        // membuat objek Segitiga dan mengisi nilai properti
        Segitiga mSegitiga = new Segitiga();
        mSegitiga.alas = 12;
        mSegitiga.tinggi = 8;

        // memanggil method luas dan keliling
        bangunDatar.luas();
        bangunDatar.keliling();

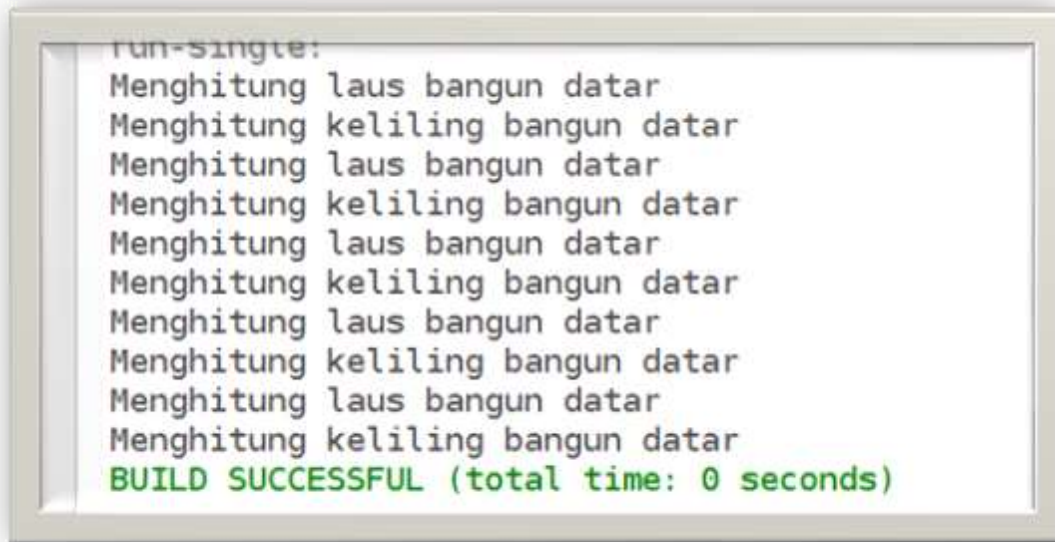
        persegi.luas();
        persegi.keliling();

        lingkaran.luas();
        lingkaran.keliling();

        persegiPanjang.luas();
        persegiPanjang.keliling();

        mSegitiga.luas();
        mSegitiga.keliling();
    }
}
```

Maka ketika main class dijalankan, hasilnya sebagai berikut :



```
run-Single:
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kenapa hasilnya bisa demikian ?

Karena yang kita panggil sebenarnya adalah method luas() dan keliling() milik parent (BangunDatar). Objek child dari BangunDatar belum memiliki method luas() dan keliling(), mengambil milik parent. Bagaimana jika ingin membuat semua class child memiliki method luas() dan keliling() yang berbeda dari parent ? Maka dapat menggunakan method overridin.

Contoh lain:

```
class Vehicle {
    protected String brand = "Ford";           // attribute Vehicle
    public void honk() {                       // method Vehicle
        System.out.println("Tuut, tuut!");
    }
}

class Car extends Vehicle {
    private String modelName = "Mustang";      // Car attribute
    public static void main(String[] args) {

        // Membuat Objek myCar
        Car myCar = new Car();

        // Memanggil the honk() method (dari class Vehicle) pada objek
        myCar
        myCar.honk();

        // Menampilkan nilai attribute brand (dari class Vehicle) dan
        nilai modelName dari class Car
        System.out.println(myCar.brand + " " + myCar.modelName);
    }
}
```

Bila diperhatikan ada modifier “protected” di Vehicle. Attribute brand di Vehicle diatur dengan modifier access “protected”. Jika diatur ke private, class Car tidak akan dapat diakses.

Keyword "Final"

Jika tidak ingin class lain mewarisi dari sebuah class, gunakan keyword Final :

```
final class Vehicle {  
    ...  
}  
  
class Car extends Vehicle {  
    ...  
}
```

Keluaran yang dihasilkan akan seperti berikut :

```
Main.java:9: error: cannot inherit from final Vehicle  
class Main extends Vehicle {  
        ^  
1 error)
```



2. Penerapan keyword extends, super, dan this

Keyword extends digunakan untuk mengaplikasikan konsep inheritance. Keyword ini menyatakan bahwa suatu class merupakan perluasan dari class lain yang dijadikan sebagai superclass. Keyword super digunakan oleh subclass untuk memanggil constructor pada superclass, sedangkan keyword "this" digunakan untuk mengakses variable instan dari objek dan menyatakan objek sekarang. Selain overriding, penerapan konsep polymorphism juga dapat dilakukan dalam bentuk overriding. Overriding dilakukan dengan cara menulis kembali method sama persis. Overriding method merupakan method pada subclass yang sama, tetapi berbeda implementasinya. Jadi, overriding method mempunyai nama method yang sama, jumlah parameter dan tipe parameter serta nilai kembalian (return) method yang di-override.

Method Overriding dilakukan saat kita ingin membuat ulang sebuah method pada sub-class atau class child. Method Overriding dapat dibuat dengan menambahkan anotasi Override di atas nama method atau sebelum pembuatan method.

Contoh penggunaan overriding pada file Persegi sebelumnya:

```
class Persegi extends BangunDatar {  
    float sisi;  
  
    @Override  
    float luas(){  
        float luas = sisi * sisi;  
        System.out.println("Luas Persegi: " + luas);  
        return luas;  
    }  
  
    @Override  
    float keliling(){  
        float keliling = 4 * sisi;  
        System.out.println("Keliling Persegi: " + keliling);  
        return keliling;  
    }  
}
```

Artinya dapat menulis ulang method "luas()" dan "keliling()" di class child. Maka dapat dibuat method overriding untuk semua class child.

```
package inheritance;

public class Lingkaran extends BangunDatar{

    // jari-jari lingkaran
    float r;

    @Override
    float luas(){
        float luas = (float) (Math.PI * r * r);
        System.out.println("Luas lingkaran: " + luas);
        return luas;
    }

    @Override
    float keliling(){
        float keliling = (float) (2 * Math.PI * r);
        System.out.println("Keliling lingkaran: " + keliling);
        return keliling;
    }

}
```

Dalam rumus luas dan keliling lingkaran dapat dimanfaatkan konstanta "Math.PI" sebagai nilai PI. Konstanta tersebut sudah ada di Java.

```

package inheritance;

public class PersegiPanjang extends BangunDatar {
    float panjang;
    float lebar;

    @Override
    float luas(){
        float luas = panjang * lebar;
        System.out.println("Luas Persegi Panjang:" + luas);
        return luas;
    }

    @Override
    float keliling(){
        float kll = 2*panjang + 2*lebar;
        System.out.println("Keliling Persegi Panjang: " + kll);
        return kll;
    }
}

```

```

package inheritance;

public class Segitiga extends BangunDatar {

    float alas;
    float tinggi;

    @Override
    float luas() {
        float luas = 1/2 * (alas * tinggi);
        System.out.println("Luas Segitiga: " + luas);
        return luas;
    }
}

```

Untuk class "Segitiga", hanya perlu melakukan override terhadap method "luas()". Karena untuk method "keliling()", segitiga memiliki rumus yang berbeda-beda. Karena kalau dapat diturunkan kembali, maka class "Segitiga" dapat menjadi "SegitigaSiku", "SegitigaSamaKaki", "SegitigaSamaSisi", dsb. Maka ketika dieksekusi class main akan memiliki hasil berikut :

```
compile-single:
run-single:
Menghitung laus bangun datar
Menghitung keliling bangun datar
Luas Persegi: 4.0
Keliling Persegi: 8.0
Luas lingkaran: 1520.5309
Keliling Lingkaran: 138.23007
Luas Persegi Panjang:32.0
Keliling Persegi Panjang: 24.0
Luas Segitiga: 96.0
Menghitung keliling bangun datar
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Polymoprhism

Polymorphism berarti banyak bentuk, dan ini terjadi ketika memiliki banyak class yang terkait satu sama lain melalui inheritance. Inheritance memungkinkan untuk mewarisi attribute dan methods dari class lain. polymorphism menggunakan methods tersebut untuk melakukan tugas-tugas yang berbeda. Hal ini memungkinkan untuk melakukan satu tindakan dengan cara yang berbeda. Bentuk yang dimaksud dapat diartikan sebagai isinya berbeda, parameternya berbeda, dan tipe datanya berbeda. Polymorphism pada java ada dua macam :

- Static polymorphism (Polimorfisme statis) yang hanya terjadi dalam satu class;
- Dynamic polymorphism (Polimorfisme dinamis) terjadi pada terhubung dengan class lain seperti inheritance.

Beda dari keduanya adalah pada cara pembuatan polimorfismenya.

Polymorphism static menggunakan method overloading, sedangkan Polymorphism dynamic menggunakan method overriding.

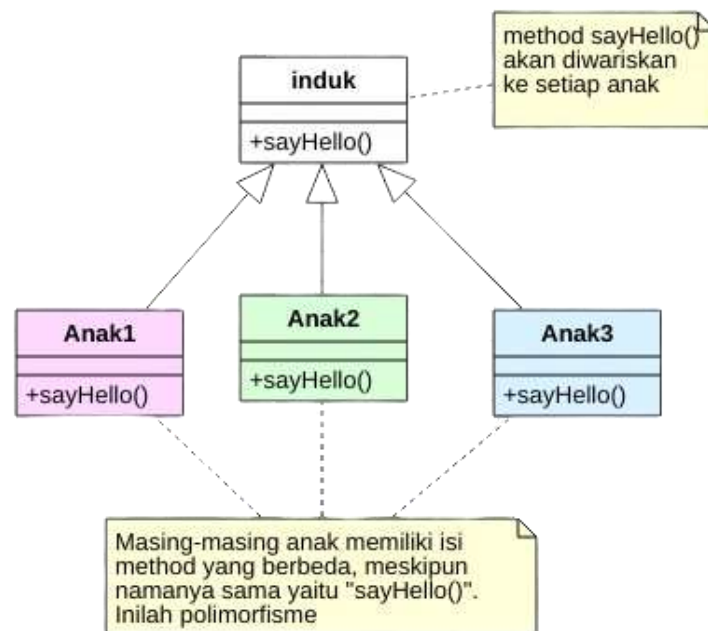
Method overloading terjadi pada sebuah class yang memiliki method yang sama Namun parameter dan tipe datanya berbeda. Yang perlu diingat untuk method overloading adalah : **Dalam satu class, Nama method sama, Tipe data dan parameter berbeda.**

Contoh overloading sebagai berikut :

```
class Lingkaran {  
  
    // method menghitung luas dengan jari-jari  
    float luas(float r){  
        return (float) (Math.PI * r * r);  
    }  
  
    // method menghitung luas dengan diameter  
    double luas(double d){  
        return (double) (1/4 * Math.PI * d);  
    }  
  
}
```

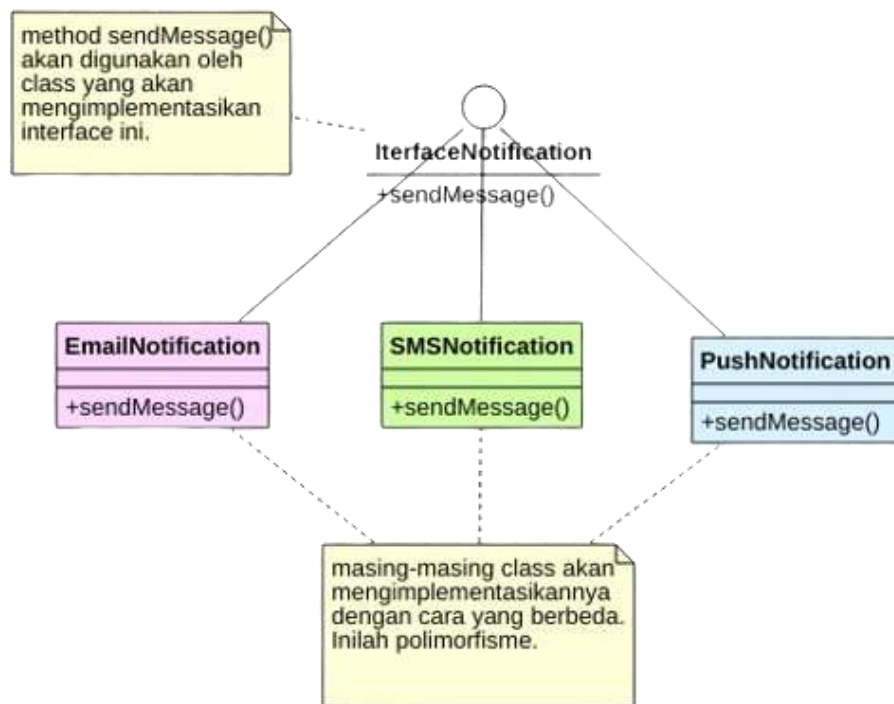
Pada class “Lingkaran” memiliki dua method yang sama yaitu “luas()” Namun parameter dan tipe datanya serta isi rumusnya juga berbeda.

Polymorphism dynamic terjadi saat menggunakan pewarisan (inheritance) dan implementasi interface. Pada saat atribut dan method dari class parent diwariskan ke class child maka class child akan memiliki method yang sama dengan class parent dan child lainnya, maka terjadilah polymorphism.



Class child akan memiliki method yang sama, Namun isi dan parameternya mungkin dapat berbeda dengan class parent. Hal tersebut karena class child melakukan method overriding (menindih method) yang diwariskan. Hal ini juga dapat terjadi saat menggunakan interface.

Contoh diagram pada implementasi interface



Interface adalah class kosong yang berisi nama-nama method yang nantinya harus diimplementasikan pada class lain. Dalam praktiknya bisa saja tiap-tiap class akan mengimplementasikan secara berbeda dengan nama method yang sama.

Sebagai contoh polymorphism, bayangkan sebuah superclass Bernama Animal yang memiliki method Bernama animalSound(). Subclass dari Animal bisa berupa Babi, Kucing, Anjing,

Burung, dan mereka juga memiliki implementasi suara hewan itu sendiri :

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}
```

Untuk mewarisi dari sebuah class, maka perlu menggunakan “extends”. Sekarang dapat membuat objek Pig dan Dog kemudian memanggil method animalSound() pada keduanya.

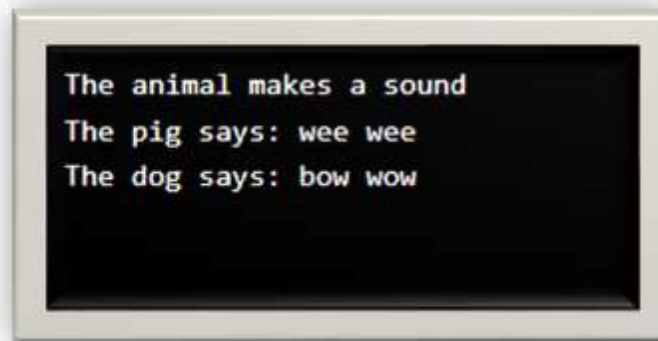
```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}

class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal(); // Create a Animal object
        Animal myPig = new Pig(); // Create a Pig object
        Animal myDog = new Dog(); // Create a Dog object
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

Keluaran yang dihasilkan akan seperti contoh berikut :



Mengapa dan kapan menggunakan "Inheritance" dan "polymorphism" ?

Berguna untuk penggunaan ulang kode:

- Kode Dapat Digunakan Kembali : Kode yang ditulis di Superclass bersifat umum untuk semua subkelas. Class child dapat langsung menggunakan kode kelas induk.
- Method Overriding : hanya dapat dicapai melalui inheritance. Ini adalah salah satu cara Java mencapai Run Time Polymorphism.





VIDEO MATERI



Kesimpulan

Inheritance merupakan pilar penting dari OOP (Pemrograman Berorientasi Objek). Ini adalah mekanisme di Java dimana satu kelas diperbolehkan mewarisi fitur (field dan method) dari kelas lain.

Inheritance berarti membuat kelas baru berdasarkan kelas yang sudah ada. Sebuah kelas yang mewarisi dari kelas lain dapat menggunakan kembali metode dan bidang kelas tersebut. Selain itu, Anda juga dapat menambahkan bidang dan metode baru. Ibarat ibu yang memiliki anak, maka terdapat kemiripan si ibu dari anak-anaknya.

Dengan inheritance maka tidak perlu menulis kode secara berulang-ulang, cukup dengan menulis class parent, maka hanya perlu sedikit merubah atribut atau method yang diperlukan sesuai kebutuhan sehingga dapat menghemat waktu.

Polymorphism memungkinkan kita melakukan suatu tindakan dengan cara yang berbeda. Dengan kata lain, polimorfisme memungkinkan Anda mendefinisikan satu antarmuka dan memiliki banyak implementasi. Polymorphism memiliki bentuk method yang berbeda-beda namun memiliki nama yang sama.

Jika ada beberapa fungsi dengan nama yang sama tetapi parameternya berbeda maka fungsi tersebut dikatakan overloading . Fungsi dapat di-overloading karena perubahan jumlah isi atau/dan perubahan jenis isi.

Overriding adalah proses di mana pemanggilan fungsi ke metode yang diganti akan diselesaikan pada saat eksekusi. Jenis polymorphism ini dicapai dengan method overriding. Overriding terjadi ketika kelas turunan memiliki definisi untuk salah satu fungsi anggota class. Fungsi tersebut akan ditimpa.

Tugas

Menerapkan konsep inheritance dan membuat overriding method.

- a. Buatlah sebuah program berbasis console untuk menerapkan konsep inheritance dalam kehidupan sekolah ! Buatlah overriding method dalam program! Aturlah agar user dapat melakukan input data saat menjalankan program! Program ini minimal terdiri dari 4 class, yaitu 1 class yang pertama sebagai superclass, 1 class yang kedua sebagai subclass dari class pertama, 1 class yang ketiga sebagai subclass dari class kedua dan 1 class lagi untuk menjalankan program ini. Lakukan kompilasi dan eksekusi program kemudian tunjukkan hasilnya!
- b. Berikan penjelasan terkait jalannya program yang Anda buat !

Daftar Pustaka

- W3Schools. Java Classes : Java Encapsulation. Diakses 2 Mei 2024 dari :
https://www.w3schools.com/java/java_encapsulation.asp
- Khanna Mukul, Scoutapm. (7 Desember 2021). Encapsulation in OOP : Definition and Example. Diakses pada 27 April 2024 dari :
https://scoutapm.com/blog/what-is-encapsulation#h_93840447845461638888736229
- W3Schools. Java Classes : Java Modifier. Diakses 2 Mei 2024 dari :
https://www.w3schools.com/java/java_modifiers.asp
- ITBOX. (23 April 2024). Apa itu Encapsulation : Konsep OOP untuk Pemula. Diakses pada 6 Mei 2024 dari : <https://itbox.id/blog/apa-itu-encapsulation-konsep-oop-untuk-pemula/>
- Jagoan Hosting. (19 September 2023). Pengertian OOP (Object Oriented Programming) dan 4 Prinsipnya. Diakses pada 5 Mei 2024 dari :
<https://www.jagoanhosting.com/blog/oop-adalah/>
- Geekforgeeks. (14 Mei 2024). Inheritance in Java. Diakses pada 24 Mei 2024 dari :
<https://www.geeksforgeeks.org/inheritance-in-java/>
- W3Schools. Java Classes : Java Inheritance. Diakses pada 23 Mei 2024 dari :
https://www.w3schools.com/java/java_inheritance.asp
- Ahmad Muhardian. Petanikode. (25 Desember 2017). Belajar Java OOP : Memahami Inheritance dan Method Overriding. Diakses pada 19 Mei 2024 dari : <https://www.petanikode.com/java-oop-inheritance/>
- Ahmad Muhardian, Petanikode. (27 Desember 2019). Belajar Java OOP : Memahami Prinsip Polimorfisme dalam OOP. Diakses pada 19 Mei 2024 dari : <https://www.petanikode.com/java-oop-polimorfisme/>
- Geekforgeeks. (1 November 2023). Polymorphism in Java. Diakses pada 17 Mei 2024 dari : <https://www.geeksforgeeks.org/polymorphism-in-java/>
- W3School. Java Classes : Java Polymorphism. Diakses pada 17 Mei 2024 dari :
https://www.w3schools.com/java/java_polymorphism.asp