

**PEMROGRAMAN BERORIENTASI OBJEK/OBJECT ORIENTED
PROGRAMMING (OOP)**



ARMAN SYAM

(200250502017)

EVIL MARDJANI

(210250502093)

**TEKNIK INFORMATIKA A
FAKULTAS ILMU KOMPUTER
UNIVERSITAS TOMAKAKA
TAHUN 2022**

PEMROGRAMAN BERORIENTASI OBJEK

Arman Syam - 200250502017

Evil Mardjani - 210250502093

Teknik Informatika
Universitas Tomakaka

Abstrak

OOP atau Object Oriented Programming adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Jadi setiap bagian dari suatu permasalahan adalah objek, untuk objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi. Sebagai contoh sebuah sepeda. Sepeda itu sendiri terbentuk dari beberapa objek yang lebih kecil lagi seperti roda, kursi, pedal, dll. Sepeda sebagai objek yang terbentuk dari objek-objek yang lebih kecil saling berhubungan, berinteraksi, berkomunikasi dan saling mengirim pesan kepada objek-objek yang lainnya. Begitu juga dengan program, sebuah objek yang besar dibentuk dari beberapa objek yang lebih kecil, objek-objek itu saling berkomunikasi, dan saling berkirip pesan kepada objek yang lain.

Kata Kunci : OOP (Object Oriented Programming)

1. PENDAHULUAN

OOP atau Object Oriented Programming adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Jadi setiap bagian dari suatu permasalahan adalah objek, untuk objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi. Sebagai contoh sebuah sepeda. Sepeda itu sendiri terbentuk dari beberapa objek yang lebih kecil lagi seperti roda, kursi, pedal, dll. Sepeda sebagai objek yang terbentuk dari objek-objek yang lebih kecil saling berhubungan, berinteraksi, berkomunikasi dan saling mengirim pesan kepada objek-objek yang lainnya. Begitu juga dengan program, sebuah objek yang besar dibentuk dari beberapa objek yang lebih kecil, objek-objek itu saling berkomunikasi, dan saling berkirim pesan kepada objek yang lain. Jika biasanya developer merangkai kode berdasarkan function dan logic, dengan OOP developer dapat mengembangkan software yang terbagi

dalam objek-objek tertentu. Memang, apa sebenarnya OOP itu?.

Object-oriented programming atau OOP adalah suatu metode pemrograman yang berorientasi pada objek. Program-program yang telah ada merupakan gabungan dari beberapa komponen-komponen kecil yang sudah ada sebelumnya. Hal itu dapat mempermudah pekerjaan seorang programmer dalam melakukan pengembangan program. Objek-objek yang saling berkaitan dan disusun kedalam satu kelompok ini disebut dengan class. Nantinya, objek-objek tersebut akan saling berinteraksi untuk menyelesaikan masalah program yang rumit. Jika sebelumnya developer harus berfokus pada logic yang akan dimanipulasi, dengan OOP, developer dapat lebih terfokus pada objeknya saja untuk dimanipulasi. Pendekatan ini menawarkan cara yang mudah untuk menangani kerumitan suatu pemrograman. Tujuan utama OOP adalah untuk mengatasi kelemahan pendekatan pemrograman konvensional.

2. METODE

2.1 Jenis Penelitian

Pendekatan penelitian kualitatif merupakan pendekatan yang lebih menekankan pada aspek pemahaman secara mendalam terhadap suatu masalah daripada melihat permasalahan untuk penelitian generalisasi. Metode penelitian ini lebih suka menggunakan teknik analisis mendalam (in-depth analysis), yakni mengkaji masalah secara kasus per kasus karena metodologi kualitatif yakin bahwa sifat suatu masalah satu akan berbeda dengan sifat dari masalah lainnya.

Case studies merupakan penelitian kualitatif dimana peneliti melakukan eksplorasi secara mendalam terhadap program, kejadian, proses, aktivitas, terhadap satu atau lebih orang. Suatu kasus terikat oleh waktu dan aktivitas dan peneliti melakukan pengumpulan data secara mendetail dengan menggunakan berbagai prosedur pengumpulan data dan dalam waktu yang berkesinambungan.

2.2 Teknik Pengumpulan Data

Yaitu merupakan langkah awal dalam metode pengumpulan data. Studi pustaka merupakan metode pengumpulan data yang diarahkan kepada pencarian data dan informasi melalui dokumen-dokumen, baik dokumen tertulis, foto-foto, gambar, maupun dokumen elektronik yang dapat mendukung dalam proses penulisan.”Hasil penelitian juga akan semakin kredibel apabila didukung foto-foto atau karya tulis akademik dan seni yang telah ada.(Sugiyono,2005:83).

3. PEMBAHASAN

Untuk dapat menguasai pemrograman Java, harus mengerti dengan baik konsep pemrograman berorientasi objek, karena Java merupakan bahasa pemrograman berorientasi objek. Pada bagian ini akan dibahas konsep-konsep penting dalam pemrograman berorientasi objek, sehingga diharapkan akan lebih mudah dalam mempelajari bahasa Java. Beberapa konsep OOP dasar, antara lain Encapsulation (Class and Object), Inheritance (Penurunan sifat), Polymorphisme, Access

Modify, Constructor, Destructor, Static Properties, super class serta sub class.

3.1 Course Intruduction

OOP adalah sebuah istilah yang diberikan kepada bahasa pemrograman yang menggunakan teknik berorientasi atau berbasis pada sebuah objek dalam pembangunan program aplikasi, maksudnya bahwa orientasi pembuatan program tidak lagi menggunakan orientasi linear melainkan berorientasi pada objek-objek yang terpisah-pisah. Suatu perintah dalam bahasa ini diwakili oleh sebuah objek yang didalamnya berisi beberapa perintahperintah standar sederhana. Objek ini dikumpulkan dalam Modul form atau Report atau modul lain dan disusun didalam sebuah projek. (Danuri, 2009). standar yang paling rendah.

Konsep OOP bermula pada era 1960-an. Sebuah bahasa pemrograman Simula memperkenalkan berbagai

konsep yang mendasari OOP dengan SIMULA I (1962-65) dan Simula 67 (1967). Kemudian pada tahun 70-an, bahasa pemrograman Smalltalk menjadi yang pertama kali disebut object-oriented. Enkapsulasi adalah suatu cara untuk menyembunyikan informasi detail dari suatu class. Dua hal yang mendasar dalam enkapsulasi yakni : Information hiding , Interface to access data. Anggota class dapat diakses baik berupa atribut maupun method secara langsung dengan menggunakan objek yang dibuat. Hal ini dikarenakan akses kontrol yang diberikan kepada atribut maupun method yang ada di dalam class tersebut adalah 'public'. Informasi dapat disembunyikan dari suatu class sehingga anggota class tersebut tidak dapat diakses dari luar, caranya adalah hanya dengan memberikan akses kontrol 'private' ketika mendeklarasikan atribut atau method. Proses ini disebut dengan information hiding.

3.1.1 Cara Kerja Oop

Langkah pertama dalam OOP yaitu mengidentifikasi semua objek yang ingin dimanipulasi oleh programmer dan bagaimana mereka saling berhubungan. Praktiknya sering dikenal sebagai data modelling atau pemodelan data. Setelah suatu objek diketahui, itu kemudian digeneralisasikan sebagai object class (kelas objek) yang mendefinisikan jenis data yang dikandungnya dan urutan logika apa pun yang dapat memanipulasinya. Setiap urutan logika yang berbeda dikenal sebagai method dan objek dapat berkomunikasi dengan antarmuka yang didefinisikan dengan baik yang disebut message.

Sederhananya, OOP berfokus pada objek yang

ingin dimanipulasi developers atau pengembang daripada logika yang diperlukan untuk memanipulasi mereka. Pendekatan pemrograman ini sangat cocok untuk program yang besar, kompleks, dan diperbarui atau dipelihara secara aktif. Karena pengorganisasian program berorientasi objek, metode ini juga kondusif untuk pengembangan kolaboratif di mana proyek dapat dibagi menjadi kelompok-kelompok. Adapun keuntungan tambahan dari OOP yaitu termasuk penggunaan kembali kode, skalabilitas dan efisiensi.

3.1.2 Konsep Dasar Oop Pemrograman

berbasis objek merupakan dari sebuah konsep dari objek elemen dasar program pengenalan

tentang object oriented programming atau oop dalam dunia nyata kita dapat menemukan objek disekitar kita seperti mobilmanusia motor dan lain lain yang ada disekitar kita secara umum ada beberapa kerangka dalam object oriented programming yaitu. Pemrograman berorientasi objek inggris. Sebagai contoh class of dog adalah suatu unit yang terdiri atas definisi definisi data dan fungsi fungsi yang menunjuk pada berbagai macam perilaku turunan dari anjing.

Oop dirancang pada konsep tertentu untuk mencapai tujuannya mengatasi kelemahan pendekatan pemrograman konvensional. Kelas kumpulan atas definisi data dan fungsi fungsi dalam suatu unit untuk

suatu tujuan tertentu. Konsep dari object oriented programming oop sendiri adalah lebih dari sekedar sebuah konsep pemrograman dimana cara berpikir tentang aplikasi yang mempelajari untuk memandang bahwa aplikasi bukan sekedar prosedur melainkan sebagai objek dan real entity.

3.1.3 Prinsip OOP

Dalam proses teknisnya OOP memiliki prinsip-prinsip yang khusus, dapun keempat prinsip OOP tersebut antara lain adalah sebagai berikut :

a. Enkapsulasi

Prinsip enkapsulasi atau kapsulisasi dalam OOP dilakukan ketika setiap objek dalam pemrograman dapat mempertahankan keadaan privat di

dalam sebuah kelas-kelas cetak biru. Dengan begitu, objek lain tidak dapat mengakses status objek tersebut secara langsung. Meski begitu, objek lain tetap dapat memanggil daftar fungsi publik karena objek mengelola statusnya sendiri melalui fungsi-fungsi publik ini.

Enkapsulasi secara umum adalah klasifikasi dari logika objek. Seperti misalnya memasukkan semua logika yang berkaitan dengan kelas B ke dalam variabel privat terkait B. Misalnya kelas Harimau, logika-logika yang masuk dalam kelas ini adalah mengaum, belang, karnivora, dan lainnya sesuai sifat B.

Namun tidak menutup kemungkinan ada variabel publik yang umum diterima oleh objek kelas lain seperti makan, berburu, berlari, dan sebagainya.

b. Abstraction

Abstraction atau abstraksi merupakan perpanjangan tangan enkapsulasi. Abstraksi merupakan proses pemilihan data dari kumpulan yang lebih besar untuk menunjukkan hanya detail relevan saja yang bisa menjadi objek. Sebagai permisalan, kamu ingin membuat aplikasi kencan daring, maka kemungkinan besar kamu akan diminta untuk mengumpulkan semua informasi tentang pengguna aplikasi tersebut. Hal

ini meliputi identitas, nomor kontak, makanan favorit, hobi, dan lain sebagainya. Jumlah data ini tentu saja amat banyak dan tidak semuanya diperlukan untuk membuat sebuah aplikasi kencan daring. Untuk memilih informasi mana yang relevan dengan aplikasi tersebut maka dilakukan proses pengambilan dan pemilihan informasi pengguna dari sebuah kumpulan besar data pengguna tadi. Proses penyaringan ini yang kemudian disebut sebagai abstraksi. Keuntungan abstraksi adalah pengembang bisa menerapkan informasi yang sama dari satu aplikasi satu ke aplikasi lainnya

dengan atau tanpa modifikasi.

c. Inheritance

Inheritance

merupakan kemampuan suatu objek untuk memperoleh beberapa atau bahkan semua properti dari objek lain dalam pemrograman OOP. Hal ini bisa dimisalkan dari seorang anak mewarisi sifat-sifat orang tuanya.

Inheritance

merupakan prinsip OOP yang dapat digunakan kembali sebagai keuntungan utama dari sebuah objek. Dalam bahasa pemrograman Java misalnya dikenal beberapa jenis inheritance seperti tunggal, ganda, hierarkis, hibrida, dan bertingkat.

Misalkan, objek Anggur masuk dalam klasifikasi Buah. Lalu, Anggur ini memperoleh properti dari kelas Buah seperti mengandung biji, berdaging, dan sebagainya. Hal ini serupa dengan objek lain dari kelas Buah tersebut sehingga sub-kelas atau objek Anggur memperoleh sifat-sifat utama dari kelas Buah ditambah dengan beberapa sifat unik objek itu sendiri.

d. Polimorfisme

Polimorfisme merupakan prinsip OOP yang memberi cara pada pengembang untuk menggunakan kelas atau klasifikasi objek persis seperti induknya sehingga tidak ada kebingungan dengan tipe-tipe campuran.

Hal ini disebabkan karena sub-kelas atau objek menyimpan fungsi dan metodenya sendiri secara unik.

Jika pengembang memiliki sebuah kelas umum seperti Tumbuhan, maka di dalamnya ada beberapa metode seperti bentuk daun, batang, waktu tumbuh, dan lain-lain sebagai fungsi dan metode umumnya. Sedangkan di dalam kelas Tumbuhan ada sub kelas atau objek seperti Durian, Rumput Gajah, Pakis, dan lainnya yang masing-masing memiliki ciri khas sendiri namun tetap mempunyai fungsi atau metode dari induk kelasnya yaitu Tumbuhan dan meliputi bentuk daun, batang, dan waktu

tumbuh. Hal ini yang lantas disebut sebagai prinsip polimorfisme dalam OOP.

3.1.4 Kelebihan Oop

Selain mendukung banyak bahasa pemrograman, berikut kelebihan dari OOP adalah sebagai berikut:

- a. Maintenance program lebih mudah

Kelebihan OOP pertama adalah masalah maintenance program yang lebih mudah. Programmer bisa lebih mudah dalam membaca dan memahami script program yang ditulis dengan melihat dokumentasi program.

- b. Update program lebih mudah

Update fitur pada program menjadi salah satu hal yang wajar kita ditemui. Dengan program yang dibangun dengan

konsep OOP, programmer bisa dengan mudah menambahkan fitur-fitur baru yang ada pada program dengan mudah. Jadi programmer tidak perlu mempelajari script dari awal sampai akhir seperti program yang dibuat dengan prosedural.

- c. Proses development lebih cepat

OOP memiliki banyak support library object yang bisa digunakan kembali, sehingga proses development bisa berjalan lebih cepat.

- d. Cost lebih rendah

Proses development yang cepat tentu akan membuat cost menjadi lebih hemat.

Script program lebih rapi dan dapat

digunakan berulang.
Kelebihan terakhir
script program yang
dibuat bisa lebih rapi
dan lebih pendek.
Script yang ada pun
bisa digunakan
berulang hanya
dengan memanggil
module yang sudah
ada.

3.1.5 Kekurangan Oop

Nah disamping
banyaknya kelebihan
yang dimiliki, ada
beberapa kekurangan dari
OOP yang perlu Anda
ketahui, yaitu :

- a. Kompleks untuk
dipelajari

Walaupun secara
fungsi OOP ini bagus,
tetapi bagi para
programmer OOP ini
memerlukan waktu
untuk bisa
memahaminya.

Program OOP
memiliki beberapa
konsep yang perlu
diketahui seperti

encapsulation,
abstraction,
inheritance,
polymorphism.

- b. Ukuran program OOP
lebih besar

Ukuran program
yang dihasilkan oleh
OOP lebih besar
dibandingkan

program yang
dikerjakan
menggunakan POP
(Pemrograman
Berbasis Prosedur)
atau dikenal dengan
prosedural.

- c. Runtime program
OOP lebih lambat

Karena ukurannya
yang lebih besar,
program OOP
memiliki runtime
yang lebih lambat
dibandingkan dengan
program prosedural.

- d. Tidak semua masalah
bisa diatasi dengan
OOP

Tidak semua
program bisa

dikerjakan dengan OOP, ada beberapa program yang memang lebih cocok menggunakan prosedural atau programming style lainnya.

3.2 Classes and project

Bahasa Java merupakan bahasa pemrograman yang berorientasi obyek sehingga konsep obyek dan kelas (class) menjadi penting. Dalam dunia nyata, ada banyak obyek misalnya obyek orang, obyek mobil, obyek pohon dsb. Misalkan obyek dalam dunia nyata adalah obyek orang. Obyek tersebut dapat diceritakan tentang ciri-cirinya, yaitu tinggi badan, berat badan, warna rambut, warna kulit, jenis kelami, menggunakan kacamata dll. Dalam pemrograman, obyek di dunia nyata akan menjadi CLASS, dan ciri-ciri obyek akan menjadi variabel class yang disebut sebagai DATA MEMBER.

Class Handphone	class Mobil	Class Orang
{	{	{
String merk;	// Variabel	String nama;
String tipe;	class	Int
String warna;	private int	tinggi_bada
double harga;	warna;	n; Int
}	private	berat_bada
	String merek;	n; String
	}	warna_kuli
		t; Boolean
		berkacama;
		}

Berdasarkan contoh diatas, pendefinisian pada pemrograman adalah :

Struktur pembuatan class, adalah sebagai berikut:

1	class NamaClass
2	{
3	//Isi class
4	}

Keterangan:

- Nama_Kelas harus sesuai dengan nama file.
- Contoh: class Handphone, maka nama filenya harus

diberi nama dengan
Handphone.java.

Atribut

Atribut merupakan ciri-ciri yang melekat pada suatu objek. Berikut adalah contoh *syntax* atribut.

```
[access_modifier]          [tipe_data]  
[nama_variabel] = [value];
```

Keterangan:

- [access_modifier] digunakan untuk memberi batasan hak *class* maupun *method*.
- *Access modifier* akan dijelaskan pada sub bab berikutnya
- [tipe_data] menjelaskan apakah variabel tersebut bertipe *String*, *int*, *double*, dan sebagainya
- [nama_variabel] merupakan sebutan (definisi) variabel tersebut
- [value] merupakan nilai dari variable tersebut

Contoh: private String warna =
“merah”;

Method

Method merupakan tindakan/aksi yang bisa dikerjakan oleh CLASS. Method merupakan fungsi-fungsi yang digunakan untuk memanipulasi nilai-nilai pada atribut dan/atau untuk melakukan hal-hal yang dapat dilakukan oleh objek itu sendiri. Dalam hal ini method dapat berisi sekumpulan program yang telah terbungkus. Dengan method, kita bias memanggil kumpulan program tersebut hanya dengan memanggil nama methodnya sehingga pekerjaan jadi lebih singkat dan tidak boros menuliskan program. Selain itu, program menjadi lebih terstruktur, praktis, dan efisien.

Contoh:

METHOD Class Handphone	METHOD Class Mobil	METHOD Class Orang
1. Memasukan warna 2. Memasukkan tipe 3. Memasukkan merek	1. Memasukan data mobil 2. Gerak mobil kiri 3. Menampilkan informasi	1. Memasukan data orang 2. Tertawa 3. Menangis 4. Menampilkan informasi

Method yang mengembalikan nilai biasanya berupa sub program berjenis fungsi. Sedangkan method yang tidak mengembalikan nilai biasanya berupa sub program berjenis prosedur.

Berikut adalah contoh *syntax* pembuatan method.

```
[access_modifier]
[tipe_data]
nama_method(.....)
```

Keterangan:

- [access_modifier] digunakan untuk memberi

batasan hak *class* maupun *method*. *Access modifier* akan dijelaskan pada sub bab berikutnya

- [tipe_data] menjelaskan apakah variabel tersebut bertipe *String*, *int*, *double*, dan sebagainya
- [nama_method] merupakan sebutan (definisi) method tersebut. Umumnya method selalu diakhiri dengan tanda kurung () (...) berisi parameter

apabila diperlukan.

Contoh

```
public void
masuk_info_mobil(int
mrk,String nm)

{
    this.merek = mrk;
}
```

Implementasi method dalam pemrograman adalah:

```
public void menangis()

{
}
```

```
public void tertawa()
```

```
{
```

Dalam pemrograman, suatu class dapat mempunyai banyak objek dan objek akan mewarisi data member dan method yang sama dari suatu class. Objek disebut juga *instance of Class* merupakan objek yang diinstan atau dibuat dari *class*. Untuk membuat obyek dari class Orang, digunakan keyword „new“. Contoh:

```
Orang org1 = new Orang ("Samuel");
```

```
Orang org2 = new Orang ("Ari");
```

Sehingga class Orang , saat ini mempunyai 2 obyek.\

Jika telah dilakukan information hiding terhadap suatu atribut pada suatu class, lalu bagaimana cara melakukan

perubahan terhadap atribut yang disembunyikan tersebut, caranya adalah dengan membuat suatu interface berupa method untuk menginisialisasi atau merubah nilai dari suatu atribut tersebut yang dinamakan dengan interface to access data.

Source Code Sample :

```
public class Siswa
```

```
{
```

```
    public String nama;
```

```
    public String nrp;
```

```
    public void Info()
```

```
{
```

```
    System.out.println("Saya  
adalah");
```

```
    System.out.println("Nama " +  
nama);
```

```
    System.out.println("Nrp " +  
nrp);
```

```
}
```



```

}

public class IsiData
{
    public static void main(String[]
args)
    {
        Siswa IT = new Siswa();

        IT.nama = "Mirna";

        IT.nrp = "0320110013";

        IT.Info();
    }
}

```

Source Code Sample :

```

/*
    Disimpan dalam file
    "manusia.java"
*/

class manusia
{
    public String nama;

```

```

public
    manusia(String n)
    {
        this.nama = n;
    }

    public String
    tampilkanNama()
    {
        return nama;
    }

    public void makan()
    {

        System.out.println("Nyamy
am... nyam...");
    }

    public void kerja()
    {

        System.out.println("Kerjaer
jaaa...");
    }

    private void bunuhDiri()

```

```

    {

        System.out.println("Dor...br
        uk...");

    }

}

/*

    Disimpan dalam file
    "andi.java"

    */

    class andi

    {

        public static void
        main(String arg[])

        {

            manusia andi= new
            manusia("Andi");

            System.out.println("Nama=
            "+ andi.tampilkanNama());

            andi.makan();

        }

    }

```

3.3 Encapsulation

Encapsulation adalah salah satu dari empat konsep OOP fundamental. Tiga lainnya adalah pewarisan, polimorfisme, dan abstraksi. Enkapsulasi adalah mekanisme membungkus data (variabel) dan kode yang bekerja pada data (methode) yang bersama-sama sebagai satu kesatuan. Didalam enkapsulasi, variabel kelas akan disembunyikan dari kelas-kelas lain, dan dapat diakses hanya melalui method kelas itu sendiri. Oleh karena itu, juga dikenal sebagai persembunyian data.

Untuk membuat enkapsulasi di Java:

- Mendeklarasikan variabel kelas sebagai private.
- Menyediakan method setter dan getter umum untuk memodifikasi dan melihat nilai-nilai variabel. Contoh:

```

public      class
SegiEmpat

{      private  int
panjang;

private int lebar;


public      void
setPanjang(int pj){

    this.panjang=pj;

}

public      void
setLebar(int lb){

    this.lebar=lb;

}

public      int
getPanjang(){

    return
this.panjang;

}

public      int
getLebar(){

    return this.lebar;

}

public int luas(){

    return
this.panjang*this.leb
ar;

}

```

```

}

```

Publik public
setXXX() dan getXXX() adalah *method* akses dari variabel instance dari kelas EncapTest. Biasanya, *method* ini disebut sebagai *getter* dan *setter*. Oleh karena itu, setiap kelas yang ingin mengakses variabel harus mengaksesnya melalui *getter* dan *setter*. Keuntungan Encapsulation adalah:

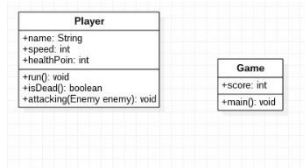
kelebihan jaringan
wireless:

- Atribut/properti dalam kelas bisa dibuat hanya-baca atau menulis saja.
- Sebuah kelas dapat memiliki total kontrol atas apa yang disimpan dalam atribut/properti.

- Pengguna kelas tidak tahu bagaimana kelas menyimpan datanya. Sebuah kelas dapat
- mengubah tipe data dari properti dan pengguna kelas tidak perlu mengubah apapun.

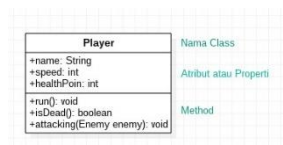
3.4 Class Relationship (Class Diagram)

Class Diagram adalah sebuah diagram yang menggambarkan hubungan antar *class*. *Class Diagram* dapat kita buat dengan aplikasi perancangan (CASE), seperti StarUML.



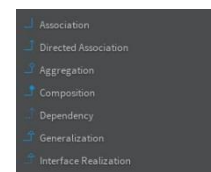
Gambar 4.1
Class
diagram

Sebuah *class* digambarkan dengan sebuah tabel 1 kolom dan 3 baris. Baris pertama berisi nama *class*; Baris kedua berisi atribut; dan Baris ketiga berisi method.



Gambar 4.2
Class diagram
(2)

Selain itu, terdapat garis yang menggambarkan hubungan antar *class*.



Gambar 4.3
Class diagram
(3)

Class Diagram biasanya digunakan oleh software engineer untuk merancang software dengan paradigma OOP.

Dalam terminologi java, kelas induk yang diturunkan disebut dengan superclass. Sedangkan kelas baru hasil turunan disebut subclass. Pada proses penurunan kelas, kelas turunan akan mewarisi sifat-sifat yang terdapat pada kelas

induknya. Selanjutnya, kelas turunan tersebut dapat memiliki sifat spesifik yang sebelumnya tidak dimiliki oleh kelas induk.

Source Code Sample :

```
class SuperClass {  
  
    public void jenis_tumbuhan() {  
  
        System.out.println("Jenis  
tumbuhan          yang  
berkembangbiak    secara  
vegetatif : “);  
  
        System.out.println("1.    Umbi  
Lapis");  
  
        System.out.println("2. Tunas");  
  
        System.out.println("3.  
Rizoma");  
  
        System.out.println("4.  
Geragih");  
  
        System.out.println("");  
  
    }  
  
    public static void main(String[]  
args) {  
  
        SuperClass    sc    =    new  
        SuperClass();  
  
        sc. jenis_tumbuhan ();
```

```
}
```

Source Code Sample :

```
class    SubClass    extends  
SuperClass {  
  
    public void Panggil() {  
  
        super. jenis_tumbuhan();  
  
        System.out.println("Beberapa  
contoh tumbuhan : “);  
  
    }  
  
    public void UmbiLapis() {  
  
        System.out.println("Umbi  
Lapis : bawang Bombay dan  
bawang putih");  
  
    }  
  
    public void Tunas() {  
  
        System.out.println("Tunas    :  
pisang dan bambu");  
  
    }  
  
    public void Rizoma() {  
  
        System.out.println("Rizoma    :  
jahe, lengkuas, kunyit, dan  
kencur");
```

```

    }

    public void Geragih() {

        System.out.println("Geragih : stroberi");

    }

    public static void
    main(String[] args) {

        SubClass scs = new
        SubClass();

        scs.Panggil();

        scs.UmbiLapis();

        scs.Tunas();

        scs.Rizoma();

        scs.Geragih();

    }

}

```

3.5 Inheritance & Polymorphism

Inheritance merupakan pewarisan atribut dan method pada sebuah class yang diperoleh dari class yang telah terdefinisi tersebut. Setiap subclass akan mewarisi state

(variabel-variabel) dan behaviour (method-method) dari superclass-nya. Subclass kemudian dapat menambahkan state dan behaviour baru yang spesifik dan dapat pula memodifikasi (override) state dan behaviour yang diturunkan oleh superclass-nya.

Subclass atau kelas turunan menyediakan state/behaviour yang spesifik yang membedakannya dengan superclass atau kelas induk, hal ini akan memungkinkan programmer Java untuk menggunakan ulang source code dari superclass yang telah ada. Programmer Java dapat mendefinisikan superclass khusus yang bersifat generik, yang disebut abstract class, untuk mendefinisikan class dengan behaviour dan state secara umum.

Pewarisan (*Inheritance*) merupakan sebuah kondisi dimana terdapat kelas induk (*super class*) dan kelas anak/ kelas turunan (*sub class*).

Melalui pewarisan, kelas induk akan menurunkan sifat-sifatnya kepada beberapa kelas anak. Sifat-sifat yang dimaksud, dapat berupa variabel, type data, dan method.

Code:

➤ Class diagram pewarisan:

```

7  import java.util.Scanner;
8
9  public class Program_inheritance
10 {
11     public static void main(String [] args)
12     {
13         kelasA superClass = new kelasA();
14         kelasB subClass = new kelasB();
15         Scanner input = new Scanner(System.in);
16
17         System.out.println(" SuperClass adalah kelas A");
18         superClass.x = 100;
19         superClass.y = 125;
20         superClass.tampilkanNilai();
21
22         System.out.println("n SubClass adalah kelas B");
23         subClass.x = 13;
24         subClass.y = 20;
25         subClass.tampilkanNilai();
26
27         System.out.print("n Masukkan nilai z : ");
28         subClass.z = input.nextInt();
29         subClass.tampilkanJumlah();
30     }
31 }

```

Gambar 5.1 Class diagram pewarisan

➤ Class kelas A:

```

13 class kelasA
14 {
15     int x;
16     int y;
17
18     void tampilkanNilai()
19     {
20         System.out.println(" Nilai x : "+x+" Nilai y : "+y);
21     }
22 }

```

Gambar 5.2 Class kelas A

➤ Class kelas B:

```

12 class kelasB extends kelasA
13 {
14     int z;
15
16     void tampilkanJumlah()
17     {
18         System.out.println("n Jumlah nilai x + y + z = " + (x+y+z));
19     }
20 }

```

Gambar 5.3 Class kelas B

➤ Output:

```

Output - examjava8 (run) X
run:
SuperClass adalah kelas A
Nilai x : 100 Nilai y : 125

SubClass adalah kelas B
Nilai x : 13 Nilai y : 20

Masukkan nilai z : 10

Jumlah nilai x + y + z = 43

```

Gambar 5.4 Output Class diagram pewarisan

Ada dua keyword utama dalam inheritance, Super dan Extends. Extends harus kita tambahkan pada definisi class yang menjadi subclass. Sedangkan Super digunakan untuk memanggil konstruktor dari superclass atau menjadi variabel yang mengacu pada superclass.

Istilah dalam inheritance yang perlu diperhatikan :

a. Extends,

Keyword ini harus kita tambahkan pada definisi class yang menjadi subclass.

b. Superclass

Superclass digunakan untuk menunjukkan hirarki class yang berarti

class dasar dari subclass/class anak.

- c. Subclass
Subclass adalah class anak atau turunan secara hirarki dari superclass.

- d. Super
Keyword ini digunakan untuk memanggil konstruktor dari superclass atau menjadi variabel yang mengacu pada superclass.

- e. Metode Overriding
Pendefinisian ulang method yang sama pada subclass.

Source Code Sample :

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

```
rumahindah C =
new rumahindah();
```

```
C.info();
}
}
```

```
class Rumah
{
    private String a =
" Rumahmewah ";
    public void
info() {
    //System.out.println ("
Dipanggil pada = "+this);
    System.out.println (" ");
```

```
System.out.println ("
Rumah = "+a);
}
}
class rumahindah
extends Rumah
{
```



```

        private String b
        = " tombol alarm ";

        public void
        info(){

        System.out.println (" ");

        super.info();namun
        dengan parameter yang
        berbeda

        System.out.println      ("
        rumahindah = "+b);

        }

        }

```

Polimorfisme atau perubahan bentuk merupakan kemampuan dari reference untuk mengubah sifat menurut object apa yang dijadikan acuan. Dengan kata lain, kita bisa menggunakan variabel dalam program untuk mengaplikasikan objek untuk memanggil method yang berbeda. Keuntungan dari polimorfisme menyediakan multiobject dari subclasses yang berbeda untuk diperlakukan

sebagai object dari superclass tunggal, secara otomatis menunjuk method yang tepat untuk menggunakannya ke particular object berdasar subclass yang termasuk di dalamnya.

Polimorfisme

(*Polymorphism*) merupakan sebuah kondisi dimana sebuah objek dapat mendefenisikan beberapa hal yang berbeda dengan cara yang sama. Dengan adanya polimorfisme, kita dapat melihat beberapa kesamaan antara sebuah kelas dengan yang lain.

Code:

➤ Class program Polimorfisme:

```

import java.*;
import java.io.*;

15
16
17 class BentokWajah
18 {
19     public String response()
20     {
21         return("Perhatikan wajah saya");
22     }
23
24 class Senyum extends BentokWajah
25 {
26     public String response()
27     {
28         return("Senyum karena senang");
29     }
30 }

```

Gambar 5.5 Class program Polimorfisme

```

32 class Tertawa extends BentukWajah
33 {
34     public String respons()
35     {
36         return("Tertawa karena gembira");
37     }
38 }
39
40 class Marah extends BentukWajah
41 {
42     public String respons()
43     {
44         return("Kemarahan Disebabkan Bertengkar");
45     }
46 }
47
48 class Sedih extends BentukWajah
49 {
50     public String respons()
51     {
52         return("Kesedihan disebabkan cemburu");
53     }
54 }

```

Gambar 5.6 Class program Polimorfisme (2)

```

54 public class ProgramPolimorfisme
55 {
56     public static void main(String[] args)
57     {
58         BentukWajah objBentuk = new BentukWajah();
59         Senyum objSenyum = new Senyum();
60         Tertawa objTertawa = new Tertawa();
61         Marah objMarah = new Marah();
62         Sedih objSedih = new Sedih();
63
64         BentukWajah[] Bentuk = new BentukWajah[5];
65         Bentuk[0] = objBentuk;
66         Bentuk[1] = objSenyum;
67         Bentuk[2] = objTertawa;
68         Bentuk[3] = objMarah;
69         Bentuk[4] = objSedih;
70
71         System.out.println("Bentuk[0] : "+ Bentuk[0].respons());
72         System.out.println("Bentuk[1] : "+ Bentuk[1].respons());
73         System.out.println("Bentuk[2] : "+ Bentuk[2].respons());
74         System.out.println("Bentuk[3] : "+ Bentuk[3].respons());
75         System.out.println("Bentuk[4] : "+ Bentuk[4].respons());
76     }
77 }

```

Gambar 5.7 Class program Polimorfisme (3)

➤ Output:

```

: Output - examjava8 (run)
run:
Bentuk[0] :Perhatikan reaksi wajah saya
Bentuk[1] :Senyum karena senang
Bentuk[2] :Tertawa karena gembira
Bentuk[3] :Kemarahan Disebabkan Bertengkar
Bentuk[4] :Kesedihan disebabkan cemburu
BUILD SUCCESSFUL (total time: 1 second)

```

Gambar 5.8 Output Class program Polimorfisme

Polimorfisme tak lengkap jika tanpa overloading. Overloading adalah suatu keadaan yakni beberapa method memiliki nama yang sama tetapi memiliki fungsionalitas yang berbeda. Ciri-ciri overloading yaitu nama method harus sama, sedangkan parameter harus berbeda. Overloading

memungkinkan polimerfisme pada kelas super dan sub kelasnya. Constructor yang terdiri dari dua atau lebih disebut overloading constructor yang termasuk ciri polimorfisme.

Source Code Sample :

```

public class
StudentInfo
{
    public static void
main(String[] args)
{
    StudentGrad
myStudent = new
StudentGrad();

    myStudent.Write(92,
1,"Candra","Dwi Putro",
2013,"Trunojoyo University of
Madura");

    myStudent.Display();
}
}

```

```

class Student
{
    public void Write(int ID, int
Grad, String Fname, String
Lname)
    {
        m_ID = ID;
        m_Graduation = Grad;
        m_First = Fname;
        m_Last = Lname;
    }

    public void Display(){
        System.out.println("Student" +
m_ID);

        System.out.println("Student :
"+m_First + " " + m_Last);

        System.out.println("Graduated
m_Graduation);
    }

    private int m_ID,
m_Graduation;

    private String m_First;

```

```

private String m_Last;
}

class StudentGrad extends
Student
{
    public void Write(int ID, int
Grad, String Fname, String
Lname, int yrGrad, String
unSch)
    {
        super.Write(ID, Grad,
Fname, Lname);

        m_UndergradSchool =
unSch;

        m_Grad = Grad;
        YearGraduated = yrGrad;
    }

    public void Display()
    {
        super.Display();

        System.out.println("Graduated:
"+ m_UndergradSchool);

```

```
System.out.println("Graduation :  
"+ YearGraduated);
```

```
}
```

```
private Integer  
YearGraduated,m_Grad;
```

```
private String  
m_UndergradSchool;
```

```
private String m_Major;
```

```
}
```

3.6 Interface & abstract classes

Interface adalah kumpulan method yang hanya memuat deklarasi dan struktur method tanpa detail implementasinya. Interface berisi sekumpulan konstanta/deklarasi method tanpa menyertakan/ menuliskan body methodnya. Method atau variabel yang terdapat pada kelas Interface dapat digunakan lebih dari satu kelas dengan cara memanggil kelas interface tersebut. Cara mendeklarasikan *interface* adalah sebagai berikut:

```
1 interface Nama_Interface  
2 {  
3     //deklarasi variabel dan/atau method  
4 }
```

Gambar 6.1 Mendeklarasikan interface

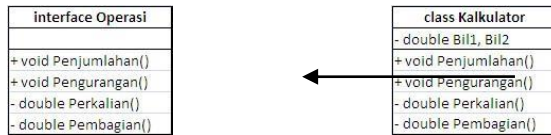
Contoh:

```
1 interface Operasi  
2 {  
3     //deklarasi variabel dan/atau method  
4     public void Penjumlahan();  
5     public void Pengurangan();  
6     public double Perkalian();  
7     public double Pembagian();  
8 }
```

Gambar 6.2 Mendeklarasikan interface (2)

Pada contoh di atas, method yang dideklarasikan pada interface Operasi tidak terdapat statement apapun, baik itu rumus atau hanya sebuah nilai balik di dalamnya. Hal ini dikarenakan interface hanyalah sebuah berisi kumpulan konstanta maupun method tanpa menyertakan/ menuliskan body methodnya. Perlu diketahui pula, bahwa an interface is not a class and classes can only implement interfaces (sebuah interface bukanlah sebuah kelas dan kelas hanya bias mengimplementasi interface). Sehingga jangan anda menganggap bahwa interface adalah super class dimana memiliki kelas trurunan. Penggunaan (implementasi)

interface dalam sebuah kelas dapat di lihat melalui skema OOP di bawah ini:



Gambar 6.3 Implementasi interface

Anak panah itu merupakan gambaran bahwa class Kalkulator merupakan implementasi dari interfaces Operasi, dimana method-method yang terdapat pada interface Operasi harus dideklarasikan ulang (overriding method) pada kelas Kalkulator. Interface dilambangkan dengan anak panah dengan garis putus-putus, sedangkan inheritance dilambangkan dengan anak panah dengan garis lurus(). Dalam pemrograman OOP, implementasi interfaces menggunakan keyword implements. Berikut adalah cara mengimplementasikan interface ke dalam pemrograman Java:

```

1 class Nama_Kelas implements Nama_Interface
2 {
3     //isi kelas
4 }
  
```

Gambar 6.4 Implementasi interface ke java

Contoh:

```

1 class Kalkulator implements Operasi
2 {
3     public void Penjumlahan()
4     {
5         //isi method Penjumlahan()
6     }
7
8     public void Pengurangan()
9     {
10        //isi method Pengurangan()
11    }
12
13    public double Perkalian()
14    {
15        //isi method Perkalian()
16        return 0;
17    }
18
19    public double Pembagian()
20    {
21        //isi method Pembagian()
22        return 0;
23    }
24 }
  
```

Gambar 6.5 Implementasi interface ke java

(2)

Abstract Class merupakan kelas yang berada pada posisi tertinggi dalam sebuah hierarki kelas. Sesuai dengan namanya, abstract class dapat didefinisikan pada class itu sendiri. Berikut adalah cara mendeklarasikan abstract class:

```

1 abstract class Nama_Kelas
2 {
3     //isi abstract class
4 }
  
```

Gambar 6.6 Abstract class

Contoh:

```

1 abstract class Manusia
2 {
3     //isi abstract class
4 }

```

Gambar 6.7 Abstract class (2)

Di dalam abstract class dapat juga diberi abstract method (optional). Penggunaan abstract method tidak diperlukan statement dalam method tersebut. Berikut adalah cara mendeklarasikan abstract method pada abstract class:

```

1 abstract class Nama_Kelas
2 {
3     //untuk sub program berjenis prosedur
4     [access_modifier] abstract void [nama_method]();
5
6     //untuk sub program berjenis function
7     [access_modifier] abstract [tipe_data] [nama_method]();
8 }

```

Gambar 6.8 abstract method pada abstract class

```

1 abstract class Manusia
2 {
3     //untuk sub program berjenis prosedur
4     public abstract void cetak();
5
6     //untuk sub program berjenis function
7     public abstract double HtgBBI();
8 }

```

Gambar 6.9 abstract class manusia

Catatan:

Apabila dalam abstract class terdapat abstract method dan kelas tersebut diturunkan ke kelas turunannya, maka method tersebut harus dideklarasikan ulang (overriding method) dengan diberi statement

pada isi methodnya. Apabila class tersebut merupakan abstract class, maka class tersebut bisa terdapat abstract method atau tidak (optional). Sedangkan apabila kelas tersebut, terdapat abstract method, maka kelas tersebut wajib berbentuk abstract class.

3.7 Static & Final Method & Variables

3.7.1 Static

Static adalah perintah khusus yang memungkinkan sebuah property atau method diakses langsung tanpa melalui object, tapi cukup menulis nama class saja. Normalnya, kita hanya bisa mengakses property dan method setelah class tersebut di instansiasi menjadi object.

```

1 class Laptop {
2     String cekInfo() {
3         return "Laptop Lenovo milik Rudi";
4     }
5 }
6
7 class BelajarJava {
8     public static void main(String args[]){
9
10         Laptop laptopRudi = new Laptop();
11         System.out.println(laptopRudi.cekInfo());
12     }
13 }
14

```

Hasil code program :

Dalam kode program ini, class **Laptop** di definisikan pada baris 1-5. Di dalamnya terdapat method **cekInfo()** yang mengembalikan string “Laptop Lenovo milik Rudi”.

Agar bisa mengakses method **cekInfo()**, kita harus buat object dari class **Laptop** terlebih dahulu. Proses instansiasi ini dilakukan pada baris 10, dimana class **laptopRudi** akan berisi object dari class **Laptop**. Selanjutnya, isi method bisa diakses dari **laptopRudi.cekInfo()**.

Jika kita coba akses method **cekInfo()** secara langsung tanpa membuat

object, hasilnya akan error:

```
1 class Laptop {
2     String cekInfo() {
3         return "Laptop Lenovo milik Rudi";
4     }
5 }
6
7 class BelajarJava {
8     public static void main(String[] args) {
9
10        System.out.println(Laptop.cekInfo());
11
12    }
13 }
```

Hasil code program :

```
BelajarJava.java:10:
error: non-static method cekInfo() cannot be accessed
      System.out.println(Laptop.cekInfo());
                        ^
1 error
```

Error terjadi karena “normalnya”, sebuah method hanya bisa diakses dari object, bukan langsung dari class.

Akan tetapi jika kita menambahkan keyword **static** sebelum nama method **cekInfo()**, method tersebut bisa diakses tanpa membuat object terlebih dahulu:

```
1  class Laptop {
2      static String cekInfo() {
3          return "Laptop Lenovo milik Rudi";
4      }
5  }
6
7  class BelajarJava {
8      public static void main(String args[]){
9
10         System.out.println(Laptop.cekInfo());
11     }
12 }
13
```

Hasil kode program:

Dengan penambahan keyword **static** di awal baris 2, method **cekInfo()** sekarang bisa diakses dengan perintah **Laptop.cekInfo()** seperti di baris 10.

Disini kita tidak perlu membuat object terlebih dahulu.

3.7.2 Final Method

Dalam membuat desain *class*, kita sering menurunkan sebuah *class* kepada *class* lain, atau yang dikenal dengan *inheritance/pewarisan*. Pemrograman objek juga membolehkan kita untuk ‘menimpa’ method milik *parent class* dengan method milik *child class*. Proses *menimpa method* atau dikenal dengan istilah *overridden method* ini dilakukan dengan cara membuat *nama method* yang sama dengan nama method yang ada di dalam *parent class*.

Bagaimana jika kita menginginkan sebuah mekanisme

untuk **melarang** class anak untuk membuat method yang akan menimpa method class induk? Atau bahkan melarang sebuah *class* untuk diturunkan sama sekali? Untuk keperluan ini, pemrograman objek PHP menggunakan keyword: **final**.

Dengan menambahkan keyword *final* kepada sebuah *method*, maka *method* tersebut tidak dapat didefinisikan ulang di dalam *child class*. Dan jika sebuah **class** ditambahkan keyword *final*, maka *class* tersebut tidak bisa diturunkan sama sekali.

Inilah *pengertian* dari *final method* dan *final class*.

Untuk membuat *final method*, kita tinggal menambahkan kata *final* sebelum *keywor*

d hak akses, seperti berikut ini:

```
2 | //... isi method
3 | }
method(){
```

Sedangkan untuk membuat *final class*, kita menambahkan kata *final* sebelum nama *class*, seperti contoh berikut ini:

```
1 | final class nama_class {
2 |     //... isi class
3 | }
```

3.7.3 Variables

Seperti yang kita ketahui, Kelas adalah prototipe atau desain dari objek apa pun yang ada. Ini mewakili satu set properti dan perilaku suatu objek.

Objek adalah keberadaan fisik dari kelas, dapat disebut sebagai turunan dari kelas.

Sebuah kelas berisi:

Variabel (yang menggambarkan sifat tertentu dari suatu objek) dan

Metode (mewakili satu set perilaku suatu objek).

Sebagai contoh, class Employee memiliki tiga variabel yang merepresentasikan property dari seorang employee dengan emp_id, emp_name dan emp_address.

Kelas juga memiliki metode yang disebut bekerja dan makan siang, yang tidak lain adalah seperangkat perilaku karyawan di kantor majikan.

```
1 # Reference variable and Instance variable example
2
3 class Employee:
4
5     def __init__(self): # Constructor
6         self.name='Pranay'
7         self.company='Amazon'
8         self.address='UK'
9
10 e=Employee()
11
12 print("Name of the Employee: ",e.name)
13 print("Name of the Company: ",e.company)
14 print("Address of the Employee: ",e.address)
15
16 ##### OUTPUT #####
17 # Name of the Employee: Pranay
18 # Name of the Company: Amazon
19 # Address of the Employee: UK
```

ref_inst.py hosted with ❤ by GitHub

[view raw](#)

Variabel dalam kelas terkait dengan objek, kelas, atau metode.

Ada tiga jenis variabel dalam OOP dengan python.

- a. Variabel Instance (Variabel tingkat objek)
- b. Variabel statis (Variabel tingkat kelas)
- c. Variabel lokal (Variabel tingkat metode)

Tiga jenis Metode dalam OOP dengan python.

- a. Metode instan
- b. Metode statis
- c. Metode kelas

Variabel referensi:

Variabel ini merupakan referensi atau penunjuk ke suatu objek untuk melakukan operasi pada objek tersebut. Variabel referensi selalu menunjuk ke suatu objek.

3.8 Exception Handling

Exception handling merupakan teknik yang diperlukan dalam mengatasi error saat program berjalan. Pada umumnya sebuah program akan berhenti secara tiba-tiba/hang apabila terdapat kesalahan/error, untuk menghindari hal ini maka diperlukanlah exception handling. Exception handling akan dapat membantu program tetap berjalan dan menampilkan hasil, walaupun ada kesalahan yang terjadi secara tiba-tiba.

Exception handling memiliki beberapa jenis i, yaitu : try-catch, try-catch-finally, throw, dan throws. Secara rinci dapat dilihat pada penjelasan berikut ini :

a. try :
menentukan bagian program yang akan terjadi pengecualian, try harus diikuti dengan catch atau finally

b. catch :
menangani kesalahan, catch harus disertai dengan try dan finally

c. finally :
mengeksekusi code yang dianggap penting, finally dapat berjalan dengan baik walaupun tidak ada exception yang terjadi

d. throw :
melempar pengecualian yang terjadi, throw digunakan di dalam body code

e. throws :

mendeklarasikan pengecualian yang terjadi pada fungsi tertentu Interface adalah kumpulan method yang hanya memuat

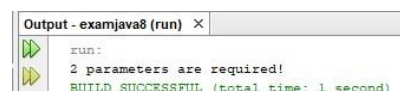
1. Exception handling dengan try – catch
Code:

```
12 class NestedTryDemo2
13 {
14     static void nestedTry(String args[])
15     {
16         try
17         {
18             int a = Integer.parseInt(args[0]);
19             int b = Integer.parseInt(args[1]);
20             System.out.println(a/b);
21         }
22         catch (ArithmeticException e)
23         {
24             System.out.println("Divide by zero error!");
25         }
26     }
27 }
```

Gambar 7.1 Exception handling try–catch

```
29 public static void main(String args[])
30 {
31     try
32     {
33         nestedTry(args);
34     }
35     catch (ArrayIndexOutOfBoundsException e)
36     {
37         System.out.println("2 parameters are required!");
38     }
39 }
```

Gambar 7.2 Exception handling try–catch(2)



```
Output - examjava8 (run) X
run:
2 parameters are required!
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 7.3 Exception handling try–catch output

2. Exeption handling dengan try-catch-finally
Code:

```

12 class FinallyDemo
13 {
14     static void myMethod(int n) throws Exception
15     {
16         try
17         {
18             myMethod(n);
19         }
20         catch (Exception e)
21         {
22             System.out.println("case pertama");
23             return;
24         }
25         catch (Exception e)
26         {
27             System.out.println("case kedua");
28             throw new RuntimeException("demo case ketiga");
29         }
30         catch (Exception e)
31         {
32             System.out.println("case keempat");
33             throw new RuntimeException("demo case keempat");
34         }
35         catch (Exception e)
36         {
37             System.out.println("case kelima");
38         }
39     }
40 }

```

Gambar 7.4 Exception handling try–catch finally

```

12 class ThrowDemo
13 {
14     public static void main(String args[])
15     {
16         try
17         {
18             throw new RuntimeException("demo demo");
19         }
20         catch (Exception e)
21         {
22             System.out.println("demo demo");
23         }
24         finally
25         {
26             System.out.println("demo demo");
27         }
28     }
29 }

```

Gambar 7.7 Exception handling dengan throw

```

35     catch (RuntimeException e)
36     {
37         System.out.println("RuntimeException terjadi: ");
38         System.out.println(e.getMessage());
39     }
40 }
41 finally
42 {
43     System.out.println("try-block entered.");
44 }
45 }

```

Gambar 7.5 Exception handling try–catch finally(2)

Output:

```

Output - examjava8 (run) X
run:
Exception caught here.
java.lang.RuntimeException: throw demo
BUILD SUCCESSFUL (total time: 2 seconds)

```

Gambar 7.8 Exception handling throw output

```

47 public static void main(String args[])
48 {
49     for (int i=1; i<=4; i++)
50     {
51         try
52         {
53             myMethod(i);
54         }
55         catch (Exception e)
56         {
57             System.out.println("Exception terjadi: ");
58             System.out.println(e.getMessage());
59         }
60         System.out.println(i);
61     }
62 }
63 }
64 }

```

Gambar 7.6 Exception handling try–catch finally(3)

4. Exception handling dengan throws
Code class balok dengan exception:

Output:

```

Output - examjava8 (run) X
run:
case pertama
try-block entered.
case kedua
try-block entered.
case ketiga
RuntimeException terjadi: demo case ketiga
try-block entered.
case keempat
try-block entered.
Exception terjadi: demo case keempat
BUILD SUCCESSFUL (total time: 1 second)

```

Gambar 7.6 Exception handling output

```

12 public class BalokDenganException
13 {
14     private double panjang, lebar, tinggi;
15     private static int jumlahBalok = 0;
16     public BalokDenganException(double panjang, double lebar, double tinggi)
17     {
18         setPanjang(panjang);
19         setLebar(lebar);
20         setTinggi(tinggi);
21         jumlahBalok++;
22     }
23     public static int getJumlahBalok()
24     {
25         return jumlahBalok;
26     }
27     //get dan set method untuk lebar balok
28     public double getLebar()
29     {
30         return lebar;
31     }
32     //get dan set method untuk panjang balok
33     public double getPanjang()
34     {
35         return panjang;
36     }
37 }

```

Gambar 7.9 class balok dengan exception

3. Exception handling dengan throw
Code:

```

37 public void setLebar(double lebar) throws IllegalArgumentOutOfRangeException
38 {
39     if (lebar <= 0)
40         throw new IllegalArgumentOutOfRangeException("lebar harus lebih dari 0");
41     else
42         this.lebar = lebar;
43 }
44 //get dan set method untuk lebar balok
45 public double getLebar()
46 {
47     return lebar;
48 }
49 public void setPanjang(double panjang) throws IllegalArgumentOutOfRangeException
50 {
51     if (panjang <= 0)
52         throw new IllegalArgumentOutOfRangeException("panjang harus lebih dari 0");
53     else
54         this.panjang = panjang;
55 }
56 //get dan set method untuk tinggi balok
57 public double getTinggi()
58 {
59     return tinggi;
60 }

```

Gambar 7.10 class balok dengan exception(2)

```

60 //set dan get method untuk lebar balok
61 public double getLebar() {
62     return lebar;
63 }
64
65 public void setLebar(double lebar) {
66     if(lebar <= 0) {
67         this.lebar = lebar;
68     }
69     else {
70         throw new IllegalArgumentException("Nilai panjang dari persegi panjang tidak boleh negatif");
71     }
72 }
73 }
74

```

Gambar 7.11 class balok dengan exception(3)

Code class test balok
dengan exception:

```

18 public class TestBalokException {
19     public static void main(String[] args) {
20         try {
21             BalokException balok1 = new BalokException(7, 2.5, 4.7);
22             BalokException balok2 = new BalokException(3, 2.7, 4.7);
23             BalokException balok3 = new BalokException(3, 2.7, 4.7);
24             BalokException balok4 = new BalokException(3, 2.7, 4.7);
25             BalokException balok5 = new BalokException(3, 2.7, 4.7);
26         } catch (Exception e) {
27             System.out.println("Error: " + e.getMessage());
28         }
29     }
30 }

```

Gambar 7.12 class test balok dengan exception

Output:

```

Output - example4 (run) X
run
java.lang.IllegalArgumentException: Nilai panjang dari persegi panjang tidak boleh negatif
Jumlah objek yang dibuat: 4
BUILD SUCCESSFUL (total time: 1 second)

```

Gambar 7.12 Output test balok dengan exception

3.3 Access Modify

Pada Java jika diinginkan suatu data untuk bisa di akses oleh object lain atau tidak bisa di akses oleh object luar lainnya digunakan Access

Modify. Access Modify lebih singkatnya, larangan suatu data bisa diakses ataupun tidak bisa diakses. Contohnya saja jika suatu data hanya dapat diubah isinya oleh method tertentu saja, dan method lainnya tidak bisa karena adanya pembatasan akses pada data tersebut. Java mempunyai 4 macam Access Modify, yaitu default, public, private, protected yang fungsinya berlainan.

1) *Default*

Tipe akses ini hanya dalam satu class itu sendiri yang dapat mengaksesnya tergantung dari blok kodenya.

2) *Public*

Tipe ini mengijinkan seluruh class dari luar atau dari dalam package bisa mengaksesnya.

Source Code Sample :

```
public class Hello

{

    public static void main(String args[])

    {

        // access modifier default

        int nilai;

    }

}
```

3) Protected

Tipe ini di gunakan untuk pewarisan dalam java, jadi klo suatu super class mempunyai sebuah member, ini dapat diakses oleh subclassnya.

SourceCodeSample:


```

public class SuperClass{

    public static void main(Strin
g args[]){

        // access modifier protected

        protected int nilai;

    }
}

```

```

}

public class SubClass{

    public static void main(Strin
g args[]){

        // merubah nilai yang ada di
superclass

        public void hitung(){

```

```

        nilai=10;

```

```

        }

    }

}

```

4) *Private*

Tipe ini hanya dapat diakses dimana class tersebut di buat.

SourceCodeSample:

```

public class Hello{

    public static void main(Strin
g args[]){

        // access modifier private

        private int nilai;

```

```

    }

    }
}

```

3.4 Constructor and Destructor

1) Constructor

Dalam konsep OOP, constructor adalah suatu method yang digunakan untuk membuat suatu object dari suatu class. Dalam Java, constructor dideklarasikan sama dengan nama class yang bersangkutan. Constructor diakses dengan menggunakan keyword new.

Source Code Sample :

```

public class Bike

{
    public Bike() {
        what_to_do_when_creating_an_object
    }

    public static void
    main(String args[])

    {
        Bike obj = new Bike();
    }
}

```

2) Destructor

Dalam konsep OOP, destructor adalah suatu method yang digunakan untuk melepaskan semua resource yang dialokasikan object semasa hidupnya.

Dalam Java, destructor tidak mempunyai peranan yang berarti karena Java memiliki fasilitas Garbage Collector, dimana alokasi memori akan secara otomatis dibebaskan apabila sudah tidak digunakan lagi.

Source Code Sample :

```

class Thing {

    public static int number_of_
    _things = 0;

    public String what;

    public Thing (String what) {

        this.what = what;

        number_of_things++;
    }
}

```

```

    }

    protected void finalize ()
    {

        number_of_things--;

    }

}

public class TestDestructor
{

    public static void main(String[] args)

    {

        Thing obj =
        new Thing("Test App");

    }

}

```

3.5 Static Properties

Seluruh property dan method hanya bisa diakses dari objek, maka static property dan static method adalah pengecualiannya. Static property dan static method adalah property (variabel) dan method (function) yang melekat kepada class, bukan kepada objek. Konsep static property memang

‘agak keluar’ dari konsep objek sebagai tempat melakukan proses, karena sebenarnya class hanya merupakan ‘blueprint’ saja. Untuk membuat static property dan static method, kita menambahkan keyword ‘static’ setelah penulisan akses level property atau method, seperti contoh berikut:

Source Code Sample :

```

// static property

    public static $harga_beli;

// static method

public static function beli_laptop
()

{

    //...isi method

}

```

Dalam contoh diatas, saya menggunakan hak akses public, tetapi kita juga bisa menggunakan hak akses lain

seperti `private` dan `protected` untuk `static property` dan `static method`. Karena `static property` dan `static method` adalah milik class, maka kita tidak perlu membuat objek untuk mengaksesnya, tapi langsung menyebutkan nama class dan menggunakan operator `::`, berikut adalah contoh pengaksesan `static property` dan `static method` dari class `laptop`:

Source Code Sample :

```
echo laptop::$harga_beli;
```

```
echo laptop::beli_laptop();
```

3.9 Objek Persistence

Objek persistence adalah istilah yang sering Anda dengar digunakan bersama dengan masalah penyimpanan objek dalam database. Kegigihan diharapkan beroperasi dengan integritas transaksional, dan karena itu tunduk pada kondisi yang ketat. (Lihat bagian Sumberdaya artikel ini untuk informasi lebih lanjut tentang pemrosesan transaksi.) Sebaliknya, layanan bahasa yang ditawarkan melalui perpustakaan dan paket bahasa standar seringkali bebas dari kendala transaksional.

Seperti yang akan kita lihat di artikel ini, bukti menunjukkan bahwa persistensi Java yang sederhana kemungkinan akan berasal dari bahasa itu sendiri, sementara fungsionalitas

database yang canggih akan ditawarkan oleh vendor database.

Tidak ada objek yang merupakan pulau

Di dunia nyata, Anda jarang menemukan objek yang tidak memiliki hubungan dengan objek lain. Objek adalah komponen dari model objek. Masalah daya tahan objek melampaui masalah daya tahan dan distribusi model objek begitu kita melakukan pengamatan bahwa objek saling berhubungan berdasarkan hubungannya satu sama lain.

3.10 Multithreading

multi-threaded yang artinya kita dapat mengembangkan program multi-threaded menggunakan [Java](#). Program multi-utas berisi dua atau lebih bagian yang dapat

berjalan secara bersamaan dan setiap bagian dapat menangani tugas yang berbeda pada saat yang sama memanfaatkan sumber daya yang tersedia secara optimal khususnya ketika komputer Anda memiliki banyak [CPU](#).

Menurut definisi, multitasking adalah ketika beberapa proses berbagi sumber daya pemrosesan yang umum seperti CPU. Multi-threading memperluas gagasan multitasking menjadi aplikasi di mana Anda dapat membagi operasi tertentu dalam satu aplikasi menjadi utas individu. Setiap utas dapat berjalan secara paralel. OS membagi waktu pemrosesan tidak hanya di antara aplikasi yang berbeda, tetapi juga di antara setiap utas dalam aplikasi. Multi-threading memungkinkan Anda menulis dengan cara di mana banyak aktivitas dapat dilanjutkan secara bersamaan dalam program yang sama.

Multithreading di Java adalah proses menjalankan

beberapa utas secara bersamaan. Thread adalah sub-proses ringan, unit pemrosesan terkecil. Multiprocessing dan multithreading, keduanya digunakan untuk mencapai multitasking. Namun, kami menggunakan multithreading daripada multiprocessing karena utas menggunakan area memori bersama. Mereka tidak mengalokasikan area memori terpisah sehingga menghemat memori, dan peralihan konteks antar utas membutuhkan waktu lebih sedikit daripada proses. Java Multithreading banyak digunakan dalam permainan, animasi.

Keuntungan MultiThreading pada [Java](#)

1. Multithread tidak memblokir pengguna karena threadnya independen dan Anda dapat melakukan beberapa operasi pada saat yang bersamaan.
2. Anda dapat melakukan banyak operasi bersama-sama,

sehingga dapat menghemat waktu.

3. Thread bersifat independen, jadi tidak memengaruhi thread lain jika pengecualian terjadi di thread tunggal.

MultiTasking pada java

Multitasking adalah proses menjalankan banyak tugas secara bersamaan. Kami menggunakan multitasking untuk memanfaatkan CPU. Multitasking dapat dicapai dengan dua cara:

1. Multitasking Berbasis Proses (Multiprocessing).
2. Multitasking Berbasis Thread (Multithreading).

Multitasking Berbasis Proses (Multiprocessing)

- Setiap proses memiliki alamat di memori. Dengan kata lain, setiap proses mengalokasikan area memori terpisah.
- Suatu proses sangatlah berat.
- Biaya komunikasi antar proses tinggi.

- Berpindah dari satu proses ke proses lainnya memerlukan waktu untuk menyimpan dan memuat register, peta memori, memperbarui daftar.

Multitasking Berbasis Thread (Multithreading)

- Thread berbagi ruang alamat yang sama.
- Thread itu sangat ringan.
- Biaya komunikasi antar thread rendah.

3.11 Using java library (java API)

Pustaka Klien Google API

untuk Java menyediakan fungsionalitas yang umum untuk semua Google API, misalnya transport HTTP, penanganan kesalahan, autentikasi, penguraian JSON, unduhan/unggahan media, dan pengelompokan. Pustaka menyertakan pustaka OAuth 2.0 yang andal dengan antarmuka yang konsisten; model data

XML dan JSON yang ringan dan efisien yang mendukung skema data apa pun; dan dukungan untuk buffer protokol.

Untuk memanggil Google API menggunakan pustaka klien Google untuk Java, Anda memerlukan pustaka Java yang dihasilkan untuk Google API yang Anda akses. Pustaka yang dihasilkan ini menyertakan pustaka inti google-api-java-client bersama dengan informasi khusus API seperti URL root. Mereka juga menyertakan kelas yang mewakili entitas dalam konteks API, dan yang berguna untuk membuat konversi antara objek JSON dan objek Java.

Beta features

Fitur yang ditandai dengan @Beta di tingkat kelas atau metode dapat berubah. Mereka mungkin dimodifikasi atau dihapus dalam rilis besar apa pun. Jangan gunakan fitur beta jika kode Anda adalah pustaka itu sendiri (yaitu, jika kode Anda digunakan pada CLASSPATH pengguna di luar kendali Anda).

Deprecated features

Fitur non-beta yang tidak digunakan lagi akan dihapus delapan belas bulan setelah rilis saat fitur tersebut pertama kali tidak digunakan lagi. Anda harus memperbaiki penggunaan Anda sebelum waktu ini. Jika tidak, semua jenis kerusakan dapat terjadi, dan Anda tidak dijamin akan mengalami kesalahan kompilasi.

Sorotan Pustaka Klien Google API untuk Java

Sangat mudah untuk memanggil Google API

Anda dapat memanggil Google API menggunakan pustaka yang dihasilkan khusus layanan Google dengan Pustaka Klien Google API untuk Java. (Untuk menemukan pustaka klien yang

dihasilkan untuk Google API, kunjungi daftar Google API yang didukung.) Berikut adalah contoh yang menggunakan Pustaka Klien API Kalender untuk Java untuk melakukan panggilan ke Google Kalender API:

```
// Show events on user's calendar.  
View.header("Show Calendars");  
CalendarList feed = client.calendarList().list().execute();  
View.display(feed);
```

Installation is easy

Jika Anda tidak menggunakan pustaka yang dihasilkan, Anda dapat mengunduh biner untuk Pustaka Klien Google API untuk Java langsung dari halaman unduhan, atau Anda dapat menggunakan Maven atau Gradle. Untuk menggunakan Maven, tambahkan baris berikut ke file pom.xml Anda:

```

<project>
<dependencies>
<dependency>
<groupId>com.google.api-client</groupId>
<artifactId>google-api-client</artifactId>
<version>1.32.1</version>
</dependency>
</dependencies>
</project>

```

Untuk menggunakan Gradle, tambahkan baris berikut ke file build.gradle Anda:

```

repositories {
    mavenCentral()
}
dependencies {
    compile 'com.google.api-client:google-api-client'
}

```

3.12 Collections (koleksi)

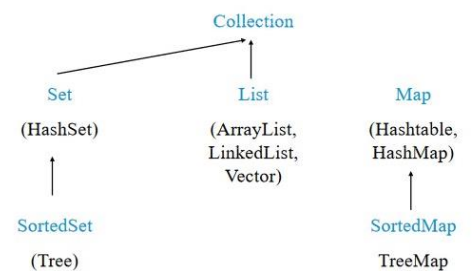
Collection merupakan suatu objek yang dapat digunakan untuk menyimpan sekumpulan obyek. Objek – objek yang terdapat di dalam Collection disebut dengan elemen. Collection ini meyimpan elemen yang bertipe

object, sehingga berbagai tipe objek dapat disimpan di dalam Collection.

API Koleksi Java

Java Collections ini terdiri dari interface Collection, list, set dan order. Di bawah ini merupakan koleksi antarmuka dan kelas hierarki. collection(Koleksi)

Koleksi merupakan tetap objek yang tidak memiliki posisi yang (tidak ada urutan tertentu) dan menerima duplikat.



List (Daftar)

Daftar merupakan pengelompokkan berdasarkan urutan, ia memiliki posisi awal dan juga posisi akhir. List juga tidak harus memiliki isi yang unit dan ia juga dapat menampung berbagai macam tipe data seperti Sting, Long, Integer bahkan Object.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Cobalist {
    public static void main(String[] x) {
        List lis = new ArrayList();
        // Masukkan value ke dalam lis
        lis.add("NaufalFikriAulia");
        lis.add("RizalFadly");
        lis.add("BudiArman");
        lis.add("MartaKarlina");
        lis.add("FitriHerawati");
        // Tampilkan value lis
        for (Iterator iterator = lis.iterator(); iterator.hasNext();) {
            String string = (String) iterator.next();
            System.out.println(string);
        }
    }
}
```

Berikut hasilnya

```
NaufalFikriAulia
RizalFadly
BudiArman
MartaKarlina
FitriHerawati
```

Set

Set merupakan sekumpulan objek yang tidak didasarkan dengan urutan (unordered) dan menolak duplikat. Pada set, setiap anggotanya harus unik. Sedangkan untuk urutan dan tataletak dari anggotanya tidak begitu penting. Set juga dapat menampung berbagai tipe data bahkan objek.

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class CobaSet {
    public static void main(String[] namakami) {
        Set setdah = new HashSet();
        // Masukkan value ke dalam setdah
        setdah.add("NaufalFikriAulia");
        setdah.add("RizalFadly");
        setdah.add("BudiArman");
        setdah.add("MartaKarlina");
        setdah.add("FitriHerawati");
        setdah.add("AnnisaCandela");
        // Tampilkan value setdah
        for (Iterator iterator = setdah.iterator(); iterator.hasNext(); ) {
            String string = (String) iterator.next();
            System.out.println(string);
        }
    }
}
```

```
RizalFadly
FitriHerawati
AnnisaCandela
NaufalFikriAulia
BudiArman
MartaKarlina
```

kamu tetap menyimpan nilai dengan key yang sama, maka nilai key yang terakhir disimpanlah yang akan tersimpan didalam Map.

Berikut hasil dari eksekusinya

Map

Sedangkan Map merupakan mendukung pencarian berdasarkan key dengan syarat harus unik. Map juga dapat menampung beragam tipe data, sama dengan List dan Set. Namun bedanya, Map dapat menyimpan data secara berpasangan yang terdiri dari key dan value. Untuk nilai dari key, harus unik dan tidak boleh ada yang sama. Namun jika

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class CobaMap {
    public static void main(String[] buah manis) {
        Map<Integer, String> mapInt_String = new HashMap<Integer, String>(); // map<Integer, String>
        // Masukkan key dan value ke dalam mapInt_String
        mapInt_String.put(1, "APEL");
        mapInt_String.put(2, "NANAS");
        mapInt_String.put(3, "JERUK");
        mapInt_String.put(4, "RAMBUTAN");
        mapInt_String.put(5, "DURIAN");
        // Tampilkan value mapInt_String
        for (Iterator iterator = mapInt_String.values().iterator(); iterator.hasNext(); ) {
            String string = (String) iterator.next();
            System.out.println(string);
        }

        Map<String, String> mapString_String = new HashMap<String, String>(); // map<String, String>
    }
}
```

```

// Masukkan key dan value ke dalam mapString_String
mapString_String.put("BUDI", "SEPEDA");
mapString_String.put("ANDRI", "MOBIL");
mapString_String.put("DENI", "TRUK");
mapString_String.put("JAJANG", "MOTOR");
mapString_String.put("ATUT", "BUS");
mapString_String.put("RUDI", "KAPAL JET");// KEY yang digunakan = RUDI
mapString_String.put("RUDI", "UFO");// Key yang digunakan = RUDI juga, mak
a value "KAPAL JET" akan ditimpa oleh "UFO"

// Tampilkan value mapString_String
System.out.println("=====");
for (Iterator iterator = mapString_String.values().iterator(); iterator
.hasNext();) {
    String string = (String) iterator.next();
    System.out.println(string);
}

// Tampilkan pasangan key-value mapString_String
System.out.println("=====");
for (Iterator iterator = mapString_String.keySet().iterator(); iterator
.hasNext();) {
    String string = iterator.next();

```

```
        System.out.println(string + " mengendarai "
            + mapString_String.get(string));
    }
}
```

Berikut ini hasil dari eksekusinya

```
GAJAH
KUDA
ZEBRA
KAMBING
SAPI
=====
MOBIL
SEPEDA
BUS
TRUK
MOTOR
UFO
=====
ANDRI mengendarai MOBIL
BUDI mengendarai SEPEDA
ATUT mengendarai BUS
DENI mengendarai TRUK
JAJANG mengendarai MOTOR
RUDI mengendarai UFO
```

Coba dilihat pada hasil eksekusi dari MAP, terlihat bahwa nilai dari “KAPAL JET” tidak pernah ditampilkan pada output. Ini dikarenakan nilai “KAPAL JET” ditimpa oleh nilai dari “UFO”.

3.13 maraking connection with database

Membuat koneksi database dengan oop, Langkah pertama yang harus kita lakukan adalah kita buat dulu database-nya. Membuat database dengan nama **malasngoding**. Supaya lebih cepat, bisa mengimport sql berikut.

```
1  -- phpMyAdmin SQL Dump
2  -- version 3.5.2.2
3  -- http://www.phpmyadmin.net
4  --
5  -- Host: 127.0.0.1
6  -- Generation Time: Mar 07, 2016 at 07:02 AM
7  -- Server version: 5.5.27
8  -- PHP Version: 5.4.7
9
10 SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11 SET time_zone = "+00:00";
12
13
14 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
15 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
16 /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
17 /*!40101 SET NAMES utf8 */;
18
19 --
20 -- Database: 'malasngoding'
21 --
22
23 --
24 --
25 -- Table structure for table 'user'
26 --
27 --
28
29 CREATE TABLE IF NOT EXISTS `user` (
30   `id` int(11) NOT NULL AUTO_INCREMENT,
31   `nama` varchar(100) NOT NULL,
32   `alamat` varchar(100) NOT NULL,
33   `usia` int(11) NOT NULL,
34   PRIMARY KEY (`id`)
35 ) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;
36
37 --
38 -- Dumping data for table 'user'
39 --
40
41 INSERT INTO `user` (`id`, `nama`, `alamat`, `usia`) VALUES
42 (1, 'Andi', 'Jakarta', 20),
43 (2, 'Budi', 'Bandung', 30);
44
45 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
46 /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
47 /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Langkah kedua adalah membuat

class database. Jadi buat sebuah

file baru dengan nama

database.php

```
1 <?php
2 //www.malasngoding.com
3 class database{
4     var $host = "localhost";
5     var $uname = "root";
6     var $pass = "";
7     var $db = "malasngoding";
8
9     function __construct(){
10         $koneksi = mysql_connect($this->host, $this->uname, $this->pass);
11         mysql_select_db($this->db);
12
13         if($koneksi){
14             echo "Koneksi database mysql dan php berhasil.";
15         }else{
16             echo "Koneksi database mysql dan php GAGAL !";
17         }
18     }
19 }
20
21 $malasngoding = new database();
22
23
24 ?>
```

Lalu menghubungkannya di function construct. Karena function construct adalah method pertama yang dijalankan jika sebuah class di akses.

Jika koneksi berhasil maka akan ditampilkan koneksi database mysql dan php berhasil. Dan jika tidak maka akan di tampilkan koneksi database mysql dan php gagal!.

Disini sebagai contoh

menghubungkan mysql dan php

dengan Teknik oop, membuat

class dengan nama database.

Kemudian mengisi data host dan

database pada propdrti-nya.

```
1 var $host = "localhost";
2 var $uname = "root";
3 var $pass = "";
4 var $db = "malasngoding";
```

```
1 if($koneksi){
2     echo "Koneksi database mysql dan php berhasil.";
3 }else{
4     echo "Koneksi database mysql dan php GAGAL !";
5 }
```

Coba jalankan pada browser dan hasilnya koneksi database mysql dan php berhasil. Tandanya koneksi database dengan Teknik oop telah berhasil.



3.14 GUI DAN SWING

Dalam Ui berbasis teks, perintah dimasukkan dari keyboard. Dalam konsol sistem

biasanya mengontrol Tindakan pengguna :

Memasukkan jumlah kelas : 3

Memasukkan jumlah siswa ; 15

Memiliki 45 siswa

Sebagian besar program menggunakan gui diucapkan gooney

Graphical, bukan hanya teks atau karakter tetapi jendela, menu, tombol.

Kamu ser : orang yang menggunakan program

Antarmuka : cara berinteraksi dengan program elemen grafis

Jendela: bagian layer yang berfungsi sebagai layer yang lebih kecil dari dalam layer

Menu: daftar alternatif yang ditawarkan kepada pengguna

Tombol : seperti tombol yang bisa ditekan

Bidang teks : pengguna dapat menulis sesuatu.

swing

Swing adalah API (Application Programming Interface) untuk membuat GUI (Graphical User Interface) untuk aplikasi yang dibuat dengan Java.

Komponen swing

1. JComponent : class induk untuk semua komponen Swing
2. JFrame : Class yang dapat membuat frame.
3. JPanel : Class yang dapat digunakan untuk menampung komponen lain.
4. JLabel : Class yang digunakan untuk menampilkan label.
5. JButton : class untuk membuat sebuah tombol
6. JCheckBox : Class untuk membuat pilihan ya/tidak

7. JTextField : Class untuk mengisi data text

Membuat aplikasi GUI dengan swing

Membuat objek dengan class JFrame. Isi judul frame sebagai parameternya

Atur setVisible dengan nilai true.

```
import javax.swing.*;

public class GUI1 {

    public static void main(String[] args) {

        JFrame f=new JFrame("Demo Swing");

        f.setVisible(true);

    }

}
```

Hasil ran adalah sebuah window kecil

Mengubah ukuran window



Method `setSize` memiliki parameter lebar (width) dan tinggi (height). Lebar dan tinggi dalam satuan pixel.

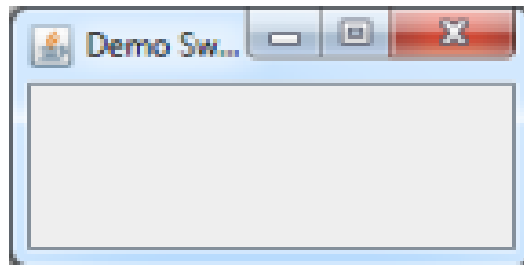
```
import javax.swing.*;
public class GUI1 {
    public static void main(String[] args) {
        JFrame f=new JFrame("Demo Swing");
        f.setSize(200, 100);
        f.setVisible(true);
    }
}
```

berbasis Swing, maka windows akan tertutup tetapi aplikasinya belum benar-benar tertutup.

Untuk benar-benar menutup maka anda harus mengatur

```
import javax.swing.*;
public class GUI1 {
    public static void main(String[] args) {
        JFrame f=new JFrame("Demo Swing");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200, 100);
        f.setVisible(true);
    }
}
```

setDefaultCloseOperation dengan konstanta JFrame.EXIT_ON_CLOSE



Menutup aplikasi

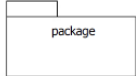
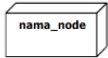
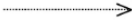

Jika anda mengklik tombol Close windows pada aplikasi

3.15 Deployment

Deployment Diagram adalah salah satu model diagram dalam UML untuk mengarahkan artefak dalam node. Deployment diagram digunakan untuk memvisualisasikan hubungan antara software dan hardware. Secara spesifik deployment diagram dapat membuat physical model tentang bagaimana komponen perangkat lunak (artefak) digunakan pada komponen perangkat keras, yang dikenal sebagai node. Ini adalah salah satu diagram paling penting dalam tingkat implementasi perangkat lunak dan ditulis sebelum melakukan coding.

Notasi – notasi pada

Deployment Diagram:

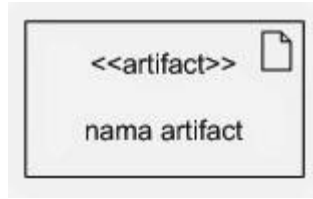
Simbol	Deskripsi
Package 	package merupakan sebuah bungkus dari satu atau lebih node
Node 	biasanya mengacu pada perangkat keras (<i>hardware</i>), perangkat lunak yang tidak dibuat sendiri (<i>software</i>), jika di dalam node disertakan komponen untuk mengkonsistenkan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen
Kebergantungan / <i>dependency</i> 	Kebergantungan antar <i>node</i> , arah panah mengarah pada <i>node</i> yang dipakai
Link 	relasi antar <i>node</i>

Deployment diagram sederhana yang disederhanakan untuk aplikasi web akan mencakup:

Nodes (application server and database server)

Contoh: Komputer/PDA, laptop, handphone, serta peralatan komunikasi data (router, hub, switch, modem)

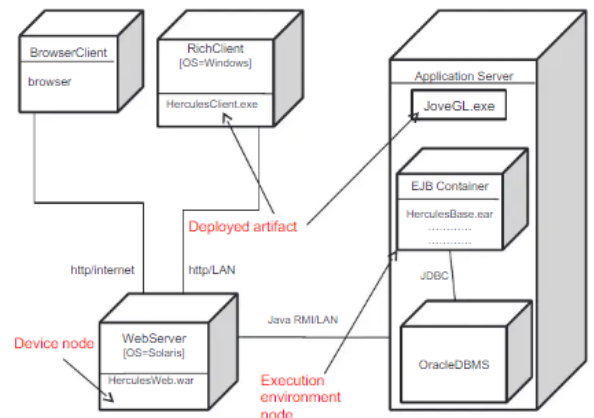
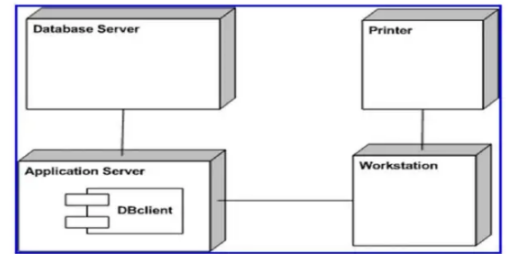
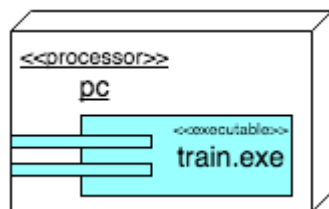
Artifacts (application client and database schema)



Artefak merupakan spesifikasi dari bentuk informasi fisik yang digunakan atau dihasilkan. Selain itu, artefak dapat dihubungkan dengan komponen pada component diagram dan hanya digambarkan dalam sebuah node


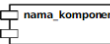
Contoh: source file, script, executable file, table di database, document word/excel, e-mail, dll.



Contoh dari Deployment Diagram:



Sedangkan Component Diagram adalah diagram UML yang menampilkan komponen dalam system dan hubungan antara mereka. Saat berurusan dengan dokumentasi sistem yang kompleks, component diagram dapat membantu memecah sistem menjadi komponen yang lebih kecil.

Jadi tujuan dari komponen diagram adalah:

Simbol	Diskripsi
Package 	package merupakan sebuah bungkusan da lebih komponen
Komponen 	Komponen sistem
Kebergantungan / dependency	Kebergantungan antar komponen, arah mengarah pada komponen yang dipaki

Simbol	Diskripsi
Antarmuka / interface  nama_interface	sama dengan konsep <i>interface</i> pa pemrograman berorientasi objek, sebagai antarmuka komponen aga mengakses langsung komponen
Link 	

Memvisualisasikan komponen dari suatu sistem

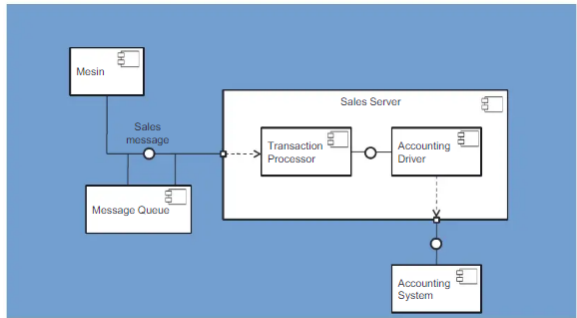
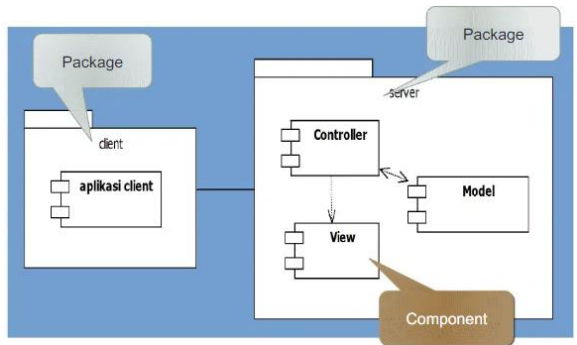
Membangun file-file yang dapat dieksekusi dengan menggunakan teknik forward dan reverse engineering

Menjelaskan organisasi dan hubungan dari komponen

Notasi – notasi pada Component Diagram:
Diagram:

Contoh dari Component

Diagram:



Kesimpulannya, component diagram menunjukkan bagaimana berbagai elemen sistem telah dikelompokkan bersama (menjadi rakitan) dan hubungan antara komponen-komponennya.

Deployment diagram lebih detail dimana diagram tersebut menjelaskan elemen perangkat keras mana yang berada.

Jadi misalnya, jika “Utility.dll” adalah komponen dan digunakan pada Client Machine (hardware). Kemudian, component diagram sistem akan menunjukkan utilitas dan linknya dengan komponen lain dalam sistem (contohnya Customer/SQL Packages). Sedangkan, deployment diagram akan menampilkan konfigurasi perangkat keras (hardware) – server basis data/ server web/ client machine dan komponen utilitasnya akan ditempatkan kedalam client machine node.

3. PENUTUP

3.1 Kesimpulan

OOP atau Object Oriented Programming adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Beberapa konsep OOP dasar, antara lain Encapsulation (Class and Object), Inheritance (Penurunan sifat), Polymorphisme, Access Modify, Constructor, Destructor, Static Properties, super class serta sub class.

3.2 Saran

Setelah mengetahui konsep penerapan Object Oriented Programming (OOP), begitu banyak kesulitan dalam mengerjakan sebuah program yang belum terpahami dengan baik.

DAFTAR PUSTAKA

Mangallo, D. (2017, Agustus 13). *MAKALAH KONSEP DASAR OBJECT-ORIENTED PROGRAMMING (OOP)*. Retrieved Juni 14, 2022, from MATERI KULIAH INFORMATIKA: <http://deslyanto-mangallo.blogspot.com/2017/08/makalah-konsep-dasar-object-oriented.html>

Perdana, A. (2021, Januari 18). *Mengenal OOP, Teknik Pemrograman Modern yang Berorientasi pada Objek*. Retrieved Juni 6, 2022, from glints: <https://glints.com/id/lowongan/oop-adalah/#.YqdaqnZBzIV>

rizdiandinata. (2016, November 22). *Konsep OOP Abstract Class Pada Java*. Retrieved Juni 13, 2022, from informatika utama: <https://informa7ics.wordpress.com/2016/11/22/konsep-oop-abstract-class-pada-java/>

Salmaa. (2021, Juni 11). *Pendekatan Penelitian: Pengertian, Jenis-Jenis, dan Contoh Lengkapnya*. Retrieved Juni 14, 2022, from deepublish: <https://penerbitdeepublish.com/pendekatan-penelitian/>