

LAPORAN PRAKTIKUM

*“Laporan ini disusun guna memenuhi salah satu syarat untuk menyelesaikan tugas akhir program studi Analisis Perancangan Sistem Jurusan Teknik Informatika pada Fakultas Teknik*

*Universitas Yudharta”.*

Disusun Oleh :

1. Thita Mar’atus Solikha (201869040031)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS YUDHARTA

PASURUAN

2019

DAFTAR ISI

Judul i

Kata Pengantar ii

Daftar Isi iii

Pertemuan ke 1 1

Pertemuan ke 2 6

Pertemuan ke 3 9

Pertemuan ke 4 13

Pertemuan ke 5 17

Pertemuan ke 6 20

Pertemuan ke 7 23

Pertemuan ke 8 26

Pertemuan ke 9 32

Pertemuan ke 10 36

Pertemuan ke 11 43

Pertemuan ke 12 48

Pertemuan ke 13 52

Pertemuan ke 14 57

Pertemuan ke 15 52

Pertemuan ke 16 57

Pertemuan ke 17 52

Pertemuan ke 18 57

Pertemuan ke 19 52

Pertemuan ke 20 57

Pertemuan ke 21 52

PERTEMUAN KE – *1*

*Kelas dan Obyek*

1. TUJUAN
2. Dapat membaca class diagram sederhana
3. Dapat menyebutkan bagian-bagian kelas
4. Dapat menjelaskan sintak kelas
5. Dapat membuat kelas sederhana
6. Dapat membedakan antara kelas dan obyek
7. Dapat menciptakan Obyek
8. TEORI SINGKAT

Sebuah sistem yang dibangun berdasarkan metoda berorientasi objek adalah sebuah sistem yang komponennya di enkapsulasi menjadi kelompok data dan fungsi, yang dapat mewarisi atribut dan sifat dari komponen lainnya, dan komponen-komponen tersebut saling berinteraksi satu sama lain.

Bahasa Pemrograman yang berorientasi OBJEK memiliki kemampuan dalam pengelolaan program yang lebih diarahkan pada pembentukan objek. Dengan menerapkan konsep ini program akan lebih mudah untuk dikembangkan karena sifatnya yang lebih modular.

Dalam konsep object oriented akan kita temukan kata object dan class, class merupakan pola / template yang menggambarkan kumpulan object yang mempunyai sifat yang sama, perilaku, atau disebut dengan himpunan object sejenis. Sementara object adalah implementasi dari class. Adapun proses pembuatan obyek dari kelas disebut instantiasi atau implementasi dari kelas.

Tabel berikut contoh ilustrasi kelas dan objek.

*Tabel : Contoh class car dan object-object nya*

|  |  |
| --- | --- |
| Class buah-buahan | Objects   * Apel * Pepaya * mangga |

Jadi, kelas adalah template untuk objek, dan objek adalah turunan dari kelas. Ketika objek individu dibuat, mereka mewarisi semua variabel dan metode dari kelas. Anda akan belajar lebih banyak tentang kelas dan objek di bab selanjutnya.

Membuat Class dan Obyek dalam Java

Segala sesuatu di Java dikaitkan dengan kelas dan objek, bersama dengan atribut dan metodenya. Sebagai contoh: dalam kehidupan nyata, mobil adalah objek. Mobil memiliki atribut, seperti berat dan warna, dan metode, seperti drive dan rem.

Kelas seperti konstruktor objek, atau "cetak biru" untuk membuat objek.

* Elemen-elemen dasar dalam mendefinisikan kelas :
  1. Field (variabel) : menyimpan data untuk setiap objek (implementasi dari atribut)
  2. Constructor : setup objek di awal
  3. Method : implementasi perilaku objek

class NamaKelas

{

Fields

Constructor

Methods

}

Untuk membuat obyek digunakan kata kunci new :

NamaKelas NamaObyek = new NamaKelas();

Untuk memanggil method dapat digunakan sintaks sebagai berikut:

namaObyek.nama\_method( [parameter] );

Terdapat 3 konsep penting dalam OOP, yaitu : enkapsulasi, pewarisan, dan polymorfisme. Enkapsulasi bertujuan untuk menyembunyikan kerumitan implementasi dan menyediakan interface bagi pengguna. Selain itu enkapsulasi juga berguna untuk proteksi data (read/write). Pewarisan adalah salah satu cara untuk menggunakan ulang kode-kode yang telah dibuat sebelumnya. Sedangkan polymorfisme adalah kemampuan obyek untuk bereaksi secara berbeda terhadap “pesan yang sama”, tergantung obyek yang menerima pesan tersebut. Kemampuan ini berkat adanya konsep method overriding dan method overloading.

1. PRAKTIK

Praktik 1 . Membuat kelas

Berikut ini adalah contoh class diagram untuk kelas Buku

Buku

- judul: string

- harga : int

-halaman : int

+setBuku():String,int,int

+tampil()

Nama Kelas

atribute

method

Kode program:

class Buku

{

String judul;

int harga;

int halaman;

public void setBuku(String judul,int harga,int halaman)

{

this.judul=judul;

this.harga=harga;

this.halaman=halaman;

}

public void tampil()

{

System.out.println("Judulbuku :"+judul);

System.out.println("Harga buku : "+harga);

System.out.println("Jumlah halaman : "+halaman);

}

}

Praktik 2. Membuat dan Menggunakan Obyek

Kelas Buku di atas hanya merupakan deklarasi dari obyek buku. Agar dapat digunakan maka Kelas Buku tersebut harus diinstantiasi. Dari satu buah kelas dapat dibuat banyak obyek. Obyek-obyek tersebut dapat dibedakan dengan suatu pengenal yang berupa nama.

public class ObyekBuku

{

public static void main(String[] args)

{

Buku buku1=new Buku();

buku1.setBuku("PBO",60000,100);

buku1.tampil();

}

}

Praktik 3. Kelas dan Obyek dengan menggunakan Konstruktor

public class Pegawai

{

String NamaPegawai;

int IdPegawai;

String PosisiPegawai;

Pegawai (String nama,int Id, String posisi) *// konstruktor*

{

NamaPegawai = nama;

IdPegawai = Id;

PosisiPegawai = posisi;

}

Pegawai () *//konstruktor*

{

NamaPegawai = "Thita Mar’atus Solikha";

IdPegawai = 2514;

PosisiPegawai = "Staf pengajar";

}

void Show()

{

System.out.println("Informasi Pegawai");

System.out.println("Nama : "+NamaPegawai);

System.out.println("Id : "+IdPegawai);

System.out.println("Posisi : "+PosisiPegawai);

}

public static void main(String args[])

{

Pegawai pegawai1 = new Pegawai();

Pegawai pegawai2 = new Pegawai("Zahwa",3313,"Staf Akademik");

pegawai1.Show();

pegawai2.Show();

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

*Atribut*

1. TUJUAN
2. Dapat mendefinisikan atribut kelas
3. Dapat membuat dan memanggil atribut
4. Dapat membedakan identifier dan konstanta
5. Dapat mengidentikasi atribut suatu kelas dan mengimplementasi menjadi kelas
6. TEORI SINGKAT

Variabel

Variabel adalah wadah yang menampung nilai-nilai yang digunakan dalam program Java. Setiap variabel harus dideklarasikan menggunakan tipe data. Misalnya, variabel dapat dinyatakan menggunakan salah satu dari delapan tipe data primitif: byte, short, int, panjang, float, double, char atau boolean. Dan, setiap variabel harus diberi nilai awal sebelum bisa digunakan.   
contoh:   
int myAge = 21;   
Variabel "myAge" dinyatakan sebagai tipe int data dan diinisialisasi ke nilai 21.

Dalam definisi kelas, ada tiga jenis variabel.

* instance variables: Setiap metode dalam definisi kelas dapat mengakses variabel tersebut
* Parameter variabel: Hanya metode dimana parameter muncul dapat mengakses variabel tersebut. Ini adalah bagaimana informasi akan diteruskan ke objek.
* local variables: Hanya metode dimana parameter muncul dapat mengakses variabel ini lokal. Variabel ini digunakan untuk menyimpan hasil antara.

Konstanta

Sebuah konstanta adalah variabel yang nilainya tidak dapat mengubah setelah telah ditetapkan. Di Jawa, kata kunci final dapat digunakan dengan tipe data primitif dan objek abadi (misalnya, String) untuk membuat konstanta.   
Namun, dengan menggunakan kata kunci final dengan variabel yang referensi objek normal tidak menjamin isi objek yang akan tetap sama. Ini hanya menjamin bahwa variabel akan selalu memegang referensi ke objek yang sama.

Misal

final int DAYS\_IN\_JANUARY = 31;

1. PRAKTIK

Praktik 1: membuat kelas Lingkaran

public class Lingkaran{ final double phi = 3.14; double jari;

public void setJari(double jari0) { this.jari=jari0; }

public double jari() { return(jari); }

public double luas() { double luas0; luas0=phi\*jari\*jari; return(luas0); }

public void tampil() { System.out.println("jari jari: "+jari); System.out.println("luas: "+luas()); }

}

Praktik 2. Menggunakan kelas Lingkaran

public class Main1

{

public static void main(String[] args)

{

Lingkaran a;

a=new Lingkaran();

a.setJari(10.00);

a.tampil();

}

}

Praktik 3. Menggunakan kelas Lingkaran

public class Main2

{

public static void main(String[] args) { int i; Lingkaran[] a; a=new Lingkaran[5];

for(i=0;i<5;i++) { a[i]=new Lingkaran(); } a[0].setJari(00.00); a[1].setJari(10.00); a[2].setJari(20.00); a[3].setJari(30.00); a[4].setJari(40.00);

for(i=0;i<5;i++) { System.out.println("Lingkaran ke: "+i); a[i].tampil(); } }

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE - 3

*METHOD*

1. TUJUAN
2. Dapat memahami kegunaan dari method
3. Dapat membuat method tanpa return value
4. Dapat mebuat method dengan return value
5. Dapat membuat method Overloading
6. TEORI SINGKAT

Method

Sebuah method menjelaskan behaviour dari sebuah object. Method juga dikenal sebagai fungsi atau prosedur. Pendeklarasian method biasa dituliskan seperti berikut ini :

<modifier> <returnType> <name>(<parameter>\*) {

<statement>\*

}

dimana,

<modifier> dapat menggunakan beberapa *modifier* yang berbeda

<returnType> dapat berupa seluruh tipe data, termasuk *void*

<name> *identifier* atas *class*

<parameter> ::= <tipe\_parameter> <nama\_parameter>[,]

Method Tanpa Return Value

Jenis method ini ditandai dengan *return type* yang berupa *void* dan pada bagian *statement* tidak terdapat keyword return. Pada pemrograman berorientasi obyek method jenis ini digunakan untuk membuat method *mutator*. Nama untuk method *mutator* sebaiknya diawali dengan kata set, misalnya: setNama(), setAlamat().

Method Dengan Return Value

Jenis method ini ditandai dengan *return type* selain *void* dan pada bagian *statement* terdapat keyword return. Pada pemrograman berorientasi obyek method jenis ini digunakan untuk membuat method *acessor*, dimana method *acessor* fungsinya adalah untuk membaca/mendapatkan nilai suatu atribut. Nama untuk method *acessor* sebaiknya diawali dengan kata get, misalnya: getNama(), getAlamat().

Method Overloading

Bahasa java mendukung method *overloading* , java dapat membedakan beberapa *method* dengan nama yang sama di dalam sebuah kelas namun parameternya berbeda. Hal ini sangat menguntungkan karena memudahkan kita dalam mengingat nama method, bayangkan bila program pada class Gambar harus diberi nama drawInterger(int i), drawString(String s), drawDouble(double d). Method *overloading* dibedakan oleh jumlah dan jenis tipe data parameternya.

Contoh method overloading :

public class Gambar{

public void draw(int i){

.....

}

public void draw(String s){

.....

}

public void draw(double d){

.....

}

public void draw(int i, double d){

.....

}

}

1. PRAKTIK

Praktik 1 : membuat method mutator dan acessor

public class Calculation0{

private int a,b;

//method mutator

public void setA(int a) { this.a=a; }

public void setB(int b) { this.b=b; }

//method acessor

public int getA() { return a;}

public int getB() { return b;}

public static void main(String args[]){

Calculation0 obj=new Calculation0();

obj.setA(10);

obj.setB(20);

System.out.println("==Data==");

System.out.println("Nilai pertama"+ obj.getA());

System.out.println("Nilai kedua"+ obj.getB());

}

}

Praktik 2 : membuat method overloading

.

class Calculation{

private int a,b,c;

//method overloading dengan perbedaan jumlah parameter

void tambah(int a,int b){

System.out.println(a+b);

}

void tambah(int a,int b,int c){

System.out.println(a+b+c);

}

public static void main(String args[]){

Calculation obj=new Calculation();

obj.tambah(10,10,10);

obj.tambah(20,20);

}

}

Praktik 3 membuat method overloading

class Calculation2{

private int a,b;

private double c,d;

//method overloading dengan perbedaan tipe data parameter

void sum(int a,int b) {

System.out.println(a+b);

}

void sum(double c,double d) {

System.out.println(c+d);

}

public static void main(String args[]){

Calculation2 obj=new Calculation2();

obj.sum(10.5,10.5);

obj.sum(20,20);

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – *4*

*Konstruktor*

1. TUJUAN

Dapat membuat dan menggunakan konstruktor

1. TEORI SINGKAT

Konstruktor adalah sebuah tipe khusus dari method yang digunakan untuk membuat dan menginisialisasi sebuah object baru.

Berikut ini adalah *property* dari konstruktor :

1. konstruktor memiliki nama yang sama dengan class

2. konstruktor tidak memiliki return value

3. konstruktor tidak dapat dipanggil secara langsung, namun harus dipanggil dengan menggunakan operator *new* pada pembentukan sebuah *class*.

Untuk mendeklarasikan konstruktor, kita tulis

<modifier> <className> (<parameter>) {

<statement>

}

Perbedaan method biasa dengan konstruktor adalah bahwa konstruktor harus memiliki nama yang sama dengan nama classnya dan tidak memiliki nilai kembalian (tipe-data).  
Konstruktor dijalankan pada saat sebuah object diinisialisasi (menggunakan kata new).

*Konstruktor default*

Apabila anda tidak mendeklarasikan satu pun konstruktor, maka Java secara otomatis menambahkan konstruktor default ke dalam class yang kita buat walaupun tidak kelihatan pada kode program. Apabila kita mendeklarasikan satu atau lebih konstruktor maka java tidak akan menambahkan kostruktor default.

*Konstruktor overloading*

Pada konstruktor juga berlaku overloading, artinya boleh mendeklarasikan lebih dari satu konstruktor, asalkan memiliki parameter yang berbeda – beda.

Pada konstruktor :

1. Memiliki konsep yang sama dengan overloading method.
2. Dibedakan berdasarkan paramater (jumlah atau tipe data).
3. Pemanggilan konstruktor tergantung pada instansiasi objek.

C. PRAKTIK :

Praktik 1 Membuat konstruktor

public class Titik{

int x;

int y;

public Titik(){

System.out.println("Konstruktor titik dijalankan!");

}

}

Praktik 2 Membuat obyek dan pemanggilan konstruktor

public class TesTitik {

public static void main(String[] args) {

Titik a=new Titik();

}

}

Praktik 3 Membuat konstruktor overloading dan pemanggilan konstruktor

Tambahkan konstruktor baru pada class Titik

public Titik(int x, int y) {

this.x = x;

this.y = y;

System.out.println("Konstruktor titik 2 dijalankan!");

}

Tambahkan skrip berikut pada class TesTitik

Titik b=new Titik(10,10);

Praktik 4

public class Manusia {

private int umur;

private String nama;

//konstruktor

public Manusia(){

umur = 0;

nama = "";

}

//konstruktor -->untuk inisiasi

public Manusia(int umur, String nama){

this.umur = umur;

this.nama = nama;

}

public void info(){

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

System.out.println("Umur : "+umur);

}

//metode main

public static void main(String args[]){

Manusia m1 = new Manusia();

m1.info();

Manusia m2 = new Manusia(20,"Altha");

m2.info();

}

}

D. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

E. TUGAS :

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE - 5

*ENKAPSULASI dan KOMPOSISI*

1. TUJUAN

Dapat memahami konsep enkapsulasi

Dapat memahami konsep komposisi

Dapat membuat program yang menggunakan enkapsulasi

Dapat membuat program yang menggunakan komposisi

1. TEORI SINGKAT

Enkapsulasi

Adalah salah satu konsep fundamental dalam *object oriented*, lainnya adalah pewarisan, polymorfisme dan abstrak. Enkapsulasi adalah suatu cara pengemasan data dan fungsi dalam sebuah kelas yang terlindungi dari akses secara sembarangan dari pihak luar. Oleh karena itu disebut sebagai proses pengkapsulan, dimana seperti obat bubuk yang terlindungi oleh bungkus kapsul supaya tidak mudah terkontaminasi bakteri. Teknik melakukan enkapsulasi adalah dengan memberikan hak akses private untuk atribut kelas, lalu untuk mengaksesnya dilakukan melalui method yang diberi hak akses public. Jika atribut dideklarasikan dengan hak akses private maka tidak dapat diakses dari luar kelas, maka data tersebut terlindungi. Keuntungan menggunakan enkapsulasi adalah dapat memodifikasi suatu implementasi program tanpa perlu membongkat kode aslinya.

Komposisi

Dalam *object oriented*, satu kelas dapat berelasi dengan kelas lain untuk menggunakan fungsionalitas *method* yang disediakan oleh kelas lain. Hubungan relasi antar kelas diantaranya adalah komposisi. Dua buah kelas dikatakan memiliki relasi komposisi jika salah satu kelas bersifat mempunyai(*own*) kelas yang lain dan apabila pemilik kelas (*owner*) dihapuskan maka kelas yang lain tidak bisa berfungsi. Contohnya kelas manusia mempunyai kelas kepala, kelas tangan, kelas kaki. Bila kelas manusia tersebut dihapus maka kelas kepala, tangan, kaki tidak bisa berfungsi.

Manusia

Kaki

Kepala

Tangan

1. PRAKTIK

Praktik Enkapsulasi

1. Ketikkan kode program berikut ini

public class Pegawai {

private String nama;

private String jabatan;

private int gaji;

public String getJabatan() { return jabatan; }

public void setJabatan(String jabatan) { this.jabatan = jabatan; }

public long getGaji() { return gaji; }

public void setGaji(int gaji) { this.gaji= gaji; }

public String getNama() { return nama; }

public void setNama(String nama) { this.nama = nama; }

public void cetakPegawai(){

System.out.println("");

System.out.println("Nama Pegawai:"+this.nama);

System.out.println("Jabatan:"+this.jabatan);

System.out.println("Gaji Pokok \t:"+this.gaji);

}

}

1. Atribut nama,jabatan, gaji memiliki hak akses private untuk mempertahankan prinsip enkapsulasi. Mengakses atribut tersebut harus menggunakan method yang hak aksesnya bukan private.

public class TestEnkapsulasi{

public static void main(String[] args) {

Pegawai dataPeg=new Pegawai();

dataPeg.setNama("Budi");

dataPeg.setJabatan("Supervisor");

dataPeg.setGaji(4000000);

dataPeg.cetakPegawai();

}

}

Praktik Komposisi

1. Ketikkan program berikut ini

public class Perusahaan {

private String nmPerush;

private String alamat;

private Pegawai peg; //komposisi dengan relasi has-a

public Perusahaan(){

this.nmPerush="PT. Lancar Jaya";

this.alamat="Yogyakarta";

this.peg=new Pegawai();

peg.setNama("David");

peg.setJabatan("Manager");

peg.setGaji(5000000);

}

public void cetakPerush(){

System.out.println("Nama Perusahaan:"+this.nmPerush);

System.out.println("Alamat :"+this.alamat);

peg.cetakPegawai();

}

}

Program no.3 berelasi komposisi dengan program no.1

1. Ketikkan program berikut ini

public class TestKomposisi{

public static void main(String[] args) {

Perusahaan kantor1=new Perusahaan();

kantor1.cetakPerush();

} }

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE 6

*PEWARISAN (INHERITANCE)*

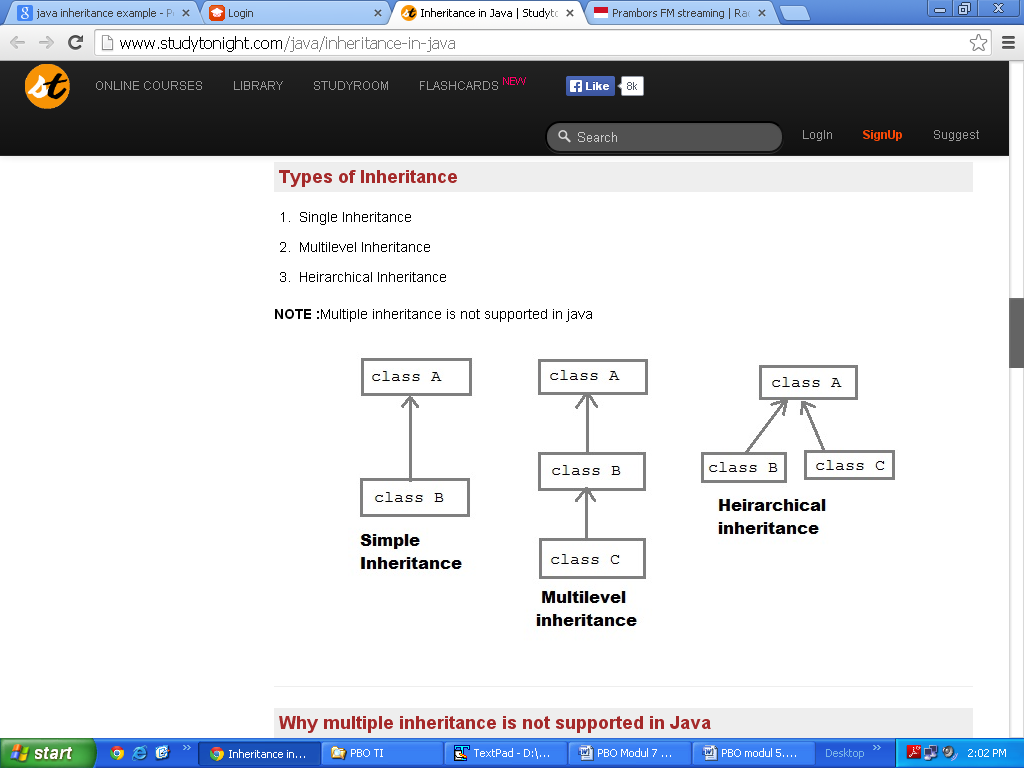
1. TUJUAN

Dapat memahami konsep inheritance

Dapat membuat program yang menggunakan konsep inheritance

1. TEORI SINGKAT

Pewarisan (Inheritance) merupakan sifat dalam bahasa berorientasi objek yang memungkinkan sifat-sifat dari suatu kelas diturunkan ke kelas lain. Sistem OOP memungkinkan kita untuk mendefinisikan suatu kelas baru dengan mewarisi sifat dari kelas lain yang sudah ada. Penurunan sifat ini bisa dilakukan dilakukan secara bertingkat-tingkat, sehingga semakin ke bawah maka kelas tersebut semakin spesifik, contoh, apabila kelas B adalah turunan dari kelas A, dalam proses pembuatan kelas turunan tersebut, Anda dapat menambahkan sifat dan perilaku baru ke dalam kelas B, yang sebelumnya tidak dimiliki dalam kelas A. Dalam terminologi java, kelas induk dinamakan dengan superclass dan kelas turunan dinamakan dengansubclass. untuk membuat kelas anak atau kelas turunan berdasarkan class yang ada, anda dapat menggunakan kata kunci extends. Acces Modifier untuk mewujudkan turunan adalah protected. Berbagai macam bentuk hirarki pewarisan dapat dilihat pada gambar di bawh ini.



1. PRAKTIK
2. Ketikkan kode program berikut ini untuk membuat kelas induk (*super class*)

class Manusia{

protected String nama;

protected int umur;

public Manusia(){

this.nama="Sastro Wardoyo";

this.umur=50;

}

public void cetak(){

System.out.println("======DATA PERSONAL====");

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

System.out.println("Umur:"+umur);

}

}

1. Berikut ini adalah kode program untuk kelas turunannya *(sub class*). Kelas ini menerima warisan data nama dan umur dari kelas Manusia.

class Mahasiswa extends Manusia{

private String nim;

private String jurusan;

public Mahasiswa(){

nama="Kurniawati";

umur=19;

this.nim="145419674";

this.jurusan="TI";

}

public void cetak(){

System.out.println("======DATA MAHASISWA====");

System.out.println("NIM :"+nim);

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

System.out.println("Umur:"+umur);

System.out.println("Jurusan:"+jurusan);

}

}

//program utama

public class TestPewarisan{

public static void main(String args[]){

Manusia orang1=new Manusia();

orang1.cetak();

System.out.println();

System.out.println();

Mahasiswa mhs1=new Mahasiswa();

mhs1.cetak();

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 7

*Pewariasan*

1. TUJUAN

Dapat mendefinisikan overiding metode

mengunakan klausa super

Dapat menggunakan klausa final pada metode dan kelas

1. TEORI SINGKAT

Dalam bab sebelumnya, kita berbicara tentang kelas-kelas yang super dan sub kelas. Jika kelas mewarisi metode dari kelas super, maka ada kesempatan untuk mengganti metode tersebut asalkan tidak ditdanai final.

Manfaat utama adalah: kemampuan untuk menentukan perilaku yang khusus untuk jenis subclass yang berarti subclass dapat menerapkan metode kelas induk berdasarkan kebutuhannya.

Dalam hal berorientasi objek, utama berarti mengesampingkan fungsi dari metode yang ada.

1. PRAKTIK

Praktik 1: membuat Hewan

class Hewan{

public void move(){

System.out.println("Hewans dapat move");

}

}

class Kucing extends Hewan{

public void move(){

System.out.println("Kucings dapat walk dan run");

}

}

public class TestKucing{

public static void main(String args[]){

Hewan a = new Hewan();

Hewan b = new Kucing();

a.move();

b.move();

}

}

Praktik 2. Menggunakan kelas Hewan

class Hewan{

public void move(){

System.out.println("Hewans dapat move");

}

}

class Kucing extends Hewan{

public void move(){

System.out.println("Kucings dapat walk dan run");

}

public void Meong(){

System.out.println("Kucings dapat Meong");

}

}

public class TestKucing{

public static void main(String args[]){

Hewan a = new Hewan();

Hewan b = new Kucing();

a.move();

b.move();

b.Meong();

}

}

Jalankan dan analisis hasilnya.

Praktik 3. Menggunakan kata kunci super

Ketika memanggil versi superclass dari metode ditimpa kata kunci super digunakan.

class Hewan{

public void move(){

System.out.println("Hewans dapat move");

}

}

class Kucing extends Hewan{

public void move(){

super.move();

System.out.println("Kucings dapat walk dan run");

}

}

public class TestKucing{

public static void main(String args[]){

Hewan b = new Kucing();

b.move();

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – *8*

*Package dan Acces modifier*

1. TUJUAN :

* Dapat menggunakan paket java
* Dapat membuat dan menggunakan package yang dibuat
* Dapat mengatur hak akses member

1. TEORI SINGKAT

Seperti yang diketahui, Java adalah pemrograman dengan menggunakan banyak kelas. Kelas-kelas tersebut dikelompokkan ke dalam kategori tertentu yang berhubungan disebut dengan *packages* (paket) . Misalnya saja paket javax.swing.\* berarti semua kelas yang berhubungan dengan javax swing berada dalam direktori tersebut.

Langkah-langkah untuk membuat kelas yang *reusable* adalah sebagai berkut:

1. Deklarasikan *public class*. Jika kelas tidak bersifat public, dia hanya bisa dipergunakan oleh kelas lain di dalam paket yang sama.
2. Pilih nama paket, dan tambahkan sebuah *package declaration* ke file *source code*  untuk deklarasi kelas *reusable*. Hanya bisa ada satu deklarasi paket saja di dalam sebuah file *source code* Java dan dia harus mendahului semua deklarasi dan pernyataan lain di dalam file tersebut.
3. Kompilasi kelas tersebut sehingga dia ditempatkan di dalam struktur direktori paket yang sesuai.
4. Import *reusable class* ke dalam sebuah program, dan gunakan kelas tersebut.

Fitur ini menyediakan mekanisme untuk mengatur class dan interface dalam jumlah banyak dan menghindari konflik pada penamaan.

Menggunakan Salah satu Paket Miliknya Java : Komponen Swing

|  |  |
| --- | --- |
| Komponen | Deskripsi |
| Jlabel | Area dimana teks yang tidak bisa diedit atau sebuah icon dapat ditampilkan |
| JTextField | Area dimana user menginputkan data dari keyboard. Area ini juga bisa menampilkan informasi |
| JButton | Area yang memicu sebuah event ketika diklik dengan mouse |
| JCheckBox | Komponen GUI yang bisa dipilih atau tidak dipilih |
| JComboBox | Sebuah drop-down list dari item dari mana user dapat membuat pilihan dengan meng-klik sebuah item dalam list atau mungkin dengan menuliskan dalam box |
| JList | Area yang berisi sebuah list dari item dari mana user dapat membuat pilihan dengan meng-klik pada beberapa elemen dalam list. Beberapa elemen dapat dipilih |
| JPanel | Sebuah kontainer dalam mana komponen dapat ditempatkan atau diorganisasi |

Object

Component

Container

JComponent

Gambar 1. Superclass untuk banyak komponen Swing

Gambar 1 memperlihatkan sebuah hirarki inheritance yang berisi class yang mendeklarasikan atribut dan lingkungan yang berhubungan dengan banyak komponen Swing. Class object adalah superclass dari hirarki class Java. Class Component (package java.awt) adalah subclass dari Object, class Container (package java.awt) adalah subclass dari Component, dan class Jcomponent (package javax.swing) adalah subclass dari Container.

Struktur Package

package nm\_package;

import registration.processing.\*;

import java.util.List;

import java.lang.\*; //imported by default

class NmClass {

/\* details of NmClass \*/

}

Access Modifier

Public : Menyatakan bahwa kelas/method/attribute tersebut dapat diakses oleh kelas lain dimanapun.

Protected : Menyatakan bahwa kelas/method/attribute tersebut dapat diakses oleh kelas lain yang berada dalam satu package atau kelas lain tersebut merupakan turunannya.

Private : Menyatakan bahwa kelas tersebut tidak dapat diakses sama sekali oleh kelas lain bahkan juga tidak dapat diturunkan.

1. PRAKTIK :
   1. Membuat sebuah paket sederhana

Tuliskan program berikut dalam folder mahasiswa yang berada di dalam direktori kerja anda. Jika belum ada buat dulu.

package mahasiswa;

public class paket

{

public static void isiPaket(){

System.out.println("ini adalah hasil import");

}

}

Simpan dengan nama paket.java pada direktori mahasiswa.

1. Menggunakan paket hasil buatan sendiri. Cobalah program berikut. Simpan dalam direktori kerja anda (di luar folder mahasiswa).

import mahasiswa.\*;

public class cobaPaket

{

public static void main(String args[])

{

paket coba=new paket();

coba.isiPaket();

}

}

1. Modifikasi class paket menjadi seperti berikut :

package mahasiswa;

public class paket

{

String nama;

int umur;

public static void isiPaket(){

System.out.println("ini hasil import");

}

public void setData(int umur, String nama){

this.umur = umur;

this.nama = nama;

}

public void info(){

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

System.out.println("Umur : "+umur);

}

}

1. Panggil method tersebut dari class cobaPaket!
2. Menggunakan paket miliknya java. Cobalah program berikut yang menggunakan paket bawaannya java.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class LabelTest extends JFrame {

private JLabel label;

public LabelTest()

{

super( "Mencoba JLabel" );

Container container = getContentPane();

container.setLayout( new FlowLayout() );

label = new JLabel( "Label dengan text" );

label.setToolTipText( "Ini adalah label1" );

container.add( label );

setSize( 500, 400 );

setVisible( true );

}

public static void main( String args[] )

{

LabelTest aplikasi = new LabelTest();

aplikasi.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

1. Menggunakan hak *access modifier*

class orang

{

private String nama ="";

*//Dideklarasikan sebagai private karena hanya dipanggil pada class orang*

protected String alamat ="";

//Dideklarasikan sebagai protected karena dibutuhkan oleh class ini *dan class dibawahnya*

public orang (String nm, String alm)

*//Dideklarasikan sebagai public karena diperlukan oleh class lainnya*

{ nama = nm;

alamat = alm;

}

public void setNama (String nm)

{ this.nama = nm;}

public void setAlamat (String alm)

{ this.alamat = alm;}

public String getNama()

{ return nama;}

public String getAlamat()

{ return alamat;}

public String displayInfo()

{ return"Nama : " + nama + ", Alamat :" + alamat;

}

}

class dosen extends orang

{ private String nip = "";

//Dideklarasikan sebagai private karena hanya dipanggil pada class dosen

public dosen (String nm, String np, String alm)

//Dideklarasikan sebagai public karena diperlukan oleh class lainnya

{ super (nm,alm);

nip = np;

}

public void setNIP (String np)

{ this.nip = np;}

public String getNIP()

{ return nip;}

public String displayInfo()

{ return super.displayInfo() + ", NIP : " + nip;

}

}

class mahasiswa extends orang

{ private String npm = "";

//Dideklarasikan sebagai private karena hanya dipanggil pada class mahasiswa

public mahasiswa (String nm, String ni, String alm)

//Dideklarasikan sebagai public karena diperlukan oleh class lainnya

{

super(nm,alm);

npm = ni;

}

public void setNpm (String ni)

{ this.npm = ni;}

public String getNpm()

{ return npm;}

public String displayInfo()

{ return super.displayInfo() + ", NPM : " + npm;

} }

public class Perkuliahan

{

public static void main (String[]args)

{

dosen dsn = new dosen ("Arch", "666","Aincrad");

mahasiswa mhs = new mahasiswa ("Azwrath", "999","Alfheim");

System.out.println(dsn.displayInfo());

System.out.println(mhs.displayInfo());

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 9

*kelas Abstrak*

1. TUJUAN

Dapat menjelaskan, mendefinisikan, membuat dan menggunakan kelas asbtrak serta dapat menjelaskan membuat dan menggunakan interface pada aplikasi

1. TEORI SINGKAT

Abstraksi mengacu pada kemampuan untuk membuat kelas abstrak di OOP. Sebuah kelas abstrak adalah salah satu yang tidak bisa dipakai. Semua fungsi lain dari kelas masih ada, dan ladang-ladangnya, metode, dan konstruktor semua diakses dengan cara yang sama. Anda hanya tidak bisa membuat sebuah instance dari kelas abstrak.

Jika kelas abstrak dan tidak dapat dipakai, kelas tidak memiliki banyak digunakan kecuali subclass. Ini biasanya bagaimana kelas abstrak terjadi selama tahap desain. Sebuah kelas induk berisi fungsi umum dari kumpulan anak kelas, tapi kelas induk itu sendiri terlalu abstrak untuk digunakan sendiri.

Abstrak Kelas:

Gunakan kata kunci abstract untuk menyatakan abstract class. Kata kunci muncul dalam deklarasi kelas di suatu tempat sebelum kata kunci kelas.

1. PRAKTIK

Praktik 1: membuat Karyawan

public abstract class Karyawan

{

private String name;

private String address;

private int number;

public Karyawan(String name, String address, int number)

{

System.out.println("Constructing Karyawan");

this.name = name;

this.address = address;

this.number = number;

}

public double computePay()

{

System.out.println("Inside Karyawan computePay");

return 0.0;

}

public void mailCheck()

{

System.out.println("Mailing a check to " + this.name

+ " " + this.address);

}

public String toString()

{

return name + " " + address + " " + number;

}

public String getName()

{

return name;

}

public String getAddress()

{

return address;

}

public void setAddress(String newAddress)

{

address = newAddress;

}

public int getNumber()

{

return number;

}

}

Perhatikan bahwa tidak ada yang berbeda dalam kelas Karyawan ini. Kelas abstrak, tetapi masih memiliki tiga atribut, tujuh metode, dan satu konstruktor.

Praktik 2. Menggunakan kelas Karyawan

public class AbstractDemo

{

public static void main(String [] args)

{

Karyawan e = new Karyawan("George W.", "Houston, TX", 43);

System.out.println("\n Panggil mailCheck pakai Karyawan");

e.mailCheck();

}

}

Ketika dikompile akan didapatkan error sebagai berikut:

Karyawan.java:46: Karyawan is abstract; cannot be instantiated

Karyawan e = new Karyawan("George W.", "Houston, TX", 43);

^

1 error

Praktik 3. Extend kelas Abstrak

public class Gaji extends Karyawan

{

private double Gaji; //Annual Gaji

public Gaji(String name, String address, int number, double Gaji)

{

super(name, address, number);

setGaji(Gaji);

}

public void mailCheck()

{

System.out.println("Within mailCheck Gaji");

System.out.println("Mailing check to " + getName()

+ " with Gaji " + Gaji);

}

public double getGaji()

{

return Gaji;

}

public void setGaji(double newGaji)

{

if(newGaji >= 0.0)

{

Gaji = newGaji;

}

}

public double computePay()

{

System.out.println("Computing Gaji pay for " + getName());

return Gaji/52;

}

}

Praktik 4. Mnggunakan Gaji

Di sini, kita tidak bisa instantiate Karyawan baru, tetapi jika kita instantiate objek Gaji baru, objek Gaji akan mewarisi tiga bidang dan tujuh metode dari karyawan.

public class AbstractDemo

{

public static void main(String [] args)

{

Gaji s = new Gaji("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

Karyawan e = new Gaji("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Panggil mailCheck pakai Gaji");

s.mailCheck();

System.out.println("\n Panggil mailCheck pakai Karyawan");

e.mailCheck();

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – *10*

*Polimorfisme*

1. TUJUAN

* Dapat membuat aplikasi yang mengimplementasi compile-time (early binding) polimorphisme
* Dapat membuat aplikasi yang mengimplementasi runtime (late binding) polimorphisme

1. TEORI SINGKAT

Polimorfisme adalah kemampuan sebuah variabel reference untuk merubah behavior sesuai dengan apa yang dipunyai object.

Polimorfisme membuat objek-objek yang berasal dari subclass yang berbeda, diperlakukan sebagai objek-objek dari satu superclass. Hal ini terjadi ketika memilih method yang sesuai untuk diimplementasikan ke objek tertentu berdasarkan pada subclass yang memiliki method bersangkutan.

Polimorfisme dapat diterjemahkan pula sebagai “sebuah method yang sama namanya (homonim) tetapi mempunyai tingkah laku yang berbeda”. Komputer membedakan method berdasarkan signature method (saat compile) atau berdasarkan reference object (saat runtime).

Ada dua bentuk polimorfisme

1. Overloading ( compile-time (early binding) polimorphisme )

Overloading adalah salah satu cara penerapan dalam konsep polimorfisme. Overload merupakan pendefinisian ulang suatu metode dalam class yang sama. Syarat overload yaitu metode dan tipe parameter harus berbeda dalam kelass yang sama.

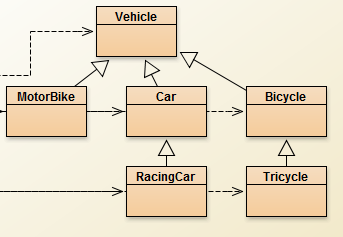
1. Overriding ( runtime (late binding) polimorphisme )

Override merupakan pendefinisian ulang suatu metode oleh subclass. Syarat Override yaitu metode, return type, dan parameter harus sama. Jika tidak sama maka bukan dianggap sebagai override tetapi metode yang baru pada subclass.

Dikatakan late-binding (run time) polymorphism karena saat compile komputer tidak tahu method mana yang harus dipanggil, dan baru akan diketahui saat run-time. Run-time polymorphism dapat diterapkan melalui overridden method. Run time polymorphism juga disebut dengan istilah dynamic binding.

Anggaplah subclass meng-override method tertentu pada superclass. Andai kita cipta objek dari subclass dan memasukkannya dalam superclass. Walau objek subclass telah dimasukkan pada superclass, sewaktu objek tersebut memanggil method yang dioverride, ia tetap memanggil method yang versi subclassnya, bukan versi superclass.

Contoh



* Class Vehicle memiliki method move().
* Class MotorBike meng-override method move().

Java akan menunggu saat runtime untuk menentukan method move() yang mana akan dipanggil.

1. PRAKTIK
   * + 1. Praktik compile-time (early binding) polimorphisme dengan overloading method

class Jumlah{

public int tambah(int x, int y){

return x + y;

}

public int tambah(int x, int y, int z){

return x + y + z;

}

public int tambah(double pi, int x){

return (int)pi + x;

}

}

public class Penjumlahan{

public static void main(String [] args)

{

Jumlah obj = new Jumlah();

System.out.println(obj.tambah(2,5)); // int, int

System.out.println(obj.tambah(2, 5, 9)); // int, int, int

System.out.println(obj.tambah(3.14159, 10)); // double, int

}

}

Pada praktik di atas polymorphism ditunjukkan dengan menggunakan berbagai method *tambah*. Komputer membedakan mana method yang dipanggil berdasarkan signature dari method yaitu parameter inputnya (jumlah parameter input, type data parametr input, dan urutan type data).

Bentuk polymorphism ini dinamakan early-binding (atau compile-time) polymorphism karena komputer mengetahui mana method tambah yang dipanggil setelah mengcompile byte code . Jadi setelah proses compile ketika code sudah menjadi byte code, komputer akan “tahu” method mana yang akan dieksekusi Bentuk ini yang kita kenal dengan overloaded method.

* + - 1. Praktik compile-time (early binding) polimorphisme dengan overloading Konstruktor

class Penggajian{

double gapok;

double masa\_kerja;

Penggajian(double g, double mk)

{

gapok = g;

masa\_kerja=mk;

}

Penggajian()

{

gapok =0;

masa\_kerja=0;

}

Penggajian(double lembur)

{

gapok = masa\_kerja = lembur;

}

double hitung\_gaji()

{

return gapok\*masa\_kerja;

}

}

class OverloadingKonstruktor

{

public static void main(String args[])

{

Penggajian Karyawan1 = new Penggajian(10,15);

Penggajian Karyawan2 = new Penggajian();

Penggajian Karyawan3 = new Penggajian(5);

double gaji;

gaji = Karyawan1.hitung\_gaji();

System.out.println("Gaji Karyawan 1= " +gaji);

gaji = Karyawan2.hitung\_gaji();

System.out.println("Gaji Karyawan 2= " +gaji);

gaji = Karyawan3.hitung\_gaji();

System.out.println("Gaji Karyawan 3= " +gaji);

}

}

1. Runtime polimorfisme dengan overriding method

class Kendaraan {

public void info() {

System.out.println("Info pada kendaraan : ");

}

}

class Roda2 extends Kendaraan {

public void info() {

System.out.println ("Info pada Roda2");

}

}

class Motor extends Roda2

{

public void info() {

System.out.println("Info pada Motor");

}

}

public class Test {

public static void main(String[] args){

Roda2 roda2ku;

Motor motorku;

Kendaraan k = new Kendaraan();

roda2ku=new Roda2();

motorku=new Motor();

k.info();

k=roda2ku;

k.info();

k=motorku;

k.info();

}

}

1. Runtime polimorfisme dengan data member

class Induk {

int x = 5;

public void Info() {

System.out.println("Ini class Induk");

}

}

class Anak extends Induk {

int x = 10;

public void Info() {

System.out.println("Ini class Anak");

}

}

public class Test2 {

public static void main(String args[])

{

Induk tes=new Anak();

System.out.println("Nilai x = " + tes.x);

tes.Info();

}

}

Ketika program di atas dijalankan akan terlihat bahwa ketika diakses atribut x dari objek tes, maka yang muncul adalah nilai x dari super kelas, bukan atribut x dari subkelas. Akan tetapi ketika diakses method Info() dari objek tes, yang muncul adalah method Info() dari sub kelas, bukan method Info() dai superkelas.

1. Runtime polimorfisme dengan passing parameter

class AlatGerak{

void bergerak(){

System.out.println(" Saya mampu bergerak");

}

}

class Sayap extends AlatGerak{

void bergerak(){

System.out.println(" Saya bisa terbang");

}

}

class Kaki extends AlatGerak{

void bergerak(){

System.out.println(" Saya bisa jalan-jalan");

}

}

class Burung {

private AlatGerak alatGerak=new AlatGerak();

Burung(){

System.out.println("Hai saya Burung");

}

public void bergerak() {

alatGerak.bergerak();

}

public void setAlatGerak(AlatGerak alatGerak) {

this.alatGerak = alatGerak;

System.out.println("Sekarang saya pakai " + alatGerak);

}

}

class BurungTest{

public static void main(String[] args){

Burung merpati=new Burung();

merpati.bergerak();

Sayap sayap=new Sayap();

Kaki kaki=new Kaki();

merpati.setAlatGerak(sayap);

merpati.bergerak();

merpati.setAlatGerak(kaki);

merpati.bergerak();

}

}

Method setAlatGerak dapat menerima berbagai type yaitu Sayap dan Kaki. Hal ini legal karena pada kenyataannya Sayap dan Kaki adalah AlatGerak. Saat method bergerak() milik object trutle dipanggil, method yang dipanggil adalah sesuai dengan method yang dimiliki oleh type yang diberikan.

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 11

Exception Handling

1. TUJUAN

Dapat mengenal berbagai macam exception

Dapat membuat exception handling

1. TEORI SINGKAT

Exception adalah sebuah masalah yang terjadi ketika program sudah pada tahap eksekusi, bukan saat tahap *compile.* Sebuah *exception* terjadi karena beberapa penyebab, misalnya: user memasukkan data yang tidak valid, file yang akan dibuka tidak ditemukan, koneksi yang belum tersambung atau tiba-tiba putus ditengah proses. Terdapat tiga macam exception:

1. Checked Exception : adalah exception yang biasanya terjadi karena kesalahan pengguna atau suatu masalah yang terjadi di luar perkiraan programmer. Contohnya sebuah file yang akan dibuka tapi file tersebut tidak dapat ditemukan maka exception terjadi. Kejadian seperti ini tidak terdeteksi pada saat tahap kompilasi.
2. Runtime Exception : adalah exception yang mungkin bisa dihindari oleh programmer. Kebalikan dari Checked Exception, bahwa Runtime Exception diabaikan pada tahap kompilasi.
3. Errors : adalah masalah yang terjadi diluar kontrol user ataupun programmer. Error sering diabaikan di dalam program karena anda tidak bisa menanganinya secara keseluruhan. Contohnya: jika kapasitas di memori penuh. Error seperti ini tidak terdeteksi pada tahap kompilasi

Hirarki kelas yang menangani exception:

Kelas paling tinggi dalam menangani exception

Object

Throwable

Exception

Error

IOexception

RuntimeException

Struktur penulisan exception adalah :

try {

// blok program tempat menuliskan statement yang akan di uji apakan ada kesalahan atau tidak

}

catch ( <tipe exception1> <nama variabel>) {

// blok program tempat menuliskan aksi program bila sebuah exception terjadi

}

. . .

catch ( <tipe exception2> <nama variabel>) {

// blok program tempat menuliskan aksi program bila sebuah exception terjadi

}

finally {

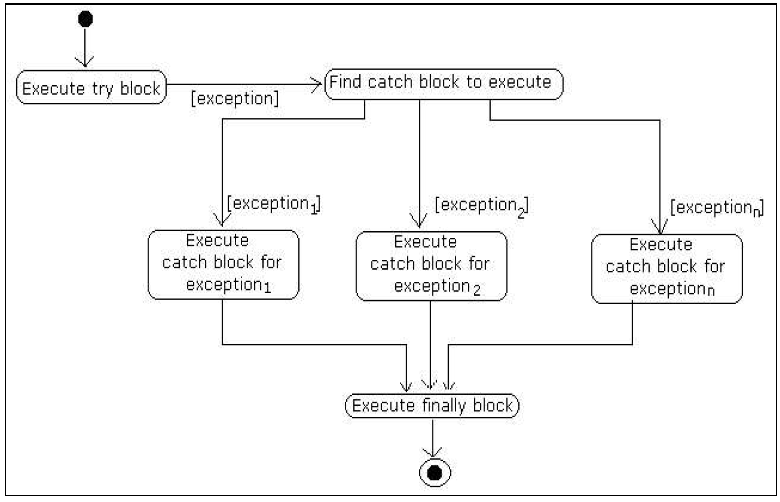
// blok program lanjutan

}

Ketentuan dalam membuat exception adalah:

1. Wajib membuat notasi blok
2. Setiap blok try boleh memiliki lebih dari satu blok catch dan hanya boleh memiliki satu blok finally
3. Blok catch dan blok finally harus muncul bersama blok try
4. Blok try harus diikuti minimal satu blok catch, atau satu blok finally, atau kedua blok catch dan finally
5. Setiap blok catch mendefinisikan penanganan exception. Di dalam header blok catch terdapat satu argumen yang akan ditangani oleh blok exception. Exception harus berasal dari class Throwable atau dari class turunannya.

Alur kerja dari exception :



1. PRAKTIK

Praktik 1 . Membuat kelas tanpa exception

1. public class BagiNol {

2. public static void main(String[] args) {

3. System.out.println("Sebelum pembagian");

4. System.out.println(5/0);

5. System.out.println("Sesudah pembagian");

6. }

7. }

Baris ke-5 tidak akan dieksekusi karena ada kesalahan pembagian dengan bilangan nol pada baris ke-4

Praktik 2 . Membuat kelas menggunakan exception

1. public class BagiNol2 {

2. public static void main(String[] args) {

3. System.out.println("Sebelum pembagian");

4. try { System.out.println(5/0); }

5. catch (Exception t) {

6. System.out.print("Pesan kesalahan: ");

7. System.out.println(t.getMessage());

8. }

9. System.out.println("Sesudah pembagian");

10. }

11. }

Program tidak berhenti di baris ke -4 , dibuktikan dengan munculnya hasil dari baris ke-9

Praktik 3 . Menggunakan lebih dari satu statement catch

public class MultiCatchException{

public static void main(String args[]){

try{

System.out.println(5/0);

}

catch(ArithmeticException e){

System.out.println(0);

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println(1);

}

catch(Exception e){

System.out.println(2);

}

}

}

Output adalah angka nol yang artinya pada blok try terdapat kesalahan *arithmetic* maka kesalahan tersebut ditangkap oleh blok catch dari kelas ArithmeticException.

Praktik 4 . Menggunakan statement finally

public class MultiCatchException{

public static void main(String args[]){

try{

System.out.println(5/0);

}

catch(ArithmeticException e){

System.out.println(0);

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println(1);

}

catch(Exception e){

System.out.println(2);

}

finally{

System.out.println(3);

}

}

}

Bagian blok *finally* adalah bagian blok yang selalu dikerjakan. Output program adalah angka nol dan tiga, artinya setelah salah satu blok *catch* dieksekusi maka alur program selanjutnya adalah mengeksekusi bagian blok *finally.*

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE- 12

Inner Class

1. TUJUAN

* Mempelajari konsep nested class
* Membuat program yang menggunakan inner class dan anonymous class

1. TEORI

Java mengijinkan kita mendefiniskan suatu kelas di dalam kelas lain, hal ini disebut sebagai *nested class*. Disebut nested karena class bersifat tersarang terhadap kelas – kelas utamanya, seperti halnya blok penyeleksian  (if, for) yang tersarang pada blok penyeleksian lainnya atau method yang tersarang pada method lainnya. *nested class* dibagi menjadi dua kategori, yaitu *static* dan *non static nested class*. Istilah *inner class* biasa digunakan untuk *non static nested class*.

Class OuterClass{

...

static class StaticNestedClass{

...

}

Class InnerClass{

...

}

}

Nested class merupakan member dari kelas Outer(kelas terluar). Non static nested class (inner class) dapat mengakses semua member dari kelas Outer bahkan yang mempunyai hak akses private. Static nested class tidak mempunyai akses kepada member dari kelas Outer. Sebagai member dari Outer class, sebuah nested class dapat dideklarasikan dengan hak akses private, public, proteced, sedangkan Outer class hanya bisa dideklarasikan dengan hak akses public.

Anonymous inner class adalah sebuah inner class yang dideklarasikan tanpa memberikan nama kelas. Anonymous class akan membuat kode program menjadi lebih ringkas karena dengan Anonymous class kita dapat mendeklarasikan dan meng-instant suatu kelas sekaligus dalam satu langkah.

1. PRAKTIK

Membuat inner class

1. Tulislah program file Luar.java berikut:

public class Luar{

private int angkaLuar;

public Luar(int angkaLuar){

this.angkaLuar=angkaLuar;

}

int getAngkaLuar(){

return angkaLuar;

}

//inner class

class Dalam{

private int angkaDalam;

public Dalam(){

angkaDalam=9;

}

public void cetakDalam(){

System.out.println("Ini angka luar:"+angkaLuar);

System.out.println("Ini angka dalam:"+angkaDalam);

}

}//batas inner class

public void cetakLuar(){

Dalam dl=new Dalam();

dl.cetakDalam();

}

public static void main(String args[]){

Luar lu=new Luar(5);

lu.cetakLuar();

}

}//oec out

1. Tulislah program file RoundShape.java berikut:

public abstract class RoundShape

{ // coordinates of center represented by an inner class

protected class Center

{ int x,y; }

protected Center C = new Center();

protected float radiusOfCircle;

// constructor

public RoundShape(int xCenter, int yCenter, float radius)

{

C.x=xCenter;

C.y=yCenter;

radiusOfCircle = radius;

}

// abstract method

abstract public float area();

}

1. Tulislah program file Circle.java berikut:

public class Circle extends RoundShape

{ // constructor

public Circle(int xCenter, int yCenter, float radius)

{ super(xCenter, yCenter, radius); }

public float area()

{

float areaOfCircle = (float)(Math.PI\*Math.pow(radiusOfCircle,2.0));

return areaOfCircle;

}

}

1. Tuliskan program file Sphere.java

public class Sphere extends RoundShape

{

public Sphere(int xCenter, int yCenter, float radius)

{ super(xCenter, yCenter, radius); }

public float area()

{ float surfaceArea = (float)(4.0\*Math.PI\*Math.pow(radiusOfCircle,2.0));

return surfaceArea;

}

}

1. Tuliskan TestRoundShape.java

import java.text.DecimalFormat;

public class TestRoundShape{

public static void main(String args[]){

DecimalFormat digit = new DecimalFormat ("0.##");

Circle lingk1=new Circle(20,5,10.0f);

System.out.println(digit.format(lingk1.area()));

}

}

Praktik membuat Anonymous Class

1. Tuliskan TestAnonymous.java

class Awal{

int x=8;

void methodAwal(){

System.out.println("Nilai x= "+x);

}

}

public class TestAnonymous{

public static void main(String args[]){

Awal noName=new Awal(){ //instant anonymous class

void methodAwal(){

x+=3;

System.out.println("x= "+x);

}

};

noName.methodAwal();

}

}

Baris program di bawah tulisan ”public static void main” adalah proses instant dari sebuah anonymous class. Obyek noName bukan obyek dari kelas Awal namun obyek dari anonymous class hal ini dapat dilihat dari nilai x=11, bukan x=8 ketika program dijalankan

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

**PERTEMUAN KE – 13**

***Java* Abstraction**

1. **TUJUAN**

Mempelajari Abstraksi data menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna.

1. **TEORI SINGKAT**

Abstraksi dapat dicapai dengan kelas atau antarmuka abstrak (yang akan Anda pelajari lebih lanjut di bab selanjutnya).

Kata kunci abstrak adalah pengubah non-akses, digunakan untuk kelas dan metode:  
• Kelas abstrak: adalah kelas terbatas yang tidak dapat digunakan untuk membuat objek (untuk mengaksesnya, itu harus diwarisi dari kelas lain).

• Metode abstrak: hanya dapat digunakan dalam kelas abstrak, dan tidak memiliki tubuh. Tubuh disediakan oleh subclass (diwarisi dari)

1. **PRAKTIK**

Praktik 1: membuat Abstrak dan Reguler :

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

Dari contoh di atas, tidak mungkin untuk membuat objek dari kelas Hewan:

Animal myObj = new Animal(); // will generate an error

Untuk mengakses kelas abstrak, itu harus diwarisi dari kelas lain. Mari kita konversi kelas Animal yang kita gunakan di bab Polimorfisme menjadi kelas abstrak:

Ingat dari bab Warisan bahwa kami menggunakan kata kunci extends untuk mewarisi dari suatu kelas.

Contoh :

// Abstract class

abstract class Animal {

// Abstract method (does not have a body)

public abstract void animalSound();

// Regular method

public void sleep() {

System.out.println("Zzz");

}

}

// Subclass (inherit from Animal)

class Pig extends Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The pig says: wee wee");

}

}

class MyMainClass {

public static void main(String[] args) {

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound();

myPig.sleep();

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 14

*Java Interface*

1. TUJUAN

* Memahami tentang Cara lain untuk mencapai abstraksi di Jawa, adalah dengan antarmuka

1. TEORI SINGKAT

Antarmuka adalah "kelas abstrak" yang sepenuhnya digunakan untuk mengelompokkan metode terkait dengan benda kosong:

1. PRAKTIK

// interface

interface Animal {

public void animalSound(); // interface method (does not have a body)

public void run(); // interface method (does not have a body)

}

Untuk mengakses metode antarmuka, antarmuka harus "diimplementasikan" (agak seperti diwariskan) oleh kelas lain dengan kata kunci implement (bukan extends). Isi metode antarmuka disediakan oleh kelas "implement":

|  |
| --- |
| // Interface  interface Animal {  public void animalSound(); // interface method (does not have a body)  public void sleep(); // interface method (does not have a body)  }  // Pig "implements" the Animal interface  class Pig implements Animal {  public void animalSound() {  // The body of animalSound() is provided here  System.out.println("The pig says: wee wee");  }  public void sleep() {  // The body of sleep() is provided here  System.out.println("Zzz");  }  }  class MyMainClass {  public static void main(String[] args) {  Pig myPig = new Pig(); // Create a Pig object  myPig.animalSound();  myPig.sleep();  }  } |

Catatan tentang Antarmuka: • Seperti kelas abstrak, antarmuka tidak dapat digunakan untuk membuat objek (dalam contoh di atas, tidak mungkin membuat objek "Hewan" di MyMainClass) • Metode antarmuka tidak memiliki tubuh - tubuh disediakan oleh kelas "implement" • Pada implementasi antarmuka, Anda harus mengganti semua metodenya • Metode antarmuka secara abstrak abstrak dan publik • Atribut antarmuka secara publik default, statis dan final • Antarmuka tidak dapat berisi konstruktor (karena tidak dapat digunakan untuk membuat objek)

Praktik membuat Multiple interfaces

Untuk mengimplementasikan beberapa antarmuka, pisahkan dengan koma:

interface FirstInterface {

public void myMethod(); // interface method

}

interface SecondInterface {

public void myOtherMethod(); // interface method

}

class DemoClass implements FirstInterface, SecondInterface {

public void myMethod() {

System.out.println("Some text..");

}

public void myOtherMethod() {

System.out.println("Some other text...");

}

}

class MyMainClass {

public static void main(String[] args) {

DemoClass myObj = new DemoClass();

myObj.myMethod();

myObj.myOtherMethod();

}

}

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 15

*Java Enums*

1. TUJUAN

* khusus yang mewakili sekelompok konstanta (variabel yang tidak dapat diubah, seperti variabel akhir).
* membuat enum, gunakan kata kunci enum (bukan kelas atau antarmuka), dan pisahkan konstanta dengan koma

1. TEORI SINGKAT

Perbedaan antara Enums dan Kelas

Enum can, seperti halnya kelas, memiliki atribut dan metode. Satu-satunya perbedaan adalah bahwa konstanta enum bersifat publik, statis, dan final (tidak dapat diubah - tidak dapat diganti).

Enum tidak dapat digunakan untuk membuat objek, dan itu tidak dapat memperluas kelas lain (tetapi dapat mengimplementasikan antarmuka).

.

1. PRAKTIK

enum Level {

LOW,

MEDIUM,

HIGH

}

Anda dapat mengakses konstanta enum dengan sintaks dot:

Level myVar = Level.MEDIUM;

Enum adalah kependekan dari "enumerasi", yang berarti "terdaftar secara khusus".

Praktik membuat Enum inside a Class  
Anda juga dapat memiliki enum di dalam kelas:

public class MyClass {

enum Level {

LOW,

MEDIUM,

HIGH

}

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

System.out.println(myVar);

}

}

Praktik membuat Enum in a Switch Statement

Enum sering digunakan dalam pernyataan sakelar untuk memeriksa nilai yang sesuai:

enum Level {

LOW,

MEDIUM,

HIGH

}

public class MyClass {

public static void main(String[] args) {

Level myVar = Level.MEDIUM;

switch(myVar) {

case LOW:

System.out.println("Low level");

break;

case MEDIUM:

System.out.println("Medium level");

break;

case HIGH:

System.out.println("High level");

break;

}

}

}

Praktik membuat Loop Through an Enum

Tipe enum memiliki metode nilai (), yang mengembalikan array dari semua konstanta enum. Metode ini berguna ketika Anda ingin mengulangi konstanta enum:

for (Level myVar : Level.values()) {

System.out.println(myVar);

}

|  |
| --- |
| The output will be:  LOW MEDIUM HIGH |

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 16

*Java User Input*

1. TUJUAN

* khusus yang mewakili sekelompok konstanta (variabel yang tidak dapat diubah, seperti variabel akhir).
* membuat enum, gunakan kata kunci enum (bukan kelas atau antarmuka), dan pisahkan konstanta dengan koma

1. TEORI SINGKAT

Kelas scanner digunakan untuk mendapatkan input pengguna, dan ditemukan dalam paket Java. util. Untuk menggunakan kelas scanner, membuat objek kelas dan menggunakan salah satu metode yang tersedia yang ditemukan di dokumentasi kelas scanner. Dalam contoh kita, kita akan menggunakan metode nextLine (), yang digunakan untuk membaca string:

1. PRAKTIK

|  |
| --- |
| import java.util.Scanner; // Import the Scanner class  class MyClass {  public static void main(String[] args) {  Scanner myObj = new Scanner(System.in); // Create a Scanner object  System.out.println("Enter username");  String userName = myObj.nextLine(); // Read user input  System.out.println("Username is: " + userName); // Output user input  }  } |

|  |  |
| --- | --- |
| Method | Description |
| nextBoolean() | Reads a boolean value from the user |
| nextByte() | Reads a byte value from the user |
| nextDouble() | Reads a double value from the user |
| nextFloat() | Reads a float value from the user |
| nextInt() | Reads a int value from the user |
| nextLine() | Reads a String value from the user |
| nextLong() | Reads a long value from the user |
| nextShort() | Reads a short value from the user |

|  |
| --- |
| System.out.println("Enter username");  String userName = myObj.nextLine(); // Read user input  System.out.println("Username is: " + userName); // Output user input  }  } |

Input Types

Pada contoh di atas, kita menggunakan metode nextLine (), yang digunakan untuk membaca Strings. Untuk membaca jenis lain, lihat tabel di bawah ini:

|  |
| --- |
| import java.util.Scanner;  class MyClass {  public static void main(String[] args) {  Scanner myObj = new Scanner(System.in);  System.out.println("Enter name, age and salary:");  // String input  String name = myObj.nextLine();  // Numerical input  int age = myObj.nextInt();  double salary = myObj.nextDouble();  // Output input by user  System.out.println("Name: " + name);  System.out.println("Age: " + age);  System.out.println("Salary: " + salary);  }  } |

PERTEMUAN KE – 17

*Java Date And Time*

1. TUJUAN

* khusus yang mewakili sekelompok konstanta (variabel yang tidak dapat diubah, seperti variabel akhir).
* membuat enum, gunakan kata kunci enum (bukan kelas atau antarmuka), dan pisahkan konstanta dengan koma

1. TEORI SINGKAT

"T" pada contoh di atas digunakan untuk memisahkan tanggal dari waktu. Anda dapat menggunakan kelas DateTimeFormatter dengan metode ofPattern () dalam paket yang sama untuk memformat atau memilah objek tanggal-waktu. Contoh berikut akan menghapus "T" dan milidetik dari tanggal-waktu:

|  |  |  |
| --- | --- | --- |
| Value | Example | Tryit |
| *yyyy-MM-dd* | "1988-09-29" | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_date_format) |
| *dd/MM/yyyy* | "29/09/1988" | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_date_formatter1) |
| *dd-MMM-yyyy* | "29-Sep-1988" | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_date_formatter2) |
| *E, MMM dd yyyy* | "Thu, Sep 29 1988" | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_date_formatter3) |

|  |
| --- |
| import java.time.LocalDateTime; // Import the LocalDateTime class  import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class  public class MyClass {  public static void main(String[] args) {  LocalDateTime myDateObj = LocalDateTime.now();  System.out.println("Before formatting: " + myDateObj);  DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");  String formattedDate = myDateObj.format(myFormatObj);  System.out.println("After formatting: " + formattedDate);  }  }  The output will be:  Before Formatting: 2020-01-06T23:11:53.866093  After Formatting: 06-01-2020 23:11:53 |

1. *PRAKTIK*
2. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

PERTEMUAN KE – 18

*Java ArrayList*

1. TUJUAN

* khusus yang mewakili sekelompok konstanta (variabel yang tidak dapat diubah, seperti variabel akhir).
* membuat enum, gunakan kata kunci enum (bukan kelas atau antarmuka), dan pisahkan konstanta dengan koma

1. TEORI SINGKAT

ArrayList class adalah resizable array, yang dapat ditemukan dalam paket java.util. Perbedaan antara array bawaan dan ArrayList di Java, adalah bahwa ukuran array tidak dapat dimodifikasi (jika kita ingin menambah atau menghapus elemen ke / dari array, kita harus membuat yang baru). Sementara elemen dapat ditambahkan dan dihapus dari ArrayList kapan pun kita mau. Sintaksnya juga sedikit berbeda. ArrayList class memiliki banyak method yang berguna. Misalnya, untuk menambahkan elemen ke ArrayList, gunakan method add().

1. **PRAKTIK**

|  |
| --- |
| *Create an ArrayList object called cars that will store strings:*  *import java.util.ArrayList; // import the ArrayList class*  *ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object* |

**Add Items**

Kelas ArrayList memiliki banyak metode yang berguna. Misalnya, untuk menambahkan elemen ke ArrayList, gunakan metode add ():

|  |
| --- |
| import java.util.ArrayList;  public class MyClass {  public static void main(String[] args) {  ArrayList<String> cars = new ArrayList<String>();  cars.add("Volvo");  cars.add("BMW");  cars.add("Ford");  cars.add("Mazda");  System.out.println(cars);  }  } |

**Access an Item**

Untuk mengakses elemen di ArrayList, gunakan metode get () dan lihat nomor indeks:

**Change an Item**

Untuk memodifikasi elemen, gunakan metode set () dan lihat nomor indeks:

|  |
| --- |
| cars.set(0, "Opel"); |

**Remove an Item**

Untuk menghapus elemen, gunakan metode hapus () dan lihat nomor indeks:

|  |
| --- |
| cars.remove(0); |

Untuk menghapus semua elemen di ArrayList, gunakan metode clear ():

|  |
| --- |
| cars.clear(); |

**ArrayList Size**

Untuk mengetahui berapa banyak elemen yang dimiliki ArrayList, gunakan metode ukuran:

|  |
| --- |
| cars.size(); |

**Loop Through an ArrayList**

Ulangi elemen ArrayList dengan for for loop, dan gunakan metode size () untuk menentukan berapa kali loop harus dijalankan:

|  |
| --- |
| public class MyClass {  public static void main(String[] args) {  ArrayList<String> cars = new ArrayList<String>();  cars.add("Volvo");  cars.add("BMW");  cars.add("Ford");  cars.add("Mazda");  for (int i = 0; i < cars.size(); i++) {  System.out.println(cars.get(i));  }  }  } |

Anda juga dapat mengulang melalui ArrayList dengan untuk masing-masing loop:

|  |
| --- |
| public class MyClass {  public static void main(String[] args) {  ArrayList<String> cars = new ArrayList<String>();  cars.add("Volvo");  cars.add("BMW");  cars.add("Ford");  cars.add("Mazda");  for (String i : cars) {  System.out.println(i);  }  }  } |

**Other Types**

|  |
| --- |
| Create an ArrayList to store numbers (add elements of type Integer):  import java.util.ArrayList;  public class MyClass {  public static void main(String[] args) {  ArrayList<Integer> myNumbers = new ArrayList<Integer>();  myNumbers.add(10);  myNumbers.add(15);  myNumbers.add(20);  myNumbers.add(25);  for (int i : myNumbers) {  System.out.println(i);  }  }  } |

Elemen dalam ArrayList sebenarnya adalah objek. Dalam contoh di atas, kami membuat elemen (objek) dari tipe "String". Ingatlah bahwa String di Java adalah objek (bukan tipe primitif). Untuk menggunakan tipe lain, seperti int, Anda harus menentukan kelas pembungkus yang setara: Integer. Untuk jenis primitif lainnya, gunakan: Boolean untuk boolean, Karakter untuk char, Double untuk double, dll:

**Sort an ArrayList**

Kelas lain yang berguna dalam paket java.util adalah kelas Collections, yang mencakup metode sort () untuk menyortir daftar berdasarkan abjad atau angka:

|  |
| --- |
| Sort an ArrayList of Strings:  import java.util.ArrayList;  import java.util.Collections; // Import the Collections class  public class MyClass {  public static void main(String[] args) {  ArrayList<String> cars = new ArrayList<String>();  cars.add("Volvo");  cars.add("BMW");  cars.add("Ford");  cars.add("Mazda");  Collections.sort(cars); // Sort cars  for (String i : cars) {  System.out.println(i);  }  }  } |

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

**PERTEMUAN KE – 19**

***Java HashMap***

1. **TUJUAN**

Mempelajari Abstraksi data menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna.

1. **TEORI SINGKAT**

**HashMap**, menyimpan item dalam pasangan "**kunci / nilai**", dan kita dapat mengaksesnya dengan indeks jenis lain (mis. **String**). Satu objek digunakan sebagai kunci (indeks) ke objek lain (nilai). Ini dapat menyimpan berbagai jenis: Kunci **String** dan nilai **Integer**, atau tipe yang sama, seperti: Kunci **String** dan nilai **String**. **HashMap class** memiliki banyak method yang berguna. Misalnya, untuk menambahkan item ke dalamnya, gunakan method **put()**. Untuk mengakses nilai di **HashMap**, gunakan method **get()** dan lihat kuncinya. Untuk menghapus item, gunakan method **remove()** dan lihat kuncinya. Untuk menghapus semua item, gunakan method **clear()**. Untuk mengetahui berapa banyak item yang ada, gunakan method **size**. Ulangi item-item dari **HashMap** dengan **for-each** loop. Gunakan method **keySet()** jika kita hanya menginginkan kunci, dan gunakan method **values()** jika kita hanya menginginkan nilai. Kunci dan nilai dalam HashMap sebenarnya adalah objek. Untuk menggunakan tipe lain, seperti **int**, kita harus menentukan wrapper class yang setara: **Integer**. Untuk jenis primitif lainnya, gunakan: Boolean untuk **boolean**, Karakter untuk **char**, Double untuk **double**, dll.

1. **PRAKTIK**

Praktik 1:

|  |
| --- |
| Create a HashMap object called capitalCities that will store String keys and String values:  import java.util.HashMap; // import the HashMap class  HashMap<String, String> capitalCities = new HashMap<String, String>(); |

Add Items

|  |
| --- |
| // Import the HashMap class  import java.util.HashMap;  public class MyClass {  public static void main(String[] args) {  // Create a HashMap object called capitalCities  HashMap<String, String> capitalCities = new HashMap<String, String>();  // Add keys and values (Country, City)  capitalCities.put("England", "London");  capitalCities.put("Germany", "Berlin");  capitalCities.put("Norway", "Oslo");  capitalCities.put("USA", "Washington DC");  System.out.println(capitalCities);  }  } |

Loop Through a HashMap

|  |
| --- |
| // Print keys  for (String i : capitalCities.keySet()) {  System.out.println(i);  } |

Other Types

|  |
| --- |
| Create a HashMap object called people that will store String keys and Integer values:  // Import the HashMap class  import java.util.HashMap;  public class MyClass {  public static void main(String[] args) {  // Create a HashMap object called people  HashMap<String, Integer> people = new HashMap<String, Integer>();  // Add keys and values (Name, Age)  people.put("John", 32);  people.put("Steve", 30);  people.put("Angie", 33);  for (String i : people.keySet()) {  System.out.println("key: " + i + " value: " + people.get(i));  }  }  } |

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

**PERTEMUAN KE – 20**

***Java Wrapper Classes***

1. **TUJUAN**

Mempelajari Abstraksi data menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna.

1. **TEORI SINGKAT**

**Wrapper Classes** menyediakan cara untuk menggunakan tipe data primitif (**int**, **boolean**, dll.) sebagai objek. Di bawah ini menunjukkan tipe primitif dan **Wrapper Classes** yang setara. byte - Byte

int - Integer

long - Long

float - Float

double - Double

boolean - Boolean

char - Character

Kadang-kadang kita harus menggunakan **Wrapper Classes**, misalnya saat bekerja dengan objek Koleksi, seperti **ArrayList**, di mana tipe primitif tidak dapat digunakan (daftar hanya dapat menyimpan objek). Untuk membuat objek wrapper, gunakan **Wrapper Classes** alih-alih tipe primitif. Untuk mendapatkan nilai, kita cukup mencetak objek. Karena kita sekarang bekerja dengan objek, kita dapat menggunakan method tertentu untuk mendapatkan informasi tentang objek tertentu.

Sebagai contoh, method berikut ini digunakan untuk mendapatkan nilai yang terkait dengan objek wrapper yang sesuai: **intValue()**, **byteValue()**, **shortValue()**, **longValue()**, **floatValue()**, **doubleValue()**, **charValue()**, **booleanValue()**. Method lain yang bermanfaat adalah method **toString()**, yang digunakan untuk mengubah objek wrapper menjadi string.

1. **PRAKTIK**

|  |  |
| --- | --- |
| **Primitive Data Type** | **Wrapper Class** |
| Byte | Byte |
| Short | Short |
| Int | Integer |
| Long | Long |
| Float | Float |
| Double | Double |
| Boolean | Boolean |
| Char | Character |

|  |
| --- |
| ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid  ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid |

**Creating Wrapper Objects**

|  |
| --- |
| public class MyClass {  public static void main(String[] args) {  Integer myInt = 5;  Double myDouble = 5.99;  Character myChar = 'A';  System.out.println(myInt);  System.out.println(myDouble);  System.out.println(myChar);  }  } |

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

***PERTEMUAN KE – 21***

***Java Exceptions***

1. **TUJUAN**

Mempelajari Abstraksi data menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna.

1. **TEORI SINGKAT**

Saat menjalankan kode Java, kesalahan yang berbeda dapat terjadi: kesalahan pengkodean yang dibuat oleh programmer, kesalahan karena input yang salah, atau hal-hal yang tidak terduga lainnya. Ketika kesalahan terjadi, Java biasanya akan berhenti dan menghasilkan pesan kesalahan. Istilah teknis untuk ini adalah: Java akan melempar **exception** (melempar kesalahan).

Pernyataan **try** memungkinkan kita untuk menentukan blok kode yang akan diuji untuk kesalahan saat sedang dieksekusi.

Pernyataan **catch** memungkinkan kita untuk menentukan blok kode yang akan dieksekusi, jika kesalahan terjadi di blok coba.

Kata kunci **try** dan **catch** berpasangan. Jika terjadi kesalahan, kita dapat menggunakan **try...catch** untuk menangkap kesalahan dan menjalankan beberapa kode untuk menanganinya. Pernyataan **finally** memungkinkan kita menjalankan kode, setelah **try...catch**, terlepas dari hasilnya. Pernyataan **throw** memungkinkan kita untuk membuat kesalahan khusus. Pernyataan **throw** digunakan bersama dengan **exception type**. Ada banyak jenis pengecualian yang tersedia di Java: **ArithmeticException**, **FileNotFoundException**, **ArrayIndexOutOfBoundsException**, **SecurityException**, dll.

1. **PRAKTIK**

|  |
| --- |
| **public class MyClass {**  **static void checkAge(int age) {**  **if (age < 18) {**  **throw new ArithmeticException("Access denied - You must be at least 18 years old.");**  **}**  **else {**  **System.out.println("Access granted - You are old enough!");**  **}**  **}** |

1. LATIHAN

* *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
* *Dikerjakan di laboratorium pada jam praktikum.*

1. TUGAS

* *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
* *Dikerjakan di rumah dan dilampirkan pada laporan.*

**ANALISA HASIL PERCOBAAN**

**Modul 1**

Modul 1a adalah Praktikum Multiple Parameters, ketika kita membuat 3 Parameter (fname yang berupa String, "is", dan age yang berupa Integer) dan membuat 3 argument - Liam, Jenny, Anja (diikuti dengan 3 urutan yang sama "is" 5,"is" 8, dan "is"31). Kita akan mendapatkan output dengan tata letak :

* Liam is 5
* Jenny is 8
* Anja is 31

Catatan : fname adalah nama lengkap, "is" adalah kata penambahan, dan age adalah umur.

Modul 1b adalah Return Values two parameters, dimana kita tidak boleh menggunakan kata kunci void untuk mengembalikan nilai. Kita bisa bisa menggunakan tipe data primitif (int, char, dll) untuk mengembalikan nilai. Ketika kita membuat 2 parameter ( x dan y yang berupa integer) diselingi operator aritmatika (penjumlahan) dan membuat 2 argument ("5" untuk parameter x, dan "3" untuk parameter y). Kita akan mendapatkan output : 8 (hasil dari penjumlahan x + y = 5 + 3).

Modul 1c adalah Return Values two parameters, sama dengan yang diatas, bedanya saya menambahkan 1 variabel ( z yang beupa Integer) untuk mempermudah mencari hasilnya.

Modul 1d adalah A Method with If...Else, Ketika kita membuat sebuah statements (If age is less than 18, print "access denied") akan diperoleh (Access denied - You are not old enough!) dan akan ada peralihan (If age is greater than 18, print "access granted") maka diperoleh (Access granted - You are old enough!). Karena kita membuat checkAge menjadi (20) maka akan diperoleh (Access granted - You are old enough!)

**Modul 2**

Modul 2 adalah Method Overloading, Ketika kita ingin membuat multiple method dengan nama yang sama, kita dapat menggunakan Method Overloading. Ketika kita membuat 2 method dengan nama yang sama, kita harus membuat parameter yang berbeda, kita menggunakan Integer pada method yang pertama dan double pada method yang kedua dan membuat plusMethod, hasilnya akan menjadi penjumlahan sesuai dengan tipe parameternya. Kita juga membuat variabel myNum1 untuk method pertama dan variabel myNum2 untuk method yang kedua. Untuk variabel yang pertama (Integer) isi dengan (8, 5) dan untuk variabel yang kedua (double) is dengan (4.3 , 6.26). Maka outputnya akan seperti ini :

* int: 13
* double: 10.559999999999999

Catatan : Beberapa method dapat memiliki nama yang sama selama jumlah dan / atau tipe parameternya berbeda.

**Modul 3**

Modul 3 adalah Java Classes and Objects, Ketika kita membuat kelas kita akan mengisinya dengan objek, dan objek tersebut terdiri dari nama kelas, nama objek dan kata kunci new. Kita akan mengisi kelas dengan variabel x berupa Integer dengan hasil 5, dan membuat objek baru dengan kata kunci new diikuti nama kelas. Maka outputnya akan menghasilkan 5.

**Modul 4**

Modul 4a adalah Accessing Attributes, sama dengan Modul 3, dan hasilnya pun juga sama.

Modul 4b adalah Override Existing Values, dimana kita akan menimpa suatu nilai yang telah ada, dengan nilai yang baru. Ketika kita membuat variabel x yang berupa Integer dengan hasil 10 dan ingin merubahnya ke 25. Buatlah sebuah objek baru didalam kelas dengan urutan nama objek, kata kunci new, dan nama kelas. Kemudian nama objek kita isi hasilnya dengan 25. Maka outputnya akan menjadi 25.

Modul 4c adalah Final Attributes, kita akan membuat suatu nilai yang telah ada, tidak dapat ditimpa dengan nilai yang baru. Caranya sama dengan Modul 4b, cukup tambahkan kata kunci final sebelum variabel x yang berupa integer yang mempunyai hasil 10.

**Modul 5**

Modul 5a adalah Static and Non-Static Attributes dan Method. Ketika membuat kelas, pastinya akan membutuhkan sebuah objek, hal ini tidak berlaku pada static method dimana tidak membutuhkan sebuah objek untuk menampilkan output, beda dengan non-static method (public) yang harus membutuhkan sebuah objek, dan jika tidak ada objek, akan menampilkan pesan error ketika menggunakan public method.

Modul 5b adalah Access Method with an Object, Kita membuat sebuah kelas baru bernama MyCar, dan tambahkan method fullThrottle() lalu isi ("The Car is going as fast as it can!") dan buat parameter maxSpeed yang berupa integer lalu isi ("Max speed is : " + maxSpeed). Kemudian buatlah Objek baru dengan diikuti kata kunci new dan nama kelas, lalu dibawahnya isi dengan nama objek.method fullThrottle() yang akan dipanggil, dan yang terakhir isi dibawahnya dengan nama objek.parameter maxSpeed() isi dengan nilai integer (mis. 200). Maka outputnya akan seperti ini :

* The car is going as fast as it can!
* Max speed is: 200

**Modul 6**

Modul 6a adalah Class Constructor, kita akan membuat Class Constructor dengan cara buatlah dahulu kelas baru dan isi dengan class atribut dengan x yang berupa integer, kemudian buatlah class constructor dibawahny dengan method public diikuti nama kelas, lalu dibawahnya isi dengan hasil x tadi dengan angka 5. Setelah selesai, lanjutkan dengan membuat objek baru. Maka outputnya akan menghasilkan angka 5.

Catatan : nama konstruktor harus cocok dengan nama kelas, dan tidak boleh memiliki kata kunci return (seperti void). Konstruktor akan dapat dipanggil saat objek dibuat.

Semua kelas memiliki konstruktor secara default, jika kita tidak membuat konstruktor kelas sendiri, Java membuat satu untuk kita. Namun, maka kita tidak dapat menetapkan nilai awal untuk atribut objek.

Modul 6b adalah Constructor Parameter, kita akan membuat Constructor Parameter dengan cara yang sama seperti Modul 6a, bedanya ada perubahan sedikit, pada class constructor yang telah kita buat tadi, angka 5 diganti dengan parameter y, dan angka 5 tadi dipindahkan di setelah urutan nama objek dan nama kelas (5).

Modul 6c adalah Multiple Constructor Parameter, ketika kita membuat Class Constructor tadi(Modul 6a dan 6b), kita hanya membuat satu constructor dan satu parameter. Kita akan membuat Multiple Constructor Parameter, buatlah 2 class atribut dengan modelYear yang berupa Integer dan modelName yang berupa String. Kemudian buatlah 2 class constructor modelYear dengan year dan modelName dengan name. Lalu pada bagian objek, isi dengan (1969, "Mustang") sesuai urutan class atribut dan constructor. Maka outputnya akan menghasikan 1969 Mustang.

**Modul 7**

Modul 7a adalah Final attributes, seperti yang kita tahu bahwa, penempatan atribut final akan menghasilkan pesan error pada output jika dijalankan, karena tidak dapat mengubah nilai atribut yang sudah ditentukan sejak awal kita membuatnya.

Modul 7b adalah abstract class, kita akan membuat 2 class yang berbeda (Person.java dan MyClass.java).

Person.java ~ buatlah kelas bernama Person.java dan isi dengan class atribut fname yang berupa String "John", age yang berupa Integer yaitu 24, dan abstract class (public abstract void study). Kemudian kita buat subclass Student, extends dari Person. Dengan menggunakan Public graduationYear sebagai Integer yaitu 2018, diikuti dibawahnya isi dengan public void study ("Studying all day long").

MyClass.java ~ buatlah kelas bernama MyClass.java dan buatlah object dari Student (subclass Person) dan dilanjutkan dibawahnya isi dengan ("Name : " memanggil dari fname),("Age : " memanggil dari age),("Graduation Year :" memanggil graduationYear). Lalu objek.study (untuk memanggil method abstract). Maka hasilnya akan seperti ini :

* Name: John
* Age: 24
* Graduation Year: 2018
* Studying all day long

**Modul 8**

Modul 8a adalah Encapsulation, yang perlu kita ketahui adalah variabel name bersifat private, dan didalam praktik modul 8a, menggunakan method Public. Maka akan menghasilkan pesan error saat dijalankan. Agar bisa dijalankan, gunakan method getName() dan setName() yang akan dijelaskan pada modul 8b.

Modul 8b adalah Method in Encapsulation, Seperti yang telah dibahas pada Modul 8a, kita harus menggunakan method getName() dan setName() agar bisa menjalanjkan method public dengan variabel name.

* get - mengembalikan nilai nama variabel.
* set - mengambil parameter (newName) dan menetapkannya ke variabel nama

Untuk penempatannya getName() yaitu setelah nama objek. dan sebelum variabel name, dan untuk penempatan setName() yaitu dibawah getName(), setelah nama objek. didalam tanda kurung ().

**Modul 9**

Modul 9a adalah Import a Class, ketika kita ingin mengimpor suatu class dari suatu paket java.util, kita akan akan menggunakan Scanner class. Untuk membuat Scanner class, buatlah sebuah objek baru dari sebuah class, dan kita isi dengan perintah yang akan muncul untuk mengisi manual pada output yang akan dijalankan nanti, coba kita isi "Enter Username".Kemudian gunakan method nextLine() untuk menampilkan apa yang telah kita ketik manual saat output dijalankan ( "Username is : " memanggil userName).

Modul 9b adalah Import a Package, kita akan mengimpor suatu package, kedalam kelas yang sudah kita buat dengan cara ketik java.util\*; maka akan otomatis mengimpor package java.util.

Modul 9c adalah User-defined Package, yang berarti membuat Paket sendiri, tidak mengimpor seperti java.util. Caranya, temukan terlebih dahulu folder mana yang berisi paket yang akan kita buat sendiri nanti ( ekstensi .java ). Setelah ketemu, gunakan kata kunci package kemudian nama folder kita. Setelah selesai, simpan dan compile dengan perintah javac YourPackage.java kemudian javac -d . YourPackage.java.

**Modul 10**

Modul 10a adalah Java Inheritance, digunakan ketika kita ingin mewariskan sesuatu. Dalam praktik Modul 10a, saya menggunakan contoh class Vehicle akan diwariskan ke Class Car, class Vehicle mempunyai brand "Ford" yang berbunyi "Tuut tuut!", dan class Car mempunyai modelName "Mustang". Dengan urutan honk, kemudian dibawahnya berupa brand tadi dilanjutkan dengan modelName, maka akan mempunyai output seperti ini : Tuut tuut! Ford Mustang.

Modul 10b adalah Final in Java Inheritance, seperti yang sudah kita bahas, jika kita tidak ingin memberikan / mewariskan sesuatu ke kelas lain, gunakan kata kunci final.

**Modul 11**

Modul 11 adalah Polymorphism, hampir sama dengan Inheritance, tetapi Polymorphism mempunyai cara yang berbeda seperti method animalSound() untuk diwariskan ke suatu hewan / binatang.

**Modul 12**

Modul 12a adalah Java Inner Class, digunakan ketika kita ingin saling mengakses dari luar maupun dalam, dengan cara membuat objek dari kelas luar dan juga membuat objek dari kelas dalam.

Modul 12b adalah Private Inner Class, Inner Class tidak seperti kelas yang biasanya, Inner Class tidak dapat dijalankan ketika menggunakan Private akan menjadikan output menampilkan pesan error.

Modul 12c adalah Static Inner Class, di Inner Class dapat dibuat tanpa menggunakan objek dengan method static. Tetapi tidak dapat diakses dari kelas luar.

Modul 12d adalah Access Outer Class from Inner Class, kita dapat mengakses kelas dalam dari kelas luar dengan cara masukkan Inner Class ke dalam Outer Class, dan juga public class untuk diisi. Berilah atribut x berupa Integer yang mempunyai nilai 10 untuk Outer Class, kemudian gunakan method return untuk dipanggil didalam public class lagi. Maka outputnya akan menghasilkan angka 10.

**Modul 13**

Modul 13 adalah Data Abstraction, cara menyembunyikan detail tertenu dan hanya menampilkan informasi yang penting - penting saja. Kita juga tidak dapat membuat objek kelas animal. Data abstraction juga menggunakan method Inheritance.

**Modul 14**

Modul 14a adalah Interfaces, Sinonim dari Abstraction, perbedaannya jika Abstraction menggunakan extends dari Inheritance, Interfaces menggunakan kata kunci implements. Dan di Interfaces juga dapat dibuat Multiple Interfaces yang akan dijelaskan pada Modul 14b.

Modul 14b adalah Multiple Interfaces, cukup pisahkan dengan koma.

**Modul 15**

Modul 15a adalah Enum, terdaftar secara khusus, itulah arti dari Enum. Untuk membuat enum, cukup ketik kata kunci enum dan tambahkan koma, saat memisahkan konstanta. Kita juga dapat mengakses konstanta enum dengan titik (.)

Modul 15b adalah Enum inside a Class, enum juga dapat dimasukkan kedalam kelas, dengan cara buatlah kelas baru dan tambahkan kata kunci enum, dan jangan lupa memisahkan konstanta dengan koma.

Modul 15c adalah Enum in a Switch Statement, caranya sama seperti Modul 15b, cukup tambahkan kata kunci enum dan pisahkan konstanta dengan koma.

**Modul 16**

Modul 16a adalah Java User Input, kita dipertemukan lagi dengan java.util.Scanner, java.util.Scanner kembali digunakan pada java user input, buatlah objek dan gunakan method dari Scanner class, contohnya nextLine() digunakan untuk membaca tipe data String. Dan saat dijalankan, kita akan diberi perintah untuk memasukkan input yang sesuai dengan perintah yang tertulis.

Modul 16b adalah Different Method Java User Input, sama dengan modul 16a, hanya saja menambahkan berbagai varian method seperti nextInt(), nextDouble(), dll.

**Modul 17**

Modul 17a adalah Display Current Date, digunakan untuk menampilkan Tanggal-Bulan-Tahun saat ini sekarang juga.

Modul 17b adalah Display Current Time, digunakan untuk menampilkan Waktu (Jam, Menit, Detik, Mili-detik) saat ini sekarang juga.

Modul 17c adalah Display Current Date and Time, digunakan untuk menampilkan Tanggal dan Waktu secara bersamaan saat ini sekarang juga.

Modul 17d adalah Formatting Date and Time, digunakan untuk memformat tata letak Tanggal dan Waktu.

**Modul 18**

Modul 18 adalah ArrayList, berbeda dengan array biasa, ArrayList memiliki sejumlah operasi yang lebih lengkap dan mudah digunakan dibandingkan dengan array biasa. ArrayList merupakan collection yang menjadi bagian dari java.util. Seperti biasa, ArrayList dapat menambah data baru secara dinamis tanpa harus menentukan ukurannya di awal. Berbagai operasi dapat Anda lakukan terhadap ArrayList seperti berikut:

* size(), untuk mencari panjang *ArrayList*
* add(), untuk menambah elemen baru
* get(), untuk mengambil elemen pada indeks tertentu
* isEmpty(), untuk memeriksa apakah *ArrayList* kosong atau tidak
* indexOf(), untuk mengetahui indeks dari suatu nilai
* contains(), untuk memeriksa apakah suatu nilai ada dalam *ArrayList*
* set(), untuk menimpa nilai pada indeks tertentu
* remove(), untuk menghapus nilai pada indeks tertentu

**Modul 19**

Modul 19 adalah HashMap, sebuah collection yang berbasis hash di Java. HashMap mengimplimentasikan interface java.util.Map. Namun, HashMap, menyimpan item dalam pasangan "key / value", dan Anda dapat mengaksesnya dengan indeks jenis lain (mis. String). HashMap juga memiliki performa yang constant time. Dan yang terpenting HashMap unggul dalam performanya, karena HashMap tidak tersinkronisasi. Satu objek digunakan sebagai key (indeks) ke objek lain (value). Ini dapat menyimpan berbagai jenis, Key string dan value Integer, atau tipe yang sama, seperti Key string dan value String.

**Modul 20**

Modul 20 adalah Wrapper Classes, Wrapper Classes memberitahu kita bagaimana caranya menggunakan tipe data primitif (int, boolean, dll.) menjadi objek.Ada kalanya dimana kita harus menggunakan wrapper classes, mis. saat digunakan di objek collection, tipe data primitif tidak dapat digunakan di ArrayList.

**Modul 21**

Modul 21 adalah Exceptions. Seperti artinya, ketika kesalahan terjadi, Java biasanya akan berhenti dan menghasilkan pesan error. Java akan menunjukkan Exception (bukan Error). Ada 2 macam Exception, Try and Catch.

* Try ~ Statements memungkinkan kita untuk menentukan blok kode yang akan diuji untuk error saat sedang dieksekusi.
* Catch ~ Statements memungkinkan kita untuk menentukan blok kode yang akan dieksekusi, jika error terjadi di blok try.
* Finally ~ Statements memungkinkan kita menjalankan kode, setelah try ... catch, tidak memunculkan hasilnya.
* Throw ~ Statements memungkinkan kita untuk membuat custom error.

**BAB V PENUTUP**

**KESIMPULAN**

Java adalah bahasa pemrograman tingkat tinggi yang sangat dinamis yang memungkinkan kita mengembangkan aplikasi web, Android, dll. Java telah menjadi bahasa yang sangat berlaku untuk Internet dan pemrograman sistem terdistribusi secara umum. Tetapi bidang aplikasinya bukan semata-mata Internet, salah satu kelebihan Java adalah dimaksudkan untuk sepenuhnya bebas dari perangkat keras ada mesin virtual Java untuk berbagai jenis komputer.

**SARAN**

Dari Laporan yang saya buat, diharapkan Laporan ini dapat bermanfaat bagi pembaca dalam memahami tentang Bahasa Pemrograman Java. Selain itu saya juga menyarankan untuk menerapkan apa yang baik dari Laporan ini dan juga mengingatkan saya apa yang dianggap pembaca kurang baik dari Laporan ini. Sebagai penyusun, saya akui tidak terlepas dari kesalahan dan keterbatasan. Karena itu penyusun mengharapkan kritik dan saran yang membangun untuk perbaikan penulisan Laporan selanjutnya. Semoga Laporan ini dapat bermanfaat bagi para pembaca.

Pasuruan, 5 Januari 2020

Thita Mar’atus Solikha

**DAFTAR PUSTAKA**

<https://www.w3schools.com/java/java_methods.asp>

<https://www.w3schools.com/java/java_methods_param.asp>

<https://www.w3schools.com/java/java_methods_overloading.asp>

[https://www.w3schools.com/java/java\_oop.asp](https://www.w3schools.com/java/java_oop.asp" \o "https://www.w3schools.com/java/java_oop.asp)

<https://www.w3schools.com/java/java_classes.asp>

<https://www.w3schools.com/java/java_class_attributes.asp>

<https://www.w3schools.com/java/java_class_methods.asp>

<https://www.w3schools.com/java/java_constructors.asp>

<https://www.w3schools.com/java/java_modifiers.asp>

<https://www.w3schools.com/java/java_encapsulation.asp>

<https://www.w3schools.com/java/java_packages.asp>

<https://www.w3schools.com/java/java_inheritance.asp>

<https://www.w3schools.com/java/java_polymorphism.asp>

<https://www.w3schools.com/java/java_inner_classes.asp>

<https://www.w3schools.com/java/java_abstract.asp>

<https://www.w3schools.com/java/java_interface.asp>

<https://www.w3schools.com/java/java_enums.asp>

<https://www.w3schools.com/java/java_user_input.asp>

<https://www.w3schools.com/java/java_date.asp>

<https://www.w3schools.com/java/java_arraylist.asp>

<https://www.w3schools.com/java/java_hashmap.asp>

<https://www.w3schools.com/java/java_wrapper_classes.asp>

<https://www.w3schools.com/java/java_try_catch.asp>

<https://en.wikipedia.org/wiki/Java_(programming_language)>

<https://en.wikipedia.org/wiki/Java_(software_platform)>

<https://en.wikipedia.org/wiki/Java>