

OBJECT-ORIENTED PROGRAMMING MODULE

LABORATORIUM PRODASE



MODUL PRAKTIKUM

OBJECT-ORIENTED PROGRAMMING

Tim Penyusun

Yusril Maulidan Raji (YRL)	116110025	Faqih Putra Kusumawijaya (FAQ)	116110016
Komang Aditya Respa. P (RPA)	116110029	Chintamy Christini (CHC)	116110033
Ainu Faisal Pambudy (NOE)	116114159	M. Edwin Baihaqi (WIN)	116110057
Renantia Indriani (RNT)	116114164	Dewansyah Adi Saputra (DEW)	116111151
Fathimah Muthi Luthfiyah (FML)	116110061	Ghoziyah Haitan Rachman(GHR)	116112155
Yusuf Rahmadi (RMD)	116110009	Charlie Sugiarto (VCS)	116110047
Rizki Dwi Kurnia D (DKD)	116114165	Putri Myke Wahyuni (MYK)	116110037
M. Akbar Satria (AKB)	116110006	Ekky Novriza Alam (EKY)	116112156
Ajeng Citra Rizkyanur (AJG)	116110009		



LABORATORIUM PROGRAMMING DAN DATABASE

PROGRAM STUDI SISTEM INFORMASI

FAKULTAS REKAYASA INDUSTRI

DEPARTMEN TELKOM ENGINEERING SCHOOL

2013 / 2014

Class, Object, Constructor dan Method

TUJUAN PRAKTIKUM

1. Praktikan dapat memahami dan menerapkan pengertian object dan class dalam java
2. Praktikan dapat memahami pengertian dan penggunaan Constructor dalam java
3. Praktikan dapat memahami konsep dan mengimplementasikan penggunaan method pada java

LANDASAN TEORI

A. Class dan Object

Class adalah cetak biru (rancangan) atau prototype atau template dari objek. Kita bisa membuat banyak objek dari satu macam class. Class mendefinisikan sebuah tipe dari objek.

Di dalam class kita dapat mendeklarasikan variabel dan menciptakan objek (instansiasi). Sebuah class mempunyai anggota yang terdiri dari atribut dan method.

Atribut adalah semua field identitas yang kita berikan pada suatu class, misal class manusia memiliki field atribut berupa nama dan umur. Method dapat kita artikan sebagai semua fungsi ataupun prosedur yang merupakan perilaku (behaviour) dari suatu class.

Bagian-bagian dari sebuah Class secara umum penulisan class terdiri atas 2 bagian yakni:

1. Class Declaration

Bentuk Umum :

```
[modifier] class <nama_kelas>
{
    <class body>
}
```

[modifier] adalah pengaturan level akses terhadap kelas tersebut. Dengan kata lain, modifier ini akan menentukan sejauh mana kelas ini dapat digunakan oleh kelas atau package lainnya. Adapun macam-macam modifier ialah :

- kosong / default / *not specified*

Kelas tersebut dapat diakses oleh kelas lain dalam satu package.

- *public*

Kelas tersebut dapat dipakai dimanapun, maupun kelas lain atau package lain.

- *Private*

Kelas tersebut tidak dapat diakses oleh kelas manapun.

2. **Class Body**

Class Body merupakan bagian dari kelas yang mendeklarasikan kode program java. Class Body tersusun atas:

- a. Konstruktor
- b. *Variable Instance* (Atribut)
- c. *Method* (dikenal juga sebagai function atau def)

Untuk dapat menggunakan kelas yang telah didefinisikan, anda harus membuat sebuah objek dari kelas tersebut (*instance class*), dengan *syntax*:

```
NamaKelas namaObjek = new NamaKelas ( [parameter] );
```

Contoh:

```
Hitungluas segitiga = new Hitungluas();
```

3. **Instance Variables (Atribut)**

Suatu objek dapat dibedakan berdasarkan sifat (behavior) yang berbeda. objek juga dapat dibedakan berdasarkan atributnya. Misalnya burung dapat dibedakan berdasarkan suara kicauan, warna bulu, bentuk tubuh, dsb. . Dalam bahasa lain dikenal juga sebagai *property* yang mana merupakan ciri-ciri dari sebuah objek.

Atribut yang membedakan suatu *instance* objek burung yang satu dengan yang lainnya disebut **instance variable**.

Bentuk Umum :

```
[modifier] <tipe_data> <nama_variabel> = [nilai_default];
```

Contoh :

```
public double tinggi;  
private int berat = 70;
```

Modifier untuk atribut, yaitu public, protected, private. Penjelasan modifier atribut serupa dengan penjelasan modifier pada kelas.

Adapun perbedaan *local* dan *instance variable* adalah :

1. *Instance variable* dideklarasikan di dalam kelas tetapi tidak di dalam method.

```
class segitiga {  
    double tinggi = 15.2; // ini adalah variabel instance  
    String jenis; // ini adalah variabel instance  
    int tambah() {  
        return 3;  
    }  
}
```

2. *Local variable* dideklarasikan di dalam method.

```
int tambah() {  
    int total = tinggi * 2; // total adalah variabel local  
    return total;  
}
```

Object adalah *instance* dari *class*. Jika *class* secara umum merepresentasikan (template) sebuah *object*, sebuah *instance* adalah representasi nyata dari *class* itu sendiri.

Contoh : Dari *class* Fruit kita dapat membuat object Mangga, Pisang, Apel dan lain lain.

4. Membuat object

Untuk membuat object, kita menggunakan perintah *new* dengan sebuah nama *class* yang akan dibuat sebagai *instance* dari *class* tersebut. Contohnya:

```
String str = new String();  
Random r = new Random();  
Pegawai p2 = new Pegawai();  
Date hari = new Date();
```

hari adalah object reference dari class Date yang akan digunakan untuk mengakses class Date. Sedangkan operator *new* adalah operator yang akan menghasilkan hari sebagai reference ke instance dari class Date().

B. Constructor

Tipe khusus method yang digunakan untuk menginstansiasi atau menciptakan sebuah objek. Nama constructor = nama kelas. Constructor TIDAK BISA mengembalikan nilai. Tanpa membuat constructor secara eksplisit-pun, Java akan menambahkan constructor default secara implisit. Tetapi jika kita sudah mendefinisikan minimal sebuah constructor, maka Java tidak akan menambah constructor default. Constructor default tidak punya parameter. Constructor bisa digunakan untuk membangun suatu objek, langsung mengeset atribut-atributnya. Konstruktor seperti ini harus memiliki parameter masukkan untuk mengeset nilai atribut. Access Modifier constructor selayaknya adalah public, karena constructor akan diakses di luar kelasnya.

Cara panggil constructor adalah dengan menambah keyword “new”. Keyword new dalam deklarasi ini artinya kita mengalokasikan pada memory sekian blok memory untuk menampung objek yang baru kita buat.

```
[modifier] namaclass (parameter) {  
    Body constructor;  
}
```

Contoh kode program

```
package DemoManusia;  
  
public class Lagu {  
    private String band;  
    private String judul;  
    public void IsiParam(String judul,String band) {  
        this.judul = judul;  
        this.band = band;  
    }  
    public Lagu() {  
        judul = "null";  
        band = "null";  
    }  
  
    public void cetakKeLayar() {  
        System.out.println("Judul : " + judul + "\nBand : " + band);  
    }  
}
```

```
package DemoManusia;

public class DemoLagu {
    public static void main(String[] args) {
        Lagu song = new Lagu();
        song.cetakKeLayar();
    }
}
```

Output :

Judul : null
Band : null

Pada kode program di atas terdapat konstruktor yang akan mengisi atribut nama dan band dengan nilai = "null". Jadi saat class Lagu di instansiasi maka nilai atribut nama dan band diisi dengan "null".

C. Method

Sebuah **method** adalah bagian-bagian kode yang dapat dipanggil oleh kelas, badan program atau method lainnya untuk menjalankan fungsi yang spesifik di dalam kelas. Secara umum method dalam java adalah sebuah fungsi.

Berikut adalah karakteristik dari method :

1. Dapat mengembalikan / melaporkan nilai balikkan (*return value*) atau tidak (*void*)
2. Dapat diterima beberapa parameter yang dibutuhkan atau tidak ada parameter sama sekali. Parameter bisa juga disebut sebagai argumen dari fungsi. Parameter berguna sebagai nilai masukkan yang hendak diolah oleh fungsi.
3. Setelah method telah selesai dieksekusi, dia akan kembali pada method yang memanggilnya.

Menggunakan method memiliki beberapa keuntungan, yaitu :

1. Method membuat program lebih mudah dibaca dan mudah untuk dipelihara / di-maintain.
2. Method membuat proses pengembangan dan perawatan (maintenance) menjadi lebih cepat.
3. Method merupakan dasar untuk melakukan membuat software yang reusable.
4. Method memungkinkan obyek-obyek yang berbeda untuk berkomunikasi dan untuk mendistribusikan beban kerja yang dipikul oleh program.

1. Deklarasi Sebuah Method

Method terdiri atas dua bagian yakni :

1. Method declaration
2. Method Body

Method dapat digambarkan sebagai sifat (*behavior*) dari suatu class. Untuk mendefinisikan method pada dalam *class* digunakan syntax :

```
[modifiers] return_type method_identifier ([arguments]) {  
    method_code_block;  
}
```

di mana :

- a. [modifiers] merepresentasikan kata kunci pada teknologi Java yang memodifikasi cara-cara penggunaan method. Contoh : public, protected, private, static, final.
- b. return_type adalah tipe nilai yang akan dikembalikan oleh method yang akan digunakan pada bagian lain dari program. Return_type pada method sama dengan tipe data pada variabel. Return_type dapat merupakan tipe data primitif maupun tipe data referensi.
- c. method_identifier adalah nama method.
- d. ([arguments]), merepresentasikan sebuah daftar variabel yang nilainya dilewatkan / dimasukkan ke method untuk digunakan oleh method. Bagian ini dapat tidak diisi, dan dapat pula diisi dengan banyak variabel.
- e. method_code_block, adalah rangkaian pernyataan / statements yang dibawa oleh method.

Contoh :

```
public int Perkalian ( int y;int z )  
{  
    return y * z ;  
}
```

2. Modifier Pada Method

Modifier menentukan level pengaksesan sebuah method. Hal ini menentukan apakah sebuah method biasa diakses oleh objek lain, objek anak, objek dalam satu

paket atau tidak dapat diakses oleh suatu object sama sekali berikut adalah beberapa jenis level access:

- **Public**
Atribut ini menunjukkan bahwa fungsi/method dapat diakses oleh kelas lain.
- **Private**
Atribut ini menunjukkan bahwa fungsi atau method tidak dapat diakses oleh kelas lain
- **Protected**
Atribut ini menunjukkan bahwa fungsi atau method bisa diakses oleh kelas lain dalam satu paket dan hanya kelas lain yang merupakan subclass nya pada paket yang berbeda.
- **Tanpa modifier**
Atribut ini menunjukkan bahwa method dapat diakses oleh kelas lain dalam paket yang sama.

```
public class MyClass {  
    private void privateMethod() {  
        System.out.println("I am private method");  
    }  
  
    void packageMethod() {  
        System.out.println("I am package method");  
    }  
  
    protected void protectedMethod() {  
        System.out.println("I am protected method");  
    }  
  
    public void publicMethod() {  
        System.out.println("I am public method");  
    }  
}
```

3. Method Tanpa Nilai Balikan

Method ini merupakan method yang tidak mengembalikan nilai. Maka dari itu, kita harus mengganti tipe kembalian dengan kata kunci *void*. Berikut ini kode program yang dimaksud:

```
class persegi {  
    double sisi;  
    // mendefinisikan method void (tidak mengembalikan nilai)  
    void cetakluas() {  
        System.out.println("Luas Persegi = " + (sisi * sisi));  
    }  
}
```

```

class persegiApp {
    public static void main(String[] args) {
        // instansiasi objek
        persegi p1 = new persegi();
        persegi p2 = new persegi();
        // mengisi data untuk masing-masing objek
        p1.sisi = 10;
        p2.sisi = 5;
        // memanggil method cetakLuas() untuk masing-masing objek
        p1.cetakluas();
        p2.cetakluas();
    }
}

```

Output :

Luas Persegi = 100.0

Luas Persegi = 25.0

4. Method Dengan Nilai Balikan

Di sini, kita akan memodifikasi program sebelumnya dengan mengganti method cetakLuas() menjadi method hitungLuas () yang akan mengembalikan nilai dengan tipe double. Berikut ini kode program yang dimaksud:

```

class Persegi {
    double sisi;
    // mendefinisikan method yang mengembalikan nilai
    double hitungluas() {
        return (sisi*sisi);
    }
}

class PersegiApp {
    public static void main(String[] args) {
        // instansiasi objek
        Persegi p1 = new Persegi();
        Persegi p2 = new Persegi();
        // mengisi data untuk masing-masing objek
        p1.sisi = 10;
        p2.sisi = 5;
        // memanggil method hitungLuas() untuk masing-masing objek
        System.out.println("Luas Persegi = " + p1.hitungluas());
        System.out.println("Luas Persegi = " + p2.hitungluas());
    }
}

```

Output :

Luas Persegi = 100.0

Luas Persegi = 25.0

5. Parameter

Dengan adanya parameter, sebuah method dapat bersifat dinamis dan general. Artinya, method tersebut dapat mengembalikan nilai yang beragam sesuai dengan nilai parameter yang dilewatkan. Terdapat dua istilah yang perlu anda ketahui dalam bekerja dengan method, yaitu parameter dan argumen. Parameter adalah variabel yang didefinisikan pada saat method dibuat, sedangkan argumen adalah nilai yang digunakan pada saat pemanggilan method. Dalam referensi lain, ada juga yang menyebut parameter sebagai parameter formal dan argumen sebagai parameter aktual. Perhatikan kembali definisi method berikut :

```
int luasPersegiPanjang(int panjang, int lebar) {  
    return panjang * lebar;  
}
```

Disini, variabel panjang dan lebar disebut parameter.

```
Luas1 = luasPersegiPanjang(10, 5);
```

Adapun pada statemen diatas, nilai 10 dan 5 disebut argumen.

Sekarang, kita akan mengimplementasikan konsep di atas ke dalam kelas Persegi. Di sini data sisi akan kita isikan melalui sebuah method. Berikut ini kode program yang dimaksud:

```
class Persegi {  
    double sisi;  
    // mendefinisikan method mengembalikan nilai  
    double hitungluas() {  
        return (sisi*sisi);  
    }  
    void isiSisi(int s) {  
        sisi = s;  
    }  
}
```

```
class PersegiApp {  
    public static void main(String[] args) {  
        // instansiasi objek  
        Persegi p1 = new Persegi();  
        Persegi p2 = new Persegi();  
    }  
}
```

```

        // mengisi data untuk masing-masing objek
        p1.isiSisi(10);
        p2.isiSisi(5);
        // memanggil method hitungLuas() untuk masing-masing objek
        System.out.println("Luas Persegi = " + p1.hitungluas());
        System.out.println("Luas Persegi = " + p2.hitungluas());
    }
}

```

Output :

Luas Persegi = 100.0
 Luas Persegi = 25.0

Bagaimana jika bagian lain dari program ingin tahu juga nilai luas itu tetapi tidak ingin menampilkannya (mencetaknya). Apabila terdapat suatu fungsi yang tidak menghasilkan suatu nilai apapun maka bagian return type ini diganti dengan void .

Contoh penggunaan return:

```

class Persegi {
    double sisi;
    // mendefinisikan method mengembalikan nilai
    double hitungluas(int sisi) {
        return (sisi*sisi);
    }
}

class PersegiApp {
    public static void main(String[] args) {
        // instansiasi objek
        Persegi p1 = new Persegi();
        Scanner input = new Scanner(System.in);
        //fungsi untuk menginputkan suatu nilai
        System.out.print("Masukan sisi persegi : ");
        int a = input.nextInt();
        // memanggil method hitungLuas()
        System.out.println("Luas Persegi = " + p1.hitungluas(a));
    }
}

```

Output :

Masukan sisi persegi : 10
 Luas Persegi = 100.0

6. Method Static

Sebuah method static dapat diakses tanpa harus melakukan instantiasi terlebih dahulu. Pemanggilan method static dilakukan dengan format :

Nama_kelas.nama_method();

Nama_kelas diberikan bila method tersebut dipanggil dari kelas yang berbeda.

Contoh penggunaan return:

```
class Bangun {
    static void luasPesegiPanjang(){
        int panjang = 4;
        int lebar = 2;
        System.out.println("Luas Persegi Panjang = " + (panjang*lebar));
    }
}

class PersegiPanjang {
    public static void main(String[] args) {
        Bangun.luasPesegiPanjang(); // pemanggilan method statik
    }
}
```

Output :

Luas Persegi Panjang = 8

JURNAL MODUL 1

1. Buatlah program menampilkan daftar member suatu perusahaan rental mobil menggunakan method dengan parameter sebanyak 6 parameter.

Contoh :

Nama Depan : Ucup
Nama Belakang : Rahmadi
ID Member : 11061100xx
Alamat : Sukabirus
Nomor Telepon : 08211988xxx
Tahun bergabung : 2011

Ucup Rahmadi, 11061100xx dengan alamat di Sukabirus dan nomor telepon 08211988xxx bergabung menjadi member di Prodase Rent Car sejak tahun 2011

2. Buatlah sebuah program untuk menghitung denda member yang telat mengembalikan mobil di Prodase Rent Car. Lama maksimal peminjaman adalah 10 hari. per hari akan di denda Rp 250000 asumsikan 1 bulan itu 30 hari.

Contoh keluaran program :

Nama = Dewansyah
ID Member = 11061100xx
Tanggal Pinjam (hh-bb-tttt) = 12-01-2014
Tanggal Kembali (hh-bb-tttt) = 13-01-2014
Lama Peminjaman = 11 hari
Telat = 1 hari
Denda = 250000

Inheritance, Abstract Class, Interface

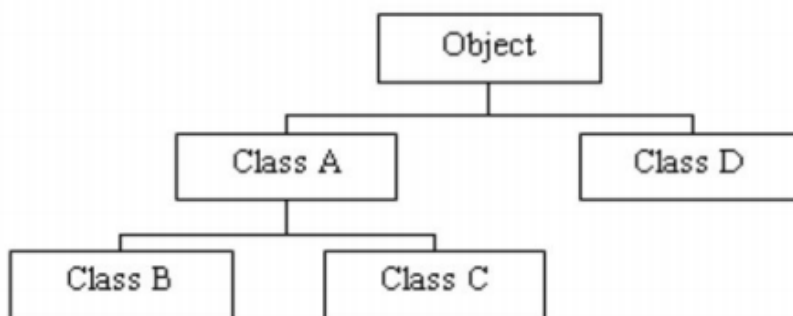
TUJUAN PRAKTIKUM

1. Praktikan dapat memahami bagaimana suatu class dapat mewariskan sifat dari class yang sudah ada.
2. Praktikan mampu mendefinisikan superclass dan subclass.
3. Praktikan dapat mengenal dan memahami konsep abstract class , abstract method, dan mampu menerapkannya dalam konsep OOP.
4. Praktikan mampu memahami konsep dan penggunaan interface.
5. Praktikan mampu mengimplementasikan inheritance, abstract class dan interface.

LANDASAN TEORI

A. Inheritance

Dalam bagian ini, kita akan membicarakan bagaimana suatu class dapat mewariskan sifat dari class yang sudah ada. Class ini dinamakan subclass dan induk class dinamakan superclass. Dalam Java, semua class, termasuk class yang membangun Java API, adalah subclasses dari superclass Object. Contoh hirarki class diperlihatkan di bawah ini. Beberapa class di atas class utama dalam hirarki class dikenal sebagai superclass. Sementara beberapa class di bawah class pokok dalam hirarki class dikenal sebagai subclass dari class tersebut.



Class hierarchy in Java.

Pewarisan adalah keuntungan besar dalam pemrograman berbasis object karena suatu sifat atau method didefinisikan dalam superclass, sifat ini secara otomatis diwariskan dari semua subclasses. Jadi, Anda dapat menuliskan kode method hanya sekali dan mereka dapat digunakan oleh semua subclass. Subclass hanya butuh mengimplementasikan perbedaannya sendiri dan induknya.

Mendefinisikan Superclass dan Subclass

Untuk memperoleh suatu class, kita menggunakan kata kunci `extends`. Untuk mengilustrasikan ini, kita akan membuat contoh class induk. Dimisalkan kita mempunyai class induk yang dinamakan `Person`.

```
public class Person {

    private String name;
    private String address;

    /*
     * Default Constructor
     */

    public Person (){
        System.out.println("Inside Peson:Constructor");
        name = "";
        address = "";
    }

    /*
     * Constructor dengan 2 parameter
     */

    public Person (String name, String address){
        this.name = name;
        this.address = address;
    }

    /*
     * Method accessor
     */

    public String getName(){
        return name;
    }

    public String getAddress(){
        return address;
    }

    public void setName(String name){
        this.name = name;
    }
}
```



```
        public void setAddress(String address){
            this.address = address;
        }
    }
```

Perhatikan bahwa atribut name dan address dideklarasikan sebagai private. Alasannya kita melakukan ini yaitu, kita inginkan atribut-atribut ini tidak dapat diakses langsung, melainkan diakses menggunakan method accessor. Catatan bahwa semua properti dari superclass yang dideklarasikan sebagai public, protected dan default dapat diakses oleh subclasses-nya. Sekarang, kita ingin membuat class lain bernama Student. Karena Student juga sebagai Person, kita putuskan hanya meng-extend class Person, sehingga kita dapat mewariskan semua properti dan method dari setiap class Person yang ada. Untuk melakukan ini, kita tulis :

```
public class Student extends Person{

    public Student(){
        System.out.println("Inside Student:Constructor");
        setName("David");
        setAddress("Jakarta");
        System.out.println("Name : " + getName() + "\nAddress : " + getAddress());
    }

}
```

Ketika object Student di-instantiate, default constructor dari superclass secara mutlak meminta untuk melakukan inisialisasi yang seharusnya. Setelah itu, pernyataan di dalam subclass dieksekusi. Untuk mengilustrasikannya, perhatikan kode berikut :

```
public class main {

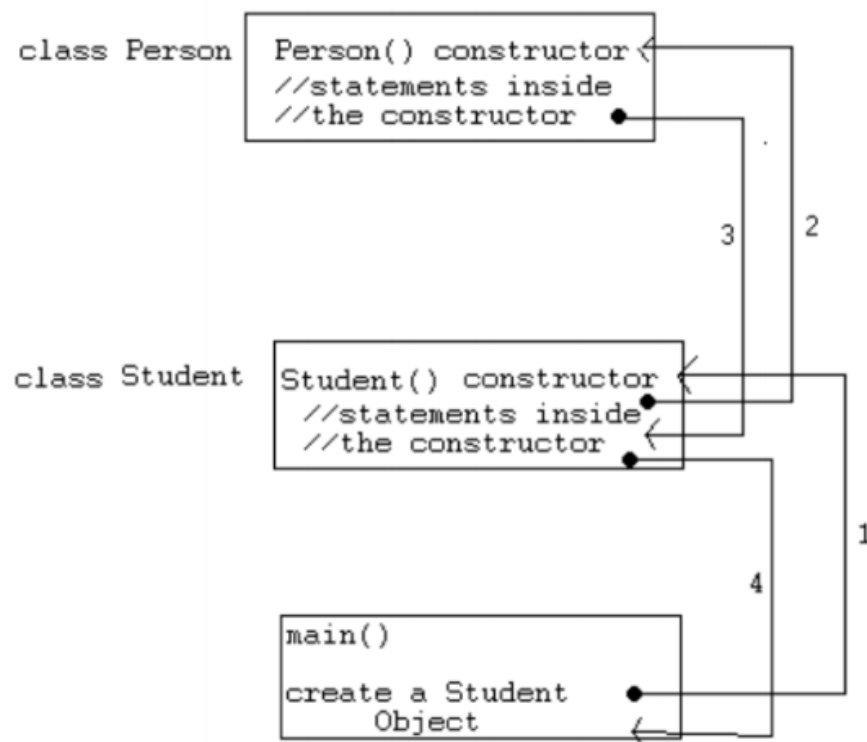
    public static void main(String[] args) {
        Student s = new Student();
    }

}
```

Dalam kode ini, kita membuat sebuah object dari class Student. Keluaran dari program adalah :

```
Inside Peson:Constructor
Inside Student:Constructor
Name : David
Address : Jakarta
```

Alur program ditunjukkan sebagai berikut :



Kata Kunci Super

Subclass juga dapat memanggil constructor secara explicit dari superclass terdekat. Hal ini dilakukan dengan pemanggil constructor super. Pemanggil constructor super dalam constructor dari subclass akan menghasilkan eksekusi dari superclass constructor yang bersangkutan, berdasar dari argumen sebelumnya.

Sebagai contoh, pada contoh class sebelumnya. Person dan Student, kita tunjukkan contoh dari pemanggil constructor super. Kode ini memanggil constructor kedua dari superclass terdekat (yaitu Person) dan mengeksekusinya.

```
public class Student extends Person{

    public Student(){
        super();
        System.out.println("Inside Student:Constructor");
        setName("David");
        setAddress("Jakarta");
        System.out.println("Name : " + getName() + "\nAddress : " + getAddress());
    }

}
```

B. Abstract Class

Sebenarnya class abstract tidak berbeda dengan class-class lainnya yaitu memiliki class member (method dan variable). Sebuah class adalah class abstrak jika salah satu methodnya dideklarasikan abstrak. Method abstrak adalah method yang tidak memiliki implementasi. Method abstrak tidak boleh memiliki badan program, badan program metode ini dapat diimplementasikan pada kelas turunannya.

Hal-hal yang perlu diperhatikan mengenai abstract class :

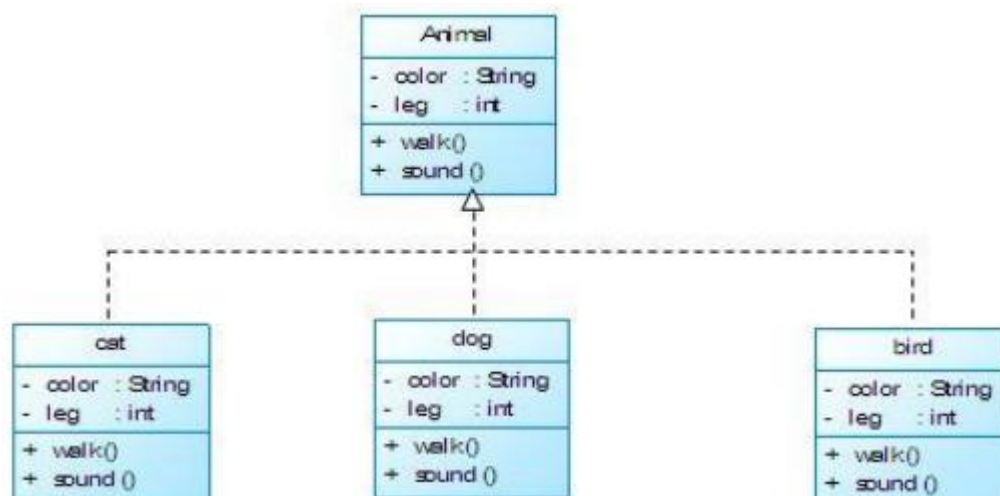
- Abstract class tidak bisa digunakan untuk membuat objectnya.
- Object hanya bisa dibuat dalam nonabstract class atau concrete class.
- Suatu abstract class harus diturunkan pada subclassnya.

Bila subclass yang diturunkan dari abstract class tidak mengimplementasikan isi semua method abstrak parent class, maka subclass tersebut harus tetap dideklarasikan secara abstrak. Deklarasi method abstrak pada subclass tersebut boleh tidak dituliskan kembali.

Berikut ini adalah ciri-ciri dari abstract class :

- Terletak pada posisi hirarki tertinggi dari hirarki kelas
- Hanya berisi variable-variabel umum dan deskripsi method tanpa detail implementasi dari abstract method yang ada di super classnya
- Mendefinisikan segala tipe action yang mungkin dengan object semua subclass dari class
- Tidak dapat diinstantiasi

Tujuan membuat kelas abstrak adalah agar satu kelas lain dapat memperluasnya (extend) dengan jalan menjadi subclass darinya. Mengenai konsep abstract class dapat diilustrasikan seperti gambar di bawah ini :



Jadi, dalam pengimplementasian abstract class, dalam satu project terdiri dari abstract class sebagai hierarki, subclass untuk pengimplementasian, dan class yang digunakan untuk output program. Contohnya , pemdeklarasian kelas Animal yang abstrak, di mana di dalamnya dideklarasikan dua variabel yaitu legs dan color. Selain itu juga dideklarasikan dua buah method, yaitu walk(), dan sound(). Method di dalam class animal tidak didefinisikan implementasi method, dengan harapan bahwa obyek turunan hewan yang spesifik, contohnya Cat.

1. Kode pada hierarki class atau abstract class

```
public abstract class Animal {  
    private String color;  
    private int leg;  
  
    public String getColor(){  
        return color;  
    }  
  
    public void setColor(String color){  
        this.color = color;  
    }  
  
    public int getLeg(){  
        return leg;  
    }  
  
    public void setLeg(int leg){  
        this.leg = leg;  
    }  
  
    /*  
    * Method abstrak  
    */  
  
    public abstract void walk();  
    public abstract void sound();  
}
```

2. Implementasi abstract class

```
public class bird extends Animal{  
  
    public bird (String color, int leg){  
        setColor(color);  
        setLeg(leg);  
    }  
}
```

```

        System.out.println("Color : " + getColor());
        System.out.println("Leg : " + getLeg());
    }

    /*
     * implement abstact method
     */

    @Override
    public void walk() {
        System.out.println("Bird have 2 legs");
    }

    @Override
    public void sound() {
        System.out.println("ciiit ciiit");
    }
}

```

3. Class main, untuk melakukan output

```

public class print {

    public static void main(String[] args) {
        bird b = new bird("white", 2);
        b.walk();
        b.sound();
    }
}

```

4. Output

```

Color : white
Leg : 2
Bird have 2 legs
ciiit ciiit

```

C. Interface

Interface adalah sebuah file java yang hanya berisi method kosong. Method kosong pada interface ditujukan untuk menjadi behavior atau sifat wajib dari class yang mengimplementasikannya.

Contoh interface :

```

public interface Car {

    public void startEngine();
    public void stopEngine();
}

```

```
}
```

Tidak seperti class biasa yang mengharuskan semua method untuk memiliki logic didalamnya, pada interface hanya diperbolehkan untuk memiliki nama method beserta tipe kembaliannya. Hal ini dimaksudkan agar object yang mengimplementasikan interface tersebut dapat berpolimorfisme dengan sifat yang sama mengikuti method minimal yang tertulis pada interface.

Pada contoh diatas dimaksudkan agar semua class yang mengimplementasikan interface memiliki method startEngine() dan stopEngine() yang tujuannya dapat membandingkan antara 2 buah object. Walaupun implementasi dari isi method tersebut berbeda-beda, namun kembalian (return type) dari method tersebut tetap sama sehingga memudahkan dalam proses pembuatan program.

Contoh class yang mengimplementasikan interface dapat dilihat pada class dibawah ini.

```
public class Drive implements Car{

    @Override
    public void startEngine() {
        System.out.println("Engine start. You can drive");
    }

    @Override
    public void stopEngine() {
        System.out.println("Engine stop. You can't drive");
    }

}
```

Pada class diatas, apabila method startEngine() dan stopEngine() dihapus, maka akan terjadi Runtime Exception karena class Drive harus memiliki semua method yang dimiliki interface-nya. Class yang mengimplementasikan sebuah interface bisa saja memiliki method tambahan, hal ini tetap diperbolehkan selama method yang berada pada interface telah dimiliki.

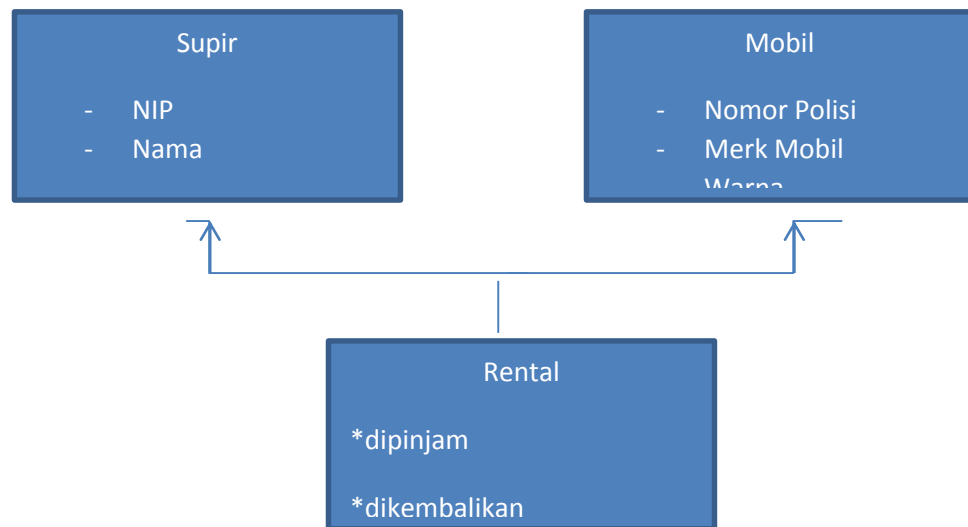
Main class :

```
public class MainClass {
    public static void main(String[] args) {
        Drive d = new Drive();
        d.startEngine();
        d.stopEngine();
    }
}
```

Output dari program :

```
Engine start. You can drive
Engine stop. You cann't drive
```

JURNAL MODUL 2



1. Analisis sistem rental mobil diatas dan tentukan class, atribut dan method yang sesuai!
2. Buatlah kode program dari hasil analisa yang telah kalian lakukan dengan menggunakan inheritance, abstract class dan interface!

MODUL 3

UML, Generalization, and Specialization

TUJUAN PRAKTIKUM

1. Mahasiswa mengetahui fungsi UML.
2. Mahasiswa mengetahui fungsi Use Case Diagram.
3. Mahasiswa memahami fungsi Class Diagram dan SequenceDiagram.
4. Mahasiswa mampu membuat Class Diagram dan Sequence Diagram untuk kasus sederhana.
5. Mahasiswa mengetahui dan memahami pengertian dan manfaat generalization dan specialization sehingga mampu mengimplementasikan konsep tersebut ke dalam sebuah kasus.

LANDASAN TEORI

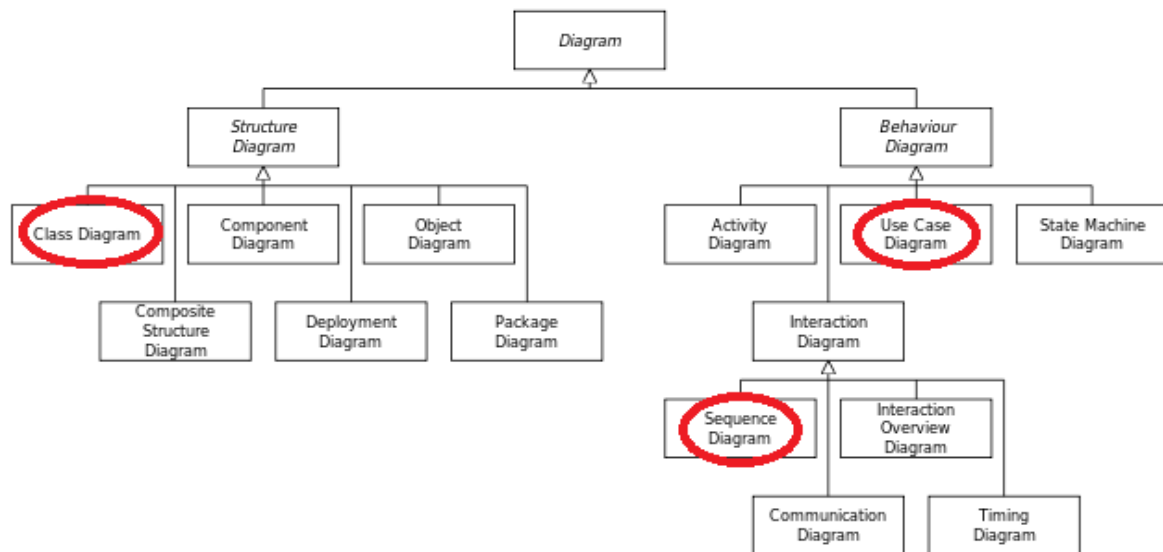
UML

3.1 Pengertian UML

UML, singkatan dari Unified Modelling Language adalah notasi diagram yang merepresentasikan desain program. UML merupakan bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak yang akan dikembangkan.

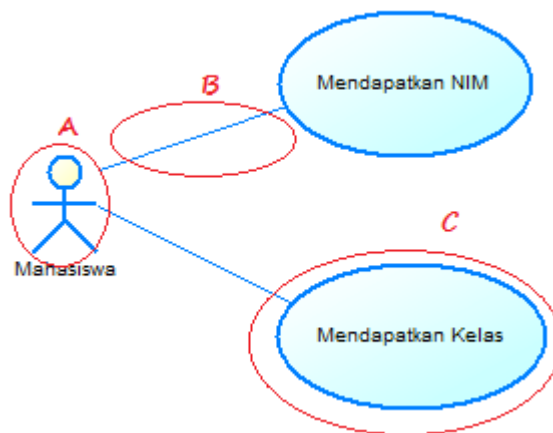
3.2 Diagram-Diagram yang Ada Dalam UML

Gambar di atas merupakan susunan hirarki pada UML. Pada modul kedua praktikum OOP, akan dibahas 3 diagram seperti yang telah dilingkar yaitu Use Case Diagram, Class Diagram dan Sequence Diagram.



3.3 Use Case Diagram

- Fungsi Use Case Diagram :
 1. Menggambarkan fungsionalitas sistem.
 2. Fokus pada “apa” yang dapat dilakukan oleh sistem.
- Contoh Use Case Diagram



Dalam sistem perkuliahan baru, seorang mahasiswa tentunya akan mendapatkan NIM dan kelas.

Keterangan :

A : Actor merupakan entitas yang berinteraksi dengan sistem.

B : Association merupakan relasi Actor dengan Use Case.

C : Use Case merupakan aktivitas yang dapat dikerjakan Actor terhadap sistem.

3.4 Class Diagram

- **Pengertian**

Kelas adalah komponen dasar dari setiap sistem perangkat lunak berorientasi objek. Class diagram menampilkan beberapa kelas dan bagaimana kelas itu berhubungan satu sama lain.

- **Elemen pada Class Diagram**

1. Kelas, yang terdiri dari :

a. Nama kelas (biasanya diawali dengan huruf besar)

b. Sekumpulan atribut

Atribut harus ditampilkan sebagai berikut: *visibility name : type multiplicity*

NamaKelas
Atribut
method()

- Di mana *visibility* salah satu dari:

- (+) public
- (-) private
- (#) protected
- (~) package

- Dan *multiplicity*:

- (n) tepat n
- (*) nol atau lebih
- (m..n) antara m dan n

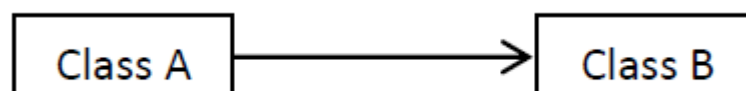
c. Sekumpulan method

2. Relasi

- Class diagram menunjukkan adanya relasi antar kelas. Salah satu jenis relasi adalah **Association (asosiasi)**. Dalam sebuah kelas, atribut akan didefinisikan. Asosiasi digunakan ketika kita ingin memberikan dua kelas yang berelasi .

- Beberapa tipe asosiasi:

a. Simple Association



- Class A menggunakan objek dari Class B.
- Class A memiliki atribut dari Class B.
- Arahnya dari Class A ke Class B.

b. Aggregation



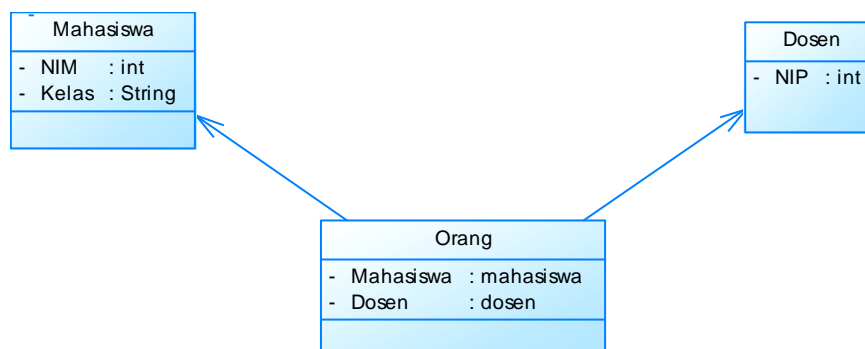
- Melambangkan situasi di mana objek dari Class B “bagian dari” atau “milik dari” Class A.
- Mengimplikasikan referensi dari A ke B.

c. Composition



- Composition mirip dengan aggregation namun mengimplikasikan relasi kepemilikan yang lebih kuat. Objek Class B merupakan bagian dari objek Class A.
- Mengimplikasikan referensi dari A ke B.
- Apabila Class B tidak ada maka Class A juga tidak ada.

• Contoh Implementasi Class Diagram



```

class Mahasiswa {
    private int NIM;
    private String Kelas;
}

class Dosen {
    private int NIP;
}

class Orang {
    Mahasiswa mahasiswa;
    Dosen dosen;
}

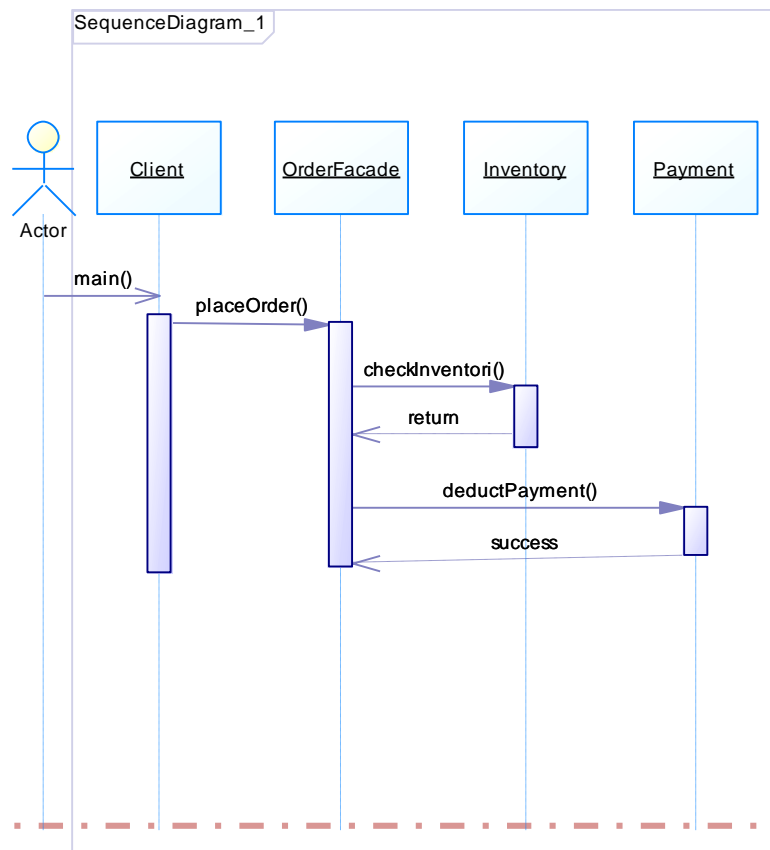
```

3.5 Sequence Diagram

- **Pengertian**

Sequence diagram mendeskripsikan bagaimana sistem bekerja pada suatu periode waktu. Sequence diagram berguna untuk memahami bagaimana objek berkolaborasi dalam skenario tertentu. Umumnya, diagram ini menunjukkan urutan pemanggilan metode di antara objek dalam suatu periode tertentu.

- **Contoh Sequence Diagram**



```
package modul2_3;
public class Inventory {
public String checkInventori(String OrderId) {
return "Inventory checked";
}
}
```

```
package modul2_3;
public class Payment {
public String deductPayment(String orderID) {
return "Payment deducted successfully";
}
}
```

```
package modul2_3;
public class OrderFacade {
private Payment pymt = new Payment();
private Inventory inventory = new Inventory();
public void placeOrder(String orderId) {
String step1 = inventory.checkInventori(orderId);
String step2 = pymt.deductPayment(orderId);
System.out.println("Following steps completed:" + step1 + " & "
+step2);
}
}
```

```
package modul2_3;
public class Client {
public static void main(String args[]){
OrderFacade orderFacade = new OrderFacade();
orderFacade.placeOrder("OR123456");
System.out.println("Order processing completed");
}
}
```

- Penjelasan Sequence Diagram

Diagram di atas menunjukkan:

- Pertama, aktor memanggil method `main()` pada class `Client`. Class `Client` mulai mengeksekusi method `main()` (diindikasikan dengan *vertical bar* - disebut *activation bar*)

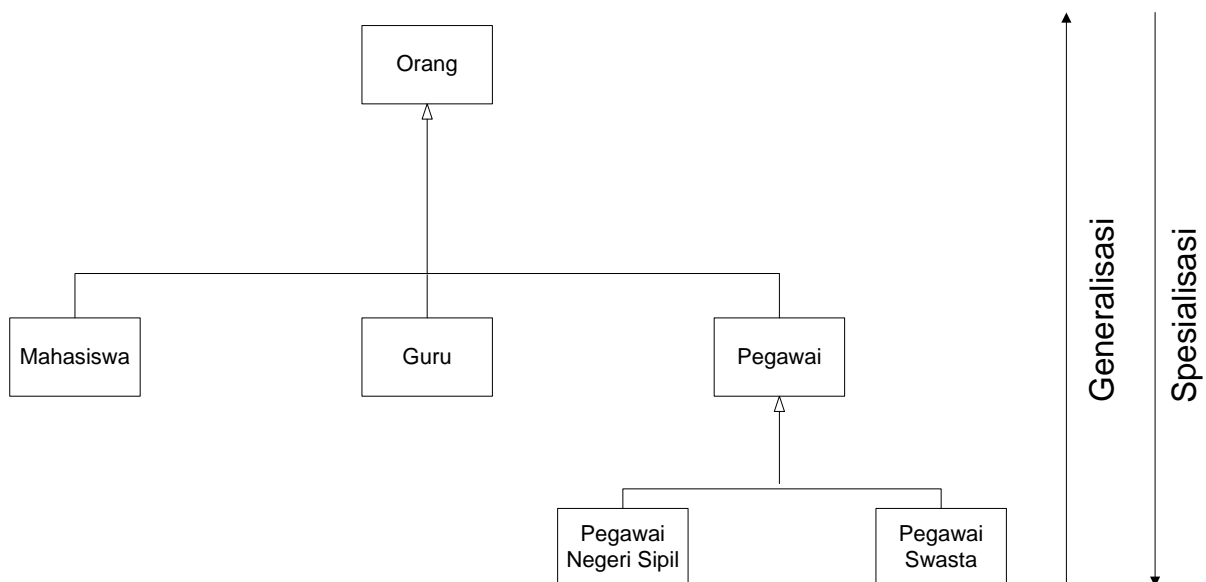
yang dimulai di titik ini. Pada method main() class Client terdapat fungsi placeOrder() yang berasal dari class OrderFacade.

- Method placeOrder() pada class Client memanggil method placeOrder() pada class OrderFacade.
- Method placeOrder() secara subsequence memanggil method checkInventori() pada class Inventory. Ketika method checkInventori() berakhir, ia melewati return value kembali ke method placeOrder().
- Setelah itu, method placeOrder secara subsequence memanggil method deductPayment() pada class Payment. Ketika method deductPayment() berakhir, maka proses akan berakhir dengan sukses dengan mencetak output sebagai berikut :

Following steps completed:Inventory checked & Payment deducted successfully
Order processing completed

Generalization (Generalisasi) and Spesialization (Spesialisasi)

- **Generalisasi** : superclass (kelas induk) yang memiliki sifat umum dari sebuah kasus.
- **Spesialisasi** : class yang mewarisi sifat dari sebuah superclass (kelas induk) dan memiliki sifat yang lebih khusus.



- Sebuah class (child class atau subclass) dapat mewarisi atribut – atribut dan operasi – operasi dari class lainnya (parent class atau super class). Berikut ini terdapat class *orang* dan class *mahasiswa* :

```
package Modul2;

public class Orang {
    protected String name;
    protected String address;

    public Orang() {
        System.out.println("Inside Person:Constructor");
        name = "";
        address = "";
    }

    public Orang(String name, String address) {
        this.name = name;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setName(String name) {
        this.name = name;
        System.out.println(name);
    }

    public void setAddress(String add) {
        this.address = add;
        System.out.println(add);
    }
}
```

```
package Modul2;

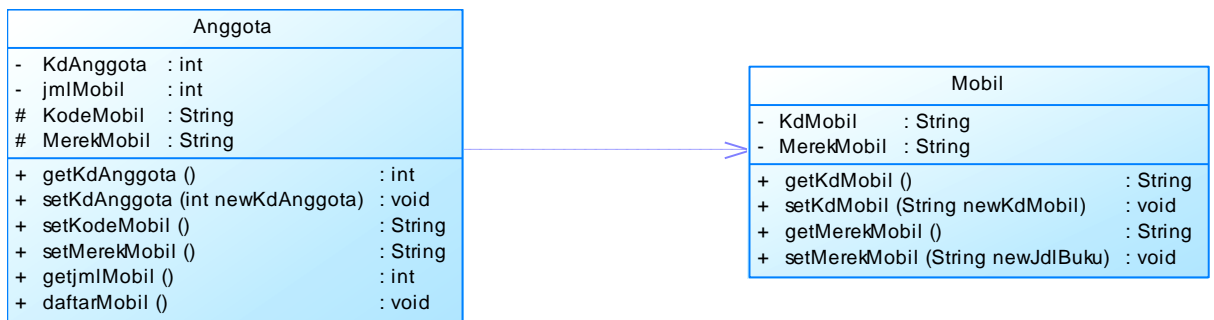
public class Mahasiswa extends Orang {

    public Mahasiswa(){
        System.out.println("Inside Student : Constructor");
    }

    public static void main(String[] args) {
        Mahasiswa Mhs = new Mahasiswa();
        Mhs.setName("Anna");
        Mhs.setAddress("Dayeuhkolot");
    }
}
```

JURNAL MODUL 3

1. Buatlah sebuah program pendataan peminjaman mobil dari Class diagram berikut ini



Dengan output :

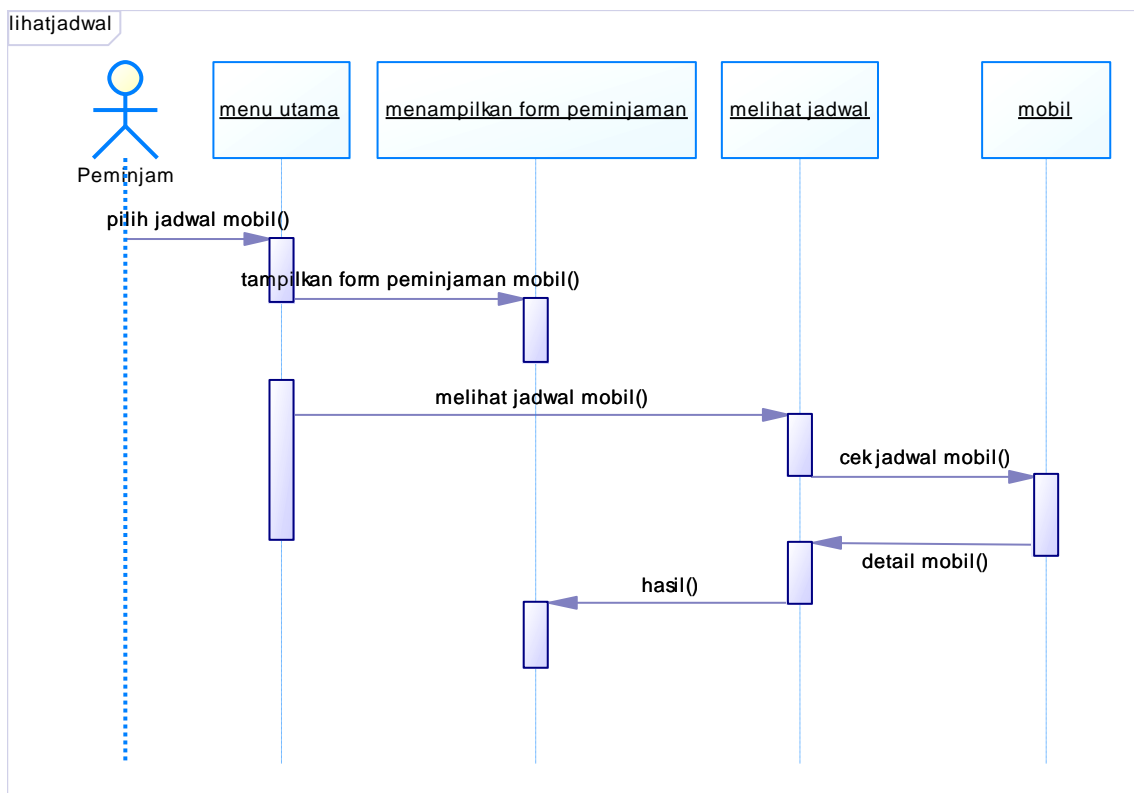
Kode Anggota : 2009-51-100

Daftar Mobil Yang Dipinjam :

D 1010 CC

KB 2272 HD

2. Buatlah sebuah program yang memfasilitasi seorang peminjam dapat melihat jadwal peminjaman mobil yang tersedia. Lihat sequence diagram berikut :



Polymorphism , Overloading & Overriding

TUJUAN PRAKTIKUM :

1. Memahami dan menerapkan konsep polymorphism dalam pemrograman
2. Memahami tentang Overloading
3. Memahami tentang Overriding
4. Praktikan mampu mengimplementasikan polymorphism, overloading & overriding

LANDASAN TEORI

A. Polymorphism

Kemampuan yang dimiliki sebuah class untuk memiliki banyak bentuk disebut juga Polymorphism.

//Animalia.java

```
public class Animalia{
    private String moveMethod;
    private String breathMethod;
    private String eatMethod;

    private String getMoveMethod() {
        return moveMethod;
    }
    protected void setMoveMethod(String moveMethod) {
        this.moveMethod = moveMethod;
    }
    private String getBreathMethod() {
        return breathMethod;
    }
    protected void setBreathMethod(String breathMethod) {
        this.breathMethod = breathMethod;
    }
    private String getEatMethod() {
        return eatMethod;
    }
    protected void setEatMethod(String eatMethod) {
        this.eatMethod = eatMethod;
    }
}
```

//Aves.java

```
public abstract class Aves extends Animalia{
    protected String sound;

    public Aves(){
        setMoveMethod("Fly");
        setBreathMethod("Lungs and Air Sacks");
    }
    @Override
    public void setEatMethod(String eatMethod){
        super.setEatMethod(eatMethod);
    }

    protected abstract String getSound();
    protected abstract void setSound(String sound);
}
```

//Duck.java

```
public class Duck extends Aves{
    public Duck(){
        setSound("quack");
    }

    @Override
    public String getSound() {
        return sound;
    }

    @Override
    protected void setSound(String sound) {
        this.sound = sound;
    }
}
```

//Main.java

```
public class Main{

    public static void main(String[] args){

        Animalia animalOrAves = new Aves() {
            @Override
            public String getSound() {
                return "unknown animal sound";
            }

            @Override
            protected void setSound(String sound) {
                setSound("unknown animal sound");
            }
        };
        Aves aves = new Aves() {
            @Override
            public String getSound() {
                return "unknown aves sound";
            }
        }
    }
}
```

```

        @Override
        protected void setSound(String sound) {
            setSound("unknown aves sound");
        }
    };

    Animalia animalOrDuck = new Duck();
    Aves avesOrDuck = new Duck();

    //method error karena animal tidakmemiliki method
    getSound
    //System.out.println(animalOrAves.getSound());

    //dibutuhkan casting agar memiliki method getSound
    System.out.println(((Aves) animalOrAves).getSound());

    System.out.println(aves.getSound());

    //samaseperti sebelumnya, dibutuhkan casting agar
    dapatmemiliki method yang diinginkan
    System.out.println(((Aves) animalOrDuck).getSound());
    System.out.println(avesOrDuck.getSound());
    }
}

```

Pada class Main.java dapat dilihat contoh polimorfism dimana Class Animal dapat dibentuk dari Class Aves (pada variable animal Or Aves) dan dapat dibentuk dari Duck (pada variable animal Or Duck).

Namun, hal ini tidak dapat terjadi sebaliknya karena subclass (class yang melakukan extends) tidak dapat berpolimorfisme dengan super class-nya.

B. Overloading

Overloading adalah suatu keadaan dimana beberapa method sekaligus dapat mempunyai nama yang sama, akan tetapi mempunyai fungsionalitas yang berbeda. Overloading ini dapat terjadi pada class yang sama atau pada suatu parent class dan subclass-nya. Tujuan dari Overloading itu sendiri adalah untuk memudahkan pemanggilan atau penggunaan method dengan fungsionalitas yang mirip.

Overloading mempunyai ciri-ciri sebagai berikut:

1. Nama method harus sama
2. Daftar parameter harus berbeda
3. Return type boleh sama, juga boleh berbeda

Contoh :

```
package lain;

class Pertambahan{

    public void tambah(){
        int a=5, b=10;
        System.out.println("Hasil Pertambahann dari metod tambah1
ke-1 = "+(a+b));
    }

    //Metod tambah1 di overloading dengan 2 parameter (int x, int y)
    public void tambah(int x, int y){
        System.out.println("Hasil Pertambahann dari metod tambah1
ke-2 = "+(x+y));
    }

}
```

```
package lain;

public class lala {
    public static void main(String [] args){
        Pertambahan pp;
        pp = new Pertambahan();
        pp.tambah();//memanggil metod tambah1 ke-1
        pp.tambah(5,5);//memanggil metod tambah1 ke-2
    }
}
```

Output :

Hasil Pertambahann dari metod tambah ke-1 = 15

Hasil Pertambahann dari metod tambah ke-2 = 10

Konstruktor Overloading

Konstruktor (konstruktor) adalah suatu method yang pertama kali dijalankan pada saat pembuatan suatu obyek. Konstruktor mempunyai cirri yaitu:

- Mempunyai nama yang sama dengan nama class
- Tidak mempunyai return type (seperti void, int, double,dan lain-lain)

Sebuah kelas dapat memiliki lebih dari satu konstruktor. Sama seperti *method*, setiap konstruktor tidak dapat memiliki komposisi argumen yang sama.

//file : Mahasiswa.java

```
public class Mahasiswa {  
    private int nim;  
    private String Nama;  
  
    public Mahasiswa () {  
        nim = 0;  
        Nama = "";  
    }  
  
    public Mahasiswa (String m ) {  
        nim = 0;  
        Nama = m;  
    }  
  
    public Mahasiswa (int ni, String nm) {  
        nim = ni;  
        Nama = nm;  
    }  
}
```

Mahasiswa.java memperlihatkan kelas Mahasiswa yang memiliki 3 buah konstruktor:

- a. Mahasiswa ()
- b. Mahasiswa (String)
- c. Mahasiswa (int , String)

Dalam eksekusi program, jika sebuah konstruktor dipanggil, maka blok kode konstruktor bersangkutan akan dijalankan. Perhatikan File cetak.java.

//File : Cetak.java

```
public class Cetak {  
    public static void main (String[] args) {  
        Mahasiswa m1 = new Mahasiswa ();  
        Mahasiswa m2 = new Mahasiswa ("deny");  
        Mahasiswa m3 = new Mahasiswa (116, "Dana");  
    }  
}
```

Pada Cetak.java diinstanstiasi 3 obyek Mahasiswa, yaitu m1, m2, dan m3. Pada instanstiasi m1, konstruktor Mahasiswa () dipanggil. Menurut konstruktor pada Mahasiswa.java, maka variabel instan pada m1 (nim dan nama) bernilai 0 . Sedangkan pada instanstiasi m2, konstruktor Mahasiswa (String) dipanggil. Sesuai dengan definisi konstruktor pada Mahasiswa.java, maka isi variabel Nama adalah deny, sedangkan nim bernilai 0. Pada instanstiasi m3, konstruktor Mahasiswa(int,String) dipanggil. Sesuai dengan definisi konstruktor pada Mahasiswa.java maka isi variabel nim adalah 116, dan nama adalah Dana.

C. Overriding

Overriding yaitu mekanisme untuk melakukan penindihan/pergantian method yang sebelumnya sudah didefinisikan pada superclass dengan method yang sama(nama nya) yang ada di subclass.

Overriding method mempunyai nama method yang sama, jumlah parameter dan tipe parameter serta nilai kembalian (return) method yang di override.

Ciri-ciri overriding:

- Nama method harus sama
- Daftar parameter harus sama
- Return type harus sama

Berbeda dengan overloading. Jika overloading, kita boleh menuliskan method yang sama namun dengan jumlah parameter yang berbeda dan nilai kembalian harus sama. Lain halnya dengan overriding, konsepnya sama dengan overloading yaitu menulis kembali method. Namun, overriding menulis kembali method sama persis. Sama mulai dari nama method dan isinya dan mengimplementasi kembali di sub classnya.

Overriding dipakai saat kita menggunakan method yang sama tapi berbeda implementasinya. Jadi overriding method mempunyai nama method yang sama, jumlah parameter dan tipe parameter serta nilai kembalian (return) method yang di override. Jika method memiliki modifier akses public, method overridenya juga harus public.

Override method merupakan method yang sama persis dengan method yang sudah ada di super kelasnya, biasanya perbedaannya adalah pada implementasi (program body). Overriding tidak bisa dilakukan dalam kelas itu sendiri. Jadi Overriding erat kaitannya dengan inheritance (pewarisan).

Contoh :

```
public class parent {  
    void dosomething() {  
        System.out.println("parent");  
    }  
}
```

```
public class child extends parent{  
    @Override //override nya di sini  
    void dosomething(){  
        System.out.println("child");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        parent p1 = new parent(); //inisiasi objek  
        parent p2 = new child();  
        p1.dosomething(); //eksekusi objek  
        p2.dosomething(); //pemanggilan method yang telah  
        di override  
    }  
}
```

Output :

```
parent  
child
```


JURNAL MODUL 4

Pada suatu hari, ada sebuah sistem rental mobil yang menggunakan reward bila peminjaman mobil yang telah dilakukan memenuhi. Reward tersebut berupa: memberikan **bonus tambahan** kepada setiap member untuk **batas jumlah jam** dalam setiap peminjaman mobil tersebut. Dimana reward tersebut berlaku untuk 4 golongan:

- Member A: Yang sudah meminjam mobil selama lebih dari 24 hingga 48 jam akan mendapat reward untuk setiap peminjaman mobil berupa:
 - Perpanjangan peminjaman sebanyak 1 jam.
- Member B: Yang sudah meminjam mobil selama lebih dari 49 hingga 72 jam akan mendapat reward untuk setiap peminjaman mobil berupa:
 - Perpanjangan peminjaman sebanyak 2 jam.
- Member C: Yang sudah meminjam mobil selama lebih dari 73 hingga 96 jam akan mendapat reward untuk setiap peminjaman mobil berupa:
 - Perpanjangan peminjaman sebanyak 3 jam.
- Member D: Yang sudah meminjam mobil selama lebih dari 97 hingga 120 jam akan mendapat reward untuk setiap peminjaman mobil berupa:
 - Perpanjangan peminjaman sebanyak 4 jam.

Sedangkan, untuk member yang masih meminjam kurang dari sama dengan 24 jam, batas peminjaman minimal 24 jam.

Sekarang, tentukanlah terlebih dahulu objek mana yang akan dijadikan superclass, dan objek mana yang akan dijadikan subclass. Setelah itu tentukan method apa yang akan di-override.

Jika sudah tahu, silahkan buat kodingan yang menerapkan overriding dari studi kasus di atas. Kemudian, tampilkan hasil compile kodingan.

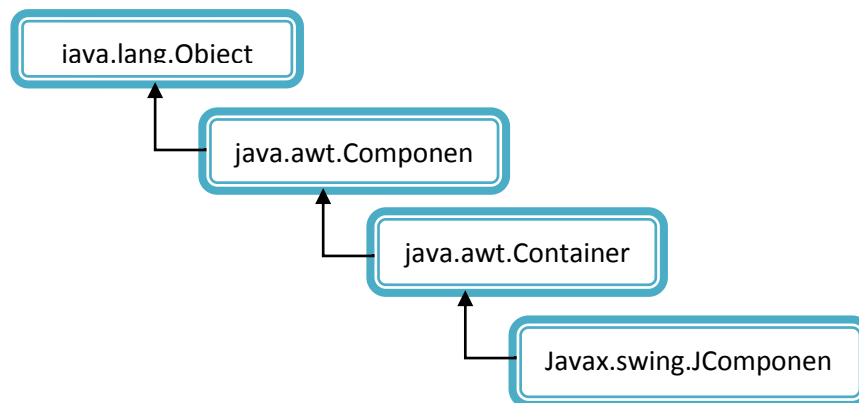
Graphical User Interface (GUI) & Events

TUJUAN PRAKTIKUM

1. Memahami Konsep Graphical User Interface
2. Membuat interface aplikasi dengan berbasis GUI
3. Memahami tentang Action Listener
4. Menerapkan Action Listener
5. Praktikan mampu mengimplementasikan GUI serta event- event yang ada

LANDASAN TEORI

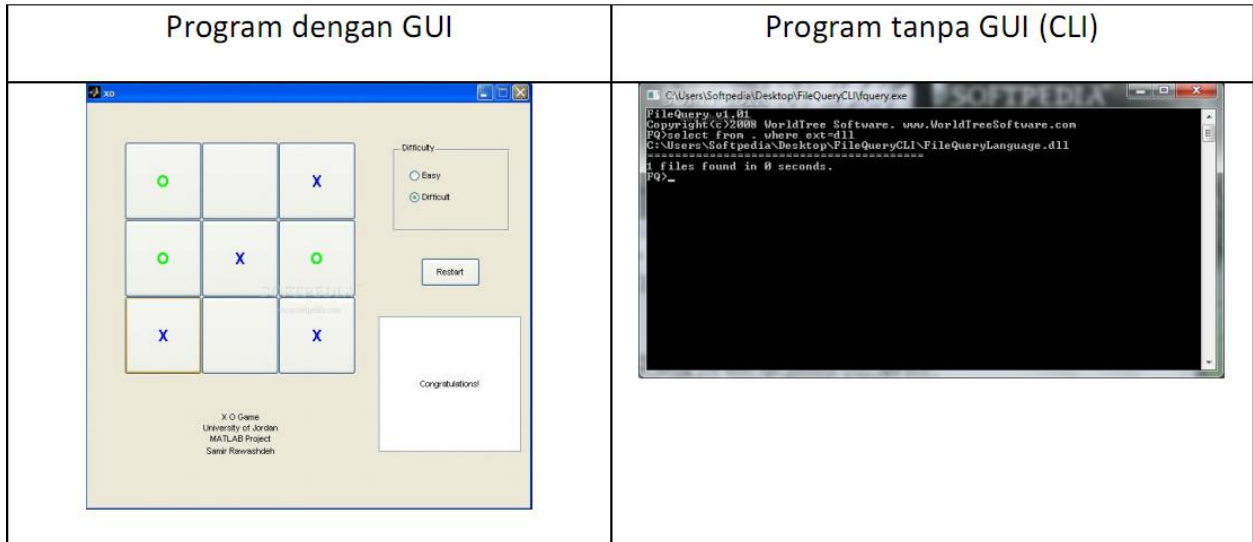
Graphical User Interface (GUI) adalah tampilan-interface grafis pada suatu aplikasi yang berfungsi untuk menjadi perantara antara pengguna dengan suatu program. Pada pemograman Java terdapat komponen untuk membangun GUI. Salah satu komponen yang dapat digunakan adalah Swing. Komponen tersebut didefinisikan dalam paket *javax.swing* yang merupakan komponen GUI yang diturunkan dari Abstract Windowing Toolkit dalam paket *java.awt*.



JComponent adalah superclass dari semua komponen Swing. Beberapa fungsionalitas yang diturunkan dari superclass ini :

- a. Pluggable look dan feel.
- b. Shortcut keys (mnemonics), akses langsung ke komponen melalui keyboard.
- c. Event Handling, penanganan suatu event.
- d. Tool Tips, teks deksripsi yang muncul ketika mouse berada di atasnya.

Perhatikan gambar berikut :



A. Form

Form adalah penahan dari isi (component) yang akan dimasukkan.

- Form : JFrame

JFrame adalah sebuah fungsi yang sering digunakan dalam pemakaian GUI. JFrame dapat diatur sizenya (width-height), diatur apakah bisa di-resize atau tidak, dll.

Contoh penggunaan JFrame :

```
import javax.swing.JFrame;

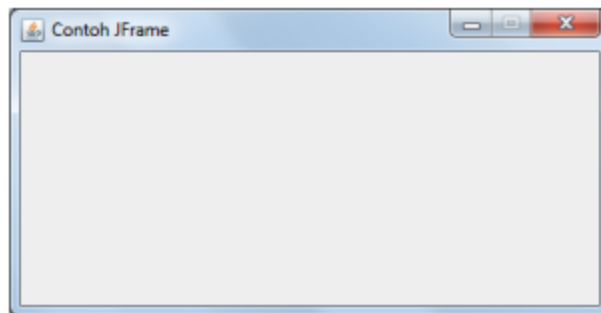
public class ExampleJFrame {
    private JFrame contohJFrame;

    public ExampleJFrame(){
        contohJFrame = new JFrame("Contoh JFrame");
    }

    public void launchFrame(){
        contohJFrame.setSize(400,200);
        contohJFrame.setResizable(false);
        contohJFrame.setVisible(true);
    }

    public static void main(String[] args) {
        ExampleJFrame a = new ExampleJFrame();
        a.launchFrame();
    }
}
```

Output :



B. Component

Component adalah isi yang ada di dalam Form, contoh komponen seperti JTextField, JLabel, JPasswordField, dll.

- JLabel

Fungsi dari Label adalah untuk menampilkan suatu teks di GUI. Teks pada umumnya bersifat read-only. Kelas untuk menampilkan label di GUI bernama JLabel.

Contoh penggunaan JLabel :

```
import java.awt.*;
import javax.swing.*;

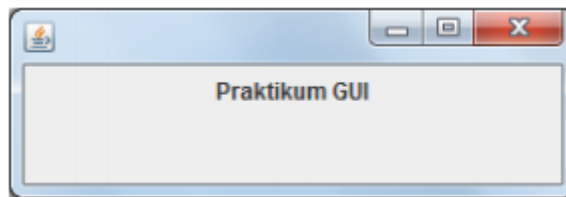
public class ExampleLabel extends JFrame{
    JLabel label1;
    FlowLayout fl;

    public ExampleLabel(){
        Container b = getContentPane();

        //Penggunaan Label
        label1 = new JLabel("Praktikum GUI");
        b.add(label1);
        setLayout(new FlowLayout());
        setSize(300,100);
        show();
    }

    public static void main(String[] args) {
        ExampleLabel a = new ExampleLabel();
    }
}
```

Output :



- JTextField – JPasswordField - JTextArea

JTextField dan JPasswordField adalah fungsi area yang digunakan untuk menampilkan, mengedit, atau menuliskan teks. Perbedaan antara JTextField dengan JPasswordField yaitu penampilan teks dengan format asterisk (*) pada JPasswordField.

Contoh Penggunaan

```
import java.awt.*;
import javax.swing.*;
public class ExampleTextField extends JFrame{
    private JTextField txt1;
    private JPasswordField pss1;
    private JTextArea txtA1;

    public ExampleTextField(){
        Container b = getContentPane();
        setLayout(new FlowLayout());

        //Penggunaan Text Field
        txt1 = new JTextField(10);
        b.add(txt1);

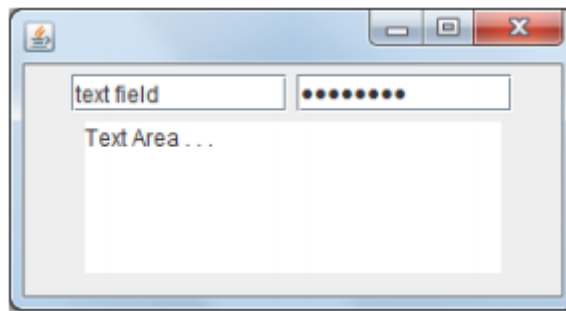
        //Penggunaan Password Field
        pss1 = new JPasswordField(10);
        b.add(pss1);

        //Penggunaan Text Area
        txtA1 = new JTextArea(5,20);
        b.add(txtA1);

        setSize(300,160);
        show();
    }

    public static void main(String[] args) {
        ExampleTextField a = new ExampleTextField();
    }
}
```

Output :



- JButton

JButton merupakan komponen mirip tombol yang terdiri dari beberapa tipe, yaitu command button, toggle button, check boxes, dan radio button. Command button mengaktifkan `ActionEvent` ketika diklik. Command button diturunkan dari kelas `AbstractButton` dan dibuat bersama dengan kelas `JButton`.

Contoh Penggunaan :

```
import java.awt.*;
import javax.swing.*;
public class ExampleJButton extends JFrame{
    private JButton btn1, btn2;

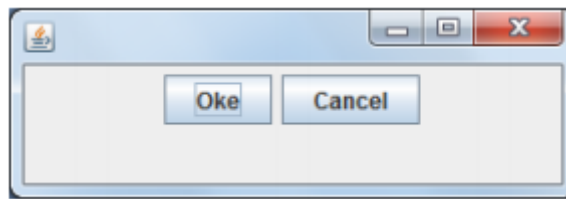
    public ExampleJButton(){
        Container b = getContentPane();

        //contoh penggunaan button
        btn1 = new JButton("Oke");
        btn2 = new JButton("Cancel");
        b.add(btn1);b.add(btn2);

        setLayout(new FlowLayout());
        setSize(300,100);
        show();
    }

    public static void main(String[] args) {
        ExampleJButton a = new ExampleJButton();
    }
}
```

Output :



- JCheckBox dan JRadioButton

JCheckBox dan JRadioButton merupakan komponen turunan dari JToggleButton. Komponen ini memiliki nilai on/off atau true/false.

Pada kelas JCheckBox mengubah nilai ItemEvent. Perubahan nilai ItemEvent ini ditangani oleh interface ItemListener yang mendefinisikan method `itemStateChanged`. Method `getStateChange` pada kelas ItemEvent mengembalikan nilai integer ItemEvent.Selected atau ItemEvent.DESELECTED.

JRadioButtons memiliki dua status yaitu `selected` dan `deselected`. Pada umumnya radio button ditampilkan dalam sebuah grup. Hanya satu radio button di dalam suatu grup yang dapat dipilih pada satu waktu. Pemilihan satu button menyebabkan button lain berstatus off.

Contoh Penggunaan JCheckBox dan JRadioButton :

```
import java.awt.*;
import javax.swing.*;
public class ExampleForm {
    private JFrame frame1;
    private JLabel label1, label2, label3, label4, label5, label6;
    private BorderLayout bl;
    private JCheckBox cb1, cb2;
    private JRadioButton rb1, rb2;
    private JPanel a, b, c, d;
    private GridLayout gl;
    private ButtonGroup bg;
    private JButton button1;

    public ExampleForm(){
        frame1 = new JFrame("Contoh Form1");
        a = new JPanel();
        b = new JPanel();
        c = new JPanel();
        d = new JPanel();
        label1 = new JLabel("Contoh CheckBox :");
        label2 = new JLabel("Contoh RadioButton :");
        label3 = new JLabel("Prodase");label4 = new
JLabel("Laboratory");
        label5 = new JLabel("Pria");label6 = new
JLabel("Wanita");
        cb1 = new JCheckBox();cb2 = new JCheckBox();
        rb1 = new JRadioButton();rb2 = new JRadioButton();
    }
}
```

```

    bg = new ButtonGroup();
    bg.add(rb1);bg.add(rb2);
    button1 = new JButton("Submit");

    bl = new BorderLayout();
    gl = new GridLayout(2,2);
}
public void launchForm(){
    frame1.setSize(360,120);
    frame1.add(a, BorderLayout.CENTER);
    frame1.add(b, BorderLayout.SOUTH);
    a.setLayout(gl);

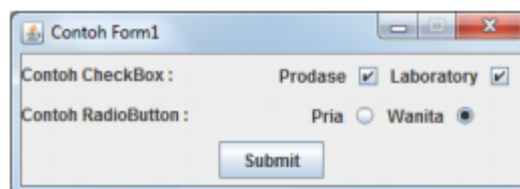
    a.add(label1);a.add(c);c.add(label3);c.add(cb1);c.add(label4);
    c.add(cb2);

    a.add(label2);a.add(d);d.add(label5);d.add(rb1);d.add(label6);
    d.add(rb2);
    b.add(button1);
    frame1.setResizable(false);
    frame1.setVisible(true);
}

public static void main(String[] args) {
    ExampleForm g = new ExampleForm();
    g.launchForm();
}
}

```

Output :



- JComboBox

JComboBox merupakan komponen GUI yang berfungsi untuk menampilkan daftar suatu item. Penggunaan konstuktur dari JComboBox adalah sebagai berikut:

JcomboBox(arrayOfNames);

Setiap item di JComboBox diberi indeks numerik. Elemen pertama diberi indeks 0 dan elemen tersebut dimunculkan sebagai item yang dipilih pada saat instance JComboBox tampil untuk pertama kalinya. Method penting JComboBox adalah sebagai berikut :

1. `getSelectedIndex()` mengembalikan indeks dari item yang sedang dipilih
2. `setMaximumRowCount(n)` menentukan jumlah maksimum elemen yang ditampilkan ketika pengguna mengklik instance `JComboBox`. Scrollbar secara otomatis dihasilkan.

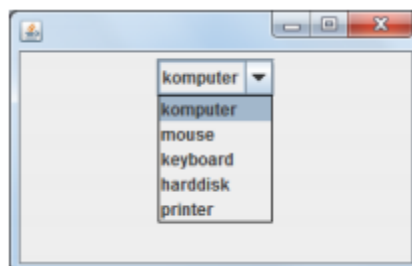
Contoh Penggunaan :

```
import java.awt.*;
import javax.swing.*;
public class ExampleJComboBox extends JFrame{
    private JComboBox cbx;

    public ExampleJComboBox(){
        Container a = getContentPane();
        String[] items = {"komputer", "mouse", "keyboard",
"harddisk", "printer"};
        cbx = new JComboBox(items);
        a.setLayout(new FlowLayout());
        a.add(cbx);
        a.add(new JScrollPane(cbx));
        setSize(300,190);
        show();
    }

    public static void main(String[] args) {
        ExampleJComboBox x = new ExampleJComboBox();
    }
}
```

Output :



- JMenuBar

Menu merupakan bagian penting dalam GUI. Objek menu dipasang pada obyek kelas yang memiliki method setJMenuBar. Menu juga mempunyai ActionEvents. Gambar berikut menunjukkan tampilan menu:

Contoh Penggunaan :

```
import java.awt.*;
import javax.swing.*;
public class ExampleJMenuBar extends JFrame{
    private JMenuBar menu;
    private JMenu file, help;
    private JMenuItem newMenu, openMenu, exitMenu, helpMenu;

    public ExampleJMenuBar(){

        Container c = getContentPane();

        //instantiasi menubar
        menu = new JMenuBar();

        //instantiasi menu
        file = new JMenu("File");
        help = new JMenu("Help");

        //instantiasi item dari menu
        newMenu = new JMenuItem("New");
        openMenu = new JMenuItem("Open");
        exitMenu = new JMenuItem("Exit");
        helpMenu = new JMenuItem("Help");

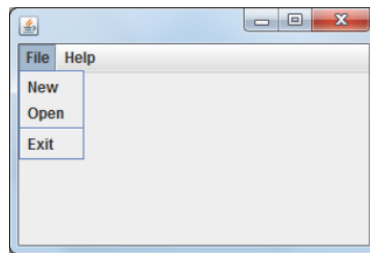
        setJMenuBar(menu);
        menu.add(file);menu.add(help);
        file.add(newMenu);file.add(openMenu);file.addSeparator();file.add(exitMenu);
        help.add(helpMenu);

        setLayout(new FlowLayout());
        setSize(300,200);
        show();

    }

    public static void main(String[] args) {
        ExampleJMenuBar mb = new ExampleJMenuBar();
    }
}
```

Output :



C. Layout Manager

Layout Manager menyusun komponen GUI di atas container. Penggunaan layout memberikan kemudahan dibandingkan dengan menentukan ukuran eksak dan posisi setiap komponen, sehingga para programmer akan lebih berkonsentrasi terhadap look and feel saja.

Pada layout manager, komponen diletakan dari kiri ke kanan dan lalu ke baris berikutnya. Komponen dapat diletakan dengan cara rata kiri, di tengah, ataupun rata kanan. Nilai defaultnya adalah di tengah. Tipe-tipe layout dapat berupa FlowLayout, BorderLayout, GridLayout.

- FlowLayout

FlowLayout adalah layout yang menyusun komponen dari kiri ke kanan, selanjutnya ke baris berikutnya. Jika ukuran windows diperbesar ukuran komponen pada FlowLayout tidak berubah. Method-method penting dari FlowLayout adalah sebagai berikut :

- a. `setAlignment(position_CONSTANT)`:
method untuk menentukan posisi layout. Nilai `position_CONSTANT` dapat berupa `FlowLayout.LEFT`, `FlowLayout.CENTER`, atau `FlowLayout.RIGHT`, yang masing-masing mengatur posisi rata kiri, rata tengah, atau rata kanan.
- b. `layoutContainer(container)`: method untuk melakukan update container.

Contoh Penggunaan :

```
import javax.swing.*;
import java.awt.*;
public class FlowLayoutTest extends JFrame {
    public FlowLayoutTest(){
        super("Contoh Flow Layout");
        FlowLayout lay = new FlowLayout();
        lay.setAlignment(FlowLayout.RIGHT);
        lay.setVgap(25); //mengatur jarak vertikal antar komponen
    }
}
```

```

lay.setHgap(25);//mengatur jarak horisontal antar komponen
Container c = getContentPane();
c.setLayout(lay);
JButton t1 = new JButton("Tombol 1");
JButton t2 = new JButton("Tombol 2");
JButton t3 = new JButton("Tombol 3");
JButton t4 = new JButton("Tombol terpanjang no 4");
JButton t5 = new JButton("Tombol 5");
JButton t6 = new JButton("Tombol 6");
c.add(t1);
c.add(t2);
c.add(t3);
c.add(t4);
c.add(t5);
c.add(t6);
}
public static void main(String[] args){
FlowLayoutTest flt = new FlowLayoutTest();
flt.setSize(500, 200);
flt.setVisible(true);
}
}

```

Output :



- **BorderLayout**
 BorderLayout merupakan default manager untuk ContentPane. Layout ini menyusun komponen ke dalam 5 region, yaitu north, south, east, west, dan center. Komponen-komponen dapat diletakan pada:
 - a. North/South. Komponen di region ini dapat diperluas secara horisontal.
 - b. East/West. Komponen di region ini dapat diperluas secara vertikal
 - c. Center. Komponen di region ini dapat diperluas secara vertikal dan horizontal

Method-method penting yang dapat digunakan pada BorderLayout adalah sebagai berikut:

- a. Konstruktors BorderLayout(hGap, vGap). Argumen hGap adalah ukuran gap horizontal antar region. Argumen vGap adalah ukuran gap vertikal antar region. Nilai defaultnya adalah 0 baik untuk vertikal maupun horizontal
- b. myContainer.add(component, position) menambahkan komponen ke layout. Argumen component menunjukkan komponen yang ditambahkan ke layout, sedangkan argumen position menunjukkan posisi peletakan komponen, sebagai contoh BorderLayout.NORTH, BorderLayout.South, BorderLayout.EAST, BorderLayout.WEST, ataupun BorderLayout.CENTER.

Contoh Penggunaan

```
import javax.swing.*;
import java.awt.*;

public class DemoBorderLayout extends JFrame {
    private JButton tombol[];
    private String names[]={"Hilangkan North","Hilangkan South",
"Hilangkan East","Hilangkan West",
"Hilangkan Center"};

    private BorderLayout lout;

    public DemoBorderLayout(){
        super ("Ini Adalah Contoh Border Layout");
        Container c = getContentPane();
        lout = new BorderLayout (10,10);
        c.setLayout(lout);
        tombol = new JButton[names.length];

        for (int i=0 ; i < names.length; i++){
            tombol[i] = new JButton(names[i]);
        }
        c.add(tombol[0], BorderLayout.NORTH);
        c.add(tombol[1], BorderLayout.SOUTH);
        c.add(tombol[2], BorderLayout.EAST);
        c.add(tombol[3], BorderLayout.WEST);
        c.add(tombol[4], BorderLayout.CENTER);
        setSize (500,300);
    }

    public static void main(String[] args) {
        DemoBorderLayout dbl = new DemoBorderLayout();
        dbl.setVisible(true);
        dbl.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Output :



- **GridLayout**

GridLayout membagi Container ke dalam suatu grid. Komponen diletakkan dalam suatu baris dari kolom dan memiliki ukuran lebar dan tinggi yang sama. Komponen-komponen ditambahkan mulai dari kiri atas, selanjutnya ke kanan. Jika baris sudah penuh, maka komponen diletakkan di baris selanjutnya, kemudian dari kiri ke kanan.

Konstruktur GridLayout adalah sebagai berikut :

- a. `GridLayout(rows, columns, hGap, vGap)`. Konstruktur ini mendefinisikan jumlah baris, kolom, dan ukuran gap horisontal maupun vertikal antar elemen dalam satuan pixel;
- b. `GridLayout(rows, columns)`. Sama halnya dengan konstruktur pertama, namun dengan nilai default `hGap` and `vGap` sama dengan 0.

Contoh Penggunaan :

```
import javax.swing.*;
import java.awt.*;
public class GridLayoutTest extends JFrame{
    private JButton tombol[];
    private String m[] = {"Satu", "Dua", "Tiga", "Empat", "Lima", "Enam"};
    private Container c;
    private GridLayout g;
    public GridLayoutTest(){
        super("Demonstrasi GridLayout");
        g = new GridLayout(2,3,5,5);
        c = getContentPane();
        c.setLayout(g);
        tombol = new JButton[m.length];
        for(int i = 0; i<m.length;i++){
            tombol[i] = new JButton(m[i]);
```

```

        c.add(tombol[i]);
    }
    setSize(300,300);
    show();
}
public static void main(String[] args) {
    GridLayoutTest glt = new GridLayoutTest();
}
}

```

Output :



Events

Events adalah sebuah cara untuk melakukan respon atas perlakuan user terhadap sebuah komponen. Contoh dari event dalam kehidupan real adalah :

- Baju Dipakai
- Rumah dimasuki Orang
- Portal dilewati
- Gelas akan jatuh
- Gelas telah jatuh
- dan lainnya.

Dalam java, hal tersebut juga dapat terjadi, semisal pada JButton, event yang terjadi adalah actionPerformed, dimana fungsi akan dijalankan ketika JButton tersebut di "klik".

Perhatikan contoh berikut!

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class theMain extends JFrame implements ActionListener {
    /*
     * Perhatikan bahwa class theMain di atas terdapat tambahan baru, yaitu
     * implements ActionListener. Tujuannya adalah JFrame theMain dapat
     * "merespon" segala jenis klik dari JButton.
     */

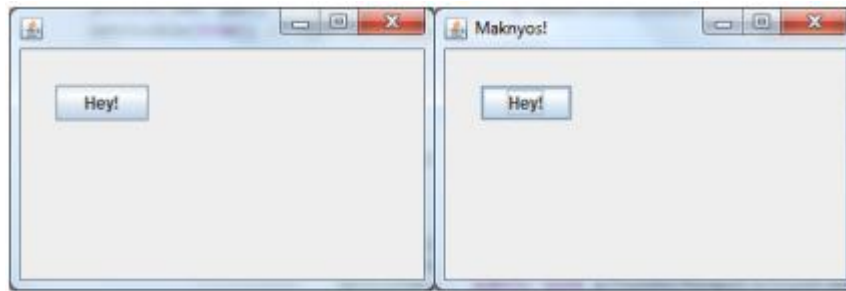
    theMain() {
        // manajemen tata letak form
        setLayout(null); // AbsoluteLayout
        setSize(300, 200); // Menentukan ukuran form dalam satuan pixel
        setVisible(true); // Agar form muncul

        // manajemen komponen
        JButton b = new JButton("Hey!"); // Membuat button
        b.setBounds(25, 25, 65, 25);
        /*
         * Mengatur ukuran button (koordinat X,
         * koordinat Y, panjang, lebar)
         */
        b.setVisible(true);
        add(b); // Memasukkan button ke dalam form
        b.addActionListener(this); /* Me-Registrasi Button untuk "direkam"
            aktivitas klik nya. */
    }

    // region Main
    public static void main(String[] args) {
        new theMain();
    }
    // endregion

    @Override
    public void actionPerformed(ActionEvent arg0) {
        /*
         * fungsi actionPerformed ini adalah autogenerate ketika
         * JFrame di atas mengimplementasikan ActionListener
         *
         * Selalu perhatikan tanda seru/silang yang ada di
         * eclipse kalian.
         */
        // Di bawah ini terdapat fungsi yang akan dijalankan
        this.setTitle("Maknyos!"); //Mengganti judul form menjadi maknyos
    }
}
```


Output :



Contoh lain, form sendiri (JFrame) juga bisa terjadi event, menggunakan implementasi WindowListener. Ketika mengimplementasi WindowListener, maka otomatis akan tergenerate fungsi berikut :

```
@Override
public void windowActivated(WindowEvent arg0) {}
// Ter-invoke ketika form window tersebut menjadi "active Window" (focused)
```

```
@Override
public void windowClosed(WindowEvent arg0) {}
// Ter-invoke ketika form window tertutup.
```

```
@Override
public void windowClosing(WindowEvent arg0) {}
// Ter-invoke ketika form window akan ditutup.
```

```
@Override
public void windowDeactivated(WindowEvent arg0) {}
// Ter-invoke ketika form tersebut kabur fokus nya (deactivated window/defocuse
```

```
@Override
public void windowDeiconified(WindowEvent arg0) {}
// Ter-invoke ketika form berubah state dari minimized menjadi normal.
```

```
@Override
public void windowIconified(WindowEvent arg0) {}
// Ter-invoke ketika form berubah state dari normal menjadi minimzed.
```

```
@Override
public void windowOpened(WindowEvent arg0) {}
// Ter-invoke ketika form "muncul" ke permukaan, atau terbuka, atau Nampak
```

Contoh Penggunaan :

```
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JFrame;

public class theMain extends JFrame implements WindowListener {
    /* * Perhatikan bahwa class theMain di atas terdapat tambahan baru, yaitu
    * implements ActionListener. Tujuannya adalah JFrame theMain dapat
    * "merespon" segala jenis klik dari JButton.
    */

    /**
     *
     */
    private static final long serialVersionUID = -3090893483416423354L;
    //Nggak Penting :p
    theMain() {
        // manajemen tata letak form
        setLayout(null); // AbsoluteLayout
        setSize(300, 200); // Menentukan ukuran form dalam satuan pixel
        setVisible(true); // Agar form muncul

        // manajemen komponen
        addWindowListener(this);
    }

    // region Main
    public static void main(String[] args) {
        new theMain();
    }
    // endregion

    @Override
    public void windowActivated(WindowEvent arg0) {
        System.out.println("windowActivated");
    }
    @Override
    public void windowClosed(WindowEvent arg0) {
        System.out.println("windowClosed");
    }
    @Override
    public void windowClosing(WindowEvent arg0) {
        System.out.println("windowClosing");
    }
    @Override
    public void windowDeactivated(WindowEvent arg0) {
        System.out.println("windowDeactivated");
    }
    @Override
    public void windowDeiconified(WindowEvent arg0) {}
    @Override
    public void windowIconified(WindowEvent arg0) {}
}
```

```
@Override
public void windowOpened(WindowEvent arg0) {
    System.out.println("windowOpened");
}

}
```

Maka ketika program dijalankan, console akan mengeluarkan tulisan “windowActivated” dan “windowOpened”, serta ketika form di tutup maka akan menampilkan tulisan “windowClosing” dan “windowDeactivated”.

Handling Mouse Events

Mouse event itu dipermissalkan seperti interaksi user saat menggerakkan mouse (moving), dragging mouse (menggerakkan mouse sambil menekan mouse button) dan mengklik mouse button.

Untuk menangani mouse event ,kita harus mengimplementasikan salah satu interface berikut

1. **MouseListener**

Mouse Listener merupakan sebuah interface yang digunakan untuk menangani reaksi event yang terjadi pada penekanan tombol mouse

2. **MouseMotionListener**

MouseMotionListener adalah sebuah interface yang digunakan untuk menangani reaksi event dari suatu pergerakan mouse terhadap komponen

3. **MouseWheelListener**

MouseWheelListener adalah sebuah interface untuk menangani reaksi event scroll pada mouse

Jika salah satu interface telah diimplementasikan ,maka kita juga harus menggunakan method method yang ada dalam interface tersebut.

Method / event yang terdapat pada MouseListener adalah

- **mouseClicked()** yaitu method untuk menangani aksi saat mouse diklik dengan syarat kode dalam mouseRelease tidak dijalankan
- **mousePressed()** yaitu method untuk menangani aksi saat menekan mouse
- **mouseReleased()** yaitu method untuk menangani aksi saat melepas button mouse.
- **mouseEntered()** yaitu method untuk menangani aksi saat cursor mouse masuk kewilayah komponen

- `mouseExited()` yaitu method untuk menangani aksi saat cursor mouse keluar dari wilayah komponen

Method / event yang terdapat pada `MouseMotionListener` adalah

- `mouseMoved()` yaitu method yang menangani aksi ketika pointer digerakkan
- `mouseDragged()` yaitu method yang menangani aksi mouse ketika didrag dan drop

Method / event yang terdapat pada `MouseWheelListener` adalah

- `mouseWheelMoved()`

Kita bisa mendapatkan koordinat x dan y ,ketika mouse ditekan dengan menggunakan 2 methods yang terdapat pada class `MouseEvent`

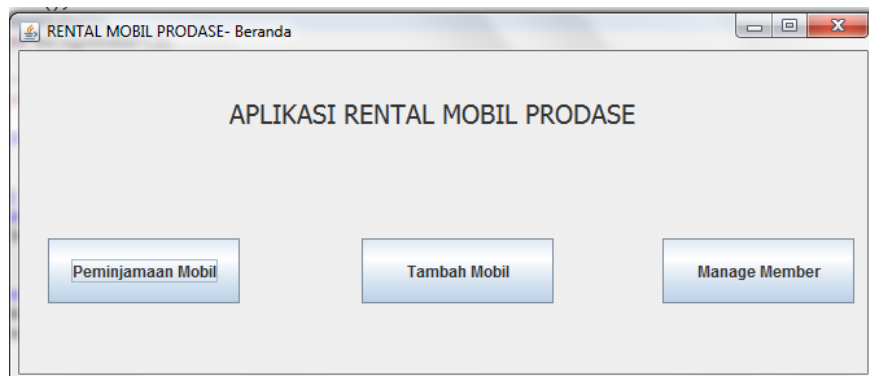
Methods tersebut adalah

1. `int getX()` : untuk mendapatkan koordinat x
2. `int getY()` : untuk mendapatkan koordinat y

Tips : Biasakan untuk menulis semua variabel komponen di dalam class, bukan di dalam method, agar semua method dapat memanipulasi komponen tersebut.

JURNAL MODUL 5

Rental mobil Prodase ingin membuat aplikasi peminjaman mobil, bantulah dengan membuat GUI yang berisi tentang Beranda, Peminjaman, Tambah daftar mobil, Manage member.



RENTAL MOBIL PRODASE-Tambah Mobil

Tambah Mobil

ID Mobil

Nama Mobil

Jenis Mobil

Harga Mobil

RENTAL MOBIL PRODASE- Manage Member

Manage User

NIM

Nama

Jenis Kelamin ☐ Laki - Laki ☐ Perempuan

Pekerjaan

Mahasiswa

Mahasiswa

Pegawai

Karyawan

Exception Handling

TUJUAN PRAKTIKUM

1. Praktikan dapat mengetahui macam-macam *exception* dalam java
2. Praktikan dapat menggunakan penanganan *exception* yang sesuai dalam proses pembuatan program
3. Praktikan mampu mengimplementasikan *exception handling* pada suatu kasus.

LANDASAN TEORI

1. Exception

Exception merupakan kondisi tidak biasa yang muncul pada saat runtime dan diwakilkan sebagai objek. Salah satu kegunaan exception ialah untuk menelusuri informasi mengenai error, sehingga memungkinkan untuk mendiagnosa penyebab masalah atau memberikan petunjuk kenapa error itu muncul.

2. Exception Handling

Exception handling merupakan suatu mekanisme untuk menangani error/kesalahan yang muncul pada saat software dijalankan.

3. Macam-macam Exception

Dalam bahasa pemrograman Java *exception* dibagi menjadi 2 macam yaitu :

a. Checked Exception

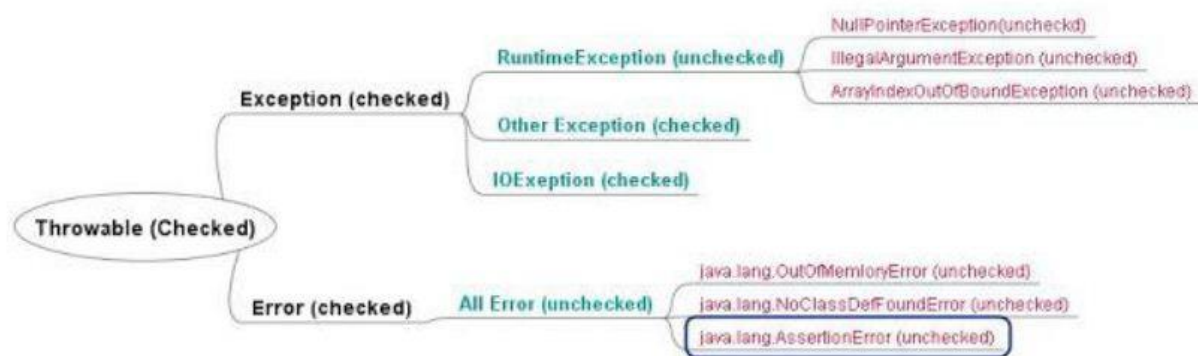
Exception yang muncul akibat dari kondisi eksternal yang dapat saja muncul dari program. *Exception* jenis ini harus ditangani oleh *programmer*. Dalam java merupakan *subclass (extend)* dari `java.lang.Exception class`. Contoh dari *checked exception* misalkan kondisi ketika sebuah *file* yang diminta tidak dapat ditemukan atau adanya *network failure*.

b. Unchecked Exception

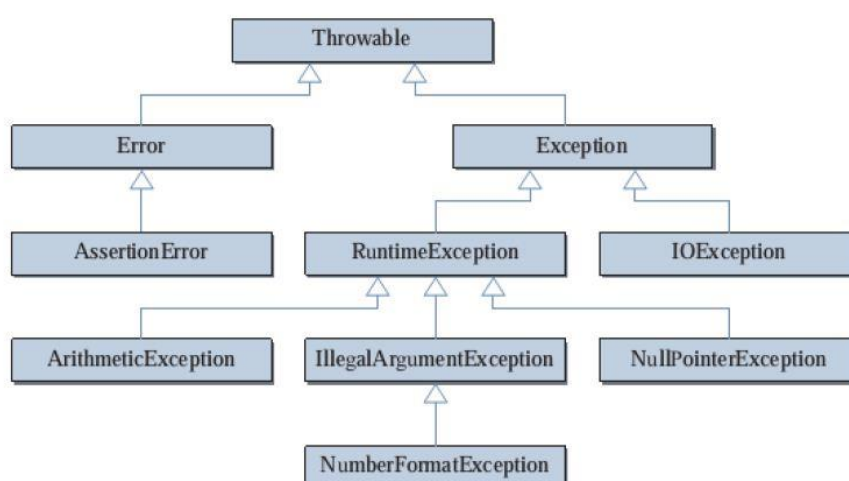
Kategori *exception* yang muncul akibat dari adanya *bugs* dalam program atau situasi yang terlalu sulit untuk ditangani. Dalam java merupakan *extend* dari `java.lang.RuntimeException class`.

Baik *checked* maupun *unchecked exception* dapat ditangani oleh seorang *programmer*. Untuk *checked exception* maka *programmer* wajib menangani *checked exception* tersebut dengan menggunakan blok `catch()` maupun memakai `throws`. Kelas *Exception* dari java merupakan kelas dasar dari *checked exception*. Kelas *Error* adalah kelas dasar yang digunakan untuk *unchecked exception* yang berasal dari *error fatal* dari program yang sulit untuk ditangani.

Berikut ini merupakan gambar pembagian beberapa *checked* dan *unchecked exception* yang terdapat dalam java



Berikut ini merupakan gambar dari hirarki *exception* yang terdapat dalam java :



Exception pada gambar di atas hanya menggambarkan beberapa *exception* yang sering ditangani pada java, sebenarnya masih ada banyak lagi *exception* yang dapat ditangani.

4. Penanganan *Exception*

Berikut ini merupakan contoh program yang dibuat tanpa menggunakan penanganan *exception*.

```
public class ExceptionHandling {  
    public static void main(String[] args) {  
        int i = 0;  
        String greetings[] = { "Hello world!", "No, I mean  
                                it!", "HELLO WORLD!" };  
  
        while (i < 4) {  
            System.out.println(greetings[i]);  
            i++;  
        }  
    }  
}
```

Jika kode di atas dijalankan maka akan menghasilkan beberapa baris output kemudian berhenti karena adanya *error*. Output program tersebut adalah sebagai berikut:

```
Hello world!  
No, I mean it!  
HELLO WORLD!  
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException: 3  
    at ExceptionHandling.main(ExceptionHandling.java:6)
```

Berikut ini merupakan syntax-syntax dalam java yang dapat digunakan untuk melakukan penanganan *exception*:

a. try – catch

Statement try mengindikasikan kode yang menghasilkan *exception*. Statemen *catch* digunakan untuk menangkap *exception* yang dilempar di dalam blok try. Dalam sintaksnya, setelah sebuah blok try, dapat ditulis beberapa blok catch, seperti dalam contoh berikut:

```
try {  
    //code that might throw a particular exception  
} catch(myExceptionType myExcept) {  
    //code to execute if a MyExceptionType exception is thrown  
} catch(ExceptionotherExcept) {  
    //code to execute if a general Exception exception is thrown  
}
```

Saat menuliskan multiple block *catch* pastikan bahwa exception yang dituliskan berurutan dari level paling bawah (*subclass* paling bawah) sampai level paling atas.

Contoh:

```
try{
    ...
} catch (Exception e) {
    ...
} catch (InputMismatchException e) {
    ...
}
```

Contoh di atas merupakan contoh penulisan *exception* yang salah, *exception* `InputMismatchException` tidak akan dieksekusi karena `InputMismatchException` berada di bawah `Exception` pada hirarkinya (lihat kembali gambar hirarki exception java).

b. finally

Blok *finally* merupakan blok yang akan dieksekusi baik jika terjadi eksepsi yang harus dijalankan (terjadi eksekusi pada blok *catch*) maupun tidak (hanya menjalankan blok *try*).

Dalam penulisan sebuah *exception* sebuah blok *try* minimal memiliki satu blok *catch* atau satu blok *finally*.

Contoh:

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class Stack {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            int num = scanner.nextInt();
            if (num < 0) {
                throw new Exception("No negative");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid Entry");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("DONE");
        }
    }
}
```

Keyword *throw* digunakan untuk melemparkan suatu eksepsi secara manual. Biasanya eksepsi yang dilempar adalah eksepsi yang kemungkinan terjadinya sudah diprediksi oleh programmer.

5. Call Stack

Jika sebuah pernyataan melemparkan sebuah exception, dan exception ini tidak ditangani di dalam method yang melingkupinya, maka exception tersebut akan dilemparkan ke method yang memanggilnya. Jika exception ini juga tidak ditangani di dalam method pemanggil, maka exception ini akan dilemparkan ke method pemanggil yang memanggil method ini. Demikian seterusnya. Jika exception tidak juga ditangani ketika ia mencapai method `main()`, maka program akan berhenti dengan tidak normal. Perhatikan kode program berikut ini :

```
public class Stack {  
    public static void main(String[] args) {  
        first();  
    }  
  
    public static void first() {  
        second();  
    }  
  
    public static void second() {  
    }  
}
```

Pada contoh di atas, method `main()` memanggil sebuah method yang bernama `first()`, dan method ini memanggil method lain yang bernama `second()`. Jika exception terjadi di dalam method `second()` dan method `second()` tidak menangani exception ini, maka exception akan dilempar ke dalam method `first()`. Misalkan di dalam method `first()` terdapat blok `catch()` yang menangani exception ini, maka exception tersebut tidak dilemparkan lagi ke method pemanggil berikutnya. Namun, jika method `first()` tidak mempunyai blok `catch()` yang sesuai, maka method berikutnya, yaitu `main()` akan dicek. Jika method `main()` juga tidak menangani exception ini, maka exception tersebut akan dicetak ke dalam output standar dan program berhenti berjalan.

6. User Defined Exception

Dalam pembangunan suatu aplikasi biasanya dibutuhkan sebuah exception yang secara manual didefinisikan. Exception ini adalah exception-exception khusus yang belum

didefinisikan oleh java. Untuk membuat *User Definied Exception* programmer dapat melakukan extend pada class *Exception* dan menggunakan method-method yang ada dilammnya. Berikut ini merupakan method-method dalam class exception yang dapat digunakan untuk membuat *User Definied Exception*.

Exception

```
getLocalizedMessage()
getMessage()
toString()
```

Berikut ini merupakan contoh sebuah user defined exception yang digunakan untuk menangani input umur yang bernilai negatif dari user.

- User defined exception

```
public class UserDefined extends Exception {
    private int age;
    public UserDefined(int age) {
        this.age = age;
    }
    public String toString() {
        return "Age should not be in negative, you are entered"
+ age;
    }
}
```

- Main class

```
import java.util.Scanner;
public class TestUserDefinedException {
    public static void main(String[] args) throws Exception {
        Scanner in=new Scanner(System.in);
        int inputNumber = in.nextInt();
        if (inputNumber < 0)
            throw new UserDefined(inputNumber);
        else
            System.out.println("Your age is : " +
inputNumber);
    }
}
```

7. Exception yang sering digunakan

Beberapa exception yang biasa terjadi adalah :

Tipe Exception	Deskripsi
----------------	-----------

ArithmeticException	Kesalahan aritmetika, seperti pembagian dengan nol.
ArrayIndexOutOfBoundsException	Indeks array keluar dari batas.
ArrayStoreException	Menunjuk ke sebuah elemen array yang tipenya tidak kompatibel.
IllegalArgumentException	Argument tidak sah digunakan untuk memanggil sebuah argument.
IndexOutOfBoundsException	Beberapa tipe indeks keluar dari batas
NegativeArraySizeException	Array dibuat dengan ukuran negative
NullPointerException	Invalid use of a null reference.
NumberFormatException	Konversi tidak sah dari string ke sebuah format numeric.
ClassNotFoundException	Class tidak ditemukan.
CloneNotSupportedException	Berusaha untuk mengklonig sebuah objek yang tidak menerapkan Cloneable interface.
IllegalAccessException	Akses ke sebuah class tidak sah.
InstantiationException	Usaha untuk membuat sebuah objek dari abstract class atau interface.
InputMismatchException	Variabel yang diinputkan tidak sesuai dengan tipe data yang dideklarasikan.
SQLException	Terjadi ketika ada kesalahan dalam pengasesan database menggunakan sintak SQL.

8. Contoh Studi Kasus dan Penyelesaian

Berikut contoh program untuk mencari hasil pembagian. Dengan inputan pembilang 12 dan penyebut 0.

```
import java.util.Scanner;

public class Bagi {
    public static int bagi( int pm, int py){
        return pm / py;}
}
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner( System.in );
    System.out.print( "Masukkan nilai pembilang: " );
    int pembilang = scanner.nextInt();
    System.out.print( "Masukkan nilai penyebut: " );
    int penyebut = scanner.nextInt();
    int hasil = bagi(pembilang, penyebut);
    System.out.printf("\nHasil: %d / %d = %d\n", pembilang, penyebut, hasil);
}
}

```

Dari program diatas akan menghasilkan outputan sebagai berikut :

```

Masukkan nilai pembilang: 12
Masukkan nilai penyebut: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at oop.Bagi.bagi(Bagi.java:11)
    at oop.Bagi.main(Bagi.java:19)
Java Result: 1

```

Dari outputan tersebut terdapat eror yang ditunjukkan pada bagian main yaitu `java.lang.AritmeticException: / by zero` atau tidak diperbolehkan untuk melakukan pembagian dengan angka 0. Oleh karenanya diperlukan exception untuk menangani hal tersebut.

Berikut program untuk menanganinya:

```

import java.util.Scanner;

public class Bagi{
    public static int bagi(int pm, int py) throws ArithmeticException {
        return pm / py;}
    public static void main( String args[] ){
        Scanner scanner = new Scanner( System.in ); // untuk input
        try {
            System.out.print("Masukkan nilai pembilang: ");
            int pembilang = scanner.nextInt();

```

```

        System.out.print("Masukkan nilai penyebut: ");
        int penyebut = scanner.nextInt();
        int hasil = bagi( pembilang, penyebut );
        System.out.println("Hasil:" + (pembilang + "/" + penyebut + "=" + hasil));
    } // akhir blok try
    catch ( ArithmeticException e){
        System.err.printf("\nException: %s\n", e);
        System.out.println("Nol adalah penyebut yang tidak sah.\n");
    } // akhir blok catch
} // akhir method main
} // akhir kelas Bagi

```

Berikut hasil outputnya :

```

Masukkan nilai pembilang: 12
Masukkan nilai penyebut: 0

Nol adalah penyebut yang tidak sah.
Exception: java.lang.ArithmeticException: / by zero

```


JURNAL MODUL 6

Soal 1

Buatlah Exception Handling pada tampilan Aplikasi Rental Mobil.

Rincian jenis Exception :

Menu Rental Mobil



Rental Mobil- Peminjaman Mobil

Menu Peminjaman

ID Mobil

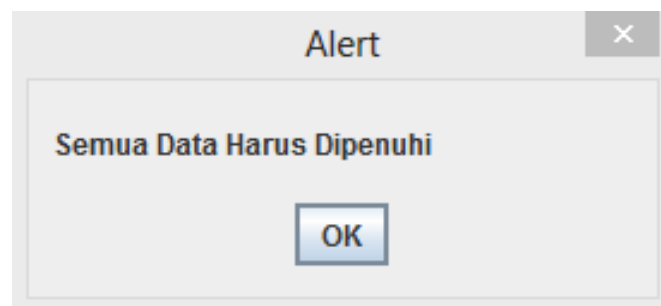
ID Peminjaman

Nama Peminjam


Tanggal Pinjam

Beranda Submit

Setiap jTextField harus diisi, jika ada bagian yang kosong, maka akan keluar notif :



Menu Tambah Mobil



Rental Mobil- Tambah Mobil

ID Mobil

Tanggal Masuk

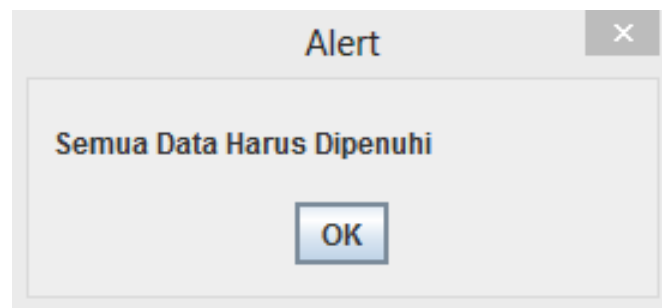
Merek Mobil

Jumlah Mobil

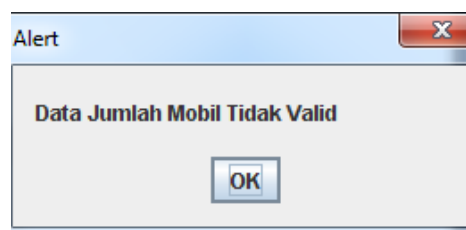
Beranda Submit

Tambahkan JLabel serta JTextField Jumlah Mobil.

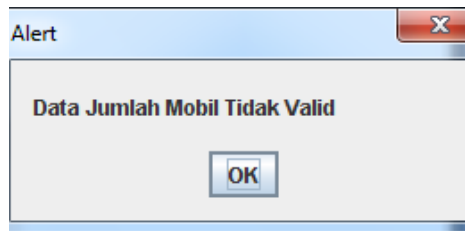
Setiap jTextField harus diisi, jika ada bagian yang kosong, maka akan keluar notif



Saat Jumlah mobil diisikan nilai selain tipe data integer, maka akan keluar pemberitahuan seperti berikut



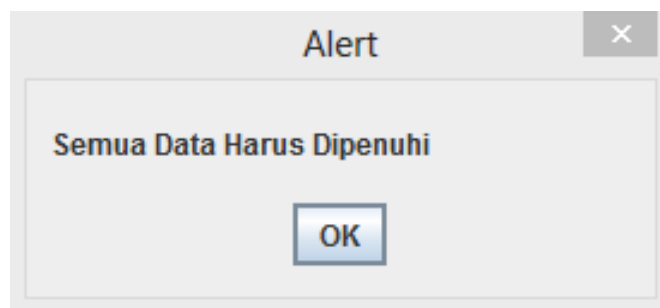
Jumlah Mobil harus satu atau lebih. Jika tidak maka akan keluar pemberitahuan seperti berikut



Menu Manage User

A screenshot of a software window titled 'Rental Mobil- Manage Member'. The window has a light gray background and a title bar with standard Windows controls. The main content area is titled 'Manage User'. It contains three input fields: 'ID Member', 'Nama Member', and 'Jenis Kelamin'. The 'Jenis Kelamin' field has two radio buttons labeled 'Laki - Laki' and 'Perempuan'. At the bottom of the window, there are two buttons: 'Beranda' on the left and 'Tambah User' on the right.

Setiap component harus diisi, jika ada bagian yang kosong, maka akan keluar pemberitahuan seperti berikut



Soal 2

Buatlah GUI dengan mengimplementasikan handling exception `ArithmeticException`, `NegativeArraySizeException`, `ArrayIndexOutOfBoundsException`, dan `NullPointerException`

