

# MODUL PRAKTIKUM

## **Pemrograman Berorientasi Objek**



*Disusun Oleh :*

Dr. Budi Setiyono, MT  
Dr. Dwi Ratna S, MT  
Asisten PBO

**DEPARTEMEN MATEMATIKA FSAD  
INSTITUT TEKNOLOGI SEPULUH NOPEMBER  
SURABAYA**

# DAFTAR ISI

**Modul 1 : Class dan Object**

**Modul 2 : Encapsulation / Enkapsulasi**

**Modul 3 : Inheritance / Pewarisan**

**Modul 4 : Polymorphism**

**Modul 5 : Penanganan Ekspansi**

**Modul 7 : GUI dan Penanganan Kejadian**

# MODUL 1

## CLASS DAN OBJEK

### A. TUJUAN PEMBELAJARAN

1. Memahami konsep pemrograman berorientasi objek
2. Memahami konsep class dan objek
3. Memahami Methods

### B. DASAR TEORI

OOP (*Object Oriented Programming*) merupakan teknik membuat suatu program berdasarkan objek dan apa yang bisa dilakukan objek tersebut. OOP terdiri dari objek-objek yang berinteraksi satu sama lain untuk menyelesaikan sebuah tugas. Kode-kode di-breakdown agar lebih mudah di-manage. *Breakdown* berdasarkan objek-objek yang ada pada program tersebut.

Kelas dapat didefinisikan sebagai cetak biru (*blueprint*) atau *prototype*/kerangka yang mendefinisikan variabel-variabel (data) dan *method-method* (perilaku) umum dari sebuah objek tertentu. Sebagai contoh, kita ambil objek *Mahasiswa*. *Mahasiswa* memiliki data seperti *nim*, *nama*, *alamat*, *IPK*, *jenis kelamin*, *jurusan*, dan sebagainya. Selain data atau ciri-ciri fisik tersebut, *mahasiswa* juga memiliki perilaku-perilaku spesifik yang dapat membedakan antara mahasiswa yang satu dengan yang lainnya, seperti *cara presentasi*, *cara belajar*, *cara mengerjakan tugas* dan sebagainya.

Dalam dunia pemrograman, sebenarnya kelas tidak jauh berbeda dengan tipe data sederhana seperti *integer*, *char*, *boolean*, dan sebagainya. Perbedaannya, tipe data sederhana digunakan untuk mendeklarasikan variabel 'normal', sedangkan kelas digunakan untuk mendeklarasikan sebuah variabel yang berupa objek. Variabel yang berupa objek ini sering disebut dengan referensi objek (*object reference*).

#### Mendefinisikan Kelas

Dalam Java, kelas didefinisikan dengan menggunakan kata kunci *class*. Berikut ini bentuk umum yang digunakan untuk mendefinisikan sebuah kelas.

```
class NamaKelas {  
    tipe data1;  
    tipe data2;  
  
    ...  
}
```

```

    tipe dataN;

    tipe method1(daftar-parameter) {
        //kode untuk method1
    }

    tipe method2(daftar-parameter) {
        //kode untuk method2
    }

    ...

    tipe methodN(daftar-parameter) {
        //kode untuk methodN
    }
}

```

Data atau variabel yang didefinisikan di dalam sebuah kelas sering disebut dengan *instance variable* yang selanjutnya akan diakses melalui *method-method* yang ada. Data dan *method* yang tergabung dalam suatu kelas sering dinamakan sebagai *class members*.

### Contoh Kelas Sederhana

Pada bagian ini akan diberikan pembahasan terkait dengan pembuatan sebuah kelas sederhana. Di sini kita akan membuat kelas *Karyawan*, yang memiliki data-data: *ID*, *nama*, *divisi*, dan *gaji*. Untuk saat ini, belum ditambahkan *method* ke dalam kelas tersebut. *Method* akan dibahas terpisah pada bagian selanjutnya dalam bab ini.

Perhatikan kode berikut ini

#### Program 6.1 Contoh pembuatan kelas.

```

Public class Karyawan
{
    String ID,nama,divisi;
    Double gaji;
}

```

Melalui kode di atas, berarti kita telah mendefinisikan sebuah tipe data baru dengan nama *Karyawan*. Penting untuk diingat bahwa pendefinisian kelas hanya akan membuat sebuah pola atau *template*, bukan membuat objek. Objek aktual dari kelas tersebut harus dibuat sendiri melalui kode berikut:

```

//membuat objek karyawan dengan nama Aurel
Karyawan Aurel = new Karyawan();

```

Di sini, *Karyawan* adalah kelas dan *Aurel* adalah objek atau *instance* dari kelas *Karyawan*. Melalui objek *Aurel*, kita dapat mengakses dan memanipulasi data-data yang terdapat pada kelas *Karyawan*, dengan cara menggunakan operator titik (.), seperti yang tampak pada kode di bawah ini.

```
Aurel.ID = "K001";  
Aurel.divisi = "Aurel DIan";  
Aurel.nama = "Marketing";  
Aurel.gaji= "2500000";
```

kode tersebut digunakan untuk mengisi nilai ke dalam data *ID*, *nama*, *divisi* dan *gaji* yang dimiliki oleh objek *Aurel* masing-masing dengan nilai “K001”, “Aurelia Dian”, “Marketing” dan 2500000.

### Mengisi Nilai pada Referensi Objek

Terdapat satu buah catatan penting yang perlu diperhatikan pada saat kita memasukkan nilai pada sebuah variabel referensi. Sebelumnya, perhatikan terlebih dahulu kode berikut:

```
Karyawan Karyawan001, Karyawan002;  
Karyawan001 = new Karyawan();  
Karyawan002 = Karyawan001;
```

Baris pertama digunakan untuk mendeklarasikan variabel referensi ke objek *Karyawan* dengan nama *karyawan001* dan *karyawan002*. Baris kedua berfungsi untuk membuat objek *Karyawan* dan menyimpan referensinya ke dalam variabel *karyawan001*. Dan baris ketiga digunakan memasukkan *karyawan001* ke dalam *karyawan002*. Ini artinya, variabel *karyawan002* berperan sebagai referensi ke objek yang sedang ditunjuk oleh *karyawan001*. Dengan demikian, variabel *karyawan001* dan *karyawan002* masing-masing menunjuk ke objek *Karyawan* yang sama

### Method

Dalam Java, kelas berisi kumpulan data dan *method*, yang selanjutnya akan saling bekerja sama dalam melakukan tugas-tugas spesifik tertentu sesuai dengan perilaku objek yang dimodelkan.

Berikut ini bentuk umum dari pembuatan *method* di dalam kelas.

```
tipe namaMethod(daftar-parameter) {  
  //kode yang akan dituliskan  
}
```

Pada bentuk umum di atas, tipe adalah tipe data yang akan dikembalikan oleh *method*. Sebagai catatan, dalam Java *method* terbagi menjadi dua: *void* dan *non-void*. *Method void* adalah *method* yang tidak mengembalikan nilai, sedangkan *method non-void* adalah *method* yang mengembalikan nilai. Jika *method* yang kita buat ditujukan untuk mengembalikan suatu nilai tertentu, maka di dalam *method* tersebut harus terdapat statemen *return*, yang diikuti dengan nilai yang akan dikembalikan, seperti berikut:

```
return nilaiKembalian;
```

*nilaiKembalian* dapat berupa konstanta maupun variabel, yang digunakan untuk menandakan nilai yang akan dikembalikan oleh *method*.

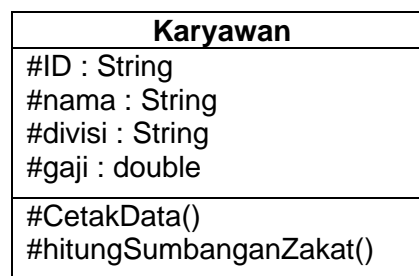
### UML Class diagram

UML (Unified Modeling Language) merupakan diagram yang biasa digunakan untuk mengilustrasikan sebuah kelas dan sekaligus hubungan antar kelas. Didalam satu class diagram terdapat tiga row atau baris dimana masing-masing row memiliki kegunaan yang berbeda-beda diantaranya :

1. Baris Pertama menunjukkan nama class
2. Baris kedua digunakan untuk menyatakan variabel yang terdapat pada class, dan
3. Baris ketiga digunakan untuk menyatakan method-method yang ada pada suatu class

Selain itu UML juga bisa digunakan untuk memberikan indikasi tentang akses modifier dari sebuah variabel atau method antara lain :

1. + : Tanda ini menunjukkan bahwa suatu variabel atau method memiliki akses modifier **public**
2. - : Tanda ini menunjukkan bahwa suatu variabel atau method memiliki akses modifier **private**
3. # : Tanda ini menunjukkan bahwa suatu variabel atau method memiliki akses modifier **protected**



Gambar 1: Class Diagram

Akses modifier akan kita pelajari lebih dalam pada modul 2.

## C. PERCOBAAN

### 1. Pembuatan class

```
Public class Karyawan
{
    String ID,nama,divisi;
    Double gaji;
}
```

### 2. Instansiasi objek dan pengaksesan data pada objek

```
public class ImplementasiKelasKaryawan
{
    public static void main(String[] args)
    {
        //membuat objek karyawan dengan nama Aurel
        Karyawan Aurel = new Karyawan();

        //mengisi nilai kedalam data-data Objek Karyawan
        Aurel.ID = "K001";
        Aurel.divisi = "Aurel DIan";
        Aurel.nama = "Marketing";
        Aurel.gaji = "2500000";

        //mencetak data-data object karyawan
        System.out.println("Data Karyawan");
        System.out.println("ID      : " + Aurel.ID);
        System.out.println("Nama    : " + Aurel.nama);
        System.out.println("Divisi   : " + Aurel.divisi);
        System.out.println("Gaji     : " + Aurel.gaji);
    }
}
```

### 3. Mengisi Nilai pada Referensi Objek

```
public class PengaruhReferensiObjek1 {

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

```

/*Instansiasi 2 objek referensi yang mengacu pada 1 objek karyawan */
Karyawan Karyawan001 = new Karyawan();
Karyawan Karyawan002 = Karyawan001;

//mengisi data Objek Karyawan melalui objek referensi 1
Karyawan001.nama = "Mischella";
Karyawan001.divisi = "HRD";

//mencetak data object karyawan yang di acu 2 objek referensi
System.out.println("Data Karyawan001");
System.out.println("Nama   : " + Karyawan001.nama);
System.out.println("Divisi  : " + Karyawan001.divisi);
System.out.println(" ");
System.out.println("Data Karyawan002");
System.out.println("Nama   : " + Karyawan002.nama);
System.out.println("Divisi  : " + Karyawan002.divisi);
}
}

```

#### 4. Methods

*Penambahan method pada deklarasi kelas Karyawan*

```

public class Karyawan {

    String ID, nama, divisi;
    double gaji;
    //method void
    void cetakData() {
        System.out.println("Data Karyawan : ");
        System.out.println("ID       : " + ID);
        System.out.println("Nama    : " + nama);
        System.out.println("Divisi  : " + divisi);
        System.out.println("Gaji    : " + gaji);
    }

    //method non void
    double hitungSumbanganZakat() {
        double zakat = gaji * 0.025;
        return zakat;
    }
}

```

*Contoh pemanggilan method*

```

public class ImplementasiMethodNonVoid {

    public static void main(String[] args) {
        //instantiasi objek Karyawan
        Karyawan karyawan001 = new Karyawan();
    }
}

```



```

        //mengisi data pada objek Karyawan
        karyawan001.ID = "K001";
        karyawan001.nama = "Agus Ramadhan";
        karyawan001.divisi = "Keuangan";
        karyawan001.gaji = 1850000;

        //memanggil method cetakData()
        karyawan001.cetakData();

        //memanggil method hitungSumbanganZakat()
        System.out.println("Sumbangan Zakat : " +
        karyawan001.hitungSumbanganZakat());
    }
}

```

#### D. LATIHAN

1. Silahkan modifikasi kelas Karyawan dan buatlah 3 objek baru di dalamnya.
2. Tambahkan method untuk mengisi data karyawan, setID, setName, setDevisi dan setGaji
3. Tambahkan method untuk mengambil data karyawan, getID, getName, getDevisi dan getGaji

#### E. TUGAS

##### **Tugas 1 : Menganalisa, membuat UML class diagram dan implementasi program**

Seorang pengusaha rental mobil kesulitan mengingat armada kendaraan yang memilikinya. Oleh karena itu pengusaha tersebut menugaskan pegawainya untuk mengidentifikasi tersebut. Hasil identifikasi dicatat dalam suatu table sebagaimana bisa dilihat pada Tabel1. Sayangnya karena merupakan pegawai baru maka ia tidak memahami nama hal yang diidentifikasi (A,B,C, D, dan E).

- a. Bantulah pegawai tersebut dalam menentukan nama hal yang diidentifikasi (A,B,C, D, dan E).
- b. Bantulah pengusaha tersebut dalam membuat UML class diagram Mobil. Tambahkan method infoMobil() yang bertujuan untuk menampilkan semua karakteristik mobil (A,B,C, D, dan E).
- c. Buatlah kelas Mobil.java yang mengimplementasikan desain UML class diagram anda!
- d. Buatlah kelas TesMobil.java yang berisi pembuatan 4 (empat) buah obyek bernama mobil1, mobil2, mobil3, mobil4. Mengeset karakteristik masing-masing dan menampilkan info karakteristik mobil.

Tabel 1. Data karakteristik mobil

Obyek	A	B	C	D	E
mobil1	Toyota	Biru	minibus	2000	7
mobil2	Daihatsu	Hitam	pick up	1500	2
mobil3	Suzuki	Silver	suv	1800	5
mobil4	Honda	Merah	sedan	1300	5

# MODUL 2

## ENKAPSULASI

### A. TUJUAN PEMBELAJARAN

1. Menerapkan konsep enkapsulasi pada class
2. Mendeklarasikan suatu konstruktor
3. Memahami access modifier

### B. DASAR TEORI

#### Enkapsulasi

Kita dapat menyembunyikan information dari suatu class sehingga anggota-anggota class tersebut tidak dapat diakses dari luar. Adapun caranya adalah cukup dengan memberikan akses kontrol private ketika mendeklarasikan suatu atribut atau method. Contoh:

```
private int nrp;
```

Encapsulation (Enkapsulasi) adalah suatu cara untuk menyembunyikan implementasi detail dari suatu class. Enkapsulasi mempunyai dua hal mendasar, yaitu :

1. information hiding
2. menyediakan suatu perantara (method) untuk pengaksesan data

```
Contoh: public class Siswa {  
    private int nrp;  
    public void setNrp(int n) {  
        nrp=n;  
    }  
}
```

#### Constructor (konstruktor)

Konstruktor adalah metode yang dapat digunakan untuk memberikan nilai awal saat objek diciptakan. Metode ini akan dipanggil secara otomatis oleh java ketika new dipakai untuk menciptakan objek dari suatu kelas.

Konstruktor memiliki beberapa karakteristik antara lain :

1. Nama constructor sama dengan nama kelas
2. Tidak memiliki nilai balik atau return value

Konstruktor memiliki beberapa tipe diantaranya :

1. Default constructor

Default constructor merupakan konstruktor yang secara otomatis dibuatkan oleh java compiler, sehingga meskipun kita tidak menuliskan atau membuat constructor secara eksplisit java sudah menyediakannya. Tujuan sebenarnya dari default constructor adalah untuk memberikan nilai awal pada objek seperti 0, NULL, dan sebagainya.

2. Parameterized constructor

Parameterized constructor sebenarnya merupakan method, yang mana sebuah method bisa memiliki beberapa parameter atau bahkan tidak memiliki parameter (default constructor). Constructor bisa memiliki beberapa parameter sebagai penentu nilai awal dari sebuah objek. Seperti layaknya method, parameter di dalam constructor juga harus memiliki tipe data tertentu. Tipe data disini bisa berupa tipe data reference ataupun tipe data primitive.

### Modifier

Modifier merupakan bentuk pengimplementasian konsep enkapsulasi. Dengan adanya modifier maka class, interface, method, dan variabel akan terkena suatu dampak tertentu.

**Tabel 2.1** Kelompok Modifier

Kelompok Modifier	Berlaku Untuk			Meliputi
	Class	Method	Variabel	
Access modifier	√	√	√	public, protected, private, dan friendly (default/ tak ada modifier).
Final modifier	√	√	√	final
Static modifier		√	√	static
Abstract modifier	√	√		abstract

**Tabel 2.2** Karakteristik Access Modifier

Modifier	Class dan Interface	Method dan Variabel
<b>Default (tak ada modifier ) Friendly</b>	Dikenali di paketnya	Diwarisi subclass di paket yang sama dengan superclassnya. Dapat diakses oleh method-method di class-class yang sepaket.
<b>Public</b>	Dikenali di manapun	Diwarisi oleh semua subclassnya. Dapat diakses dimanapun.
<b>Protected</b>	Tidak dapat diterapkan	Diwarisi oleh semua subclassnya. Dapat diakses oleh method-method di class-class yang sepaket.
<b>Private</b>	Tidak dapat diterapkan	Tidak diwarisi oleh subclassnya Tidak dapat diakses oleh class lain.

**Tabel 2.3** Karakteristik Permitted Modifier

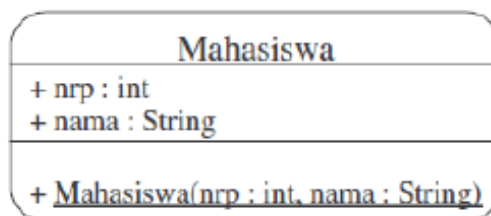
Modifier	Class	Interface	Method	Variabel
<b>Abstract</b>	Class dapat berisi method abstract. Class tidak dapat diinstantiasi Tidak mempunyai constructor	Optional untuk dituliskan di interface karena interface secara inheren adalah abstract.	Tidak ada method body yang didefinisikan. Method memerlukan class kongkrit yang merupakan subclass yang akan mengimplementasikan method abstract.	Tidak dapat diterapkan.
<b>Final</b>	Class tidak dapat diturunkan.	Tidak dapat diterapkan.	Method tidak dapat ditimpa oleh method di subclass-subclassnya	Berperilaku sebagai konstanta

<b>Static</b>	Tidak dapat diterapkan.	Tidak dapat diterapkan.	Mendefinisikan method (milik) class. Tidak memerlukan instant object untuk menjalankannya. Method ini tidak dapat menjalankan method yang bukan static serta tidak dapat mengacu variable yang bukan static.	Mendefinisikan variable milik class. Tidak memerlukan instant object untuk mengacunya. Variabel ini dapat digunakan bersama oleh semua instant objek.
---------------	-------------------------	-------------------------	--	---

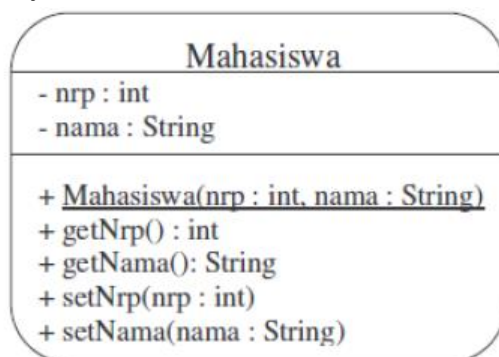
## C. PERCOBAAN

### 1. Melakukan enkapsulasi pada suatu class

Implementasikan UML class diagram Mahasiswa sebelum dan setelah dilakukan enkapsulasi!



Jika enkapsulasi dilakukan pada class diagram diatas, maka akan berubah menjadi:



### 2. Konstruktor Default

```
public class Konstruktor {  
    public Konstruktor(){  
        System.out.println("default konstruktor");  
    }  
}
```

```
public class AksesKonstruktor {  
    public static void main(String[] args) {  
        Konstruktor konstruktor = new Konstruktor();  
    }  
}
```

### 3. Konstruktor Parameter

```
public class ParamConstructor {  
    String nama;  
    int semester;  
    String nim;  
    public ParamConstructor(String namaKu, int semesterKu, String  
        nimKu){  
        this.nama = namaKu;  
        this.nim = nimKu;  
        this.semester = semesterKu;  
    }  
    public void infoKu(){  
        System.out.println("Nama : "+nama +"\nNim : "+ nim  
            +"\nSemester : "+ semester);  
    }  
}
```

## D. LATIHAN

1. Buatlah class **Buku** dengan variabel String namaPengarang, String judulBuku, int tahunTerbit, int cetakanKe dan double hargaJual. Selanjutnya buatlah constructornya serta sebuah method untuk menampilkan informasi buku.

## E. TUGAS

Buatlah sebuah class Pelajar dengan ketentuan sebagai berikut:

- memiliki atribut : nip, nama, nilaiUjian1, nilaiUjian2, nilaiTugas
- memiliki method nilaiAkhir untuk menghitung nilai akhir dari pelajar tersebut dimana rumusnya adalah :
- Ada method isLulus yang digunakan untuk mengecek apakah seorang siswa lulus atau tidak, dimana dinyatakan lulus bila nilai akhirnya sama dengan 60 keatas. Dan ada pula method status yang digunakan untuk menampilkan status lulus atau tidaknya pelajar tersebut.

# MODUL 3

## PEWARISAN (INHERITANCE)

### A. TUJUAN PEMBELAJARAN

1. mampu menerapkan konsep pewarisan
2. mampu menerapkan overloading dan overriding
3. mampu menterjemahkan konsep pewarisan dalam UML.

### B. DASAR TEORI

#### Pewarisan

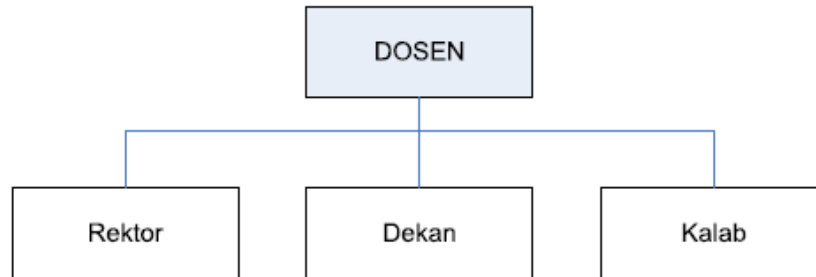
Pewarisan (inheritance) adalah suatu cara pembuatan class baru dengan menggunakan kembali class yang sudah didefinisikan sebelumnya dengan menambahkan atribut dan method baru. Sehingga dengan demikian class baru tersebut tetap memiliki variabel dan fungsi yang dimiliki oleh class sebelumnya. Pada konsep pewarisan ada beberapa istilah yang perlu diketahui, yaitu:

- Sub class, digunakan untuk menunjukkan class anak atau turunan secara hirarkis dari super class.
- Super class, digunakan untuk menunjukkan class induk secara hirarkis dari sub class (class anak).

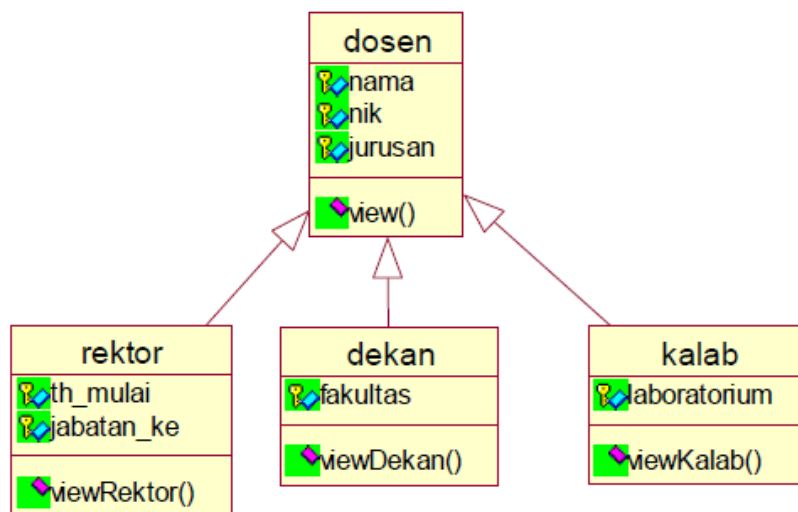


- Extends, digunakan untuk menunjukkan bahwa suatu class merupakan turunan dari class lain. Misal A extends B, berarti class A adalah turunan dari class B. A merupakan subclass, sedangkan B merupakan superclass.
- Super, digunakan untuk memanggil konstruktor dari super class atau memanggil variabel yang mengacu pada super class. Misal super(x,y,z), berarti atribut x, y, dan z diambil dari atribut pada class induk.

**Gambar 3.1** menunjukkan hirarki klas dosen. Klas dosen tersebut memiliki turunan berupa klas rektor, klas dekan dan klas kalab. Apabila dirancang ke dalam diagram class, akan nampak seperti pada **Gambar 3.2**.



**Gambar 3.1** Hirarki klas dosen.



**Gambar 3.2** Class diagram untuk hirarki dosen.

Pada **Gambar 3.2** tersebut, class induk (class dosen) memiliki atribut nama, nik dan jurusan. Method yang dimiliki oleh class dosen adalah view(). Class turunan dari class dosen ada tiga class. Pada class rektor, terdapat tambahan atribut berupa th\_mulai dan jabatan\_ke, serta method viewRektor(). Pada class dekan terdapat tambahan atribut fakultas, dan method viewDekan(). Pada class kalab terdapat tambahan atribut laboratorium, dan method viewKalab(). Pendefinisian class dosen seperti terlihat pada **Gambar 3.3**.

```

1 // Class dosen sebagai super class
2 package dosen_ui_v1;
3
4 public class dosen {
5     protected String nama;
6     protected String nik;
7     protected String jurusan;
8
9     //Membuat konstruktor
10    dosen (String namaX, String nikX, String jurX)
11    {
12        nama = namaX;
13        nik = nikX;
14        jurusan = jurX;
15    }
16
17    //Menampilkan informasi
18    public void view()
19    {
20        System.out.println("Nama : "+nama);
21        System.out.println("NIK : "+nik);
22        System.out.println("Jurusan: "+jurusan);
23    }
24 }

```

Konstruktor untuk class dosen

**Gambar 3.3** Pendefinisian class dosen pada Contoh 7.1.

Selanjutnya pendefinisian class rektor, dekan, dan kalab seperti terlihat pada **Gambar 3.4 – 3.6**.

```

1 package dosen_ui_v1;
2
3 public class rektor extends dosen {
4     private int th_mulai;
5     private int jabatan_ke;
6
7     //Inisialisasi
8     rektor(String namaX, String nikX, String jurX, int thX, int keX)
9     {
10        super(namaX, nikX, jurX);
11        th_mulai = thX;
12        jabatan_ke = keX;
13    }
14
15    //Menampilkan informasi
16    public void viewRektor()
17    {
18        System.out.println("Th mulai jabatan : "+th_mulai);
19        System.out.println("Jabatan rektor ke- : "+jabatan_ke);
20    }
21 }

```

Memanggil variabel yang mengacu pada dosen (super class)

**Gambar 3.4** Pendefinisian class rektor Contoh 7.1.

```

1 package dosen_uui_v1;
2
3 public class dekan extends dosen {
4     private String fakultas;
5
6     //Inisialisasi
7     dekan(String namaX, String nikX, String jurX, String fakX)
8     {
9         super(namaX, nikX, jurX);
10        fakultas = fakX;
11    }
12
13    //Menampilkan informasi
14    public void viewDekan()
15    {
16        System.out.println("Fakultas : "+fakultas);
17    }
18 }

```

**Gambar 3.5** Pendefinisian class dekan Contoh 7.1.

```

1 package dosen_uui_v1;
2
3 public class kalab extends dosen {
4     private String laboratorium;
5
6     //Inisialisasi
7     kalab(String namaX, String nikX, String jurX, String labX)
8     {
9         super(namaX, nikX, jurX);
10        laboratorium = labX;
11    }
12
13    //Menampilkan informasi
14    public void viewKalab()
15    {
16        System.out.println("Laboratorium : "+laboratorium);
17    }
18 }

```

**Gambar 3.6** Pendefinisian class kalab Contoh 7.1.

Program utama untuk pemanggilan fungsi sebagaimana terlihat pada **Gambar 3.7**.

```

1 /* contoh pewarisan pada class dosen
2 */
3
4 package dosen_uui_v1;
5
6 public class Main {
7
8     public static void main(String[] args) {
9         rektor rek = new rektor("Andi", "885230202", "Informatika", 2006, 2);
10        dekan dek = new dekan("Ahmad", "995230101", "T. Kimia", "TI");
11        kalab lab = new kalab("Indah", "035230302", "Informatika", "KSC");
12
13        rek.view();
14        rek.viewRektor();
15
16        dek.view();
17        dek.viewDekan();
18
19        lab.view();
20        lab.viewKalab();
21    }
22 }

```

**Gambar 3.7** Program utama menunjukkan pewarisan klas dosen.

## Overriding

Overriding adalah suatu cara untuk mendefinisikan ulang method yang ada pada class induk apabila class anak menginginkan adanya informasi yang lain. Overriding dilakukan dengan cara menulis ulang method yang ada pada class induk dengan syarat bahwa nama dan parameter fungsi tersebut harus sama (tidak boleh diubah). Meskipun fungsi telah ditulis ulang oleh class anak, fungsi yang asli pada class induk masih dapat dipanggil di class anak dengan menggunakan class super.

### Contoh 3.2

Konsep pewarisan pada contoh 3.1 dapat dipadukan dengan konsep overriding. **Gambar 3.8** menunjukkan class dosen yang tidak jauh berbeda dengan class dosen pada contoh 3.1.

```
1  /* Contoh penggunaan overriding
2  */
3
4  package dosen_uui_v2;
5
6  public class dosen {
7      protected String nama;
8      protected String nik;
9      protected String jurusan;
10
11      //Inisialisasi
12      dosen (String namaX, String nikX, String jurX)
13      {
14          nama    = namaX;
15          nik     = nikX;
16          jurusan = jurX;
17      }
18
19      //Menampilkan informasi
20      public void view()
21      {
22          System.out.println("Nama    : "+nama);
23          System.out.println("NIK     : "+nik);
24          System.out.println("Jurusan: "+jurusan);
25      }
26  }
```

Method yang akan dilakukan

**Gambar 3.8** Pendefinisian class dosen pada Contoh 7.1.

Konsep overriding digunakan pada method view(). Method ini ditulis ulang pada setiap subclass yaitu subclass rektor, dekan, dan kalab dengan menambahkan instruksi tambahan yang dibutuhkan pada setiap class. **Gambar 3.9 – 3.11** menunjukkan class tersebut.

```

1  package dosen_uui_v2;
2
3  public class rektor extends dosen {
4      private int th_mulai;
5      private int jabatan_ke;
6
7      //Inisialisasi
8      rektor(String namaX, String nikX, String jurX, int thX, int keX)
9      {
10         super(namaX, nikX, jurX);
11         th_mulai = thX;
12         jabatan_ke = keX;
13     }
14
15     //Menampilkan informasi
16     public void view()
17     {
18         super.view();
19         System.out.println("Th mulai jabatan : "+th_mulai);
20         System.out.println("Jabatan rektor ke- : "+jabatan_ke);
21     }
22 }

```

Overriding method view() pada super class dosen

**Gambar 3.9** Pendefinisian class rektor Contoh 3.2.

```

1  package dosen_uui_v2;
2
3  public class dekan extends dosen {
4      private String fakultas;
5
6      //Inisialisasi
7      dekan(String namaX, String nikX, String jurX, String fakX)
8      {
9         super(namaX, nikX, jurX);
10         fakultas = fakX;
11     }
12
13     //Menampilkan informasi
14     public void view()
15     {
16         super.view();
17         System.out.println("Fakultas : "+fakultas);
18     }
19 }

```

**Gambar 3.10** Pendefinisian class dekan Contoh 3.2.

```

1  package dosen_uui_v2;
2
3  public class kalab extends dosen {
4      private String laboratorium;
5
6      //Inisialisasi
7      kalab(String namaX, String nikX, String jurX, String labX)
8      {
9         super(namaX, nikX, jurX);
10         laboratorium = labX;
11     }
12
13     //Menampilkan informasi
14     public void view()
15     {
16         super.view();
17         System.out.println("Laboratorium : "+laboratorium);
18     }
19 }

```

**Gambar 3.11** Pendefinisian class kalab Contoh 3.2.

Untuk melakukan pemanggilan, program utama diberikan pada **Gambar 3.12**.

```
1  /* contoh overriding
2  */
3
4  package dosen_ui_v2;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          dosen P;
10         rektor rek = new rektor("Andi", "885230202", "Informatika", 2006, 2);
11         dekan dek = new dekan("Ahmad", "995230101", "T. Kimia", "TI");
12         kalab lab = new kalab("Indah", "035230302", "Informatika", "KSC");
13
14         P = rek;
15         P.view();
16         P = dek;
17         P.view();
18         P = lab;
19         P.view();
20     }
21 }
```

Method pada sub class  
sebagai hasil overriding

**Gambar 3.12** Program utama menunjukkan konsep overriding klas dosen.

### Overloading

Overloading fungsi adalah penulisan beberapa fungsi (dua atau lebih) yang memiliki nama yang sama. Pada bahasan overloading dikenal istilah signature. Signature sebuah fungsi adalah parameter lengkap dengan tipe datanya yang terdapat dalam fungsi tersebut. Misal terdapat fungsi:

```
public int jumlahMahasiswa (int laki2, int
perempuan,
String kelas);
```

maka signature dari fungsi tersebut adalah (int, int, String).

Suatu fungsi dikatakan di-overload manakala terdapat beberapa fungsi dengan nama yang sama namun memiliki signature yang berbeda-beda, sebagai contoh:

```
public void infoMahasiswa (int laki2, int
perempuan, String
kelas)
{
...
}
public void infoMahasiswa (int mhsLama, int
mhsBaru, int
mhsCutu, int angkatan)
{
...
}
```

Contoh 3.3:

Berikut terdapat class mahasiswa (**Gambar 3.13**). Pada class tersebut terdapat dua subclass dengan nama yang sama yaitu infoMahasiswa namun memiliki signature yang berbeda. Subclass pertama digunakan untuk menghitung jumlah mahasiswa kelas A angkatan 2008. Sedangkan pada subclass kedua digunakan untuk menghitung jumlah mahasiswa aktif sampai tahun 2008.

```

1 package informasimhs;
2
3 public class mahasiswa {
4
5     //Informasi tentang mahasiswa Kelas A
6     public void infoMahasiswa (int laki2, int perempuan, String kelas)
7     {
8         int jumlah = laki2 + perempuan;
9         System.out.println(kelas+" jumlah mahasiswa = "+jumlah);
10    }
11
12    //Informasi tentang mahasiswa sampai tahun 2008
13    public void infoMahasiswa (int mhsLama, int mhsBaru, int mhsCuti, int angkatan)
14    {
15        int jumlah = mhsLama + mhsBaru + mhsCuti;
16        System.out.println("Sampai tahun "+angkatan+" jumlah mahasiswa = "+jumlah);
17    }
18 }

```

**Gambar 3.13** Dua class bernama infoMahasiswa dengan signature berbeda.

Pada program utama dilakukan pemanggilan terhadap kedua class tersebut, masing-masing dengan parameter yang bersesuaian (**Gambar 3.14**).

```

1 /* Contoh overloading
2 */
3
4 package informasimhs;
5
6 public class Main {
7
8     public static void main(String[] args) {
9
10        mahasiswa M = new mahasiswa();
11        M.infoMahasiswa(60, 18, "Kelas A angkatan 2008");
12        M.infoMahasiswa(1000, 400, 25, 2008);
13    }
14 }

```

**Gambar 3.14** Pemanggilan fungsi yang di-overload pada Contoh 3.3.

Overloading fungsi ditentukan oleh signature nya, tidak ditentukan oleh nilai balikan fungsi. Dalam satu class, apabila ada dua atau lebih fungsi yang memiliki nilai balikan yang berbeda namun memiliki signature yang sama, maka fungsi tersebut tidak dapat dikatakan sebagai overloading.

Contoh 3.4:

Berikut terdapat dua class bernama jumlah untuk untuk menjumlahkan dua bilangan. Class jumlah yang pertama memiliki nilai balikan yang bertipe integer untuk signature (int x1, int x2). Sedangkan class jumlah yang kedua memiliki nilai balikan yang bertipe double untuk signature (double y1, double y2). Class yang dibentuk sebagaimana terlihat pada **Gambar 3.15**.

```

1  /* class penjumlahan
2  */
3
4  package jumlahbil;
5
6  public class penjumlahan {
7
8      //Bilangan integer
9      public int jumlah (int x1, int x2)
10     {
11         return(x1+x2);
12     }
13
14     //Bilangan real
15     public double jumlah (double y1, double y2)
16     {
17         return(y1+y2);
18     }
19 }
20

```

**Gambar 3.15** Dua class bernama jumlah dengan signature berbeda.

Pada program utama dilakukan pemanggilan terhadap kedua class tersebut, masing-masing dengan parameter yang bersesuaian (**Gambar 3.16**).

```

1  /* Contoh polimorfisme untuk penjumlahan bilangan
2  */
3
4  package jumlahbil;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          int x1=10, x2=15;
10         double y1=10.5, y2=15.8;
11         penjumlahan P = new penjumlahan();
12         System.out.println(x1+" + "+x2+" = "+P.jumlah(x1,x2));
13         System.out.println(y1+" + "+y2+" = "+P.jumlah(y1,y2));
14     }
15 }

```

**Gambar 3.16** Pemanggilan fungsi yang di-overload padaContoh 7.4.

## C. LATIHAN

Lakukan praktek diatas dan lakukan evaluasi

## D. TUGAS



Buatlah sebuah **class kendaraan** dengan turunannya **kendaraan darat**, **kendaraan laut**, kemudian diturunkan lagi menjadi **sepeda motor** dan **perahu layar**.

# MODUL 4

## POLIMORFISME

### A. TUJUAN PEMBELAJARAN

1. mampu menerapkan konsep polimorfisme dalam pemrograman dengan Java

### B. DASAR TEORI

Polimorfisme digunakan untuk menyatakan suatu nama yang merujuk pada beberapa fungsi yang berbeda (Sinaga, 2004). Pada polimorfisme, rujukan dapat dilakukan pada berbagai tipe objek. Hal ini dilakukan karena setiap objek dimungkinkan memiliki instruksi yang berbeda. Dalam mengimplementasikan polimorfisme, perlu diperhatikan hal-hal sebagai berikut (Rickyanto, 2005):

1. Method yang dipanggil harus melalui variabel dari super class.
2. Method yang dipanggil juga harus merupakan method yang ada pada super class.
3. Signature method harus sama baik yang ada pada super class maupun di subclass.
4. Method access attribute pada subclass tidak boleh lebih terbatas daripada yang ada pada super class.

Contoh 8.1:

Pada **Gambar 4.1** merupakan program untuk membangun class kendaraan. Pada class kendaraan mewaris ke tiga class, yaitu class pesawat, mobil, dan kapal.

**Gambar 4.2 – 4.4** menunjukkan pembentukan ketiga subclass tersebut.

```
1  /* super class kendaraan
2  */
3
4  package transportasi;
5
6  public class kendaraan {
7      private String model;
8
9      //Inisialisasi
10     public kendaraan(String model)
11     {
12         this.model = model;
13     }
14
15     //Informasi yang merupakan method tanpa instruksi
16     public void informasi() {}
17
18 }
```

**Gambar 4.1** Pendefinisian class kendaraan pada Contoh 4.1.

```

1  /* Untuk kelas pesawat
2  */
3  package transportasi;
4
5  public class pesawat extends kendaraan{
6      private String nama;
7      private String jenis;
8
9      public pesawat(String nama)
10     {
11         super("Pesawat");
12         this.nama = nama;
13         jenis = "belum teridentifikasi";
14     }
15
16     public pesawat(String nama, String jenis)
17     {
18         super("Pesawat");
19         this.nama = nama;
20         this.jenis = jenis;
21     }
22
23     public void informasi()
24     {
25         System.out.println("Nama pesawat adalah "+nama);
26         System.out.println("Jenis pesawat adalah "+jenis);
27     }
28 }

```

**Gambar 4.2** Pendefinisian class pesawat Contoh 4.1.

```

1  /* Untuk kelas mobil
2  */
3  package transportasi;
4
5  public class mobil extends kendaraan {
6      private String nama;
7      private String jenis;
8
9      public mobil(String nama)
10     {
11         super("Mobil");
12         this.nama = nama;
13         jenis = "belum teridentifikasi";
14     }
15
16     public mobil(String nama, String jenis)
17     {
18         super("Mobil");
19         this.nama = nama;
20         this.jenis = jenis;
21     }
22
23     public void informasi()
24     {
25         System.out.println("Nama mobil adalah "+nama);
26         System.out.println("Jenis mobil adalah "+jenis);
27     }
28 }

```

**Gambar 4.3** Pendefinisian class mobil Contoh 4.1.

```

1  /* Untuk kelas kapal
2  */
3  package transportasi;
4
5  public class kapal extends kendaraan {
6      private String nama;
7      private String jenis;
8
9      public kapal(String nama)
10     {
11         super("Kapal");
12         this.nama = nama;
13         jenis = "belum teridentifikasi";
14     }
15
16     public kapal(String nama, String jenis)
17     {
18         super("Kapal");
19         this.nama = nama;
20         this.jenis = jenis;
21     }
22
23     public void informasi()
24     {
25         System.out.println("Nama kapal adalah "+nama);
26         System.out.println("Jenis kapal adalah "+jenis);
27     }
28 }

```

**Gambar 4.4** Pendefinisian class kapal Contoh 4.1.

Pada program utama dilakukan pemanggilan terhadap ketiga class tersebut, masing-masing dengan parameter yang bersesuaian (**Gambar 4.5**).

```

1  //Contoh polimorfisme untuk alat2 transportasi
2
3  package transportasi;
4
5  public class Main {
6
7      public static void main(String[] args) {
8          kendaraan P;
9          pesawat psw = new pesawat("Boeing 707", "Pesawat Komersial");
10         mobil mbl1 = new mobil("Toyota Kijang", "Jeep");
11         mobil mbl2 = new mobil("Suzuki Baleno", "Sedan");
12         mobil mbl3 = new mobil("VW Combi");
13         kapal kpl = new kapal("Queen Mary 2", "Kapal Pesiar");
14
15         P = psw;
16         P.informasi();
17         P = mbl1;
18         P.informasi();
19         P = mbl2;
20         P.informasi();
21         P = mbl3;
22         P.informasi();
23         P = kpl;
24         P.informasi();
25     }
26 }

```

**Gambar 4.5** Program utama untuk contoh 4.1.

### **C. LATIHAN**

Lakukan praktek diatas dan lakukan evaluasi

### **E. TUGAS**

Buatlah semua program yang mengimplemtasikan konsep polymorphisme dari sebuah class dengan nama class hewan

# MODUL 5

## PENANGANAN EKSEPSI

### A. TUJUAN PEMBELAJARAN

1. mampu menangani berbagai kesalahan dengan penanganan eksepsi dalam pemrograman dengan Java.

### B. DASAR TEORI

Saat kita membuat program, sebisa mungkin program kita terhindar dari kesalahan. Namun, yang lebih penting adalah bagaimana dalam program kita dapat mengantisipasi kemungkinan munculnya kesalahan pada saat program kita dieksekusi. Java menyediakan sebuah mekanisme penanganan kesalahan yang biasa disebut dengan *exception-handling*. Dalam hal ini setiap kesalahan akan dibentuk menjadi objek.

Dalam Java, *runtime error* (kesalahan-kesalahan yang terjadi pada saat program sedang berjalan) direpresentasikan dengan eksepsi. Eksepsi adalah suatu objek yang dibuat pada saat program mengalami suatu kondisi yang tidak wajar (*abnormal*). Eksepsi dapat dibangkitkan secara otomatis oleh *Java runtime* maupun secara manual oleh kita sendiri melalui kode yang kita tulis.

Java menyediakan lima buah kata kunci untuk menangani eksepsi, yaitu: *try*, *catch*, *throw*, *throws*, dan *finally*. Kata kunci *try* digunakan untuk membuat blok yang berisi statemenstatemen yang mungkin menimbulkan eksepsi. Jika dalam proses eksekusi runtunan statemen tersebut terjadi sebuah eksepsi, maka eksepsi akan dilempar ke bagian blok penangkap yang dibuat dengan kata kunci *catch*. Pada kasus-kasus tertentu, terkadang kita juga ingin melempar eksepsi secara manual. Untuk itu, kita harus menggunakan kata kunci *throw*. Jika kita ingin membangkitkan sebuah eksepsi tanpa menuliskan blok *try*, maka kita perlu menambahkan kata kunci *throws* pada saat pendeklarasian *method*. Dalam mendefinisikan blok *try*, kita dapat menuliskan statemen tambahan, yaitu menggunakan kata kunci *finally*. Statemen tambahan ini pasti akan dieksekusi baik terjadi eksepsi maupun tidak.

Berikut ini bentuk umum penanganan eksepsi di dalam Java.

```
try {  
    //kumpulan statemen yang mungkin menimbulkan  
    eksepsi  
} catch (TipeEksepsi1 objekEksepsi1) {  
    //penanganan untuk tipe eksepsi1  
} catch (TipeEksepsi2 objekEksepsi2) {
```

```

        //penanganan untuk tipe eksepsi2
    }
    ...
    finally {
        //statemen tambahan yang pasti akan dieksekusi
    }

```

Di sini, *TipeEksepsi* adalah tipe dari eksepsi yang terjadi. Tipe tersebut direpresentasikan dengan sebuah kelas, misalnya: *NullPointerException*, *ArithmeticException*, *ArrayIndexOutOfBoundsException*, dan sebagainya.

Perhatikan kode program berikut ini:

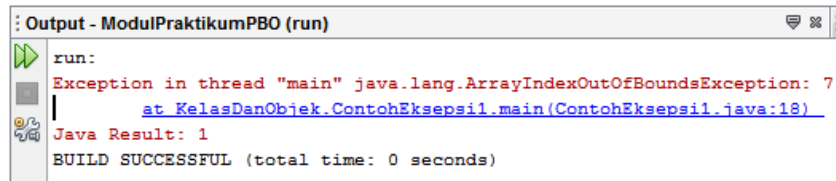
**Program 5.1** Contoh program yang menimbulkan eksepsi

```

class ContohEksepsi1 {
    public static void main(String[] args) {
        int[] arrayInteger = new int[5];
        // SALAH, karena tidak terdapat indeks ke-7
        arrayInteger[7] = 9;
    }
}

```

Jika dijalankan, program di atas akan membangkitkan eksepsi dengan tipe *ArrayIndexOutOfBoundsException* karena kita mengakses indeks yang tidak terdapat di dalam array A. Array A terdiri dari 5 elemen dengan indeks 0 sampai 4. Dengan demikian, Jika kita mengakses indeks ke-7 maka Java akan memberikan pesan kesalahan berikut:



**Gambar 9.1** Pesan kesalahan hasil eksekusi program 9.1

Ini artinya, indeks 7 menimbulkan eksepsi *ArrayIndexOutOfBoundsException*. Kode yang menimbulkan kesalahan tersebut terdapat pada *method main()* di dalam kelas *ContohEksepsi1*, yang tersimpan dalam file *ContohEksepsi1.java*, aris ke-18.

Tampak pada contoh di atas bahwa ketika terjadi sebuah eksepsi maka program akan dihentikan secara tidak normal. Oleh karena itu, kita perlu mendefinisikan sebuah mekanisme yang dapat menangani kesalahan tersebut. Mekanisme inilah yang dinamakan dengan *exception-handling*.

### Menggunakan Kata Kunci *try* dan *catch*

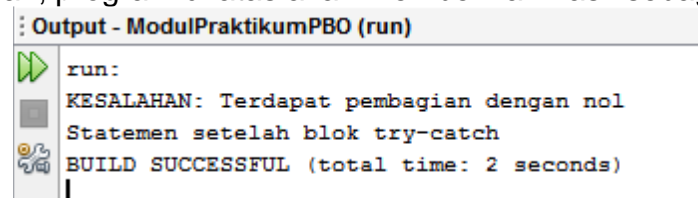
Pada bagian sebelumnya telah dicontohkan bagaimana Java mencegah eksepsi secara *default*. Namun, pada umumnya, pencegahan eksepsi dilakukan oleh kita sendiri, sebagai *programmer*. Untuk melakukan hal tersebut, kita dapat menggunakan blok *try*-

*catch*. Berikut ini contoh program yang menggunakan blok *try-catch* untuk menangani kesalahan yang muncul.

#### Program 5.2 Contoh implementasi blok *try-catch*

```
class ContohEksepsi2 {
    public static void main(String[] args) {
        int pembilang = 7;
        int penyebut = 0;
        try {
            int hasil = pembilang / penyebut; // SALAH
// tidak dieksekusi
            System.out.println("Hasil = " + hasil);
        } catch (Exception e) {
            System.out.println("KESALAHAN: "
                + "Terdapat pembagian dengan nol");
        }
        System.out.println("Statemen setelah blok
trycatch");
    }
}
```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:



```
Output - ModulPraktikumPBO (run)
run:
KESALAHAN: Terdapat pembagian dengan nol
Statemen setelah blok try-catch
BUILD SUCCESSFUL (total time: 2 seconds)
```

**Gambar 5.2** Hasil eksekusi program 9.2

Pada program di atas, pada saat muncul eksepsi, program akan langsung meloncat ke bagian *catch*. Namun bagian statemen yang dituliskan setelah blok *try-catch* tetap akan dieksekusi oleh program.

Dengan menggunakan blok *try-catch* seperti contoh-contoh di atas, meskipun di dalam program sebenarnya terkandung kesalahan, namun proses eksekusi program tetap dapat dilanjutkan, tidak dihentikan secara tiba-tiba. Inilah yang disebut dengan proses pengebakan kesalahan atau pengebakan eksepsi.

#### Pengebakan Beberapa Tipe Eksepsi

Pada kenyataan di lapangan, sebuah blok *try-catch* biasanya terdiri atas beberapa bagian blok penangkap. Hal tersebut bertujuan agar dalam satu blok *try-catch*, kita dapat mengatasi beberapa kemungkinan eksepsi yang terjadi. Berikut ini contoh program yang akan menunjukkan bagaimana cara mendefinisikan sebuah blok *try-catch* yang dapat digunakan untuk mengebak beberapa tipe eksepsi.



### Program 5.3 Contoh pengebakan beberapa tipe eksepsi

```
class ContohMultiEksepsi {
    public static void cobaEksepsi(int pembilang, int
penyebut) {
        try {
            int hasil = pembilang / penyebut;
            System.out.println("Hasil bagi: " + hasil);
            int[] Arr = {1, 2, 3, 4, 5}; // array dengan 5
elemen
            Arr[ 10] = 23; // mengakses indeks ke-10
        } catch (ArithmeticException eksepsi1) {
            System.out.println(
                "Terdapat pembagian dengan0");
        } catch (ArrayIndexOutOfBoundsException eksepsi2) {
            System.out.println("Indeks di luar rentang");
        }
    }

    public static void main(String[] args) {
        cobaEksepsi(4, 0); // menimbulkan rithmeticException
        System.out.println();
        // menimbulkan ArrayIndexOutOfBoundsException
        cobaEksepsi(12, 4);
    }
}
```

Hasil eksekusi program di atas adalah sebagai berikut:

```
Output - ModulPraktikumPBO (run)
run:
Terdapat pembagian dengan 0
Hasil bagi: 3
Indeks di luar rentang
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Gambar 5.3** Hasil eksekusi program 9.3

Tampak pada hasil di atas bahwa ketika kita melewati argumen ke dalam *method test()* dengan nilai 4 dan 0, program akan membangkitkan eksepsi *ArithmeticException*. Sedangkan Jika argumen bernilai 12 dan 4, maka program akan melakukan pembagian dengan benar dan menghasilkan nilai 3 (hasil dari 12/4).

Namun, pada saat mengeksekusi statemen:

```
Arr[10] = 23;
```

program akan membangkitkan eksepsi *ArrayIndexOutOfBoundsException* - *Exception*. Hal ini disebabkan karena array *Arr* hanya terdiri dari 5 buah elemen.

### Menggunakan Kata Kunci *throw*

Kita dapat membangkitkan eksepsi secara manual dengan menggunakan kata kunci *throw*. Berikut ini bentuk umum penggunaan kata kunci tersebut.

```
throw eksepsi;
```

eksepsi yang dimaksud harus berupa objek *Throwable* maupun objek dari kelas-kelas turunannya. Kita tidak dapat melempar objek non-*Throwable*, misalnya objek *String*. Sebagai contoh, Jika kita ingin membangkitkan eksepsi *NullPointerException*, maka kita dapat menuliskan kodenya sebagai berikut:

```
throw NullPointerException();
```

Program berikut akan menunjukkan bagaimana cara menggunakan kata kunci *throw* untuk membangkitkan eksepsi *NullPointerException*.

#### Program 5-4 Contoh pengebakan eksepsi dengan kata kunci *throw*

```
class Mahasiswa {
    private String nim;
    private String nama;

    public void setNIM(String inputNIM) {
        try {
            nim = inputNIM;
            if (inputNIM == null) {
                throw new NullPointerException();
            }
        } catch (NullPointerException npe) {
            System.out.println("KESALAHAN: "
                + "NIM tidak boleh null");
        }
    }

    public String getNIM() {
        return nim;
    }

    public void setNama(String inputNama) {
        try {
            nama = inputNama;
            if (nama == null) {
                throw new NullPointerException();
            }
        }
    }
}
```

```

        } catch (NullPointerException npe) {
            System.out.println("KESALAHAN: " + "Nama
mahasiswa tidak boleh null");
        }
    }

    public String getNama() {
        return nama;
    }
}

class DemoThrow {
    public static void main(String[] args) {
        Mahasiswa mhs = new Mahasiswa();
        mhs.setNIM(null);
        mhs.setNama("Nadilla");
        System.out.println("\nNIM : " + mhs.getNIM());
        System.out.println("Nama : " + mhs.getNama());
    }
}

```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:

```

Output - ModulPraktikumPBO (run)
run:
KESALAHAN: NIM tidak boleh null
NIM : null
Nama : Nadilla
BUILD SUCCESSFUL (total time: 2 seconds)

```

**Gambar 5.4** Hasil eksekusi program 5.4

Tampak pada hasil di atas, ketika kita melewati nilai *null* ke dalam *method setNIM()* dari kelas *Mahasiswa*, program akan membangkitkan eksepsi *NullPointerException* yang kita lempar atau kita bangkitkan secara manual dengan menggunakan kata kunci *throw*.

### Menggunakan Kata Kunci *throws*

Jika kita tidak menyertakan blok *try-catch* di dalam *method* yang mengandung kode-kode yang mungkin menimbulkan eksepsi, maka kita harus menyertakan klausa *throws* pada saat pendeklarasian *method* yang bersangkutan. Jika tidak, maka program tidak dapat dikompilasi. Berikut ini bentuk umum dari penggunaan kata kunci *throws*.

```

tipe nama-method(daftar-parameter) throws tipe-eksepsi1,
tipe-eksepsi2, ..., tipe-eksepsiN {
    //badan-method
}

```

Berikut ini contoh program yang menggunakan kata kunci *throws* untuk menangani eksepsi yang muncul.

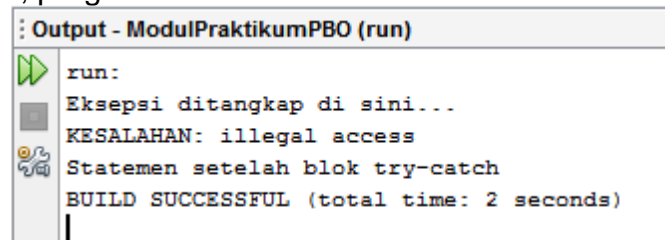
**Program 5.5** Contoh pengebakan eksepsi dengan kata kunci *throws*

```
Class DemoThrows {
    public static void cobaEksepsi() throws
        IllegalAccessException {
        throw new IllegalAccessException(
            "KESALAHAN: illegal
access");
    }

    public static void main(String[] args)
    {
        try {
            cobaEksepsi();
        } catch (Exception e) {
            System.out.println("Eksepsi
ditangkap di sini...");

            System.out.println(e.getMessage());
        }
        System.out.println("Statemen
setelah blok trycatch");
    }
}
```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:



```
Output - ModulPraktikumPBO (run)
run:
Eksepsi ditangkap di sini...
KESALAHAN: illegal access
Statemen setelah blok try-catch
BUILD SUCCESSFUL (total time: 2 seconds)
```

**Gambar 5.5** Hasil eksekusi program 5.5

Perhatikan deklarasi *method cobaEksepsi()* di atas. Di situ kita menambahkan klausa *throws* yang diikuti dengan nama eksepsi yang akan dilempar/dibangkitkan. Melalui teknik seperti ini, *compiler* akan mengetahui bahwa *method* tersebut dapat membangkitkan eksepsi *IllegalAccessException* sehingga program dapat dikompilasi dengan sukses.

### Menggunakan Kata Kunci *finally*

Terkadang kita ingin menempatkan kode yang pasti akan dieksekusi, baik terjadi eksepsi maupun tidak. Untuk melakukan hal itu, kita perlu menempatkan kode tersebut di dalam bagian finalisasi dari blok *try-catch*. Blok finalisasi ini dibuat dengan menggunakan kata kunci *finally*. Bentuk umum pendefinisian blok *finally* ini sebagai berikut:

```
try
{
//statemen yang mungkin menimbulkan eksepsi A, B, danC
} catch (A ea) {
//blok penangkap untuk eksepsi A
} catch (B eb) {
//blok penangkap untuk eksepsi B
} catch (C ec) {
//blok penangkap untuk eksepsi C
} finally {
//statemen yang pasti akan dieksekusi, baik
terjadi
//eksepsi maupun tidak
}
```

Untuk mengetahui cara kerja dari kata kunci *finally*, perhatikan program berikut ini.

#### Program 5.6 Contoh penggunaan kata kunci *finally*

```
class DemoFinally {
    public static void cobaEksepsi(int pembilang, int
penyebut) {
        try {
            int hasil = pembilang / penyebut;
            System.out.println("Hasil bagi: " + hasil);
            int[] Arr = {1, 2, 3, 4, 5}; // array dengan 5
elemen
            Arr[10] = 23; // mengakses indeks ke-10
        } catch (ArithmeticException eksepsi1) {
            System.out.println("Terdapat pembagian dengan
0");
        } catch (ArrayIndexOutOfBoundsException eksepsi2) {
            System.out.println("Indeks di luar rentang");
        } finally {
            System.out.println("Ini adalah statemen dalam
blok finally");
        }
    }

    public static void main(String[] args) {
        cobaEksepsi(4, 0); //
```

```

    menimbulkanArithmeticException
        System.out.println();
        cobaEksepsi(12, 4); // menimbulkan
    ArrayIndexOutOfBoundsException
    }
}

```

Pada saat dijalankan, program di atas akan memberikan hasil sebagai berikut:

```

Output - ModulPraktikumPBO (run)

run:
Terdapat pembagian dengan 0
Ini adalah statemen dalam blok finally

Hasil bagi: 3
Indeks di luar rentang
Ini adalah statemen dalam blok finally
BUILD SUCCESSFUL (total time: 2 seconds)

```

**Gambar 5.6** Hasil eksekusi program 9.6

## C. LATIHAN

Lakukan praktek diatas dan lakukan evaluasi

## E. TUGAS

Buatlah program penyisipan array dan gunakan blok try-catch untuk menangani kesalahannya. Contoh

1	2
---	---

Sisipkan 3 di indeks ke-1 maka array berubah

1	3	2
---	---	---

# MODUL 6

## GUI DAN EVENT HANDLING

### A. TUJUAN PEMBELAJARAN

1. Memahami Konsep Graphical User Interface
2. Membuat interface aplikasi dengan berbasis GUI
3. Memahami tentang penanganan kejadian
4. Praktikan mampu mengimplementasikan GUI serta event event yang ada

### B. DASAR TEORI

#### ***Pemrograman Swing***

Graphical User Interface (GUI) adalah tampilan-interface grafis pada suatu aplikasi yang berfungsi untuk menjadi perantara antara pengguna dengan suatu program. Pada pemrograman 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`.

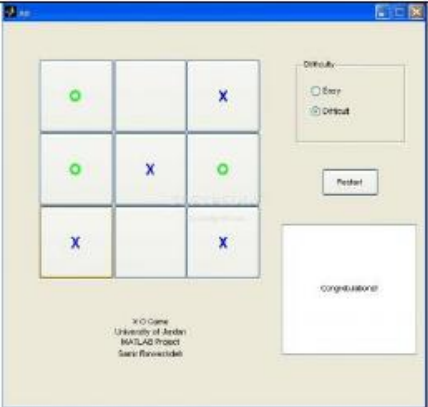
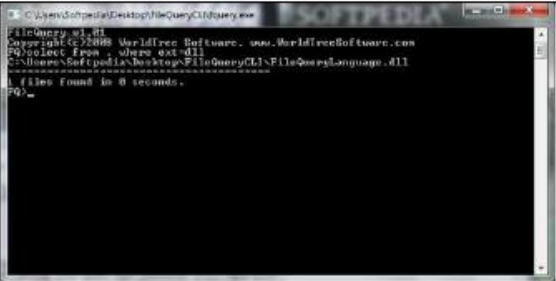
Tidak seperti beberapa komponen AWT yang menggunakan native code, keseluruhan Swing ditulis menggunakan bahasa pemrograman Java. Swing menyediakan implementasi platform-independent dimana aplikasi yang dikembangkan dengan platform yang berbeda dapat memiliki tampilan yang sama.

Seperti pada package AWT, package dari Swing menyediakan banyak kelas untuk membuat aplikasi GUI. Package tersebut dapat ditemukan di `javax.swing`. Perbedaan utama antara keduanya adalah komponen Swing ditulis menyeluruh menggunakan Java mengingat yang belakangan tidak. Kesimpulannya, program GUI ditulis menggunakan banyak kelas dari package Swing yang mempunyai tampilan look and feel yang sama meski dijalankan pada beda platform. Lebih dari itu, Swing menyediakan komponen yang lebih menarik seperti color chooser dan option pane.

Nama dari komponen GUI milik Swing hampir sama persis dengan komponen GUI milik AWT. Perbedaan jelas terdapat pada penamaan komponen. Pada dasarnya, nama komponen Swing sama dengan nama komponen AWT tetapi dengan tambahan huruf J pada prefixnya. Sebagai contoh, satu komponen dalam AWT adalah `button` class. Sedangkan pada Swing, nama komponen tersebut menjadi `Jbutton` class.

Berikut adalah beberapa contoh komponen swing dan penjelasan dari Swing

1. **JComponent** : Kelas induk untuk semua komponen Swing, tidak termasuk toplevel container checkbox class dalam package AWT
2.  **JButton** : Tombol “push”. Korespondensi pada button class dalam package AWT.
3. **JCheckBox** : Item yang dapat dipilih atau tidak oleh pengguna. Korespondensi pada
4. **JFileChooser** : Mengijinkan pengguna untuk memilih sebuah file. Korespondensi pada cfilechooser class dalam package AWT
5. **JTextField** : Mengijinkan untuk mengedit text satu baris. Korespondensi pada textfield class dalam package AWT.
6. **JFrame** : Turunan dan korepondensi pada frame class dalam package AWT tetapi keduanya sedikit tidak cocok dalam kaitannya dengan menambahkan komponen pada kontainer. Perlu mendapatkan content pane yang terbaru sebelum menambah sebuah komponen.
7. **JPanel** : Turunan Jcomponent. Kontainer class sederhana tetapi bukan top-level. Korespondensi pada panel class dalam package AWT.
8. **JApplet** : Turunan dan korepondensi ke Applet class dalam package AWT. Juga sedikit tidak cocok dengan applet class dalam kaitannya dengan menambahkan komponen pada container
9. **JOptionPane** : Turunan Jcomponent. Disediakan untuk mempermudah menampilkanpop- up kotak dialog.
10. **JDialog** : Turunan dan korespondensi pada dialog class dalam package AWT. Biasanya digunakan untuk menginformasikan sesuatu kepada pengguna atau prompt pengguna untuk input.
11. **JColorChooser** : Turunan Jcomponent. Mengijinkan pengguna untuk memilih Warna

Program dengan GUI	Program tanpa GUI (CLI)
	



### Pengaturan tata letak komponen

Pengaturan tata letak pada java dapat dilakukan dengan setlayout yaitu FlowLayout, GridBagLayout, BorderLayout, BoxLayout, SpringLayout dan CardLayout, selain itu bisa juga dengan metode *setBounds(x,y,p,l)*. Dimana x dan y adalah koordinat, dan p adalah panjang objek dan l adalah lebar objek. Tetapi sebelum melakukannya, pastikan bahwa layout sudah dibuat null, dengan cara memanggil metode *getContentPane().setLayout(null)*. Setelah itu komponen di tempel pada window dengan cara *getContentPane().add(<nama objek>)*.

Berikut contohnya :

```
getContentPane().setLayout(null);  
but.setBounds(50,250,150,24);  
lb.setBounds(50,50,100,24);  
jt.setBounds(50,100,200,100);  
getContentPane().add(but);  
getContentPane().add(lb);  
getContentPane().add(jt);  
but.addActionListener(this);  
show();
```

### Penanganan Kejadian

Penanganan kejadian atau ActionEvent pada java merupakan suatu cara untuk berinteraksi antara program dengan user. Penanganan kejadian yang akan dibahas berikut ini adalah paket dari java.awt.event.\*. Tabel berikut memperlihatkan beberapa event dan event listener yang umumnya diperlukan dalam aplikasi.

Event	Event Listener
Klik button, menekan Enter di text field, atau memilih item menu	ActionListener
Menutup window	WindowListener
Menekan button mouse saat kursor berada diatas komponen	MouseListener
Memindahkan kursor	MouseMotionListener
Menampilkan komponen	ComponentListener
Komponen mendapat fokus	FocusListener
Mengubah pemilihan tabel atau list	ListSelectionListener

Penanganan kejadian yang digunakan dalam penulisan ini adalah kelas yang terdapat pada paket `java.awt.event`. Sebelumnya pada penamaan class harus mengimplementasikan `ActionListener`. Pada paket ini dapat digunakan oleh komponen AWT maupun SWING, agar objek dapat menangani suatu kejadian, objek tersebut harus didaftarkan sebagai *listener*. Contohnya adalah sebagai berikut :

```
 JButton But = new JButton("Kirim");  
 But.addActionListener(this);
```

Dan penulisan isi respon dari objek ditulis pada metode `void actionPerformed(ActionEvent e){}`.

### C. LATIHAN

#### 1. Program 1 :

```
import javax.swing.*;  
import java.awt.event.*;  
class Coba extends JFrame implements ActionListener  
{  
    JButton But = new JButton("Kirim");  
    but.addActionListener(this);  
    show();  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
public void actionPerformed(ActionEvent e)  
{  
    if(e.getSource()==but){String kata="";  
    kata=jt.getText();  
    JOptionPane.showMessageDialog(null,kata);}  
}
```

*Catatan : Disini kita menggunakan template `JOptionPane`, yang memang digunakan untuk menampilkan message box.*

## Program 2

*// Nama File : GUIfak.java*

*// Nama Program : Program untuk menghitung nilai faktorial*

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

class GUIfak extends JFrame implements ActionListener
{
    double hasil;
    Container con = new Container();
    JButton hapus, ok;
    JTextField tex1, tex2;
    double fakt(double angka)
    {
        if(angka == 0)
        {
            return 1;
        }
        else
        {
            return angka*fakt(angka-1);
        }
    }
}

public GUIfak()
{
    super("faktorial niy...!!");
    setSize (300,250);
    ok = new JButton ("faktorial");
    hapus = new JButton ("hapus");
    ok.addActionListener (this);
    hapus.addActionListener (this);
    JPanel tombol = new JPanel ();
    hapus.setEnabled (false);
    tombol.setLayout (new GridLayout(1,2,10,10));

    tombol.add (hapus);
    tombol.add(ok);
    tex1 = new JTextField("");
    tex2 = new JTextField("");
    tex2.setEditable (false);
    JPanel tex = new JPanel();
    tex.setLayout(new GridLayout (2,1,10,10));
    tex.add(tex1);
    tex.add(tex2);
    con = getContentPane();
    con.setLayout(null);
    tex.setBounds(100,80,100,50);
    tombol.setBounds(50,150,200,30);
    con.add(tombol);
}
```

```

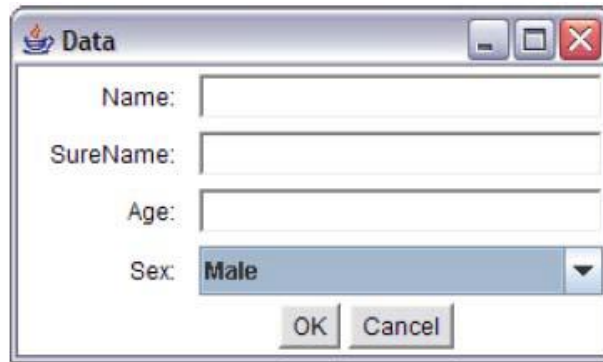
        con.add(tex);
        show();
    }
    public void actionPerformed (ActionEvent e)
    {
        try
        {
            if (e.getSource()==ok)
            {
                hapus.setEnabled(true);
                String a;
                double angka = Double.parseDouble(tex1.getText());
                hasil = fakt(angka);
            }
            tex2.setText(Double.toString(hasil));
            if (e.getSource() == hapus)
            {
                tex1.setText("");
                tex2.setText("");
                hapus.setEnabled(false);
            }
        }

        catch(Exception ex)
        {
            hapus.setEnabled(false);
            JOptionPane.showMessageDialog(this, "Masukkan nilai yang benar..!!");
        }
    }
    public static void main(String[] Zzzz)
    {
        GUIfak q = new GUIfak();
        q.setResizable(false);
        q.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

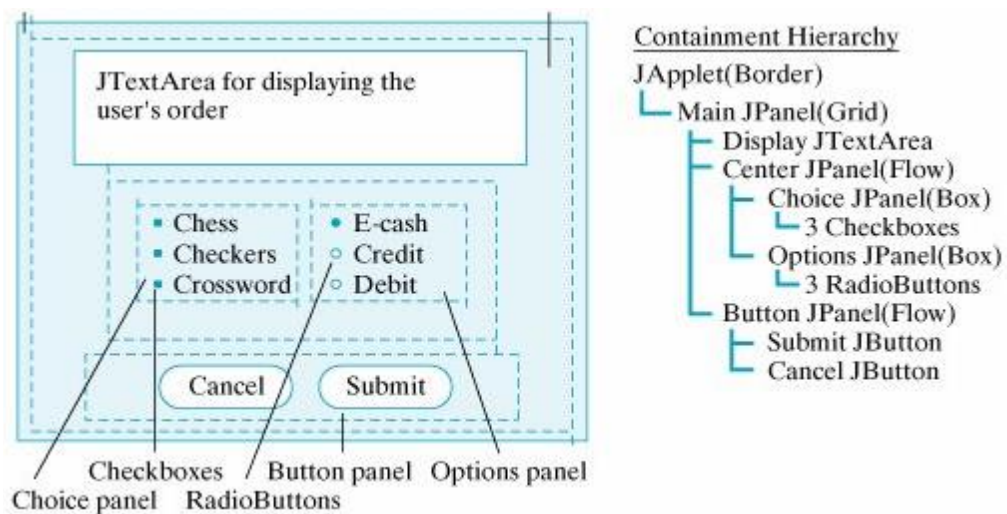
```

#### D. TUGAS

1. Membuat tamplan input dan menampilkan hasil input ketikaditekan tombol OK



2. Desain user interface sesuai petunjuk pada gambar berikut:



Petunjuk : Gantilah JApplet dengan JPanel