MODUL 3
OOP-1

A. TUJUAN

- Praktikan dapat mengerti konsep dasar OOP.
- Praktikan dapat membandingkan pemrograman berorientasi objek dengan prosedural.
- Praktikan mengerti dan dapat mengimplementasikan Class, Object, Method, Constructor, dan UML pada program sederhana.

B. PERANGKAT

NetBeans IDE 7.2

C. DASAR TEORI

1. Definisi OOP

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. Dianjurkan diimplementasikan untuk program dengan berbagai ukuran karena lebih mudah untuk men-*debug*.

2. Modifier

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

Tabel 3.1 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 3.2 Karakteristik Access Modifier

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

Tabel 3.3 Karakteristik Permitted Modifier

Modifier	Class	Interface	Method	Variabel
Abstract	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.
Final	Class tidak dapat diturunkan.	Tidak dapat diterapkan.	Method tidak dapat ditimpa oleh method di subclass-subclassnya	Berperilaku sebagai konstanta

Static	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.

3. *Class*

Class adalah cetak biru (rancangan) atau prototipe 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.

Contoh :

- Class Manusia → Obyek :

• Data : nama (String), umur (integer).

Script:

```
public class Manusia {
   //definisi atribut
   private String nama;
   private int umur;
   //definisi method
   public void setNama(String a){
         nama=a;
   }
   public String getNama(){
         return nama;
   }
   public void setUmur(int a){
         umur=a;
   }
   public int getUmur(){
         return umur;
   }
}
```

4. *Object*

Object (objek) secara lugas dapat diartikan sebagai instansiasi atau hasil ciptaan dari suatu class. Asumsikan cetakan kue adalah class, maka kue yang dihasilkan dari cetakan tersebut merupakan objek dari class cetakan kue. Dalam pengembangan OOP lebih lanjut, sebuah objek dapat dimungkinkan terdiri atas objek-objek lain. Atau, bisa jadi sebuah objek merupakan turunan dari objek lain, sehingga mewarisi sifat-sifat induknya dan memiliki sifat tambahan.

- **Keyword "new"**

  "new" digunakan untuk melakukan instansiasi/ membuat sebuah object baru.

  Contoh:

```
Manusia objekManusia = new Manusia();
```

5. *Method*

Method biasa kita kenal sebagai *function* dan *procedure*. Dikatakan fungsi bila method tersebut melakukan suatu proses dan mengembalikan suatu nilai (*return value*), dan dikatakan prosedur bila method tersebut hanya melakukan suatu proses dan tidak mengembalikan nilai (*void*). Dalam OOP, method digunakan untuk memodularisasi program melalui pemisahan tugas dalam suatu class. Pemanggilan method menspesifikasikan nama method dan menyediakan informasi (parameter) yang diperlukan untuk melaksanakan tugasnya.

Deklarasi method yang *non-void* atau mengembalikan nilai (fungsi)

```
[modifier]Type-data namaMethod(parameter1,parameter2,…parameterN)
{
Deklarasi-deklarasi dan proses ;
return nilai-kembalian;
}
```

Deklarasi method yang *void* atau tidak mengembalikan nilai (prosedur)

```
[modifier]void namaMethod(parameter1,parameter2,…parameterN)
{
Deklarasi-deklarasi dan proses ;
}
```

6. *Constructor*

Tipe khusus method yang digunakan untuk menginstansiai 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. Constructor 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 menambahkan keyword "*new*". *Keyword new* dalam deklarasi ini artinya kita mengalokasikan pada memori sekian blok memori untuk menampung objek yang baru kita buat.

Deklarasi Constructor:

```
public class DemoManusia {
   public static void main(String[] args) { //program utama
   Manusia arrMns[] = new Manusia[3]; //buat array of object
            Manusia objMns1 = new Manusia(); //constructor pertama
            objMns1.setNama("Markonah");
            objMns1.setUmur(76);
            Manusia objMns2 = new Manusia("Mat Conan");
   //constructor kedua
   Manusia objMns3 = new Manusia("Bajuri", 45); //constructor ketiga
            arrMns[0] = objMns1;
            arrMns[1] = objMns2;
            arrMns[2] = objMns3;
            for(int i=0; i<3; i++) {
                  System.out.println("Nama : "+arrMns[i].getNama());
                  System.out.println("Umur : "+arrMns[i].getUmur());
                  System.out.println();
            }
   }
}
```

Hasil Running:

```
Nama : Markonah
Umur : 76
Nama : Mat Conan
Umur : 0
Nama : Bajuri
Umur : 13
```

Contoh 2 Deklarasi Constructor:

```
//PROGRAM KOTAK OBYEK
class Kotak {
   double panjang;
   double lebar;
   double tinggi;
   // Mendefinisikan constructor untuk kelas Kotak
   Kotak() {
         panjang = 4;
         lebar = 3;
         tinggi = 2;
   }
   double hitungVolume() {
         return (panjang * lebar * tinggi);
   }
}
class DemoConstructor1 {
   public static void main(String[] args) {
         Kotak k1;
         k1 = new Kotak();

         System.out.println("Volume k1 = " + k1.hitungVolume());

   }
}
```

7. This

Suatu besaran referensi khusus yang digunakan di dalam method yang dirujuk untuk objek yang sedang belaku. 'This' digunakan ketika nama atribut yang sama dengan nama variable lokal.

Deklarasi This:

```
public class Lagu {
   private String band;
   private String judul;
   public void IsiParam(String judul,String band) {
         this.judul = judul;
         this.band = band;
   }

   public void cetakKeLayar() {
         if(judul==null && band==null) return;
System.out.println("Judul : " + judul +"\nBand : " + pencipta);
   }
}

public class DemoLagu {
   public static void main(String[] args) {
         Lagu song = new Lagu();
         song.IsiParam("Dance Beside","All American Reject ");
         song.cetakKeLayar();
   }
}
```

Hasil Running:

```
Judul : Dance Beside
Pencipta: All American Reject
```

8. *Overload*

Didalam Java, kita dapat membuat dua atau lebih konstruktor/ method yang mempunyai nama sama dalam satu kelas, tetapi jumlah dan tipe argumen dari masing-masing constructor atau method haruslah berbeda satu dengan yang lainnya. Hal ini yang dinamakan *overloading*.

Contoh 1 Deklarasi Overload:

```
public void setHarga(int harga){}
public void setHarga(double harga){}
public void setHarga(float harga){}
public void setHarga(float harga, String jumlah){}
```

Contoh 2 Deklarasi Overload:

```
public class Phone{
  private String merk;
  private int harga;
  public Phone(){
  }
  public Phone(String merk){
    this.merk=merk;
  }
  public Phone(String merk, int harga){
    this.merk=merk;
    this.harga=harga;
  }
  public void isiPhone(String merk){
    this.merk=merk;
  }
  public void isiPhone(String merk, int harga){
    this.merk=merk;
    this.harga=harga;
  }
  public void lihatPhone(){
    System.out.println(" Merk : "+merk);
    System.out.println(" Harga : "+harga);
    System.out.println("");
  }
}
public class DemoOverLoading{
  public static void main(String args[]){
   System.out.println("");
   Phone p1 = new Phone();
   Phone p2 = new Phone("Nokia");
   Phone p3 = new Phone("Sony Ericsoon",500);
```

```
   System.out.println("Perbedaan Output dari masing2
konstruktor");
   p1.lihatPhone();
   p2.lihatPhone();
   p3.lihatPhone();

   Phone p4,p5;
   p4 = new Phone();
   p5 = new Phone();
   p4.isiPhone("Samsung");
   p5.isiPhone("Samsung", 5000);
   System.out.println("Perbedaan Output dari masing2 method");
   p4.lihatPhone();
   p5.lihatPhone();
  }
}
```

Hasil Running:

```
Perbedaan Output dari masing2 konstruktor
Merk : null
Harga : 0

Merk : Nokia
Harga : 0

Merk : Sony Ericsoon
Harga : 500

Perbedaan Output dari masing2 method
Merk : Samsung
Harga : 0

Merk : Samsung
Harga : 5000
```

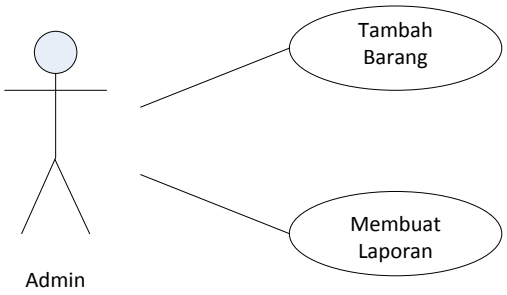
9. UML (*Unified Modeling Language*)

Pemodelan (*modeling*) adalah proses merancang peranti lunak (software) sebelum melakukan pengkodean (*coding*). Model peranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting, karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin kompleks sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik. Dengan menggunakan model, diharapkan pengembangan peranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti scalability, robustness, security, dan sebagainya.

Unified Modeling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang,dan mendokumentasikan sistem peranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Secara konsep dasar,UML mendefinisikan delapan diagram sebagai berikut.

a. Use Case Diagram

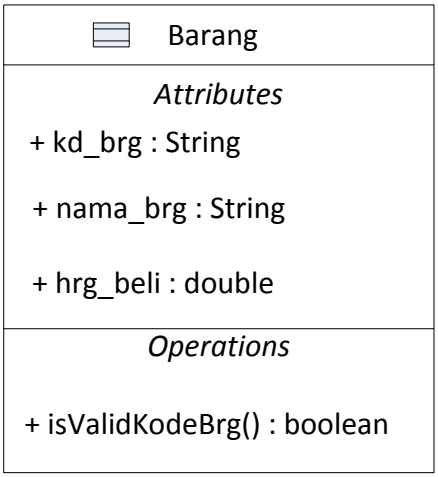
Menggambarkan fungsionalitas dari sebuah sistem (apa fungsinya), yang merepresentasikan sebuah interaksi antara aktor dan sistem (sebuah pekerjaan). Misalnya menambah data/membuat laporan.



Gambar 3.1 Use Case Diagram

b. Class Diagram

Class adalah sebuah spesifikasi objek, yang memiliki atribut/properti dan layanan/fungsional (metode/fungsi). Class diagram menggambarkan struktur dan deskripsi kelas, package dan objek beserta hubungan satu sama lain, seperti hal pokok: Nama (dan stereotype), Atribut, dan Metode.



Gambar 3.2 Contoh Class Diagram

Hubungan antarkelas :

1. **Asosiasi,** yaitu hubungan statis antar kelas , biasanya menggambarkan kelas yang memiliki atribut berupa kelas lain. Terdapat banyak jenis asosiasi, misalnya :

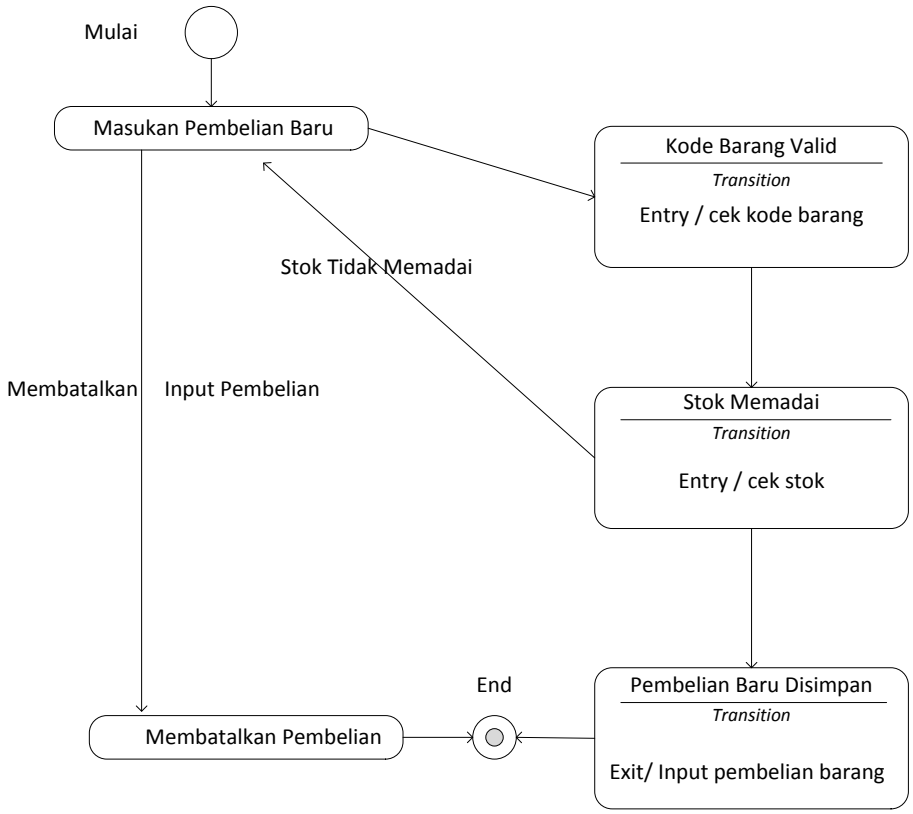
a. Asosiasi sederhana : bentuk asosiasi sederhana (———).
b. Agregasi yaitu hubungan yang menyatakan bagian, biasanya hubungan data master dan detailnya. Misal satu pembelian terdiri dari sejumlah item barang (◆———).
c. *Navigability* : menunjukan arah query/komunikasi antar objek, bisa satu atau dua arah, terlihat pada tanda panahnya (———→).
d. Campuran / Composit : campuran asosiasi (◆———→).

2. **Generalisasi,** yaitu hubungan hierarkis antara anak dan bapak, karena kelas dapat diturunkan dari kelas lain dan mewarisi semua atribut dan metode kelas asalnya serta menambahkan fungsionalitas baru (———▶).
3. **Implementasi (***realization***),** yaitu hubungan antara objek yang menjamin adanya pola khusus dalam perilaku anggota objek lainnya. Ini dapat diwujudkan dengan adanya kelas yang mengimplemtasikan interface tertentu (- - - - - - - - - - - - ▶).
4. **Ketergantungan (***dependency***)** yaitu sebuah kelas membutuhkan objek lain untuk bisa memfungsikan dirinya sendiri dengan baik (------> ---------).
5. **Hubungan dinamis**, yaitu rangkaian pesan (message) yang di-passing dari satu kelas ke kelas lain.

c. Statechart Diagram

Diagram ini menggambarkan transisi dan perubahan keadaan suatu objek, akibat dari stimulus/input yang diterimanya. State digambarkan dalam bentuk segiempat dengan sudut membulat dan memiliki nama

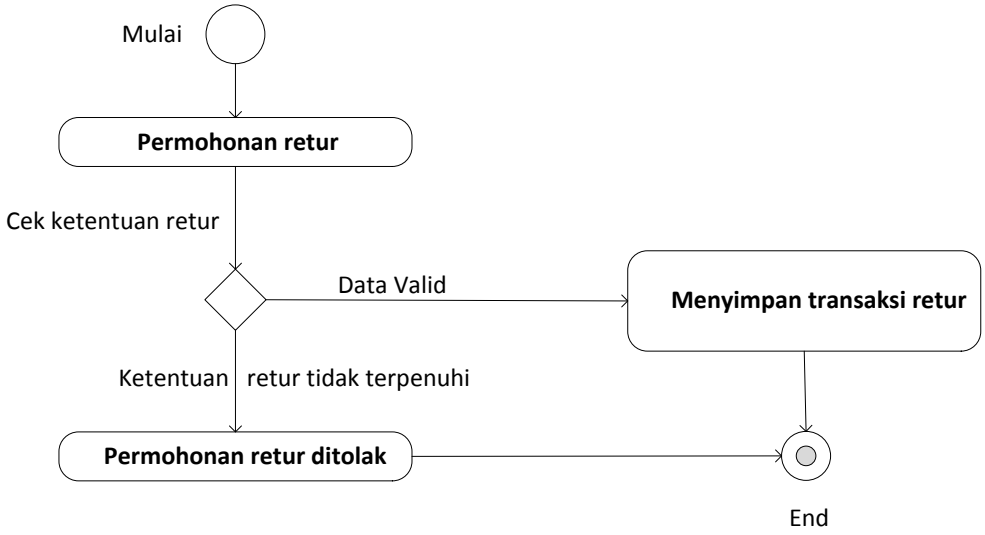
sesuai kondisinya. Transisi antar-state umumnya memiliki kondisi yang merupakan syarat terjadinya transisi yang bersangkutan.



Gambar 3.3 Contoh Statechart Diagram

d. Activity Diagram

Diagram ini menggambarkan berbagai aktivitas dalam sistem yang sedang dirancang, mulai dari titik awal, melalui kondisi (*decision*) yang mungkin terjadi, kemudian sampai pada titik akhir. Diagram ini juga mampu menggambarkan proses parallel yang mungkin terjadi pada beberapa eksekusi. Diagram ini tidak menggambarkan perilaku/proses internal sebuah sistem maupun interaksi antar-subsistem, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas secara umum (global).



Gambar 3.4 Contoh Activity Diagram

e. Sequence Diagram

Diagram ini menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. Sequence diagram terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). Biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebauh event untuk menghasilkan sebuah output tertentu.

f. Collaboration Diagram

Collaboration diagram menggambarkan interaksi antar objek seperti sequence diagram, tetapi lebih menekankan pada peran amsing-masing objek dan bukan pada waktu penyampaian *message*. Setiap message memiliki urutan angka, level tertinggi dari nomer 1, sedangkan untuk message dari level yang sama memiliki prefiks sang sama.

g. Component Diagram

Diagram ini menggambarkan struktur dan hubungan antar-komponen peranti lunak, termasuk ketergantungan (dependency). Diantaranya modul berisi kode, baik berisi source kode, binary, library, executable.

User interface adalah level terakhir yang bisa dilihat oleh pengguna, sedangkan sistem pendukung lain seperti sistem operasi/database dan mesin logic program tidak akann terlihat oleh pengguna.

h. Deployment Diagram

Deployment Diagram Menggambarkan detail bagaimana komponen dibentuk dan didistribusikan (*deploy*) dalam infrastruktur sistem. Dimana komponen akan terletak pada mesin, server atau peranti keras apa. Bagaimana jaringan pada lokasi tersebut, kisalnya server, client dan hal-hal lain yang bersifat fisik.