MODUL PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBYEK

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

MODUL 1
DASAR PEMROGRAMAN JAVA

A. TUJUAN

- Praktikan mampu memahami tentang dasar-dasar pemrograman java.
- Praktikan mampu memahami tentang deklarasi variabel, operator, kondisional & pilihan, serta string.
- Praktikan dapat mengimplementasikannya dalam program sederhana dengan menggunakan NetBeans.

B. PERANGKAT

- NetBeans IDE 7.2

C. DASAR TEORI

1. Deklarasi Variabel

Bahasa pemrograman pada umumnya mengenal adanya variabel yang digunakan untuk menyimpan nilai atau data. Java dikenal dengan bahasa pemrograman yang bersifat *strongly typed* yang artinya diharuskan mendeklarasikan tipe data dari semua variabel dan apabila lupa atau salah mengikuti aturan pendeklarasian variabel maka akan mendapat *error* pada saat proses kompilasi.

1.1 Tipe data

Java memiliki dua jenis tipe data yang dikategorikan menjadi dua yaitu tipe data primitif dan tipe data referensi.

1.1.1 Tipe Data Primitif

Macam tipe data primitif diantaranya :

- **Integer (Bilangan bulat)**

  Integer merupakan tipe data numerik yang digunakan untuk mendefinisikan bilangan bulat. Tipe data numeric yang termasuk integer diantaranya :

Tabel 1.1 Tipe Data Primitif Integer

Tipe	Deskripsi
Byte	-128 s/d +127 menempati 8 bits di memori
Short	-32768 s/d +32767 menempati 16 bits di memori
Int	-2147483648 s/d +2147483647 menempati 32 bits di memori
Long	-9223372036854775808 s/d +9223372036854775807 menempati 64 bits di memori

- **Floating Point (Bilangan pecahan)**

  Floating point digunakan untuk menangani bilangan decimal atau perhitungan yang lebih detail dibanding integer.

Tabel 1.2 Tipe Data Primitif Floating Point

Tipe	Deskripsi
Float	$-3.4x10^{8}$ s/d $+3.4x10^{8}$
Double	$-1.7x10^{308}$ s/d $+1.7x10^{308}$

- **Char**

  Char adalah karakter tunggal yang pendefinisiannya di awal dan akhir menggunakan tanda petik tunggal (‘). Tipe char mengikuti aturan Unicode, sehingga bisa dapat menggunakan kode untuk kemudian diikuti bilangan dari 0 sampai 65535, tetapi yang biasa digunakan adalah bilangan heksadesimal dari 0000 sampai FFFF.

- **Boolean**

  Tipe data Boolean terdiri dari dua nilai saja, yaitu *true* dan *false*. Boolean sangat penting untuk mengevaluasi suatu kondisi.

1.1.2 Tipe Data Referensi

Kelebihan pemrograman dengan orientasi objek adalah dapat mendefinisikan tipe data baru yang merupakan objek dari *class* tertentu. Tipe data ini digunakan untuk mereferensikan objek atau class tertentu, seperti String.

- **Variabel**

  Variabel merupakan *container* yang digunakan untuk menyimpan suatu nilai pada sebuah program dengan tipe tertentu. Untuk mendefinisikan variabel, suatu identifier dapat digunakan untuk menamai variabel tersebut.

- **Identifier**

  Identifier adalah kumpulan karakter yang dapat digunakan untuk menamai variabel, method, class, interface, dan package. Dalam pemrograman Java identifier bisa disebut sah apabila diawali dengan :

  - Huruf /abjad
  - Karakter Mata Uang
  - Underscore(_)

  Identifier dapat terdiri dari :

  - Huruf / abjad
  - Angka
  - Underscore (_)

  Identifier tidak boleh mengandung @, spasi atau diawali dengan angka serta tidak boleh menggunakan *keyword* yang telah digunakan di pemrograman java. Selain karakter, Unicode juga dapat digunakan sebagai identifier.

1.2 Mendeklarasikan Variabel

Sintaks dasar :

[tipe data] [nama variabel]

Menuliskan tipe data dari variabel, contoh :

```
int bilangan;
char karakter;
float bildesimal;
boolean status;
```

{Setelah dideklarasikan sesuai dengan tipe data, selanjutnya memberi nilai variabel tersebut dengan tanda =}

```
Bilangan = 20;
Karakter = 'k';
Bildesimal = 22.2f;
Status = true;
```

{melakukan deklarasi tipe data dan memberi nilai variabel dalam satu baris}

```
int bilangan = 20;
char karakter = 'k';
boolean status = true;
```

{membuat variabel menjadi konstanta sehingga tidak bisa diubah lagu nilainya dengan menambahkan keyword sebelum tipe data}

```
Final int a = 10;
Final float pajak = 15.5;
```

{membuat agar konstanta dapat diakses oleh kelas lain tanpa harus membuat objek terlebih dulu dilakukan penambahan modifier public dan keyword static}

```
Public static final a = 10;
```

2. Operator

Operator adalah simbol khusus yang menyajikan operasi khusus pada satu, dua, atau tiga operand dan kemudian mengembalikan hasilnya.

Jenis operator antara lain :

2.1 Operator Aritmatika

Operator ini digunakan pada operasi-operasi aritmatika seperti penjumlahan, pengurangan, pembagian dan lain-lain.

Tabel 1.3 Operator Aritmatika

Operator	Keterangan
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Modulus (sisa bagi)
++	Increment (menaikkan nilai dengan 1)
--	Decrement (menurunkan nilai dengan 1)

2.2 Operator Relasional

Untuk membandingkan 2 nilai (variabel) atau lebih digunakan operator relasional, dimana operator ini akan mengembalikan atau menghasilkan nilai *True* atau *False*.

Tabel 1.4 Operator Relasional

Operator	Keterangan
==	Sama dengan
!=	Tidak sama dengan
>	Lebih besar
<	Lebih kecil
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan

2.3 Operator Kondisional

Operator ini menghasilkan nilai yang sama dengan operator relasional, hanya saja penggunaannya lebih pada operasi – operasi Boolean.

Tabel 1.5 Operator Kondisional

Operator	Keterangan
&&	Operasi AND
||	Operasi OR
^	Operasi XOR
!	Operasi NOT (Negasi)

Tabel 1.6 Contoh Operasi Kondisional

A	B	A&&B	A||B	A^B	!A
True	True	True	True	False	False
TRUE	False	False	True	True	False
False	True	False	True	True	True
False	False	False	False	False	True

2.4 Operator Shift dan Bitwise

Kedua operator ini digunakan untuk memanipulasi nilai dari bitnya, sehingga diperoleh nilai yang lain.

Tabel 1.7 Operator Shift dan Bitwise

Operator	Keterangan
&	Operasi bitwise AND
|	Operasi bitwise OR
^	Operasi bitwise XOR
~	Operasi bitwise NOT
>>	Operasi shift right (geser ke kanan sebanyak n bit)
>>>	Operasi shift right zero fill
<<	Operasi shift left (geser ke kiri sebanyak n bit)

Tabel 1.8 Contoh Operasi Bitwise

A	B	A&B	A|B	A^B	~A
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

2.5 Operator Assignment

Operator assignment dalam java digunakan untuk memberikan suatu nilai ke sebuah variabel. Operator assignment hanya berupa '=', namun selain itu dalam Java beberapa shortcut assignment operator yang penting.

Tabel 1.9 Operator Assignment

Operator	Contoh	Ekivalen dengan
+=	b+=a	b=b+a
-=	b-=a	b=b-a
*=	b*=a	b=b*a
/=	b/=a	b=b/a
%=	b%=a	b=b%a
&=	a&=b	a=a&b
|=	a|=b	a=a|b
^=	a^=b	a=a^b
<<=	a<<=b	a=a<<b
>>=	a>>=b	a=a>>b
>>>=	a>>>=b	a=a>>>b

Contoh 1:

```
public class contoh1a {
public static void main (String[]args){
int x, y, c;
x=4&6;
y=5>>>2;
c=(2+4)*6;
System.out.println(y);
System.out.println(c);
System.out.println(x);
```

```
public class contoh1b {
public static void main (String[]args) {
int a = 20;
int b = 10;
System.out.println("Hasil dari a%b =" +(a%b));
System.out.println("Hasil dari a*b =" +(a*b));
System.out.println("Hasil dari a ditambah" +(a++));
}}
```

Contoh 2:

```
public class Konversi{
public static void main(String args[]){
float m, cm, inci;
m = 30;
cm = m * 100;
inci = m * 100 / 2.54f;
System.out.println("Ukuran dalam CM = " + cm);
System.out.println("Ukuran dalam Inci = " + inci);}}
```

3. Kondisional dan Pilihan

3.1 IF

Statement if memungkinkan sebuah program untuk dapat memilih beberapa operasi untuk dieksekusi, berdasarkan beberapa pilihan. Terdapat tiga jenis statement If diantaranya :

- **If**

  Bentuk If adalah yang paling sederhana, mengandung suatu pernyataan tunggal yang dieksekusi jika ekspresi bersyarat adalah benar.

  Sintaks dasar:

```
If (ekspresi_kondisional) {
statement1;
statement2;
…
}
```

- **If, else**

  Untuk melakukan beberapa operasi yang berbeda jika salah satu ekspresi kondisional bernilai salah, maka digunakan statement else. Bentuk if-else memungkinkan dua alternatif operasi pemrosesan.

  Sintaks dasar:

```
If (ekspresi_kondisional) {
   statement1;
   statement2;
…
}else {
   statement1;
   statement2;
…
}
```

- **If, else if, else**

  Bentuk if, else if, else memungkinkan untuk tiga atau lebih alternative pemrosesan.

  Sintaks dasar:

```
If (ekspresi_kondisional) {
   statement1;
   statement2;
…
}else if (ekspresi_kondisional){
   statement1;
   statement2;
…
} else {
   statement1;
   statement2;
…
}
```

Contoh :

1. **If**

```
public class IfSatuPilihan{
public static void main(String args[]){
int bil;
bil=0;
if (bil==0)
System.out.println("Bilangan Nol");
}}
```

2. **If, else**

```
import java.util.Scanner;
public class IfDuaPilihan{
public static void main(String args[]){
      Scanner masuk = new Scanner(System.in);
      int bil;
```

```
        System.out.print("Masukkan bilangan : ");
        bil=masuk.nextInt();
        if (bil==0)
        System.out.println("Bilangan Nol");
        else
        System.out.println("Bilangan Bukan Nol");}}
```

3. If, else if, else

```
import java.util.Scanner
public class Buah{
public static void main(String args[]){Scanner masuk =
new Scanner(System.in);
int pil;
System.out.print("Masukkan pilihan : ");
pil = masuk.nextInt();
if (pil==1)
System.out.println("Buah Pepaya");
else if(pil==2)
System.out.println("Buah Stroberi");
else if(pil==3)
System.out.println("Buah Apel");
else
System.out.println("Bukan Buah deh");}}
```

3.2 Switch

Switch adalah pernyataan yang digunakan untuk menjalankann salah satu pernyataan dari beberapa kemungkinan statement untuk dieksekusi, berdasarkan nilai dari sebuah ungkapan dan nilai penyeleksi. Setiap ungkapan diungkapkan dengan sebuah nilai integral konstan, seperti sebuah nilai dengan tipe byte, short, int atau char.

Sintaks dasar :

```
switch (ekspresi){
      case value1:
                  statement1;
                  statement2;
                  break;
  case value2:
                  statement1;
                  statement2;
                  break;
            …
            [default:]
                  statement1;
                  statement2;
      }
```

Keterangan

Case : menandai posisi kode dimana eksekusi dilaksanakan.

Value1, dst : konstanta integer atau karakter ataupun ekspresi yang mengevaluasi keduanya.

Default : berfungsi sama seperti else pada statement if.

Break : dapat menghentikan perulangan walaupun kondisi untuk berhenti belum terpenuhi.

Continue :dengan statement ini kita bisa melewatkan operasi yang dilakukan dalam iterasi sesuai dengan kondisi tertentu.

Contoh :

```
import java.util.Scanner;
public class CaseJurusan{
    public static void main(String args[]){
    Scanner masuk = new Scanner(System.in);
          int pil;
          System.out.print("Masukkan pilihan :S1 TT ");
          pil = masuk.nextInt();
          switch (pil) {
           case 1:System.out.println("S1 TE");
            break;
            case  2:System.out.println("S1 SK");
            break;
           case  3:System.out.println("D3 TT");
            break;
           case  4:System.out.println("D3 IF");
            break;
           case 5:System.out.println("S1 TI");
            break;
          default:
            System.out.println("Input salah!");
            break;
            }
          }
          }
```

MODUL 2
Array, Method, dan Pengulangan

A. TUJUAN

- Mahasiswa memahami tentang pengertian array serta dapat membuat program dengan menggunakan array.
- Mahasiswa memahami tentang pengertian sub program dan dapat membuat sub program sederhana.
- Mahasiswa dapat menyelesaikan permasalahan dengan menggunakan perulangan, baik rekursif, for, do..while dan while.

B. PERANGKAT

- NetBeans IDE 7.2

C. DASAR TEORI

1. Array / Larik

1.1 Definisi Array / Larik

Larik adalah sebuah struktur data yang terdiri dari data yang bertipe sama. Ukuran larik bersifat tetap, larik akan mempunyai ukuran yang sama pada saat sekali dibuat. Larik dalam Java adalah obyek, disebut juga sebagai tipe referensi. Sedangkan elemen dalam larik Java bisa primitif atau referensi. Posisi dari larik biasa disebut sebagai elemen. Elemen larik dimulai dari 0 (nol). Penyebutan larik diberikan dengan cara menyebutkan nama lariknya dan diikuti dengan indeksnya, dimana indeks dituliskan diantara tanda kurung siku. Gambar 1. memperlihatkan gambaran larik dengan 10 elemen, dimana setiap elemennya bertipe integer, dengan nama A.

Tabel 2.1 Contoh Array

Nama	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
Isi larik	12	-56	23	45	-16	-2	85	41	15	20

1.2 Deklarasi dan Menciptakan Array / Larik

Sebagai sebuah obyek, larik harus diciptakan dengan menggunakan kata cadang new. Deklarasi dan penciptaan variabel larik gambar 1 adalah sebagai berikut.

```
int A[] = new int[10];
```

larik dideklarasikan dan langsung diciptakan . Atau

```
int A[];
A = new int[10];
```

larik dideklarasikan, baru pada pernyataan berikutnya larik diciptakan.

Contoh:

```
import java.util.Scanner;
public class Larik1
{
    public static void main (String args[])
    {
      Scanner masuk=new Scanner(System.in);
      float nilai[]=new float[5];
      System.out.println("masukkan 5 buah data nilai");
        for(int i=0;i<5;i++)
        {
            System.out.print("Data ke"+(i+1)+": ");
            nilai[i]=masuk.nextFloat();
        }
      System.out.println("data nilai yang dimasukkan");
         for(int i=0;i<5;i++)
             System.out.println(nilai[i]);

    }
}
```

Hasil Output:

```
masukkan 5 buah data nilai
Data ke1: 2
Data ke2: 4
Data ke3: 5
Data ke4: 7
Data ke5: 9
data nilai yang dimasukkan
2.0
4.0
5.0
```

```
7.0
9.0
```

1.3 Array Multi Dimensi (N-Dimensi)

Kita juga bisa membuat variabel larik yang tipe elemennya adalah larik. Dengan cara demikian, kita membuat larik dua dimensi. Dengan larik dua dimensi, maka kita mempunyai elemen yang berindeks tidak hanya satu, tetapi dua. Kita bisa membayangkan larik dua dimensi tersebut seperti sebuah tabel yang berisi baris dan kolom. Penyebutan sel tabel selalu diikuti dengan penyebutan baris berapa dan kolom berapa.

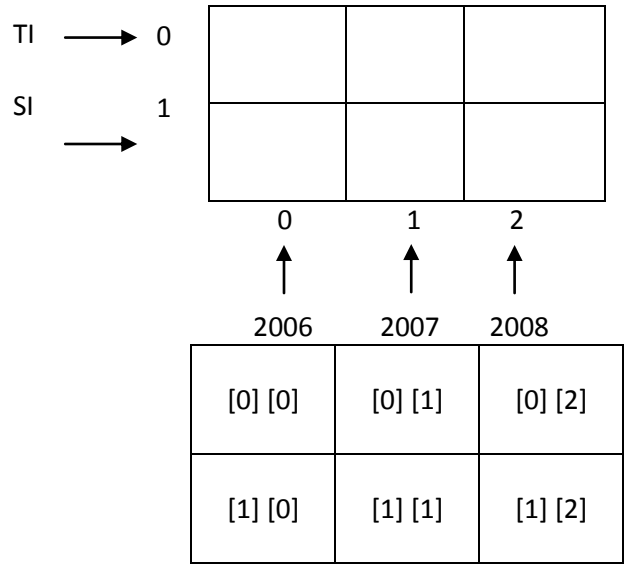
Contoh :

Diberikan data kelulusan mahasiswa sebuah perguruan tinggi sebagai berikut.

Tabel 2.2 Contoh Array Multi Dimensi

Jurusan	2006	2007	2008
Teknik Informatika	110	125	135
Sistem Informasi	56	75	80

```
int data_lulus [2] [3]
```



Contoh:

```
public class ArrayDimensiDua
{
	public static void main(String [] args)
	{
	int [][] piksel = new int[2][3];
	// mengisi elemen tertentu
	piksel[0][0] = 70;
	piksel[0][1] = 18;
	piksel[0][2] = 45;
	piksel[1][0] = 75;
	piksel[1][1] = 66;
	piksel[1][2] = 89;
	//menampilkan elemen array
	int i,j;
	for(i=0;i<2;i++){
		for (j=0; j<3;j++)
		System.out.print(piksel[i][j] +" ");
		System.out.println("");
	}
	}
}
```

Hasil Output :

```
70 18 45
75 66 89
```

1.4 String

String adalah kelas yang menangani deretan karakter. Kelas ini mendukung sejumlah metode yang sangat berguna untuk memanipulasi string, misalnya untuk mengkonversikan setiap huruf kecil menjadi huruf besar atau sebaliknya, memperoleh jumlah karakter dan sebagainya. String sebenarnya merupakan *class* yang terdapat pada library Java.

Kelas string memiliki banyak konstruktor, seperti tabel berikut:

Tabel 2.3 Konstruktor String

Konstruktor	Keterangan
String()	**Menciptakan obyek string yg berisi string kosong (jumlah karakter = 0)**
String(char[]v)	**Menciptakan obyek string yg berisi string yg berasal dari array yg dirujuk oleh v**
String(String v)	**Menciptakan obyek string yg isinya sama dengan obyek string argumennya**

Metode dalam kelas string memperlihatkan sejumlah metode penting dalam kelas string, seperti :

- copyValueOf(char data[])
- copyValueOf(char data[], int offset, int jum)
- valueOf(boolean b)
- valueOf(double c)
- cocat(String s)
- length()
- trim()
- dan lain-lain

Kelas StringBuffer adalah kelas yg menyimpan string yang konstan, begitu obyek string telah diciptakan maka string tidak dapat diubah. Konstruktor kelas ini antara lain :

- StringBuffer() digunakan untuk menciptakan StringBuffer yang kosong
- StringBuffer(int n) digunakan untuk menciptakan StringBuffer dengan n karakter
- StringBuffer(String s) digunakan untuk menciptakan StringBuffer dengan string berupa s.

Contoh :

```
public class ContohString
{
      public static void main(String args[])
      {
            byte data[] = new byte[6];
            data[0] = 64;
            data[1] = 65;
            data[2] = 66;
            data[3] = 67;
            data[4] = 68;
            data[5] = 69;
            String s1 = "Selamat Pagi";
            String s2 = new String("Good Morning");
            String s3 = new String(data);
            String s4 = new String(data, 2, 3);
            System.out.println("s1 = " + s1);
            System.out.println("s2 = " + s2);
            System.out.println("s3 = " + s3);
            System.out.println("s4 = " + s4);
      }
}
```

Hasil output :

```
s1 = Selamat Pagi
s2 = Good Morning
s3 = @ABCDE
s4 = BCD
```

Pada program di atas, pernyataan seperti :

```
String s1 = "Selamat Pagi";
```

Sebenarnya identik dengan :

```
String s1 = new String("Selamat Pagi");
```

Pernyataan

```
String s3 = new String(data);
```

akan membuat string yang tersusun atas karakter-karakter yang nilainya sama seperti elemen-elemen pada array data, maka s3 berisi string @ABCDE adalah karakter @ = 64, A=65 dan seterusnya.

Pernyataan :

```
String s4 = new String(data, 2, 3);
```

Angka 3 menyatakan jumlah karakter yg menyusun string dan angka 2 menyatakan karakter pertama pada string, hasil diambil pd indeks ke-2 array.

2. Method

2.1 Method Tanpa Variabel

Method (atau dalam beberapa bahasa pemrograman sering disebut fungsi atau prosedur) adalah sub program yang membiarkan seorang programer untuk membagi program dengan membagi masalah ke dalam beberapa sub masalah yang bisa diselesaikan secara modular. Dengan cara demikian, maka pembuatan program bisa lebih dimanajemen.

Kelas (*class*) adalah program java yang akan dieksekusi. Method ada di dalam kelas. Java mempunyai kumpulan kelas yang sudah dimiliki yang tersimpan di dalam paket-paket. Kumpulan kelas tersebut ada di dalam *Java Application Interface* (Java API) atau *Java class libraries* dan beberapa *libraries* lainnya.

FORMAT METHOD SECARA UMUM

```
tipe_return-value
nama_method(parameter1,parameter2,…,parameterN)
{
```

```
      deklarasi dan pernyataan;
}
```

Elemen yang diperlukan dari deklarasi method adalah tipe kembalian method, nama, kurung buka dan tutup ( ) dan isi method yang diawali dan diakhiri dengan kurung kurawal buka dan tutup { }. Secara umum, deklarasi method mempunyai 6 komponen, yaitu:

1. Modifier - seperti public, private, dan yang lain yang akan kita pelajari kemudian.
2. Tipe kembalian (*return type*)—tipe data dari nilai yang dikembalikan oleh method, atau void jika method tidak mempunyai nilai kembalian.
3. Nama method—aturan untuk penamaan field diterapkan untuk nama method tetapi kesepakatannya adalah sedikit berbeda.
4. Daftar parameter – pemisah antar parameter input adalah koma, diawali oleh tipe datanya, yang diletakkan diantara tkita kurung (…daftar parameter….). Jika tidak ada parameter, harus menggunakan kurung buka tutup saja ( ).
5. Daftar exception—tidak akan masuk dalam pembahasan di sini
6. Isi method, diletakkan di antara kurung kurawal buka dan tutup { }—kode-kode method, termasuk deklarasi variabel lokal ada di sini.

Contoh:

```
public class Fungsi2
{
      public static void kalimat()
{
      System.out.println("Di dalam method kalimat");
      }

      public static void main(String args[])
{
            kalimat();
            System.out.println("Di dalam main");
            kalimat();
      }
}
```

Hasil output:

```
Di dalam method kalimat
Di dalam main
Di dalam method kalimat
```

2.2 Method dengan Variabel

Method (atau dalam beberapa bahasa pemrograman sering disebut fungsi atau prosedur) adalah sub program yang membiarkan seorang programer untuk membagi program dengan membagi masalah ke dalam beberapa sub masalah yang bisa diselesaikan secara modular. Dengan cara demikian, maka pembuatan program bisa lebih dimanajemen.

Contoh :

```
public class FungsiParameter
{
public static int jumlah(int a){
return a;
}
public static void main(String args[]){
System.out.println("Hasil pemanggilan method jumlah ");
System.out.println(jumlah(5));
}
}
```

Hasil Output:

```
Hasil pemanggilan method jumlah
5
Press any key to continue . . .
```

Parameter pada baris kedua disebut sebagai parameter formal, dan pada baris ke 8 disebut parameter aktual.

Ada 2 buah parameter yaitu:

- parameter formal adalah parameter yang tertulis dalam definisi method
- Parameter aktual parameter yang berada pada inputan langsung pada saat penggunaan method tersebut.

Parameter bisa lebih dari satu dengan dipisahkan tanda koma,. Yang perlu diperhatikan pada saat pemanggilan method adalah jumlah, urutan dan tipe

parameter aktual harus sesuai dengan jumlah urutan dan tipe parameter formal.

Pemberian Variabel Dalam Method

Ada dua tipe data variable passing pada method, yaitu *pass-by-value* dan *pass-by-reference.*

- ***Pass-by-value***

  Ketika *pass-by-value* terjadi, method membuat sebuah salinan dari nilai variable yang dikirimkan ke mthod. Walaupun demikian method tidak dapat secara langsung memodifikasi nilai variable pengirimnya meskipun parameter salinannya sudah dimodifikasi nilainya di dalam method.

- ***Pass-by-reference***

  Ketika sebuah *pass-by-reference* terjadi, alamat memori dari nilai pada sebuah variable dilewatkan pada saat pemanggilan method. Ini tidak seperti pada *pass-by-value*, method dapat memodifikasi variable asli dengan menggunakan alamat memori tersebut, meskipun berbeda nama variable yang digunakan dalam method dengan variable aslinya, kedua variable ini menunjukkan lokasi dari data yang sama.

3. Pengulangan

3.1 Pengulangan dengan while

Pernyataan ini berguna untuk memproses suatu pernyataan atau beberapa pernyataan beberapa kali. Selama ungkapan bernilai benar, pernyataan akan selalu dikerjakan.

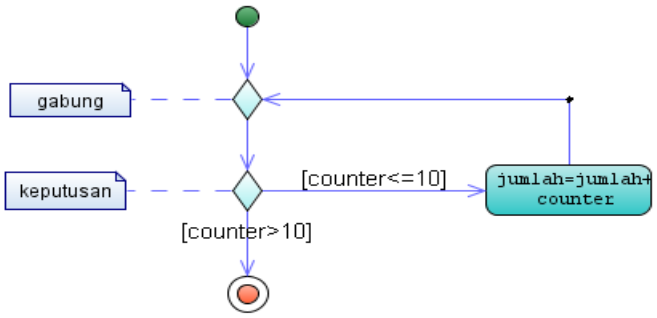
Bentuknya :

```
while (ungkapan)
      Pernyataan;
```

Keterangan :

- bagian pernyataan akan diekseskusi selama ungkapan dalam **while** bernilai benar.
- Pengujian terhadap ungkapan pada **while** dilakukan sebelum bagian pernyataan.
- Kemungkinan pernyataan pada **while** tidak dijalankan sama sekali, jika ketemu kondisi yang pertama kali bernilai salah.

Activity diagramnya adalah seperti gambar berikut :



Gambar 2.1 Pengulangan dengan while

Catatan :
Pernyataan perulangan dengan while akan selalu dikerjakan jika ungkapan selalu benar. Oleh karena itu, kita harus membuat kondisi suatu saat ungkapan bernilai salah agar perulangan berakhir.

Contoh:

```
import java.util.Scanner;
      public class UlangWhile1
      {
            public static void main(String args[])
            {
                  Scanner masuk = new Scanner(System.in);
                  int bil;
                  bil=1;
                  while (bil<=5) {
                        System.out.println(bil);
                        bil++;
                  }
            }
      }
```

Hasil Output:

```
1
2
3
```

```
4
5
```

3.2 Pengulangan dengan do-while

Seperti halnya perulangan dengan while, perulangan dengan do ... while ini juga digunakan untuk mengerjakan sebuah atau sekelompok pernyataan berulang-ulang. Bedanya dengan while adalah pernyataan do ... while akan mengecek kondisi di belakang, sementara while cek kondisi ada di depan.

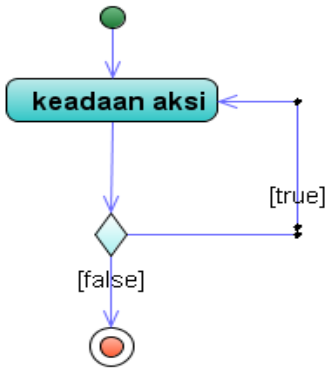
Bentuknya :

```
do
  {
        pernyataan1;
        pernyataan2;
        ..........
       pernyataan_N;
  }
  while (ungkapan)
```

Keterangan :

- Bagian pernyataan1 hingga pernyataanN dijalankan secara berulang sampai ungkapan bernilai salah.
- Pengujian ungkapan dilakukan setelah bagian pernyataan, maka pada pernyataan **do ... while** minimal akan dijalankan sekali, karena begitu masuk ke blok perulangan, tidak ada cek kondisi tetapi langsung mengerjakan pernyataan.

Activity diagramnya :



Gambar 2.2 Pengulangan dengan do-while

Contoh:

Buatlah program mencetak konversi suhu dari celcius ke fahreinhet mulai dari 1 sampai 10 dengan membuat tabel.

```
public class UlangDo2
{
      public static void main(String args[])
      {
         int c;
         double f;
         System.out.println("-----------------------");
         System.out.println("CELCIUS     FAHREINHEIT");
         System.out.println("-----------------------");
         c=1;
         do
{
         f=1.8 * c + 32;
         System.out.println("Celcius:"+c+"Fahreinhet:+f);
         c++;
         } while (c<=10);
         System.out.println("-----------------------");
      }
}
```

Hasil Output :

```
-----------------------
 CELCIUS          FAHREINHEIT
-----------------------
Celcius : 1 Fahreinhet : 33.8
Celcius : 2 Fahreinhet : 35.6
Celcius : 3 Fahreinhet : 37.4
Celcius : 4 Fahreinhet : 39.2
Celcius : 5 Fahreinhet : 41.0
Celcius : 6 Fahreinhet : 42.8
Celcius : 7 Fahreinhet : 44.6
Celcius : 8 Fahreinhet : 46.4
Celcius : 9 Fahreinhet : 48.2
Celcius : 10 Fahreinhet : 50.0
-----------------------------
Press any key to continue . . .
```

3.3 Pengulangan dengan for

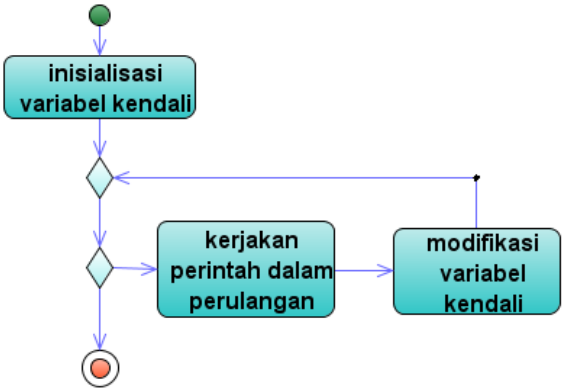
Sama seperti pernyataan perulangan while dan do...while, pernyataan for juga digunakan untuk mengerjakan pernyataan atau sekelompok pernyataan secara berulang. Bedanya adalah dengan pernyataan for perulangan akan dikerjakan dalam hitungan yang sudah pasti, sementara while dan do...while tidak.

Bentuknya :

```
for (ungkapan1;ungkapan2;ungkapan3)
      Pernyataan;
```

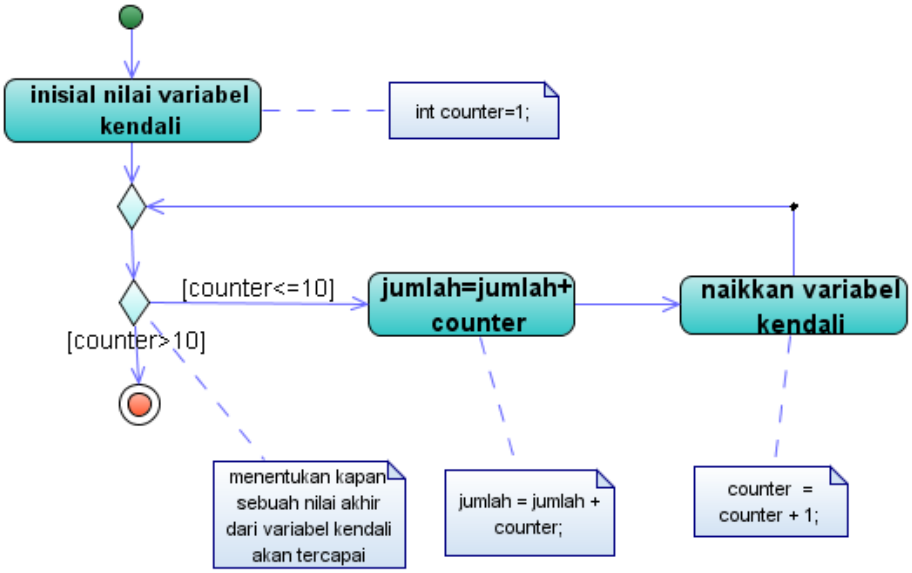
Keterangan :

- ungkapan1 merupakan pernyataan inisialisasi
- ungkapan2 sebagai kondisi yang menentukan pengulangan terhadap pernyataan atau tidak
- ungkapan3 digunakan sebagai pengatur variabel yang digunakan didalam ungkapan1



Gambar 2.3 Activity Diagram untuk perulangan dengan FOR

Contoh activity diagram untuk perulangan dengan for:



Gambar 2.4 Contoh Activity Diagram untuk perulangan dengan FOR

Contoh:

```
public class UlangFor
{
    public static void main (String args[])
    {
      int bil;
      for (bil=1;bil<=5;bil++)
        System.out.println(bil);
    }
}
```

Maka akan ditampilkan hasil Output seperti berikut:

```
1
2
3
4
5
```

4. Rekursi

Relasi perulangan adalah persamaan-persamaan untuk menentukan satu atau lebih urutan-urutan secara rekursif. Beberapa relasi perulangan tertentu dapat "diselesaikan" untuk mendapatkan definisi bukan-rekursif.

Penggunaan rekursi dalam suatu algoritma memiliki kelebihan dan kekurangan. Kelebihan utamanya adalah biasanya kesederhanaan. Kekurangan utamanya adalah terkadang algoritma tersebut membutuhkan memori yang sangat banyak jika kedalaman rekursi sangat besar. Rekursif dapat mendefinikan barisan, fungsi dan himpunan.

Langkah-langkah untuk mendefinisikan secara rekursif:

1. Langkah basis : Tentukan anggota awalnya.
2. Langkah rekursif : Bentuk aturan untuk membuat anggota baru dari anggota yang telah ada.

Dalam bahasa pemrograman, rekursif adalah proses dimana fungsi memangil dirinya sendiri. Berikut ini contoh pemrograman factorial dengan rekursif.

```
public class Factorial{
   public static void main (String args[]){
         Factorial f = new Factorial();
         System.out.print("8! =  ");
         System.out.print(f.HitungFactorial(8));
   }

   public int HitungFactorial(int x){
         if(x==1){
```

```
                return 1;
          }
          else {
                return x * HitungFactorial(x-1);
          }
    }
}
```

Contoh Program Modul 2 yang mencakup array, method, dan pengulangan:

```
package matrik;
import java.util.Scanner;
public class Main {
public void kalimat(){
    System.out.print("Isi matriks adalah : ");
}
public int hitungluas(int p,int l){
    int luas;
    luas=p*l;
    return luas;
}
public int hitungvolume(int p,int l,int t){
    int volume;
    volume=p*l*t;
    return volume;
}
    public static void main(String[] args) {
        int p,l,t;
        int data[];
        Scanner masuk=new Scanner(System.in);
      System.out.print("masukkan panjang : ");
      p=masuk.nextInt();
      System.out.print("masukkan lebar : ");
      l=masuk.nextInt();
      System.out.print("masukkan tinggi : ");
      t=masuk.nextInt();
      data=new int[3];
      Main saya=new Main();
      data[0]=saya.hitungluas(p,l);
      data[1]=saya.hitungvolume(p,l,t);
      data[2]=10;
      int bil=0;
      while (bil<=2) {
                          saya.kalimat();
                          System.out.println(data[bil]);
                bil=bil+1;
             }
    }
}
```

keluaran program di atas:

```
masukkan panjang : 2
masukkan lebar : 3
masukkan tinggi : 4
Isi matriks adalah : 6
Isi matriks adalah : 24
Isi matriks adalah : 10
```

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
```

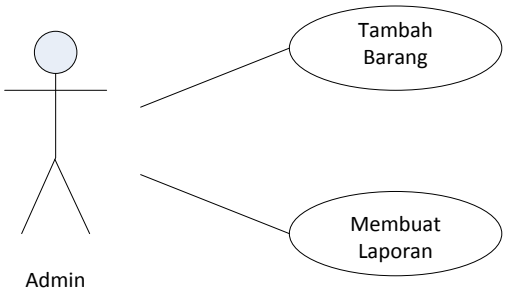
9. UML (*Unified Modeling Language*)

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

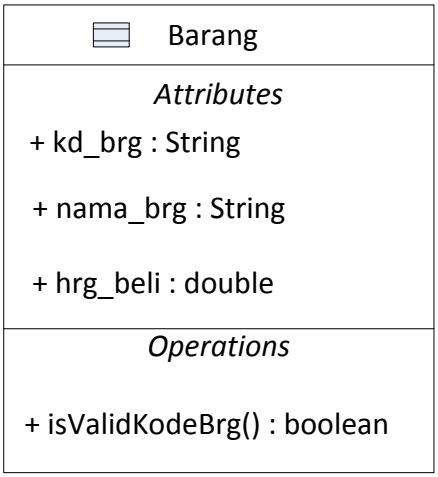
Menggambarkan fungsionalitas dari sebuah sistem (apa fungsinya), yang merepresentasikan sebuah interaksi antara aktor dan sistem (sebuah pekerjaan). Misalnya menambah data/membuat laporan.



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 :

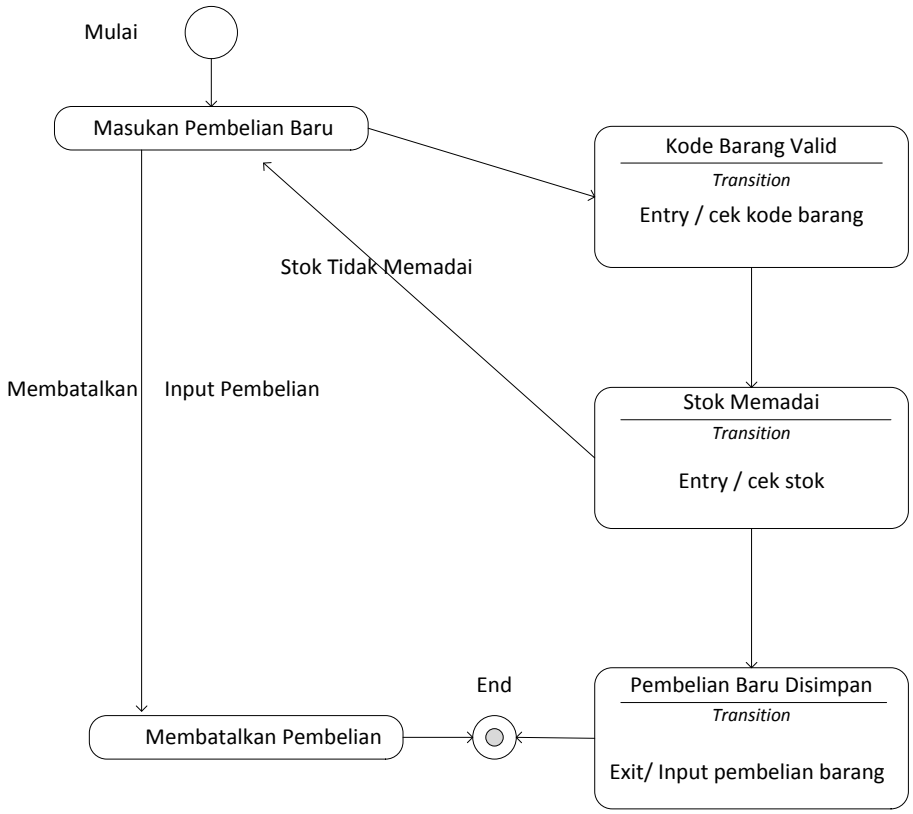
a. Asosiasi sederhana : bentuk asosiasi sederhana (———).
b. Agregasi yaitu hubungan yang menyatakan bagian, biasanya hubungan data master dan detailnya. Misal satu pembelian terdiri dari sejumlah item barang (◆———).
c. *Navigability* : menunjukan arah query/komunikasi antar objek, bisa satu atau dua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

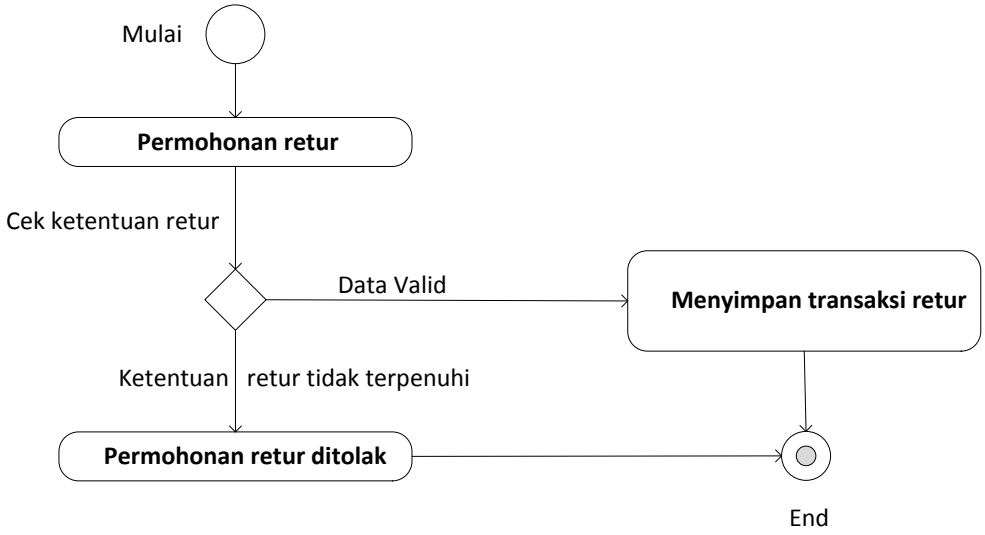
sesuai kondisinya. Transisi antar-state umumnya memiliki kondisi yang merupakan syarat terjadinya transisi yang bersangkutan.



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.

Modul 4
OOP-2 dan Pewarisan

A. TUJUAN

- Praktikan mampu memahami dan mengerti lebih dalam tentang OOP terutama tentang *inner class*.
- Praktikan mampu mengerti konsep dasar dan implementasi tentang pewarisan, *polymorphisme, abstract class* dan *enkapsulasi*.

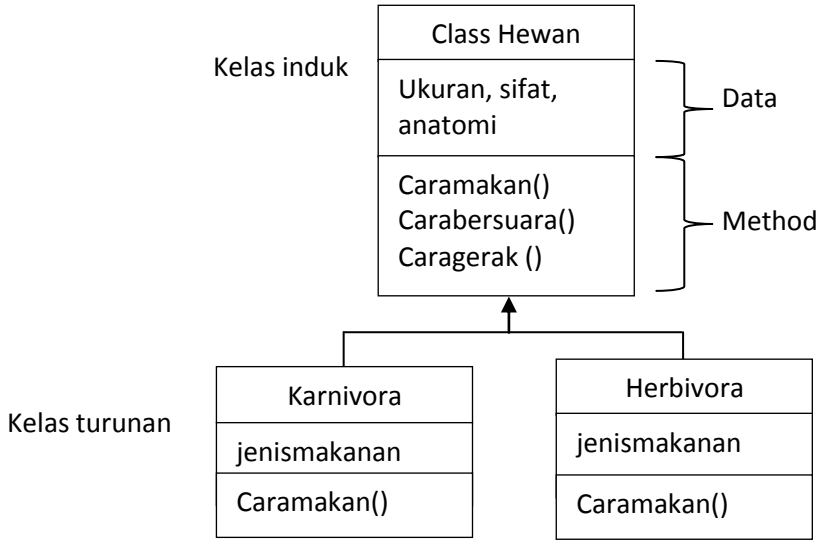
B. PERANGKAT

NetBeans IDE 7.2

C. DASAR TEORI

1. Pewarisan (*Inheritance*)

Pewarisan merupakan proses penciptaan kelas baru dengan mewarisi karakteristik kelas yang sudah ada (biasa disebut kelas induk), ditambah dengan karakteristik unik kelas baru tersebut (biasa disebut turunan). Dalam java, kelas induk ini dinamakan *superclass* dan kelas turunan dinamakan *subclass*.



Gambar 4.1 Contoh pewarisan pada kelas hewan

Hewan adalah *superclass* dari karnivora dan herbivora. Kelas turunan karnivora dan herbivora ini memiliki data dan method yang dimiliki kelas hewan.

Dalam java, format penulisan untuk membuat *subclass* adalah:

```
class namasuperclass {
// body kelas
}

class  namasubclass extends namasuperclass{
// body kelas
}
```

Contoh Program:

```
public class PersegiPanjang{
    private int panjang;
    private int lebar;

    public void setPanjang(int p){
    panjang=p;
    }
    public void setLebar(int l){
    lebar=l;
    }
    public int getPanjang(){
    return panjang;
    }
    public int getLebar(){
    return lebar;
    }
    public int Luas(){
    int luas=panjang*lebar;
    return luas;
    }
}
```

Kemudian kita buat kelas Balok yang merupakan turunan dari kelas PersegiPanjang

```
public class Balok extends PersegiPanjang{
    private int tinggi;
    public void setTinggi(int t){
    tinggi=t;
    }
    public int getTinggi(){
    return tinggi;
    }
    public int Volume(){
    int v=getPanjang()*getLebar()*tinggi;
    return v;
    }
}
```

Sedangkan untuk program utamanya:

```
public class DemoPewarisan{
   public static void main(String args[]){
         PersegiPanjang a= new PersegiPanjang();
         a.setPanjang(5);
         a.setLebar(5);
         System.out.println("");
         System.out.println("Contoh Program Pewarisan");
         System.out.println("");
         System.out.println("Superclass PersegiPanjang");
         System.out.println(" Panjang  : "+a.getPanjang());
         System.out.println(" Lebar  : "+a.getLebar());
         System.out.println(" Luas  : "+a.Luas());
         System.out.println("");

         Balok b= new Balok();
         /* kelas balok tinggal memanggil method yang ada didalam
         kelas persegi */
         b.setPanjang(4);
         b.setLebar(3);
         b.setTinggi(5);
         System.out.println("Subclass Balok");
         System.out.println(" Panjang  : "+b.getPanjang());
         System.out.println(" Lebar  : "+b.getLebar());
         System.out.println(" Tinggi  : "+b.getTinggi());
         System.out.println(" Volume  : "+b.Volume());
   }
}
```

Sehingga hasil keluaran programnya adalah:

```
Contoh Program Pewarisan

Superclass PersegiPanjang
Panjang  : 5
Lebar  : 5
Luas  : 25

Subclass Balok
Panjang  : 4
Lebar  : 3
Tinggi  : 5
Volume  : 60
```

Pewarisan menggunakan kata kunci **super** dapat dilakukan dengan format penulisan:

```
super(daftarParameter)
```

Contoh Program:

```
class Kotak(int p, int l, int t) {
   panjang = p;
   lebar = l;
   tinggi = t;
}

class  KotakPejal extends Kotak{
   private double berat;
   KotakPejal(int p, int l, int t, int b) {
   super(p, l, t);  // memanggil constructor kelas Kotak
   berat = b;
   }
}
```

2. Enkapsulasi

Enkapsulasi merupakan proses pembungkusan (encapsulation) dari suatu kelas atau biasa disebut *information hiding*. Terdapat tiga tingkat akses yang terkait dengan enkapsulasi, yaitu:

- Private

  Ketika mendeklarasikan data dan method dengan private, maka data dan method tersebut hanya dapat diakses oleh kelas yang memilikinya saja.

- Protected

  Ketika mendeklarasikan data dan method dengan protected, maka data dan method tersebut dapat diakses oleh kelas yang memilikinya dan kelas-kelas yang masih memiliki hubungan turunan.

- Public

  Data dan method yang bersifat public akan dapat diakses oleh semua bagian dalam program. Semua bagian dalam program adalah semua kelas yang memiliki hubungan turunan maupun yang tidak memiliki hubungan sama sekali

Tabel 4.1 Perbedaan Tingkat Akses Enkapsulasi

	Private	Protected	Public
Akses dalam satu kelas	bisa	bisa	bisa
Akses dalam kelas turunan	tidak	bisa	bisa
Akses dalam kelas bukan turunan	tidak	tidak	bisa

Contoh Program:

```
class musikPop {
  private String judulLagu;
// hanya dapat dikenali oleh kelas musikPop dan turunan-turunannya

   protected void setJudul(String nama) {
     judulLagu = nama;
   }
// hanya dapat dikenali oleh kelas A dan turunan-turunannya

  protected String getJudul () {
    return judulLagu;
  }
}


class musikJPop extends musikPop {
  private int tahunTerbit;
// constructor kelas B
  musikJPop(String judul, int tahun) {
//judulLagu = judul;  // SALAH, karena a tidak dikenali di sini
    setJudul(judul);  // menggunakan method setJudul()
    tahunTerbit = tahun;
  }
  public void showData() {
    // menggunakan method getJudul()
    System.out.println("Judul Lagu : " + getJudul());
    System.out.println("Tahun Terbit : " + tahunTerbit);
  }
}

class musikJazz {
  private String penyanyi;
  public void setPenyanyi(String nama) {
//setJudul('Indonesia Raya');     /* SALAH, setJudul() tidak
dikenal di sini */
    penyanyi = nama;}
   public String getPenyanyi() {
    return penyanyi;}
   public void showPenyanyi() {
 //System.out.println("Judul lagu : " + getJudul());// SALAH
    System.out.println("Penyanyi : " + penyanyi);}}


class DemoEnkapsulasi {
  public static void main(String[] args) {
// melakukan instansiasi terhadap kelas musikJPop
   musikJPop obj = new musikJPop("I Feel My Soul", 2008);
   obj.showData();
   obj.setJudul();
   System.out.println("Judul lagu : " + obj.getJudul());
  }}
```

Hasil running program di atas akan menampilkan:

```
Judul Lagu JPop : I Feel My Soul
Tahun Terbit : 2008
Judul Lagu Pop: Separuh Aku
```

3. Inner Class

Yaitu kelas yang dibuat di dalam kelas. Inner class ini dapat mengakses data dan method dari kelas yang berada di bagian luarnya, meskipun data bersifat private. Dan bagian private dari inner class tetap tidak dapat di akses oleh kelas di bagian luarnya.

Contoh Program:

```
class Luar {
  private int a;
   public void setA(int nilai) {
     a = nilai;
  }
   public int getA() {
     return a;
  }

// Mendefinisikan inner class
  class Dalam {
  private int x = 10;  // hanya dikenali oleh kelas Dalam
  public void showData() {
      System.out.println("Nilai a dari kelas Luar  : " + a);
      System.out.println("Nilai x dari kelas Dalam : " + x);
    }
}

public void showData() {
  // Baris di bawah ini SALAH, karena x tidak dikenal di sini
  //System.out.println("Nilai x dari kelas Dalam : " + x);

    Dalam obj = new Dalam();
    obj.showData();
  }
}
```

Sedangkan program utamanya:

```
class DemoInnerClass {
  public static void main(String[] args) {
Luar obj1  = new Luar();
//Dalam obj2 = new Dalam();  // SALAH

obj1.setA(5);
obj1.showData();
  }
}
```

4. Polymorfisme

Polimorfisme merupakan kemampuan suatu obyek untuk mengungkap banyak hal melalui satu cara yang sama. Misalnya kelas B, C, dan D adalah turunan kelas dari kelas A, maka kita dapat menjalankan method-method dari kelas B, C, dan D hanya dari obyek yang diinstansiasi dengan kelas A. Polimorfisme juga dapat diartikan sebagai suatu konsep pada bahasa pemrograman yang mengizinkan kelas induk untuk mendefinisikan sebuah *method general* untuk semua kelas turunannya dan selanjutnya kelas turunannya dapat memperbarui implementasi dari *method* tersebut secara lebih spesifik sesuai dengan karakteristiknya masing-masing.

5. Override

Yaitu suatu method yang ada pada kelas induk (*Superclass*) dan didefinisikan kembali oleh kelas turunan (*subclass*) dengan nama methode dan daftar parameter yang sama persis. Dan method pada kelas induknya tersebut akan disembunyikan keberadaannya. Jika kita panggil method yang sudah di-*override* dari instansiasi kelas turunannya, maka method yang dipanggil itu merupakan method dari kelas turunan tersebut, bukan method kelas induk lagi.

Perbedaan Override dengan Overload dapat dilihat dalam tabel berikut:

Tabel 4.2 Perbedaan Overload dan Override

	Overload	Override
Terdapat dalam satu kelas	ya	tidak
Terdapat dalam kelas turunannya	ya	Ya
Nama method dalam satu kelas	sama	-
Nama method pd kelas turunannya	sama	Sama
Jumlah parameter pd satu kelas	berbeda	-
Jumlah parameter pd kelas turunannya	berbeda	sama

Contoh Program:

```
public class Induk{
   public void panggilAku(){
       System.out.println("");
       System.out.println("Hallo, ini induk yang dipanggil");
   }
}

public class Anak{
   //method sama dengan method induk atau override
   public void panggilAku(){
       System.out.println("");
       System.out.println("Hallo, ini anak yang dipanggil");
   }
}

public class DemoOverride{
   public static void main(String args[]){
   Anak a= new Anak();
   a.panggilAku();
   }
}
```

Hasil keluaran ketika class DemoOverride dijalankan adalah:

```
Halo, ini anak yang dipanggil
```

6. Abstraksi

Yaitu penyembunyian kerumitan dari suatu proses. Biasa berbentuk kelas murni yang tidak boleh memiliki objek, dan satu/lebih method-methodnya yang *abstract* harus diimplementasikan (override) oleh kelas turunannya. Dan kelas abstrak ini tidak dapat diinstansiasi.

Format penulisan abstrak dalam java adalah

```
abstract class NamaKelas {
// deklarasi attribute
// definisi/prototype method
}
```

Contoh Program:

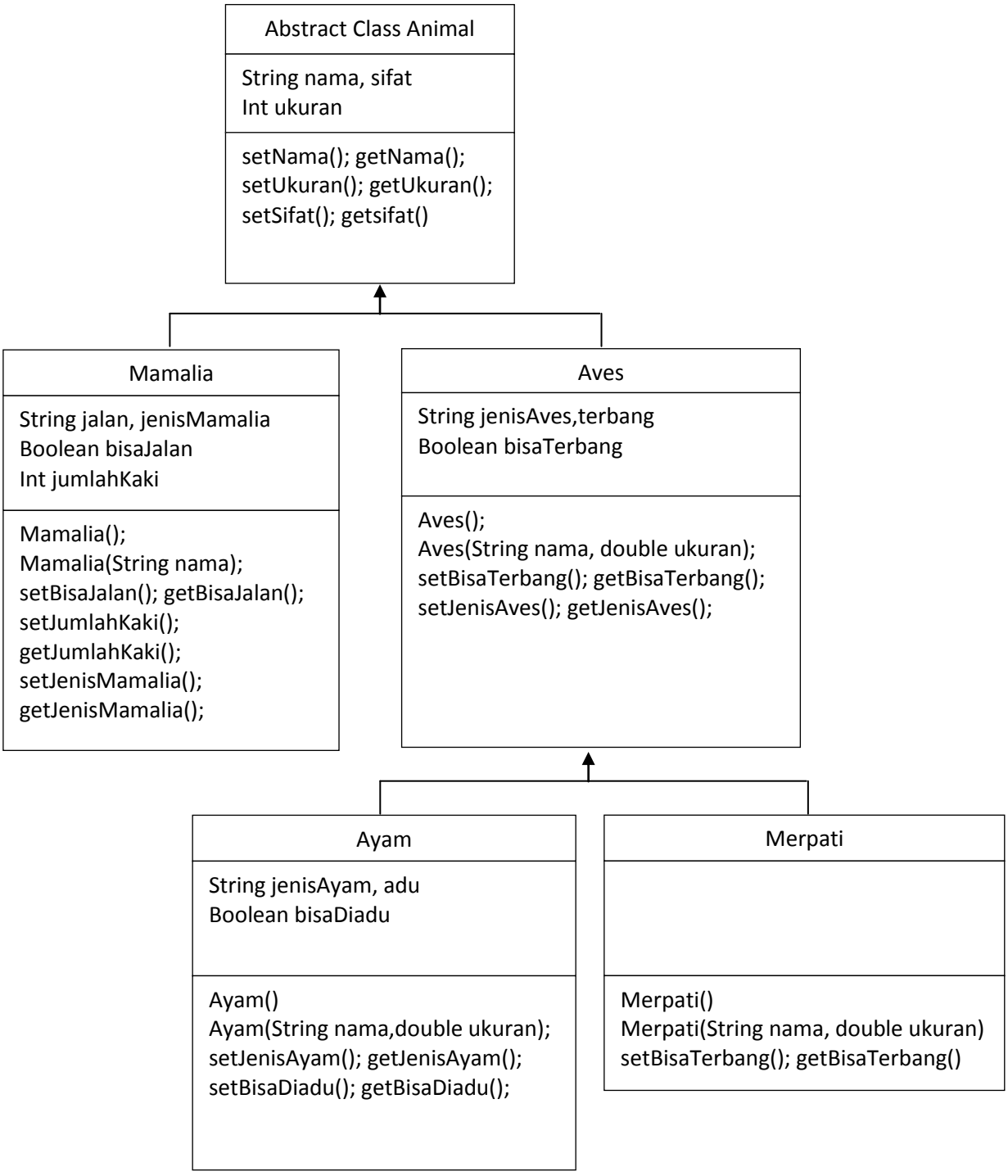
```
abstract class Hewan {
   protected String nama;
   protected int jumKaki;
   protected boolean bisaTerbang = false;
   public Hewan(String nama, int kaki, boolean terbang) {
       this.nama = nama;
       jumKaki = kaki;
       bisaTerbang = terbang;
   }
   public abstract void bersuara();
```

```
   public void lihatHewan() {
       System.out.println("");
       System.out.println("nama  : "+nama);
       System.out.println("jumlah kaki  : "+jumKaki);
       System.out.println("bisa terbang  : "+bisaTerbang);
   }
}

class Sapi extends Hewan {
   public Sapi() {
    super("sapi", 4, false);
   }
   public void bersuara() {
    System.out.println("\nmoooaahhhh,mooooaaahhh");
   }
   public static void main(String[] args) {
       Sapi s = new Sapi();
       s.lihatHewan();
       s.bersuara();
   }
}

class Perkutut extends Hewan {
   public Perkutut(){
    super("perkutut",2,true);
   }
   public void bersuara() {
    System.out.println("\ncuit, cuit, cuit");
   }
   public static void main(String[] args) {
       Perkutut p = new Perkutut();
       p.lihatHewan();
       p.bersuara();
   }
}
```

Ketika class Sapi dijalankan, maka keluarannya:

```
nama  : sapi
jumlah kaki  : 4
bisa terbang  : false

moooaahhhh,mooooaaahhh
```

Contoh UML dengan kelas turunan tingkat 2.



Gambar 4.2 Contoh UML dengan kelas turunan tingkat 2

DAFTAR PUSTAKA

1. Budi Raharjo, Imam Heryanto, Arif Haryono, Mudah Belajar Java, Informatika, Bandung, 2012
2. Tim Dosen Common Laboratory, Modul Praktikum Pemrograman Berorientasi Obyek, Common Laboratory, Fakultas Informatika, ITTelkom, Bandung, 2009.
3. Indra Yatini, Sumiatun, Modul Praktikum Algoritma & Pemrograman, UPT Laboratorium STMIK AKAKOM, Yogyakarta, 2009.