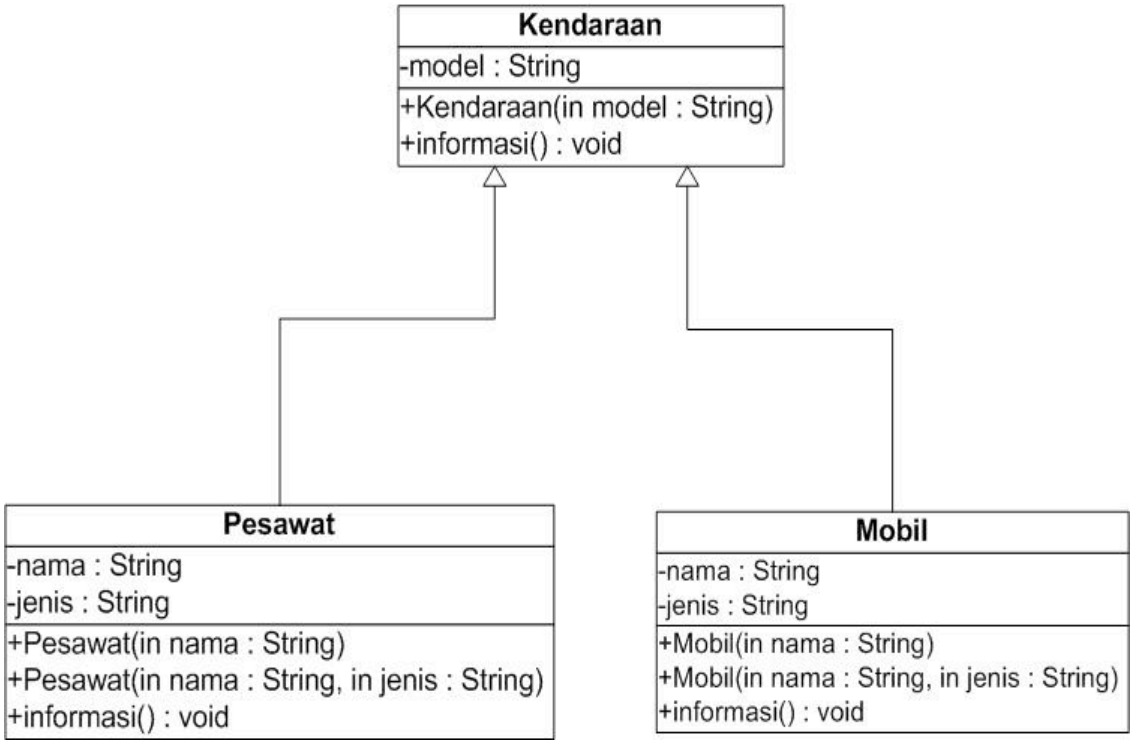
POLIMORFISME

Polimorfisme merupakan suatu cara untuk menyatakan suatu nama yang merujuk pada beberapa fungsi yang berbeda (Sinaga, 2004). Pada polimorfisme rujukan dapat dilakukan pada berbagai tipe objek. Ha; ini dilakukan karena setiap objek dimungkinkan memiloki instruksi yang berbeda. Dalam mengimplementasikan polimorfisme perlu diperhatikan hal-hal sebagai berikut:

1. Method yang dipanggil harus melalui variable dari superclass
2. Method yang dipanggil juga harus merupakan method yang ada pada super calss
3. Signatur method harus sama baik yang ada di super calss maupun yang ada di subclass
4. Method akses atribut pada subclass tidak boleh lebih terbatas dari pada yang ada di super class.

Contoh Implementasi Polimorfisme:




```
1   package polimorfisme;
2
3   public class Kendaraan {
4       private String model;
5
6       public Kendaraan(String model){
7           this.model = model;
8       }
9
10      public void informasi(){
11
12      }
13
14  }
15
```

Kelas Kendaran

```java
package polimorfisme;

public class Pesawat extends Kendaraan{
    private String nama;
    private String jenis;

    public Pesawat(String nama) {
        super("pesawat");
        this.nama = nama;
        this.jenis = "Belum teridentifikasi";
    }

    public Pesawat(String nama, String jenis) {
        super("pesawat");
        this.nama = nama;
        this.jenis = jenis;
    }

    @Override
    public void informasi(){
        System.out.println("Nama : " + nama);
        System.out.println("Jenis : " + jenis);
    }

}
```

Kelas Pesawat

```java
package polimorfisme;

public class Mobil extends Kendaraan {

    private String nama;
    private String jenis;

    public Mobil(String nama) {
        super("Mobil");
        this.nama = nama;
        this.jenis = "Belum teridentifikasi";
    }

    public Mobil(String nama, String jenis) {
        super("Mobil");
        this.nama = nama;
        this.jenis = jenis;
    }

    @Override
    public void informasi() {
        System.out.println("Nama : " + nama);
        System.out.println("Jenis : " + jenis);
    }
}

```

Kelas Mobil

```
1  package polimorfisme;
2
3  public class Polimorfisme {
4
5      public static void main(String[] args) {
6
7          Kendaraan k;
8          Pesawat psw  = new Pesawat("Boing 171");
9          Mobil mbl = new Mobil("Ferrari");
10
11         k = psw;
12         k.informasi();
13         k = mbl;
14         k.informasi();
15
16     }
17 }
18
```

Kelas Main

Dinamic Binding

Pada ilmu komputer (computer science), istilah binding berhubungan dengan suatu identitas atau *identifier,* seperti sebuah fungsi atau nama variabel (variable name), pada bagian kode atau data. Pada skenario yang paling umum. static binding, pemetaan (mapping) ini diketahui pada waktu compile (compile time). Pada dynamic binding, objek dipetakan (mapped) oleh sebuah fungsi yang tidak diketahui pada compile time dan hanya dapat ditentukan selama run time program. Untuk alasan ini. dynamic binding juga disebut dengan *late binding.* Meski dynamic binding menawarkan fleksibilitas yang tidak tersedia pada static binding, dynamic binding juga memerlukan lebih banyak performance cost dibandingkan dengan static binding.

Dynamic binding berkaitan erat dengan *polymorphism,* yang merupakan bagian dan pemrograman berorientasi objek (object oriented programming) Polymorphism memungkinkan nama metode yang sama dapat diimplementasikan pada cara yang berbeda Jika kode tidak ditulis dengan cara di mana metode yang tepat (precise method) tidak dapat ditentukan pada compile time, maka kemudian dynamic binding harus digunakan

Dynamic memungkinkan fleksibilitas penggunaan metode abstrak (abstract mode) tanpa mengetahui implementasi spesifik yang akan digunakan. Keuntungan menggunakan dynamic binding adalah bahwa kode lebih "faersirf (clean) dan lebih dapat diperlahankan dibandingkan dengan alternatif lainnya. dan kode harus diupdate setiap kali suatu tipe input dimasukkan

```
1   package polimorfisme;
2
3   public class Polimorfisme {
4
5       public static void main(String[] args) {
6
7           Kendaraan k;
8           Pesawat psw  = new Pesawat("Boing 171");
9           Mobil mbl = new Mobil("Ferrari");
10
11          k = psw;
12          k.informasi();
13          k = mbl;
14          k.informasi();
15
16      }
17  }
18
```

Dynamic Blinding

Casting

1. **Casting tipe data Primitif**

casting adalah mengubah tipe data.atau bias disebut juga dengan type casting. Tidak sembarang casting. Ada syaratnya, sayaratnya adalah tipe tujuan harus mempunyai panjang atau ukuran lebih besar atau kompatibel.

Tipe asal	Tipe tujuan
char	int, long, float, double
byte	short, char, int, long, float, double
short	int, long, float, double
int	long, float, duoble
long	float, double
float	double

JAVA akan melakukan automatic casting jika kedua tipe kompatibel.

Contohnya:

```
int a = 10;
long b = a;
System.out.println("nilai b = " + b);
//Hasilnya: nilai b = 10
```

Jika kedua tipe tidak kompatibel, perlu dilakukan cara khusus untuk melakukan casting. Misalnya casting dari int (32 bit) ke byte (8 bit).

Caranya:

```
int intKU = 238;
byte byteKU = (byte) intKU;
```

Ada beberapa kendala jika melakukan casting dengan tipe yang tidak kompatibel. Antara lain:

Terjadi pemotongan

Misalnya casting dari float ke int

```
float floatKU = 23.8F;
int intKU = (int) floatKU;
System.out.println("nilaiku " + intKU);
//Hasilnya: nilaiku 23
```

Nilai salah

Ini akan terjadi jika kita melakukan casting dari tipe yang lebih besar ke tipe yang lebih kecil. Jadi tipe yang lebih kecil tidak bias menampung dan menampilkan modulusnya (hasil bagi).

Contoh:

```
int myINT = 300;
 byte myBYTE = (byte) myINT;
 System.out.println("nilaiku " + myBYTE);
//Hasilnya: nilaiku 44
```

```
//karena daya tampung byte hanya sebesar 256
```

2. Casting Objek

Instance dari sebuah kelas dapat diubah ke instance kelas yang lain, dengan syarat kelas-kelas itu harus terhubung dengan mekanisme inheritance (kelas dengan subkelasnya).

Contoh :

Employee merupakan super class dari VicePresident

```
Employee emp = new Employee();
VicePresident pre = new VicePresident();
```

Implisit casting

```
emp = pre;
```

Ekplisit casting

```
pre = (VicePresident) emp;
```

3. Konversi tipe primitif ke objek dan sebaliknya

java.lang yang terdiri dari dari kelas-kelas yang berhubungan dengan untuk setipa tipe data primitifnya yaitu : Float, Boolean, Byte, dan sebagainya. Selain itu terdapat duan nama kelas yang berbeda dengan nama tipe data primitifnya yaitu : Character digunakan utuk variable char, dan Integer untuk variabel int (kedua kelas ini di sebut Wrapper Class).

Contoh :

Konversi sebuah objek ke tipe data primitifnya.

```
Integer dataCount = new Integer(100);
int count = dataCount.intValue();
```

Konversi sebuah objek String ke tipe data numeric

```
String hasil=”4000”;
int hasil = Integer.parseInt(hasil);
```