Pemrograman Berorientasi Object

POLIMORFISME

TEAM TEACHING PBO

Polimorfisme

Definisi Polimorfisme

Static Polymorphism

Dynamic Polymorphism

Polymorphic Arguments

Kata kunci **instanceof**

Definisi

Artinya adalah “dapat memiliki berbagai bentuk”

Polymorphism (polimorfisme) dalam PBO adalah kemampuan untuk mempunyai beberapa bentuk class yang berbeda.

Pada dasarnya, semua class di Java adalah bersifat polimorfik, karena selain bertipe class itu sendiri, dia juga bertipe class **Object**.

- Karena semua class yang ada di Java, termasuk yang kita buat sendiri, adalah **turunan** dari class Object

Sebagai analogi, terdapat class **Binatang** dan class **Kucing**, dimana **Kucing** adalah subclass dari **Binatang**. Maka bisa kita katakan, semua kucing adalah binatang,

Contoh Polimorfisme dalam Dunia Nyata

Manusia dapat memakan semua jenis buah. Tetapi cara memakannya berbeda. Apel dapat dimakan, Pisang juga dapat dimakan (karena Apel dan Pisang termasuk dalam golongan Buah), hanya saja caranya berbeda. Pisang harus dikupas dulu.

Nah dalam hal ini, object Manusia memiliki method MakanBuah dengan parameter object Buah. Parameter ini dapat menerima object yang merupakan sub-class dari Buah (Apel, Pisang, Semangka, dsb). Barulah didalam method MakanBuah dideteksi, apakah Buah-nya itu Apel, atau Pisang, atau yang lain. Kemudian diperlakukan dengan berbeda (Apel langsung dimakan, Pisang dikupas dahulu, dan sebagainya).

Jenis-jenis Polimorfisme

Terdapat dua jenis polimorfisme:

Static Polymorphism

- Yaitu ketika ada method overloading.

Dynamic Polymorphism

- Yaitu ketika ada method overriding.

Static Polimorphism

Static Polymorphism pada Java terjadi jika ada **method overloading**.

- Method overloading: beberapa method dengan nama yang sama tapi beda *signature* dalam satu class.

Pada saat meng-compile program (compile-time), Java tahu method mana yang dipanggil dengan cara mencocokkan *signature* dari method yang dipanggil.

Oleh karena itu static polymorphism disebut juga **compile time polymorphism** atau **static binding**.

Contoh

```java
public class DemoOverload
{
    // method 1
    public int add(int x, int y)
    {
        return x + y;
    }

    // method 2
    public int add(int x, int y, int z)
    {
        return x + y + z;
    }

    // method 3
    public int add(double x, int y)
    {
        return (int)x + y;
    }

    // method 4
    public int add(int x, double y)
    {
        return x + (int)y;
    }
}
```

```java
public class TestOverload
{
    public static void main(String[] args)
    {
        DemoOverload demo = new DemoOverload();

        System.out.println(demo.add(2,3));        //method 1 called

        System.out.println(demo.add(2,3,4));      //method 2 called

        System.out.println(demo.add(2,3.4));      //method 4 called

        System.out.println(demo.add(2.5,3));      //method 3 called
    }
}
```

Contoh

Pada contoh diatas, terdapat empat macam method **add()**.

Compiler Java akan melihat saat method-method tersebut dipanggil dan mencocokkan *signature* nya untuk menentukan method mana yang dieksekusi.

Dynamic Polymorphism

Dynamic Polymorphism terjadi saat ada pembuatan object bertipe parent-class tapi memanggil konstruktor sub-class, dan ada overriding method yang dipanggil.

- Pembuatan object bertipe parent-class dengan memanggil konstruktor sub-class:

```
Employee emp = new Manager();
```

  - Dimana class **Manager** adalah subclass dari **Employee**.
  - Pada contoh diatas, object **emp** bertipe **Employee** tapi memanggil konstruktor **Manager**.
  - Dapat dikatakan, object **emp** bertipe **Employee** tapi memegang referensi **Manager**.

Contoh (1)

```
public class Vehicle
{
    public void move()
    {
        System.out.println("Vehicles can move!!");
    }
}
```

```
public class MotorBike extends Vehicle
{
    public void move()
    {
        System.out.println("MotorBike can move and accelerate too!!");
    }
}
```

Contoh (2)

```
public class TestMotorBike
{
    public static void main(String[] args)
    {
        Vehicle vh = new MotorBike();

        vh.move();      // prints "MotorBike can move and accelerate too!!"

        vh = new Vehicle();

        vh.move();      // prints "Vehicles can move!!"
    }
}
```

Dari percobaan diatas dapat kita lihat bahwa, pertama kita membuat object **vh** yang bertipe **Vehicle**, namun memegang referensi ke class **MotorBike**.

Pada saat pemanggilan method **move()** yang pertama, yang dieksekusi adalah method **move()** milik class **MotorBike**.

Kemudian object **vh** dikembalikan referensinya ke class **Vehicle**

Setelah itu pemanggilan method **move()** yang diseksekusi adalah milik **Vehicle**

Contoh

Hasilnya:

```
D:\Java\Polimorfisme>java TestMotorBike
MotorBike can move and accelerate too!!
Vehicles can move!!
```

Polymorphic Arguments

Adalah parameter method yang menerima suatu nilai yang bertipe sub class.

Polymorphic Arguments

```
public class Pegawai
{
    // some class members
}
```

```
public class Manajer extends Pegawai
{
    // some class members
}
```

```
public class Test
{
    public static void Proses(Pegawai peg)
    {
        // do something
    }

    public static void main(String args[])
    {
        Manajer man = new Manajer();
        Proses(man);
    }
}
```

Class **Manajer** adalah sub class dari **Pegawai**.

Dapat kita lihat pada class **Test,** terdapat method **Proses()** yang menerima parameter bertipe **Pegawai**.

Akan tetapi, pada saat method **Proses** dipanggil, parameternya diberi object **man** yang bertipe **Manajer.**

Pernyataan **instanceof**

Pernyataan **instanceof** sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments.

Pernyataan **instanceof**

```java
public class Pegawai
{
    // some class members
}
```

```java
public class Manajer extends Pegawai
{
    // some class members
}
```

```java
public class Kurir extends Pegawai
{
    // some class members
}
```

```java
public class Test
{
    public static void Proses(Pegawai peg)
    {
        if (peg instanceof Manajer)
        {
            // lakukan tugas Manajer
        }
        else if (peg instanceof Kurir)
        {
            // lakukan tugas Kurir
        }
        else
        {
            // lakukan tugas lain
        }
    }

    public static void main(String args[])
    {
        Manajer man = new Manajer();
        Kurir kur = new Kurir();
        Proses(man);
        Proses(kur);
    }
}
```

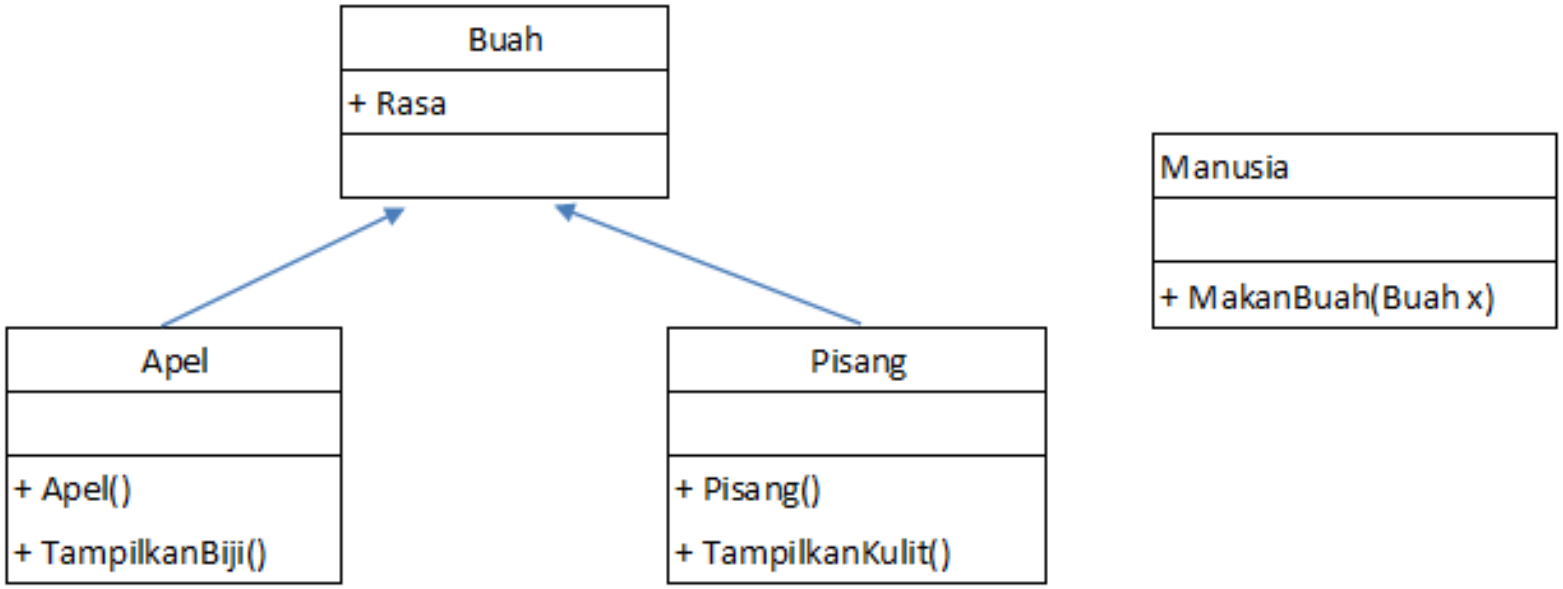
Casting Object

Seringkali penggunaan **instanceof** diikuti dengan casting object dari parameter ke tipe asalnya.

Contoh, kita modifikasi method **Proses** pada class **Test** diatas:

```
if (peg instanceof Manajer)
{
    Manajer man = (Manajer)peg;
    // lakukan tugas manajer
}
else if (peg instanceof Kurir)
{
    Kurir kur = (Kurir)peg;
    // lakukan tugas Kurir
}
else
{
    // lakukan tugas lain
}
```

Contoh Polimorfisme & Casting object



Contoh Polimorfisme & Casting object (lanjutan)

```
public class Buah
{
    public String Rasa;
}
```

```
public class Apel extends Buah
{
    public Apel()
    {
        Rasa = "manis agak asem";
    }

    public void TampilkanBiji()
    {
        System.out.println("Biji apel berukuran kecil");
    }
}
```

```
public class Pisang extends Buah
{
    public Pisang()
    {
        Rasa = "sepet";
    }

    public void TampilkanKulit()
    {
        System.out.println("Kulit pisang berwarna kuning");
    }
}
```

Contoh Polimorfisme & Casting object (lanjutan)

```
public class Manusia
{
    public void MakanBuah(Buah x)
    {
        if(x instanceof Apel)              // jika x = Apel
        {
            System.out.println("Apel langsung dimakan");
            ((Apel)x).TampilkanBiji();
        }
        else if(x instanceof Pisang)       // jika x = Pisang
        {
            System.out.println("Pisang dikupas terlebih dahulu");
            ((Pisang)x).TampilkanKulit();
        }

        System.out.println("Rasanya " + x.Rasa);
    }
}
```

Mengecek tipe object

Melakukan casting object x yg bertipe Buah ke tipe Apel

Contoh Polimorfisme & Casting object (lanjutan)

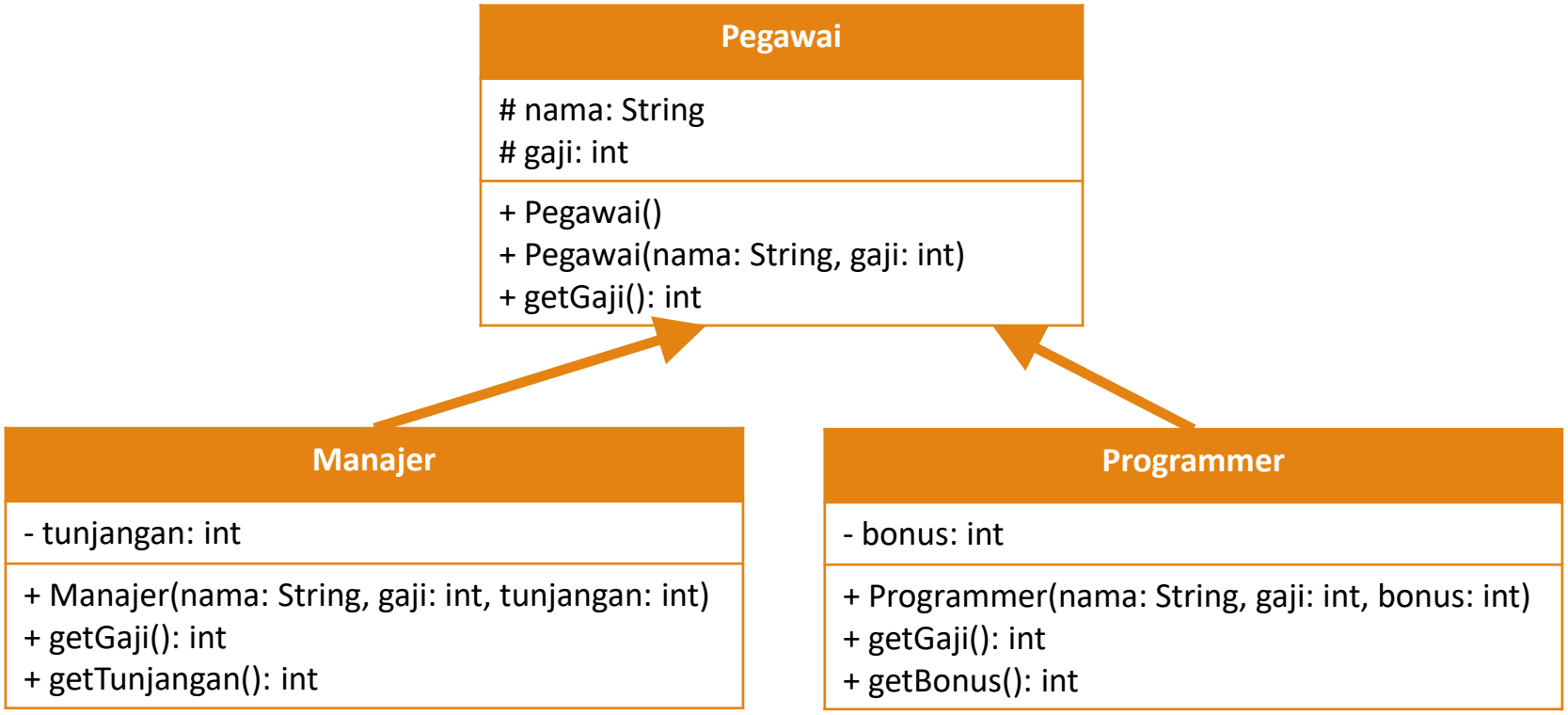
```
public class TestBuah
{
    public static void main(String[] b)
    {
        Manusia ilham = new Manusia();

        Apel myApel = new Apel();
        Pisang myPisang = new Pisang();

        ilham.MakanBuah(myPisang);
    }
}
```

Hasilnya:

```
D:\Java>javac TestBuah.java

D:\Java>java TestBuah
Pisang dikupas terlebih dahulu
Kulit pisang berwarna kuning
Rasanya sepet

D:\Java>
```

Exercise 1

Buatlah program dari class diagram dibawah:



Exercise 1

Kemudian buat class **Bayaran** dan **TestBayaran** dibawah ini untuk pengetesan:

```
public class Bayaran
{
    public int hitungBayaran(Pegawai pg)
    {
        int uang = pg.getGaji();

        if(pg instanceof Manajer)
        {
            uang += ((Manajer)pg).getTunjangan();
        }
        else if(pg instanceof Manajer)
        {
            uang += ((Programmer)pg).getBonus();
        }

        return uang;

    }
}
```

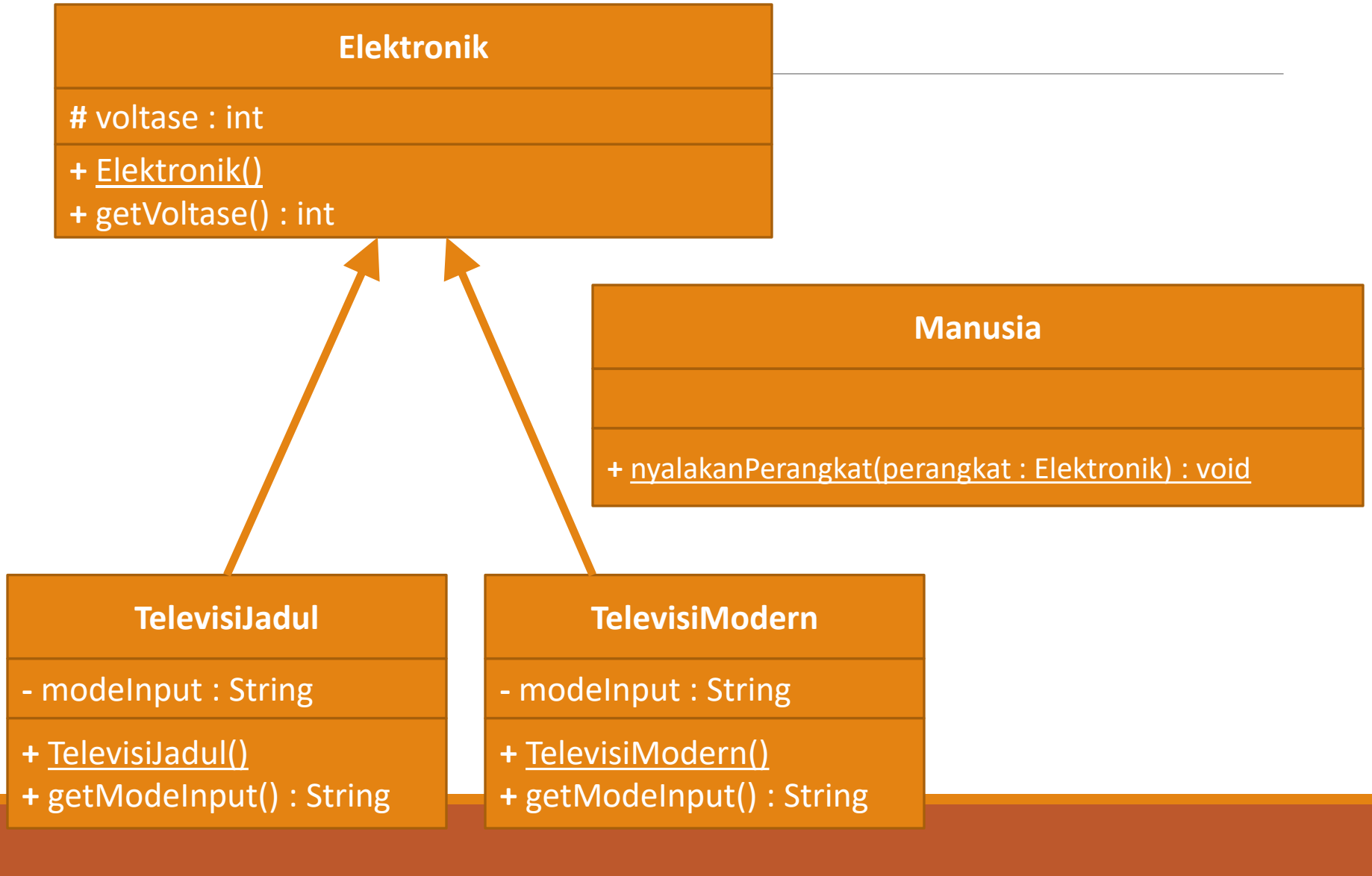
```
public class TestBayaran
{
    public static void main(String[] args)
    {
        Manajer man = new Manajer("Agus", 800, 50)
        Programmer prog = new Programmer("Budi", 600, 30);
        Bayaran hr = new Bayaran();

        System.out.println("Bayaran manajer: " + hr.hitungBayaran(man));
        System.out.println("Bayaran programmer: " + hr.hitungBayaran(prog));
    }
}
```

Exercise 1

Hasil yang diharapkan:

```
Bayaran untuk Manajer : 850
Bayaran untuk Programmer : 630
```

Exercise 2


Elektronik
voltase : int
+ Elektronik()
+ getVoltase() : int
Manusia
+ nyalakanPerangkat(perangkat : Elektronik) : void
TelevisiJadul
- modeInput : String
+ TelevisiJadul()
+ getModeInput() : String
TelevisiModern
- modeInput : String
+ TelevisiModern()
+ getModeInput() : String

Exercise 2

Buat program sesuai dengan class diagram diatas, kemudian buat class **TestElektronik** seperti dibawah:

```
public class TestElektronik
{
    public static void main(String[] args)
    {
        Manusia indro = new Manusia();
        TelevisiJadul tvjadul = new TelevisiJadul();
        TelevisiModern tvmodern = new TelevisiModern();

        indro.nyalakanPerangkat(tvjadul);
        indro.nyalakanPerangkat(tvmodern);
    }
}
```

Hasil yang diharapkan:

```
D:\Java\Polimorfisme>java TestElektronik
Nyalakan televisi jadul dengan input: DVI
Voltase televisi: 220
Nyalakan televisi modern dengan input: HDMI
Voltase televisi: 220
```