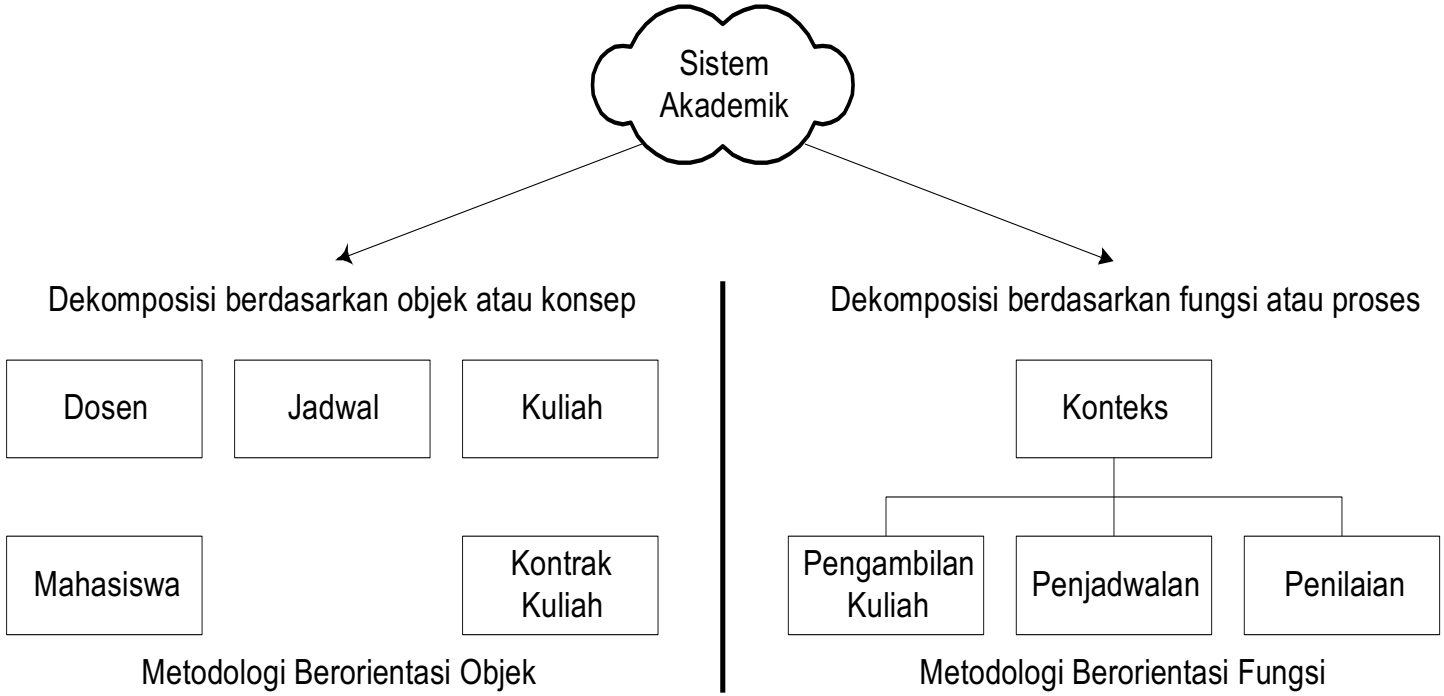
PBO-5
Fitur PBO : Abstraksi, Enkapsulasi, Pewarisan, Polymorphisme

Royana Afwani

2020

Paradigma Pemrograman



PERANGKAT LUNAK OBJEK

- Perangkat lunak yang dibangun dari kelas-kelas dan objek-objek yang saling berinteraksi satu sama lainnya.
  - Kelas
    Deskripsi statis dari sesuatu.
  - Objek
    Sesuatu yang diciptakan (instansiasi) dari kelas.

KELAS DAN OBJEK

- Sebuah kelas menentukan struktur dan tingkah laku sebuah objek
- Kelas adalah cetak biru dari objek, mendefinisikan atribut (state) dan perilaku (behaviour/method) dari objek
- Objek adalah instans dari kelas

KELAS DAN OBJEK

- Contoh

  Kelas Mahasiswa memiliki atribut NIM, Nama, Alamat, dan memiliki perilaku (behaviour/method) seperti melakukan registrasi, perkuliahan, ujian, praktikum, dll.

KELAS DAN OBJEK

- ✓Ani, Budi, dan Gunawan adalah objek (instans) dari kelas mahasiswa.
- ✓Mereka masing-masing memiliki atribut-atribut nama, nim, alamat (yang nilainya masing-masing berbeda).
- ✓Mereka juga memiliki behaviour yang sama yaitu melakukan registrasi, perkuliahan, ujian, praktikum.

FITUR PEMROGRAMAN BERORIENTASI OBJEK

- ABSTRAKSI
- ENKAPSULASI
- PEWARISAN
- POLYMORPHISM

Fitur PBO : Abstraksi

ABSTRAKSI

- Abstraksi adalah suatu proses mendaftarkan entitas-entitas (benda-benda) yang akan berinteraksi dalam suatu aplikasi
- Entitas-entitas tersebut nantinya akan menjadi kelas, dan dari kelas lahirlah objek
- Setelah entitas-entitas didaftarkan, maka selanjutnya ditentukan sifat-sifat (state/atribut) dan hal yang dapat dilakukan oleh entitas tersebut (behaviour/method)

Abstraksi

- Contoh: dalam sistem kepegawaian, dimana didalamnya terdapat banyak entitas di antaranya adalah pegawai.

- Didalam entitas pegawai terdapat atribut-atribut yang akan membedakan satu pegawai dengan pegawai lainnya, misalkan NIP, Nama, Alamat, Tanggal Lahir, dll.

Abstraksi

- Dalam aplikasi yang akan dibuat, entitas-entitas tersebut akan menjadi kelas-kelas. Di dalam kelas-kelas inilah didefinisikan atribut-atribut dan method-method

- Kelas adalah *blue print* dari objek. Dari kelas lahirlah objek
- Kelas tidak memiliki data, tapi objeklah yang menyimpan data

CONTOH KASUS

Contoh masalah yang akan dibuat desain pemecahannya adalah sebagai berikut :

- sebuah koperasi menginginkan sebuah aplikasi pelaporan rekapitulasi stok barang yang mencatat persediaan, pemasukan dan pengeluaran barang. Laporan ini dilakukan seminggu sekali;
- aplikasi ini akan dijalankan oleh seorang petugas khusus logistik. Petugas tersebut akan mencatat Stok barang didapat dari supplier-supplier, baik dari dalam maupun luar kota. Penjualan barang dilakukan pada sebuah mini market, yang pembelinya berasal dari anggota koperasi maupun yang bukan anggota koperasi;
- kemudian ada kebijakan kalau barang berupa makanan yang masa kedaluwarsanya sudah lewat, harus dibuang, dan yang masa kedaluwarsanya tinggal 6 bulan lagi, diberikan diskon 50% (asumsi makananya semuanya mempunyai masa kedaluwarsa > 1 tahun). Kebijakan lainnya adalah : untuk item barang yang tinggal <= 40% dari seharusnya, harus segera ditambah stoknya.

IDENTIFIKASI DOMAIN MASALAH

- Domain masalah didefinisikan sebagai ruang lingkup permasalahan yang akan dipecahkan. Ruang lingkup ini membatasi permasalahan sehingga permasalahan tidak meluas, dan tetap pada fokusnya.
- Domain masalah ini dapat ditentukan dari permintaan / *requirements* yang diberikan oleh pihak yang meminta aplikasi (*customer*). Setelah permintaan tersebut dikumpulkan, maka developer (dalam hal ini programmer) dapat membuat pernyataan tentang apa yang akan dibuatnya , misalnya : *"Membuat sistem pelaporan stok barang pada koperasi yang mengakomodasi kebijakan-kebijakan kedaluwarsa dan potongan harga"* .

IDENTIFIKASI ENTITAS

- Setelah domain masalah diidentifikasi, developer sudah dapat melakukan identifikasi terhadap entitas-entitas (benda-benda) yang akan berinteraksi untuk memecahkan masalah.
- Lakukan identifikasi atas beberapa sifat dari entitas tersebut, antara lain: entitas dapat bersifat konseptual atau berupa benda fisik, entitas memiliki atribut atau karakteristik, entitas memiliki operasi
- Dalam fase ini, kegiatan utamanya adalah mengumpulkan entitas-entitas yang mungkin terlibat sebanyak-banyaknya. Kemudian daftarkan atribut dan operasi (method) yang dimiliki oleh setiap entitas.

Entitas	Atribut	Operasi
Barang	– jumlah – harga beli – tanggal kadaluwarsa – harga jual – ID barang – diskon	– hitung tanggal kedaluwarsa – hitung harga jual – hitung diskon
Supplier	– nama – lokasi – ID supplier – jenis barang	– modifikasi nama supplier – modifikasi lokasi supplier
Koperasi	– daftar transaksi – gudang	– modifikasi daftar transaksi
Pembeli	– ID pembeli – nama pembeli	– .modifikasi ID pembeli – modifikasi nama pembeli
Gudang	– daftar stok barang – koperasi	– modifikasi stok barang – set koperasi
Daftar Transaksi	– data penjualan [ ] – data pembelian [ ]	– tambah data penjualan – tambah data pembelian
Daftar Stok Barang	– Barang [ ] – jumlahBarang[]	– tambah barang – kurangi barang

SELEKSI ENTITAS

Seleksi entitas dilakukan dengan mempertimbangkan hal-hal berikut :

- Relevansi dengan permasalahan

  - Apakah entitas tersebut eksis pada batasan-batasan yang ditentukan pada permasalahan ?

  - Apakah entitas tersebut dibutuhkan untuk menyelesaikan masalah ?

- Eksistensi entitas haruslah *independent*

Hasil Seleksi Entitas

BARANG
jumlah harga beli tanggal kedaluwarsa harga jual ID barang
hitung tanggal kedaluwarsa hitung harga jual hitung diskon

KOPERASI
daftar stok barang daftar transaksi
modifikasi daftar transaksi

DAFTAR TRANSAKSI
data penjualan [ ] data pembelian [ ]
tambah data penjualan tambah data pembelian

DAFTAR STOK BARANG
barang [] jumlah barang []
tambah barang kurangi barang

Kesimpulan : Abstraksi

1. Menentukan domain sistem/aplikasi
2. Mendaftar entitas-entitas dalam sistem/aplikasi → Kelas
3. Mendata atribut dan behaviournya
4. Menseleksi entitas (kelas) yang pasti akan digunakan saja.

Enkapsulasi

Enkapsulasi

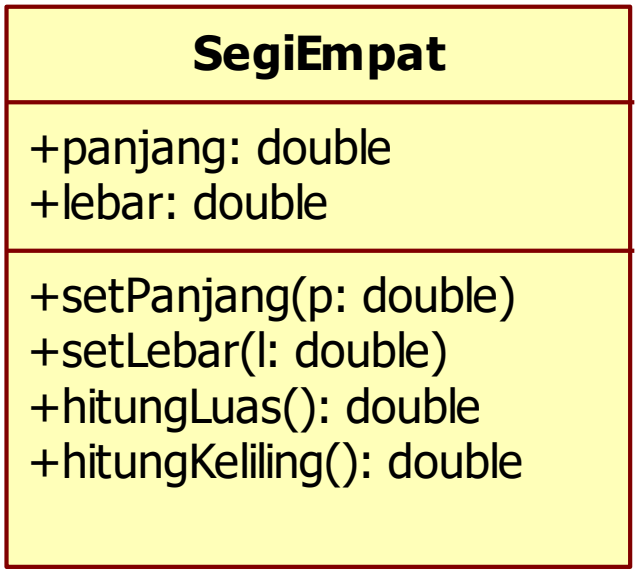
- Enkapsulasi adalah suatu cara untuk menyembunyikan implementasi detil (berupa atribut atau method) dari suatu kelas.
- Dalam OOP, kebutuhan akan enkapsulasi muncul karena adanya sharing data antar method
- Dalam pemrograman, prinsip enkapsulasi dinyatakan dengan identifier private, default, protected, public

Struktur Dasar Kelas

- Struktur dasar sebuah kelas pada intinya ada 4 bagian, yaitu :
  - deklarasi kelas;
  - deklarasi dan inisialisasi atribut;
  - pendefinisian *method*;
  - komentar.

PEMBUATAN KELAS

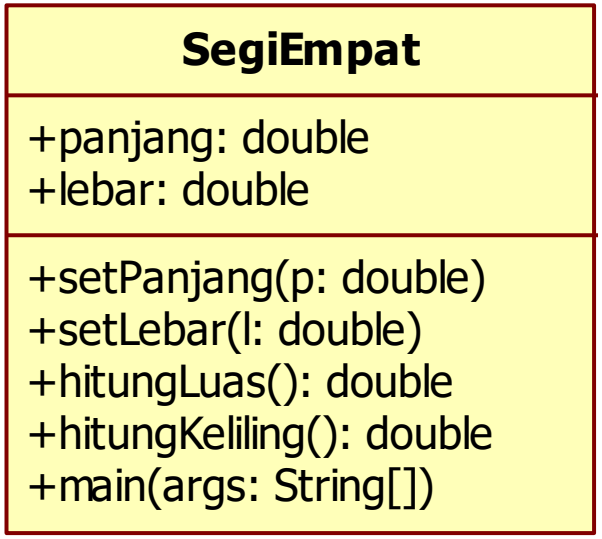
- Diagram UML (Unified Modeling Language)



Kode Kelas SegiEmpat

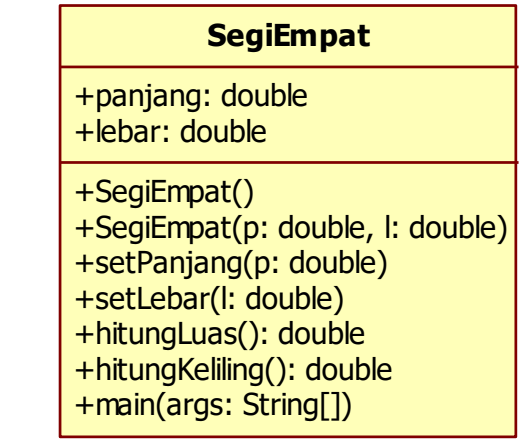
```
public class SegiEmpat{
   //deklarasi variabel (atribut)
   public double panjang;
   public double lebar;

   //deklarasi method
   public void setPanjang(double p){
          panjang = p;
   }
   public void setLebar(double l){
          lebar = l;
   }
   public double hitungLuas(){
          return panjang*lebar;
   }
   public double hitungKeliling(){
          return 2*(panjang + lebar);
   }
}
```

KELAS UTAMA (MAIN)




```
public class SegiEmpat{
    public double panjang;
    public double lebar;
    public void setPanjang(double p){
           panjang = p;
    }
    public void setLebar(double l){
           lebar = l;
    }
    public double hitungLuas(){
           return panjang*lebar;
    }
    public double hitungKeliling(){
           return 2*(panjang + lebar);
    }
    public static void main(String[] args){
           SegiEmpat se1;
           se1 = new SegiEmpat(); //pembentukan objek
           SegiEmpat se2 = new SegiEmpat(); // pembentukan objek
           se1.setPanjang(10);
           se1.setLebar(5);
           se2.setPanjang(5.5);
           se2.setLebar(2.3);
           System.out.println("Luas segi empat 1 ="+se1.hitungLuas());
           System.out.println("Keliling segi
           empat 2 ="+se2.hitungKeliling());
    }
}
```

KELAS DENGAN KONSTRUKTOR



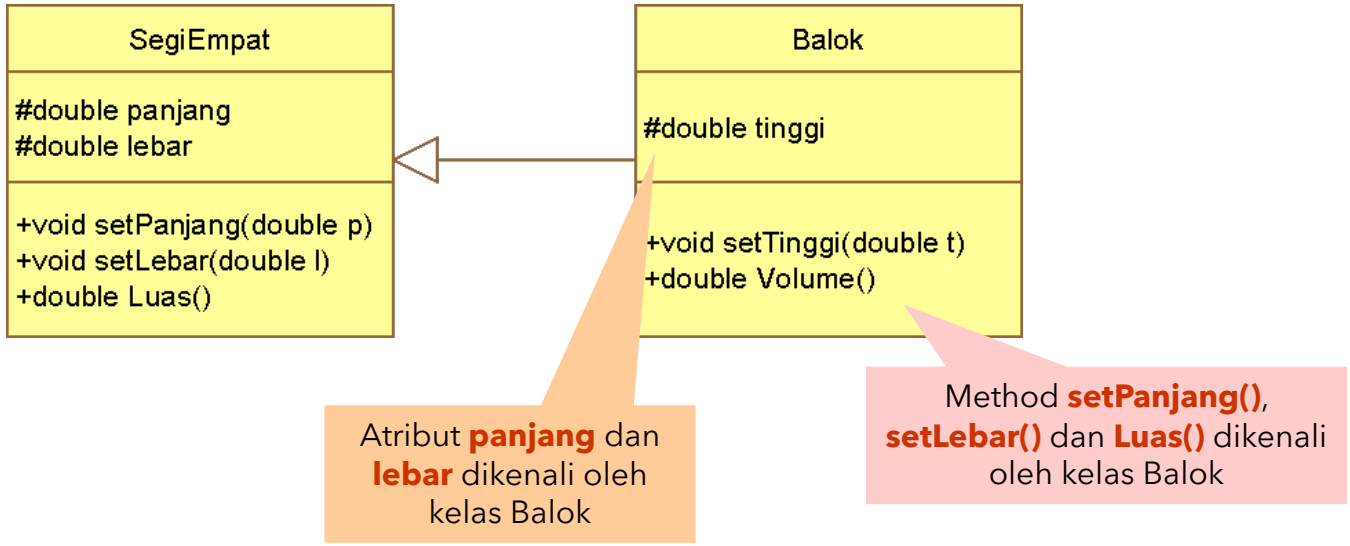

```
public class SegiEmpat{
    public double panjang;
    public double lebar;

    //deklarasi konstruktor
       public SegiEmpat(){
       }
       public SegiEmpat(double p, double l){
        panjang = p;
        lebar = l;
     }
     //deklarasi method
    public void setPanjang(double p){
           panjang = p;
    }
    public void setLebar(double l){
           lebar = l;
    }
    public double hitungLuas(){
           return panjang*lebar;
    }
    public double hitungKeliling(){
           return 2*(panjang + lebar);
    }
```

```
public static void main(String[] args){
           SegiEmpat se1;
           se1 = new SegiEmpat(); //pembentukan
    objek
           se1.setPanjang(10);
           se1.setLebar(5);
           SegiEmpat se2 = new
    SegiEmpat(5.5,2.3); // pembentukan objek
           System.out.println("Luas segi empat 1
    ="+se1.hitungLuas());
           System.out.println("Keliling segi
    empat 2 ="+se2.hitungKeliling());

    }
}
```

Pewarisan

Apa yang Disebut Pewarisan ?

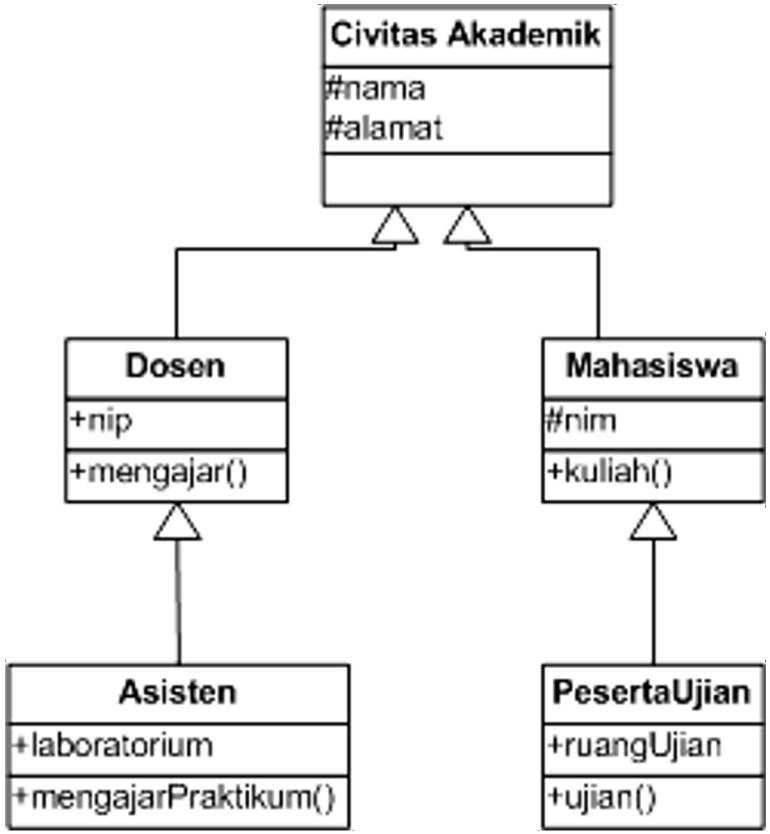
- Kemampuan sebuah kelas untuk **mewariskan** seluruh atau sebagian atribut dan methodnya ke kelas lain, sehingga atribut dan method tersebut **dikenal** oleh kelas yang menerima pewarisan **tanpa harus menuliskannya**.



Apa yang Disebut Pewarisan ? (lanjutan)

- Kelas yang mewariskan disebut **kelas induk**, ***super class***, atau ***base class***.
- Kelas yang menerima pewarisan disebut **kelas anak**, **kelas turunan**, atau ***subclass***.
- Merupakan implementasi dari relasi antar kelas ***generalization-specialization*** atau "**is-a**".

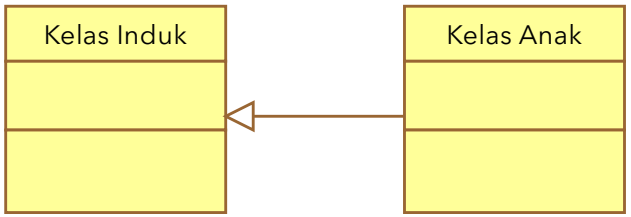
CONTOH PEWARISAN



Bentuk-bentuk Pewarisan

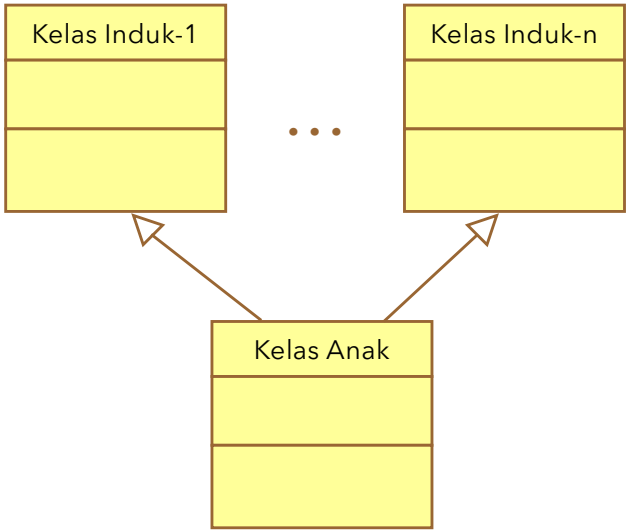
- Pewarisan Tunggal (*Single Inheritance*)

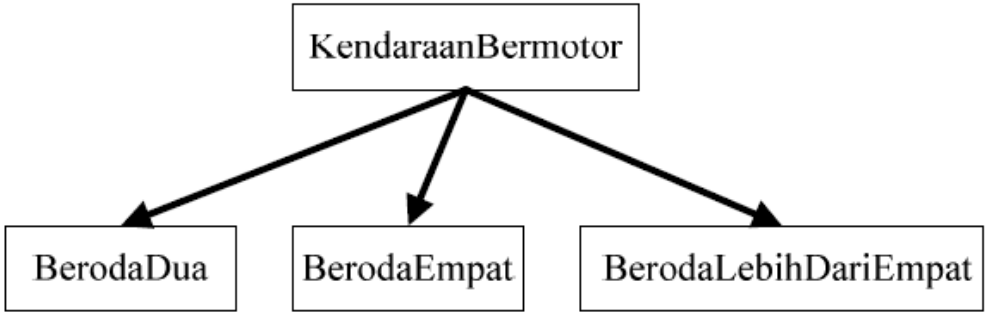
  Kelas anak menerima pewarisan dari **satu** kelas induk.



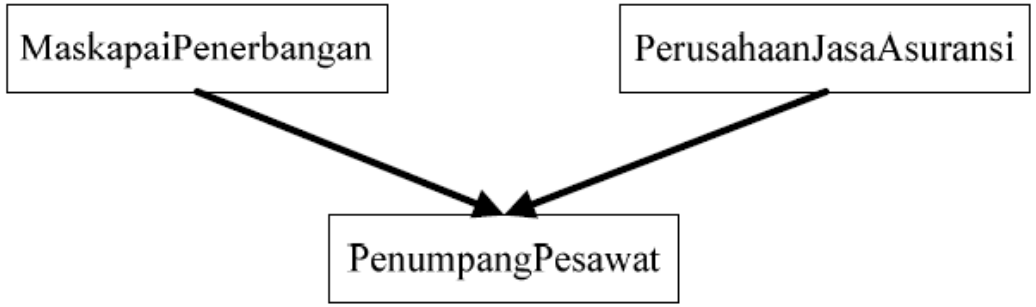
- Pewarisan Majemuk (*Multiple Inheritance*)

  Kelas anak menerima pewarisan dari **beberapa** kelas induk. Bahasa Java tidak mengakomodasi *multiple inheritance*





Gambar 1-7 *Single Inheritance*



Gambar 1-8 *Multiple Inheritance*

Pewarisan dalam Java

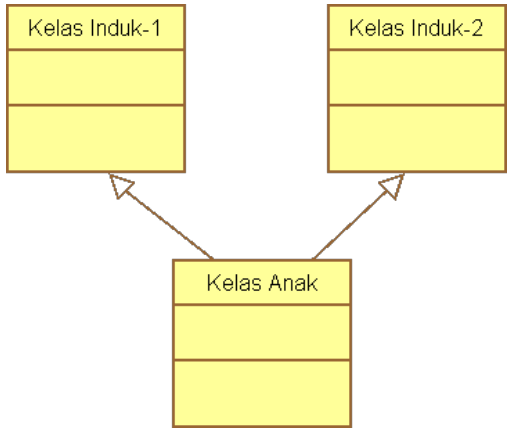
- Dinyatakan dengan menggunakan kata kunci (*keyword*) **extends**:

```
class KelasAnak extends KelasInduk {
  // Atribut kelas anak
  // Method kelas anak
}
```

- Semua atribut dan method dengan *access modifier* **non-private** akan **diwariskan** dari kelas induk ke kelas anak.
- Method yang diwariskan dapat didefinisikan ulang di kelas anak (*override*) → polymorphism.

Pewarisan dalam Java (lanjutan)

- Hanya memperbolehkan pewarisan tunggal.
- Pewarisan majemuk harus dinyatakan sebagai pewarisan tunggal yang ditulis secara berjenjang.

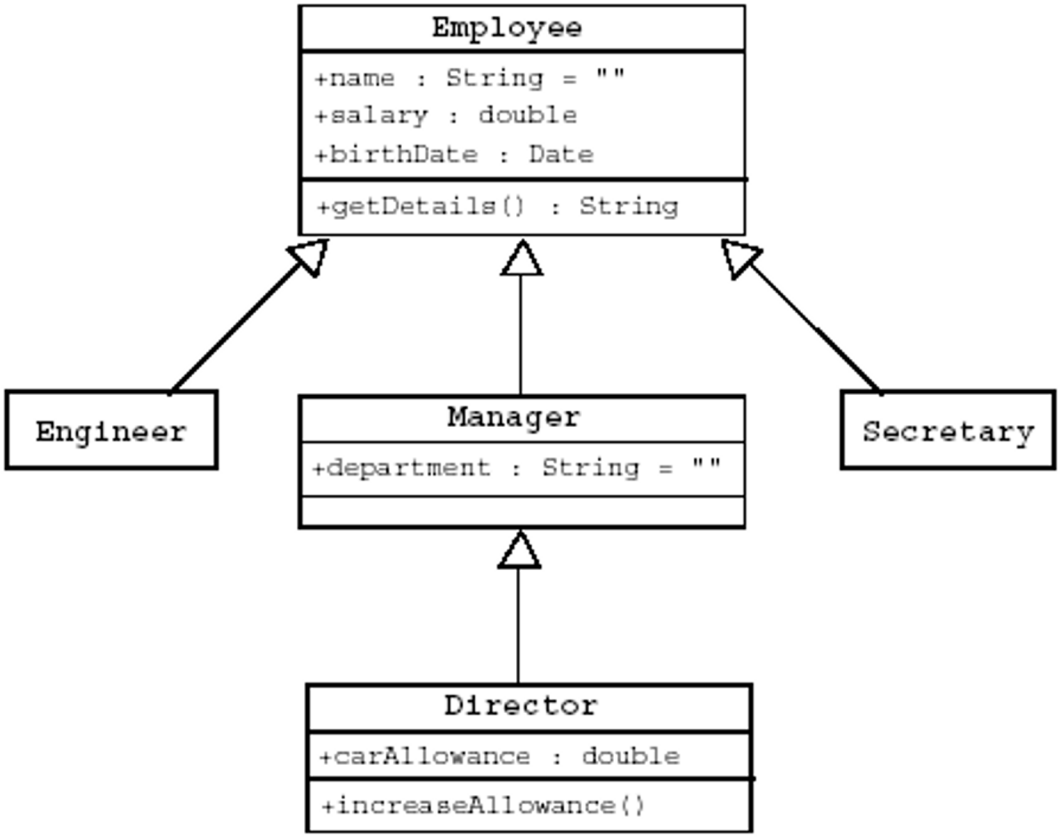


Pewarisan majemuk asal

Pewarisan majemuk setelah "transformasi"

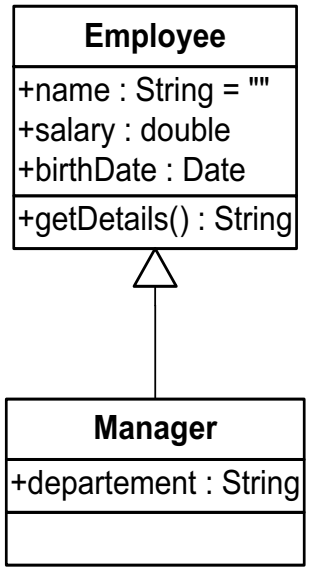
Pewarisan dalam Java (lanjutan)

- Single dan multilevel inheritance



Contoh Program Pewarisan

- Contoh 1




```
public class Employee {
  public String name;
  public Date birthDate;
  public double salary;

  public String getDetails() {...}
}

public class Manager extends
Employee {
  public String department;
}
```

Sepeda.java

```
public class Sepeda{
    public int gir;

        void setGir(int pertambahanGir) {
                gir= gir+ pertambahanGir;
        }

        int getGir() {
                return gir;
        }
}
```

Class SepedaGunung Mewarisi Class Sepeda

```
public class SepedaGunung extends
   Sepeda{

   private int sadel;

   void setSadel (int jumlah) {
            sadel = getGir() - jumlah;
   }

   int getSadel(){
            return sadel;
   }
}
```

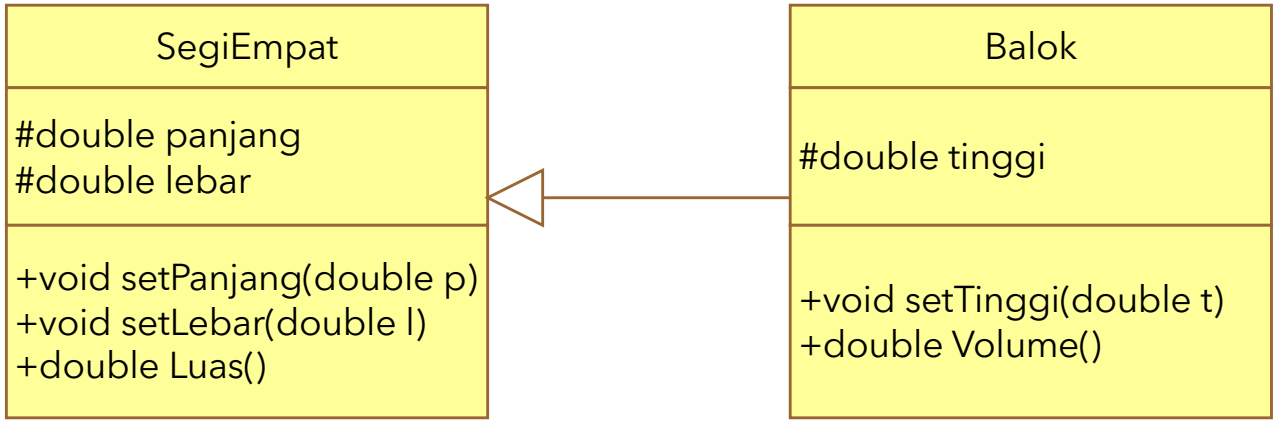
SepedaGunung.java

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

       SepedaGunung sg=new SepedaGunung();

        sg.setGir(3);
        System.out.println(sg.getGir());

        sg.setSadel(1);
        System.out.println(sg.getSadel());
    }
}
```

SepedaGunungBeraksi.java

Contoh Program Pewarisan

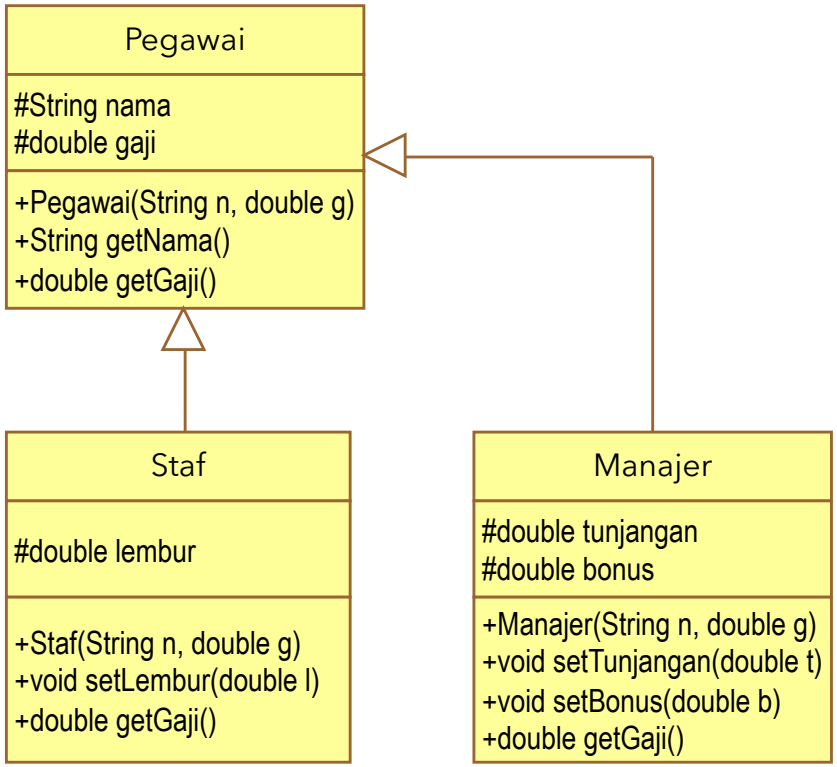
- Contoh 2



- SegiEmpat.java
- Balok.java
- Driver.java

Contoh Program Pewarisan (lanjutan)

- Contoh 3



- Pegawai.java
- Staf.java
- Manajer.java
- Driver.java

Kapan Pewarisan ?

- Ada beberapa atribut dan method yang sama yang digunakan oleh beberapa kelas berbeda (reduksi penulisan kode).
- Ada satu atau beberapa kelas yang sudah pernah dibuat yang dibutuhkan oleh aplikasi (*reusability*).
- Ada perubahan kebutuhan fungsional atau *feature* aplikasi dimana sebagian atau seluruh perubahan tersebut tercakup di satu atau beberapa kelas yang sudah ada (*extend*).

Keyword `super`

- `super` digunakan untuk merefer superclass dari suatu class, yaitu untuk merefer member dari superclass,baik data attributes maupun methods

Pewarisan dalam Java (lanjutan)

- Untuk memanggil konstruktor kelas induk dari (konstruktor) kelas anak digunakan notasi **super()**, dan harus ditulis di baris pertama pada konstruktor kelas anak.

```
public Balok() {
  super();
  tinggi = 0;
}
```

- Untuk memanggil method kelas induk dari kelas anak digunakan notasi **super.namaMethod()**.

```
public double getGaji() {
  double pokok = super.getGaji();
  return (pokok+lembur);
}
```

Keyword super

```
1.class Employee {
2.  public String name;

3.  public Employee (String s) {
4.       name = s
5.  }
6.}
```

```
1.class Manager extends Employee {
2.  private String alamat;
3.  public Manager(String nama, String s) {
4.       super.name = nama ;
5.       alamat = s;
6.  }
7.}
```

Kontrol pengaksesan

- Dalam dunia riil, suatu entitas induk bisa saja tidak mewariskan sebagian dari apa-apa yang ia punyai kepada entitas turunan karena sesuatu hal.
- Demikian juga dengan konsep inheritance dalam OOP.
- Suatu parent class dapat tidak mewariskan sebagian member-nya kepada subclass-nya.
- Sebagai contoh, kita coba untuk memodifikasi class Pegawai.
- Hal ini dipengaruhi oleh **access modifier**.

Hak Akses

Modifier	Kelas yang sama	Kelas Turunan	Object dlm Package yg sama	Object yg berbeda package
Private	V	X	X	X
Protected	V	V	V	X
Public	V	V	V	V

Contoh

```
public class Pegawai {
        private String nama;
        public double gaji;
}
```

Contoh

- Coba untuk mengkompilasi class Manajer pada contoh sebelumnya.
- Apa yang terjadi?
- Pesan kesalahan akan muncul seperti ini :

```
Manajer.java:5: nama has private access in Pegawai
                nama=n;
```

- Ini membuktikan bahwa class Manajer tidak mewarisi data member nama dari parent class-nya (Pegawai).

protected

- Protected mempunyai kemampuan akses yang lebih besar daripada private dan default.
- Protected feature dari suatu class bisa diakses oleh semua class dalam satu package.

Example: protected

```
1. package adifferentpackage; // Class Ski now in
// a different package
2. Public class Ski {
3.     protected void applyWax() { . . . }
4. }
```

```
1. Public package sportinggoods;
2. class DownhillSki extends Ski {
3.     void tuneup() {
4.             applyWax(); → OK
5.             // other tuneup functionality here
6.     }
7. }
```

Konstruktor tidak diwariskan

- Konstruktor dari parent class tidak dapat diwariskan ke subclass-nya.
- Konsekuensinya, setiap kali kita membuat suatu subclass, maka kita harus memanggil konstruktor parent class.
- Pemanggilan konstruktor parent harus dilakukan pada baris pertama dari konstruktor subclass.

Konstruktor tidak diwariskan

- Jika kita tidak mendeklarasikannya secara eksplisit, maka kompiler Java akan menambahkan deklarasi pemanggilan konstruktor parent class di konstruktor subclass.

Konstruktor tidak diwariskan

- Sebelum subclass menjalankan konstruktornya sendiri, subclass akan menjalankan constructor superclass terlebih dahulu.
- Hal ini terjadi karena secara implisit pada constructor subclass ditambahkan pemanggilan `super()` yang bertujuan memanggil constructor superclass oleh kompiler.

Misalnya saja kita mempunyai dua buah class sebagai berikut :

```
public class Parent
{

}
```

```
public class Child extends Parent {

}
```

Contoh

- Pada saat program tersebut dikompilasi, maka kompiler Java akan menambahkan :
  - konstruktor class Parent
  - konstruktor class Child
  - pemanggilan konstruktor class Parent di kostruktor class Child

Sehingga program tersebut sama saja dengan yang berikut ini :

```
public class Parent {
        public Parent() {

        }
}
```

```
public class Child extends Parent {
        public Child() {
                super();
        }
}
```

pemanggilan kostruktor class Parent

Contoh

```
public class Child extends Parent {
        int x;
        public Child() {
                x = 5;
                super();
        }
}
```

X

```
public class Child extends Parent {
        int x;
        public Child() {
                super();
                x = 5;
        }
}
```

√

Constructor pada inheritance

- **Misalkan kita buat class parent bernama Person sbb :**

```
public class Person {
      protected String name;
      protected String address;

      public Person(){
            System.out.println("Inside Person:Constructor");

      }

}
```

- **Sekarang, kita buat class lain bernama Student yang meng-extends class Person.**

```
public class Student extends Person {
       public Student(){
              System.out.println("Inside Student:Constructor");
       }
}
```

Alur Eksekusi Constructor

- Ketika sebuah object Student diinstansiasi, *default constructor* dari superclass Student dipanggil secara implisit untuk melakukan inisialisasi seperlunya.
- Setelah itu, pernyataan di dalam constructor subclass baru dijalankan.

Penjelasan

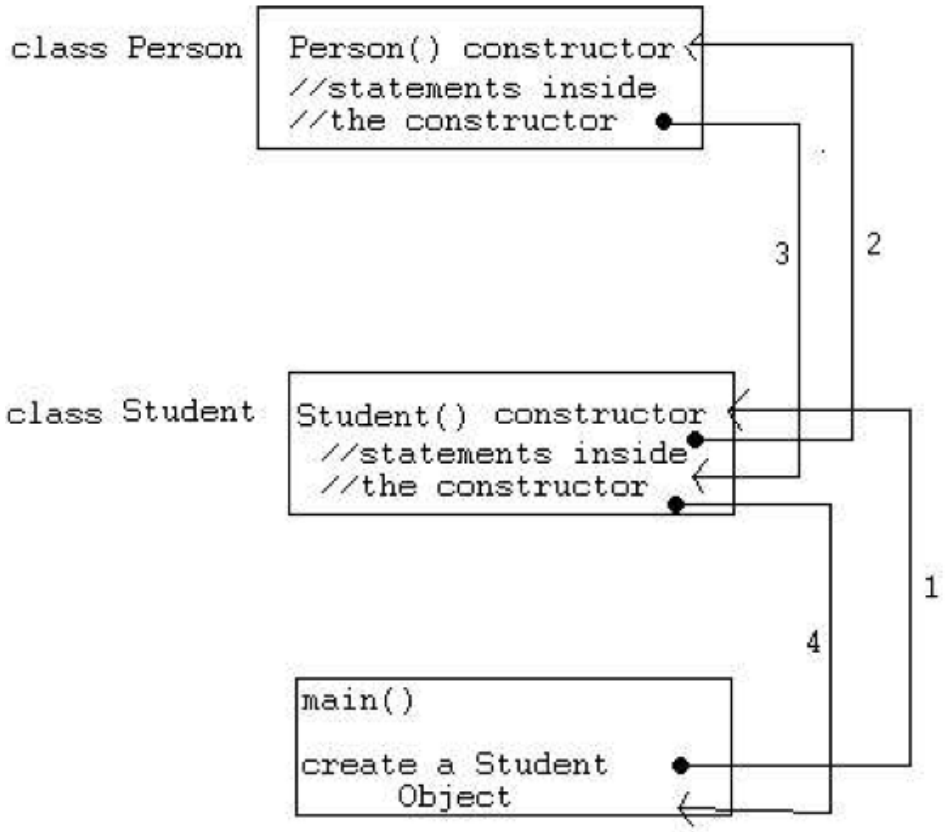
- Untuk memperjelasnya, perhatikan kode dibawah ini,

```
public static void main( String[] args ){
     Student anna = new Student();
}
```

- Dalam kode ini, kita membuat sebuah object dari class Student. Hasil dari program adalah:

```
Inside Person:Constructor
Inside Student:Constructor
```

Diagram Alur Eksekusi Constructor

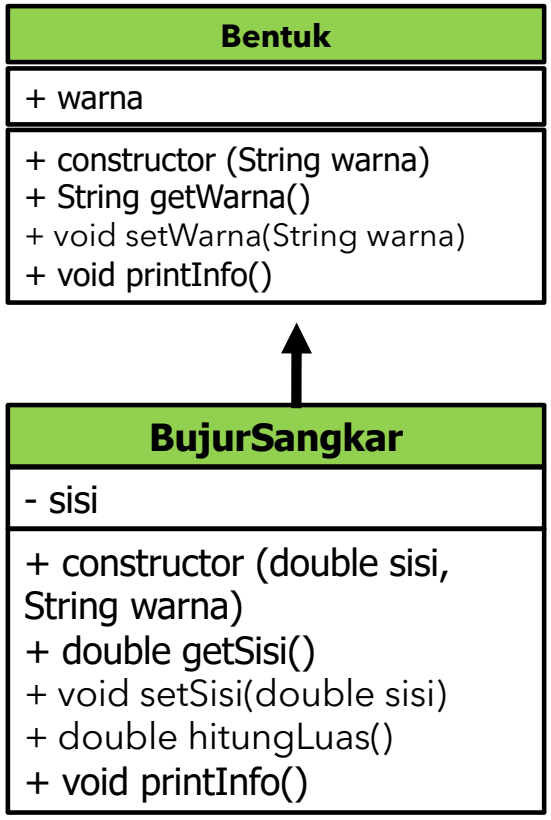


Latihan: Inheritance Matematika

1. Buat class MatematikaCanggih yang merupakan inherit dari class Matematika
    1. Tambahkan method modulus(int a, int b) yang menghitung modulus dari a dan b
    2. Operator modulus adalah %

2. Buat class MatematikaCanggihBeraksi yang memanggil method pertambahan, perkalian dan modulus

Latihan 1



Untuk kelas Bentuk :

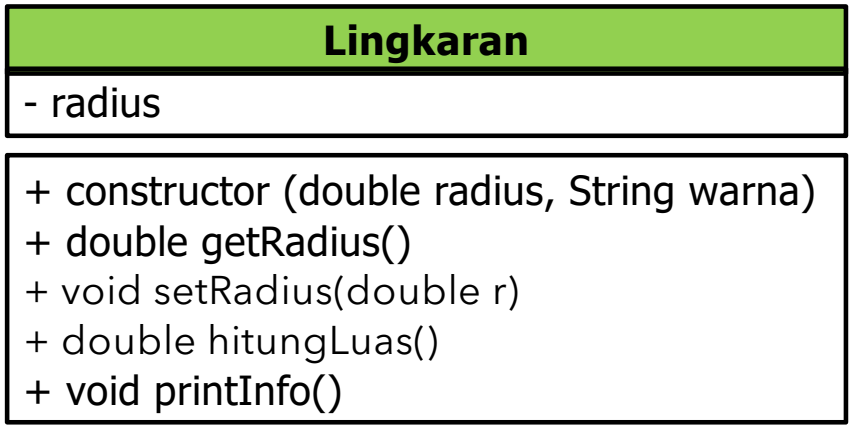
- **getWarna** adl method yang akan mengembalikan nilai variabel warna
- **setWarna** adl method untuk mengubah nilai variabel warna
- **printInfo** adl method yang akan menuliskan "Bentuk berwarna [warna]"

Untuk kelas BujurSangkar :

- **getSisi** adl method yang akan mengembalikan nilai variabel sisi
- **setSisi** adl method untuk mengubah nilai variabel sisi
- **hitungLuas** adl method yang akan mengembalikan hasil perhitungan luas bujursangkar
- **printInfo** adl method yang akan menuliskan "Bujursangkar berwarna [warna], luas = [luas]"

Latihan 2

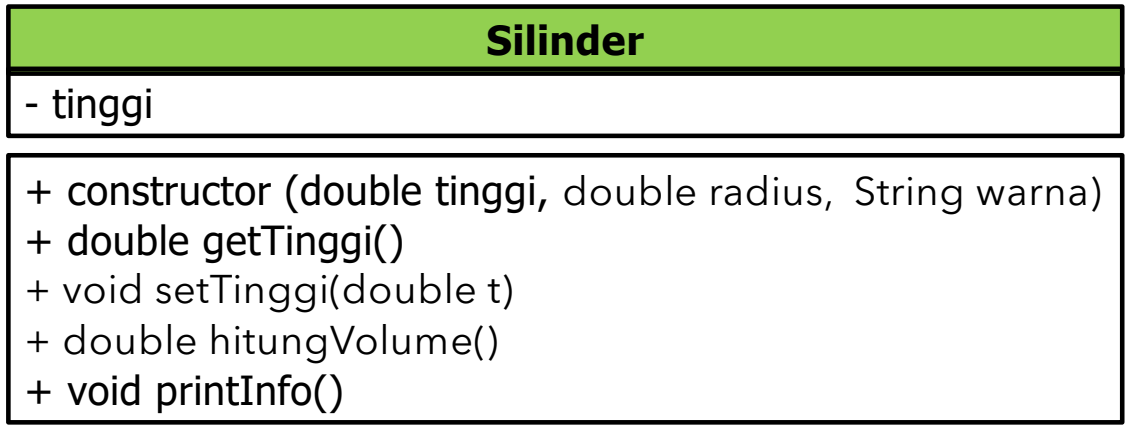
- Buatlah kelas Lingkaran sbg turunan kelas Bentuk.



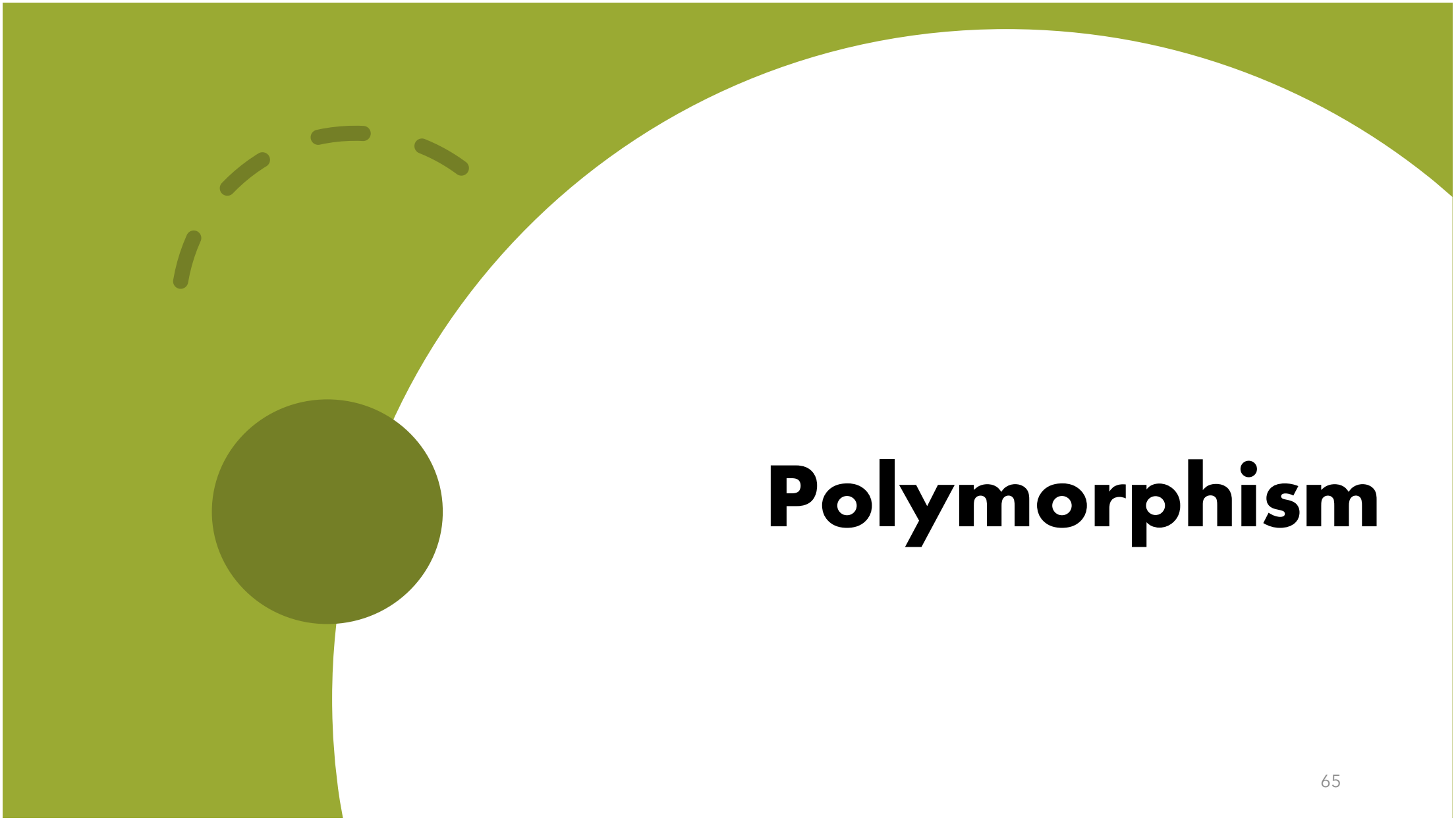
- **constructor** akan menginisialisasi radius dan warna
- **getRadius** adl method yang akan mengembalikan nilai variabel radius
- **setRadius** adl method untuk mengubah nilai variabel radius
- **hitungLuas** adl method yang akan mengembalikan hasil perhitungan luas lingkaran (PHI jadikan sbg konstanta kelas)
- **printInfo** adl method yang akan menuliskan “Lingkaran [warna], luas = [luas]”

Latihan 3

- Buatlah kelas Silinder sbg turunan kelas Lingkaran



- **constructor** akan menginisialisasi variabel tinggi, radius, dan warna
- **getTinggi** adl method yg akan mengembalikan tinggi
- **setTinggi** adl method yg akan mengubah tinggi
- **hitungVolume** adl method yg akan mengembalikan hasil perhitungan volum silinder
- **printInfo** adl method yg akan menuliskan "Silinder warna [warna], volume = [volume]"


Polymorphism

Polymorphism

- *Polymorphism* memungkinkan suatu method memiliki bentuk yang berbeda
- *Polymorphism* ada 2 jenis :
  - Overloading
  - Override

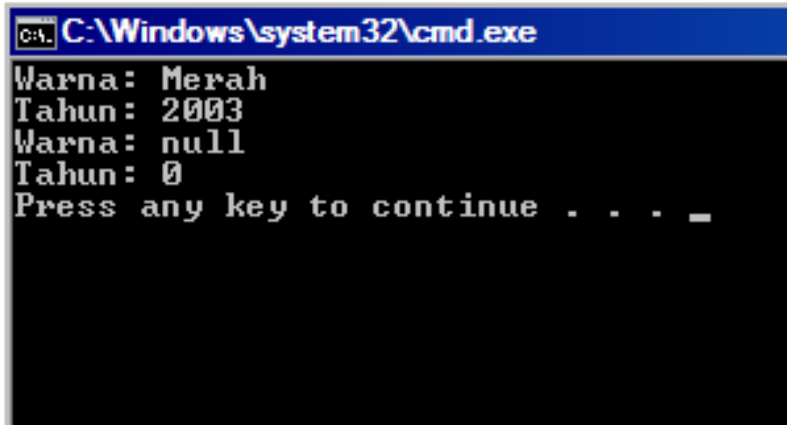
Polymorphism

- Kemampuan untuk memperlakukan object yang memiliki perilaku (bentuk) yang berbeda
- Implementasi konsep polymorphism:
  1. Overloading: Kemampuan untuk menggunakan nama yang sama untuk beberapa method yang berbeda parameter (tipe dan atau jumlah)
  2. Overriding: Kemampuan subclass untuk menimpa method dari superclass, yaitu dengan cara menggunakan nama dan parameter yang sama pada method

Polymorphism – Overloading

```
class Mobil {
    String warna;
    int tahunProduksi;

    public Mobil(String warna, int
    tahunProduksi){
        this.warna = warna;
        this.tahunProduksi = tahunProduksi;
    }

    public Mobil(){
    }

    void info(){
        System.out.println("Warna: " + warna);
        System.out.println("Tahun: " + tahunProduksi);
    }
}
```

```
public class MobilKonstruktor{
  public static void main(String[] args){
      Mobil mobilku = new Mobil("Merah", 2003);
      mobilku.info();

      Mobil mobilmu = new Mobil();
      mobilmu.info();
      }
}
```



Polymorphism – Overloading

```
class Lingkaran{
   void gambarLingkaran(){
   }
    void gambarLingkaran(int diameter){
   ...
   }
    void gambarLingkaran(double diameter){
   ...
   }
    void gambarLingkaran(int diameter, int x, int y){
   ...
   }
    void gambarLingkaran(int diameter, int x, int y, int warna, String
   namaLingkaran){
   ...
   }
}
```

Polymorphism - Overriding

```
public class Sepeda{
    private int gir;

      void setGir(int pertambahanGir) {
            gir= gir+ pertambahanGir;
      }

      int getGir() {
            return gir;
      }
}
```

Polymorphism - Overriding

```
public class SepedaGunung extends
   Sepeda{

   void setGir(int pertambahanGir) {
         super.setGir(pertambahanGir);
          gir = 2*getGir();
   }
}
```

SepedaGunung.java

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

         SepedaGunung sg=new
            SepedaGunung();

         sg.setGir(2);
          System.out.println(sg.getGir());

          sg.setGir(3);
          System.out.println(sg.getGir());
      }
}
```

SepedaGunungBeraksi.java

Latihan: Overloading pada Matematika

1. Kembangkan class Matematika, MatematikaCanggih dan MatematikaBeraksi
2. Lakukan overloading pada Method yang ada (pertambahan, pengurangan, perkalian, pembagian, modulus)
3. Tambahkan method baru bertipe data double (pecahan) dan memiliki 3 parameter
4. Uji di kelas MatematikaBeraksi dengan parameter pecahan: 12.5, 28.7, 14.2
5. Uji konsep overloading dengan:
   pertambahan(12.5, 28.7, 14.2)
   pertambahan(12, 28, 14)
   pertambahan(23, 34)
   pertambahan(3.4, 4.9)

Overloading Constructor

```
1. public class Employee {
2.           private static final double BASE_SALARY = 15000.00;
3.           private String name;
4.           private double salary;
5.           private Date birthdate;

6.           public Employee (String name, double salary, Date DoB) {
7.                       this.name = name;
8.                       this.salary = salary;
9.                       birthdate = DoB;
10.          }
11.          public Employee(String name, double salary) {
12.                      this(name,salary,null);
13.          }
14.          public Employee(String name, Date DoB)   {
15.                      this(name,BASE_SALARY,DoB);
16.          }
17.          public Employee(String name)            {
18.                      this(name,BASE_SALARY);
19.          }
20.          //more Employee code...
21. }
```

Overloading Method

```
1. public class Employee {
2.           private static final double BASE_SALARY = 15000.00;
3.           private String name;
4.           private double salary;
5.           private Date birthdate;

6.           public Employee (String name, double salary, Date DoB) {
7.                       this.name = name;
8.                       this.salary = salary;
9.                       birthdate = DoB;
10.          }
11.          public void set_Employee(String name, double salary) {
12.                      this(name,salary,null);
13.          }
14.          public void set_Employee(String name, Date DoB)       {
15.                      this(name,BASE_SALARY,DoB);
16.          }
17.          public void set_Employee(String name)    {
18.                      this(name,BASE_SALARY);
19.          }
20.          //more Employee code...
21. }
```

Overriding Methods

- Sub class bisa mendefinisikan ulang suatu method yang sudah diwariskan dari parent class
- Sub class bisa mendefinisikan suatu method yang secara fungsionalitas berbeda dengan method yang diwariskan dari parent class, dengan syarat memiliki kesamaan :
  - Nama Method
  - Return Type
  - Argument list

Overriding Methods

```
1. class Employee {
2.   private String name;
3.   private MyDate birthDate;
4.   private float  salary;

5.   public String getDetails() {
6.     return "Name: " + name + "\nSalary: " + salary
7.            + "\nBirth Date: " + birthDate;
8.   }
9. }
```

```
1. class Manager extends Employee {
2.   protected String department;

3.   public String getDetail() {
4.     return super.getDetails() + "\nDepartment: " +
5.     department;
6.   }
7. }
```

Penerapan Pewarisan

```java
public class Kendaraan {
    private String nama_pemilik;
    protected String nomor_mesin;
    public int harga;

    public Kendaraan(){
        nama_pemilik = "Negara";
        nomor_mesin = "0";
        harga = 0;
        System.out.println("Ctor Kendaraan");
    }

    public Kendaraan(String pemilik, String mesin, int price){
        nama_pemilik = pemilik;
        nomor_mesin = mesin;
        harga = price;
        System.out.println("Copy Ctor Kendaraan");
    }

    public void deskripsi(){
        System.out.println("Kendaraan ini milik " + nama_pemilik);
        System.out.println("Nomor Mesin kendaraan ini " + nomor_mesin);
        System.out.println("Harga kendaraan ini = " + harga);
    }

    public void identitas(){
        System.out.println("Ini kendaraan");
    }
}
```

```java
public class Motor extends Kendaraan{
    public Motor(){
        System.out.println("Ctor Motor");
        System.out.println("Harga Motor = " + harga);// bisa
        System.out.println("Nomor Mesin Motor = " + nomor_mesin);// bisa
        //System.out.println("Pemilik Motor = " + nama_pemilik);// TIDAK bisa
    }
}
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Motor m1 = new Motor();
    m1.deskripsi();
}
```

```
Output - Konsep_PBO (run)
run:
Ctor Kendaraan
Ctor Motor
Harga Motor = 0
Nomor Mesin Motor = 0
Kendaraan ini milik Negara
Nomor Mesin kendaraan ini 0
Harga kendaraan ini = 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penerapan Pewarisan

```
11  public class Motor extends Kendaraan{
12      public Motor(){
13          System.out.println("Ctor Motor");
14          System.out.println("Harga Motor = " + harga);// bisa
15          System.out.println("Nomor Mesin Motor = " + nomor_mesin);// bisa
16          //System.out.println("Pemilik Motor = " + nama_pemilik);// TIDAK bisa
17      }
18  }
```

```
17      public static void main(String[] args) {
18          // TODO code application logic here
19          Motor m1 = new Motor();
20          m1.deskripsi();
21      }
```

```
Output - Konsep_PBO (run)
run:
Ctor Kendaraan
Ctor Motor
Harga Motor = 0
Nomor Mesin Motor = 0
Kendaraan ini milik Negara
Nomor Mesin kendaraan ini 0
Harga kendaraan ini = 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penerapan Polymorphism

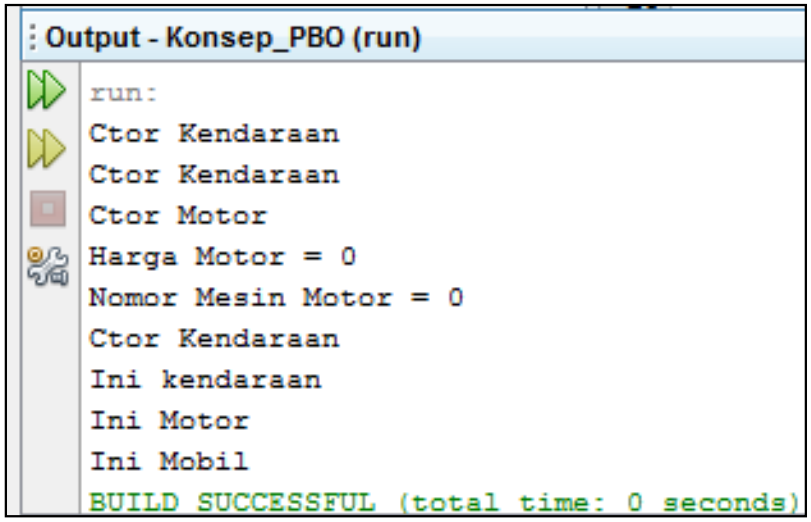
```java
public class Kendaraan {
    private String nama_pemilik;
    protected String nomor_mesin;
    public int harga;

    public Kendaraan(){...}

    public Kendaraan(String pemilik, String mesin, int price){...}

    public void deskripsi(){...}

    public void identitas(){
        System.out.println("Ini kendaraan");
    }
}
```

```java
public class Motor extends Kendaraan{
    public Motor(){...}

    @Override
    public void identitas(){
        System.out.println("Ini Motor");
    }
}
```

```java
public class Mobil extends Kendaraan {

    @Override
    public void identitas(){
        System.out.println("Ini Mobil");
    }
}
```

Penerapan Polymorphism

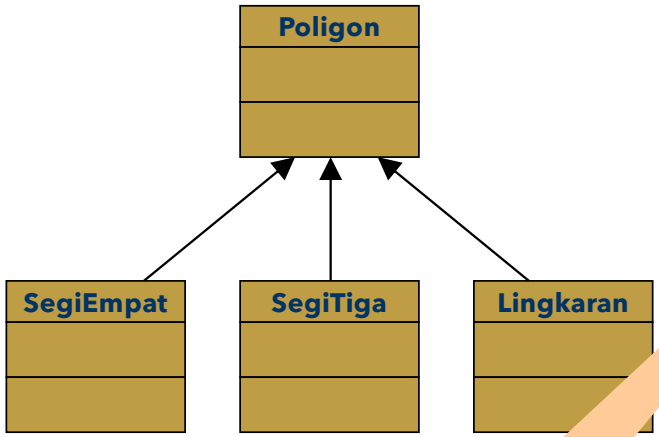
```
17      public static void main(String[] args) {
18          // TODO code application logic here
19          Kendaraan k = new Kendaraan();
20          Kendaraan mtr = new Motor();
21          Kendaraan mbl = new Mobil();
22
23          k.identitas();
24          mtr.identitas();
25          mbl.identitas();
26      }
```



Apa yang Disebut Polimorfisme ? (lanjutan)

- Kemampuan suatu objek untuk digunakan di banyak tujuan berbeda dengan nama yang sama.
- Kemampuan objek dalam memberikan respon yang berbeda terhadap message yang mempunyai nama yang sama.
- Disebut juga:
  - *dynamic binding*
  - *late binding*
  - *run-time binding*

Gambaran Polimorfisme




```
// Program Utama
class TesPoligon {
  Poligon p = new Poligon();
  ...

  SegiEmpat s = new SegiEmpat();
  SegiTiga  t = new SegiTiga();
  Lingkaran l = new Lingkaran();

  p = s; // ok krn instan kelas
  anak juga instan kelas induk
  ...
}
```

variabel **p** menunjuk **objek** yang diciptakan dari kelas **Poligon**

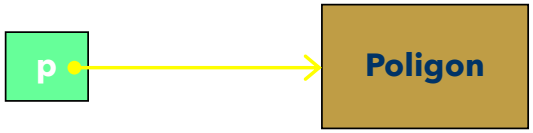
variabel **p** diubah sehingga menunjuk **objek** dari kelas **SegiEmpat**

variabel **s** menunjuk **objek** yang diciptakan dari kelas **SegiEmpat**

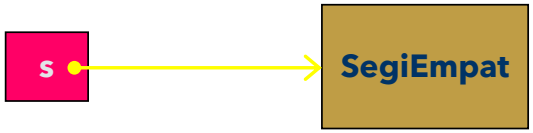
Gambaran Polimorfisme (lanjutan)

Sebelum:

Poligon p = new Poligon();

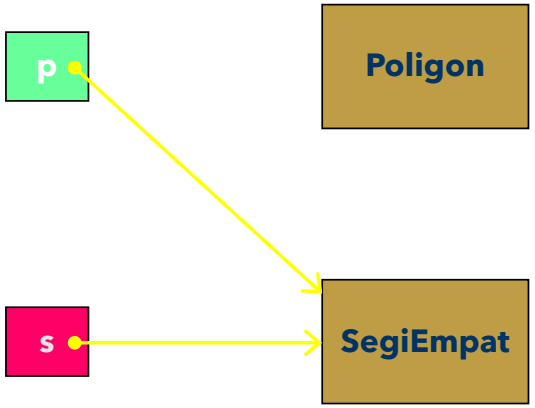


SegiEmpat s = new SegiEmpat();



Sesudah:

p = s;



Gambaran Polimorfisme (lanjutan)

```
class SegiEmpat {
  . . .
  public String display() {
    return ("Segi Empat"
     + "\no Panjang  : " + getPanjang()
     + "\no Lebar    : " + getLebar()
     + "\no Luas     : " + Luas()
     + "\no Keliling : " + Keliling());
  }
}
```

```
class Lingkaran {
  . . .
  public String display() {
    return ("Lingkaran"
     + "\no jari-jari: " + getRadius()
     + "\no Luas     : " + Luas()
     + "\no Keliling : " + Keliling());
  }
}
```

```
class TesPoligon {
  public static void main(String args[])
  {
    // Deklarasi array
    Poligon[] p = new Poligon[2];

    // Add array poligon
    p[0] = new SegiEmpat(17, 8);
    p[1] = new Lingkaran(10);

    // Display informasi
    for (int i=0; i<p.length; i++) {
      System.out.println(p[i].display());
    }
  }
}
```

Method **display ()** yang dieksekusi ditentukan berdasarkan rujukan ke masing-masing objeknya

Contoh Polimorfisme

- Array yang berisi bangun geometri berbeda-beda

Kelas induk:

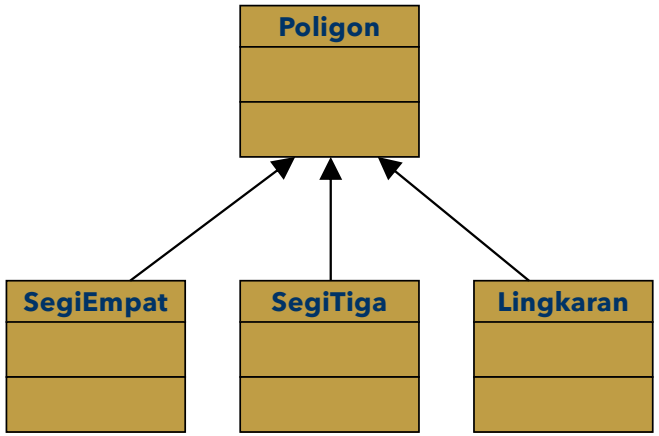
- Poligon.Java

Kelas anak:

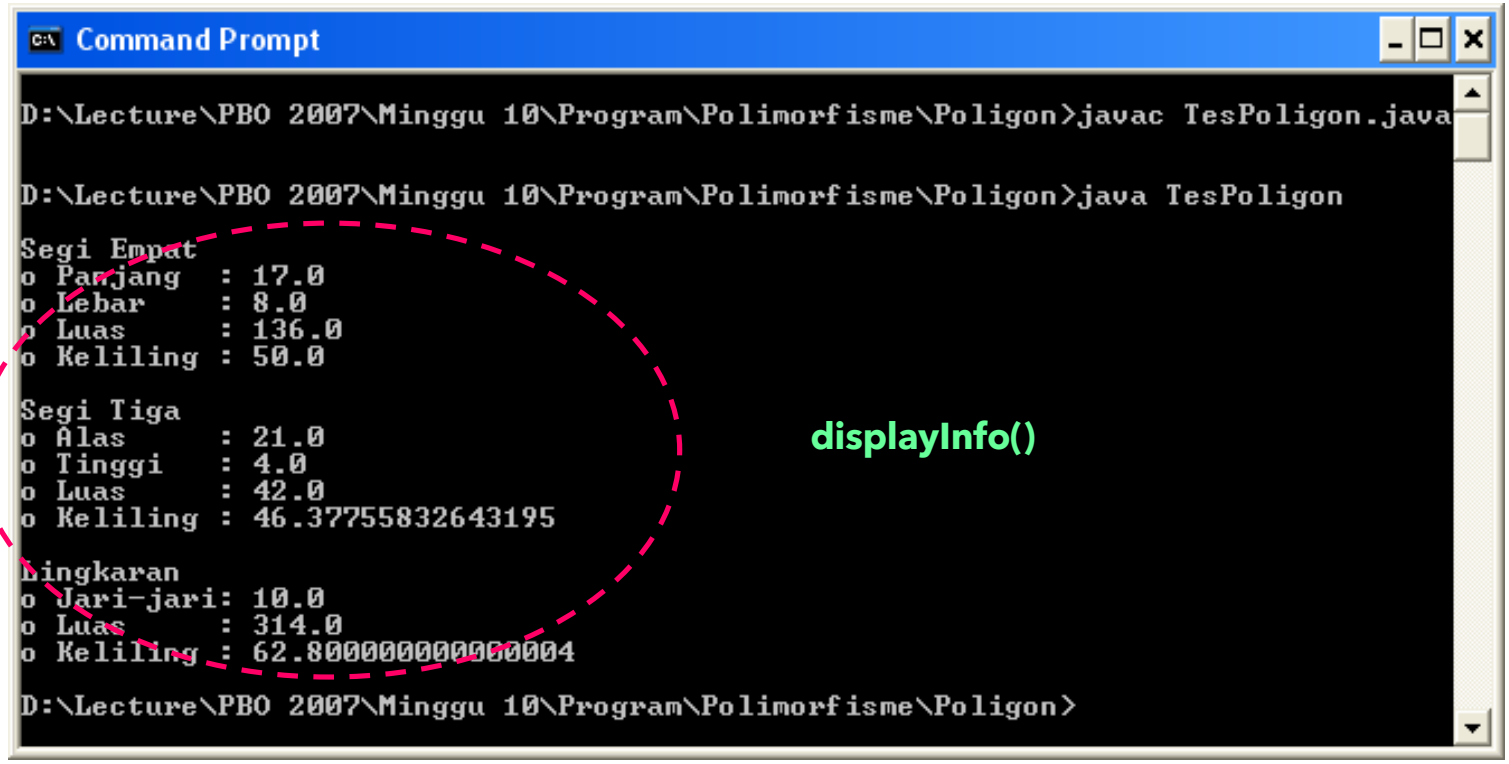
- SegiEmpat.java
- SegiTiga.java
- Lingkaran.java

Program utama:

- TesPoligon.java



Contoh Polimorfisme (lanjutan)



Tugas PBO

- Bentuk Kelompok Maksimal terdiri atas 2 orang.
- Buat sebuah program OOP sederhana untuk suatu permasalahan (permasalahan ditentukan oleh kelompok) ➔ Bahasa Java
- Setiap kelompok membuat abstraksi keterkaitan dan deskripsi kelas yang dibutuhkan.
- Permasalahan harus memuat kondisi pewarisan dan polymorphisme

Tugas PBO

- Buat slide presentasi untuk tugas tersebut
- Content:
    1) Permasalahan
    2) Abstraksi kebutuhan dan keterhubungan kelas (terdapat inheritance)
    3) Deskripsi atribut dan method yang dibutuhkan (beserta alasan penggunaan identifier → enkapsulasi)
    4) Contoh Penggunaan Polymorphisme
    5) Source code
    6) Screen shoot output program

TERIMA KASIH