Kodehive

Java Fundamental | Bootcamp

OOP II - Inheritance

Subjects : OOP II

1. Constructor
2. Inheritance
3. Extends
4. IsA & Has-A
5. Interface (Abstract Class)
6. Abstract Method
7. Implements

Constructor - Method

```
public class KelasAdaConstrutor {

    public KelasAdaConstrutor() {
        // buat method1 tanpa void
        // dan nama methodnya sama persis dgn class-nya
        System.out.println("Pertama kali dijalankan jika instance!");
        System.out.println("Nama Method sama dengan nama Class");
    }

    public void methodKipas() {
        // method2 dgn nama methodKipas
        System.out.println("Kipas angin berputar");
    }

}
```

Constructor - Method

```
public class KelasPanggilConstructor {

    Run | Debug
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        KelasAdaConstrutor kac = new KelasAdaConstrutor();
        kac.methodKipas();

        // meski kita hanya memanggil methodKipas
        // tp method KelasAdaConstrutor akan tetap dijalankan
        // dijalankan yg pertama kali
        // nah method KelasAdaConstrutor itu disebut dgn Construtor
        // Constructor --> method tanpa void dan namanya sama
        // namanya sama sperti nama classnya

    }
}
```

Pertama kali dijalankan jika instance!
Nama Method sama dengan nama Class
Kipas angin berputar

Kodehive academy

Constructor - Variable

```
public class KelasConstructorVariable {
    String namaDepan;
    int usia;

    public KelasConstructorVariable() {
        namaDepan = "WILLIAM";
        usia = 24;
    }

}
```

Constructor - Variable

- Jelaskan mengapa outputnya seperti itu?

```
public class KelasPanggilConstructorVariable {

    Run | Debug
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        KelasConstructorVariable kcv = new KelasConstructorVariable();

        System.out.println(kcv.namaDepan);

        System.out.println(kcv.usia);

    }

}
```

```
WILLIAM
24
```

Kodehive academy

Encapsulation

```java
public class Encapsulation {

    private String nama;
    private String alamat;
    private int usia;

    public String getNama() {
        return nama;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
    public String getAlamat() {
        return alamat;
    }
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
    public int getUsia() {
        return usia;
    }
    public void setUsia(int usia) {
        this.usia = usia;
    }
}
```

Encapsulation merupakan salah satu konsep OOP yang cukup penting.

Encapsulation – Subclass Instance

```java
public class SubClassEncapsulation {
    Run | Debug
    public static void main(String[] args) {
        Encapsulation encap = new Encapsulation();

        // create data
        encap.setNama("Ryo");
        encap.setAlamat("Jakarta Selatan");
        encap.setUsia(26);

        // ambil data
        System.out.println("-- Biodata Karyawan --");
        System.out.println("Nama    : "+encap.getNama());
        System.out.println("Alamat  : "+encap.getAlamat());
        System.out.println("usia    : "+encap.getUsia());
    }
}
```

Setelah kita buat encapsulation kemudian pada subclass ini kita lakukan instance terlebih dahulu, selanjutnya kita gunakan method public dari class encapsulation untuk mengisi data dan mengambil data.

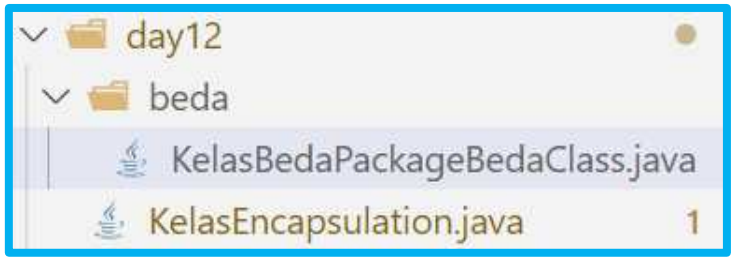
```
-- Biodata Karyawan --
Nama    : Ryo
Alamat  : Jakarta Selatan
usia    : 26
```

Method Instance

```
public class KelasEncapsulation {

    public void metodePublic() {
        System.out.println("METODE PUBLIC");
    }

    protected void metodeProtected() {
        System.out.println("METODE PROTECTED");
    }

    private void metodePrivate() {
        System.out.println("METODE PRIVATE");
    }

}
```

Method Instance

- Lalu kita buat sebuah kelas berbeda dengan beda package juga.
- Setelah kita instance kelas Encapsulation sebelumnya, ternyata hanya dapat memanggil method Public.
- Sedangkan kita butuh method Protected dan Private juga.
- Lalu bagaimana mengatasinya?



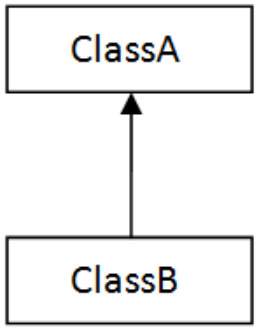

```
import day12.KelasEncapsulation;

public class KelasBedaPackageBedaClass {

    Run | Debug
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        KelasEncapsulation ke = new KelasEncapsulation();
        ke.metodePublic();
        // Hanya metodePublic saja yang dapat dipanggil
    }

}
```

METODE PUBLIC

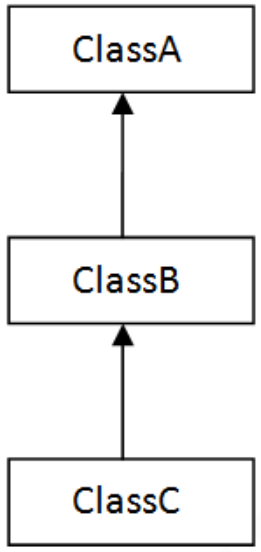
Inherintance

- Inheritance atau pewarisan adalah jawaban dari permasalahan tersebut.
- Kelas yang mewarisi kelas lain, dapat menggunakan method dan variable dari kelas tsb, terkecuali method atau variable yang bersifat private.
- Ada 2 sintaks di dalam inherintance yaitu
  1. Extends
  2. Implements

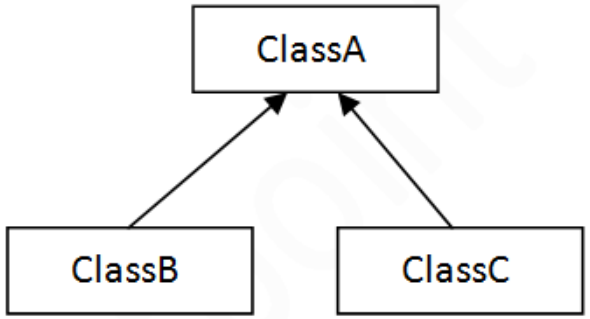
Inheritance - Type



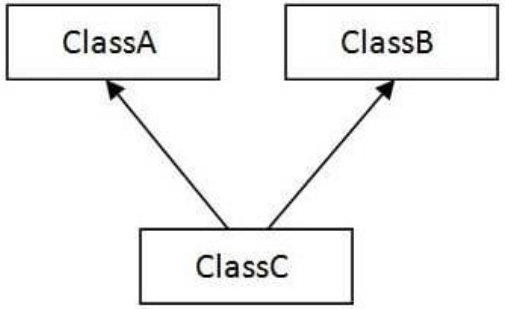
1) Single



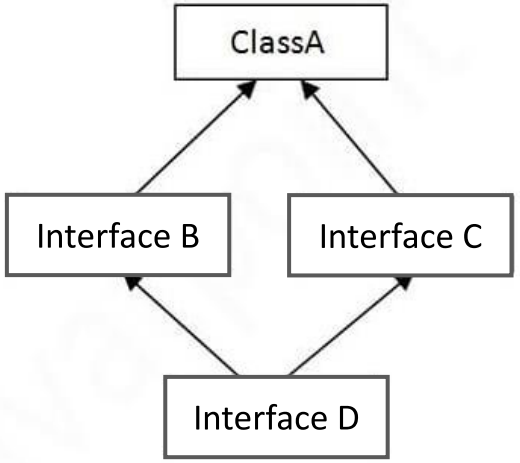
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Inheritance - Extends

- Pertama kita buat satu kelas baru di package berbeda

```
package day12.beda;

public class KelasInheritanceExtendsDasar{

    Run | Debug
    public static void main(String[] args) {

    }

}
```

Inheritance - Extends

- Lalu setelah nama Kelas, kita ketik **extends** seperti di line 3

```
package day12.beda;

public class KelasInheritanceExtendsDasar ex{
                                          extends

    Run | Debug
    public static void main(String[] args) {

    }

}
```

Inheritance - Extends

- Dalam contoh ini, setelah ketik extends, maka kita ketik nama kelas yang mau di-inheritance, yaitu KelasEncapsulation

```
package day12.beda;

import day12.KelasEncapsulation;

public class KelasInheritanceExtendsDasar extends KelasEncapsulation{

    Run | Debug
    public static void main(String[] args) {

    }

}
```

Inheritance - Extends

- Instance kelas dirinya sendiri

```
import day12.KelasEncapsulation;

public class KelasInheritanceExtendsDasar extends KelasEncapsulation {

    Run | Debug
    public static void main(String[] args) {
        KelasInheritanceExtendsDasar kied = new KelasInheritanceExtendsDasar();
        // lalu kita instance kelas dirinya sendiri
    }

}
```

Inheritance - Extends

- Akhirnya kita dapat menggunakan metod public dan protected dari kelas yang diextends.

```
public class KelasInheritanceExtendsDasar extends KelasEncapsulation{

    Run | Debug
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        KelasInheritanceExtendsDasar kied = new KelasInheritanceExtendsDasar();
        // lalu kita instance kelas dirinya sendiri

    kied.metodeProtected();
    kied.metodePublic();
    }
}
```

```
METODE PROTECTED
METODE PUBLIC
```

Inheritance – Extends - Example

- KelasBambang inheritance (warisi) KelasWilliam dengan menggunakan extends
- Jadi KelasBambang bisa panggil method KelasWilliam dengan instance dirinya sendiri.

```java
public class KelasWilliam {

    int tinggi = 165;
    String status = "sekolah";

    public void berlari() {
        System.out.println("Berlari Kencang");
    }

    public void melukis() {
        System.out.println("Melukis Indah");
    }

}
```

```java
public class KelasBambang extends KelasWilliam{

    public void cobaPanggil() {

        KelasBambang kb = new KelasBambang();
        kb.berlari();
        kb.melukis();

        int height = kb.tinggi;
        String pekerjaan = kb.status;

        System.out.println(height +" dan "+pekerjaan);

    }

    Run | Debug
    public static void main(String args[]) {
        KelasBambang kbi = new KelasBambang();
        kbi.cobaPanggil();
    }
}
```

Inheritance – Extends - Relationship

- Ada 2 relasi dalam inheritance extends yaitu
  1. IS-A
  2. HAS-A

Inheritance – Extends - Relationship

- KelasVespa IS-A KelasMotor → KelasVespa adalah sebuah KelasMotor
- KelasVespa HAS-A KelasMesin → KelasVespa punya sebuah KelasMesin

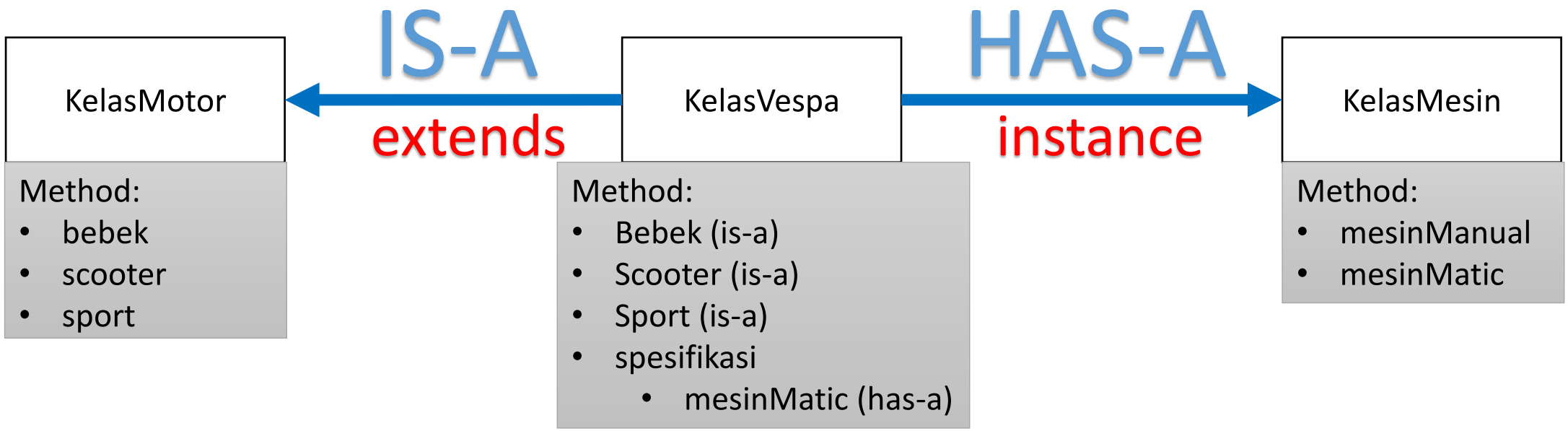
Inheritance – Extends - Relationship


```java
public class KelasMotor {

    public void bebek() {
        System.out.println("Motor Bebek");
    }

    public void scooter() {
        System.out.println("Motor Scooter");
    }

    public void sport() {
        System.out.println("Motor Sport");
    }

}
```

```java
public class KelasVespa extends KelasMotor{
    //KelasVespa Extends KelasMotor
    // Artinya KelasVespa IS-A KelasMotor

    public void spesifikasi(){
        //method spesifikasi

        KelasMesin km = new KelasMesin();
        km.mesinMatic();
      //KelasVespa instance KelasMesin
      //Artinya KelasVespa HAS-A KelasMesin

    }
}
```

```java
public class KelasMesin {

    public void mesinManual() {
        System.out.println("Mesin Bergigi");
    }

    public void mesinMatic() {
        System.out.println("Mesin Matik");
    }

}
```

Inheritance – Extends - Relationship



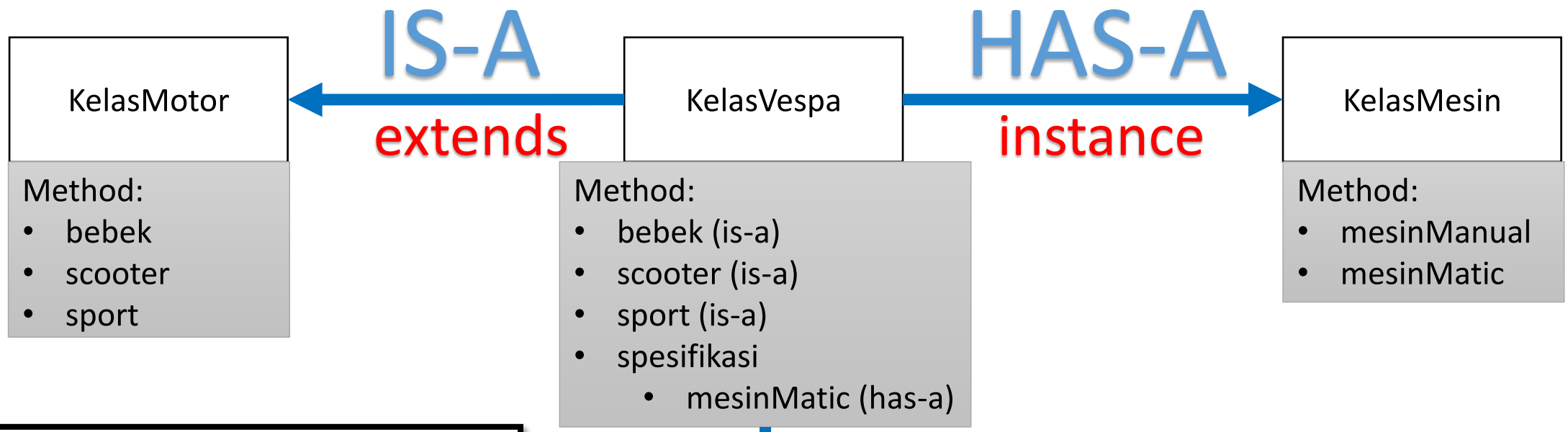
Inheritance – Extends - Relationship

```java
public class KelasScoopy {
    Run | Debug
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        KelasVespa kv = new KelasVespa();
        kv.scooter();
        // panggil method scooter dari KelasVespa
        // perlu diingat method scooter awalnya dari KelasMotor
        // krn KelasVespa extends KelasMotor
        // jadi KelasVespa punya method scooter secara otomatis

        kv.spesifikasi();
        // panggil method spesifikasi dari KelasVespa
        // di dalam method spesifikasi terdapat jalankan 1 method
        // yaitu method mesinMatic dari KelasMesin

        //Sehingga ketika KelasScoopy dijalankan
        // Outputnya seperti dibawah
        // Motor Scooter
        // Mesin Matik
    }
}
```

```
Motor Scooter
Mesin Matik
```

```java
public class KelasVespa extends KelasMotor{
    //KelasVespa Extends KelasMotor
    // Artinya KelasVespa IS-A KelasMotor

    public void spesifikasi(){
        //method spesifikasi

        KelasMesin km = new KelasMesin();
        km.mesinMatic();
      //KelasVespa instance KelasMesin
      //Artinya KelasVespa HAS-A KelasMesin

    }
}
```

Inheritance – Extends - Relationship




```
public class KelasScoopy {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        KelasVespa kv = new KelasVespa();
        kv.scooter();
        kv.spesifikasi();


    }

}
```

Instance KelasVespa

KelasScoopy

Running:
- scooter
- spesifikasi

```
Console
<terminated> KelasSc
Motor Scooter
Mesin Matik
```

Kodehive

Inheritance – Extends - Relationship

Jadi yang dimaksud dengan inheritance relationship adalah:

- IS-A menggunakan sintaks extends
- HAS-A menggunakan sintaks instance

Inheritance – Extends vs Implements

- Pada slide sebelumnya kita mengetahui bahwa inheritance atau pewarisan mempunyai dua turunan yaitu extends dan implements.
- Sebelumnya juga kita sudah mempelajari extends itu bagaimana dan serta konsep relationship di dalamnya yaitu IS-A dan HAS-A.
- Sekarang kita akan belajar mengenai apa itu implements.

Inheritance – Implements

Implements adalah konsep OOP yang digunakan untuk mengimplementasikan abstract class.

Dengan implements memungkinkan kita untuk dapat memanggil seluruh method yang dibuat pada interface dengan mudah, namun implements tidak membolehkan pemanggilan method satu persatu layaknya extends / instance.

Inheritance – Extends vs Implements

```java
public class OrangTua {

    public void mobil() {
        System.out.println("MOBIL");
    }

    public void motor() {
        System.out.println("MOTOR");
    }

    public void rumah() {
        System.out.println("RUMAH");
    }
}
```

```java
public class AnakKedua extends OrangTua{

    public void pilihWarisan() {
        AnakKedua ap = new AnakKedua();
        ap.motor();
    }
}
```

```java
public class AnakPertama extends OrangTua{

    public void pilihWarisan() {
        AnakPertama ap = new AnakPertama();
        ap.mobil();
    }
}
```

Ada 3 Kelas:

1. OrangTua
2. AnakPertama
3. AnakKedua

AnakPertama hanya pilih method mobil

AnakKedua hanya pilih method motor

Inheritance – Extends vs Implements

```java
public class OrangTua {

    public void mobil() {
        System.out.println("MOBIL");
    }

    public void motor() {
        System.out.println("MOTOR");
    }

    public void rumah() {
        System.out.println("RUMAH");
    }
}
```

```java
public class AnakKedua extends OrangTua{

    public void pilihWarisan() {
        AnakKedua ap = new AnakKedua();
        ap.motor();
    }
}
```

```java
public class AnakPertama extends OrangTua{

    public void pilihWarisan() {
        AnakPertama ap = new AnakPertama();
        ap.mobil();
    }
}
```

- Dengan Extends, AnakPertama maupun AnakKedua dapat memilih method dari yang di-extends-nya.
- Pertanyaannya:
- Ada cara lainkah jika ingin kelas yang me-inheritance, kelas tsb wajib memanggil seluruh method?

Inheritance – Extends vs Implements

```java
public class OrangTua {

    public void mobil() {
        System.out.println("MOBIL");
    }

    public void motor() {
        System.out.println("MOTOR");
    }

    public void rumah() {
        System.out.println("RUMAH");
    }
}
```

```java
public class AnakKetiga extends OrangTua{

    public void pilihWarisan() {
        AnakKetiga ap = new AnakKetiga();
        ap.mobil();
        ap.motor();
        ap.rumah();
    }
}
```

Pertanyaannya:

- Ada cara lainkah jika ingin kelas yang me-inheritance, kelas tsb wajib memanggil seluruh method?
- Kita bisa panggil methodnya satu-persatu seperti di kelas AnakKetiga

Inheritance – Extends vs Implements

```java
public class Leluhur {

    public void warisan1() {
        System.out.println("WARISAN 1");
    }

    public void warisan2() {
        System.out.println("WARISAN 2");
    }

    public void warisan3() {
        System.out.println("WARISAN 3");
    }

    //...

    public void warisan100() {
        System.out.println("WARISAN 100");
    }
}
```

```java
public class Keturunan extends Leluhur{

    public void pilihWarisan() {
        Keturunan k = new Keturunan();
        k.warisan1();
        k.warisan2();
        //..
        //..
        //..
        //..
    }
}
```

Pertanyaannya:

- Bagaimana kalau ada 1 Kelas memiliki banyak method?
- Leluhur semisal punya 100Method
- Keturunan wajib mewarisi 100method tsb
- Kemungkinan untuk lupa dipanggil salah satu method pasti ada

Inheritance – Extends vs Implements

```java
public class Leluhur {

    public void warisan1() {
        System.out.println("WARISAN 1");
    }

    public void warisan2() {
        System.out.println("WARISAN 2");
    }

    public void warisan3() {
        System.out.println("WARISAN 3");
    }

    //...

    public void warisan100() {
        System.out.println("WARISAN 100");
    }
}
```

```java
public class Keturunan extends Leluhur{

    public void pilihWarisan() {
        Keturunan k = new Keturunan();
        k.warisan1();
        k.warisan2();
        //..
        //..
        //..
        //..
    }
}
```

Pertanyaannya:

- Lalu bagaimana cara yang tepat jika kita ingin membuat suatu kelas yang inherit kelas lainnya, dan memakai seluruh method yang tapi tidak terlewat?
- Jawab:
- Dengan IMPLEMENTS

Inheritance - Implements

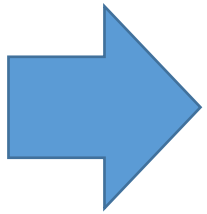
- Kita buat kelas OrangTuaBaru sama persis dengan Kelas OrangTua yang mana punya 3 method : mobil, motor, rumah.
- 3 method tersebut nantinya digunakan seluruhnya di kelas yang meng-inherit kelas OrangTuaBaru.

```
public interface OrangTuaBaru {

    public void mobil(); //abstract method

    public void motor(); //abstract method

    public void rumah(); //abstract method

}
```

Inheritance - Implements

- Selanjutnya kita ubah class menjadi interface

```java
public class OrangTuaBaru {

    public void mobil() {
        System.out.println("MOBIL");
    }

    public void motor() {
        System.out.println("MOTOR");
    }

    public void rumah() {
        System.out.println("RUMAH");
    }

}
```



```java
public interface OrangTuaBaru {

    public void mobil() {
        System.out.println("MOBIL");
    }

    public void motor() {
        System.out.println("MOTOR");
    }

    public void rumah() {
        System.out.println("RUMAH");
    }

}
```

Inheritance - Implements

- Ketika class diubah menjadi interface, maka akan muncul error. Kenapa? Karena kelas dengan tipe interface hanya dapat diisi oleh method dengan tipe abstract (artinya tidak ada isinya).

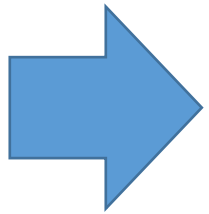
```
// public void mobil() artinya
// method namanya mobil tipe encapnya public
public void mobil() {
    System.out.println("MOBIL");
    // ini body dari method atau biasa disebut isi method
    // dalam hal ini cuma ada perintah cetak MOBIL
}
```

```
// public void mobil() artinya
// method namanya mobil tipe encapnya public
public void mobil(); // tidak ada isi method, jadi bisa disebut abstract method
```

Inheritance - Implements

- Jadi abstract method berarti method yang tidak punya body (isi). Kenapa abstract? Karena tidak ada isi jadi belum jelas isi atau tujuan dari method tersebut.
- Lalu kita ubah semua methodnya menjadi abstract method

```
public interface OrangTuaBaru {

    public void mobil() {
        System.out.println("MOBIL");
    }

    public void motor() {
        System.out.println("MOTOR");
    }

    public void rumah() {
        System.out.println("RUMAH");
    }

}
```



```
public interface OrangTuaBaru {

    public void mobil(); //abstract method

    public void motor(); //abstract method

    public void rumah(); //abstract method

}
```

Inheritance - Implements

- Selanjutnya kita buat 1 kelas baru yang akan inherit kelas OrangTuaBaru, kita namakan AnakPertamaBaru

```
public class AnakPertamaBaru {


}
```

Inheritance - Implements

- Lalu kita ketik implements

```
public class AnakPertamaBaru implements {


}
```

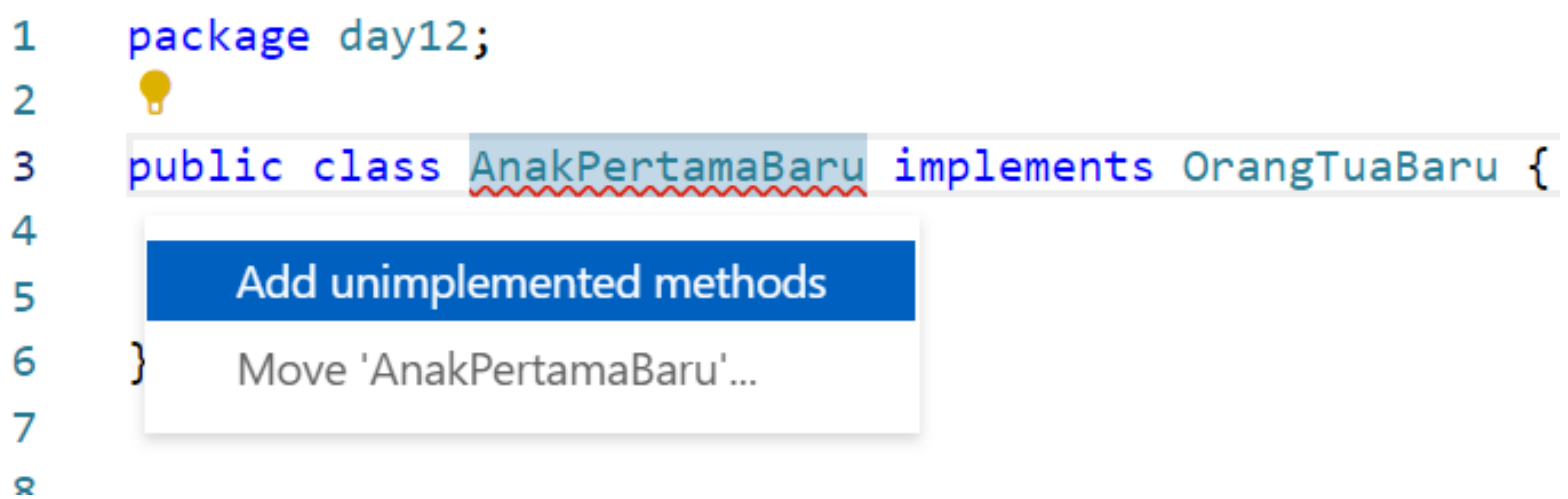
Inheritance - Implements

- Tambahkan Kelas yang mau di-inherit dengan cara implements, yaitu OrangTuaBaru

```
public class AnakPertamaBaru implements OrangTuaBaru {



}
```

Inheritance - Implements

- Lalu kitak klik icon errornya, klik unimplement method



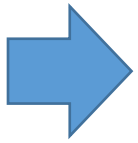
Inheritance - Implements

```
public class AnakPertamaBaru implements OrangTuaBaru{

    @Override
    public void mobil() {
        // TODO Auto-generated method stub
        System.out.println("KELUARGA");
        //isinya bisa kita sesuaikan
    }

    @Override
    public void motor() {
        // TODO Auto-generated method stub
        System.out.println("MATIK");
    }

    @Override
    public void rumah() {
        // TODO Auto-generated method stub
        System.out.println("2LANTAI");
    }

}
```

- Akhirnya kita telah berhasil punya kelas AnakPertamaBaru yang memiliki seluruh method dari OrangTuaBaru dengan cara implements

Inheritance – Implements – Define Method

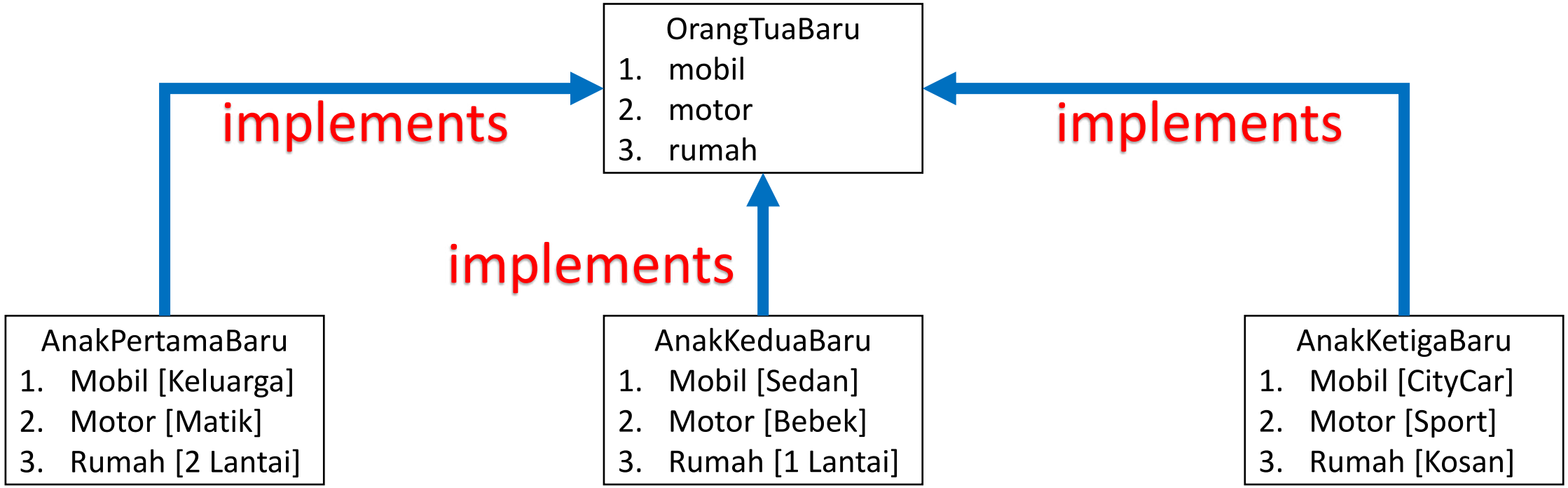
```java
public class AnakPertamaBaru implements OrangTuaBaru{

    @Override
    public void mobil() {
        // TODO Auto-generated method stub

    }

    @Override
    public void motor() {
        // TODO Auto-generated method stub
    }

    @Override
    public void rumah() {
        // TODO Auto-generated method stub
    }

}
```



```java
public class AnakPertamaBaru implements OrangTuaBaru{

    @Override
    public void mobil() {
        // TODO Auto-generated method stub
        System.out.println("KELUARGA");
        //isinya bisa kita sesuaikan
    }

    @Override
    public void motor() {
        // TODO Auto-generated method stub
        System.out.println("MATIK");
    }

    @Override
    public void rumah() {
        // TODO Auto-generated method stub
        System.out.println("2LANTAI");
    }

}
```

Inheritance – Implements - Define Method

- Kita punya 3 kelas berbeda yang implements 1 kelas yang sama dengan tipe interface dimana masing-masing berbeda isi methodnya.



Inheritance – Implements - Define Method

```
public interface OrangTuaBaru {

    public void mobil(); //abstract method

    public void motor(); //abstract method

    public void rumah(); //abstract method

}
```

Inheritance – Implements - Define Method

```java
public class AnakPertamaBaru implements OrangTuaBaru{

    @Override
    public void mobil() {
        // TODO Auto-generated method stub

    }

    @Override
    public void motor() {
        // TODO Auto-generated method stub
    }

    @Override
    public void rumah() {
        // TODO Auto-generated method stub
    }

}
```

```java
public class AnakKeduaBaru implements OrangTuaBaru{

    @Override
    public void mobil() {
        // TODO Auto-generated method stub
        System.out.println("SEDAN");
        //isinya bisa kita sesuaikan
    }

    @Override
    public void motor() {
        // TODO Auto-generated method stub
        System.out.println("BEBEK");
    }

    @Override
    public void rumah() {
        // TODO Auto-generated method stub
        System.out.println("1LANTAI");
    }

}
```

```java
public class AnakKetigaBaru implements OrangTuaBaru{

    @Override
    public void mobil() {
        // TODO Auto-generated method stub
        System.out.println("CITYCAR");
        //isinya bisa kita sesuaikan
    }

    @Override
    public void motor() {
        // TODO Auto-generated method stub
        System.out.println("SPORT");
    }

    @Override
    public void rumah() {
        // TODO Auto-generated method stub

    }

}
```