Pemrograman Berorientasi Objek
Inheritance dan Relasi Kelas

Adam Mukharil Bachtiar

Teknik Informatika UNIKOM

Inheritance dan Relasi Kelas

1. Definisi inheritance
2. SubClass dan SuperClass
3. Istilah inheritance
4. Pengujian inheritance
5. Format sintaks inheritance
6. Hubungan hak akses
7. Kata kunci super

Inheritance dan Relasi Kelas

8. Relasi Kelas
9. Multiplicity
10. Association
11. Composition
12. Aggregation
13. Dependency


INHERITANCE

Definisi Inheritance

MENYATAKAN PEWARISAN DARI SATU KELAS KE KELAS LAINNYA. C++ MENDUKUNG MULTIPLE INHERITANCE SEDANGKAN JAVA HANYA MENDUKUNG SINGLE INHERITANCE.

SuperClass dan SubClass

1

SUPERCLASS : CLASS YANG DITURUNKAN (KELAS BASIS ATAU ANCHESTOR CLASS).

SUBCLASS : CLASS YANG MEWARISI (DERIVED CLASS).

2

Manfaat Inheritance

1

SPESIALISASI : MEMBUAT KELAS BARU YANG LEBIH SPESIFIK.

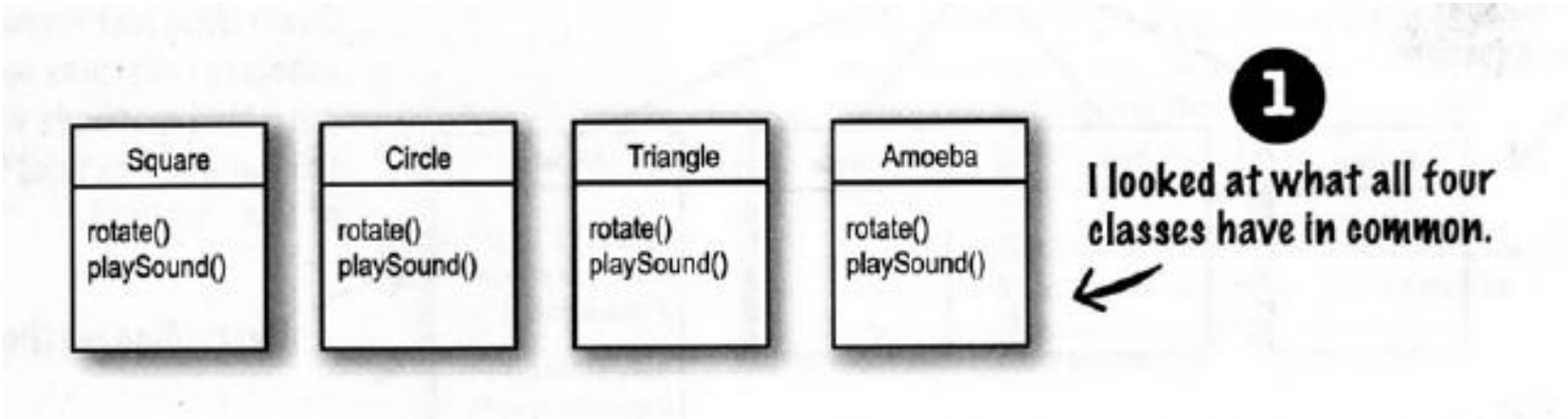
REUSABILITY : PENGGUNAAN KEMBALI KODE DARI SUPER CLASS

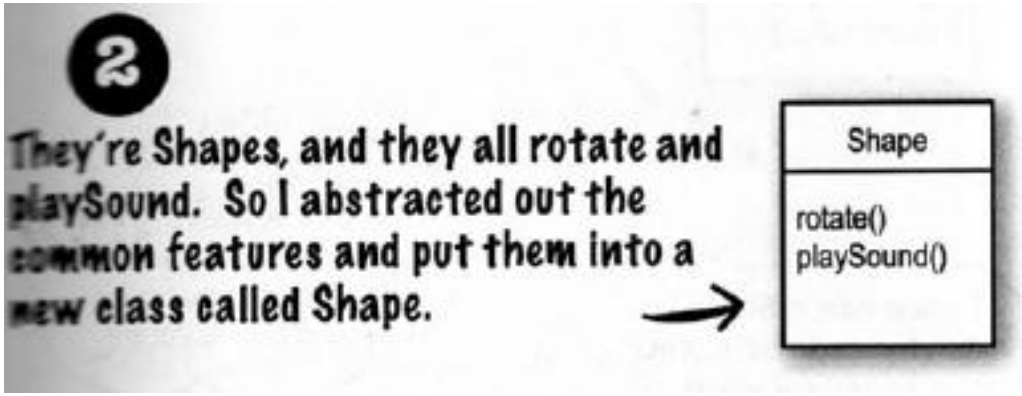
2

Keuntungan Inheritance

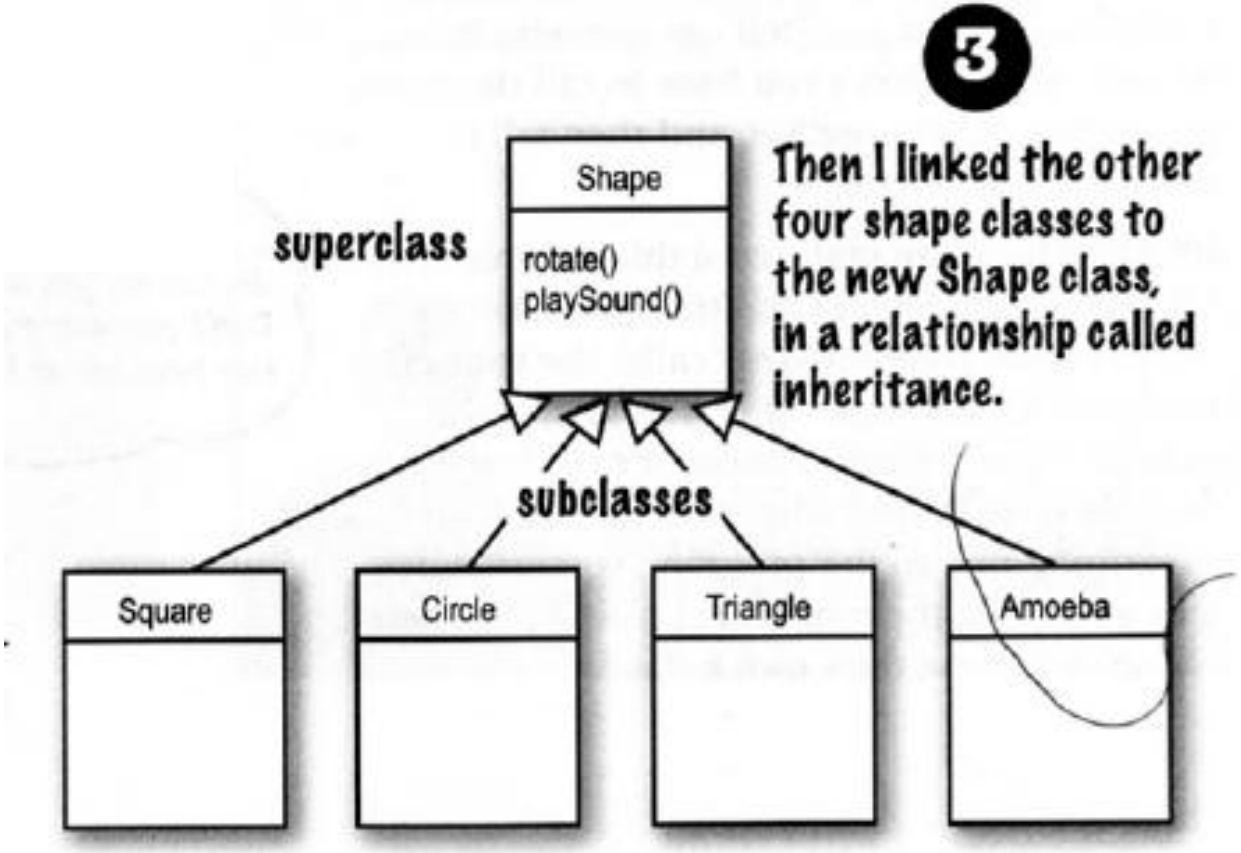
1. Bisa memanfaatkan atribut dan method kelas super.
2. Bisa membuat atribut dan method baru pada kelas anak.
3. Bisa memanfaatkan konstruktor pada kelas super.

Langkah-Langkah Inheritance

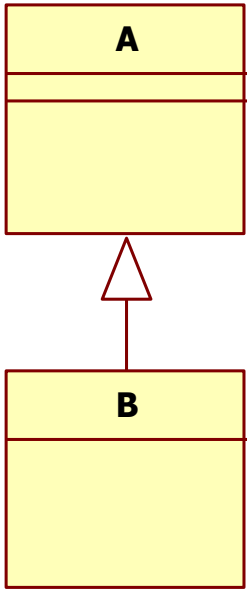




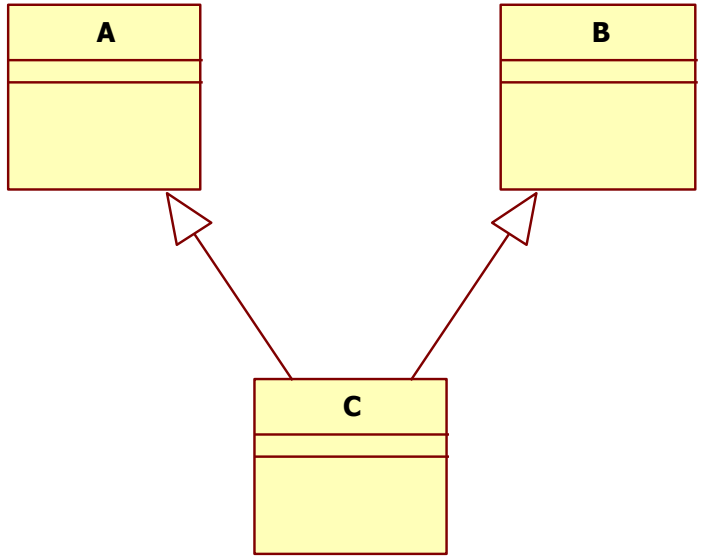
Langkah-Langkah Inheritance



Istilah Inheritance

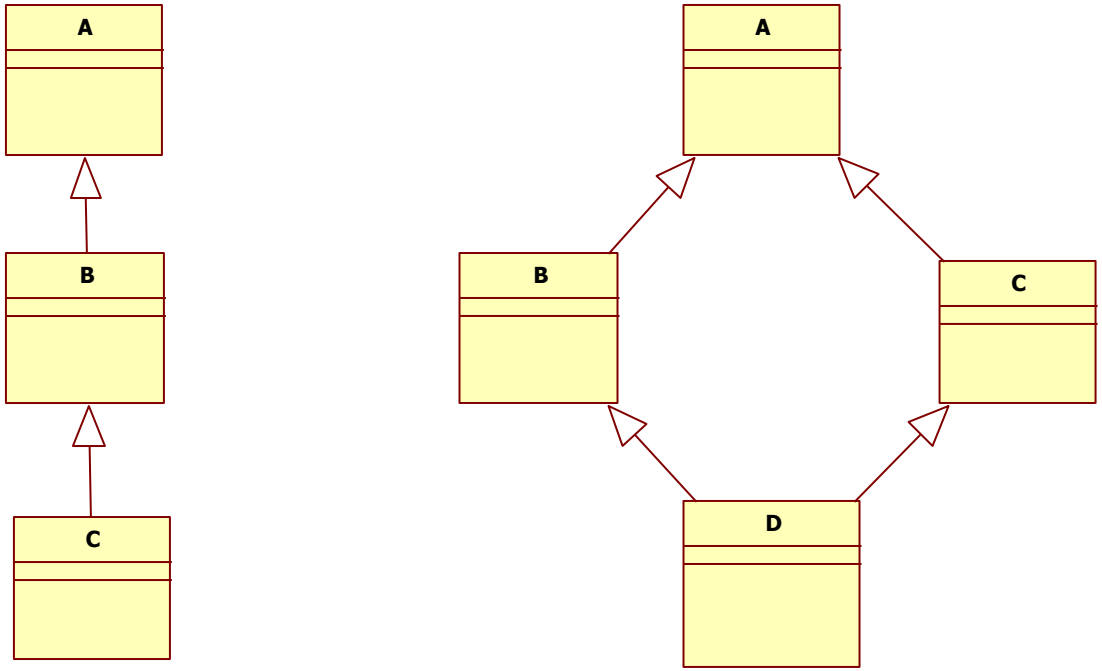


Single Inheritance



Multiple Inheritance

Istilah Inheritance



Repeated Inheritance

Pengujian Inheritance

Contoh:

Burung (mempunyai sayap, bertelur, dan berparuh). Salah satu contoh burung adalah elang.

Pengujiannya:

Elang **IS A** burung.

Format Sintaks Inheritance

Format C++:

```
class <subclass> : <hak_akses> <superclass>
```

Contoh:

```
class mobil:public kendaraan
```

Format Sintaks Inheritance

Format Java:

```
<hak_akses> class <subclass> extends <superclass>
```

Contoh:

```
public class Mobil extends Kendaraan
```

Gambaran Inheritance

C++

```
class A
{
      . . .
};

class B : public A
{
     . . .
}
```

JAVA

```
class A{
     . . .
};



public class B
extends A{
     . . .
}
```

Inheritance C++

```
class bilangan{
      private:
              int x;
              int y;

      public:
             bilangan(){
                       x=3;
                       y=4;
             }

             int getx(){
                 return x;
             }

             int gety(){
                 return y;
             }
};
```

```
class jumlah_bilangan : public
bilangan{
  public:
   void tampil(){
     int a,b;
     a=getx();
     b=gety();
     cout<<"x + y = "
         <<(a+b)<<endl<<endl;
   }
};

int main(int argc, char *argv[])
{
    jumlah_bilangan bil;
    bil.tampil();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Inheritance Java

```
public class Bilangan {
    private int x;
    private int y;

    public Bilangan(){
        x=3;
        y=4;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }
}
```

Inheritance Java

```
public class JumlahBilangan extends Bilangan {
    public void tampil(){
        int a,b;
        a=getX();
        b=getY();
        System.out.println("x + y = "+(a+b));
    }
}

public class MainBilangan {
    public static void main(String[] args) {
        JumlahBilangan bil=new JumlahBilangan();
        bil.tampil();
    }
}
```

Pemanggilan Method

make a new Wolf object

```
Wolf w = new Wolf();
```

calls the version in Wolf

```
w.makeNoise();
```

calls the version in Canine

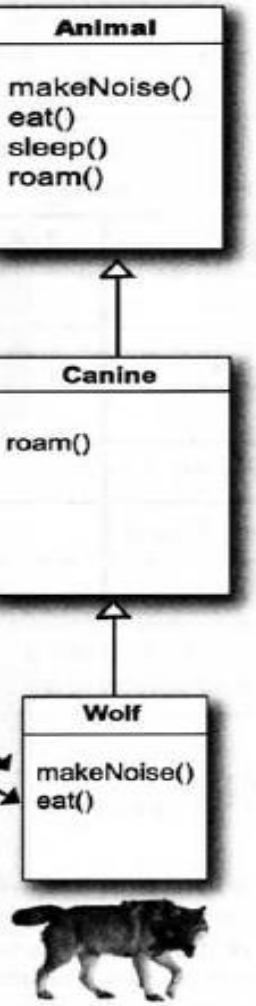
```
w.roam();
```

calls the version in Wolf

```
w.eat();
```

calls the version in Animal

```
w.sleep();
```



Hubungan Hak Akses

Penentu Hak Akses	Super Class	Sub Class
Private	Private	Tidak Diwariskan
	Protected	Private
	Public	Private
Protected	Private	Tidak Diwariskan
	Protected	Protected
	Public	Protected
Public	Private	Tidak Diwariskan
	Protected	Protected
	Public	Public

Kata Kunci Super

KATA KUNCI SUPER DIGUNAKAN UNTUK MENGAKSES METHOD MILIK SUPERCLASS. FORMAT :

SUPER.<NAMA METHOD SUPERCLASS>

Penggunaan Kata Kunci Super (JAVA)

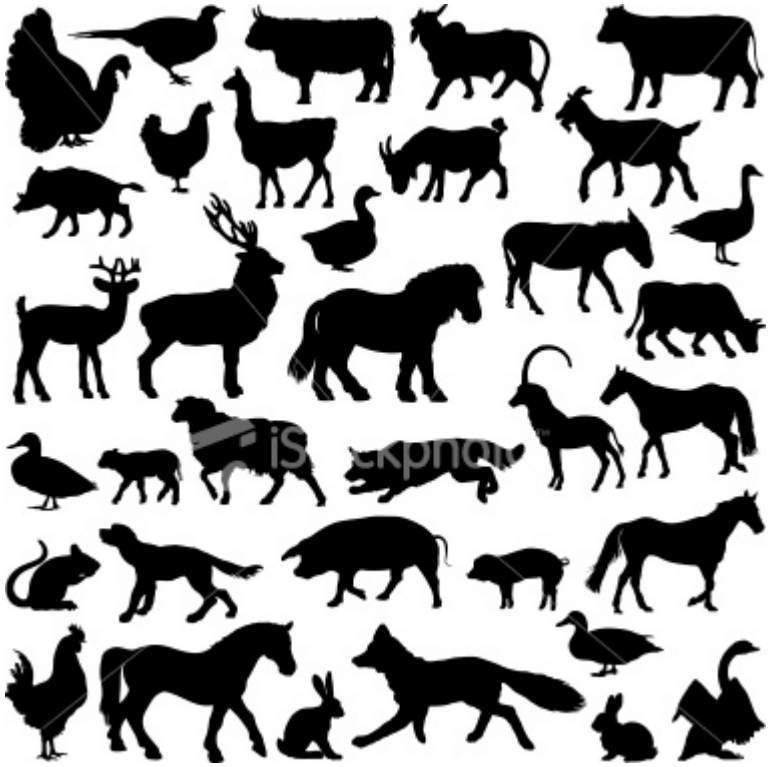
```
public class OrangTua {
    public void cetakOrtu(){
        System.out.println("Halo saya milik orang tua!");
    }
}

public class Anak extends OrangTua{
    public void cetakAnak(){
        super.cetakOrtu();
        System.out.println("Saya milik anak!");
    }
}

public class TesterSuper {
public static void main(String[] args) {
        Anak x=new Anak();
        x.cetakAnak();
    }
}
```

RELASI KELAS

HAS A VS IS A

TIDAK SEMUA RELASI KELAS MENGGUNAKAN RELASI IS A. BEBERAPA CLASS MENGGUNAKAN KONSEP HAS A SEBAGAI RELASI.

Hubungannya IS A....



Hubungannya HAS A....

BAGAIMANA PBO MENANGANI HAL INI???

Relasi Kelas

1. PBO MENGAMBIL REALITA DARI KEHIDUPAN DUNIA NYATA.

2. OBJEK-OBJEK DI DUNIA NYATA MEMILIKI RELASI TERTENTU .

3. RELASI ADALAH KONEKSI LOGIS, HUBUNGAN ANTAR OBJEK/KELAS

Multiplicity

1. KARAKTERISTIK DARI RELASI YANG TERJADI.

2. LEVEL RELASI KELAS:
   - INSTANCE LEVEL (TINGKAT OBJEK)
   - CLASS LEVEL (TINGKAT KELAS)
   - GENERAL LEVEL

Multiplicity

3 MENUNJUKKAN JUMLAH/KARDINALITAS HUBUNGAN ANTAR CLASS.

4 MENUNJUKKAN APAKAH RELASI TERSEBUT MERUPAKAN OPSIONAL ATAU MANDATORY (WAJIB).

Contoh Multiplicity

Antara class komik dan class halaman:

- Mutliplicity: **komik** 1 → **halaman** 1..*
- Komik minimal berjumlah satu
- Halaman komik minimal 1 sampai ~ (tak terhingga)
- Sifat : **Mandatory**

Multiplicity

Multiplicity	Keterangan
0..1	0 atau 1
1	Tepat 1
0..*	0 atau lebih
1..*	1 atau lebih
*	Tidak tentu jumlahnya

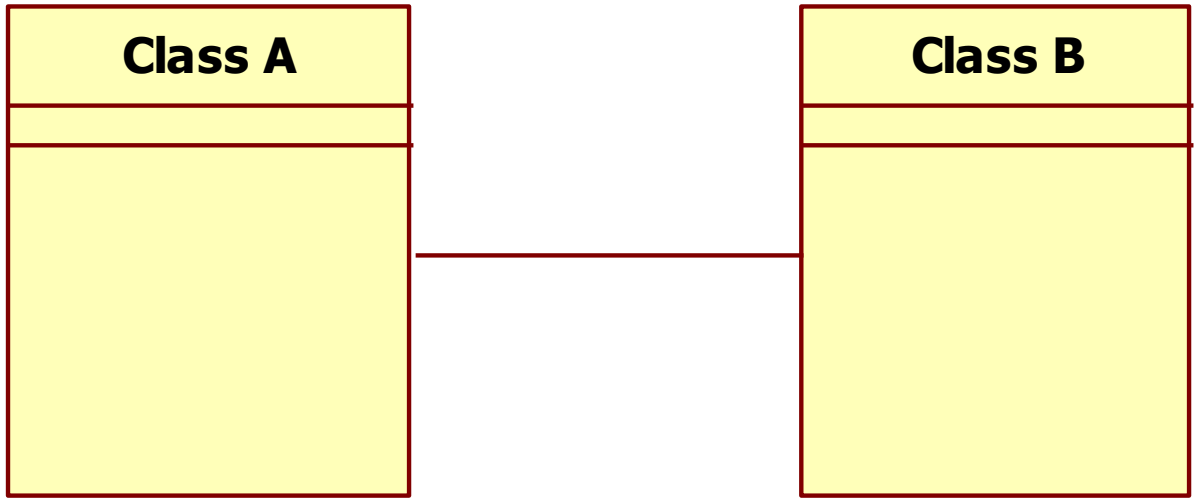
Relasi Kelas

1. Association
2. Composition
3. Aggregation
4. Dependency

Association

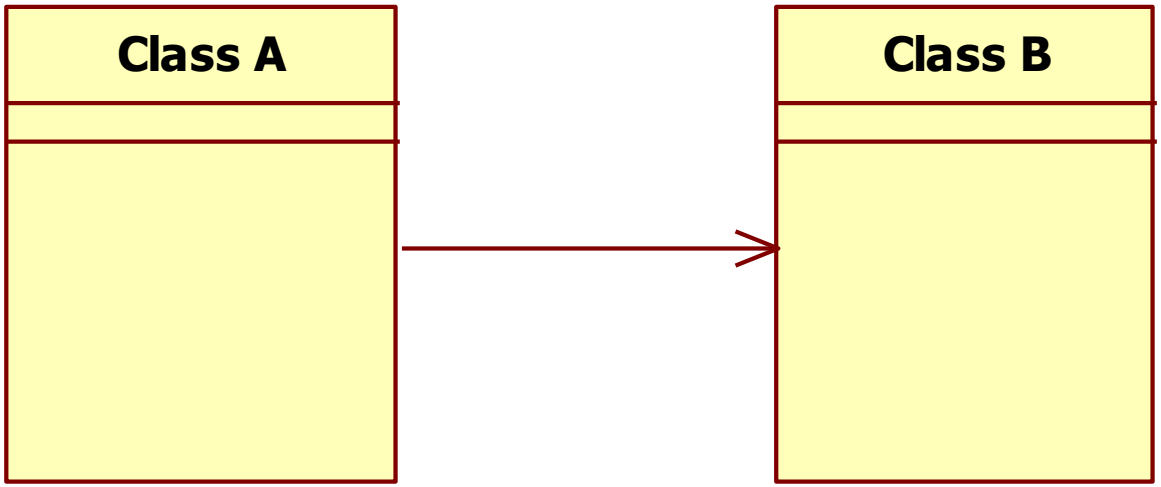
1. Relasi yang terjadi pada class-class dimana salah satu instance dari class tersebut **menjadikan** instance dari class lainnya sebagai **atribut** dirinya.
2. Relasi struktural yang menunjukkan **penggunaan** suatu class di class lainnya.
3. Association bisa **uni-directional** (satu arah) atau **bidirectional** (dua arah).

Association



Bidirectional (dua arah)

Association



Uni-directional (satu arah)

Transformasi Kode Association

Object design model before transformation

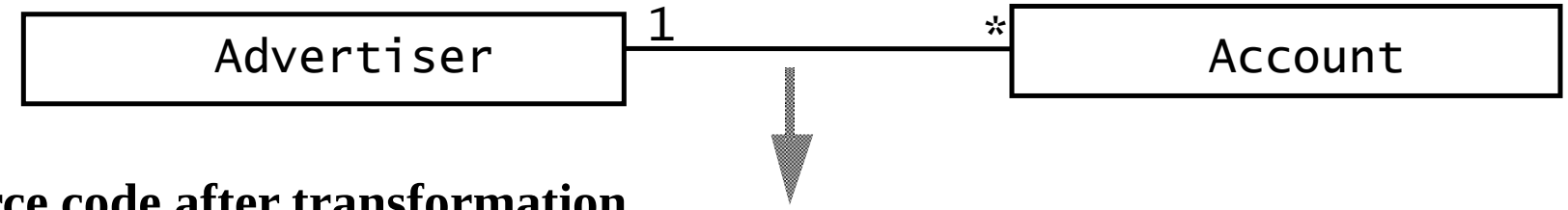
Source code after transformation ?

```
public class Advertiser {
    /* The account field is initialized
     * in the constructor and never
     * modified. */

    private Account account;

    public Advertiser() {
          account = new Account(this);
    }
    public Account getAccount() {
          return account;
    }
}
```

```
public class Account {
    /* The owner field is
    initialized
     * during the constructor
    and
     * never modified. */

    private Advertiser owner;

    public Account(Advertiser
    owner) {
          this.owner = owner;
    }
    public Advertiser getOwner()
    {
          return owner;
    }
}
```

Transformasi Kode Association

Object design model before transformation



Source code after transformation

```
public class Advertiser {
   private Set accounts;

   public Advertiser() {
        accounts = new HashSet();
   }

   public void addAccount(Account
   a) {
        accounts.add(a);
        a.setOwner(this);
   }

   public void
   removeAccount(Account a) {
        accounts.remove(a);
        a.setOwner(null);
   }
}
```

```
public class Account {
   private Advertiser owner;

   public void setOwner(Advertiser
   newOwner) {
     if (owner != newOwner) {
          Advertiser old = owner;
          owner = newOwner;
          if (newOwner != null)

     newOwner.addAccount(this);
          if (oldOwner != null)

     old.removeAccount(this);
     }
   }
}
```

Transformasi Kode Association

Object design model before transformation

Source code after transformation

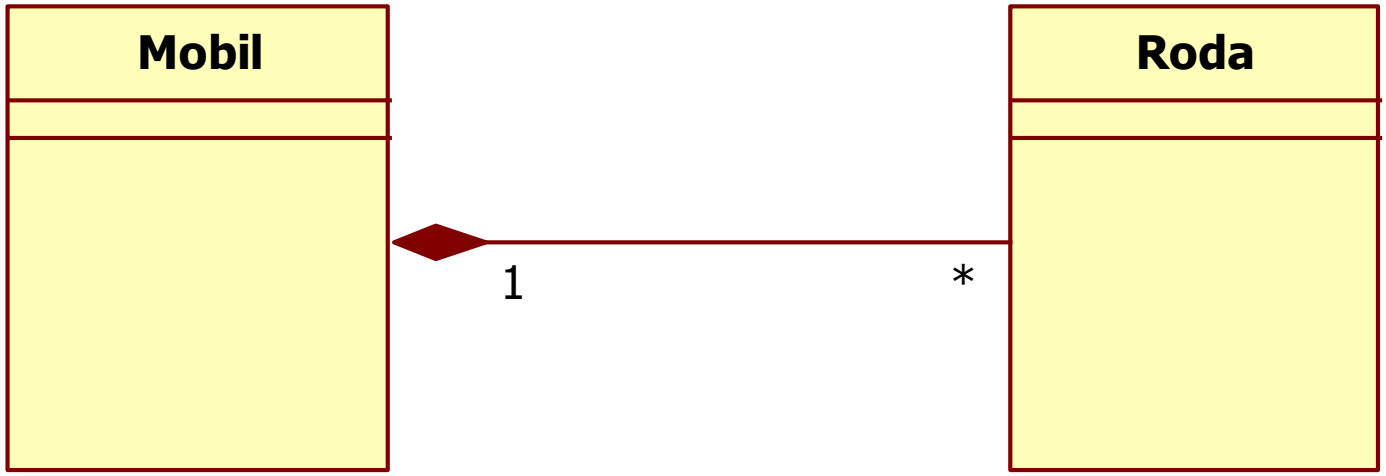
```
public class Tournament {
   private List players;
   public Tournament() {
      players = new
   ArrayList();
   }
   public void addPlayer(Player
   p) {
      if (!players.contains(p))
   {
            players.add(p);

      p.addTournament(this);
      }
   }
}
```

```
public class Player {
   private List tournaments;
   public Player() {
      tournaments = new
   ArrayList();
   }
   public void
   addTournament(Tournament t) {
      if
   (!tournaments.contains(t)) {
      tournaments.add(t);
      t.addPlayer(this);
      }
   }
}
```

Composition

1. Menyusun objek-objek **sederhana** menjadi suatu objek yang lebih **kompleks**.
2. Relasi “**Has-a**”.
3. Contoh: RAM, Processor, Motherboard, Harddisk membentuk **komputer**.
4. “A computer **has a** processor”.

Composition



Contoh Composition (JAVA)

```
public class Roda {
    private String merk;
    private int ring;

    public Roda(String merk, int ring) {
        this.merk = merk;
        this.ring = ring;
    }

    public String getMerk() {
        return merk;
    }

    public int getRing() {
        return ring;
    }
}
```

Contoh Composition (JAVA)

```
public class Mobil {
    Roda roda;//komposisi --> has a
    String warna;

    public void setWarna(String warna) {
        this.warna = warna;
    }

    public String getWarna() {
        return warna;
    }

    public void setRoda(String m, int r){
        roda=new Roda(m,r);
    }

    public String getMerkRoda() {
        return roda.getMerk();
    }

    public int getRingRoda(){
        return roda.getRing();
    }
}
```

Contoh Composition (JAVA)

```
public class TesterRelasi {

public static void main(String[] args) {

        //Komposisi
        Mobil mobil=new Mobil();
        mobil.setWarna("Biru");
        mobil.setRoda("Bridgestone",15);

        System.out.println("Warna mobil      : "+mobil.getWarna());
        System.out.println("Merk roda mobil  : "+mobil.getMerkRoda());
        System.out.println("Ukuran ring roda : "+mobil.getRingRoda());

}

}
```

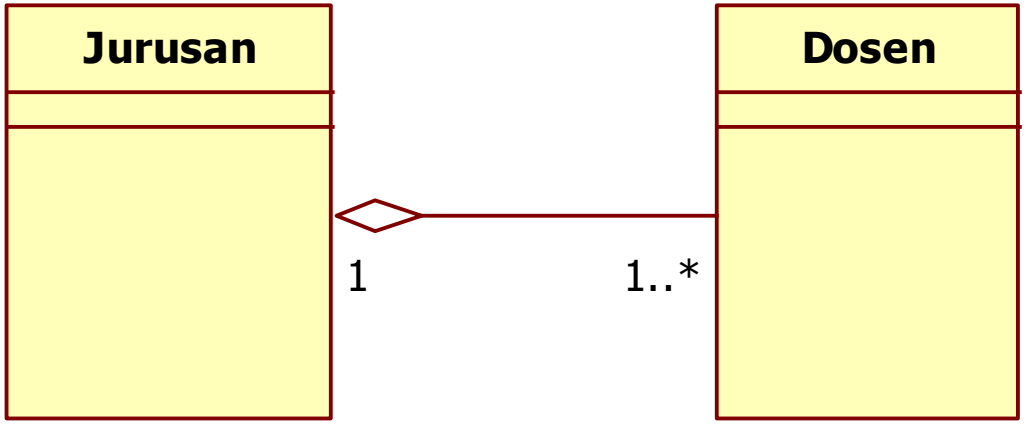
Aggregation

1. Mirip seperti composition.
2. Perbedaannya terletak pada kekuatan dan kepemilikan objek pada kelas-kelas yang berelasi.
3. Aggregation memiliki relasi yang lebih lemah dibandingkan composition.

Contoh Aggregation

1. Dosen-dosen berkumpul membentuk suatu jurusan.
2. Jurusan berkumpul membentuk fakultas.
3. Fakultas-fakultas dikumpulkan menjadi universitas.

Contoh Aggregation



Contoh Aggregation (JAVA)

```
public class Dosen {
    private String nip;
    private String nama;

    public Dosen(String nim, String nama) {
        this.nip = nim;
        this.nama = nama;
    }

    public String getNama() {
        return nama;
    }

    public String getNim() {
        return nip;
    }
}
```

Contoh Aggregation (JAVA)

```
public class Jurusan {
    private String namaJurusan;
    private Dosen dosen;

    public Jurusan(String namaJurusan) {
        this.namaJurusan = namaJurusan;
    }

    public void setDosen(Dosen dosen) {
        this.dosen = dosen;
    }

    public Dosen getDosen() {
        return dosen;
    }

    public String getNamaJurusan() {
        return namaJurusan;
    }
}
```

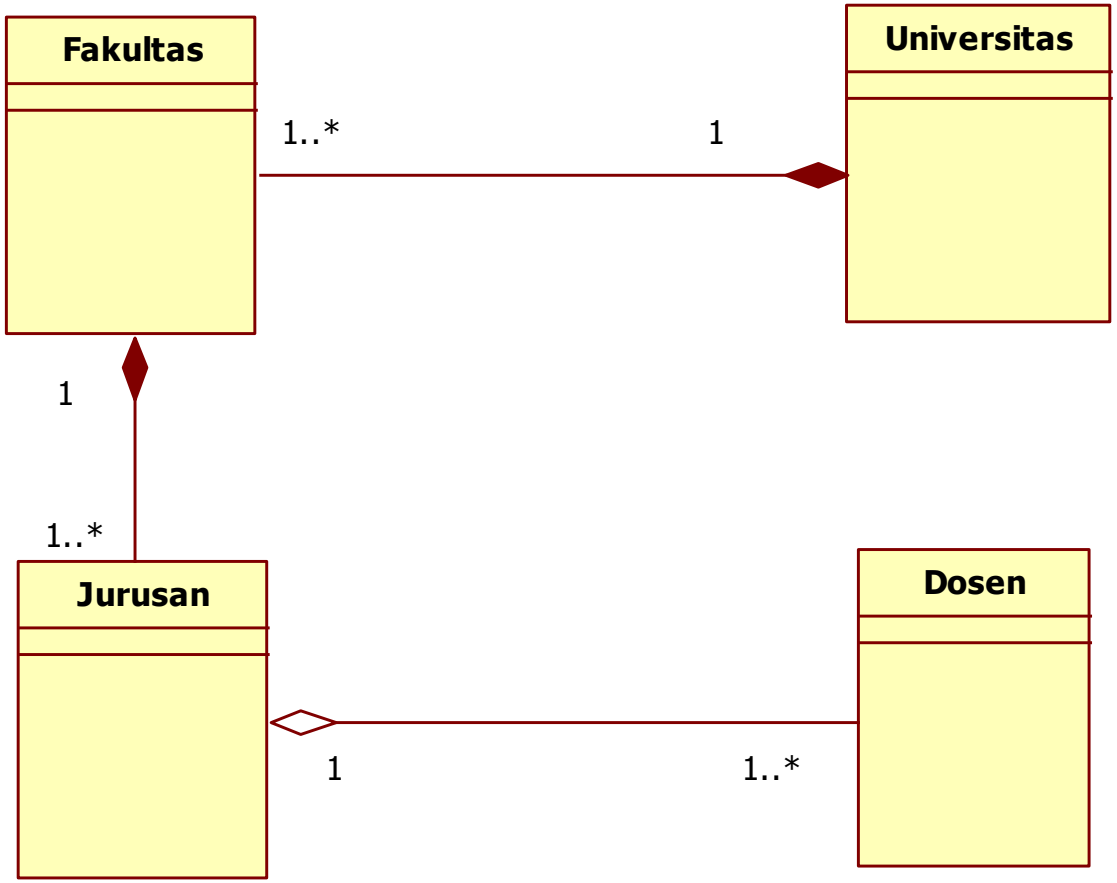
Contoh Aggregation (JAVA)

```
public class TesterAgregasi {

public static void main(String[] args) {
        Dosen dosen=new Dosen("1","Adam");
        Jurusan jurusan=new Jurusan("Teknik Informatika");

        jurusan.setDosen(dosen);

        System.out.println("NIP Dosen       : "+jurusan.getDosen().getNim());
        System.out.println("Nama Dosen      : "+jurusan.getDosen().getNama());
        System.out.println("Jurusan         : "+jurusan.getNamaJurusan());
    }
}
```

Composition VS Aggregation

1. **Agregasi tidak ada kepemilikan**. Bila objek yang dibentuk hilang maka objek-objek penyusunnya tetap ada.
2. **Komposisi ada kepemilikan**. Bila objek yang dibentuk hilang maka objek-objek penyusunnya akan hilang juga.

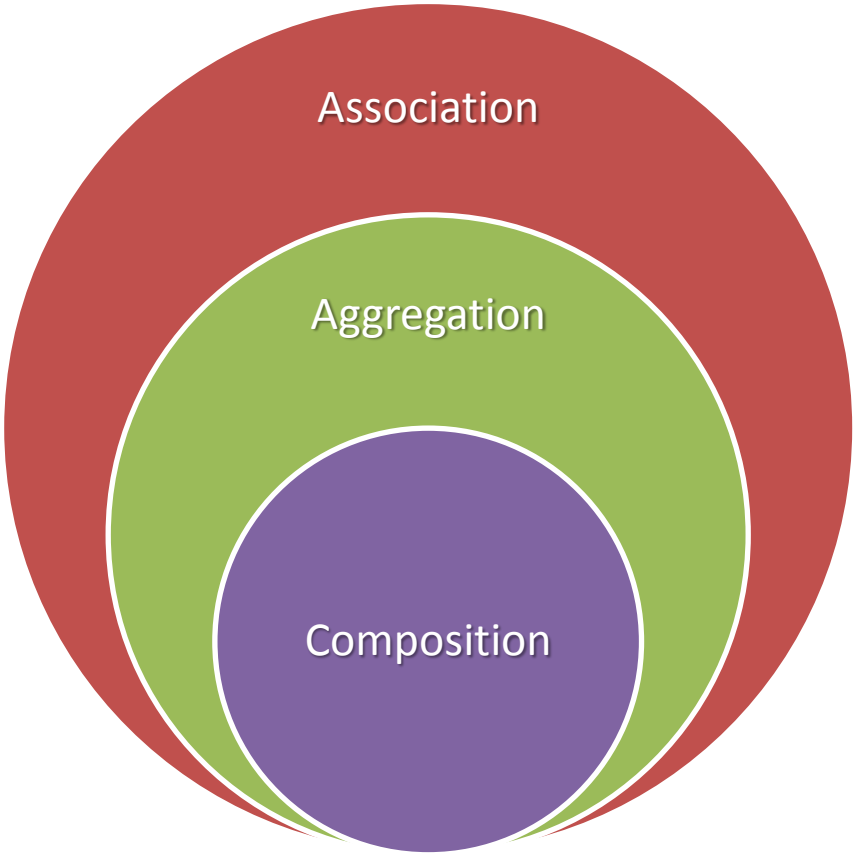
Composition VS Aggregation



Bila Universitas ditutup maka Fakultas dan Jurusan akan hilang akan tetapi

Dosen tetap akan ada. Begitupun relasi antar Fakultas dengan Jurusan

Association VS Composition VS Aggregation



Dependency

1. Relasi yang menggambarkan penggunaan suatu class pada class yang lainnya.
2. Contoh: class A memiliki dependency dengan kelas B maka jika kelas B berubah maka kelas A pun akan berubah.

Dependency

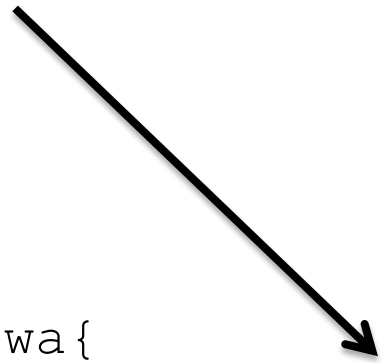
Bentuk Dependency

1. Penggunaan kelas B sebagai **parameter method** di kelas A.
2. Penggunaan kelas B sebagai **return value** di kelas A.
3. Penggunaan kelas B sebagai **variabel lokal** di kelas A.

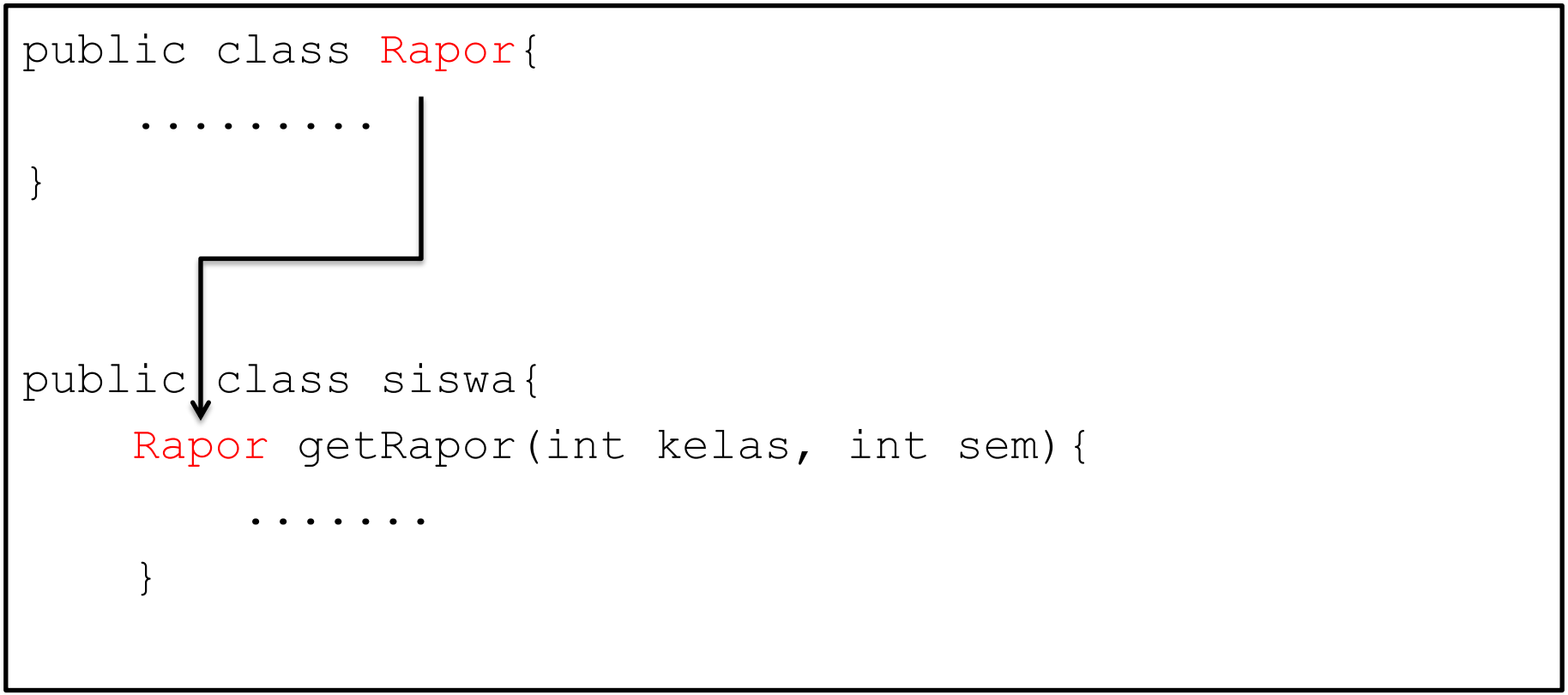
Bentuk Dependency

Dependency Parameter (JAVA)

```
public class Rapor{
    .........
}


public class siswa{
    boolean kenaikanKelas(Rapor rapor,int kelas){
        .......
    }
```



Dependency Return Value (JAVA)



Dependency Variabel Lokal (JAVA)

```
public class Rapor{
    .........
}


public class siswa{
    void isiRapor(){
        Rapor rapor=new Rapor();
    }
```

CHARLESPHOENIX.COM

THE END IN KODACHROME, SOMEWHERE, USA, 1969