Pemrograman Berorientasi Obyek

Package & Hubungan Antar Kelas

anton@ukdw.ac.id

Package

- Package dalam Java berarti melakukan **pengelompokan** terhadap class-class yang berelasi ke dalam **satu unit kerja**.
- Kita bisa memakai package-package default Java ataupun yang kita buat sendiri, dengan cara **import**.
- Jika kita melakukan **import** terhadap satu package, maka kita bisa memakai **semua class** dalam package tersebut.

Package

- Semua .class yang terletak pada satu direktori tertentu dengan sendirinya membuat satu **package** yang **tidak bernama**
- Nama paket biasanya ditulis dgn **huruf kecil**
- Package java diawali dengan **java.*** dan **javax.***
- Nama paket biasanya diberi nama sesuai dengan pembuatnya
  - Bersifat terbalik (reverse)
  - **id.ac.ukdw.www**

Package

- Setiap package harus **unik**
- Setiap class dalam 1 package dapat saling akses (ingat sifat class modifier!)
- Package dapat terdiri dari: **class, abstract class** dan **interfaces**
- Bisa digabungkan dalam 1 file **JAR** (Java Archive)

Package

- Secara **default**, Java mengimport package **java.lang** sehingga kita bisa memakai class-class seperti System, Integer, String walaupun kita belum mengimport package apapun.

- Syntax untuk import adalah:

```
import <nama package>;
```

Package

- Contoh jika kita ingin memakai **class Date** dalam **package util**, maka kita harus menulis:

```
import java.util.Date;
import java.util.*;
```

- Statement pertama mengimport satu class **Date** secara spesifik sedangkan statement kedua mengimport semua class di package **util**.

Package

- Cara lain untuk menggunakan class dari package lain adalah menggunakan **explicit package referencing**.
- Dilakukan dengan cara memakai nama package **secara lengkap** untuk mendeklarasikan sebuah objek dari class tersebut.
- Contoh:

```
java.util.Date  tgl;
```

Package bawaan Java 1.6

`java.lang` — basic language functionality and fundamental types

`java.util` — collection data structure classes

`java.io` — file operations

`java.math` — multiprecision arithmetics

`java.nio` — the New I/O framework for Java

`java.net` — networking operations, sockets, DNS lookups, ...

`java.security` — key generation, encryption and decryption

`java.sql` — Java Database Connectivity (JDBC) to access databases

`java.awt` — basic hierarchy of packages for native GUI components

`javax.swing` — hierarchy of packages for platform-independent rich GUI components

`java.applet` — classes for creating and implementing applets

Package

- Jika menggunakan statement **import**, maka harus diletakkan **di luar class** dan berada di baris-baris **awal** program java kita.

- Untuk membuat package kita dapat menuliskan:
  - **`package <nama package>;`**
  - di awal file java kita (paling atas/baris pertama)

Package

- Misalkan kita akan membuat satu package dimana kita akan meletakkan class **mahasiswa** dengan class-class **lain yang berelasi**.
- Yang perlu kita lakukan pertama adalah membuat **class-class** tersebut
- Kemudian membuat **folder** bagi class tersebut (misalkan package tersebut bernama **universitas**)

Package

- Semua class yang akan kita kelompokkan harus diletakkan dalam **satu folder** dengan nama sama dengan **nama package**-nya.
- Maka dari itu, kita masukkan semua class yang akan kita jadikan package universitas ke dalam folder **universitas**.
- Kita juga perlu menambahkan kode berikut ke bagian atas tiap file java (tiap class) yang akan dibuat dalam 1 package

```
package universitas;
```

Package

- Untuk mengkompile class-class dalam package, cukup dikompile dari **luar folder** package tersebut.

```
C:\javac universitas/*.java
```

Package

- Package juga dapat dibuat **nested** (bertumpuk/bersarang) yang berarti satu package di dalam package lain.
- Hal ini menunjukkan bahwa package bersifat **hirarkis**. Penulisannya dipisahkan dengan tanda titik (.)
- Contoh: **package** id.ac.ukdw.www;
- id
  - ac
    - ukdw
      - www

Hirarki Package

- Java sepertinya menganggap Package seperti **hirarki folder**, tapi kenyataannya **tidak**!
- import java.awt.*;
  - Kita import semua class dalam java.awt.*
- Tapi kita tidak otomatis mengimpor class pada java.awt.color.*
- Harus dilakukan manual:
  - import java.awt.color.*;

Static Import

- Jika kita memiliki variabel dan method static
  - public static final double PI 3.141592653589793
  - public static double cos(double a)
- Dapat diakses sbb:
  - double r = **Math.cos**(**Math.PI** * theta);
- Atau diimport:
  - import **static** java.lang.Math.PI; atau
  - import **static** java.lang.Math.*;

Package : Contoh

- Misalkan folder awal kita adalah di **D:\Program**
- Kita buat class **Dosen** dan **Mahasiswa**
- Class Dosen:

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

    public Dosen(String nik, String nama) {
        this.nik = nik;
        this.nama = nama;
    }
    public void cetakDosen() {
        System.out.println(this.nik+" ("+this.nama+")");
    }
}
```

Package : Contoh

- Class Mahasiswa:

```
public class Mahasiswa {
    private String nim;
    private String nama;
        public Mahasiswa(String nim, String nama)  {
        this.nim = nim;
        this.nama = nama;
    }
    public void cetakMhs()  {
        System.out.println(this.nim+" ("+this.nama+")");
    }
}
```

Package : Contoh

- Kemudian kita buat class **CobaMhs** sebagai class launcher yang isinya demikian:

```
class CobaMhs
{
   public static void main(String[] args)
   {
       Mahasiswa baru;
       baru = new Mahasiswa("22002529",“Antonius");
   }
}
```

- Kemudian kalau sudah berhasil, kita buat folder **CivitasAkademika** dan masukkan Class Mahasiswa dan Dosen ke dalamnya.

Package : Contoh

- Lalu kita tambahkan syntax: **`package CivitasAkademika;`** ke class Dosen dan Mahasiswa.
- Dengan begitu class Dosen dan Mahasiswa menjadi satu package, yaitu package **CivitasAkademika**.
- Untuk menggunakannya di class **CobaMhs** lagi, kita harus **mengimpor** package **CivitasAkademika** terlebih dahulu.

Penggunaan

- Import

```
import CivitasAkademika.*;
class CobaMhs
{
   public static void main(String[] args)
   {
       Mahasiswa baru;
       baru = new Mahasiswa(“12345678",“Anton");
   }
}
```

- Lalu kompile ulang dan jalankan!

Kegunaan Package

- Pengelompokan class sejenis dan berelasi
- Standarisasi penamaan vendor pembuat package tersebut
- Mempermudah dalam penggunaan kelas-kelas karena packagenya sudah teroganisir
- Menghindari name confict:
  - **`Rectangle r1`** dan **`graphics.Rectagle r2`**
- Membantu pengembangan JAR file
  - Executablenya Java

Class Relationship

- OOP mengambil realita dari kehidupan sehari-hari
- Obyek-obyek di sekitar kita memiliki hubungan **relasi** tertentu
- **Relationship**: koneksi logis, hubungan antar obyek atau kelas

Class Relationship

- **Multiplicity**: karakteristik dari relasi yang terjadi
- Level class relationship
  - Instance Level:
    - relasi terjadi pada **object-object** yang terbentuk
  - Class Level:
    - relasi yang terjadi pada **class-class** yang dibuat
  - General Level:
    - relasi yang terjadi pada **level class** maupun **object**

Multiplicity

- Menunjukkan **jumlah / kardinalitas** hubungan antar kelas

- Menunjukkan apakah relasi tersebut bersifat **opsional** atau **mandatory**
  - Mandatory = wajib

Multiplicity

Relasi	
0..1	0 atau 1
1	Tepat 1
0..*	0 atau lebih
1..*	1 atau lebih
*	Tidak tentu jumlahnya

Contoh Multiplicity

- Antara kelas Buku dan kelas Halaman (Page)
  - Multiplicitynya : Buku 1 -- Halaman 1 ... *
  - Artinya Buku berjumlah min 1
  - Halaman berjumlah minimal 1 – tak terhingga
- Berarti sifatnya **Mandatory**

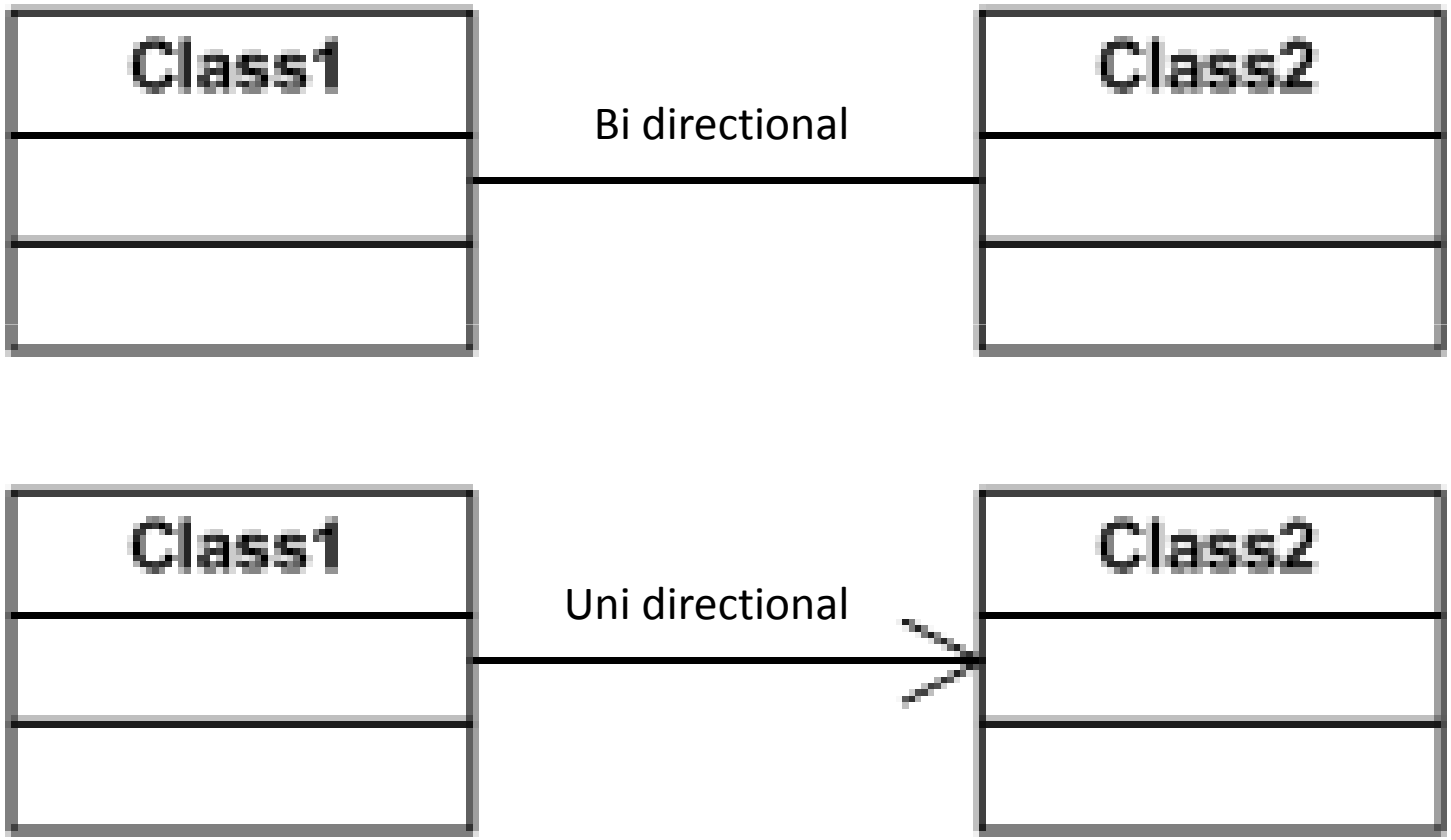
Asosiasi

- Relasi yang terjadi pada class-class dimana salah satu instance dari class tersebut **memanggil/mengaktifkan** instance dari class lainnya

- Relasi struktural, menunjukkan **penggunaan** suatu class di class lainnya

- Asosiasi bisa **uni-directional** (satu arah) atau **bi-directional** (dua arah)

Asosiasi


Class1
Bi directional
Class2
Class1
Uni directional
Class2

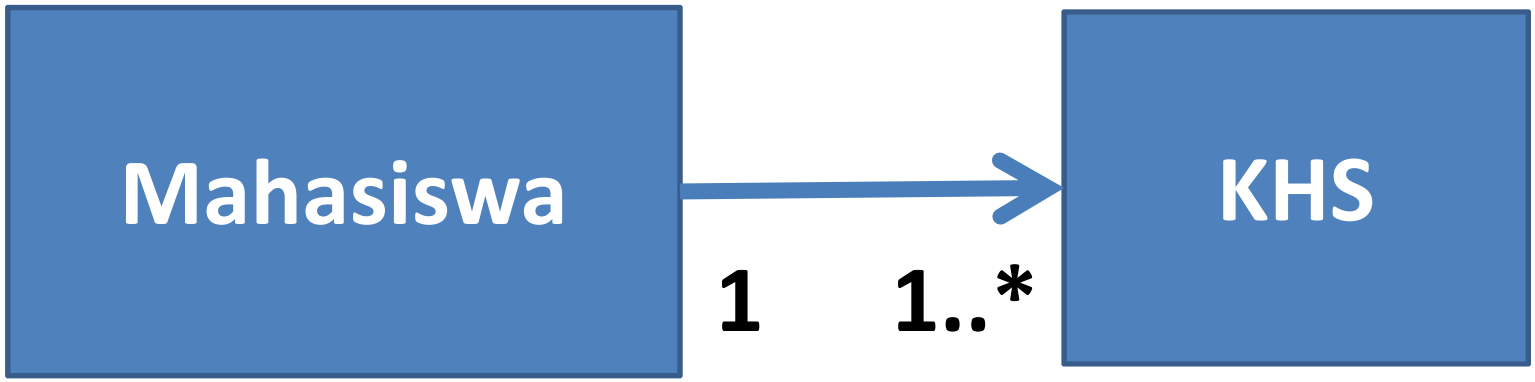
Asosiasi

- Contoh:

```
public class Mahasiswa {
        private KHS[] khs;
        void printKHS() {
                ...
        }
        ...
}
```

- Satu mahasiswa memiliki **1 atau lebih** KHS

Asosiasi



Komposisi

- Menyusun object-object **sederhana** menjadi suatu object yang lebih **kompleks**
- “Has-a” relationship
- Contoh: roda, gearbox, mesin, jok, kemudi, bagasi membentuk sebuah **mobil**
- “A car has a gearbox”

Komposisi

- Relasi komposisi ditunjukkan dengan garis penghubung dengan bentuk **diamond** berwarna **hitam** di ujungnya.

Komposisi

- Contoh:

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

      Roda(String m, int r) {
            this.merk = m;
            this.ring = r;
      }
}
```

Komposisi

- Contoh:

```
class Mobil {
    Roda rodaMbl; //Mobil berisi class Roda
    String warna;

     public void setWarna(String warnanya) {
          this.warna = warnanya;
     }
     public void setRoda(String m, int r) {
           rodaMbl = new Roda(m, r);
     }
}
```

Komposisi

- Contoh:

```
public class KendaraanKita {
   public static void main(String[] args) {
      Mobil mb = new Mobil();
      mb.setWarna("Merah");
      mb.setRoda("Bridgestone",15);

      System.out.println("Warna:"+mb.warna);
      System.out.println("Roda: "+mb.rodaMbl.merk);
      System.out.println("Ukuran: "+mb.rodaMbl.ring);
   }
}
```

Agregasi

- Agregasi *mirip* dengan komposisi
- Perbedaan pada **kekuatan** keterikatan antara object yang terbentuk dengan object-object yang menyusunnya
  - Agregasi memiliki relasi yang **kurang kuat** dibandingkan dengan Komposisi
- Relasi agregasi ditandai dengan garis penghubung dengan bentuk **diamond** berwarna **putih** di ujungnya

Agregasi

- Contoh:

  Dosen-dosen berkumpul membentuk suatu program studi

  Program studi-program studi dikumpulkan menjadi fakultas

  Fakultas-fakultas dikumpulkan menjadi universitas

Agregasi

Agregasi

- Pemain Bola Dengan Klub Bola
  - Jika klub bola **tidak ada**, pemain bola **tetap ada**

```
class PemainBola {
  private String nama;
  private String alamat;
  public PemainBola ( String nama, String alamat)
  {
    this.nama=nama;
    this.alamat=alamat;
  }
}
```

```
public class KlubBola
{
  private String nama;
  private PemainBola pemainno1;
  public KlubBola(String nama){ this.nama=nama; }
  public void setPemainSatu(PemainBola pemain)
  {
    this.pemainno1=pemain;
  }

 public static void main(String args[])
 {
  KlubBola juventus = new KlubBola("Juventus");
  PemainBola orang1 = new PemainBola("Del Piero","Sukabirus");
  juventus.setPemainSatu(orang1);
 }
}
```

Agregasi vs Komposisi

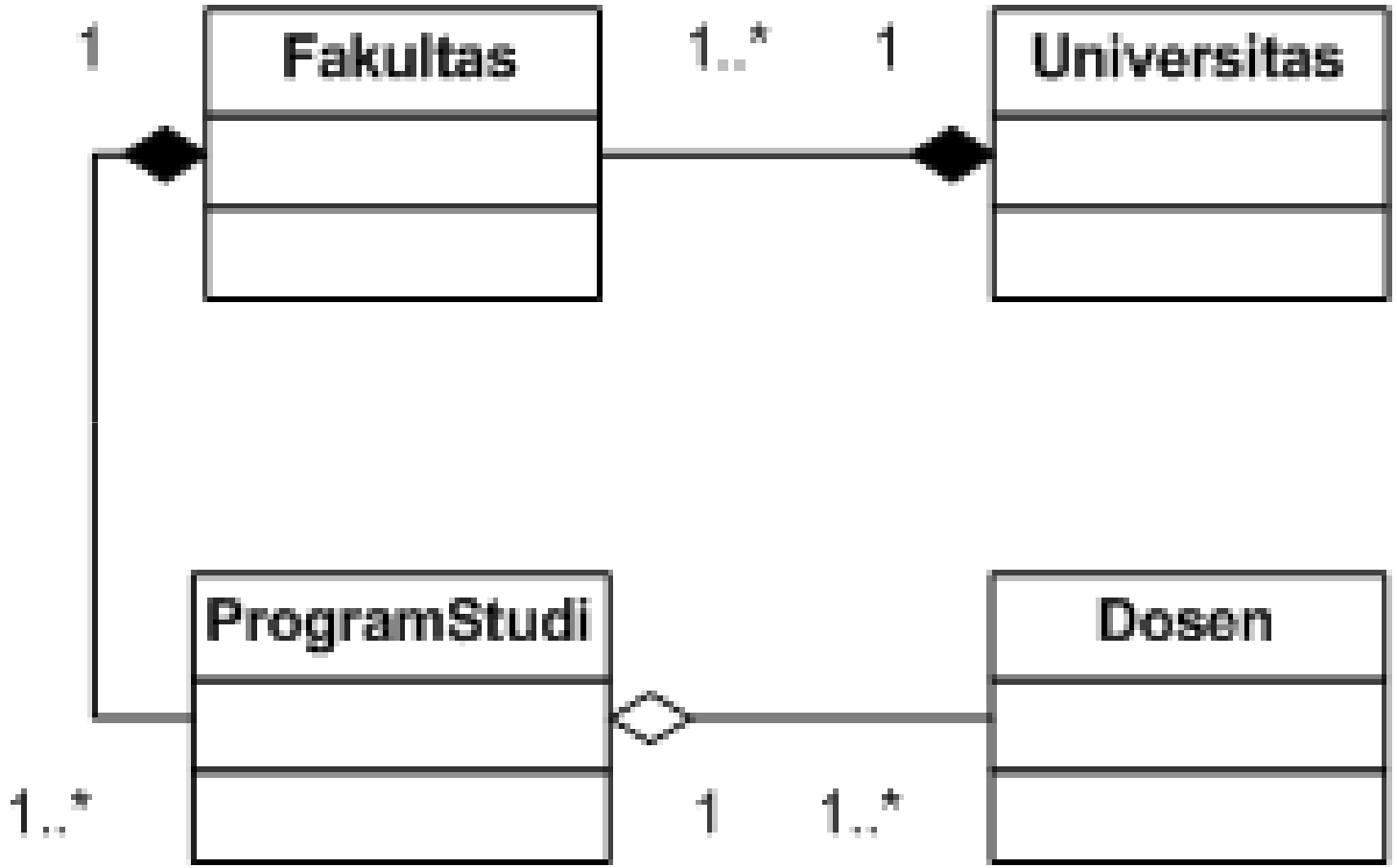
- Agregasi **tidak ada** kepemilikan.
  - Bila object yang dibentuk hilang maka object-object penyusunnya akan tetap ada

- Komposisi **ada** kepemilikan.
  - Bila object yang dibentuk hilang maka object-object penyusunnya juga akan hilang

Agregasi vs Komposisi

- Universitas terdiri dari 1 atau lebih fakultas
- Fakultas terdiri dari 1 atau lebih Program Studi
- Program Studi memiliki 1 atau lebih Dosen

Agregasi vs Komposisi



Bila Universitas ditutup maka Fakultas dan Program Studi akan hilang, tapi Dosen tidak.
Bila suatu Fakultas ditutup maka program studi-program studi di bawahnya juga akan hilang

Contoh

- Suatu lingkaran memiliki 1 titik pusat
- Merupakan agregasi atau komposisi ?

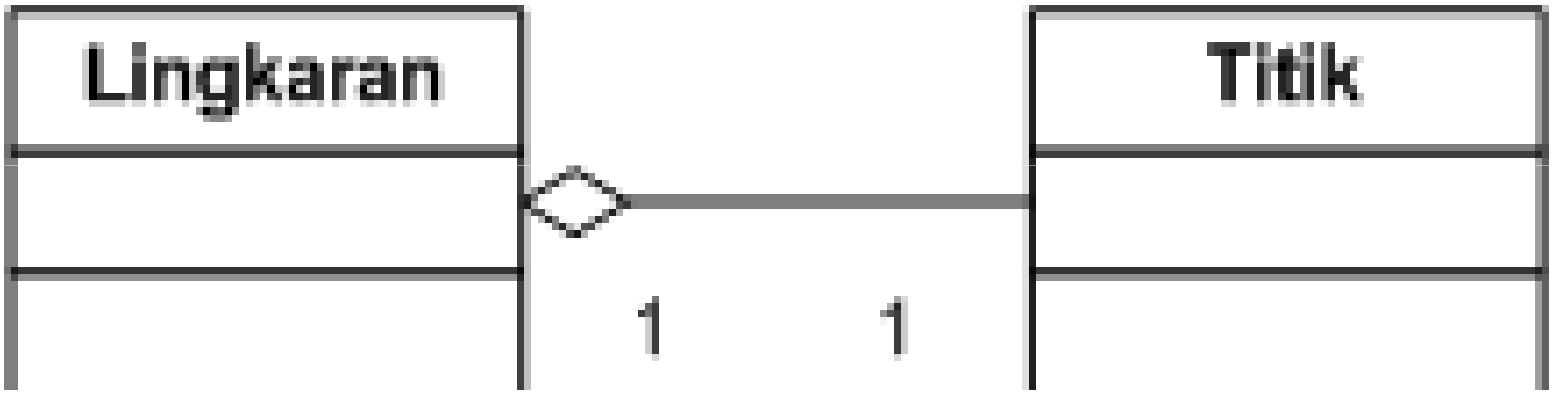
Contoh

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

    public void setKoordinat(int a, int b) {
        x = a;
        y = b;
    }
    //… (bagian selanjutnya dihilangkan)
}
```

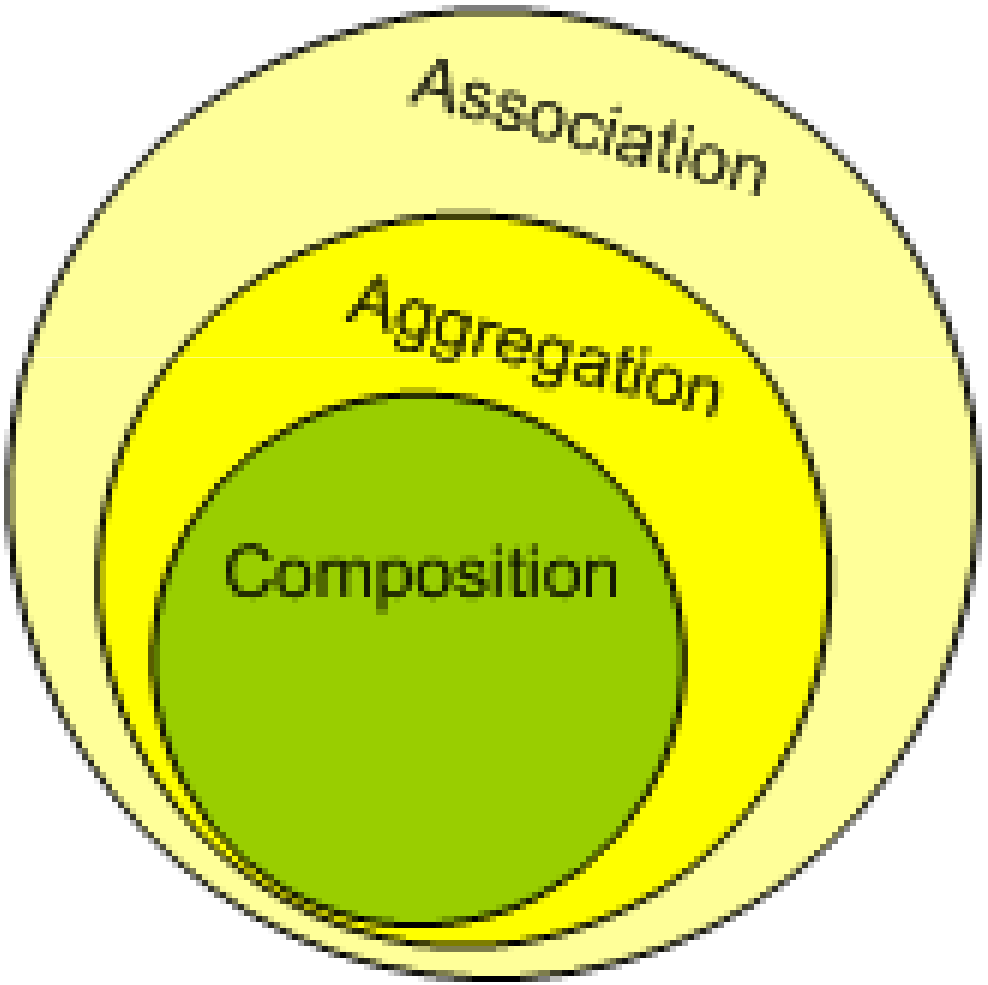
Contoh

```
public class Lingkaran {
    private int radius;

    private Titik titikPusat;
    …//bagian selanjutnya dihilangkan

}
```

Contoh



Asosiasi, Agregasi dan Komposisi



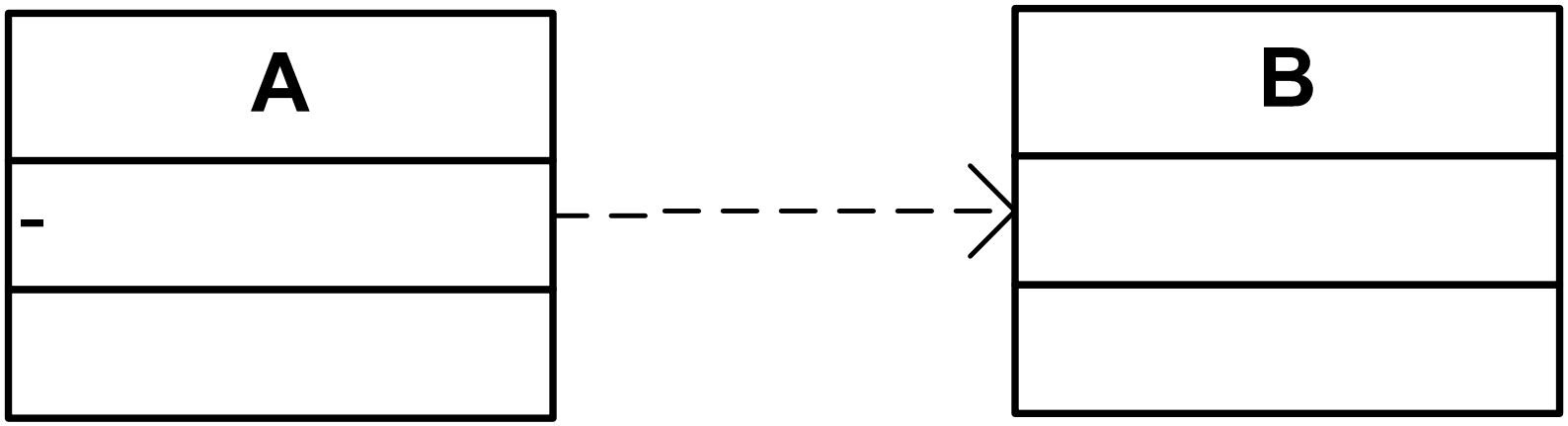
Dependency

- Relasi yang menggambarkan **ketergantungan** suatu class pada class lainnya

- Contoh: class A memiliki dependency pada class B. Bila class B berubah maka class A juga harus diubah

- Relasi dependency digambarkan dengan garis **putus-putus**

Dependency



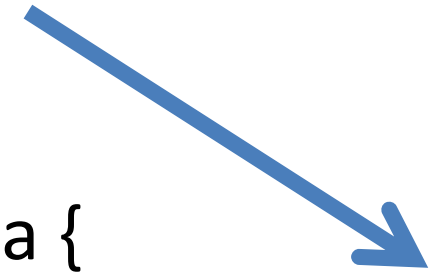
Dependency

- Ada 3 bentuk dependency
  - Penggunaan class B sebagai **parameter** pada fungsi di class A
  - Penggunaan class B sebagai **nilai kembalian** (return value) pada fungsi di class A
  - Penggunaan class B sebagai **variabel lokal** pada fungsi di class A

Dependency

- Penggunaan class B sebagai **parameter** pada fungsi di class A

```
class KHS { … }

class mahasiswa {
  float hitungIPKSemester(KHS khs, int sem) {
    …
  }
```



Dependency

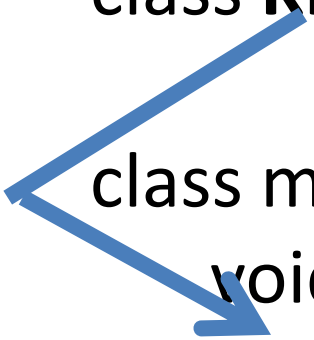
- Penggunaan class B sebagai **nilai kembalian** pada fungsi di class A

```
class KHS { … }

class mahasiswa {
    KHS getKHS(int sem) { … }
}
```

Dependency

- Penggunaan class B sebagai variabel **lokal** pada fungsi di class A

```
class KRS { … }

class mahasiswa {
    void susunKRS {
        KRS krs = new KRS(5);
        …
    }
```



NEXT

- Array and Collections