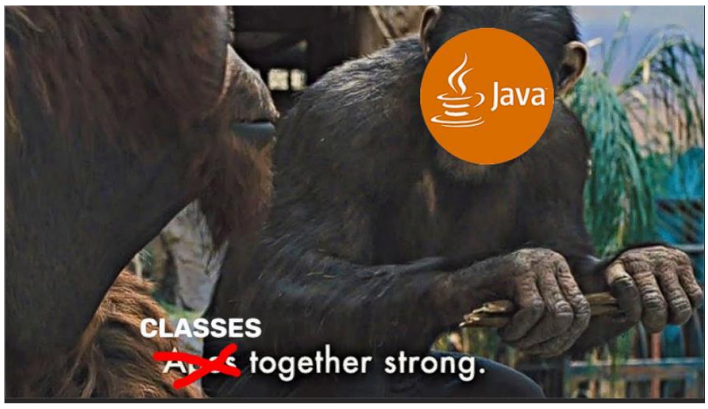
Praktikum PBO 24/25

Modul 4
Relasi

Program Studi Informatika
Universitas Atma Jaya Yogyakarta

Kevin Philips Tanamas

RELASI



Tujuan

Setelah mempelajari modul ini, praktikan diharapkan mampu:

- Memahami konsep relasi antar kelas dalam pemrograman berorientasi objek (PBO).
- Mengimplementasikan relasi antar kelas dalam bahasa pemrograman Java.

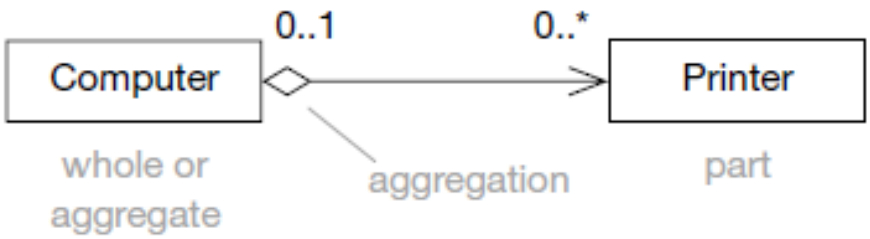
Pendahuluan

Ketika mendengar kata Relasi, maka yang akan terbayang adalah tentang **hubungan**, misalnya relasi antar sesama mahasiswa atau antara mahasiswa dengan dosen, dan seterusnya. Begitu juga dalam konsep PBO, relasi di sini berkaitan tentang bagaimana setiap **kelas-kelas yang ada saling berhubungan/berelasi untuk membangun sebuah sistem yang utuh**.

Jenis-Jenis Relasi

Dalam konsep PBO sendiri, ada beberapa jenis relasi yang dapat terjadi antar kelas, yakni:

1. **Pewarisan (*Inheritance*)**
   Untuk relasi ini tidak akan dijelaskan lebih lanjut karena sudah dibahas di **Modul Pewarisan**.
2. **Agregasi**
   Agregasi adalah hubungan **"HAS-A" (memiliki)**, di mana suatu objek tersusun dari objek lain. Relasi ini bersifat **lemah** (*weak relationship*), yang berarti objek penyusunnya **tetap dapat berdiri sendiri** meskipun objek agregatnya dihapus.
   Contoh Relasi Agregasi:
   - **Komputer** (*Whole/Aggregate*) dengan **Printer** (*Part*):
     Komputer memiliki Printer, apabila komputer dihilangkan, printer masih bisa berdiri sendiri (tetap ada).



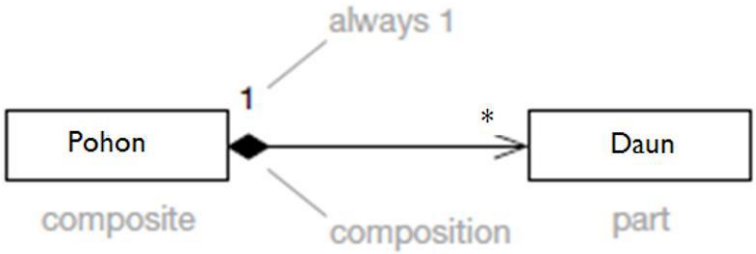
- **Mobil** (*Whole/Aggregate*) dengan **Mesin** (*Part*)
  Mobil memiliki mesin, apabila mobil dihilangkan, mesin masih bisa berdiri sendiri (tetap ada).

Relasi agregasi ditandai dengan simbol **diamond kosong** dan multiplicity yang **bisa 0** pada sisi objek agregatnya (*whole*).

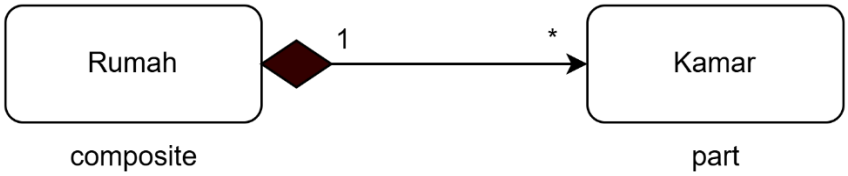
3. **Komposisi**
   Komposisi adalah hubungan **"HAS-A" (memiliki)** yang **kuat** (*strong relationship*) antara dua objek, di mana:
   - Objek penyusun (*part*) **tidak bisa ada sendiri** tanpa objek utama (*whole*).
   - Jika objek utama dihapus, maka objek penyusun juga **ikut hilang** (*dependent*).

   Contoh Relasi Komposisi:

   - **Pohon** (*Whole/Composite*) dengan **Daun** (*Part*)
     Pohon memiliki banyak daun, dan jika pohon ditebang, daunnya juga akan mati dan tidak bisa ada sendiri.



   - **Rumah** (*Whole/Composite*) dengan **Kamar** (*Part*)
     Rumah memiliki beberapa kamar, jika rumah dihancurkan, kamar juga ikut hilang karena bagian dari rumah.



Relasi ini ditandai dengan simbol **diamond penuh** dan multiplicity yang **harus 1** pada objek kompositnya.

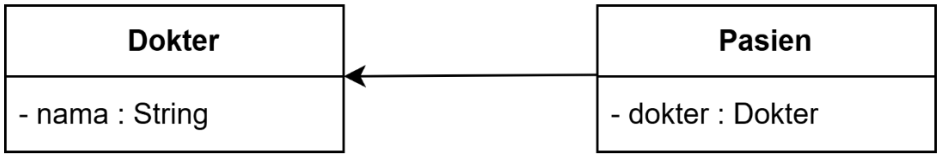
4. **Asosiasi**

Asosiasi dalam PBO merujuk pada **hubungan antara objek-objek** dari kelas yang berbeda, di mana satu objek dapat berinteraksi dengan objek lainnya. Konsep ini menggambarkan cara kelas dan objek saling berhubungan satu sama lain. Asosiasi dapat dikelompokkan ke dalam beberapa jenis, seperti asosiasi **satu arah** dan asosiasi **dua arah**, serta dapat melibatkan berbagai jenis hubungan multiplicity, seperti **one-to-one**, **one-to-many**, atau **many-to-many**.

Berikut merupakan jenis-jenis relasi asosiasi:

- **Asosiasi Satu Arah** (*Unidirectional Association*)

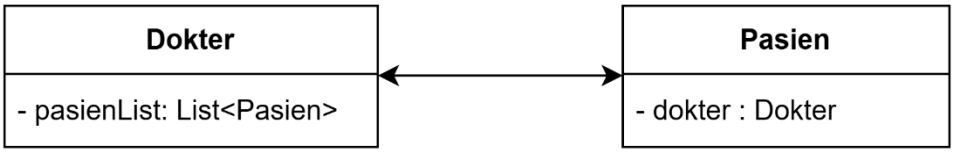
  Dalam asosiasi satu arah, satu kelas dapat mengakses objek dari kelas lain, tetapi sebaliknya tidak terjadi. Ini berarti, objek dari kelas yang satu hanya dapat merujuk atau berinteraksi dengan objek dari kelas lainnya tanpa sebaliknya. Contoh Asosiasi Satu Arah:



  Pada contoh di atas, kelas Pasien memiliki referensi ke objek Dokter. Namun, **kelas Dokter tidak memiliki referensi ke objek Pasien**, sehingga ini adalah asosiasi satu arah.

- **Asosiasi Dua Arah** (*Bidirectional Association*)

  Dalam asosiasi dua arah, kedua kelas dapat **saling merujuk** atau berinteraksi satu sama lain. Contoh Asosiasi Dua Arah:



  Pada contoh ini, baik objek Dokter maupun Pasien **saling mengetahui** satu sama lain. Jika sebuah Pasien ditambahkan ke Dokter, maka Dokter akan memiliki referensi ke daftar pasien yang ditangani, dan objek Pasien tahu siapa dokter yang menangani mereka.

5. **Dependensi**
   Dependensi adalah hubungan di mana **satu kelas bergantung pada kelas lain** untuk berfungsi dengan benar. Dalam konteks ini, perubahan pada kelas yang menyediakan layanan bisa **berdampak** pada kelas yang menggunakannya. Dependensi menunjukkan hubungan ketergantungan sementara antara suatu kelas (A) dengan kelas lain (B). Ketika kelas A mengalami perubahan, kelas B yang bergantung padanya dapat terpengaruh, tetapi hubungan ini **tidak bersifat permanen**.

   Beberapa penerapan dependensi dalam kode:
   - Penggunaan kelas B sebagai **parameter** dalam metode kelas A
     Contoh: *void cetakLaporan(Laporan laporan)*
   - Penggunaan kelas B sebagai **nilai balikan** dalam metode kelas A
     Contoh: *Laporan ambilLaporanTerbaru()*
   - Penggunaan kelas B sebagai **variabel lokal** dalam metode kelas A
     Contoh: *void prosesLaporan() { Laporan laporan = new Laporan(); }*

Perbedaan Agregasi vs Komposisi vs Asosiasi

Aspek	Asosiasi	Agregasi	Komposisi
Ketergantungan	Tidak bergantung	Lemah	Kuat
Whole-Part	Tidak ada konsep "Whole-Part"	Ada, tetapi part bisa hidup sendiri	Ada, dan part tidak bisa hidup sendiri
Jika salah satu objek dihapus?	Tidak ada efek ke objek lain	Part tetap ada	Part ikut terhapus
Contoh	Dokter ↔ Pasien	Komputer → Printer	Pohon → Daun

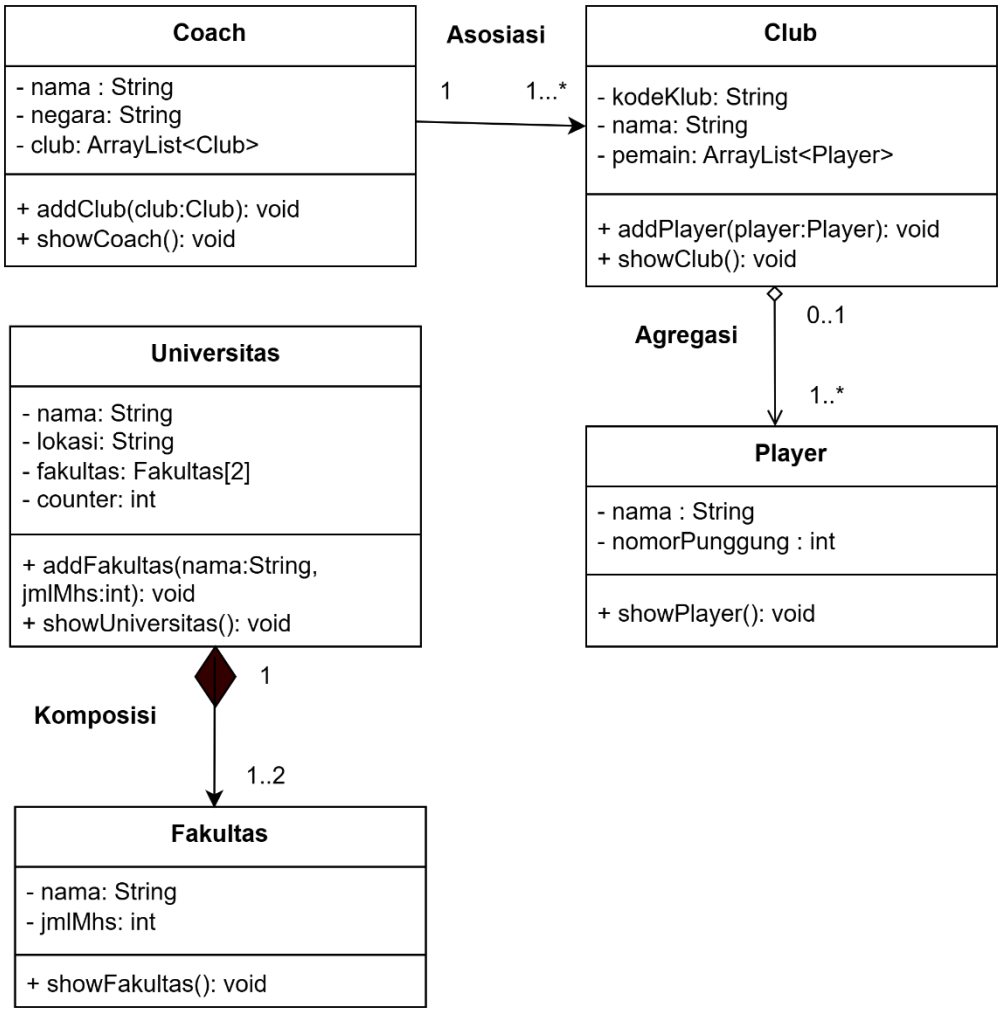
ArrayList (tambahan)

ArrayList adalah implementasi dari List yang dapat digunakan untuk menyimpan kumpulan objek secara **dinamis** (tidak dibatasi ukurannya). Dalam pemrograman berorientasi objek sering menggunakan ArrayList untuk merepresentasikan hubungan **one-to-many** atau **many-to-many** antar objek. Beberapa method yang sering digunakan dalam penggunaan ArrayList:

- add(), untuk menambah elemen.
- get(), untuk mengakses index.
- set(), untuk mengubah elemen.
- remove(), untuk menghapus elemen tertentu.
- clear(), untuk menghapus semua elemen.
- size(), untuk mengetahui ukuran ArrayList/banyak elemen.

Guided & Penjelasan

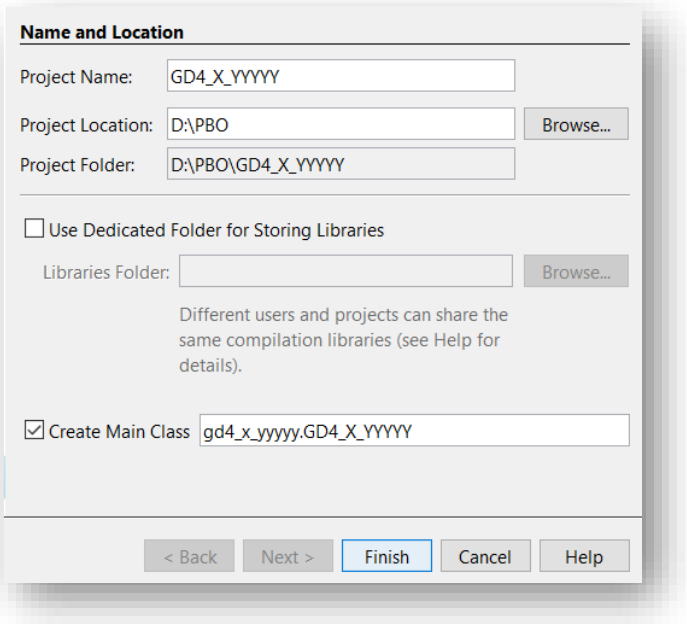
Pada Guided kali ini, kita akan melakukan implementasi pada konsep-konsep relasi diatas. Berikut adalah UML diagram dan relasi-relasi yang dimaksud:



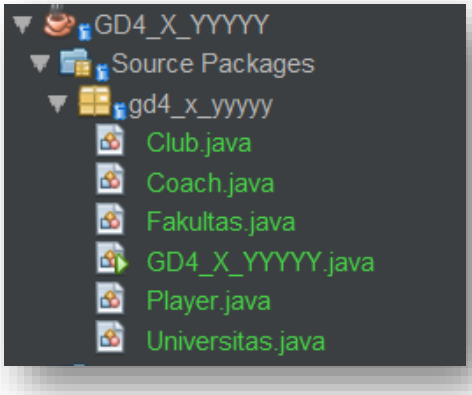
1. **Relasi Agregasi**
   Relasi agregasi terjadi antara kelas Club dengan kelas Player atau pemain. di mana kelas **Club sebagai agregat** dan kelas **Player adalah penyusunnya**. Disini kelas Club menggunakan **ArrayList** untuk menampung objek player.
2. **Relasi Asosiasi**
   Relasi asosiasi satu arah terjadi dari terhubungnya kelas Coach dengan kelas Club (Klub yang pernah dilatih oleh Coach). Asosiasi antara Coach dan Club menunjukkan bahwa **seorang Coach bisa melatih banyak klub**, tetapi **klub tetap bisa eksis tanpa Coach.**
3. **Relasi Komposisi**
   Relasi komposisi terjadi antara Universitas dan Fakultas di mana kelas Universitas sebagai **komposit** dan kelas Fakultas sebagai **penyusunnya**. Untuk relasi komposisi memiliki perbedaan dibagian method **add** nya.

Langkah-langkah pengerjaan:

1. Buatlah sebuah project baru pada Netbeans dengan nama **GD4_X_YYYYY (X= Kelas, YYYYY = 5 digit NPM terakhir).**



2. Buatlah 5 kelas baru dengan nama **Coach, Club, Player, Universitas, dan Fakultas**



3. Kita akan mulai pada relasi **Agregasi dan Asosiasi** terlebih dahulu. Oleh karena itu, silahkan isi **kelas Player** sebagai objek penyusun dari **kelas Club** dengan kode sebagai berikut:

```java
package gd4_x_yyyyy;

/**
 *
 * @author asus
 */
public class Player {
    private String nama;
    private int nomorPunggung;

    public Player(String nama, int nomorPunggung){
        this.nama = nama;
        this.nomorPunggung = nomorPunggung;
    }

    public void showPlayer(){
        System.out.println("\t\tNama Pemain      : "+nama);
        System.out.println("\t\tNomor Punggung   : "+nomorPunggung);
    }
}
```

4. Kemudian kita akan mengisi kelas **Club sebagai objek agregat** dari kelas Player dan jangan lupa melakukan **import ArrayList dan List** dengan kode sebagai berikut:

```java
package gd4_x_yyyyy;

import java.util.ArrayList;

/**
 *
 * @author asus
 */
public class Club {
    private String kodeKlub, nama;
    private ArrayList<Player> player; // Relasi one-to-many: Satu club memiliki banyak pemain

    public Club(String kodeKlub, String nama){
        this.kodeKlub = kodeKlub;
        this.nama = nama;
        this.player = new ArrayList<>(); // Pendefinisian Array List
    }

    public void addPlayer(Player p){
        player.add(p);
    }

    public void showClub(){
        System.out.println("\tKode Klub      : "+kodeKlub);
        System.out.println("\tNama Klub      : "+nama);
        System.out.println("\tJumlah Pemain : "+player.size());
        for(int i =0; i<player.size(); i++){
            System.out.println("\n\t\t----- Tampil Pemain -------");
            System.out.println("\t\tPemain ke - : "+(i+1));
            player.get(i).showPlayer();
        }
    }
}
```

Dari kode di sini, silahkan cermati pada method **addPlayer**. Tampak bahwa method ini menyimpan objek Player yang dibuat **DI LUAR** fungsi tersebut. Objek yang dibuat di luar tadi, akan di-passing di parameter **addPlayer(Player P).**

Hal ini selaras dengan yang dimaksud oleh relasi Agregasi, di mana objek penyusun dari Club yaitu Player, bersifat **mandiri** karena jika objek Club ini dihapus pun, objek Player yang kita buat di luar fungsi addPlayer tadi tidak berdampak atau ikut terhapus. Ini yang menyebabkan hubungan mereka lemah, karena independensi dari objek penyusunnya.

5. Selanjutnya, silahkan lengkapi kelas **Coach** dengan kode sebagai berikut:

```
package gd4_x_yyyyy;

import java.util.ArrayList;

/**
 *
 * @author asus
 */
public class Coach {
    private String nama, negara;
    private ArrayList<Club> club; // Relasi one-to-many: Satu pelatih dapat melatih beberapa klub

    public Coach(String nama, String negara){
        this.nama = nama;
        this.negara = negara;
        this.club = new ArrayList<>(); // Menginisialisasi ArrayList klub yang bisa dilatih oleh coach
    }

    public void addClub(Club c){
        if(c != null){ // Pengecekan null pada inputan club
            club.add(c); // Method untuk menambahkan objek ke list
        }
    }

    public void showCoach(){
        System.out.println("\n------ Tampil Coach --------");
        System.out.println("Nama         : " + nama);
        System.out.println("Negara       : " + negara);
        System.out.println("Jumlah Klub : " + club.size());

        for(int i = 0; i < club.size(); i++){
            System.out.println("\n\t----- Tampil Klub -----");
            System.out.println("\tKlub ke - " + (i+1) + ":");
            club.get(i).showClub(); // Cara pemanggilan klub melalui ArrayList
        }
    }

}
```

Dari kode di sini, silahkan cermati pada method **addClub**. Tampak bahwa method ini menyimpan objek Club yang dibuat **DI LUAR** fungsi tersebut. Objek yang dibuat di luar tadi, akan di-passing di parameter **addClub(Club C).**
Jika kita perhatikan, tidak ada perbedaan sintaks dari relasi agregasi dan asosiasi:

- **Agregasi** : addPlayer(Player P).
- **Asosiasi**: addClub(Club C).

Bagian yang ditebalkan sebagai objek yang disimpan oleh masing-masing kelas tadi, sama-sama dibuat **di LUAR** fungsi yang berkaitan. Body Codenya juga tidak ada perbedaan, di mana isi array penampungnya langsung diisikan dengan objek yang ada di parameternya.

6. Relasi Agregasi dan Asosiasi telah selesai diimplementasikan. Berikutnya mari kita implementasikan relasi **Komposisi**. Berikut adalah kode untuk kelas **Fakultas** sebagai objek penyusun dari kelas **Universitas**.

```java
package gd4_x_yyyyy;

/**
 *
 * @author asus
 */
public class Fakultas {
    private String nama;
    private int jmlMhs;

    public Fakultas(String nama, int jmlMhs){
        this.nama = nama;
        this.jmlMhs = jmlMhs;
    }

    public void showFakultas(){
        System.out.println("\tNama Fakultas      : "+nama);
        System.out.println("\tJumlah Mahasiswa   : "+jmlMhs);
    }
}
```

7. Jangan lupa, tambahkan pula kode di kelas **Universitas** sebagai **komposit** dari kelas **Fakultas**.

```java
package gd4_x_yyyyy;

/**
 *
 * @author asus
 */
public class Universitas {
    private String nama, lokasi;
    private Fakultas fakultas[]; // Relasi one-to-many: Satu universitas memiliki banyak fakultas
    int counter = 0;

    public Universitas(String nama, String lokasi){
        this.nama = nama;
        this.lokasi = lokasi;
        this.fakultas = new Fakultas[2]; // Universitas dapat memiliki maksimal 2 fakultas
    }

    public void addFakultas(String nama, int jmlMhs){
        fakultas[counter] = new Fakultas(nama, jmlMhs); // Menambahkan fakultas ke dalam array
        counter++;
    }

    public void showUniversitas(){
        System.out.println("\n------ Tampil Universitas ------");
        System.out.println("Nama Universitas     : "+nama);
        System.out.println("Lokasi Universitas   : "+lokasi);
        System.out.println("Jumlah Fakultas      : "+counter);
        System.out.println("\n\t----- Tampil Fakultas -------");
        for(int i =0; i<counter; i++){
            System.out.println("\n\tData Fakultas ke - : "+(i+1));
            fakultas[i].showFakultas();
        }
    }
}
```

Terdapat perbedaan pada kode relasi **komposisi**, yaitu pembuatan objek Fakultas dilakukan **DI DALAM** method **addFakultas**. Hal ini dilakukan karena ketika Universitas tidak ada, maka fakultas juga akan ikut hilang.

8. Kita sudah menyelesaikan semua kelas yang diperlukan. Saatnya kita melanjutkan ke **main class** kita yaitu **GD4_X_YYYYY.java**. Silahkan isikan kode berikut untuk main class project ini:

```
public class GD4_X_YYYYY {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("\n----- GUIDED MODUL 4 RELASI PBO 24/25 -------");

        System.out.println("\n[1]. Relasi Agregasi dan Asosiasi");
        Player player1 = new Player("Cristiano Ronaldo", 7);
        Player player2 = new Player("Toni Kroos", 8);
        Player player3 = new Player("Andre Onana", 24);
        Player player4 = new Player("Marcus Rashford", 10);

        Club club1 = new Club("RMA", "Real Madrid");
        Club club2 = new Club("MUN", "Manchester United");
        Coach coach1 = new Coach("Jose Mourinho", "Portugal");

        coach1.addClub(club1);
        coach1.addClub(club2);

        club1.addPlayer(player1);
        club1.addPlayer(player2);
        club2.addPlayer(player3);
        club2.addPlayer(player4);
        coach1.showCoach();

        System.out.println("\n[2]. Relasi Komposisi");
        Universitas u1 = new Universitas("Universitas Atma Jaya Yogyakarta", "Jl.Babarsari no 44");
        u1.addFakultas("Fakultas Teknologi Industri", 1000);
        u1.addFakultas("Fakultas Tekno Biologi", 500);
        u1.showUniversitas();
    }

}
```

Output Guided

Selamat teman-teman sudah menyelesaikan guided modul relasi! Silahkan cek outputnya apakah sudah sesuai dengan gambar berikut ini:

```
run:

----- GUIDED MODUL 4 RELASI PBO 24/25 -------

[1]. Relasi Agregasi dan Asosiasi

------ Tampil Coach --------
Nama         : Jose Mourinho
Negara       : Portugal
Jumlah Klub : 2

         ----- Tampil Klub -----
         Klub ke - 1:
         Kode Klub      : RMA
         Nama Klub      : Real Madrid
         Jumlah Pemain : 2

                  ----- Tampil Pemain -------
                  Pemain ke - : 1
                  Nama Pemain      : Cristiano Ronaldo
                  Nomor Punggung   : 7

                  ----- Tampil Pemain -------
                  Pemain ke - : 2
                  Nama Pemain      : Toni Kroos
                  Nomor Punggung   : 8

         ----- Tampil Klub -----
         Klub ke - 2:
         Kode Klub      : MUN
         Nama Klub      : Manchester United
         Jumlah Pemain : 2

                  ----- Tampil Pemain -------
                  Pemain ke - : 1
                  Nama Pemain      : Andre Onana
                  Nomor Punggung   : 24

                  ----- Tampil Pemain -------
                  Pemain ke - : 2
                  Nama Pemain      : Marcus Rashford
                  Nomor Punggung   : 10

[2]. Relasi Komposisi

------ Tampil Universitas ------
Nama Universitas     : Universitas Atma Jaya Yogyakarta
Lokasi Universitas   : Jl.Babarsari no 44
Jumlah Fakultas      : 2

         ----- Tampil Fakultas -------

         Data Fakultas ke - : 1
         Nama Fakultas      : Fakultas Teknologi Industri
         Jumlah Mahasiswa   : 1000

         Data Fakultas ke - : 2
         Nama Fakultas      : Fakultas Tekno Biologi
         Jumlah Mahasiswa   : 500
BUILD SUCCESSFUL (total time: 0 seconds)
```

Format Pengumpulan

Kumpulkan Guided dengan melakukan **zip** pada project yang telah dibuat, dengan format penamaan dibawah ini:

GD4_X_YYYYY.zip

X = Kelas

YYYYY = 5 digit NPM

Hint

- Silahkan pelajari bagaimana relasi ini berhubungan untuk membuat sebuah sistem, misalnya agregasi bersamaan dengan komposisi.
- Pelajari bagaimana **kardinalitas** antara relasi itu, apakah 0..1, 1..*, dan sebagainya.
- Pelajari juga bagaimana cara menggunakan **ArrayList**, serta method-method nya.

Kontak Asisten

Jika ada yang ingin ditanyakan, atau berdiskusi, silahkan kontak ke

- Teams: 220711789@students.uajy.ac.id
- Whatsapp: 081312648799

Sumber Pembelajaran

https://www.w3schools.blog/java-association-aggregation-and-composition

https://www.youtube.com/watch?v=KzexzfQRxDE

https://www.w3schools.com/java/java_arraylist.asp

--- Selamat Belajar, GBU ---