5️⃣

Modul 5 - Agregasi dan komposisi

Durasi Modul = 2-5 menit

Capaian pembelajaran:

- Memahami konsep agregasi dalam pemrograman berorientasi objek
- Mengidentifikasi karakteristik dan implementasi hubungan agregasi antar kelas
- Memahami konsep komposisi dalam pemrograman berorientasi objek
- Membedakan antara agregasi dan komposisi dalam desain kelas
- Mengimplementasikan hubungan agregasi dan komposisi dalam kode program Java
- Menganalisis penggunaan agregasi dan komposisi dalam pemodelan sistem berorientasi objek

Deskripsi Pembelajaran

Dalam modul ini, kita akan mempelajari dua konsep penting dalam pemrograman berorientasi objek: agregasi dan komposisi. Kedua konsep ini merupakan jenis hubungan antar kelas yang memungkinkan kita untuk membangun struktur yang lebih kompleks dan fleksibel dalam desain perangkat lunak.

Agregasi adalah hubungan "has-a" di mana satu kelas memiliki atau menggunakan objek dari kelas lain, namun kedua kelas dapat berdiri sendiri. Kita akan mempelajari cara mengidentifikasi dan mengimplementasikan hubungan agregasi dalam kode Java.

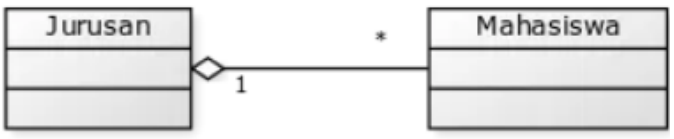
Komposisi, di sisi lain, adalah bentuk hubungan yang lebih kuat di mana satu kelas tidak hanya memiliki objek dari kelas lain, tetapi juga bertanggung jawab atas siklus hidup objek tersebut. Kita akan membahas perbedaan antara

agregasi dan komposisi, serta bagaimana menggunakan keduanya dalam pemodelan sistem berorientasi objek.

Melalui contoh kode dan diagram UML, kita akan melihat bagaimana konsep-konsep ini diterapkan dalam praktik pemrograman. Pada akhir modul ini, Anda akan dapat membedakan antara agregasi dan komposisi, serta menggunakannya secara efektif dalam desain dan implementasi program berorientasi objek.

5.1. Agregasi

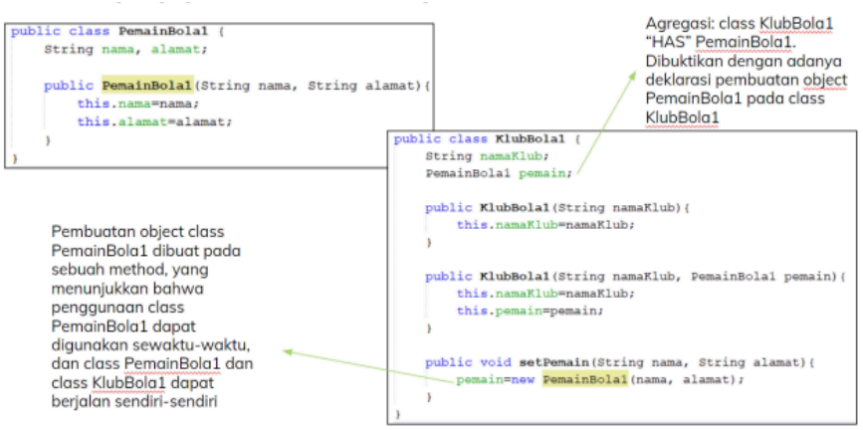
1. **Agregasi** merupakan **hubungan** antara **dua kelas** dimana kelas yang satu merupakan
   bagian dari kelas yang lain
   , namun kedua kelas ini dapat berdiri sendiri-sendiri
2. Hubungan antar class yang menyatakan hubungan **“has-a.”**
3. **Simbol** yang digunakan adalah: Diamond kosong berwarna putih yang diarahkan pada kelas yang memiliki, tidak dibubuhkan panah pada ujung yang tidak memiliki simbol diamond kosong putih *(white hollow diamond)*. Kemudian juga dibubuhkan kardinalitas seperti pada hubungan asosiasi.
4. **Simbol panah** menyatakan suatu kelas ***navigable*** terhadap kelas lain



kelas diagram dengan hubungan agregasi

📌 Jika **hubungan asosiasi** adalah saling menggunakan, di sini **hubungan yang terjadi** adalah memiliki. Dilihat pada gambar kelas diagram di atas, bahwa kelas Jurusan memiliki kelas Mahasiswa sebagai variable nya. Meskipun kelas Jurusan memiliki kelas Mahasiswa, namun kedua kelas tersebut dapat dibuat secara independen.

Penjelasan dengan menggunakan potongan kode akan bisa lebih rinci di bagian menggunakan kelas lain seperti dibawah ini




```
public class PemainBola1 {
    String nama, alamat;

    public PemainBola1(String nama, String alamat){
        this.nama=nama;
        this.alamat=alamat;
    }
}
```

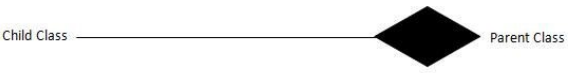
```
public class KlubBola1 {
    String namaKlub;
    PemainBola1 pemain;

    public KlubBola1(String namaKlub){
        this.namaKlub=namaKlub;
    }

    public KlubBola1(String namaKlub, PemainBola1 pemain){
        this.namaKlub=namaKlub;
        this.pemain=pemain;
    }

    public void setPemain(String nama, String alamat){
        pemain=new PemainBola1(nama, alamat);
    }
}
```

Class implementasi yang menegaskan hubungan agregasi kedua class di atas adalah sebagai berikut:

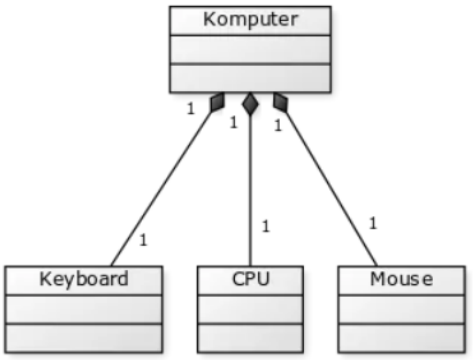

```
public class PemainBolaKlubBola1 {
    public static void main(String[] args) {
        //Cara 1
        PemainBola1 p1=new PemainBola1("David","Jl. Umban Sari");
        KlubBola1 klub1=new KlubBola1("MU", p1);

        System.out.println("Nama Klub: "+klub1.namaKlub);
        System.out.println("Nama Pemain: "+klub1.pemain.nama);
        System.out.println("Nama Pemain: "+klub1.pemain.alamat);

        //Cara 2
        KlubBola1 klub2=new KlubBola1("Chelsea");
        klub2.setPemain("George", "Jl. Paus");

        System.out.println();
        System.out.println("Nama Klub: "+klub2.namaKlub);
        System.out.println("Nama Pemain: "+klub2.pemain.nama);
        System.out.println("Nama Pemain: "+klub2.pemain.alamat);
    }
}
```

```
Nama Klub: MU
Nama Pemain: David
Nama Pemain: Jl. Umban Sari

Nama Klub: Chelsea
Nama Pemain: George
Nama Pemain: Jl. Paus
```

5.2. Komposisi



1. **Komposisi** adalah jenis hubungan ***"whole-part"*** yang **lebih kuat** di mana kelas bagian (part) secara eksplisit dimiliki oleh kelas utama (whole).
2. **Komposisi** digambarkan menggunakan **diamond hitam** yang menyatakan memiliki bagian seperti pada agregasi, tapi **lebih spesial.** Kemudian menggunakan **garis tidak putus ke kelas yang dimiliki.**
3. Objek part tidak dapat diakses “diluar” objek utuhnya, artinya objek part sifatnya private.
4. Pesan yang ditujukan kepada objek part harus dikirimkan kepada objek utuhnya, kemudian objek utuh tersebut akan meneruskan kepada objek partnya.



kelas diagram hubungan komposisi

> Pada paragraf di atas dikatakan **“memiliki bagian seperti pada agregasi, namun lebih spesial”**, **apa yang menjadikan hubungan komposisi lebih spesial** adalah karena setiap kelas tidak dapat berdiri sendiri. Seperti contoh kelas diagram di samping, **bahwa** kelas `Komputer` **tidak akan**

> **terbentuk tanpa adanya** kelas Keyboard, Mouse dan CPU. Begitu pula sebaliknya, **apabila tidak terdapat kelas Komputer**, maka **kelas kelas yang lain tidak akan berfungsi.**

Penjelasan dengan menggunakan potongan kode akan bisa lebih rinci di bagian menggunakan kelas lain seperti dibawah ini

Class `Cpu`

```
public class Cpu {
  private String nama;
  public Cpu(String nama) {
    this.nama = nama;
  }
  public String getNama() {
    return this.nama
  }
}
```

Class `Keyboard`

```
public class Keyboard {
  private String jenis;
  public Keyboard(String jenis) {
    this.jenis = jenis;
  }
  public String getJenis() {
    return this.jenis;
  }
}
```

Class `Komputer`

```
public class Komputer {
  private String merk;
  private String tipe;
  private Mouse mouse;
  private Keyboard keyboard;
  private Cpu cpu;
  public Komputer(String merk, String tipe) {
    this.merk = merk;
    this.tipe = tipe;
  }
  public addKomponen(Mouse mouse, Keyboard keyboard, Cpu cpu) {
    this.mouse = mouse;
    this.keyboard = keyboard;
    this.CPU = cpu;
  }
}
```

Class `Mouse`

```
public class Mouse {
  private String merk;
  public Mouse(String merk) {
    this.merk = merk;
  }
  public String getMerk() {
    return this.merk;
  }
}
```

Class `Main Komputer`

```
public class MainKomputer {
  public static void main(String[] args) {
    // membuat objek komputer
    Komputer komputer = new Komputer('Asus','A450CC');
    // membuat komponen nya
    komputer.addKomponen(new Mouse('Logitec'), new Keyboard('Logitec'), new Cpu('intel i7'));
  }
}
```

📌 Dapat dilihat dari file *mainKomputer.java* bahwa komponen kelas `Mouse, Keyboard, Cpu` baru dibuat setelah komputer dibuat. A**pabila objek komputer dihapus maka semua komponen juga terhapus.** Begitulah ilustrasi dari hubungan komposisi.

📚REFERENSI

Aziefah, & Nurmasari, J. (2013). Pemrograman Berorientasi Objek. Pekanbaru: Politeknik Caltex Riau.

Barclay, K., & Savage, J. (2004). Object Oriented Design with UML and Java. Elsevier.

Deitel, P., & Deitel, H. (2012). Java: How to Program (9 ed.). US: Prentice Hall.