**BAB VI**

**ASOSIASI, AGREGASI, DAN KOMPOSISI**

Tujuan Pada Pertemuan ini :

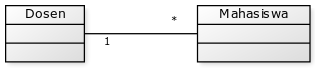
1. Memahami konsep relasi antar kelas pada pemrograman berorientasi objek.
2. Memahami konsep dan dapat menerapkan Asosiasi pada pemrograman berorientasi objek.
3. Memahami konsep dan dapat menerapkan Agregasi pada pemrograman berorientasi objek.
4. Memahami konsep dan dapat menerapkan Komposisi pada pemrograman berorientasi objek.
   1. Asosiasi
      1. Hubungan Asosiasi antar *Class*

Dalam Obyek Oriented Programming Sebagaimana tujuan dari Object Oriented Programming / Pemrograman Berorientasi Objek dalam hal ini adalah Java, yaitu kode program dibuat agar dapat mereplikasi objek-objek di dunia nyata, sehingga membuat kode program menjadi lebih mudah terbaca dan terpelihara dengan baik. Ketika berbicara mengenai dunia nyata, maka pastinya memiliki berbagai hubungan (relationship).

Asosiasi adalah hubungan yang bisa saling menggunakan di dalam sebuah kelas, dan tidak saling memiliki. Relasi asosisasi biasa di sebut “a” relationship yang berarti bahwa sebuah object “menggunakan object yang lain”. Relasi dimana semua object memiliki *lifecycle* nya sendiri dan tidak ada yang bertindak sebagai *owner*. Kaitan yang lemah Objek dapat berupa grup atau kelompok, namun objek ini tidak secara komplit tergantung satu dengan lainnya. Hubungan antara dua kelas tidak konseptual terkait tetapi dalam konteks masalah perlu memanfaatkan layanan masing-masing, hubungan yang terjadi antara kelas yang ada. Asosiasi memungkinkan suatu kelas untuk menggunakan atau mengetahui atribut atau operasi yang dimiliki oleh kelas lain. Hubungan semantic bi-directional diantara class-class. Ini bukan aliran data seperti pada pemodelan desain dan analisa terstruktur, data diperbolehkan mengalir dari kedua arah. Asosiasi diantara class-class artinya ada hubungan antara object-object pada class-class yang berhubungan. Banyaknya object yang terhubung tergantung dengan beragamnya object (multiplicity) yang ada asosiasi.

Relationship ini akan menetapkan koneksi antar objek. Dengan demikian akan menjelaskan bagaimana objek-objek saling terhubung satu sama lainnya dan bagaimana masing-masing dari objek-objek tersebut akan berperilaku., kelas – kelas yang terbentuk dapat memiliki hubungan satu dengan yang lainya, sesuai dengan kondisi hubungan satu dengan yang lainya, sesusai dengan kondisi dari kelas – kelas yang bersangkutan.

Sebagai gambaran contoh bayangkanlah sebuah mobil angkot, yang di dalamnya ada seorang supir, seorang penumpang dan satu penumpang lainnya. Ketika supir dan dua penumpang berada dalam mobil, mereka berasosiasi dan mereka semua menuju ke arah atau alamat yang sama, mereka menempati tempat yang sama, yakni di dalam mobil. Namun kaitan asosiasi ini sifatnya lemah, supir dapat menurunkan salah satu atau kedua penumpang pada arah atau alamat yang terpisah, jadi penumpang tersebut tidak lagi berasosiasi dengan objek lainnya.

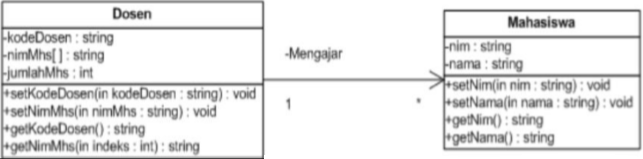


Gambar 6.1 Diagram kelas dengan hubungan asosiasi

Pada Gambar 6.1 dapat dilihat pada gambar kelas diagram di atas, hubungan asosiasi digambarkan dengan satu garis tidak putus - putus dan tidak memiliki anak panah pada kedua ujungnya. Kemudian juga dibubuhkan kardinal pada garis tersebut.

Hubungan yang bisa saling menggunakan di dalam sebuah kelas, dan tidak saling memiliki. Misal pada gambar di atas terdapat kelas Dosen dan kelas Mahasiswa, dilihat dari kardinalitasnya, satu dosen bisa memiliki banyak mahasiswa. Apabila dosen dihilangkan, mahasiswa masih bisa digunakan karena tidak saling memiliki.

Contoh dari asosiasi ada dosen dan mahasiswa, banyak mahasiswa dapat berasosiasi dengan satu dosen dan sati mahasiswa dapat berasosiasi dengan banyak dosen, jadi sebagai contoh nyata ada dosen yang tahu data mahasiswa yang di ajarkanya mahasiswa pun tahu siapa dosen yang mengajar dengan menyocokan nim mahasiswa dan kodeDosen, tipe data dari atribut link mengikuti tipe data asli pada class asal.



Gambar 6.2 Implementasi relasi asosiasi

Pada Gambar 6.2 dari dua potongan kode di atas, dapat kita ketahui bahwa kelas Dosen menggunakan variable nim mahasiswa yang terdapat pada kelas Mahasiswa. Kita dapat menghapus setiap kelas tanpa mempengrahui kelas lain. Hubungan seperti di atas adalah hubungan asosiasi.

Pada implementasi terlihat bahwa tidak ada relasi yang kuat antarakelas dosen dan kelas mahasiswa, hanya ada atribut dari kelas dosen yangserupa dengan atribut dari kelas mahasiswa yang menandakan bahwa keduakelas itu berasosiasi, yaitu atribut nimMahasiswa pada kelas dosen danatribut nim pada kelas mahasiswa.

* + 1. Penerapan Relasi Asosiasi

Class Mahasiswa :

|  |
| --- |
| * 1. class mahasiswa {   2. private String nim;   3. private String nama;   5. public void setnama (String nama) {   6. this.nama = nama;   7. }   8. public void setnim (String nim) {   9. this.nim = nim;   10. }   11. public String getnim () {   12. return this.nim;   13. }   14. public String getnama () {   15. return this.nama;   16. }   17. } |

Penjelasan :

Baris 1 : Deklarasi class mahasiswa

Baris 2 - 3 : Deklarasi variabel nim dan nama bertipe String

Baris 5 - 7 : Membuat Setter nama

Baris 8 - 10 : Membuat Setter nim

Baris 11 - 13 : Membuat Getter nim

Baris 14 - 16 : Membuat Getter nama

Class Dosen :

|  |
| --- |
| * 1. class dosen {   2. private String Kddosen;   3. private String[] nimMHS;   4. private int JmlMahasiswa;   6. public void setKddosen (String Kddosen) {   7. this.Kddosen = Kddosen;   8. }   9. public void setNimMahasiswa (String nimMhs) {   10. if (JmlMahasiswa<5) {   11. nimMHS[JmlMahasiswa] = nimMhs;   12. JmlMahasiswa++;   13. }   14. }   15. public int getJmlMahasiswa () {   16. return this.JmlMahasiswa;   17. }   18. public String getKddosen () {   19. return this.Kddosen;   20. }   21. public String getmahasiswa (int i) {   22. return (nimMHS[i]);   23. }   24. }   25. public static void main(String[] args) {   26. mahasiswa m1 = new mahasiswa();   27. m1.setnim(“18104005”);   28. m1.setnama(“Yoi”);   29. mahasiswa m2 = new mahasiswa();   30. m2.setnim(“18104006”);   31. m2.setnama(“Henken”);   32. dosen d = new dosen();   33. d.setKddosen(“CKO”);   34. d.setNimMahasiswa(m1.getnim());   35. d.setNimMahasiswa(m2.getnama());   36. System.out.println(“Kode dosen : “ + d.getKddosen());   37. System.out.println(“Mengajar mahasiswa : “);   38. for (int i = 0; i<d.getJmlMahasiswa(); i++) {   39. System.out.println(“ – “ + d.getmahasiswa(i));   40. }   41. }} |

Penjelasan :

Baris 1 : Deklarasi class dosen

Baris 2 - 4 : Deklarasi Variabel

Baris 6 - 8 : Membuat Setter Kddosen

Baris 9 - 14 : Membuat Setter NimMahasiswa

Baris 15 - 17 : Membuat Getter JmlMahasiswa

Baris 18 - 20 : Membuat Getter Kddosen

Baris 21 - 23 : Membuat Getter mahasiswa dan mengembalikkan array nimMHS

Baris 25 : Deklarasi method main

Baris 26 : Deklarasi membuat objek mahasiswa bernama m1

Baris 27 : Memanggil setNim dengan parameter

Baris 28 : Memanggil setNama dengan parameter

Baris 30 : Deklarasi membuat objek mahasiswa bernama m2

Baris 31 : Memanggil setNim dengan parameter

Baris 32 : Memanggil setNama dengan parameter

Baris 34 : Deklarasi membuat objek Dosen bernama d

Baris 36 : Memanggil setKddosen dengan parameter

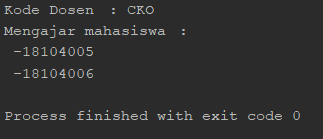
Baris 37 : Memanggil setNimMahasiswa dengan parameter yang diambil dari m1

Baris 38 : Memanggil setNimMahasiswa dengan parameter yang diambil dari m2

Baris 40 : Mencetak getter Kddosen dari objek d

Baris 41 - 44 : Perulangan untuk mencetak getter Mahasiswa dari objek d

Keluarannya :



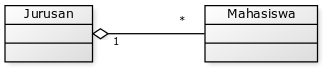
Gambar 6.3 Output Program yang menerapkan Asosiasi

* 1. Agregasi
     1. Hubungan Agregasi antar *Class*

Agregasi merupakan hubungan antara dua kelas dimana kelas yang satu merupakan bagian dari kelas yang lain namun kedua kelas ini dapat berdiri sendiri-sendiri. Simbol yang digunakan juga berupa “hollow diamond” sedangkan simbol panah menyatakan suatu class navigable terhadap class yang lain. Agregasi juga dapat diartikan sebagai suatu bentuk hubungan yang khusus dari asosiasi dimana sebuah object memiliki lifecycle nya sendiri tapi dengan kepemilikan dan class child tidak dapat memiliki class parentnya. Contoh agregasi dari mahasiswa dan jurusannya seperti berikut:

* Mahasiswa memiliki objek sendiri
* Jurusan memiliki objek sendiri
* Mahasiswa menjadi bagian dari jurusannya

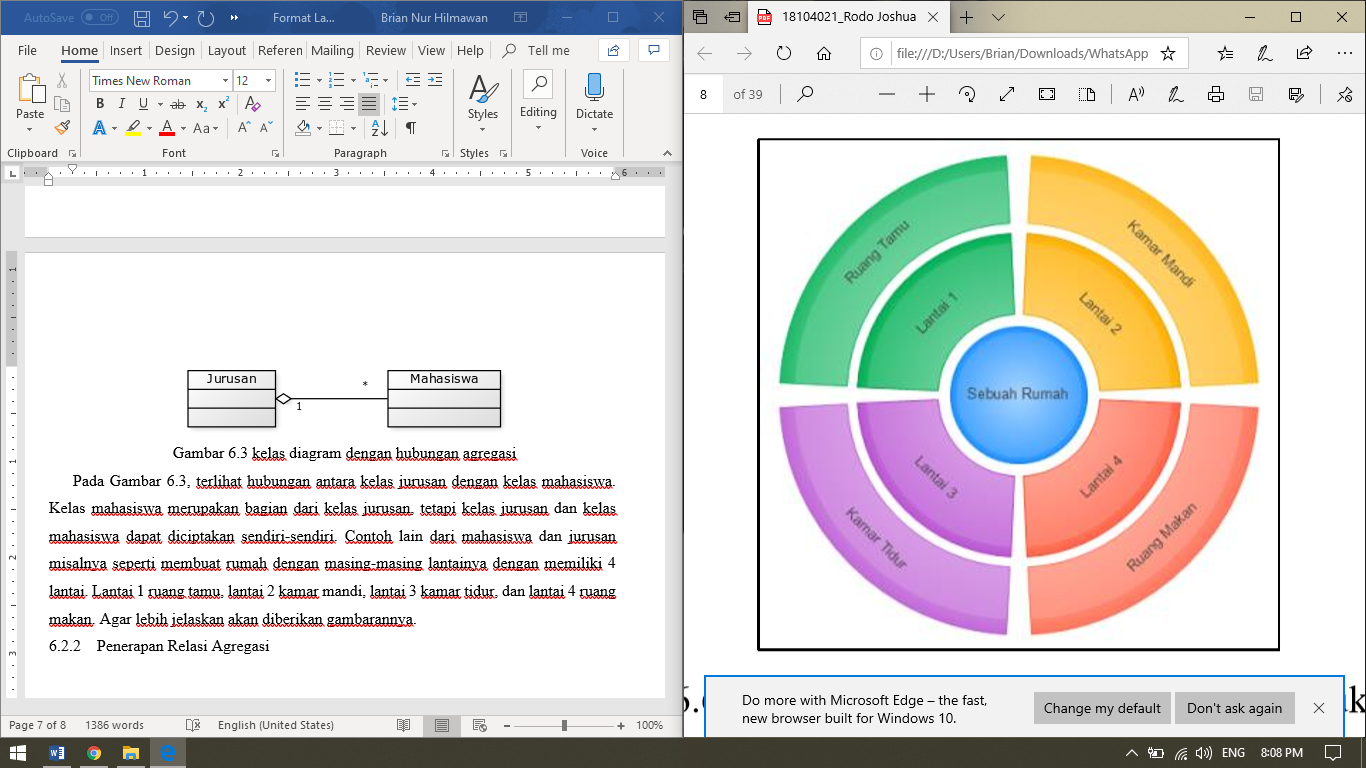
Setelah itu agar mudah dipahami disini ada contoh penggunaan agregasi agar lebih paham agregasi dalam class diagram seperti apa bentuknya. Contohnya seperti berikut ini:



Gambar 6.4 Kelas diagram dengan hubungan agregasi

Pada Gambar 6.4, terlihat hubungan antara kelas jurusan dengan kelas mahasiswa. Kelas mahasiswa merupakan bagian dari kelas jurusan, tetapi kelas jurusan dan kelas mahasiswa dapat diciptakan sendiri-sendiri. Contoh lain dari mahasiswa dan jurusan misalnya seperti membuat rumah dengan masing-masing lantainya dengan memiliki 4 lantai. Lantai 1 ruang tamu, lantai 2 kamar mandi, lantai 3 kamar tidur, dan lantai 4 ruang makan. Agar lebih jelaskan akan diberikan gambarannya.

Sehingga seperti pada gambar berikut :



Gambar 6.5 Contoh hubungan agregasi dalam bentuk berbeda

Gambar 6.5 contoh hubungan agregasi dalam bentuk berbeda Gambar 6.4 menjelaskan bahwa hubungan agregasi dapat dipahami seperti gambar di atas. Dimana gambar tersebut menjelaskan bahwa rumah memiliki beberapa bagian lantai dimana dimana setiap bagian dibuat sendiri-sendiri. Jadi, jika salah satu lantai dihilangkan tidak akan mempengaruhi bangunan karna objek lantai dibuat diluar object rumah sehingga class rumah tetap exist.

Berikut ini adalah contoh implementasi agregasi berupa program java yang menampilkan class alamat dan class penduduk yang dimaksudkan pada contoh ini adalah seorang penduduk pasti memiliki alamat, sehingga penduduk merupakan bagian alamat. Walaupun penduduk merupakan bagian dari alamat, namun data penduduk dapat dipergunakan tanpa melibatkan data alamat. Contoh class ini menjelaskan sebuah class merupakan bagian dari class lain tetapi dapat berdiri sendiri. berikut contoh implementasinya pada program java agar lebih mudah dipahami:

|  |
| --- |
| [modifier]class Alamat{  //deklarasi variabel data alamat String AlmTinggal,KabKota,KodePos,Provinsi;  //constructor alamat{  ……  }  }  [modifier]class Penduduk{  //deklarasi variabel data penduduk String Nik,Nama,Pekerjaan;  Alamat almt;  //contructor alamat{  ……  }  }  [modifier]class mainAgregasi{  public static void main(String[]args){  Alamat pkp = new Alamat(“isi parameter sesuai constructor”);  Penduduk ani = new Penduduk(“isi parameter sesuai constuctor”,pkp);  tampilkan(ani.Nik); //output Nik  tampilkan(ani.Nama); //output Nama  tampilkan(ani.almt.AlmTinggal); //output AlmTinggal  tampilkan(ani.almt.KabKota); //output KabKota  tampilkan(ani.almt.Provinsi); //output Provinsi  }  } |

Setelah mengetahui apa itu asosiasi dan agregasi maka dapat diketahui bagaimana perbedaan antara asosiasi dan agregasi seperti berikut:

* Asosisasi: merupakan hubungan “Link”, menyimpan nilai atribut dengan tipe data asli, digambarkan dengan garis tegas.
* Agregasi: merupakan hubungan “Bagian”, menyimpan nilai atribut dengan tipe data class bentukan, digambarkan dengan hollow diamond.

Setelah mengetahui perbedaan dari asosiasi dan agregasi diharapkan agar bisa menjadi paham mengenai relasi asosiasi dan agregasi karna setelah asosiasi dan agregasi yaitu akan diperkenalkan relasi komposisi yang merupakan gabungan dari relasi asosiasi dan agregasi.

* + 1. Penerapan Relasi Agregasi

*Class* Mahasiswa :

|  |
| --- |
| 1. public class Mahasiswa { 2. private String nim; 3. private String nama; 4. public Mahasiswa(String nim, String nama) { 5. this.nim = nim; 6. this.nama = nama; 7. } 8. public String getNim() { 9. return nim; 10. } 11. public String getNama() { 12. return nama; 13. } 14. } |

Penjelasan :

Baris 1 : Deklarasi class Mahasiswa

Baris 2 - 3 : Deklarasi variabel

Baris 4 - 6 : Membuat constructor dengan 2 parameter

Baris 8 - 10 : Membuat getter untuk NIM

Baris 11 – 13 : Membuat getter untuk Nama

*Class* Jurusan :

|  |
| --- |
| 1. import java.util.ArrayList; 2. import java.util.List; 3. public class Jurusan { 4. private String kodeJurusan; 5. private String namaJurusan; 6. private List<Mahasiswa> mahasiswaList = new ArrayList<>(); 7. public Jurusan(String kodeJurusan, String namaJurusan) { 8. this.kodeJurusan = kodeJurusan; 9. this.namaJurusan = namaJurusan; 10. } 11. public List<Mahasiswa> getMahasiswaList() { 12. return mahasiswaList; 13. } 14. public void setMahasiswaList(List<Mahasiswa> mahasiswaList) { 15. this.mahasiswaList = mahasiswaList; 16. } 17. public void addMahasiswa(Mahasiswa mahasiswa){ 18. this.mahasiswaList.add(mahasiswa); 19. } 20. public String getKodeJurusan() { 21. return kodeJurusan; 22. } 23. public String getNamaJurusan() { 24. return namaJurusan; 25. } 26. public void getDataJurusan(){ 27. System.out.println("Nama Jurusan\t: " + getNamaJurusan() + "\n"); 28. System.out.println("List Mahasiswa\t: "); 29. for(Mahasiswa mahasiswa : mahasiswaList){ 30. System.out.println("Nama\t: " + mahasiswa.getNama()); 31. } 32. } 33. } |

Penjelasan :

Baris 1 - 2 : Menambah library

Baris 4 : Deklarasi class Jurusan

Baris 6 - 7 : Deklarasi variabel

Baris 8 : Deklarasi Array list dengan tipe data Mahasiswa

Baris 10 - 12: Membuat constructor dengan 2 parameter

Baris 15 - 17 : Membuat getter untuk mahasiswaList

Baris 19 - 21 : Membuat setter untuk mahasiswaList

Baris 23 - 25 : Membuat method untuk menambah Mahasiswa

Baris 27 - 29 : Membuat getter untuk kodeJurusan

Baris 31 - 33 : Membuat getter untuk namaJurusan

Baris 35 - 42 : Membuat getter untuk menampilkan Data Jurusan

*Class* DemoAgregasi :

|  |
| --- |
| 1. import java.util.ArrayList; 2. import java.util.List; 3. public class DemoAgregasi { 4. public static void main(String[] args) { 5. Jurusan se = new Jurusan("1810", "Software ENGINEERING"); 6. Mahasiswa mahasiswa1 = new Mahasiswa("18104005", "Bagus"); 7. Mahasiswa mahasiswa2 = new Mahasiswa("18104006", "Bayu"); 8. Mahasiswa mahasiswa3 = new Mahasiswa("18104007", "Sasongko"); 9. List<Mahasiswa> mahasiswaList= new ArrayList<>(); 10. mahasiswaList.add(mahasiswa1); 11. mahasiswaList.add(new Mahasiswa("18104001", "Ai")); 12. mahasiswaList.add(mahasiswa3); 13. se.setMahasiswaList(mahasiswaList); 14. se.getDataJurusan(); 15. } 16. } |

Penjelasan :

Baris 1 - 2 : Menambah Library

Baris 4 : Deklarasi class DemoAgregasi

Baris 5 : Deklarasi method main

Baris 6 : Membuat objek jurusan dengan parameter

Baris 8 - 10 : Membuat objek jurusan dengan parameter

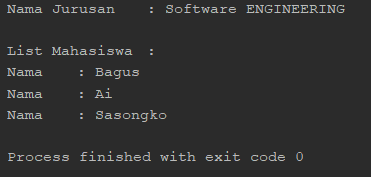
Baris 12 : Deklarasi Array List dengan tipe data Mahasiswa

Baris 15 - 17 : Menambah elemen mahasiswaList

Baris 18 : Memanggil method setMahasiswaList

Baris 19 : Memanggil method getDataJurusan

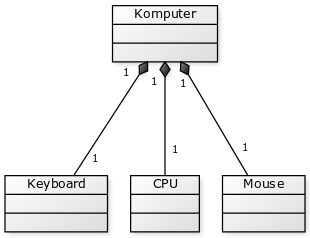
Keluarannya :



Gambar 6.6 Output Program menggunakan Agregasi

* 1. Komposisi
     1. Hubungan Komposisi antar *Class*

Komposisi merupakan bentuk khusus dari agregasi dimana terdapat hubungan antar kelas yang lebih kuat, karena yang menjadi part (bagian) tidak akan ada tanpa adanya kelas yang menjadi whole. Contoh Penerapan Relasi Komposisi:



Gambar 6.7 Kelas diagram hubungan komposisi

Pada Gambar 6.7, diagram kelas di atas terlihat bahwa kelas CPU, Monitor, dan Mouse semuanya merupakan bagian dari kelas Komputer dan ketika kelas Komputer musnah maka kelas CPU, Monitor, dan Mouse akan ikut musnah.

Pada inheritance atau konsep IS-A (adalah), misalnya class Mobil extends class Kendaraan, dalam hal ini Mobil juga adalah Kendaraan. Jadi class Mobil akan memiliki semua fitur atau properti dari class Kendaraan selain dari properti lainnya yang dimiliki oleh class Mobil itu sendiri. Pada inheritance child class IS-A parent class. Namun, komposisi menggunakan konsep HAS-A (memiliki). Ini berbeda dengan inheritance, di mana pada komposisi, suatu class akan menggunakan kembali fungsionalitas dengan membuat referensi ke suatu objek class yang ingin digunakan kembali. Contohnya mobil memiliki setir, rumah memiliki pintu, kantor memiliki karyawan.

Komposisi dalam pemrograman Java itu ada karena setiap objek dari class yang berbeda tidak bisa selalu berhubungan dengan inheritance. Misalnya class Mobil dan class Mesin, sekarang apakah mobil adalah mesin? Tentu bukan, jadi konsep inheritance tidak bisa digunakan. Sekarang pertanyaannya adalah apakah mobil memiliki mesin? Ya, ini bisa diterima. Jadi untuk menghubungkan class semacam ini maka komposisi digunakan. Komposisi dapat memodelkan objek yang terdiri dari objek lain sehingga hubungan HAS-A dapat terjadi di antara objek-objek tersebut. Komposisi juga merupakan asosiasi terkuat, ini artinya ketika suatu objek di musnahkan (di garbage collection) maka objek lain yang menyusun atau terkandung dalam objek tersebut juga akan ikut musnah. Ada beberapa perbedaan antara komposisi dan inheritance pada java.

Berikut adalah perbedaan antara komposisi dan inheritance pada program java:

1. Fleksibilitas

Komposisi lebih fleksibel daripada inheritance. Karena ketika Anda menggunakan inheritance maka Anda harus menetapkan terlebih dahulu class mana yang akan diwariskan. Dan hal ini tidak bisa diubah pada saat runtime. Sedangkan pada komposisi Anda hanya mendefinisikan Tipe yang akan digunakan yang dapat memegang implementasi yang berbeda.

1. Dengan inheritance penggunaan kode kembali lebih terbatas

Ketika Anda menggunakan inheritance maka Anda hanya memperluas satu class, artinya Anda hanya dapat menggunakan kembali hanya satu class dan tidak lebih dari satu. Kalau Anda ingin memanfaatkan fungsionalitas dari beberapa class maka komposisi harus digunakan.

1. Tidak ada kontrol akses dalam inheritance

Dalam inheritance tidak ada kontrol akses. Dalam arti kita mengekspos semua method superclass ke class lain yang memiliki akses ke subclass. Jadi, ketika ada method baru atau celah keamanan di superclass maka subclass menjadi rentan.Sedangkan dalam komposisi akses dapat dibatasi. Karena pada komposisi Anda memilih method mana yang akan digunakan sehingga lebih aman daripada inheritance.

1. Unit testing

Komposisi memiliki ruang lingkup pengujian yang lebih mudah dibandingkan dengan inheritance. Ini karena dalam komposisi Anda tahu tentang semua method yang digunakan dari class lain. Sedangkan pada inheritance semuanya sangat bergantung pada superclass dan sulit mengetahui apakah semua method superclass akan digunakan. Dengan kata lain, pada inheritance Anda harus menguji semua method dari superclass untuk menguji subclass.

1. Enkapsulasi

Meskipun inheritance dan komposisi memungkinkan kembali penggunaan kode (reusable code) tapi inheritance memecah enkapsulasi. Alasannya adalah pada inheritance subclass akan tergantung pada perilaku superclass. Oleh sebab itu, ketika ada perubahan pada superclass maka subclass juga akan ikut terpengaruh.Jadi, ketika suatu class tidak didokumentasikan dengan baik dan subclass tidak menggunakan superclass dengan cara yang semestinya maka setiap perubahan yang terjadi di superclass dapat merusak fungsionalitas subclass.

1. Class Final

Seperti yang sudah Anda ketahui, ketika class dideklarasikan final maka class tersebut tidak bisa diwariskan. Jadi kode yang ada pada class tersebut tidak bisa digunakan kembali (non-reusable). Tapi dengan komposisi Anda tetap bisaa menggunakan kode dari class final.

* + 1. Penerapan Relasi Komposisi

*Class* CPU

|  |
| --- |
| 1. public class CPU { 2. private String Merk; 3. private int Kecepatan; 4. public CPU(String merk, int kecepatan) { 5. Merk = merk; 6. Kecepatan = kecepatan; 7. } 8. public void DisplaySpecCPU(){ 9. System.out.println(this.Merk + ", " + this.Kecepatan + "GHz"); 10. } 11. } |

Penjelasan :

Baris 1 : Deklarasi class

Baris 2 - 3 : Deklarasi variabel

Baris 5 - 7 : Deklarasi constructor dengan 2 parameter

Baris 10 - 12 : Method untuk menampilkan data

*Class* Monitor :

|  |
| --- |
| 1. public class Monitor { 2. private String Merk; 3. public Monitor(String merk) { 4. Merk = merk; 5. } 6. public void DisplaySpecMonitor(){ 7. System.out.println(this.Merk); 8. } 9. } |

Penjelasan :

Baris 1 : Deklarasi class

Baris 2 : Deklarasi variabel

Baris 4 - 6 : Deklarasi constructor dengan 1 parameter

Baris 7 - 9 : Method untuk menampilkan data

*Class* Mouse :

|  |
| --- |
| 1. public class Mouse { 2. private String Merk, Type; 3. public Mouse(String merk, String type) { 4. Merk = merk; 5. Type = type; 6. } 7. public void DisplaySpecMouse(){ 8. System.out.println(this.Merk + ", " + this.Type); 9. } 10. } |

Penjelasan :

Baris 1 : Deklarasi class

Baris 2 : Deklarasi variabel

Baris 4 - 7 : Deklarasi constructor dengan 1 parameter

Baris 8 - 10 : Method untuk menampilkan data

*Class* Komputer

|  |
| --- |
| 1. package Komposisi; 2. public class Komputer { 3. private String merk; 4. private String tipe; 5. private Mouse mouse; 6. private Monitor monitor; 7. private CPU cpu; 8. public Komputer(String merk, String tipe) { 9. this.merk = merk; 10. this.tipe = tipe; 11. } 12. public void addKomponen(Mouse mouse, Monitor monitor, CPU cpu){ 13. this.mouse = mouse; 14. this.monitor = monitor; 15. this.cpu = cpu; 16. } 17. public String getMerk() { 18. return merk; 19. } 20. public void setMerk(String merk) { 21. this.merk = merk; 22. } 23. public String getTipe() { 24. return tipe; 25. } 26. public void setTipe(String tipe) { 27. this.tipe = tipe; 28. } 29. public Mouse getMouse() { 30. return mouse; 31. } 32. public void setMouse(Mouse mouse) { 33. this.mouse = mouse; 34. } 35. public Monitor getMonitor() { 36. return monitor; 37. } 38. public void setMonitor(Monitor monitor) { 39. this.monitor = monitor; 40. } 41. public CPU getCpu() { 42. return cpu; 43. } 44. public void setCpu(CPU cpu) { 45. this.cpu = cpu; 46. } 47. } |

Penjelasan :

Baris 1 : Mendeklarasikan package

Baris 3 : Deklarasi class

Baris 4 - 8 : Deklarasi variabel

Baris 10 - 13 : Membuat constructor dengan 2 parameter

Baris 15 - 19 : Membuat method untuk menambah data dengan nama addKomponen

Baris 21 - 27 : Membuat getter dan setter dari Merk

Baris 29 - 31 : Membuat getter dan setter dari Tipe

Baris 37 - 43 : Membuat getter dan setter dari Mouse

Baris 45 - 51 : Membuat getter dan setter dari Monitor

Baris 53 - 59 : Membuat getter dan setter dari CPU

*Class* MainKomputer

|  |
| --- |
| 1. public class MainKomputer { 2. public static void main(String[] args) { 3. Komputer komputer = new Komputer("Asus","A450CC"); 4. komputer.addKomponen(new Mouse("Logitec", "M90"), new Monitor("LG"), new CPU("intel i7", 2)); 5. System.out.println("Merk Komputer : "+komputer.getMerk()); 6. System.out.println("Type Komputer : "+komputer.getTipe()); 7. System.out.print("Mouse : "); 8. komputer.getMouse().DisplaySpecMouse(); 9. System.out.print("Cpu : "); 10. komputer.getCpu().DisplaySpecCPU(); 11. System.out.print("Monitor : "); 12. komputer.getMonitor().DisplaySpecMonitor(); 13. } 14. } |

Penjelasan :

Baris 1 : Deklarasi class

Baris 2 : Deklarasi method main

Baris 3 : Membuat variabel objek dari Komputer

Baris 4 : Memanggil method addKomponen untuk menambah data

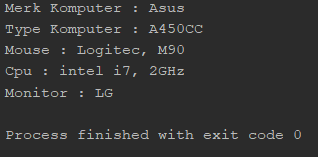
Baris 5 : Menampilkan data dari merk Komputer

Baris 6 : Menampilkan data dari type Komputer

Baris 7 - 8 : Menampilkan data dari Mouse

Baris 9 - 10 : Menampilkan data daro CPU

Baris 11 - 12 : Menampilkan data dari Monitor



Gambar 6.7 Output Program menggunakan Komposisi

* 1. Latihan

(isikan source code tugas/latihan yang diberikan asisten pada saat praktikum, dijelaskan per baris. Tambahkan screenshot!)

Nb1. Nomer Halaman Melanjutkan Modul Sebelumnya

Nb2. Jangan lupa asistensi !