Modul 12

Design Pattern 1 (Composite - Observer)

Praktikum Pemrograman Berorientasi Objek

2023

Tujuan:

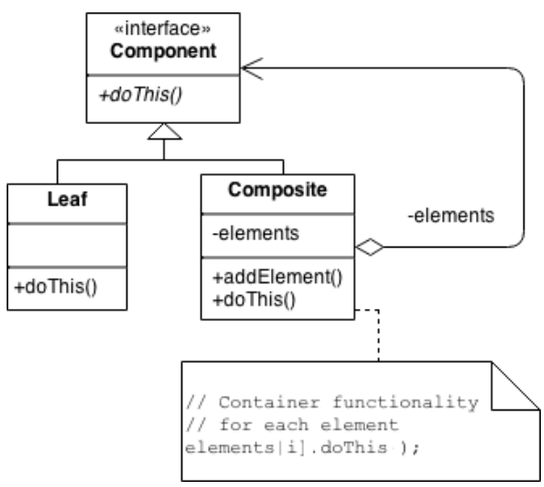
1. Memahami cara kerja Design Pattern Composite dan Observer
2. Menerapkan design pattern dalam penyelesaian suatu permasalahan

Dasar Teori:

Design pattern **composite** dan design pattern **Observer** merupakan contoh dari **Structural Patterns** dan juga **Behavioral Patterns.**

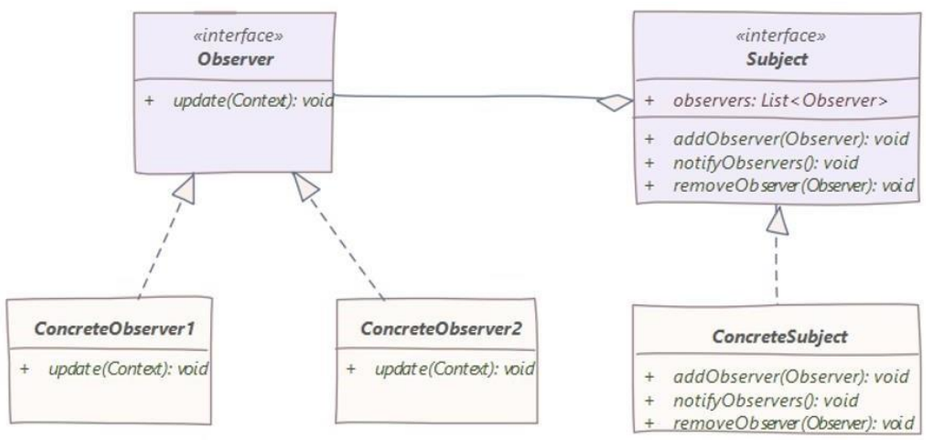
- **Structural Patterns**
  Structural Pattern sendiri akan membantu dalam menjabarkan suatu kelas dan objek menjadi struktur yang lebih besar dengan fleksibel dan efisien. Referensi: https://refactoring.guru/design-patterns/structural-patterns

  Dalam praktikum ini, salah satu structural pattern yang akan diterapkan yaitu **Composite.** Composite sendiri biasanya digunakan untuk menyusun objek ke dalam struktur pohon dan mengerjakan struktur tersebut seakan-akan objek tersebut merupakan objek individu. Salah satu contohnya adalah dengan menggunakan struktur organisasi suatu perusahaan dimana akar dari pohon tersebut yaitu kepala perusahaannya, kemudian cabang-cabangnya merupakan karyawannya. Karyawannya pun masih bisa memiliki karyawan yang berada di bawahnya lagi.



- **Behavioral Patterns**
  Behavioral pattern dapat membantu menunjukkan algoritma dan penugasan tanggung jawab antar objek. Referensi:
  https://refactoring.guru/design-patterns/behavioral-patterns

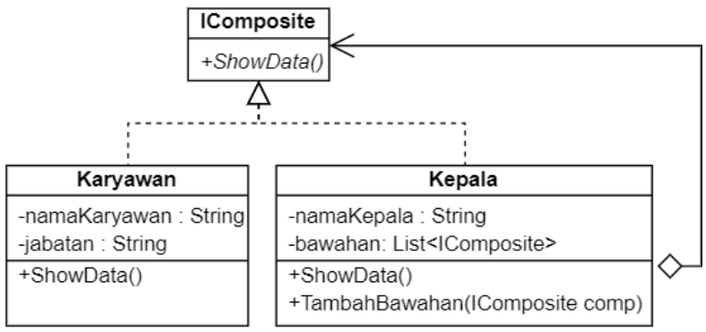
  Behavioral pattern yang digunakan pada praktikum ini yaitu **Observer.** Observer biasanya dapat digunakan dalam kasus seperti notify pada youtube (Lonceng) dimana semua orang yang mengaktifkan notify nya, akan mendapat notifikasi setiap channel yang diinginkan mengupload video.



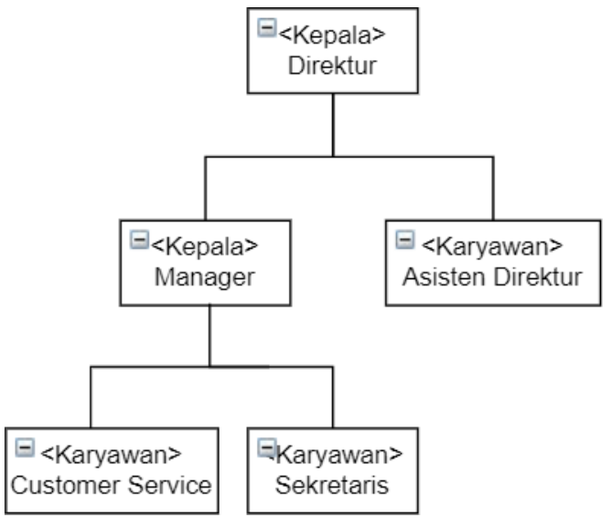
Pada Modul ini, akan terdapat 2 Guided, yaitu Guided untuk Composite dan Guided untuk Observer

Guided 1 – Composite

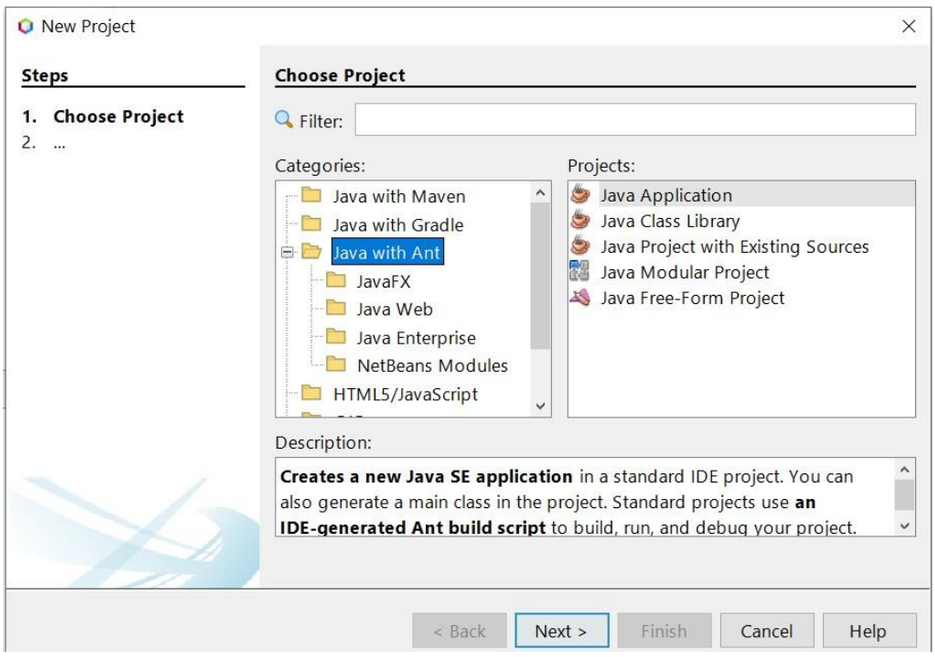
GD12_X_YYYYY_1



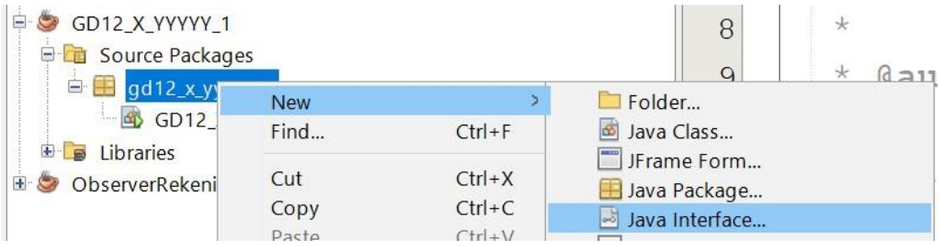
Dalam guided kali ini, kasus yang diberikan sebagai contoh yaitu, seorang Kepala perusahaan yang bisa memiliki bawahan berupa kepala lain maupun karyawan. Dimana kelas Kepala merupakan Composite, sehingga bisa memiliki bawahan lagi. Sedangkan kelas Karyawan merupakan Leaf, sehingga tidak bisa memiliki bawahan. Struktur seperti gambar di bawah yang akan kita buat dalam guided kali ini.

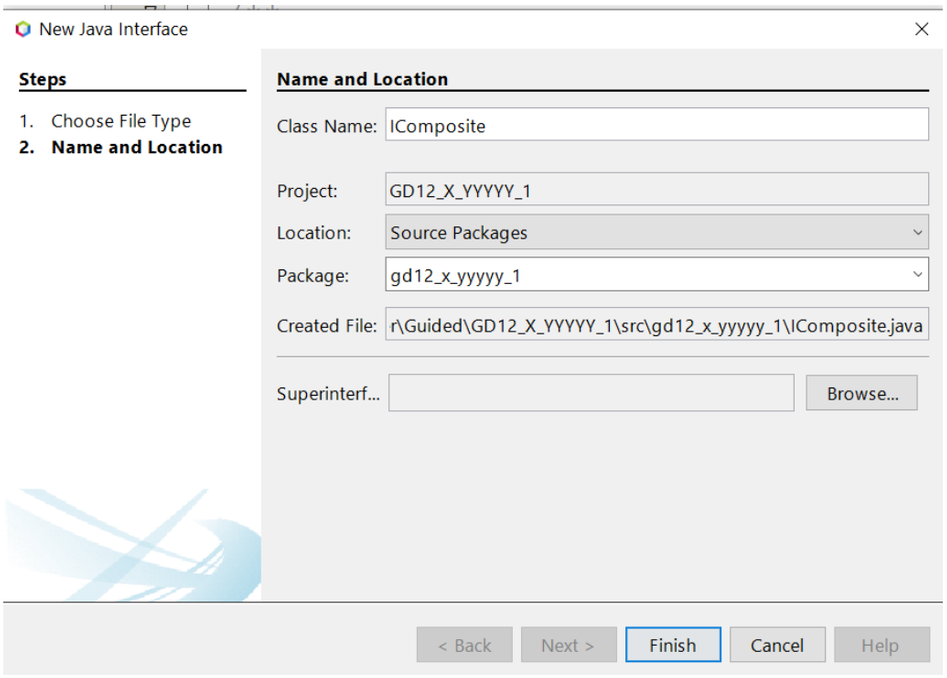


1. Pertama, buat projek terlebih dahulu dengan nama GD12_X_YYYYY_1.

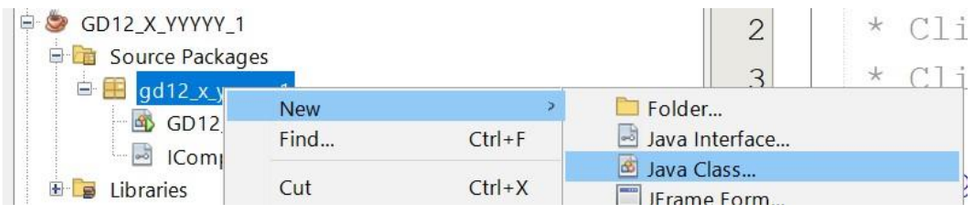


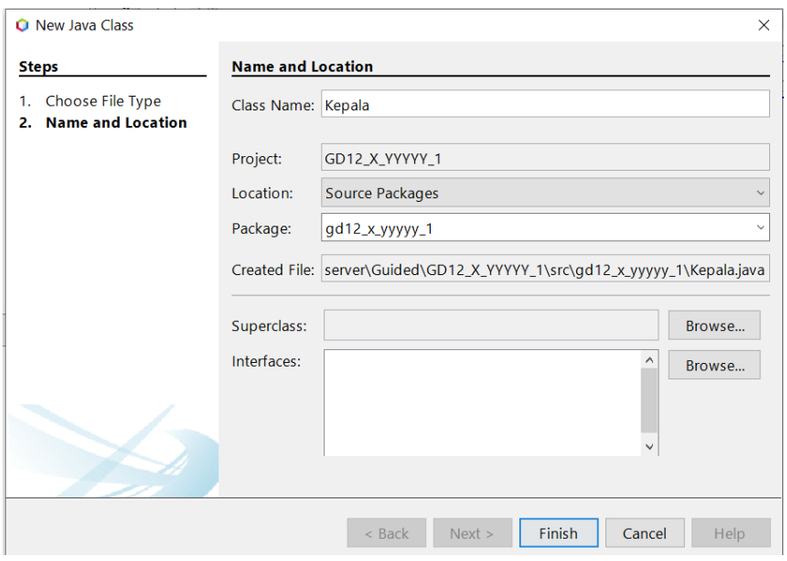
2. Kemudian setelah projek terbentuk, kita akan membuat interface sebagai componentnya dengan nama IComposite.

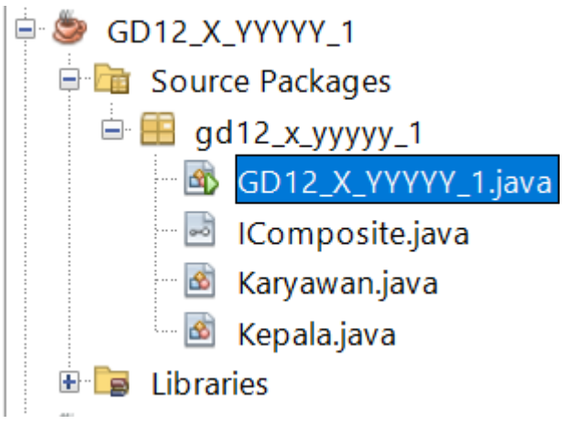




3. Kemudian kita akan membuat 2 java class yaitu, Kepala dan Karyawan.







4. Selanjutnya, mari kita mengisi interface IComposite (Component) terlebih dahulu.

```
10   */
     public interface IComposite {
12       public static StringBuffer space = new StringBuffer();
13
         public void ShowData();
15   }
```

space akan digunakan untuk mengatur tata letak saat menampilkan data.

5. Setiap kelas yang dibutuhkan untuk menjadi struktur composite ini (dimana pada guided kali ini, kelas tersebut yaitu Karyawan dan Kepala), harus diimplementkan dengan interface IComposite, kita akan mengisi kelas Karyawan.java (leaf) terlebih dahulu.

```
12   public class Karyawan implements IComposite{
         private String namaKaryawan;
         private String jabatan;
15
16       public Karyawan(String namaKaryawan, String jabatan) {
17           this.namaKaryawan = namaKaryawan;
18           this.jabatan = jabatan;
19       }
20
21       @Override
         public void ShowData() {
23           System.out.println(namaKaryawan +" - "+ jabatan);
24       }
25   }
26
```

6. Setelah Karyawan.java selesai, kita akan melanjutkan ke Kepala.java / compositenya.

```
7  import java.util.ArrayList;
8
9  public class Kepala implements IComposite{
       private String namaKepala;
       private ArrayList<IComposite> bawahan;
12
13     public Kepala(String namaKepala) {
14         this.namaKepala = namaKepala;
           this.bawahan = new ArrayList<IComposite>();
16     }
17
18     public void TambahBawahan(IComposite comp)
19     {
20         bawahan.add(comp);
21     }
22
```

```
23     @Override
       public void ShowData() {
25         System.out.println(IComposite.space + "Bapak Kepala " + namaKepala);
26         IComposite.space.append("    ");
27         // Range-Based Loop
28         /**
29          * for(tipeData Nama : ArrayList<tipeData>)
30          * {
31          *     Akan beriterasi sebanyak isi dari ArrayList<tipeData>
32          *     tidak perlu melakukan pengecekan index keberapa karena
33          *     Nama sudah mewakili objectnya (lebih jelas divideo)
34          * }
35          *
36          * Sama saja dengan
37          * for(int i = 0; i < ArrayList<tipeData>.size(); i++)
38          * {
39          *     tetapi di for biasa, kita perlu mengakses manual isi
40          *     dari list tersebut berdasarkan i / indexnya
41          * }
42          */
43
44         for(IComposite comp : bawahan)
45         {
46             System.out.print(IComposite.space + "Bawahan dari " + namaKepala + " ");
47             comp.ShowData();
48         }
49
50         IComposite.space.setLength(IComposite.space.length() - 3);
51     }
52 }
53
```

Kali ini, kita akan menggunakan Range-Based Loop dimana perulangan dengan bentuk seperti ini sangat mempermudah dalam melakukan iterasi dalam sebuah array / list / arrayList. Jika membutuhkan perulangan sejumlah bilangan yang pasti / membutuhkan index, sebaiknya menggunakan perulangan biasa (contoh bilangan pasti, ingin melakukan perulangan 3x saja. Contoh menggunakan index, khusus index ke-3, maka akan melakukan sesuatu).

Untuk konsep dari compositenya sendiri, sama saja dengan konsep penggunaan struktur data rekursif dimana pada fungsi rekursif, terdapat pemanggilan terhadap fungsi itu sendiri. Dapat kita

lihat pada baris ke 47, dimana terdapat pemanggilan ke fungsi ShowData() (*comp.ShowData()*), apabila *comp* merupakan kelas Kepala, maka akan terdapat pemanggilan terhadap fungsi yang sama (*ShowData()* milik Kepala). Tetapi jika *comp* merupakan kelas Karyawan, akan memanggil fungsi *ShowData()* milik kelas Karyawan. Sehingga walaupun seorang Kepala memiliki bawahan berupa Kepala lain, akan mengakibatkan terjadinya rekursif.

7. Terakhir, mari kita isi main class nya, jangan lupa mengganti Nama dan NPM kalian.

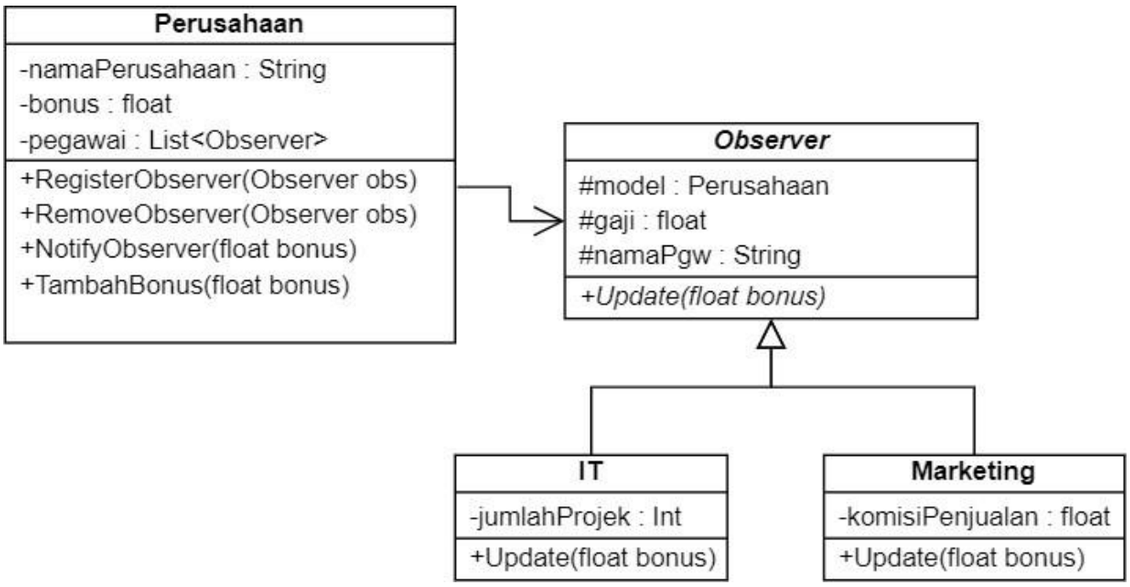
```
16      public static void main(String[] args) {
17          // TODO code application logic here
18          Kepala direktur = new Kepala("Direktur NAMAPRAKTIKAN");
19          Kepala manager = new Kepala("Manager NPMPRAKTIKAN");
20
21          Karyawan asisten = new Karyawan("Bobi", "Asisten Direktur");
22          Karyawan cs = new Karyawan("Mari", "Customer Service");
23          Karyawan sekret = new Karyawan("Studi", "Sekretaris");
24
25          direktur.TambahBawahan(manager);
26          direktur.TambahBawahan(asisten);
27
28          manager.TambahBawahan(cs);
29          manager.TambahBawahan(sekret);
30
31          direktur.ShowData();
32      }
33  }
34
```

Berikut merupakan hasil outputnya saat di Run:

```
run:
Bapak Kepala Direktur NAMAPRAKTIKAN
   Bawahan dari Direktur NAMAPRAKTIKAN    Bapak Kepala Manager NPMPRAKTIKAN
      Bawahan dari Manager NPMPRAKTIKAN Mari - Customer Service
      Bawahan dari Manager NPMPRAKTIKAN Studi - Sekretaris
   Bawahan dari Direktur NAMAPRAKTIKAN Bobi - Asisten Direktur
BUILD SUCCESSFUL (total time: 0 seconds)
```

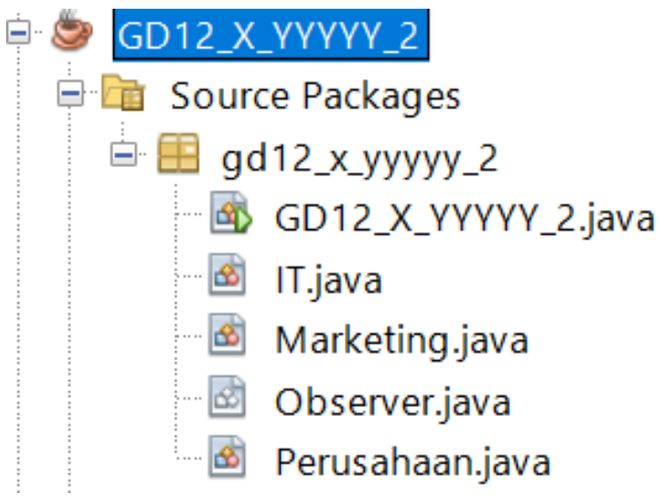
Guided 2 – Observer

GD12_X_YYYYY_2



Kasus yang diberikan pada guided ini yaitu, terdapat dua observer yaitu IT dan Marketing, yang bisa didaftarkan pada class Subjectnya (Perusahaan). Observer tersebut mengamati atribut kelas Perusahaan yang bernama bonus. Ketika terdapat perubahan terhadap variabel bonus melalui fungsi TambahBonus(float bonus), maka semua observer yang mengamati subjek ini akan diberi tahu dan akan menjumlahkan gaji yang lama dengan variabel bonus yang dipengaruhi juga oleh jumlahProjek yang dikerjakan, maupun jumlah komisiPenjualan yang didapat.

1. Pertama kita akan membuat semua class berdasarkan class diagramnya. Terdapat empat java class yaitu IT, Marketing, Observer dan Perusahaan. Observer merupakan kelas abstract.



2. Kita akan mengisi kode kelas abstract (Observer) terlebih dahulu.

```
public abstract class Observer {
    protected Perusahaan model;
    protected String namaPgw;
    protected float gaji;

    public Observer(Perusahaan model, float gaji, String namaPgw)
    {
        this.model = model;
        this.gaji = gaji;
        this.namaPgw = namaPgw;
        this.model.RegisterObserver(this);
    }

    public abstract void Update(float bonus);
}
```

3. Kemudian kita akan mengisi kedua class yang merupakan kelas anak dari Observer yaitu IT dan Marketing, mari kita isi mulai dari IT.java.

```
public class IT extends Observer{
    private int jumlahProjek;

    public IT(float gaji, int jumlahProjek, String nama, Perusahaan model) {
        super(model, gaji, nama);
        this.jumlahProjek = jumlahProjek;
    }

    @Override
    public void Update(float bonus) {
        float tempGaji = gaji;
        gaji += bonus * (jumlahProjek + 1);
        System.out.println(namaPgw + "-- Gaji sebelum: " + tempGaji
                + " Gaji sesudah: " + gaji);
    }
}
```

4. Kelas Marketing.java memiliki code yang hampir sama dengan IT, tetapi terdapat sedikit perbedaan pada rumus perhitungan bonusnya.

```
11  public class Marketing extends Observer{
        private float komisiPenjualan;
13
14      public Marketing(float gaji, float komisiPenjualan, String nama, Perusahaan model) {
15          super(model, gaji, nama);
16          this.komisiPenjualan = komisiPenjualan;
17      }
18
19      @Override
        public void Update(float bonus) {
21          float tempGaji = gaji;
            gaji += bonus + komisiPenjualan;
23          System.out.println(namaPgw + "-- Gaji sebelum: " + tempGaji
24                                     + " Gaji sesudah: " + gaji);
25      }
26  }
27
```

5. kita akan melanjutkan ke kelas subjectnya yaitu Perusahaan.java.

```
7   import java.util.ArrayList;
8
9   public class Perusahaan{
        private String namaPerusahaan;
11      private float bonus;
        private ArrayList<Observer> pegawai;
13
14      public Perusahaan(String namaPerusahaan) {
15          this.namaPerusahaan = namaPerusahaan;
16          bonus = 0;
            this.pegawai = new ArrayList<Observer>();
18      }
19
20      public void RegisterObserver(Observer obs) {
21          pegawai.add(obs);
22      }
23
```

```
24      public void RemoveObserver(Observer obs) {
25          pegawai.remove(obs);
26      }
27
28      public void NotifyObserver(float bonus) {
29          for(Observer obs : pegawai)
30          {
31              obs.Update(bonus);
32          }
33      }
34
35      public void TambahBonus(float bonus)
36      {
37          this.bonus = bonus;
38          NotifyObserver(bonus);
39      }
40  }
41
```

6. Terakhir, kita akan mengerjakan main classnya.

```
public class GD12_X_YYYYY_2 {

    public static void main(String[] args) {
        Perusahaan perus = new Perusahaan("Perusahaan DP1");

        Observer pgw1 = new Marketing(10000, 40000, "Market", perus);
        Observer pgw2 = new IT(30000, 3, "IT", perus);

        perus.TambahBonus(5000);

        perus.RemoveObserver(pgw2);
        System.out.println("\nSetelah Remove\n");
        perus.TambahBonus(2000);
    }
}
```

Berikut merupakan outputnya:

```
run:
Market-- Gaji sebelum: 10000.0 Gaji sesudah: 55000.0
IT-- Gaji sebelum: 30000.0 Gaji sesudah: 50000.0

Setelah Remove

Market-- Gaji sebelum: 55000.0 Gaji sesudah: 97000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Pengumpulan:

- **Ada 2 Project, GD12_X_YYYYY_1 dan GD12_X_YYYYY_2, kemudian di zip menjadi 1 dengan format GD12_X_YYYYY**
  X = Kelas
  YYYYY = 5 Digit NPM

Silahkan dijadikan satu dan dikumpulkan dengan format GD12_X_YYYYY.zip. Jika ada pertanyaan silahkan hubungin aku sebagai pemengang modul atau asisten lain. Selamat Belajar.