**Form Diskusi Pertemuan 11**

**Design Patterns 01**

**Tujuan diskusi:**

* Mahasiswa mampu mengimplementasi Observer dan Composite pattern dengan menggunakan bahasa Java.

**Tutorial 11.1: Non Observer**

|  |
| --- |
| 1. Perhatikan kode program berikut ini. Kode program ini menunjukkan konsep observer secara konvensional.   //simpan dalam file BinObserver.java  public class BinObserver {  public void update( int val ) {  System.out.println ("Bin : " + Integer.toBinaryString(val));  }  }  //simpan dalam file OctObserver.java  public class OctObserver {  public void update( int val ) {  System.out.println ("Oct : " + Integer.toOctalString(val));  }  }  //simpan dalam file Subject.java  public class Subject {  private int subjectState;  private OctObserver octObserver;  private BinObserver binObserver;  public Subject() {  octObserver = new OctObserver();  binObserver = new BinObserver();  }  public void setState( int val ) {  subjectState = val;  octObserver.update( subjectState );  binObserver.update( subjectState );  }  }  //simpan dalam file NonObserver.java  import java.io.\*;  public class NonObserver {  public static void main(String[] args) throws IOException {  Subject subj = new Subject();  String valueStr;  InputStreamReader values = new InputStreamReader (System.in);  BufferedReader br = new BufferedReader(values);  System.out.print("Input integer : ");  valueStr = br.readLine();  subj.setState (Integer.parseInt(valueStr));  }  }   1. Unduh projek **NonObserver**  yang disediakan di situs kuliah. Gunakan Netbeans, cermati kode program dan eksekusi project tersebut. 2. **Gambar relasi kelas (kelas diagram dari kode program tersebut)** |
|  |
| 1. Dalam kode program di atas, ada 2 observer yaitu BinObserver dan OctObserver. Ada keinginan untuk menambahkan observer yang ketiga yaitu HexObserver yang memiliki kode program berikut ini:   //simpan dalam file HexObserver.java  public class HexObserver {  public void update( int val ) {  System.out.println ("Hex : " + Integer.toHexString(val));  }  }  Lakukan modifikasi terhadap kelas Subject sehingga apabila diuji dengan NonObserver akan menghasilkan keluaran seperti berikut ini: |
| 1. Apa kelemahan yang terjadi dengan struktur seperti di atas. |
|  |
| 1. Apa solusi atas kelemahan pada poin 5 di atas ? (Ingat tutorial pertemuan 10, tutorial 10.4 terutama pada langkah 2 dan 3). |
|  |

**Tutorial 11.2: Observer**

|  |
| --- |
| 1. Dari tutorial 11.1 terlihat bahwa terjadi keterikatan yang kuat antara kelas Subject dan kelas-kelas observer yang ada. Penambahan kelas observer mengakibatkan terjadinya perubahan atau modifikasi pada kelas Subject. Ada prinsip perancangan yang dilanggarkan dalam tutorial 11.1 tersebut.   Untuk mengurangi keterikatan tersebut, maka solusinya adalah dengan menggunakan pattern Observer, dengan struktur lengkap seperti berikut ini:  Gambar 1. Struktur lengkap pattern Observer  Dalam tutorial ini, kita mengasumsikan bahwa subject yang digunakan hanya satu, sementara observernya bersifat dinamis. Dengan demikian kita tidak menggunakan interface atau kelas induk Subject. Subject yang digunakan adalah hanya concrete subject saja. Struktur kelas yang digunakan dalam tutorial ini adalah:  Gambar 2. Struktur pattern Observer dalam tutorial   1. Buatlah project dengan nama **Observer** 2. Buatlah kelas **Subject**, dengan kode   import java.util.\*;  public class Subject {  private Vector<Observer> views;  private int value;  /\*\* Creates a new instance of Subject \*/  public Subject() {  views = new Vector<Observer>();  }  public void attach (Observer obs) {  views.add(obs);  }  public void notifyObserver(int val) {  for (int i=0; i < views.size(); i++)  views.elementAt(i).**update**(val);  }  public void setState(int val) {  value = val;  **notifyObserver**(val);  }  }   1. Buatlah kelas abstrak **Observer**, dengan kode   public **abstract** class Observer {  private Subject model;  public Observer (Subject mod) {  model = mod;  // Observers mendaftar ke Subject  model.**attach**(this);  }  public abstract void update(int val);  }   1. Buatlah kelas **BinObserver** , **OctObserver** kelas **HexObserver,** dengan kode program berikut ini:   //simpan dalam file BinObserver.java  public class BinObserver extends Observer {  public BinObserver(**Subject mod**) {  super (mod);  }  public void **update**( int val ) {  System.out.println ("bin : " + Integer.toBinaryString(val));  }  }  //simpan dalam file OctObserver.java  public class OctObserver extends Observer {  public OctObserver (**Subject mod**) {  super (mod);  }  public void **update**( int val ) {  System.out.println ("Oct : " + Integer.toOctalString(val));  }  }  //simpan dalam file HexObserver.java  public class HexObserver extends Observer {  public HexObserver (**Subject mod**) {  super (mod);  }  public void **update**( int val ) {  System.out.println ("hex : " + Integer.toHexString(val));  }  }   1. Buatlah kelas **ObserverTest** dengan kode berikut ini:   import java.io.\*;  public class ObserverTest {  public static void main(String[] args) throws IOException {  Subject subj = new Subject();  HexObserver hexObserver = new HexObserver(**subj**);  OctObserver octObserver = new OctObserver(**subj**);  BinObserver binObserver = new BinObserver(**subj**);  String valueStr;  InputStreamReader values = new InputStreamReader (System.in);  BufferedReader br = new BufferedReader(values);  do {  System.out.print("Input integer : ");  valueStr = br.readLine();  if (valueStr.equalsIgnoreCase("00"))  break;  **subj.setState( Integer.parseInt(valueStr));**  } while(true);  }  }   1. Eksekusi program. 2. Amati kode program di tutorial 11.1 dan tutorial 11.2**.** Apa perbedaan kode program di kedua di tutorial 11.1 dan 11.2 ? |
|  |

**Tutorial 11.3: Composite Pattern**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Buatlah project dengan nama **Composite** 2. Tutorial ini akan mencoba menunjukkan bagaimana penerapan design pattern composite dalam struktur organisasi. Kode program dibuat berdasarkan ide struktur berikut ini:   Gambar 3. Struktur lengkap pattern Composite  Pada tutorial ini, akan diimplementasikan Composite Pattern, dengan kelas-kelas yang terlibat adalah sebagai berikut:   |  |  |  | | --- | --- | --- | | No | Kelas dalam Composite Pattern | Kelas Tutorial | | 1 | Component | Employee | | 2 | Composite | Manager | | 3 | Leaf | Staf | | 4 | Client | CompositeTest |   Gambar 4. Struktur lengkap pattern Composite sesuai Kasus   1. Buatlah kelas **Employee** berikut ini.   public abstract class Employee {  public static StringBuffer space = new StringBuffer();  // atribut-atribut kelas Employee sebagai sebuah konsep  protected String role; // jabatan employee  protected String name;  protected float asuransi; // premi asuransi employee  public Employee (String nm, String rl, float asr) {  name = nm; role = rl;  asuransi = asr;  }  // fungsi-fungsi Employee sebagai abstract component  // dalam pattern composite. Override di kelas turunan  public abstract void addChild(Employee e);  public abstract Employee getChild(int i);  public abstract void traverseEmp();  // fungsi-fungsi getter kelas Employee sebagai sebuah konsep  public String getRole() { return role; }  public String getName() { return name; }  public float getAsuransi() { return asuransi ; }  }   1. Buat kelas **Manager** dan kelas **Staf**, yang merupakan turunan dari kelas **Employee**   //Manager.java  import java.util.ArrayList;  import java.util.List;  public class Manager extends Employee {  // atribut kelas Manager sebagai composite class dalam  // pattern composite  List<Employee> employees = new ArrayList<Employee>();  public Manager(String nm, String rl){  super(nm, rl, 1000000);  }  // fungsi-fungsi kelas Manager sebagai composite class  // dalam pattern composite  public void addChild(Employee e) { // Catat siapa saja yang  employees.add(e); // menjadi bawahan manager  }  public Employee getChild(int i) { // Cari tahu bawahan untuk  return employees.get(i); // seorang manager  }  public void **traverseEmp**() { // tampilkan semua employee  System.out.println( Employee.space + getName() + "(" +  getRole() + " - " + getAsuransi() + ")" );  Employee.space.append(" ");  **for (Employee e : employees)**  **{**  **e.traverseEmp();**  **}**  Employee.space.setLength(Employee.space.length() - 3);  }  }  //Staf.java  public class Staf extends Employee {  public Staf (String nm, String rl){  super(nm, rl, 750000);  }  // fungsi-fungsi kelas Staf sebagai leaf class dalam  // pattern composite  public void addChild(Employee employee) {} // do nothing. Kelas Staf  // tidak memiliki bawahan  public Employee getChild(int i) {  return null;  }  public void **traverseEmp**() {  System.out.println( Employee.space + getName() + "(" +  getRole() + " - " + getAsuransi() + ")" );  }  }   1. Buatlah kelas **CompositeTest** dengan kode berikut ini:   public class CompositeTest {  public static void main(String[] args) {  Employee direktur = new Manager("Arief Nugraha", "Direktur");  Employee mankeu = new Manager("Suryana", "Manajer Keuangan");  Employee mansdm = new Manager("Susila","Manajer SDM");  Employee kaakun = new Manager("Septarini","Kabag Akuntansi");  Employee kaaudt = new Manager("Herjuna","Kabag Audit");  Employee kapsdm = new Manager("Dewi Yani", "Kabag SDM");  Employee kaprom = new Manager("Eleonora", "Kabag Promosi");  Employee skeu1 = new Staf("Panduwinata", "Staf Keuangan");  Employee skeu2 = new Staf("Megadewi", "Staf Keuangan");  Employee ssdm1 = new Staf("Trifina", "Staf SDM");  Employee ssdm2 = new Staf("Kanda", "Staf SDM");  Employee ssdm3 = new Staf("Arjuna", "Staf SDM");  direktur.addChild(mankeu); direktur.addChild(mansdm);  mankeu.addChild(kaakun); mankeu.addChild(kaaudt);  kaakun.addChild(skeu1);  kaaudt.addChild(skeu2);  mansdm.addChild(kapsdm); mansdm.addChild(kaprom);  kapsdm.addChild(ssdm1); kapsdm.addChild(ssdm2);  kaprom.addChild(ssdm3);  direktur.traverseEmp();  }  }   1. Eksekusi program. |
| 1. [tuliskan hasil pengamatan Anda dari penggunaan pattern composite ini] |

**Tutorial 11.4: Kombinasi Composite Observer Pattern**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Buatlah project dengan nama **CompositeObserver** 2. Tutorial ini akan mencoba menunjukkan bagaimana penerapan design pattern composite dan observer dalam satu kesatuan. Kode program dibuat mengkombinasikan tutorial 11.3 dengan penambahan kelas **Asuransi** sebagai **subject**.   Pada tutorial ini, akan diimplementasikan Composite dan Observer Pattern, dengan kelas-kelas yang terlibat adalah sebagai berikut:   |  |  |  | | --- | --- | --- | | No | Kelas dalam Composite & Observer Pattern | Kelas Tutorial | | 1 | Component, abstract observer | Employee | | 2 | Composite, concrete observer | Manager | | 3 | Leaf, concrete observer | Staf | | 4 | Subject | Asuransi | | 5 | Client | CompositeObserverTest |   ----------   1. Modifikasi kelas **Employee** yang ada di tutorial 11.3 dengan menambah kode yang berwarna merah di bawah ini.   public abstract class Employee {  public static StringBuffer space = new StringBuffer();  // atribut-atribut kelas Employee sebagai sebuah konsep  protected String role; // jabatan employee  protected String name;  protected float asuransi; // premi asuransi employee  protected Asuransi model;  public Employee (String nm, String rl, float asr) {  name = nm; role = rl;  asuransi = asr;  }  public Employee (String nm, String rl, float asr, Asuransi mod) {  name = nm; role = rl;  asuransi = asr;  model = mod;  model.attach(this); // Employee sebagai observers mendaftar  // ke Asuransi (sebagai subject)  }  // fungsi-fungsi Employee sebagai abstract component  // dalam pattern composite. Override di kelas turunan  public abstract void addChild(Employee e);  public abstract Employee getChild(int i);  public abstract void traverseEmp();  // fungsi-fungsi getter kelas Employee sebagai sebuah konsep  public String getRole() { return role; }  public String getName() { return name; }  public float getAsuransi() { return asuransi ; }  /////////////////////////////////////////////////////  // atribut kelas Employee sebagai abstract observer  // dalam pattern observer  private Asuransi model;  /////////////////////////////////////////////////////  /////////////////////////////////////////////////////  // fungsi-fungsi kelas Employee sebagai abstract observer  // dalam pattern observer  public abstract void update(float val);  /////////////////////////////////////////////////////  }   1. Modifikasi kelas kelas **Manager** dan kelas **Staf**, dengan menambah kode program yang berwarna merah.   //Manager.java  import java.util.ArrayList;  import java.util.List;  public class Manager extends Employee {  // atribut kelas Manager sebagai composite class dalam  // pattern composite  List<Employee> employees = new ArrayList<Employee>();  public Manager(String nm, String rl){  super(nm, rl, 1000000);  }  public Manager(String nm, String rl, Asuransi mod){  super(nm, rl, 1000000, mod);  }  // fungsi-fungsi kelas Manager sebagai composite class  // dalam pattern composite  public void addChild(Employee e) { // Catat siapa saja yang  employees.add(e); // menjadi bawahan manager  }  public Employee getChild(int i) { // Cari tahu bawahan untuk  return employees.get(i); // seorang manager  }  public void traverseEmp() { // tampilkan semua employee  System.out.println( Employee.space + getName() + "(" +  getRole() + " - " + getAsuransi() + ")" );  Employee.space.append(" ");  for (Employee e : employees)  {  e.traverseEmp();  }  Employee.space.setLength(Employee.space.length() - 3);  }  ///////////////////////////////////////////////////////  // fungsi kelas Manager sebagai concrete observer class  // pattern observer  public void update(float val) {  asuransi = asuransi + asuransi \* val;  }  ///////////////////////////////////////////////////////  }  //Staf.java  public class Staf extends Employee {  public Staf (String nm, String rl){  super(nm, rl, 750000);  }  public Staf (String nm, String rl, Asuransi mod){  super(nm, rl, 750000, mod);  }  // fungsi-fungsi kelas Staf sebagai leaf class dalam  // pattern composite  public void addChild(Employee employee) {} // do nothing. Kelas Staf  // tidak memiliki bawahan  public Employee getChild(int i) {  return null;  }  public void traverseEmp() {  System.out.println( Employee.space + getName() + "(" +  getRole() + " - " + getAsuransi() + ")" );  }  ///////////////////////////////////////////////////////  // fungsi kelas Staf sebagai concrete observer dalam  // pattern composite  public void update(float val) {  asuransi = asuransi + asuransi \* val;  }  }   1. Buat kelas Asuransi sebagai subject dalam pattern observer   import java.util.Vector;  public class Asuransi {  // atribut kelas Asuransi sebagai subject dalam pattern observer  private Vector<Employee> views;  public Asuransi() {  views = new Vector<Employee>();  }  // fungsi-fungsi kelas Asuransi sebagai subject dalam  // pattern observer  public void attach (Employee obs) {  views.add(obs);  }  public void notifyObserver(float val) {  for (int i=0; i < views.size(); i++)  views.elementAt(i).update(val);  }  public void setState(float val) {  notifyObserver(val);  }  }   1. Buatlah kelas **CompositeObserverTest** dengan kode berikut ini:   import java.io.\*;  public class CompositeObserverTest {  public static void main(String[] args) throws IOException {  Asuransi kis = new Asuransi();  Employee direktur = new Manager("Arief Nugraha", "Direktur", kis);  Employee mankeu = new Manager("Suryana", "Manajer Keuangan", kis);  Employee mansdm = new Manager("Susila","Manajer SDM", kis);  Employee kaakun = new Manager("Septarini","Kabag Akuntansi", kis);  Employee kaaudt = new Manager("Herjuna","Kabag Audit", kis);  Employee kapsdm = new Manager("Dewi Yani", "Kabag SDM", kis);  Employee kaprom = new Manager("Eleonora", "Kabag Promosi", kis);  Employee skeu1 = new Staf("Panduwinata", "Staf Keuangan", kis);  Employee skeu2 = new Staf("Megadewi", "Staf Keuangan", kis);  Employee ssdm1 = new Staf("Trifina", "Staf SDM", kis);  Employee ssdm2 = new Staf("Kanda", "Staf SDM", kis);  Employee ssdm3 = new Staf("Arjuna", "Staf SDM", kis);  direktur.addChild(mankeu); direktur.addChild(mansdm);  mankeu.addChild(kaakun); mankeu.addChild(kaaudt);  kaakun.addChild(skeu1);  kaaudt.addChild(skeu2);  mansdm.addChild(kapsdm); mansdm.addChild(kaprom);  kapsdm.addChild(ssdm1); kapsdm.addChild(ssdm2);  kaprom.addChild(ssdm3);  direktur.traverseEmp();  InputStreamReader values = new InputStreamReader (System.in);  BufferedReader br = new BufferedReader(values);  do {  System.out.print("\nInput kenaikan premi [0..1] : ");  String valueStr = br.readLine();  if (valueStr.equalsIgnoreCase("00"))  break;  **kis.setState(Float.parseFloat(valueStr));**  direktur.traverseEmp();  } while(true);  }  }   1. Eksekusi program. |
| 1. Gambarkan relasi kelas (kelas diagram dari kode program tersebut) pada tutorial 11.4 |

**No Kelompok :**

**Anggota :**

|  |  |
| --- | --- |
| **NPM Anggota** | **Nama Anggota** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |