**PRAKTIKUM-5 PDPL**

***STRUCTURAL PATTERN - PROXY***

**Mata Kuliah : Pola Desain Perangkat Lunak**

**Semester : 6 (Enam)**

**Dosen : Tifanny Nabarian, S.Kom., M.T.I.**

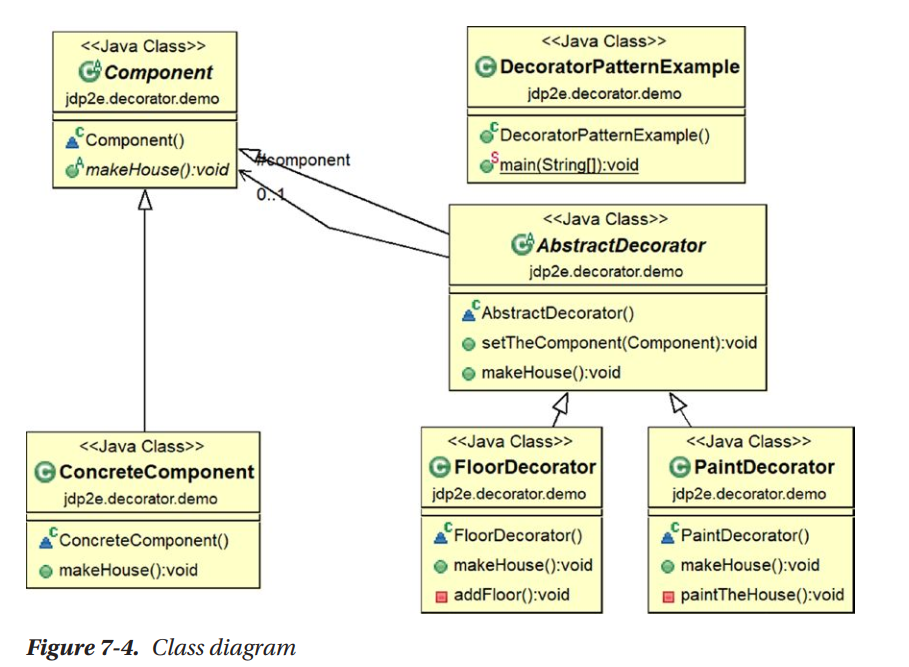
**Nama Mahasiswa : Muhammad yazid supriadi**

**NIM : 0110217042**

**Instruksi**

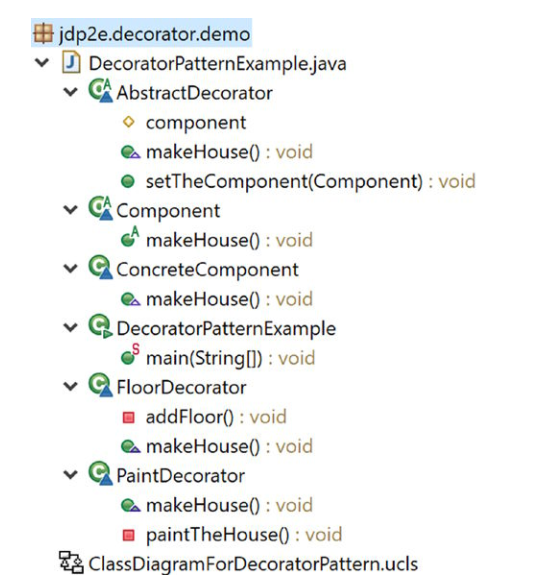
* Kerjakan tugas di bawah ini secara **individu.**
* Kerjakan secara **berurutan.**
* Modifikasi *source code*, dengan cara menambahkan **NIM** Anda pada setiap *caption* yang ditampilkan di *result* yang di*screenshot*.

1. **Perhatikan ilustrasi berikut ini:**



Gambar 1. Ilustrasi Relasi

1. **Pahamilah struktur package yang akan dibuat.**



Gambar 2. Package Explorer

1. **Buatlah Kelas Component dan ConcreteComponent dengan source sebagai berikut:**

**Component.java**

|  |
| --- |
| **package** jdp2e.decorator.demo;  **abstract** **class** Component {  **public** **abstract** **void** makeHouse();  } |

**ConcreteComponent.java**

|  |
| --- |
| **package** jdp2e.decorator.demo;  **public** **class** ConcreteComponent **extends** Component{  **public** **void** makeHouse()  {  System.***out***.println("Original House is complete. It is closed for modification.");  }  } |

1. **Lanjutkan dengan membuat kelas AbstractDecorator, FloorDecorator, dan PaintDecorator dengan sourcode sbb.,**

**AbstractDecorator.java**

|  |
| --- |
| **package** jdp2e.decorator.demo;  **public** **abstract** **class** AbstractDecorator {  **protected** Component component ;    **public** **void** setTheComponent(Component c)  {  component = c;  }    **public** **void** makeHouse()  {  **if** (component != **null**)  {  component.makeHouse();//Delegating the task  }  }  } |

**FloorDecorator.java**

|  |
| --- |
| **package** jdp2e.decorator.demo;  **public** **class** FloorDecorator **extends** AbstractDecorator  {  **public** **void** makeHouse()  {  **super**.makeHouse();  //Decorating now.  System.***out***.println("\*\*\*Floor decorator is in action\*\*\*");  addFloor();  /\*You can put additional stuffs as per your need\*/  }    **private** **void** addFloor()  {  System.***out***.println("I am making an additional floor on top of it.");  }  } |

**PaintDecorator.java**

|  |
| --- |
| **package** jdp2e.decorator.demo;  **public** **class** PaintDecorator **extends** AbstractDecorator {  **public** **void** makeHouse()  {  **super**.makeHouse();    //Decorating now.  System.***out***.println("\*\*\*Paint decorator is in action now\*\*\*");  paintTheHouse();  //You can add additional stuffs as per your need  }  **private** **void** paintTheHouse()  {  System.***out***.println("Now I am painting the house.");  }  } |

1. **Lanjutkan dengan membuat kelas client -> DecoratorPatternExample, dengan source sbb.:**

**DecoratorPatternExample.java**

|  |
| --- |
| **package** jdp2e.decorator.demo;  **public** **class** DecoratorPatternExample {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  System.***out***.println("\*\*\*Decorator pattern Demo\*\*\*\n");  ConcreteComponent withoutDecorator = **new** ConcreteComponent();  withoutDecorator.makeHouse();    System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");    //Using a decorator to add floor  System.***out***.println("Using a Floor decorator now.");  FloorDecorator floorDecorator = **new** FloorDecorator();  floorDecorator.setTheComponent(withoutDecorator);  floorDecorator.makeHouse();    System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");    //Using a decorator to add floor to original house and then  //paint it.  System.***out***.println("Using a Paint decorator now.");  PaintDecorator paintDecorator = **new** PaintDecorator();  //Adding results from floor decorator  paintDecorator.setTheComponent(withoutDecorator);  paintDecorator.makeHouse();  System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  }  } |

1. **Update source pada langkah 5, dengan menampilkan NIM Anda. Kemudian jalankan *main* program tersebut.**

|  |
| --- |
| *Screenshot Hasil Running Main Program* |

1. Berikan penjelasan terkait proses implementasi *Decorator* pada source 3,4 dan 5.

|  |
| --- |
| **Pada proses ke 3 pertama tama kita membuat sebuah abstract class component dengan class abstract dan juga kita membuat sebuah class ConctreteComponent yang mengambil extends dari class Component yang dimana pada class ConcreteComponent kita membuat sebuah method makeHouse()**  **Pada proses ke 4 kita membuat sebuah AbstractDecorator , pada class ini mengambil dari class Component . dengan component kita membuat sebuah method makeHouse(). Jika componentnya belum ada maka akan dibuat sebuah componentnya.**  **Lalu kita membuat class FloorDecoration dan PaintDecoration yang mengambil class dari AbstractDecorator . disini kita memakai super untuk merepresantikan objek dari class indeknya.**  **Pada proses ke 5 kita membuat sebuah class client bernama DecoratorPatternExample untuk mengimplementasikan pada client dan juga mencetak output pada fungsi client main ini** |

1. Menurut pendapat Anda setelah mengimplementasikan *sourcode* di atas, apa perbedaan mendasar dari metode Structural Pattern **Adapter** dan Structural Pattern **Proxy,** dan **Decorator**?

|  |
| --- |
| **Pada adapter mengubah interface servicenya mempertahankan kebiasaan pada servicenya .**  **Pada proxy mengubah kebiasaan pada service tapi mempertahankan interface pada servicenya.**  **Dengan kata lain adapter menyediakan interface yang berbeda dan proxy menyediakan interface yang sama**  **Namun pada decorator dia bisa menambah interface tambahan. Tentu bisa menambahkan method tambahan juga pada interfacenya.** |

**\*\*\***