**PRAKTIKUM-5 PDPL**

***STRUCTURAL PATTERN - PROXY***

**Mata Kuliah : Pola Desain Perangkat Lunak**

**Semester : 6 (Enam)**

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

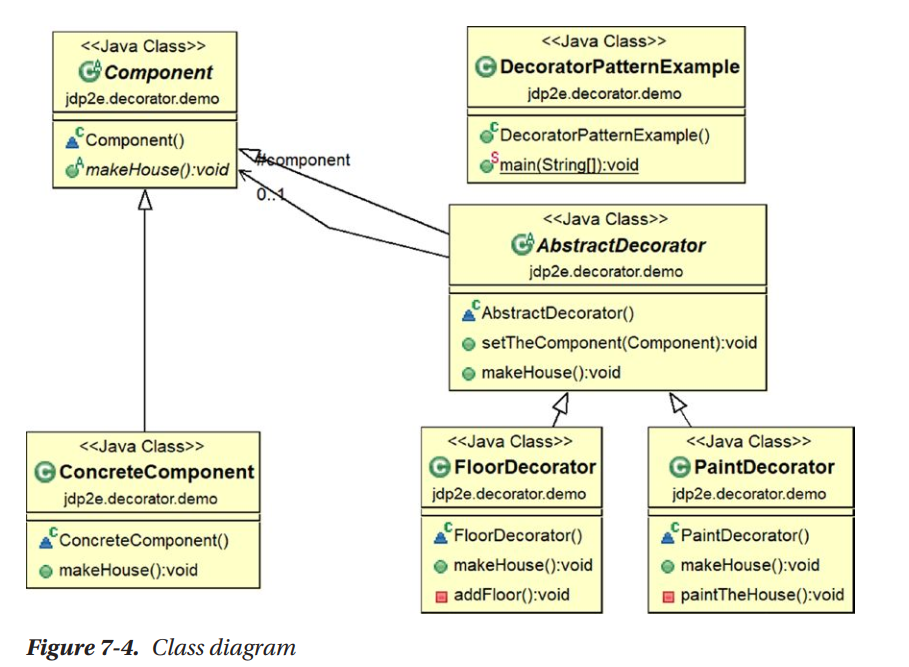
**Nama Mahasiswa : Chairil Hilman Syah**

**NIM : 0110217054**

**Instruksi**

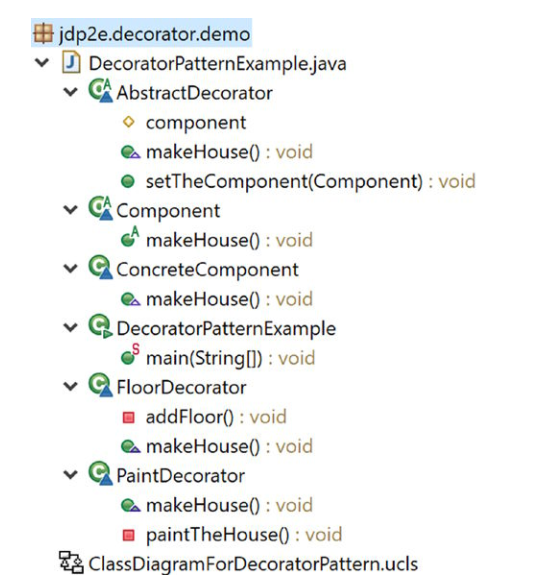
* Kerjakan tugas di bawah ini secara **individu.**
* Kerjakan secara **berurutan.**
* Modifikasi *source code*, dengan cara menambahkan **NIM** Anda pada setiap *caption* yang ditampilkan di *result* yang di*screenshot*.

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.");  }  } |
|  |

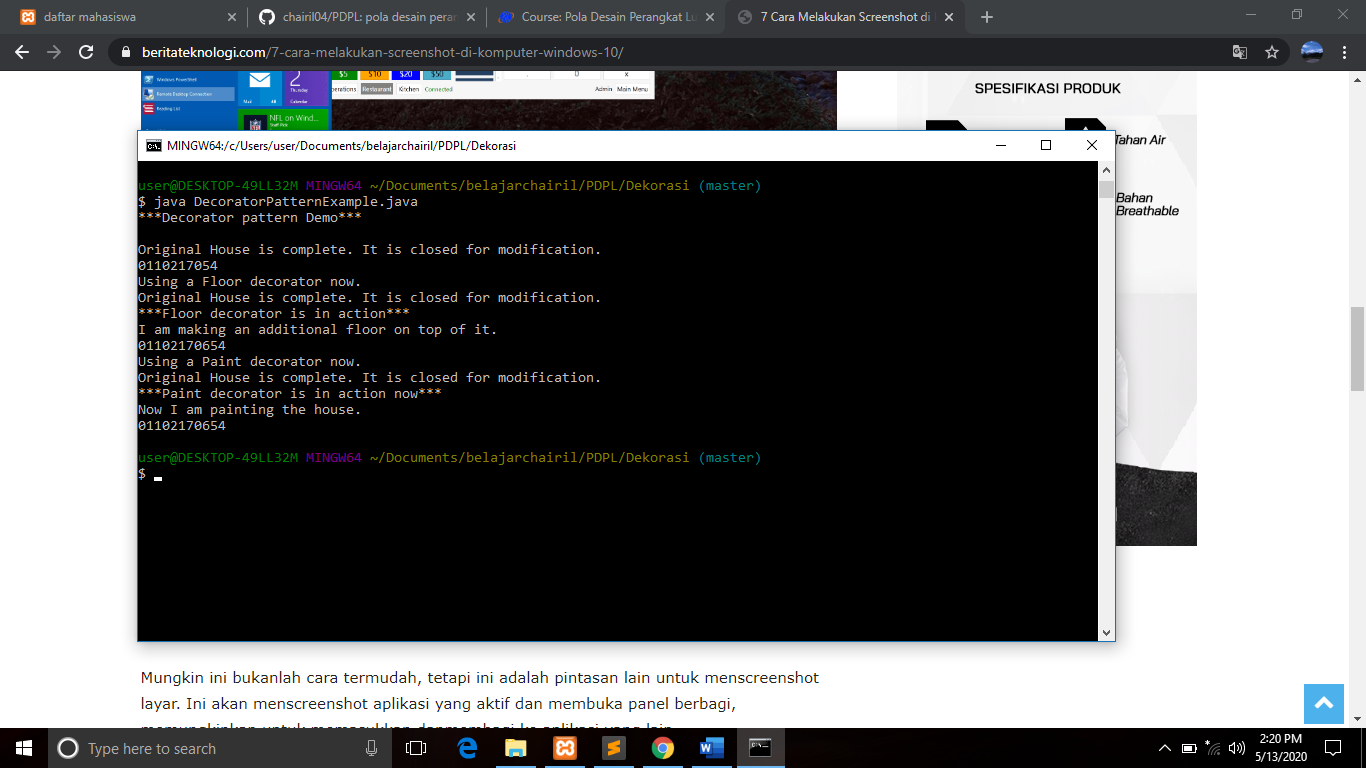
**PaintDecorator.java**

|  |
| --- |
| **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.**
2. Berikan penjelasan terkait proses implementasi *Decorator* pada source 3,4 dan 5.

|  |
| --- |
| **Penjelasan pertahapan :**   1. Pada tahapan ke 3, di sini kita membuat sebuah kelas yang apabila dalam analogi kehidupannya adala sebuah rumah nya lengkap dengan pengukuhan bahwa rumah ini tidak dapat dirubah. 2. Pada tahap ke 4, di sini hadir solusi untuk memodifikasi rumah tersebut yaitu dengan cara dekorasi yakni kita melakukan apa yang disebut dengan wrapping atau pembungkusan pada hal hal tambahan. Dan hal-hal yang tidak dapat di modifikasi kita bungkus dalam basedecoration dan setelah itu dilangkah tersebut kita membuat beberapa massage/tambahan yang akan kita lakukan. 3. Pada tahapan ke 5, tahap ini adala program utama yang akan kita eksekusi dan percobaan apakah decoration yang kita buat akan berjalan atau tidak. |

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

|  |
| --- |
| Semua pola melibatkan pembungkus bagian dalam objek/kelas dengan luar, sehingga mereka sangat mirip secara struktural. Namun perbedaan mendasar mnurut saya dan menurut apa yang say abaca dari sumber lain, yaitu:   1. Adaptor mengubah antarmuka dari dalam ke luar. 2. Proxy merangkum akses di luar ke dalam. 3. Dekorator memodifikasi atau memperluas perilaku dalam dengan luar. |

**\*\*\***