**PRAKTIKUM-5 PDPL**

***STRUCTURAL PATTERN - PROXY***

**Mata Kuliah : Pola Desain Perangkat Lunak**

**Semester : 6 (Enam)**

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

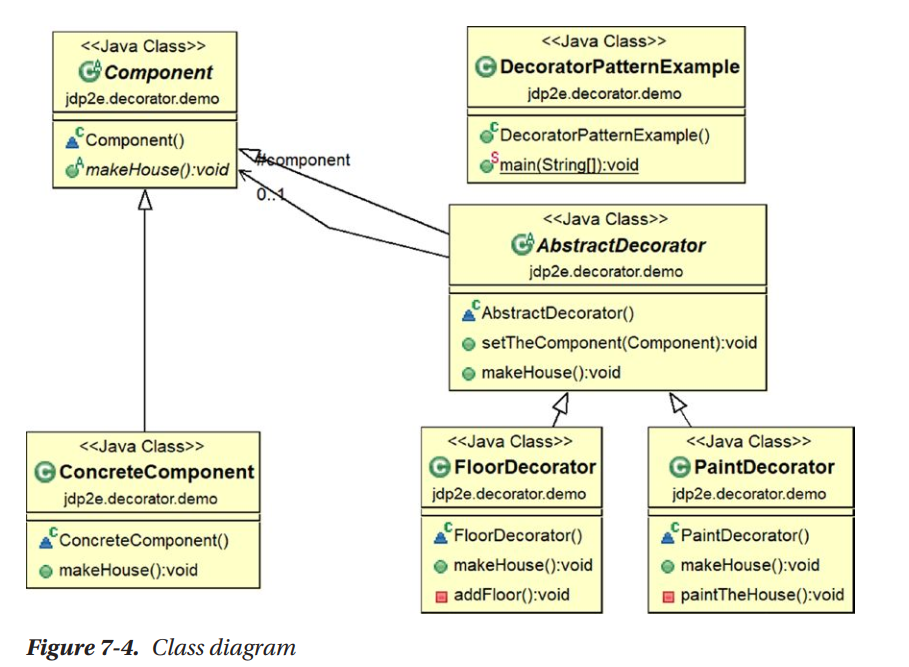
**Nama Mahasiswa : Muhammad Azhar Rasyad**

**NIM : 0110217029**

**Instruksi**

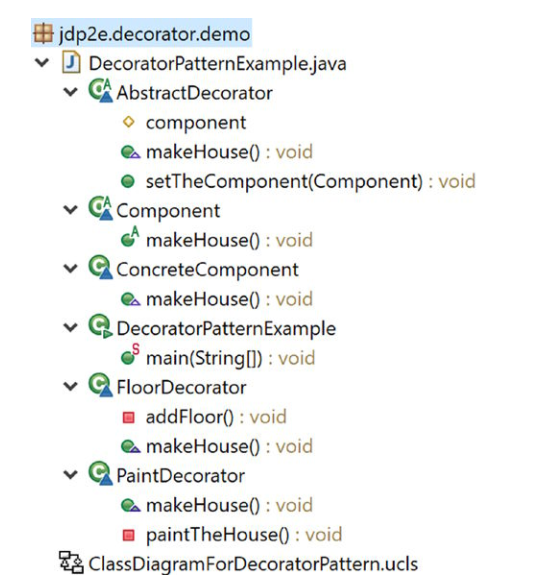
* Kerjakan tugas di bawah ini secara **individu.**
* Kerjakan secara **berurutan.**
* Modifikasi *source code*, dengan cara menambahkan **NIM** Anda pada setiap *caption* yang ditampilkan di *result* yang di*screenshot*.

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.

|  |
| --- |
| Contoh decorator yang digunakan kali ini yaitu dalam pembuatan sebuah rumah, berikut code awal yang digunakan dalam membuat sebuah rumah.    Terlihat pada kode di atas dalam membuat rumah menggunakan kelas ConcreteComponent dengan method makeHouse(), berikut kode dari method tersebut    Kode di atas hanyalah sebuah ilustrasi, akan tetapi intinya jika method tersebut dipanggil maka dapat membuat sebuah rumah, sehingga outputnya akan seperti berikut    Langkah selanjutnya adalah mendekorasi rumah tersebut, dalam contoh kali ini hanya mendekorasi dalam menambahkan lantai (**Floor**) dan mengecat rumah (**Paint**). Dekorasinya dimulai dari menambahkan lantai dengan menggunakan kode berikut    Untuk menambahkan lantai perlu memanggil kelas FloorDecorator dan memasangnya ke rumah yang telah dibuat sebelumnya dengan method setTheComponent() yang berada pada kelas AbstractDecorator karena kelas FloorDecorator mengextend pada kelas tersebut sehingga saling terhubung.    Isi dari method setTheComponent() sebagai berikut    Terlihat kode di atas mengambil fungsi dari kelas Component yang berisi abstrak, supaya banyak komponen rumah yang dapat ditampung dan tidak hanya menambahkan lantai saja    Kembali ke bagian berikut    Setelah diset kedalam komponen maka selanjutnya memanggil rumah yang sebelumnya dibuat yaitu withoutDecorator sehingga setTheComponent(withoutDecorator) maka dekorasi ditujukan untuk rumah tersebut.  Langkah selanjutnya adalah eksekusi dengan menggunakan makeHouse() akan tetapi method tersebut bukan berarti membuat rumah baru lagi melainkan agar rumah tersebut masih dapat digunakan meskipun adanya dekorasi dengan kode berikut    Terlihat pada kode di atas method makeHouse() menggunakan super yang artinya memanggil method makeHouse() yang asli atau dalam hal ini ibaratnya rumah yang asli masih dapat digunakan dan selanjutnya menambahkan lantai dengan method addFloor() sehingga outputnya akan terlihat seperti berikut    Sampai sini artinya rumah yang sebelumnya baru dibuat masih dapat digunakan ditambah dengan dekorasi menambahkan lantai. Jika sudah selesai memasang lantai maka selanjutnya mengecat rumah tersebut dengan kode berikut    Kode di atas sama seperti mendekorasi lantai sebelumnya, yang berbeda hanya method untuk mengecat yaitu    Kode di atas hampir sama seperti mendekorasi lantau sebelumnya dan hanya berbeda pada method paintTheHouse(), sehingga hasilnya seperti berikut    Jika sudah selesai, akhirnya rumah tersebut telah berhasil dibuat beserta dengan dekorasi-dekorasinya yaitu menambahkan lantai dan mencat rumah tersebut. |

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

|  |
| --- |
| Menurut saya, Adapter dan Proxy masih memiliki kesamaan akan tetapi jika dibandingkan dengan Decorator menurut saya terlalu berbeda dengan alasan berikut:  Jikalau Adapter menggantikan suatu task, maka task yang digantikan tersebut memang tidak bisa menjalankan task tersebut akan tetapi task tersebut masih berjalan.  Jikalau Proxy menggantikan suatu task ketika suatu task tersebut sedang tidak ada dan bukan tidak bisa menjalankan task tersebut.  Jikalau Decorator menambahkan suatu komponen ke dalam object tanpa mengganggu object tersebut dan bukannya menggantikan sebuah task sebagaimana Adapter dan Proxy. |

**\*\*\***