**PRAKTIKUM-5 PDPL**

***STRUCTURAL PATTERN - PROXY***

**Mata Kuliah : Pola Desain Perangkat Lunak**

**Semester : 6 (Enam)**

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

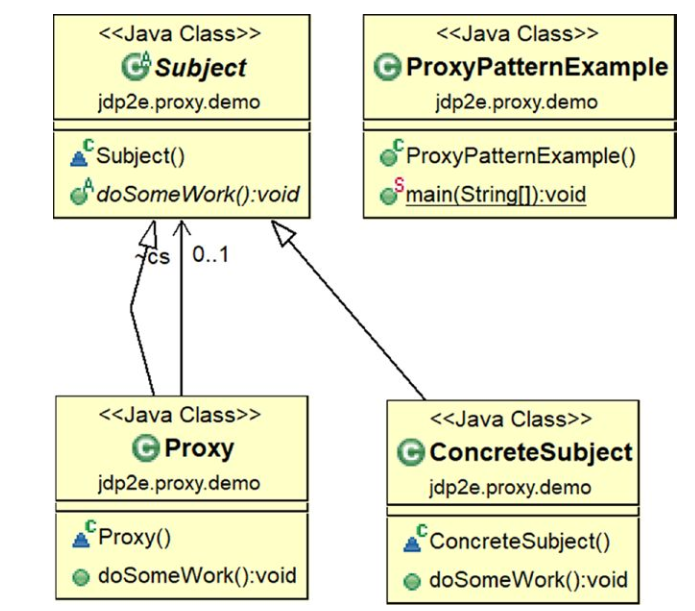
**Nama Mahasiswa : Muhammad Yazid Supriadi**

**NIM : 0110217042**

**Instruksi**

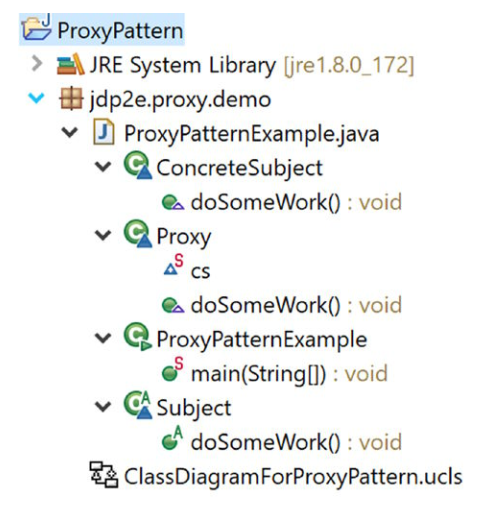
* Kerjakan tugas di bawah ini secara **individu.**
* Kerjakan secara **berurutan.**
* Modifikasi *source code*, dengan cara menambahkan **NIM** Anda pada setiap *caption* yang ditampilkan di *result* yang di*screenshot*.

1. **Perhatikan ilustrasi berikut ini:**



Gambar . Ilustrasi Relasi

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



Gambar . Package Explorer

1. **Buatlah Kelas Subject dan ConcreteSubject dengan source sebagai berikut:**

**Subject.java**

|  |
| --- |
| **package** jdp2e.proxy.demo;  **public** **abstract** **class** Subject {  **public** **abstract** **void** doSomeWork();  } |

**ConcreteSubject.java**

|  |
| --- |
| **package** jdp2e.proxy.demo;  **public** **class** ConcreteSubject **extends** Subject{  @Override  **public** **void** doSomeWork()  {  System.***out***.println("doSomeWork() inside ConcreteSubject is invoked.");  }  } |

1. **Lanjutkan dengan membuat kelas Proxy dengan sourcode sbb.,**

**Proxy.java**

|  |
| --- |
| **package** jdp2e.proxy.demo;  **public** **class** Proxy **extends** Subject{  **static** Subject *cs*;  @Override  **public** **void** doSomeWork()  {  System.***out***.println("Proxy call happening now...");  //Lazy initialization:We'll not instantiate until the method is  //called  **if** (*cs* == **null**)  {  *cs* = **new** ConcreteSubject();  }  *cs*.doSomeWork();  }  } |

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

**ProxyPatternExample.java**

|  |
| --- |
| **package** jdp2e.proxy.demo;  **public** **class** ProxyPatternExample {  **public** **static** **void** main(String[] args)  {  System.***out***.println("\*\*\*Proxy Pattern Demo\*\*\*\n");  Proxy px = **new** Proxy();  px.doSomeWork();  }  } |

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 *Proxy* pada source 3,4 dan 5.

|  |
| --- |
| **Proses proxy terjadi dimana client tidak harus mengakses langsung ke sebuah object . akan tetapi dengan banyaknya nanti client yang mengakses hanya perlalu mengakses proxy tanpa harus mengakses langsung ke object. Sebagai contoh ketika kita ingin menyetor uang ke bank kita tidak harus membawa banyak uang tersebut ke bank kita bisa menggunakan sebauh credit card ke bank . sehingga efisien .**  **Pada source ke 3 pertama kita membuat sebuah class yang bernamanya object dan juga kita membuat sebuah class yang bernama ConcreteSubject yang melakukan overriding pada class subject.**  **Lalu pada proses ke 4 kita membuat sebuah class proxy yang mengamnil extends dari class subject yang mana di class subject terdapat doSemeWork() method.dan pada proses ke 4 ini kita mendfinisikan fungsi doSomeWork() , dan mengecek apakah ada concrete subject kalo belum ada maka kita membuat sebuah concrete subject jika sudah ada kita jalankan method doSomeWork().**  **Pada proses ke 5 merupakan sisi client ketika ingin mengakses subject maka kita hanya perlu memanggil proxynya saja** |

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

|  |
| --- |
| **Pada proxy proxy mengubah kebiasaan pada servicenya dan mengubah interfacenya. Sedangkan pada adapter mengubah interface servicenya dan mempertahankan kebiasaan pada servicenya.**  **Pada proxy request tidak diakses secara langsung. Akan tetapi object target yang memproses request. Sedangkan pada adapter juga request tidak diakses secara langsung akan tetapi pada adapter di redirect terlebih dahulu ke adaptee dan memproses request tersebut via adaptee saja.**  **Mungkin itu terkait perbedaan yang ada pada proses deigin pattern proxy dan adapter. Sedangkan ada satu persamaan anatar keduanya yaitu keduanya sama sama menambahkan layer pada design patternya** |

**\*\*\***