**PRAKTIKUM-3 PDPL**

***CREATIONAL PATTERN - BUILDER***

**Mata Kuliah : Pola Desain Perangkat Lunak**

**Semester : 6 (Enam)**

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

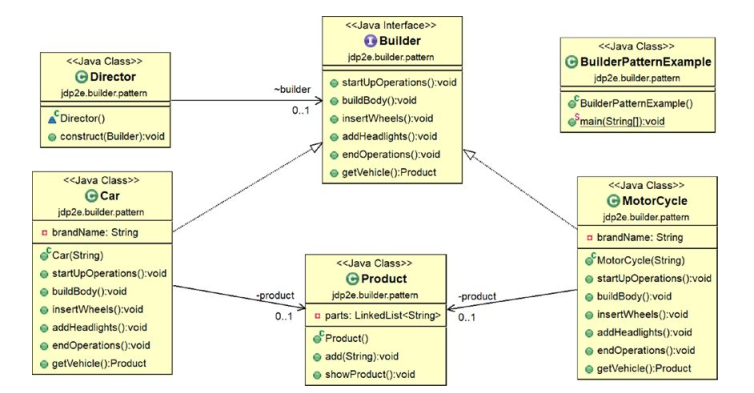
**Nama Mahasiswa : Muhammad Azhar Rasyad**

**NIM : 0110217029**

**Instruksi**

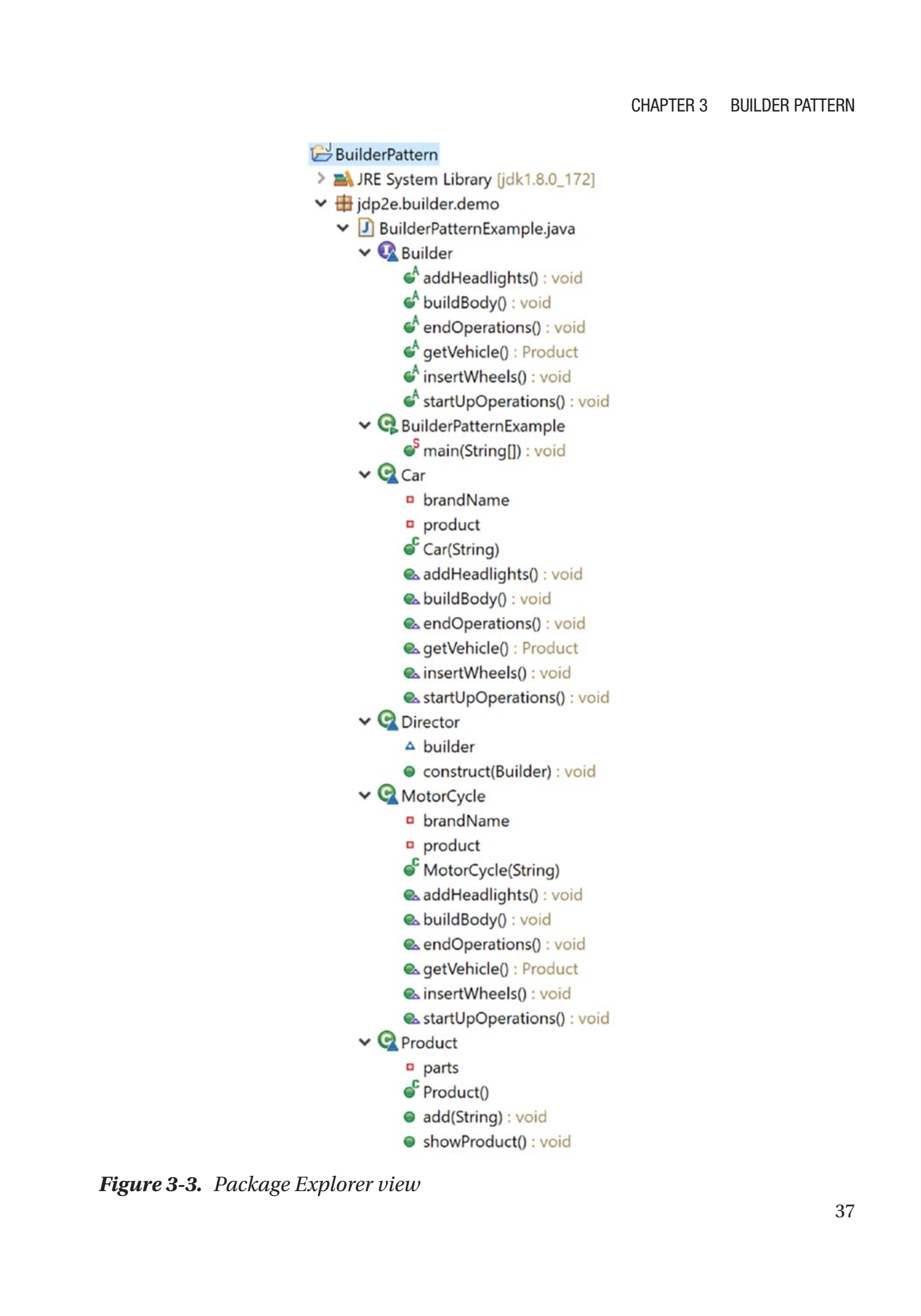
* Kerjakan tugas di bawah ini secara **individu.**
* Kerjakan secara **berurutan.**
* Modifikasi *source code*, dengan cara menambahkan **NIM** Anda pada setiap *caption* yang ditampilkan di *result* yang di*screenshot*.

1. **Perhatikan ilustrasi berikut ini:**



Gambar 1. Ilustrasi Relasi

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



Gambar 2. Package Explorer

1. **Buatlah interface Builder dengan source sebagai berikut:**

**Builder.java**

|  |
| --- |
| **package** jdp2e.builder.demo;  //The common interface  **interface** Builder  {  **void** startUpOperations();  **void** buildBody();  **void** insertWheels();  **void** addHeadlights();  **void** endOperations();    /\*The following method is used to retrieve the object that is  constructed.\*/  Product getVehicle();  } |

\***Warning**: akan terdapat *error* pada line Product … (hilangkan error dengan melanjutkan ke langkah selanjutnya)

1. **Lanjutkan dengan membuat kelas Product, Car, Motorcycle dan Director, dengan masing-masing sourcode sbb.,**

**Product.java**

|  |
| --- |
| **package** jdp2e.builder.demo;  **import** java.util.LinkedList;  **public** **class** Product {  /\* You can use any data structure that you prefer.  I have used LinkedList<String> in this case.\*/  **private** LinkedList<String> parts;  **public** Product()  {  parts = **new** LinkedList<String>();  }    **public** **void** add(String part)  {  //Adding parts  parts.addLast(part);  }  **public** **void** showProduct()  {  System.***out***.println("\nProduct completed as below :");  **for** (String part: parts)  System.***out***.println(part);  }  } |

**Car.java**

|  |
| --- |
| **package** jdp2e.builder.demo;  **public** **class** Car **implements** Builder{  **private** String brandName;  **private** Product product;  **public** Car(String brand)  {  product = **new** Product();  **this**.brandName = brand;  }  **public** **void** startUpOperations()  {  //Starting with brand name  product.add(String.*format*("Car model is :%s",**this**.brandName));  }  **public** **void** buildBody()  {  product.add("This is a body of a Car");  }  **public** **void** insertWheels()  {  product.add("4 wheels are added");  }  **public** **void** addHeadlights()  {  product.add("2 Headlights are added");  }  **public** **void** endOperations()  { //Nothing in this case  }  **public** Product getVehicle()  {  **return** product;  }  } |

**Motorcycle.java**

|  |
| --- |
| **package** jdp2e.builder.demo;  **public** **class** Motorcycle **implements** Builder {  **private** String brandName;  **private** Product product;    **public** Motorcycle(String brand)  {  product = **new** Product();  **this**.brandName = brand;  }    **public** **void** startUpOperations()  { //Nothing in this case  }  **public** **void** buildBody()  {  product.add("This is a body of a Motorcycle");  }  **public** **void** insertWheels()  {  product.add("2 wheels are added");  }  **public** **void** addHeadlights()  {  product.add("1 Headlights are added");  }  **public** **void** endOperations()  {  //Finishing up with brand name  product.add(String.*format*("Motorcycle model is :%s", **this**.brandName));  }  **public** Product getVehicle()  {  **return** product;  }  } |

**Director.java**

|  |
| --- |
| **package** jdp2e.builder.demo;  **public** **class** Director {  Builder builder;  // Director knows how to use the builder and the sequence of steps.  **public** **void** construct(Builder builder)  {  **this**.builder = builder;  builder.startUpOperations();  builder.buildBody();  builder.insertWheels();  builder.addHeadlights();  builder.endOperations();  }  } |

1. **Lanjutkan dengan membuat kelas PrototypePatternExample, dengan source sbb.:**

**BuilderPatternExample.java**

|  |
| --- |
| **package** jdp2e.builder.demo;  **public** **class** BuilderPatternExample {  **public** **static** **void** main(String[] args)  {  System.***out***.println("\*\*\*Builder Pattern Demo\*\*\*");  Director director = **new** Director();  Builder fordCar = **new** Car("Ford");  Builder hondaMotorycle = **new** Motorcycle("Honda");    // Making Car  director.construct(fordCar);  Product p1 = fordCar.getVehicle();  p1.showProduct();    //Making MotorCycle  director.construct(hondaMotorycle );  Product p2 = hondaMotorycle.getVehicle();  p2.showProduct();  }  } |

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 *builder* pada source 3,4 dan 5.

|  |
| --- |
| Saya akan menjelaskan proses implementasi *builder* mulai dari *main program* hingga output yang ditampilkan, berikut penjelasan saya :  **BuilderPatternExample.java**, source ini digunakan sebagai *main program* untuk dijalankan pertama kali dan memanggil source yang lainnya. Pada source ini memanggil seluruh source lainnya mulai dari Director, Builder, Car, Motorcycle, dan Product untuk diimplementasikan. Berikut detail penjelasan source ini :   * Director director = new Director(), kode tersebut digunakan untuk membuat objek yaitu director dari kelas Director.java dan memanggil method constructnya   + **Director.java**, source ini digunakan untuk mengarahkan objek yang dibuat dari kelas Builder.java dan memanggil method-method pada buildernya * Builder fordCar = new Car(), kode tersebut digunakan untuk membuat objek yaitu fordCar dari kelas Builder.java dan memanggil method Car, hal tersebut dapat dilakukan karena pada kelas Car telah mengimplement kelas Builder.java sehingga saling terhubung   + **Builder.java**, source ini hanya sebagai perantara antara kelas Car.java dengan Director.java agar dapat memanggil method-method yang ingin dibuat   + **Car.java**, source ini merupakan proses utama dalam pembuatan product Car karena seluruh methodnya berada di source ini mulai dari proses ketika dibuat, pemasangan bahan-bahannya, hingga product dibuat. Khusus untuk proses membuat product menggunakan kelas **Product.java** melalui kelas Builder.java     - **Product.java**, source ini digunakan untuk menampilkan setiap method yang telah dibuat dari kelas Car.java sehingga setiap methodnya diproses dan dijadikan part menggunakan LinkedList * Builder hondaMotorcycle = new Motorcycle(), kode tersebut digunakan untuk membuat objek yaitu hondaMotorcycle dari kelas Builder.java dan memanggil method Motorcycle, hal tersebut sama seperti sebelumnya dapat dilakukan karena pada kelas Motorcycle telah mengimplement kelas Builder.java sehingga saling terhubung   + **Motorcycle.java**, sama seperti source Car.java namun ada beberapa perbedaan dari nilai methodnya * director.construct(fordCar), kode tersebut pada objek director memanggil method constructnya dengan memasukkan nilai objek fordCar yang nantinya diproses pada kelas Director.java seperti penjelasan diatas * Product p1 = fordCar.getVehicle(), kode ini untuk membuat objek p1 dari kelas Product.java dengan memanggil objek fordCard dan method getVehicle() yang mengembalikan nilai product * p1.showProduct(), kode tersebut untuk menampilkan hasil dari method-method yang dibuat pada objek fordCar sebelumnya dari kelas Car.java sehingga dalam membuat Car selesai * Untuk penjelasan selanjutnya mengenai objek hondaMotorcycle sama seperti fordCar akan tetapi ada perbedaan antara Car dan Motorcycle   Inti dari penjelasan diatas BuilderPatternExample memanggil Director yang kemudian memanggil Builder, kemudian Builder memanggil Car serta Motorcycle dan terakhir memanggil Product, sehingga terbuatlah Product |

1. Berikan penjelasan terkait peran kelas **Director**!

|  |
| --- |
| **Peran kelas Director untuk memanggil method pada kelas Car dan Motorcycle menggunakan tipe data Builder yang juga memanggil interface Builder.java** |

1. Berikan penjelasan terkait peran kelas **Builder**!

|  |
| --- |
| **Peran kelas Builder yaitu menyusun method-method dari kelas Car dan Motorcycle yang kemudian akan ditampilkan pada kelas Product.java** |

1. Menurut pendapat Anda setelah mengimplementasikan *sourcode* di atas, apa kelemahan dari implementasi metode Builder?

|  |
| --- |
| **Menurut saya terdapat redudansi antara kelas Car dan Motorcycle sehingga menuliskan method yang sama akan tetapi hanya berbeda isinya, hal tersebutlah yang menurut saya merupakan kelemahan implementasi metode Builder yang apabila ingin menambahkan product baru perlu menuliskan ulang method-methodnya meskipun method tersebut hampir sama dengan product yang lain** |

**\*\*\***