LAPORAN PRAKTIKUM

NIM	**:**	2003073
Nama	**:**	Ica Natasya
Kelas	**:**	D3TI.2C
Mata Kuliah	**:**	**Pemrograman Perangkat Bergerak (TIU3403)**
Praktikum ke / Judul	**:**	5/ SOLID Programming Dart
Tanggal Praktikum	**:**	21 Maret 2022
Dosen Pengampu	**:**	Fachrul Pralienka Bani Muhamad, S.ST., M.Kom

PROGRAM STUDI D3 TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
POLITEKNIK NEGERI INDRAMAYU
2022

NIM : 2003073
Nama : Ica Natasya

PRAKTIKUM 5
SOLID PROGRAMMING DART

5.1. TUJUAN PRAKTIKUM

Tujuan Umum
Mahasiswa menjelaskan prinsip SOLID serta mampu membuat program dengan prinsip SOLID

Tujuan Khusus
Mahasiswa dapat
1. Menjelaskan prinsip SOLID secara umum
2. Membuat program *Single Responsibility Principle* dengan Dart
3. Membuat program *Open Closed Principle* dengan Dart
4. Membuat program *Liskov Substitution Principle* dengan Dart
5. Membuat program *Interface Segregation Principle* dengan Dart
6. Membuat program *Dependency Inversion Principle* dengan Dart

5.2. TEORI SINGKAT
SOLID adalah sebuah akronim dari lima prinsip *object-oriented design* (OOD) oleh Robert C. Martin. Prinsip ini biasa diterapkan dalam pemrograman berorientasi objek. Kelima prinsip ini mengembangkan sebuah program dengan mempertimbangkan pemeliharaan serta pengembangan lebih lanjut agar kode mudah dirawat, mudah dimengerti serta fleksibel. Tujuan prinsip SOLID adalah untuk membantu *programmer* dalam menghindari *bad code*, membantu dalam *refactoring* kode program serta mengembangkan aplikasi secara *Agile* atau *Adaptive*.

Manfaat Prinsip SOLID :
1. Prinsip SOLID adalah salah satu fondasi utama dalam mengembangkan kode program agar lebih mudah dimengerti, dikelola, dan dikembangkan.
2. Prinsip SOLID mampu diterapkan di banyak bahasa yang berparadigma OOP.
3. Banyak perusahaan mensyaratkan pemahaman prinsip SOLID ketika mencari seorang *developer*.
4. Menerapkan prinsip SOLID akan menghindarkan dari membuat aplikasi yang mudah memiliki bug karena penerapan desain yang buruk.

Adapun singkatan dari prinsip SOLID yaitu:
S - Single responsibility principle (SRP)
O - Open closed principle (OCP)
L - Liskov substitution principle (LSP)
I - Interface segregation principle (ISP)
D - Dependency inversion principle (DIP)

Single Responsibility Principle

Single Responsibility Principle adalah adalah prinsip pertama dari 5 prinsip desain SOLID. Prinsip ini menyatakan bahwa kelas atau objek harus memikul tanggung jawab tunggal dalam fungsionalitas keseluruhan program. Seperti yang dijelaskan oleh Martin dalam *Agile Software Development, Principles, Patterns, and Practices*, bahwa tanggung jawab dapat didefinisikan sebagai “poros perubahan”. Memiliki banyak tanggung jawab berarti memiliki banyak alasan untuk berubah yang berarti potensi yang lebih tinggi untuk dilanggar saat basis kode seseorang berkembang. Ilustrasi analogi single responsibility principle dapat dilihat pada Gambar 1.

Gambar 1. Ilustrasi Analogi Pelanggaran *Single Responsibility Principle*

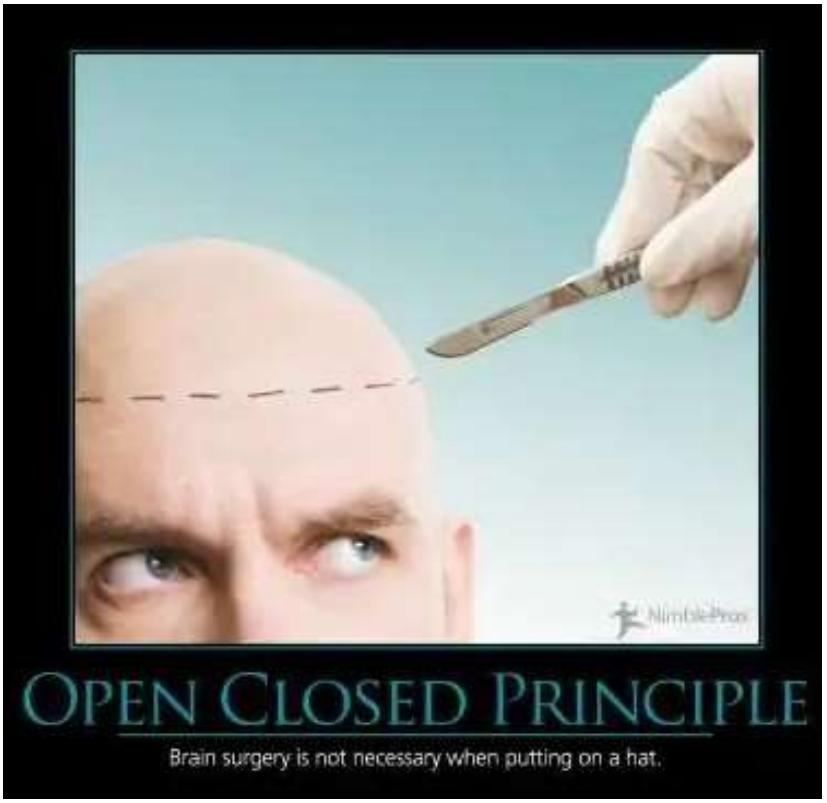
Dijelaskan bahwa pada Gambar 1 tidak boleh ada objek yang memiliki lebih dari satu tanggung jawab, misalnya pisau multi fungsi dan sendok yang sekaligus menjadi garpu. Hal ini bertujuan untuk menyederhanakan proses pemeliharaan, serta mengurangi potensi kesalahan apabila perlu dilakukan perubahan suatu rancangan objek (class) di kemudian waktu. Pendekatan umum untuk mempersempit tanggung jawab adalah dengan memanfaatkan komponen seperti antarmuka untuk mengabstraksikan fungsionalitas tertentu ke dalam kelas yang terpisah. Sebagai contoh, perhatikan implementasi class diagram yang disajikan pada Gambar 2.

Gambar 2. Ilustrasi *Class* Diagram Pelanggaran *Single Responsibility Principle*

Dapat dilihat pada Gambar 2 adalah contoh class diagram yang melanggar *single responsibility principle*, karena dalam class tersebut terdapat lebih dari 1 (satu) tanggung jawab yaitu *method* hitungLuas() harusnya terdapat dalam class terpisah misalnya class `PerhitunganPersegi`.

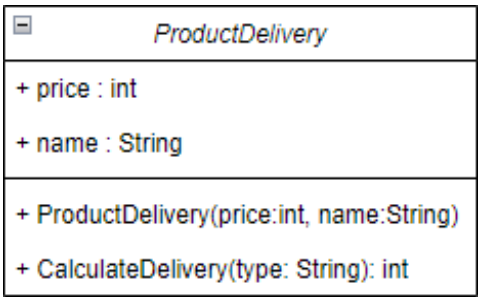
Open-closed Principle

Open-Closed Principle menentukan bahwa entitas OOP harus terbuka untuk ekstensi, tetapi ditutup untuk modifikasi. Tujuan dari prinsip ini adalah membuat komponen yang ada tahan terhadap perubahan persyaratan. Sebagai ilustrasi *Open-closed Principle*, dianalogikan dengan objek Topi dan objek Kepala, yaitu untuk memasangkan objek Topi, tidak perlu dilakukan operasi (modifikasi) terhadap objek Kepala (lihat Gambar 3).



Gambar 3. Ilustrasi Analogi Pelanggaran *Open-Closed Principle*

Untuk mengimplementasikan *Open-Closed Principle*, beberapa konsep tersedia dalam bahasa OOP, termasuk abstraksi, antarmuka, dan generik. Dengan menggunakan ini, dimungkinkan untuk memisahkan dasar-dasar dari yang spesifik. Terbuka untuk Ekstensi berarti bahwa implementasi baru dapat ditambahkan untuk memperluas fungsionalitas tertentu, sedangkan Tertutup untuk Modifikasi berarti bahwa fungsionalitas dasar suatu komponen tidak akan berubah.



Gambar 4. Ilustrasi *class* diagram pelanggaran *Open closed principle*

Pada Gambar 4 dijelaskan bahwa terdapat suatu rancangan *class* diagram `ProductDelivery` yang memiliki variabel `price` dan `name` serta memiliki *constructor* `ProductDelivery` dan `CalculateDelivery`. Jika terdapat perubahan pada `CalculateDelivery` terhadap produk baru (misalnya saja harga medium, small dan large berubah), maka modifikasi kode program tidak terhindarkan. Jika kita langsung melakukan perubahan kode program pada *class* tersebut, maka hal tersebut merupakan **pelanggaran** pada prinsip *Open-Close Principle.*

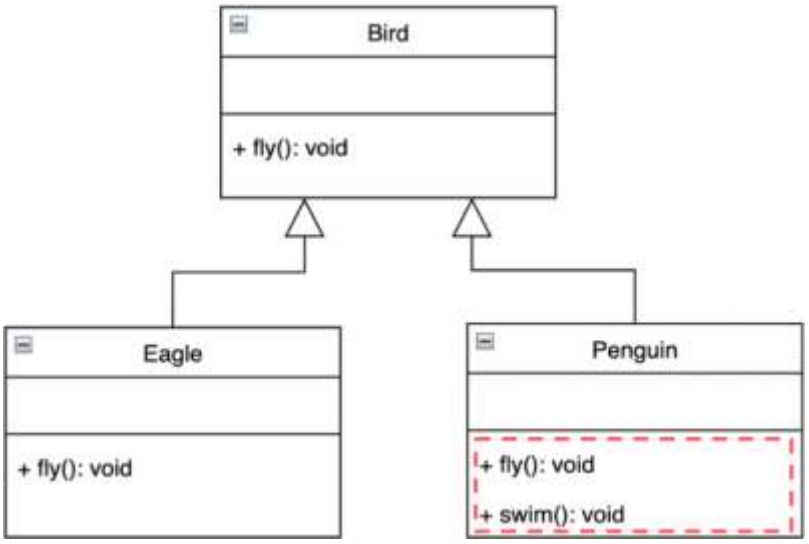
Liskov Substitution Principle

Prinsip ini pertama kali dikenalkan oleh Barbara Liskov pada suatu *conference* tahun 1987 serta pada publikasi makalahnya (jurnal) pada tahun 1994. Pada jurnal tersebut dijelaskan bahwa Liskov Substitution Principle (LSP) mengatur bahwa suatu objek *superclass* harus dapat diganti *(replaceable)* dengan objek *subclass* tanpa merusak fungsionalitas perangkat lunak. Pada tahun 2003, Robert C. Martin menyisipkan LSP sebagai salah satu pilar dari SOLID *Principles* pada bukunya yang berjudul *Agile Software Development, Principles, Patterns, and Practices*. Prinsip ini menguraikan strategi tingkat tinggi untuk mengembangkan perangkat lunak yang lebih fleksibel, dapat dipelihara, serta dapat diperluas. LSP didukung oleh desain berorientasi objek melalui penerapan konsep *inheritance* dan *polymorphism*.

Sebagai contoh, dianalogikan *inheritance* (pewarisan) dan *polymorphism* pada objek `Bebek` dan objek `MainanBebek` (lihat Gambar 5). Objek `MainanBebek` memiliki sifat yang sama dengan `Bebek`, yaitu `MainanBebek` dapat bersuara seperti `Bebek` dan dapat berenang seperti `Bebek`. Namun demikian, hal ini dapat melanggar prinsip *Liskov Substitution*, mengingat tidak ada objek `Bebek` yang hidup dengan menggunakan baterai, hanya `MainanBebek` saja lah yang membutuhkannya. Ini berarti *superclass* (`Bebek`) tidak dapat diganti sepenuhnya oleh *subclass* (`MainanBebek`).

Gambar 5. Ilustrasi Analogi Pelanggaran *Liskov Substitution Principle*

Prinsip pada *LSP* cenderung untuk lebih memperhatikan *semantics* (arti) dibandingkan dengan sintaks. Sebagai contoh tambahan, berikutnya diilustrasikan dengan rancangan *class diagram* antara `Bird`, `Eagle`, dan `Penguin` (lihat Gambar 6).



Gambar 6. Ilustrasi *Class Diagram* Pelanggaran *Liskov Substitution Principle*

Jika diperhatikan (pada Gambar 6), `Eagle` dan `Penguin` adalah `Bird`. Namun demikian, *method* `terbang()` pada `Penguin` tidak dapat didefinisikan, mengingat

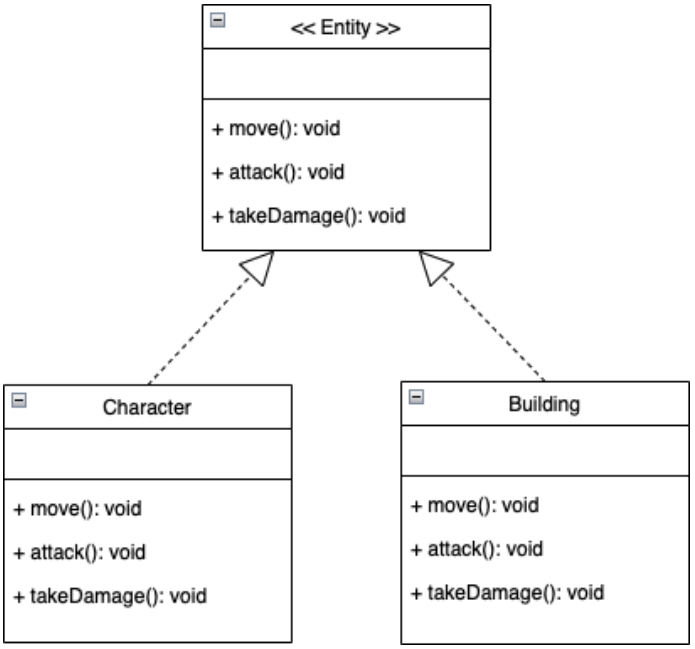
`Penguin` tidak mampu terbang, sehingga objek suatu *superclass* tidak dapat digantikan sepenuhnya oleh *subclass*-nya (pelanggaran LSP).

Interface Segregation Principle

Interface Segregation principle adalah prinsip yang menyatakan bahwa sebuah objek client tidak boleh dipaksa untuk mengimplementasikan sebuah interface yang tidak ia gunakan. Atau objek client tidak boleh bergantung pada metode yang tidak ia gunakan.
Sebuah interface tidak boleh menyediakan semua service untuk client objek. Satu interface hanya memiliki satu tugas spesifik untuk tiap clientnya. Prinsip ini menjelaskan pendekatan untuk menghindari kasus dimana klien digabungkan ke metode yang tidak mereka gunakan.
Sebagai contoh, diilustrasikan pelanggaran *Interface Segregation Principle* melalui objek `Handphone` (lihat Gambar 7).

Gambar 7. Ilustrasi Analogi Pelanggaran *Interface Segregation Principle* (objek handphone)

Pada Gambar 7 dijelaskan bahwa objek handphone dipaksa untuk mengimplementasikan beberapa port (method) yang tidak dibutuhkan dari objek charger.



Gambar 8. Ilustrasi Pelanggaran *Interface Segregation Principle* (objek bangunan)

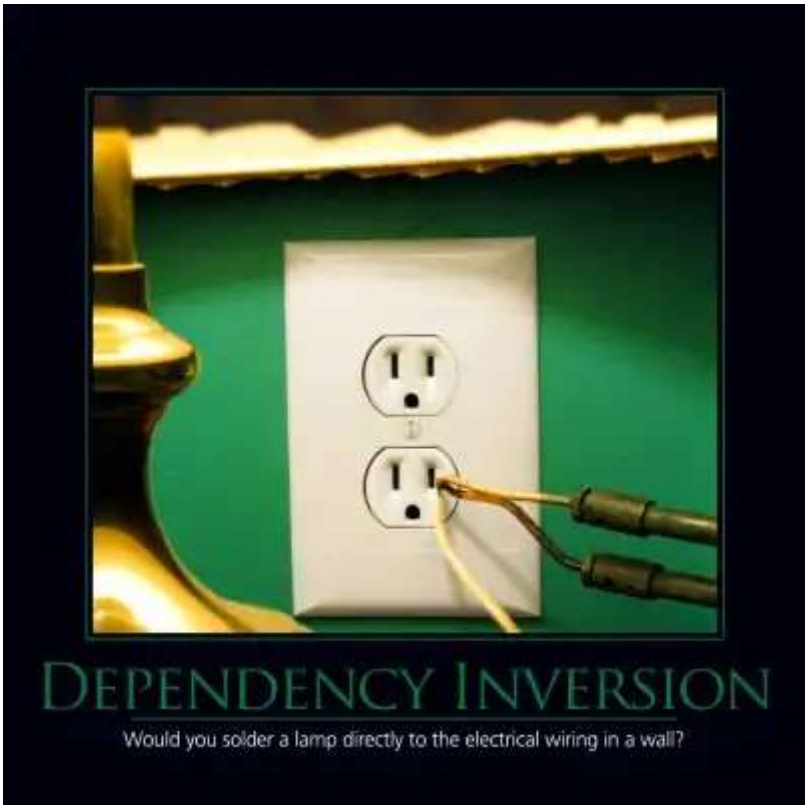
Pada Gambar 8 merupakan contoh pelanggaran *Interface Segregation Principle.* Bahwa class Bangunan **tidak dapat bergerak** dan **menyerang**, namun dipaksa untuk mengimplementasikan method `move()` dan method `attack()` dari superclassnya.

Dependency Inversion Principle

Ide umum pada prinsip ini yaitu “Modul tingkat tinggi tidak boleh bergantung pada modul tingkat rendah, keduanya harus bergantung pada abstraksi”. Oleh karena itu, pada *Dependency Inversion Principle* menggunakan konsep PBO yaitu abstraksi sebagai pemisah modul tingkat dan modul tingkat rendah. Berdasarkan ide yang diusulkan Robert C. Martin, *Dependency Inversion Principle* terdiri dari dua bagian:

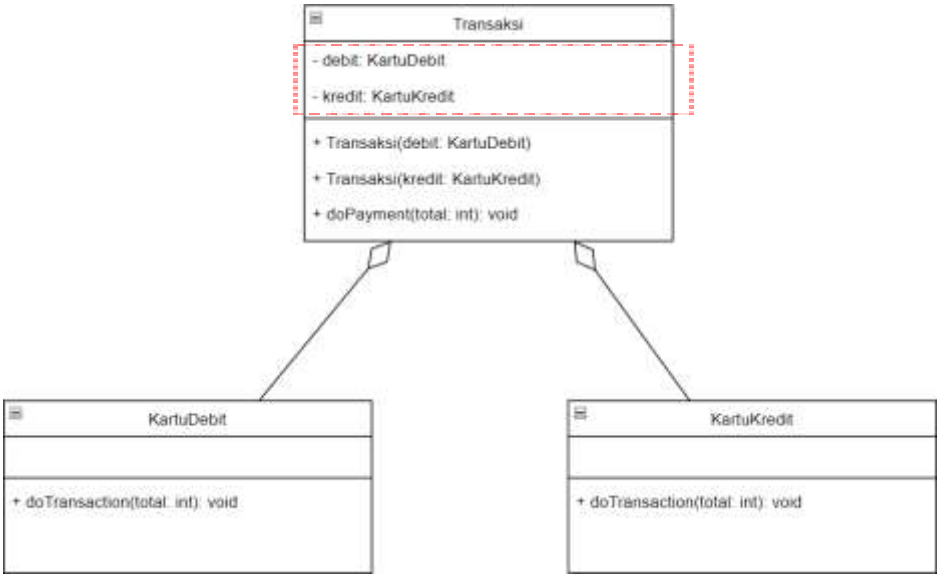
1. Modul tingkat tinggi tidak seharusnya bergantung pada modul tingkat rendah. Keduanya harus bergantung pada abstraksi.
2. Abstraksi tidak seharusnya bergantung pada detail, melainkan detail lah yang bergantung pada abstraksi.

Sebagai contoh, diilustrasikan analogi pelanggaran *Dependency Inversion Principle* melalui penggunaan objek port listrik (lihat Gambar 9).



Gambar 9. Ilustrasi Analogi Pelanggaran *Dependency Inversion Principle*

Pada Gambar 9 dijelaskan bahwa untuk menyalakan objek lampu kita tidak perlu menyolder objek kabel ke objek port listrik. Diperlukan suatu abstraksi yang dapat menghubungkan listrik ke lampu tanpa menyolder kabel langsung ke port listrik, yaitu abstraksi colokan. Adapun contoh lain mengenai pelanggaran prinsip ini digambarkan melalui *class diagram* mengenai konsep `Transaksi`, `KartuKredit`, dan `KartuDebit` (lihat Gambar 10).



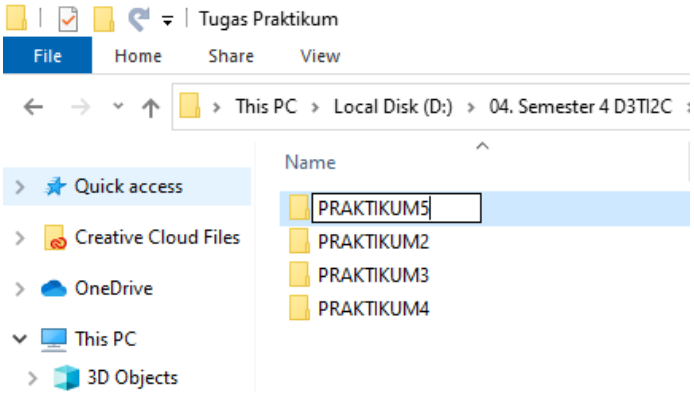
Gambar 10. Ilustrasi *Class Diagram* Pelanggaran *Dependency Inversion Principle*

Pada Gambar 8 dijelaskan bahwa untuk melakukan pembayaran dibutuhkan properti `KartuKredit` dan `KartuDebit`. Hal ini berarti Transaksi bergantung pada `KartuDebit` dan `KartuKredit` setiap kali `doPayment()` dipanggil. Hal ini melanggar konsep *Dependency Inversion Principle*, dimana tidak terimplementasikannya suatu abstraksi dari `KartuKredit` dan `KartuDebit`, sehingga `Transaksi` sangat bergantung pada *class* non abstrak.

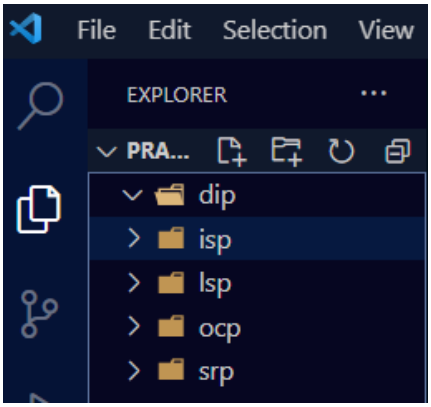
5.3. PELAKSANAAN PRAKTIKUM

Langkah-langkah praktikum SOLID

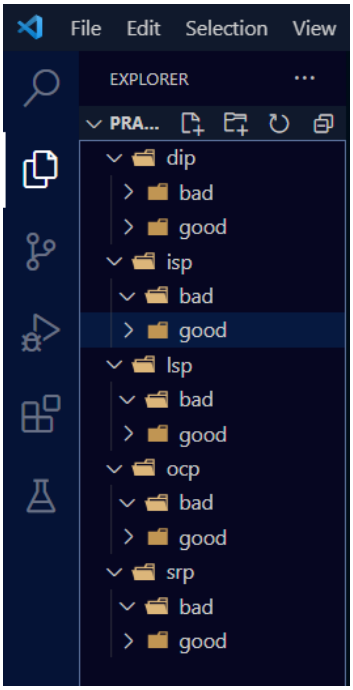
1. Buka aplikasi *VSCode*
2. Klik menu *File > Open Folder*
3. Buatlah *folder* baru bernama `praktikum5`



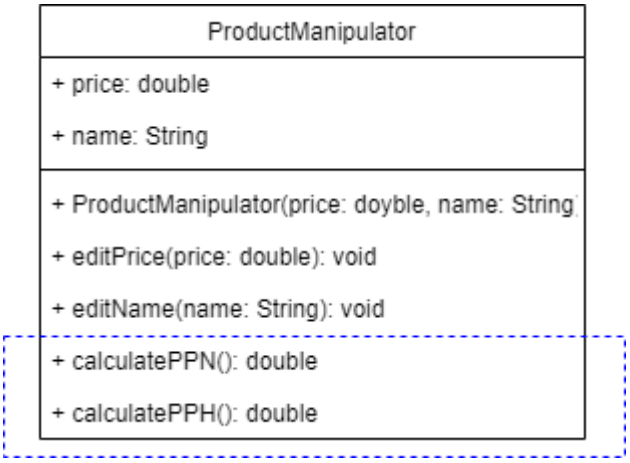
4. Klik tombol *select folder*
5. Buatlah masing-masing *folder* terpisah di dalam `praktikum5`, bernama `srp`, `ocp`, `lsp`, `isp`, dan `dip`



6. Selanjutnya buatlah *folder* `bad` dan `good` pada masing-masing *folder* `srp`, `ocp`, `lsp`, `isp`, dan `dip`



7. Buatlah gambar *class diagram* berikut dan letakkan di dalam *folder* `praktikum5/srp/bad/srp-bad.jpg` atau `.png`



8. Tulis dan simpan kode program 5.1 `praktikum5/srp/bad/product_manipulator.dart`

PRAKTIKUM5 - product_manipulator.dart

```dart
class ProductManipulator  {
  double? price;
  String? name;

  ProductManipulator(this.price, this.name);

  void editPrice(double price) {
    this.price = price;
  }

  void editName(String name) {
    this.name = name;
  }

  double calculatePPN() {
    return this.price! * 10/100;
  }

  double calculatePPH() {
    return this.price! * 0.25 / 100;
  }
}
```

9. Tulis dan simpan kode program 5.2 `praktikum5/srp/bad/main.dart`

PRAKTIKUM5 - main.dart

```dart
import 'product_manipulator.dart';

void main() {
  var product1 = ProductManipulator(20000, "Material 10k");
  product1.editPrice(12000);
  product1.editName("Materai 10000");
  print(product1.calculatePPN());
  print(product1.calculatePPH());
}
```

10. Jalankan kode program melalui menu Terminal > New Terminal, lalu ketik perintah `dart srp/bad/main.dart`

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\srp\bad\main.dart
1200.0
30.0
```

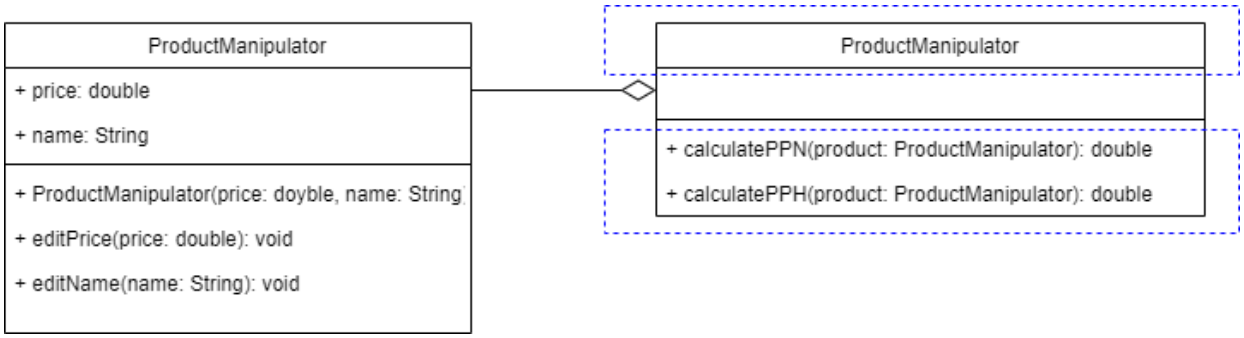
11. Selesai

5.4. LATIHAN

Buatlah *file* gambar *class diagram* (**.png* atau **.jpg*) dan kode program (**.dart*) pada *folder* yang sesuai dengan sub-judul latihan. Kemudian eksekusi kode program yang sudah dibuat serta tampilkan (*screenshot*) hasilnya!

Latihan Single Responsibility

Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Single Responsibility Principle*, kemudian tampilkan hasilnya!



Kode program 5.3 `praktikum5/srp/good/product_manipulator.dart`

PRAKTIKUM5 - product_manipulator.dart

```dart
class ProductManipulator {
  double? price;
  String? name;

  ProductManipulator(this.price, this.name);

  void editPrice(double price){
    this.price=price;
  }

  void editName(String name) {
    this.name=name;
  }
}
```

Kode program 5.4 `praktikum5/srp/good/product_calculator.dart`

PRAKTIKUM5 - product_calculator.dart

```dart
import 'product_manipulator.dart';

class ProductCalculate {


  double calculatePPN(ProductManipulator product){
    return product.price! * 10 /100;
  }

  double calculatePPH(ProductManipulator product){
    return product.price! * 0.25 / 100;
  }
}
```

Kode program 5.5 `praktikum5/srp/good/main.dart`

PRAKTIKUM5 - main.dart

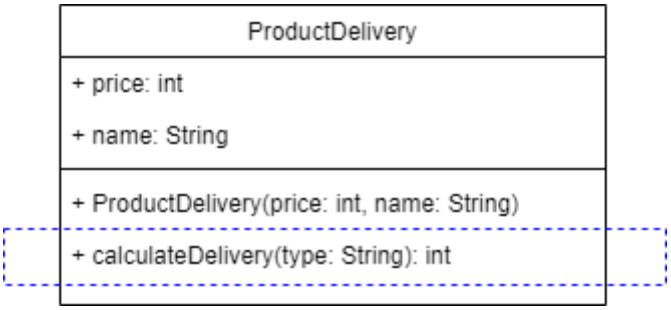
```dart
import 'product_manipulator.dart';
import 'product_calculator.dart';

void main() {
  var product1 = ProductManipulator(25000, 'Buku');

  product1.editPrice(35000);
  product1.editName('Buku Tulis');

  var product2 = ProductCalculate();

  print(product2.calculatePPH(product1));
  print(product2.calculatePPN(product1));
}
```

Hasil kode program 5.5

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\srp\good\main.dart
87.5
3500.0
```

Latihan Open-closed

Buatlah gambar dan kode program berikut untuk menunjukkan pelanggaran *Open-closed Principle*, kemudian tampilkan hasilnya!



Kode program 5.6 `praktikum5/ocp/bad/product_delivery.dart`

PRAKTIKUM5 - product_delivery.dart

```dart
class ProductDelivery {
  int? price;
  String? name;

  ProductDelivery(this.price, this.name);

  int calculateDelivery(String type) {
    switch (type) {
      case "Large":
        return this.price! + 1000;
      case "Medium":
        return this.price! + 500;
      default:
        return this.price! + 100;
    }
  }
}
```

Kode program 5.7 `praktikum5/ocp/bad/main.dart`

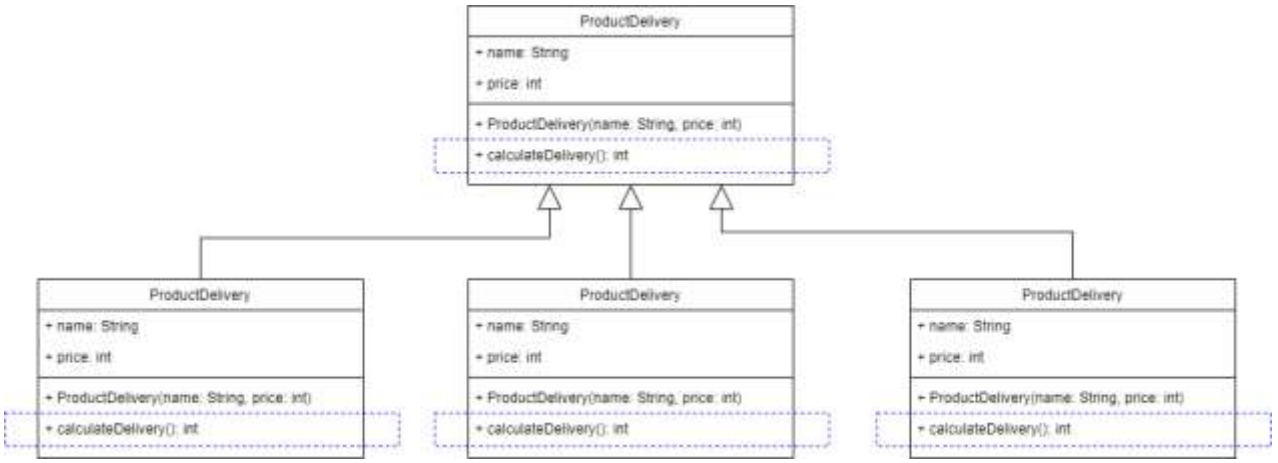
PRAKTIKUM5 - main.dart

```dart
import 'product_delivery.dart';

void main() {
  var product1 = ProductDelivery(12000, "Popcorn");
  print(product1.calculateDelivery("Medium"));
}
```

Hasil kode program 5.7

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\ocp\bad\main.dart
12500
```

Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Open-closed Principle*, kemudian tampilkan hasilnya!



Kode program 5.8 `praktikum5/ocp/good/product_delivery.dart`

PRAKTIKUM5 - product_delivery.dart

```dart
abstract class ProductDelivery {
  int? price;
  String? name;

  ProductDelivery(this.price, this.name);

  int calculateDelivery();
}
```

Kode program 5.9 `praktikum5/ocp/good/large_product.dart`

PRAKTIKUM5 - large_product.dart

```dart
import 'product_delivery.dart';

class LargeProduct extends ProductDelivery {
  LargeProduct(int? price, String? name) : super(price, name);

  @override
  int calculateDelivery() {
    return this.price! + 1000;
  }
}
```

Kode program 5.10 `praktikum5/ocp/good/medium_product.dart`

PRAKTIKUM5 - medium_product.dart

```dart
import 'product_delivery.dart';

class MediumProduct extends ProductDelivery {

  MediumProduct(int? price, String? name) : super(price, name);

  @override
  int calculateDelivery() {
    return this.price! + 700;
  }
}
```

Kode program 5.11 `praktikum5/ocp/good/small_product.dart`

PRAKTIKUM5 - small_product.dart

```dart
import 'product_delivery.dart';

class SmallProduct extends ProductDelivery {

  SmallProduct(int? price, String? name) : super(price, name);

  @override
  int calculateDelivery() {
    return this.price! + 500;
  }
}
```

Kode program 5.12 `praktikum5/ocp/good/main.dart`

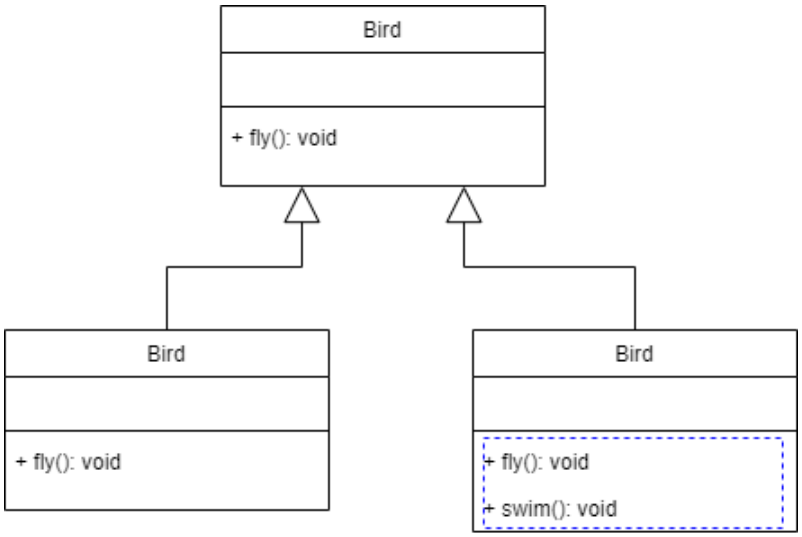
```dart
import 'large_product.dart';
import 'medium_product.dart';
import 'small_product.dart';

void main() {
  LargeProduct

  LargeProduct1 = new LargeProduct(12000, "Popcorn");
  print(LargeProduct1.calculateDelivery());

  MediumProduct MediumProduct1 = new MediumProduct(10000, "Popcorn");
  print(MediumProduct1.calculateDelivery());

  SmallProduct SmallProduct1 = new SmallProduct(75000, "Popcorn");
  print(SmallProduct1.calculateDelivery());
  }
```

Hasil kode program 5.12

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\ocp\good\main.dart
13000
10700
75500
```

Latihan Liskov Substitution

Buatlah gambar dan kode program berikut untuk menunjukkan pelanggaran *Liskov Substitution Principle*, kemudian tampilkan hasilnya!



Kode program 5.13 `praktikum5/lsp/bad/bird.dart`

PRAKTIKUM5 - bird.dart

```dart
class Bird {

  void fly() {
    print("Bird fly!");
  }
}
```

Kode program 5.14 `praktikum5/lsp/bad/eagle.dart`

PRAKTIKUM5 - eagle.dart

```dart
import 'bird.dart';

class Eagle extends Bird {

  @override
  void fly() {
    print("Eagle fly!");
  }
}
```

Kode program 5.15 `praktikum5/lsp/bad/penguin.dart`

PRAKTIKUM5 - penguin.dart

```dart
import 'bird.dart';

class Penguin extends Bird {

  @override
  void fly() {
    // penguin tidak bisa terbang
  }

  void swim() {
    print("Penguin swim!");
  }
}
```

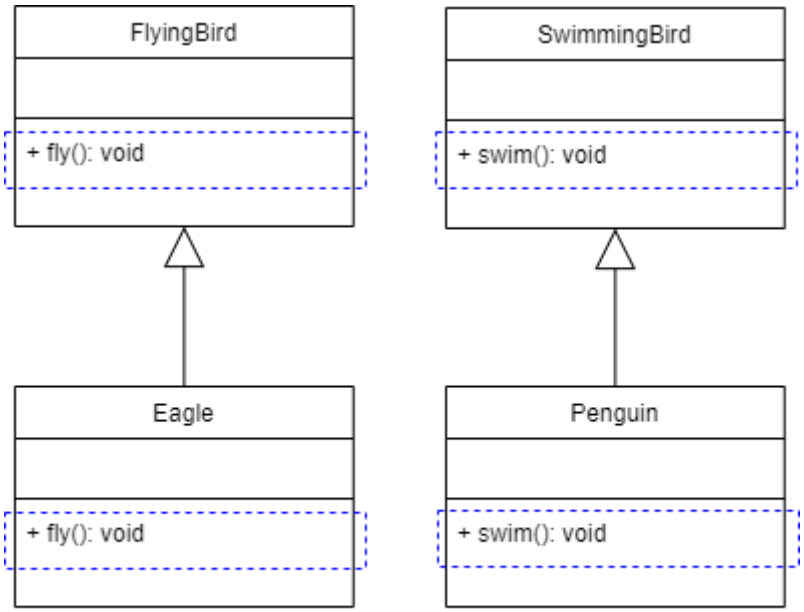
Kode program 5.16 `praktikum5/lsp/bad/main.dart`

```dart
import 'eagle.dart';
import 'penguin.dart';

void main() {
  var eagle = Eagle();
  eagle.fly();

  var penguin = Penguin();
  penguin.fly();
  penguin.swim();
}
```

Hasil kode program 5.16

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\lsp\bad\main.dart
Eagle fly!
Penguin swim!
```

Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Liskov Substitution Principle*, kemudian tampilkan hasilnya!



Kode program 5.17 `praktikum5/lsp/good/flying_bird.dart`

```dart
class FlyingBird {

  void fly() {
    print("Bird fly!");
  }
}
```

Kode program 5.18 `praktikum5/lsp/good/eagle.dart`

PRAKTIKUM5 - eagle.dart

```dart
import 'flying_bird.dart';

class Eagle extends FlyingBird {

  @override
  void fly() {
    print("Eagle fly!");
  }
}
```

Kode program 5.19 `praktikum5/lsp/good/swimming_bird.dart`

PRAKTIKUM5 - swimming_bird.dart

```dart
class SwimmingBird {

  void swim () {
    print("Bird swim!");
  }
}
```

Kode program 5.20 `praktikum5/lsp/good/penguin.dart`

PRAKTIKUM5 - penguin.dart

```dart
import 'swimming_bird.dart';

class Penguin extends SwimmingBird {

  @override
  void swim() {
    print("Penguin swim!");
  }
}
```

Kode program 5.21 `praktikum5/lsp/good/main.dart`

PRAKTIKUM5 - main.dart

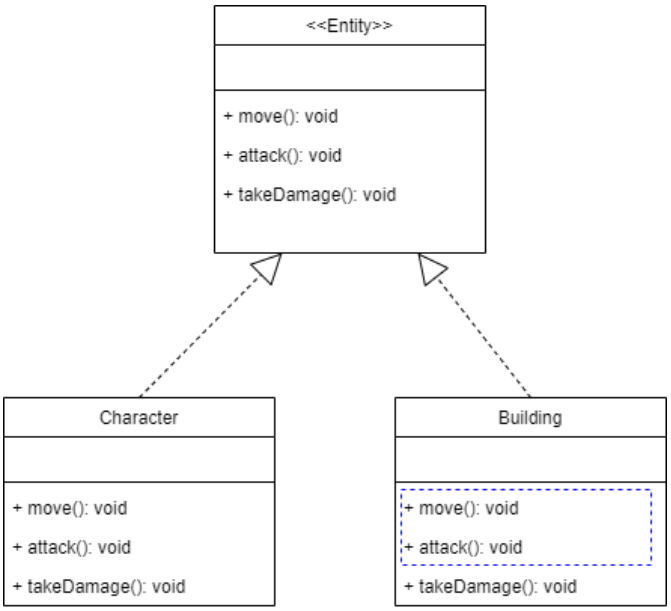
```dart
import 'eagle.dart';
import 'penguin.dart';

void main() {
  var eagle = Eagle();
  eagle.fly();

  var penguin = Penguin();
  penguin.swim();
}
```

Hasil kode program 5.21

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\lsp\good\main.dart
Eagle fly!
Penguin swim!
```

Latihan Interface Segregation

Buatlah gambar dan kode program berikut untuk menunjukkan pelanggaran *Interface Segregation Principle*, kemudian tampilkan hasilnya!



Kode program 5.22 `praktikum5/isp/bad/entity.dart`

PRAKTIKUM5 - entity.dart

```dart
class Entity {
  void move(){}
  void attack(){}
  void takeDamage(){}
}
```

Kode program 5.23 `praktikum5/isp/bad/building.dart`

PRAKTIKUM5 - building.dart

```dart
import 'entity.dart';

class Building implements Entity {

  void move() {
    print('move building');
  }
  void attack() {
    print('attack building');
  }
  void takeDamage() {
    print('take damage building');
  }
}
```

Kode program 5.24 `praktikum5/isp/bad/character.dart`

PRAKTIKUM5 - character.dart

```dart
import 'entity.dart';

class Character implements Entity{

  void move() {
    print('move character');
  }
  void attack() {
    print('attack character');
  }
  void takeDamage() {
    print('take damage character');
  }
}
```

Kode program 5.25 `praktikum5/isp/bad/main.dart`

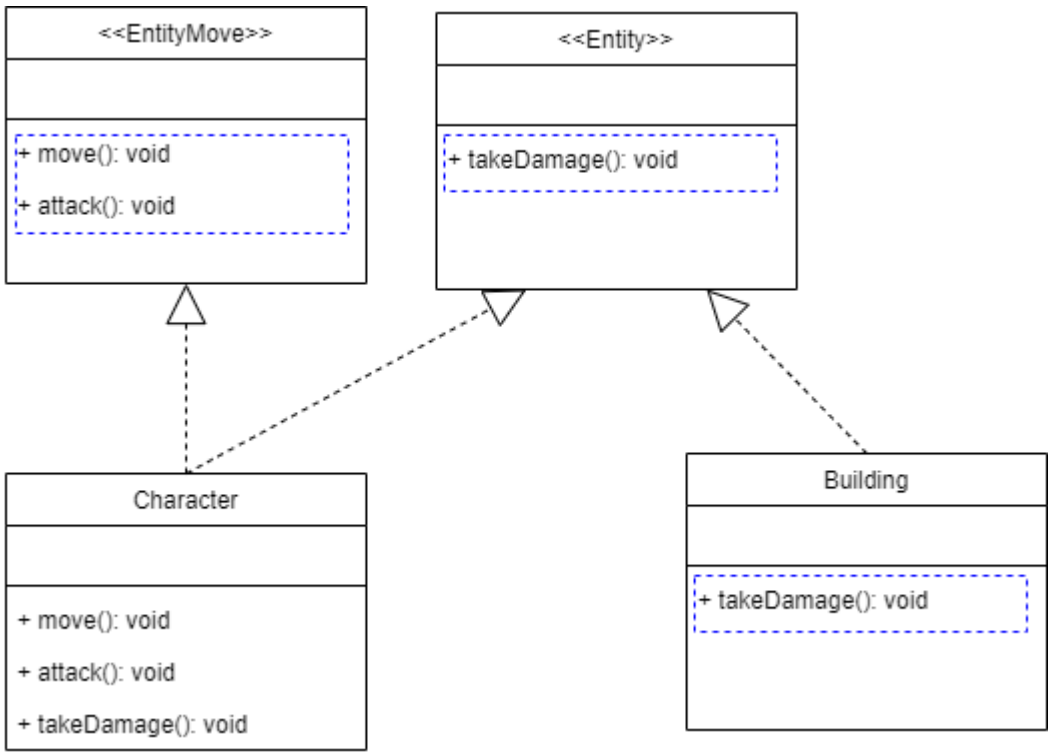
PRAKTIKUM5 - main.dart

```dart
import 'building.dart';
import 'character.dart';

void main() {

  var b1 = Building();
  var c1 = Character();

  b1.move();
  c1.move();
}
```

Hasil kode program 5.25

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\isp\bad\main.dart
move building
move character
```

Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Interface Segregation Principle*, kemudian tampilkan hasilnya!



Kode program 5.26 `praktikum5/isp/good/entity.dart`

PRAKTIKUM5 - entity.dart

```dart
class Entity {
  void takeDamage(){}
}
```

Kode program 5.27 `praktikum5/isp/good/entity_move.dart`

PRAKTIKUM5 - entity_move.dart

```dart
class EntityMove {
  void move(){}
  void attack(){}
}
```

Kode program 5.28 `praktikum5/isp/good/character.dart`

PRAKTIKUM5 - character.dart

```dart
import 'entity.dart';
import 'entity_move.dart';

class Character implements Entity, EntityMove {

  void move() {
    print('move character');
  }
  void attack() {
    print('attack character');
  }
  void takeDamage() {
    print('take damage character');
  }
}
```

Kode program 5.29 `praktikum5/isp/good/building.dart`

PRAKTIKUM5 - building.dart

```dart
import 'entity.dart';

class Building implements Entity {
  void takeDamage() {
    print('take damage building');
  }
}
```

Kode program 5.30 `praktikum5/isp/good/main.dart`

PRAKTIKUM5 - main.dart

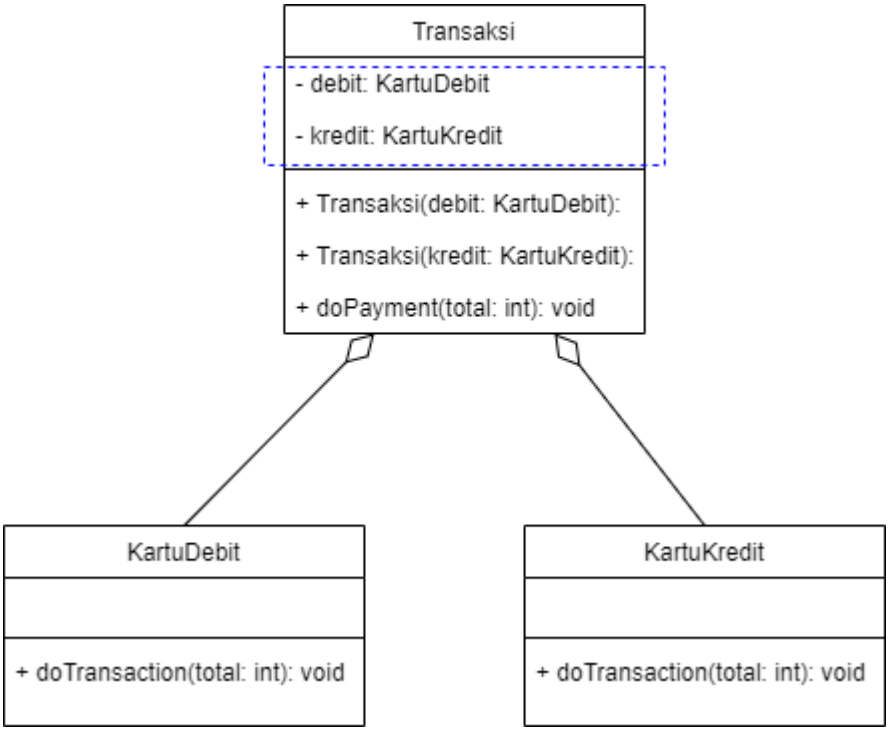
```dart
import 'building.dart';
import 'character.dart';

void main() {

  var b1 = Building();
  var c1 = Character();


  b1.takeDamage();
  c1.move();
  c1.attack();
}
```

Hasil kode program 5.30

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\isp\good\main.dart
take damage building
move character
attack character
```

Latihan Dependency Inversion

Buatlah gambar dan kode program berikut untuk menunjukkan pelanggaran *Dependency Inversion Principle*, kemudian tampilkan hasilnya!



Kode program 5.31 `praktikum5/dip/bad/kartu_debit.dart`

PRAKTIKUM5 - kartu_debit.dart

```dart
class KartuDebit {
  void doTransaksi(int total) {
    print("Transaksi sejumlah $total dengan kartu debit");
  }
}
```

Kode program 5.32 `praktikum5/dip/bad/kartu_kredit.dart`

PRAKTIKUM5 - kartu_kredit.dart

```dart
class KartuKredit {
  void totransaksi(int total) {
    print("Transaksi sejumlah $total dengan kartu kredit");
  }
}
```

Kode program 5.33 `praktikum5/dip/bad/transaksi.dart`

PRAKTIKUM5 - transaksi.dart

```dart
import 'kartu_debit.dart';
import 'kartu_kredit.dart';

class Transaksi {
  KartuDebit? _kartuDebit;
  KartuKredit? _kartuKredit;

  Transaksi.debit(this._kartuDebit);
  Transaksi.kredit(this._kartuKredit);

  void doPayment(int total) {
    _kartuDebit!.doTransaksi(total);
  }

  /**
   * Sengaja dikomentari
   * karena doPayment() kartu kredit
   * bergantung pada pemanggilan constructor kartu kredit
   */
  // void doPayment(int total) {
  // _kartuKredit!.doTransaksi(total);
  // }
}
```

Kode program 5.34 `praktikum5/dip/bad/main.dart`

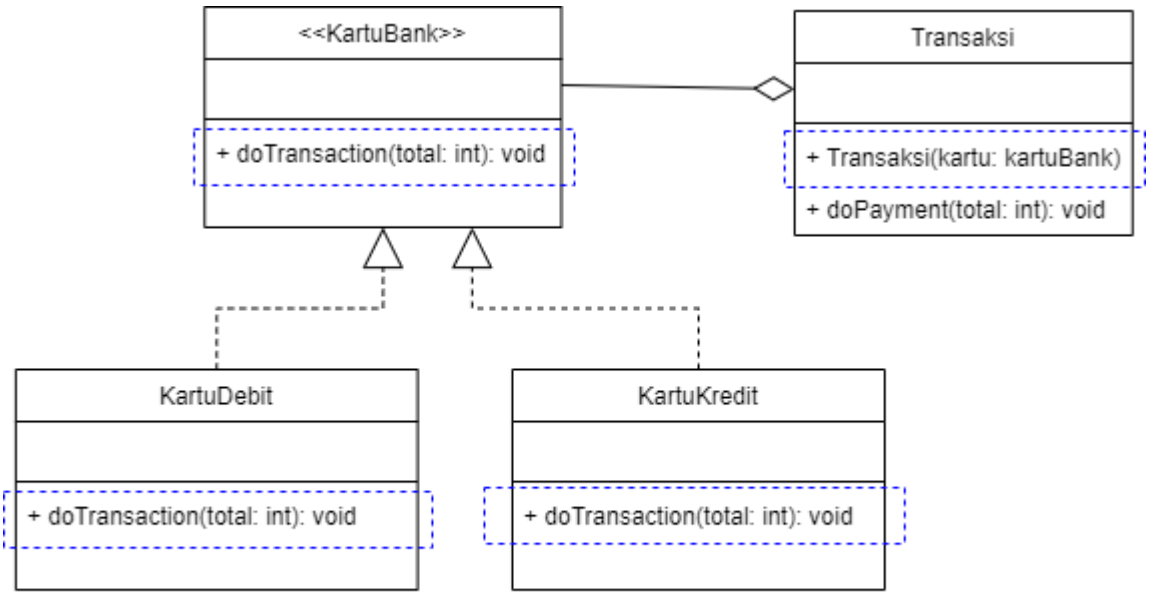
PRAKTIKUM5 - main.dart

```dart
import 'kartu_debit.dart';
import 'kartu_kredit.dart';
import 'transaksi.dart';

void main() {
  var kartuDebit = KartuDebit();
  var kartuKredit = KartuKredit();

  var transaksi1 = Transaksi.debit(kartuDebit);
  transaksi1.doPayment(2000);
}
```

Hasil kode program 5.35

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\dip\bad\main.dart
Transaksi sejumlah 2000 dengan kartu debit
```

Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Dependency Inversion Principle*, kemudian tampilkan hasilnya!



Kode program 5.36 `praktikum5/dip/good/kartu_bank.dart`

PRAKTIKUM5 - kartu_bank.dart

```dart
class KartuBank {
  void doTransaction(int total) {}
}
```

Kode program 5.37 `praktikum5/dip/good/kartu_debit.dart`

PRAKTIKUM5 - kartu_debit.dart

```dart
import 'kartu_bank.dart';

class KartuDebit implements KartuBank {
  @override
  void doTransaction(int total) {
    print("Transaksi sejumlah $total dengan kartu debit");
  }
}
```

Kode program 5.38 `praktikum5/dip/good/kartu_kredit.dart`

PRAKTIKUM5 - kartu_kredit.dart

```dart
import 'kartu_bank.dart';

class KartuKredit implements KartuBank {
  @override
  void doTransaction(int total) {
    print("Transaksi sejumlah $total dengan kartu kredit");
  }
}
```

Kode program 5.39 `praktikum5/dip/good/transaksi.dart`

PRAKTIKUM5 - transaksi.dart

```dart
import 'kartu_bank.dart';

class Transaksi {
  KartuBank? _kartuBank;

  Transaksi(this._kartuBank);

  void doPayment(int total) {
    _kartuBank!.doTransaction(total);
  }
}
```

Kode program 5.39 `praktikum5/dip/good/main.dart`

PRAKTIKUM5 - main.dart

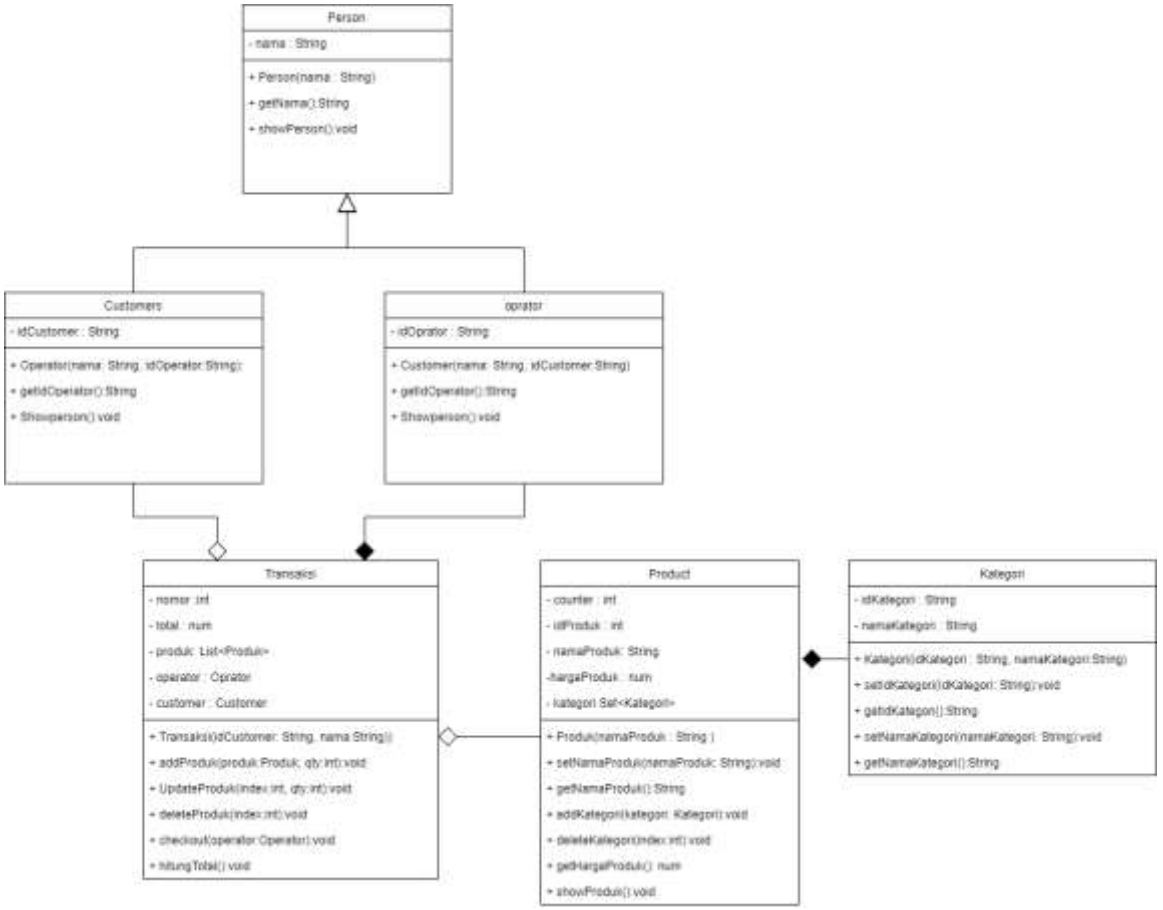
```dart
import 'kartu_debit.dart';
import 'kartu_kredit.dart';
import 'transaksi.dart';

void main() {
  var kartuKredit = KartuKredit();
  var kartuDebit = KartuDebit();

  var transaction2 = Transaksi(kartuDebit);
  transaction2.doPayment(7000);
}
```

Hasil kode program 5.39

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\dip\good\main.dart
Transaksi sejumlah 7000 dengan kartu debit
```

5.5. TUGAS INDIVIDU

1. Buatlah class diagram yang menerapkan prinsip SOLID berdasarkan studi kasus Produk Aplikasi pada mata kuliah Proyek 3!



2. Buatlah kode program dart berdasarkan studi kasus class diagram soal nomor 1!

```
import 'person.dart';

class Customer extends Person{
  String? _idCustomer;

  Customer(String nama, this._idCustomer) : super(nama);

  String get idCustomer => _idCustomer!;

  void showPerson() {
    print (_idCustomer);
  }
}
```

```
class Kategori{
  String? _idKategori;
  String? _namaKategori;

  Kategori(this._idKategori, this._namaKategori);

  void set idKategori (String idKategori) => _idKategori;
  String get idKategori => _idKategori!;

  void set namaKategori (String namaKategori) => _namaKategori;
  String get namaKategori => _namaKategori!;

  void add(Kategori kategori) {}
}
```

PRAKTIKUM5 - person.dart

```dart
class Person {
  String? _nama;

  Person(this._nama);

  set nama(String? nama) => _nama;
  String get nama => _nama!;

  void showPerson() {
    print (_nama);
  }
}
```

PRAKTIKUM5 - operator.dart

```dart
import 'person.dart';

class Operator extends Person{
  String? _idOperator;

  Operator(String nama, this._idOperator) : super(nama);

  String get idCustomer => _idOperator!;

  void showPerson() {
    print (_idOperator);
  }
}
```

PRAKTIKUM5 - transaksi.dart

```dart
import 'operator.dart';
import 'customer.dart';
import 'produk.dart';

class Transaksi {
  static int? _nomor;
  num? _total;
  Operator? _operator;
  Customer? _customer;

  List<Produk> _produk = [];

  Transaksi(String _idCustomer, String? nama) {
    _customer = new Customer("kabrot", "1");
  }

  void addProduk(Produk produk, int qty) {
    _produk.add(produk);
    _total = qty;
  }

  void updateProduk(int index) {
    print(_produk[index]);
  }

  void checkout(Operator operator) {
    _operator = operator;
  }

  void hitungTotal() {
    print("nama pemesan ${_customer.nama}");
    print("----------------");
    for(var produk in _produk){

      produk.showProduk();

      print ("total pesanan ${_total}");
      print ("harga produk ${produk.hargaProduk}");
      print("total harga produk ${_total! * produk.getHargaProduk()}");
    }
    print("operator ${_operator!.nama}");
    print("");
  }
}
```

PRAKTIKUM5 - produk.dart

```dart
import 'kategori.dart';

class Produk {
  static int counter = 1;
  int? _idProduk;
  String? _namaProduk;
  num? _hargaProduk;
  Set<Kategori> _kategori = new Set<Kategori>();

  Produk(this._namaProduk, this._hargaProduk) {
    counter++;
  }

  void set namaProduk(String namaProduk) => _namaProduk;
  String get namaProduk => _namaProduk!;

  void addKategori(Kategori kategori) {
    _kategori.add(kategori);
  }
  void deleteKategori(int index) {
    _kategori.remove(index);
  }

  num get hargaProduk => _hargaProduk!;
  num getHargaProduk(){
    for(var ktgproduk in _kategori){
      _hargaProduk = _hargaProduk!;
    }

    return _hargaProduk!;
  }

  void showProduk() {
    print("nama Produk $namaProduk dengan Kategori : ");
    for(var kategori in _kategori){
      print("${kategori.namaKategori}");
    }
  }
}
```

Main.dart

PRAKTIKUM5 - main.dart

```dart
import 'customer.dart';
import 'operator.dart';
import 'produk.dart';
import 'kategori.dart';
import 'transaksi.dart';

void main() {
  var operator1 = Operator("aldi", "001");
  var operator2 = Operator("ica", "002");

  var customer1 = Customer("Sahrul", "1");

  var kategori1 = Kategori('1', "buah");
  var kategori2 = Kategori('2', "sayur");

  var produk1 = Produk("tomat", 20000);
  var produk2 = Produk("brokoli", 15000);

  produk1.addKategori(kategori1);
  produk1.addKategori(kategori2);

  var transaksi1 = Transaksi("Sahrul", "1");

  transaksi1.addProduk(produk1, 1);
  transaksi1.checkout(operator1);
  transaksi1.hitungTotal();

  produk2.addKategori(kategori2);

  var transaksi2 = Transaksi("sahrul", "1");

  transaksi2.checkout(operator2);
  transaksi2.addProduk(produk2, 3);
  transaksi2.hitungTotal();
}

```

Hasil Kode Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM5> dart .\prak5\main.dart
nama pemesan Sahrul
-----------------
nama Produk tomat dengan Kategori :
buah
sayur
total pesanan 1
harga produk 20000
total harga produk 20000
operator aldi

nama pemesan Sahrul
-----------------
nama Produk brokoli dengan Kategori :
sayur
total pesanan 3
harga produk 15000
total harga produk 45000
operator ica
```

5.6. REFERENSI

- Alberto Miola. “Flutter Complete Reference Create Beautiful, Fast and Native Apps for Any Device”.Independently Published. 2020.
- Sanjib Sinha. “Beginning Flutter with Dart: A Step by Step Guide for Beginners to Build a Basic Android or iOS Mobile Application”. Lean Publishing. 2021.
- Simone Alessandria, Brian Kayfirz. “Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart”. Packt Publishing. Birmingham - Mumbai. 2021.
- Thomas Bailey, Alessandro Biessek. “Flutter for Beginners Second Edition: An introductory guide to building cross-platform mobile applications with Flutter 2.5 and Dart”. Packt Publishing. Birmingham - Mumbai. 2021.
- Dieter Meiller. “Modern App Development with Dart and Flutter 2: A Comprehensive Introduction to Flutter”. Walter de Gruyter GmbH. Berlin - Boston. 2021.
- Priyanka Tyagi. “Pragmatic Flutter: Building Cross-Platform Mobile Apps for Android, iOS, Web & Desktop”. CRC Press Taylor & Francis Group, LLC. London - New York. 2022.
- Zack West. “Liskov Substitution Principle (LSP): SOLID Design for Flexible Code”. https://www.alpharithms.com/liskov-substitution-principle-lsp-solid-114908/. Diakses tanggal 14 Maret 2022.
- Zack West. “SOLID: Guidelines for better Software Development”. https://www.alpharithms.com/solid-guidelines-for-better-software-development-055500/. Diakses tanggal 14 Maret 2022.