**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**

## PRAKTIKUM 5

SOLID PROGRAMMING DART

## 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

## 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)

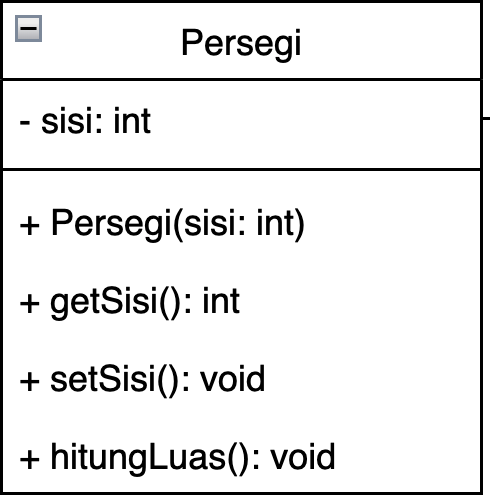
I - Interface segregation principle (ISP) D - Dependency inversion principle (DIP)

## 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*

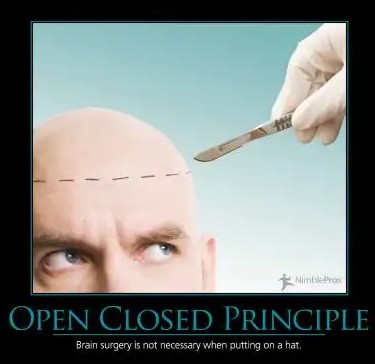
Dijelaskan bahwa pada Gambar 1 tidak boleh ada objek yang memiliki lebih dari satu tanggung jawab, misalnya pisau multi fungsi dan sendok yang sekaligus menjadi garpu. Hal ini bertujuan untuk menyederhanakan proses pemeliharaan, serta mengurangi potensi kesalahan apabila perlu dilakukan perubahan suatu 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.

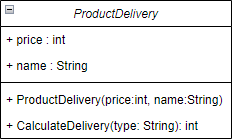
## Open-closed Principle

*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

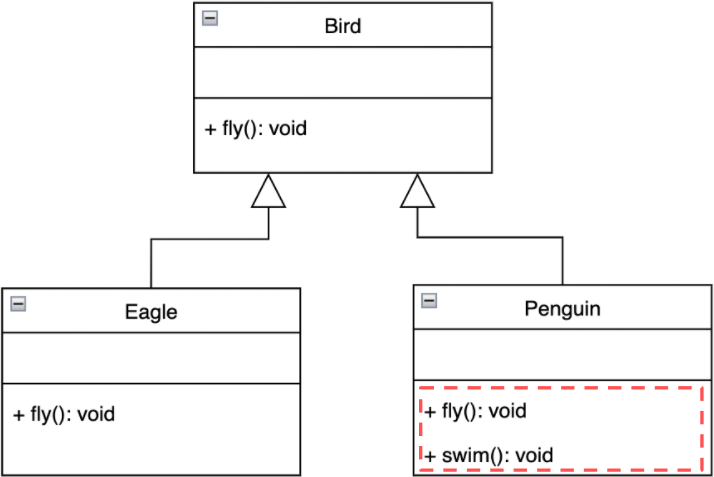
Prinsip ini pertama kali dikenalkan oleh Barbara Liskov pada suatu *conference* tahun 1987 serta pada publikasi makalahnya (jurnal) pada tahun 1994. Pada jurnal tersebut dijelaskan bahwa Liskov Substitution Principle (LSP) mengatur bahwa suatu objek *superclass* harus dapat diganti *(replaceable)* dengan 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

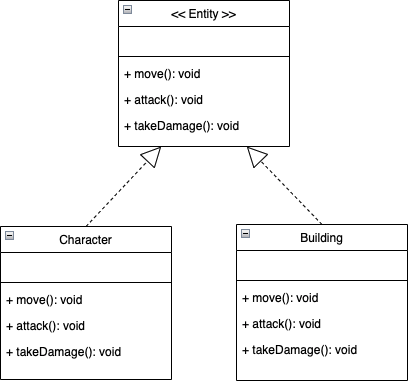
Interface Segregation principle adalah prinsip yang menyatakan bahwa sebuah objek client tidak boleh dipaksa untuk mengimplementasikan sebuah interface yang tidak ia gunakan. Atau objek client tidak boleh bergantung pada metode yang tidak ia gunakan.

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)

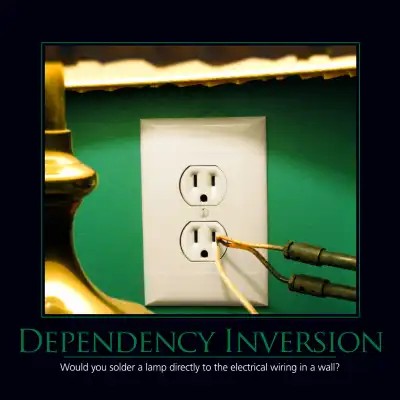
Pada Gambar 8 merupakan contoh pelanggaran *Interface Segregation Principle.* Bahwa class Bangunan **tidak dapat bergerak** dan **menyerang**, namun dipaksa untuk mengimplementasikan method move() dan method attack() dari superclassnya.

## 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).

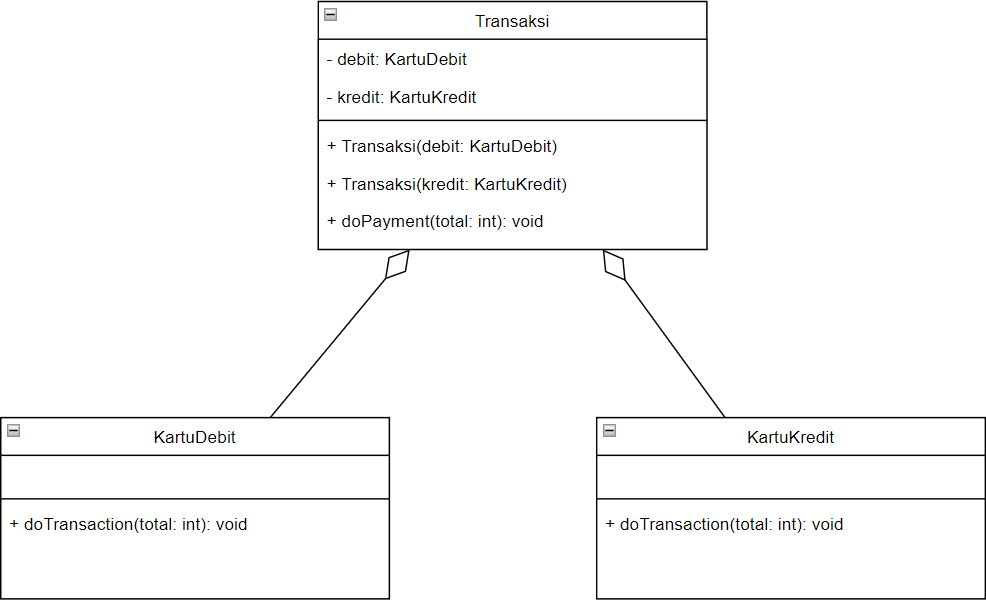


**Gambar 9**. Ilustrasi Analogi Pelanggaran *Dependency Inversion Principle*

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*

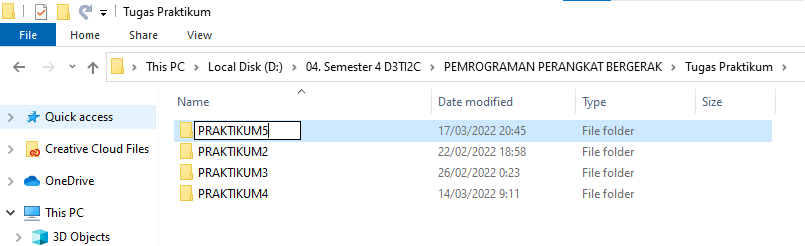
Pada Gambar 8 dijelaskan bahwa untuk melakukan pembayaran dibutuhkan properti

KartuKredit dan KartuDebit. Hal ini berarti Transaksi bergantung pada

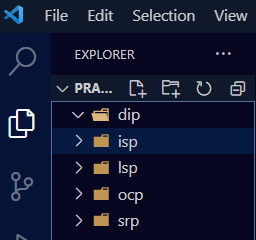
KartuDebit dan KartuKredit setiap kali doPayment() dipanggil. Hal ini melanggar konsep *Dependency Inversion Principle*, dimana tidak terimplementasikannya suatu abstraksi dari KartuKredit dan KartuDebit, sehingga Transaksi sangat bergantung pada *class* non abstrak.

## PELAKSANAAN PRAKTIKUM Langkah-langkah praktikum SOLID

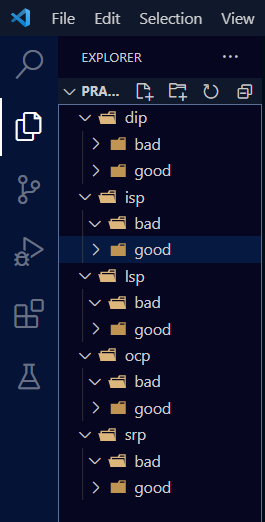
* + 1. Buka aplikasi *VSCode*
    2. Klik menu *File* > *Open Folder*
    3. Buatlah *folder* baru bernama praktikum5

**

* + 1. Klik tombol *select folder*
    2. Buatlah masing-masing *folder* terpisah di dalam praktikum5, bernama srp, ocp, lsp, isp, dan dip

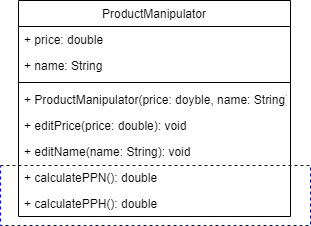


* + 1. Selanjutnya buatlah *folder* bad dan good pada masing-masing *folder* srp, ocp, lsp, isp, dan dip



* + 1. Buatlah gambar *class diagram* berikut dan letakkan di dalam *folder*

# praktikum5/srp/bad/srp-bad.jpg atau .png



* + 1. Tulis dan simpan kode program 5.1

# praktikum5/srp/bad/product\_manipulator.dart

# 

* + 1. Tulis dan simpan kode program 5.2 praktikum5/srp/bad/main.dart



* + 1. Jalankan kode program melalui menu Terminal > New Terminal, lalu ketik perintah

# dart srp/bad/main.dart

# 

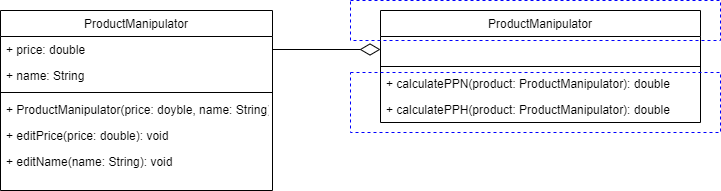
* + 1. Selesai

## LATIHAN

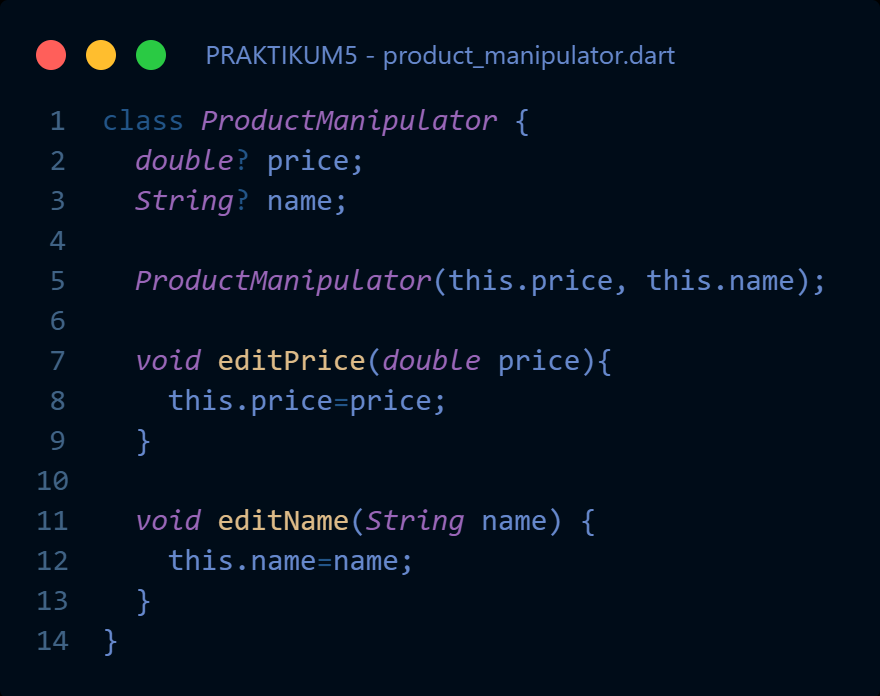
Buatlah *file* gambar *class diagram* (*\*.png* atau *\*.jpg*) dan kode program (*\*.dart*) pada *folder* yang sesuai dengan sub-judul latihan. Kemudian eksekusi kode program yang sudah dibuat serta tampilkan (*screenshot*) hasilnya!

## Latihan Single Responsibility

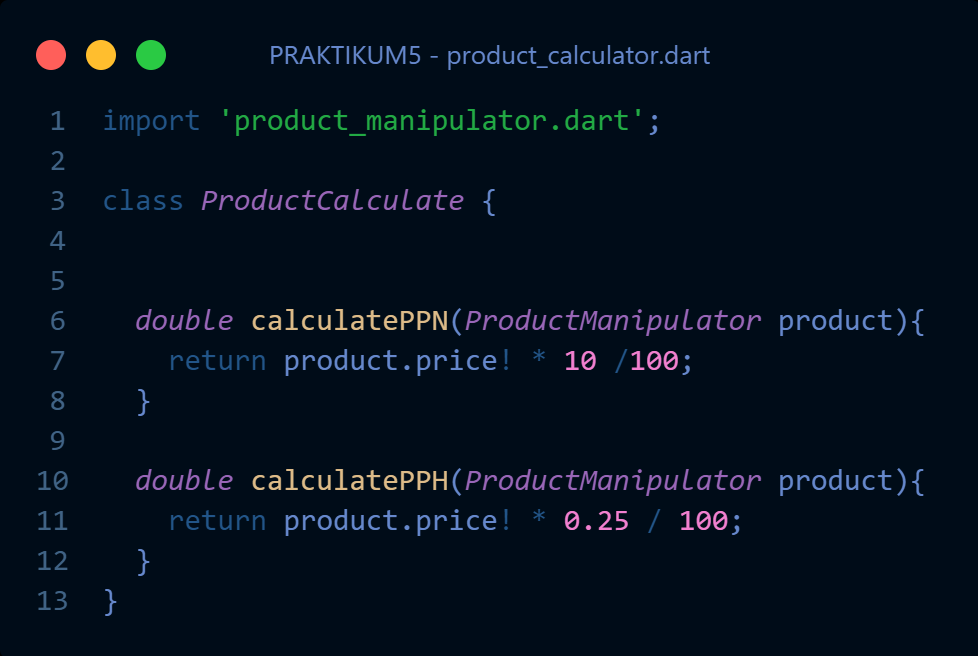
Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Single Responsibility Principle*, kemudian tampilkan hasilnya!



Kode program 5.3 praktikum5/srp/good/product\_manipulator.dart



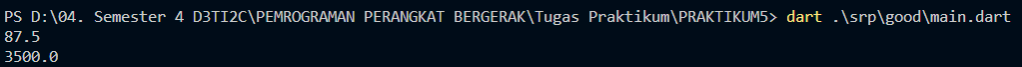
Kode program 5.4 praktikum5/srp/good/product\_calculator.dart



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

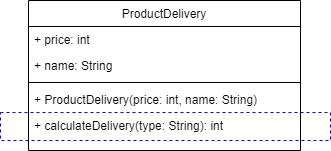


Hasil kode program 5.5

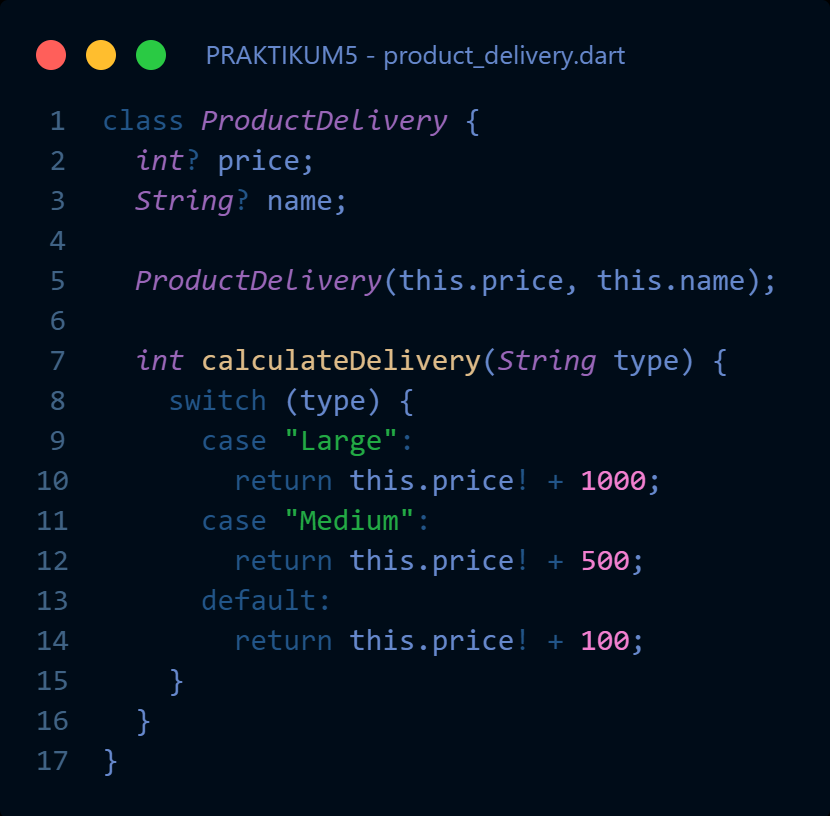


## Latihan Open-closed

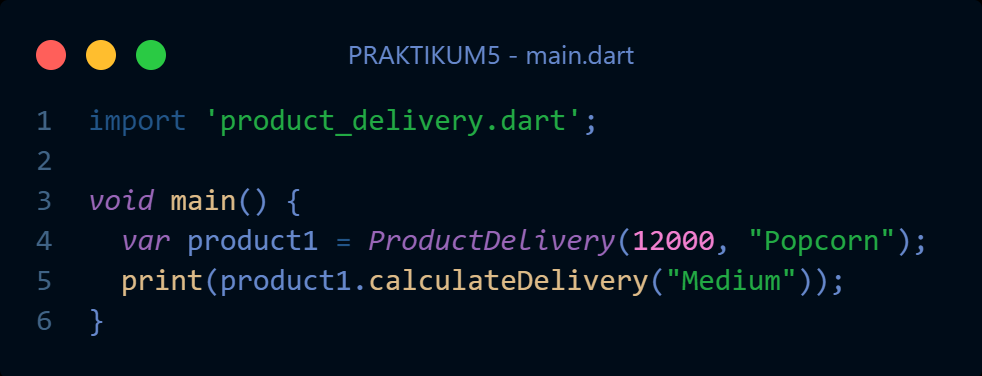
Buatlah gambar dan kode program berikut untuk menunjukkan pelanggaran *Open-closed Principle*, kemudian tampilkan hasilnya!



Kode program 5.6 praktikum5/ocp/bad/product\_delivery.dart



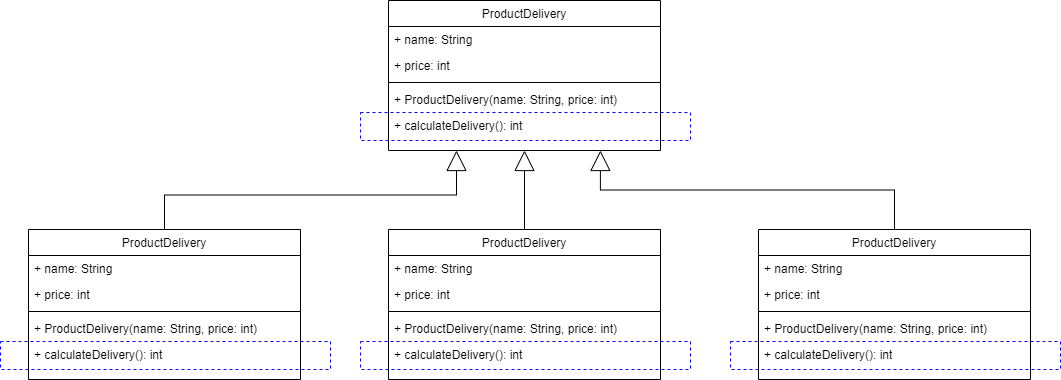
Kode program 5.7 praktikum5/ocp/bad/main.dart



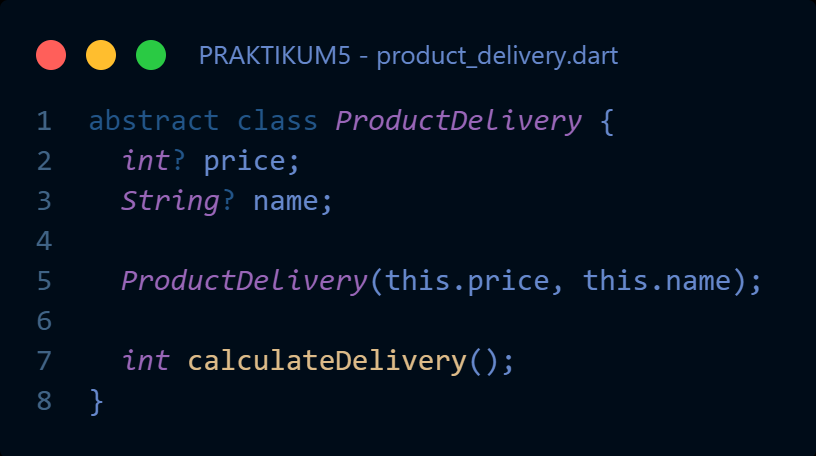
Hasil kode program 5.7



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



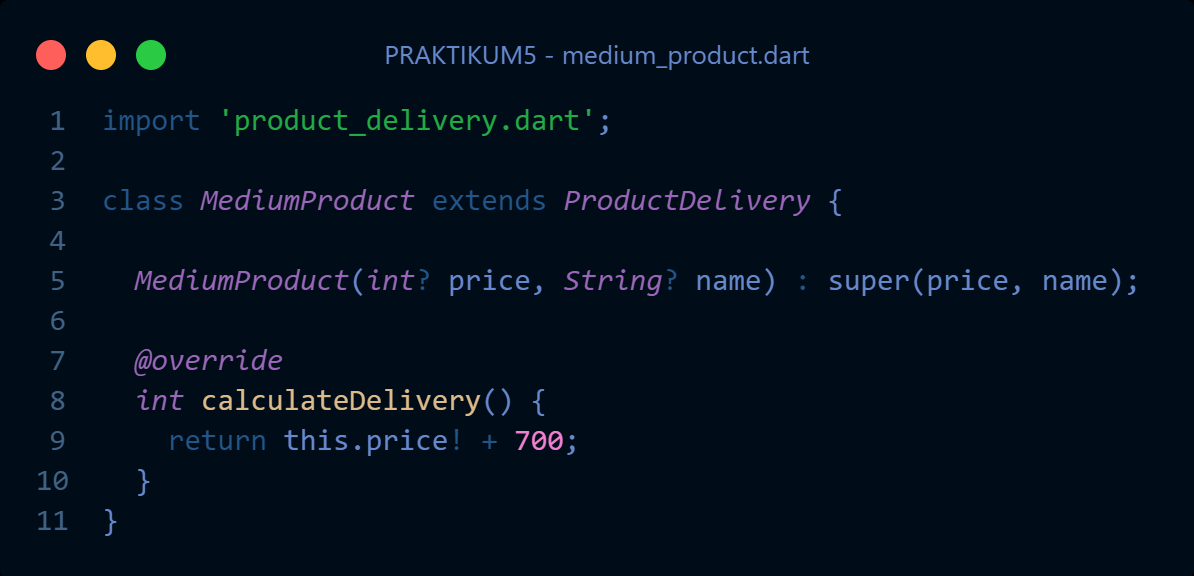
Kode program 5.8 praktikum5/ocp/good/product\_delivery.dart



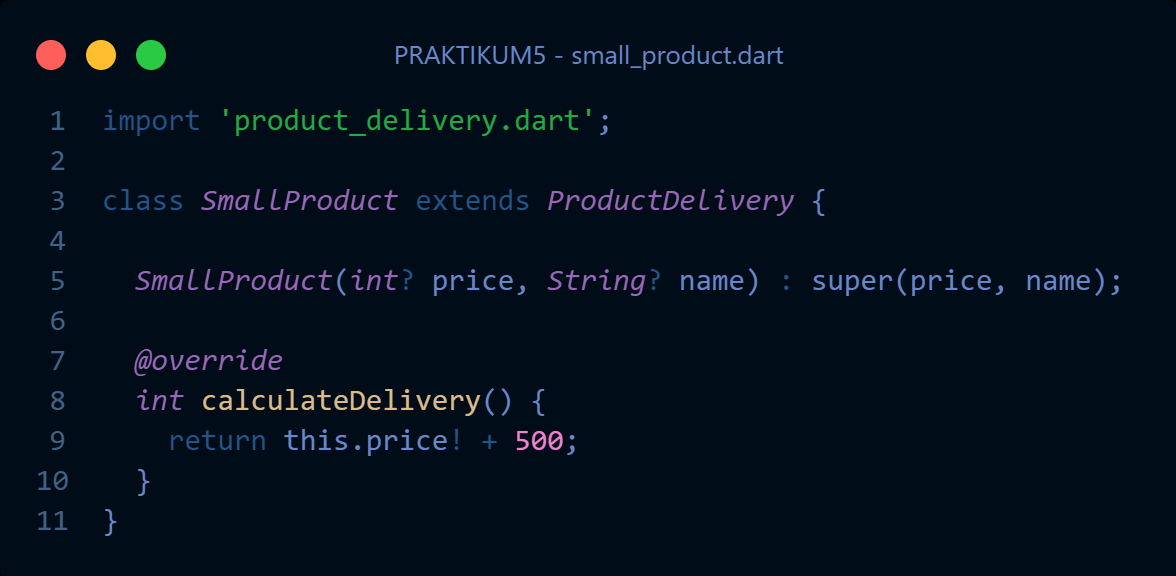
Kode program 5.9 praktikum5/ocp/good/large\_product.dart



Kode program 5.10 praktikum5/ocp/good/medium\_product.dart



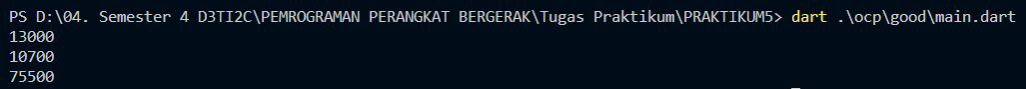
Kode program 5.11 praktikum5/ocp/good/small\_product.dart



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

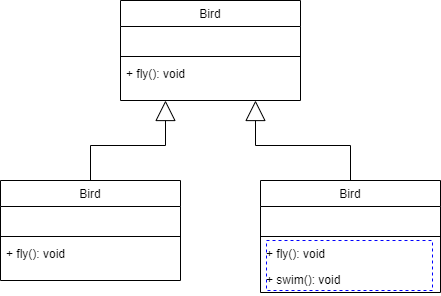


Hasil kode program 5.12



## 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



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



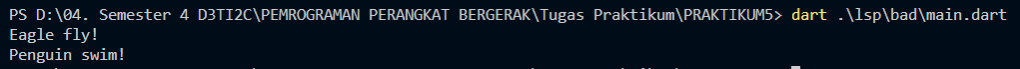
Kode program 5.15 praktikum5/lsp/bad/penguin.dart



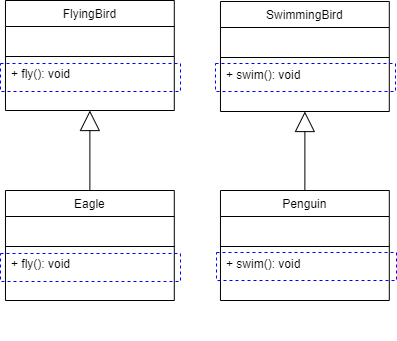
Kode program 5.16 praktikum5/lsp/bad/main.dart



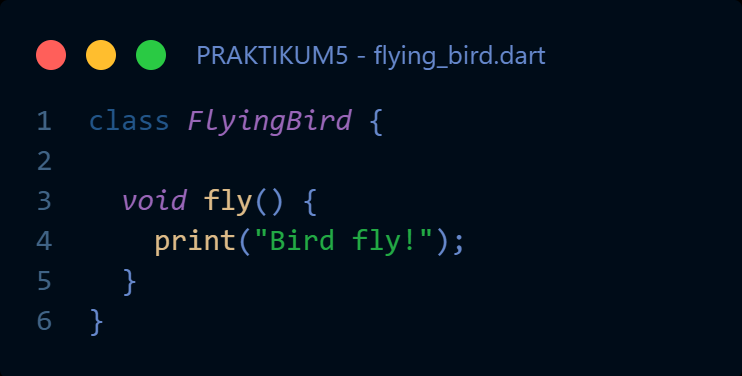
Hasil kode program 5.16



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



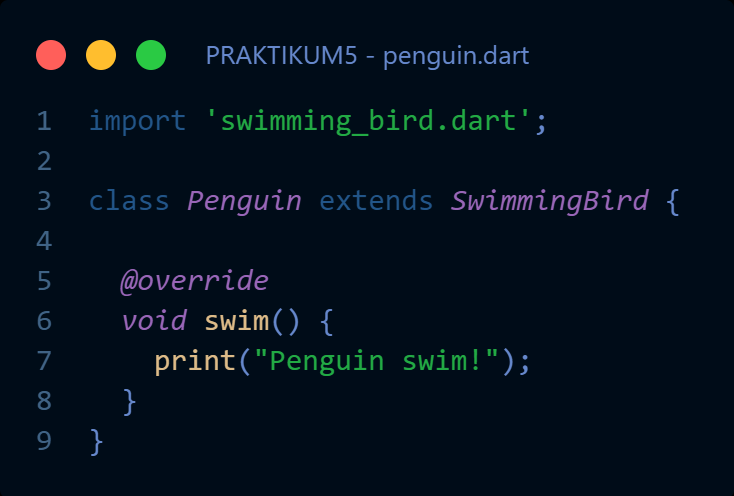
Kode program 5.18 praktikum5/lsp/good/eagle.dart



Kode program 5.19 praktikum5/lsp/good/swimming\_bird.dart



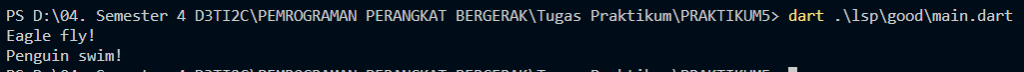
Kode program 5.20 praktikum5/lsp/good/penguin.dart



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

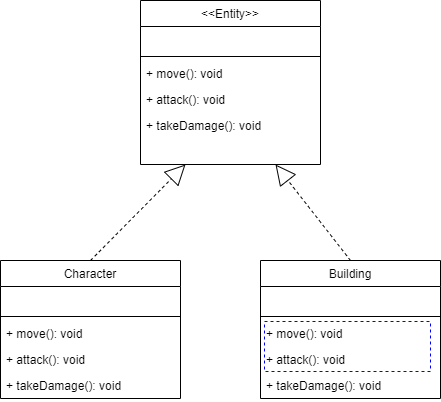


Hasil kode program 5.21

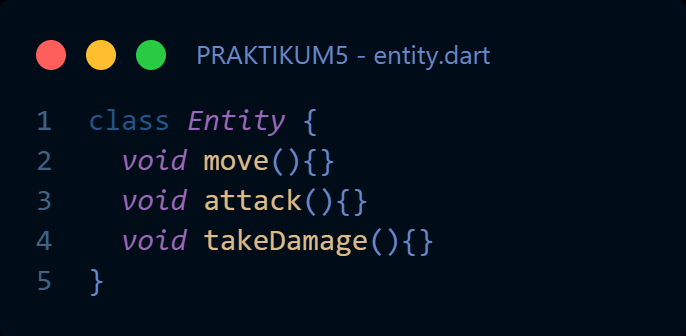


## Latihan Interface Segregation

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



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



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



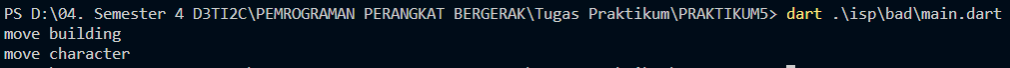
Kode program 5.24 praktikum5/isp/bad/character.dart



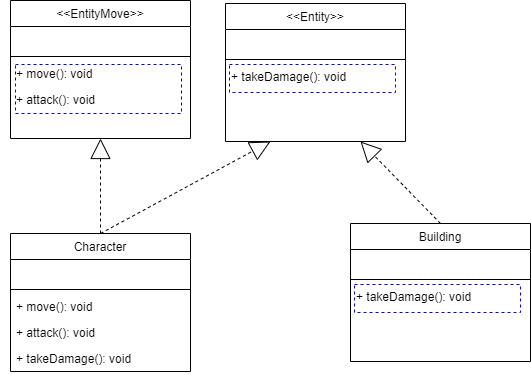
Kode program 5.25 praktikum5/isp/bad/main.dart



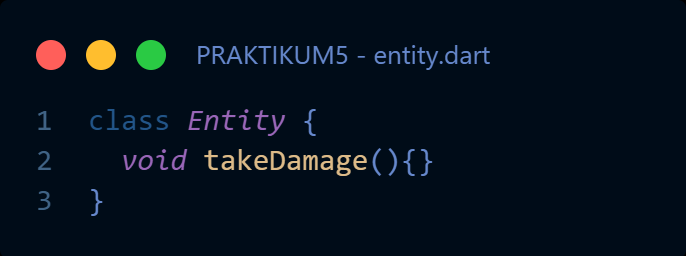
Hasil kode program 5.25



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



Kode program 5.27 praktikum5/isp/good/entity\_move.dart



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



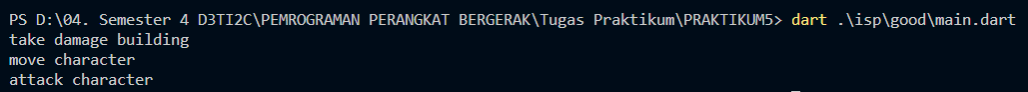
Kode program 5.29 praktikum5/isp/good/building.dart



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

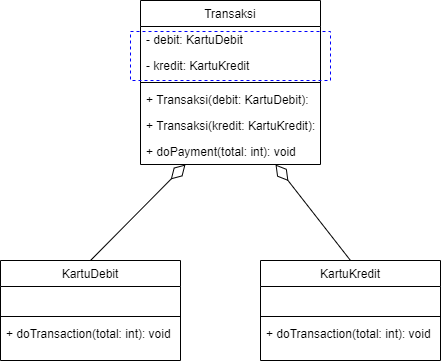


Hasil kode program 5.30

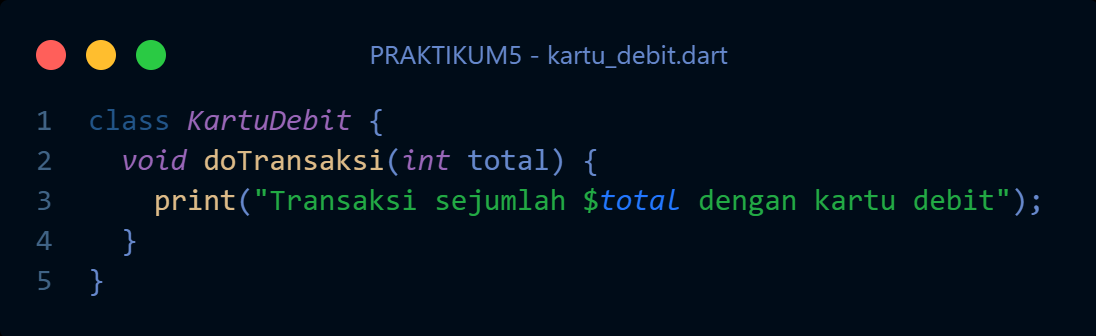


## Latihan Dependency Inversion

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



Kode program 5.32 praktikum5/dip/bad/kartu\_kredit.dart



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



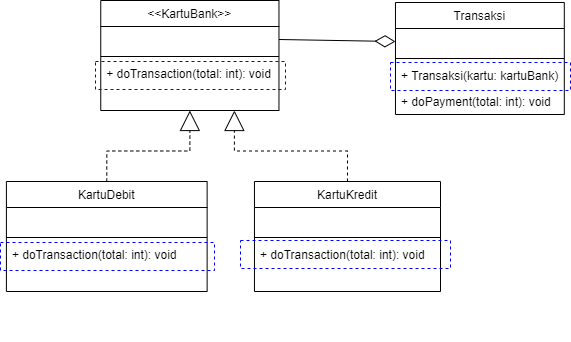
Kode program 5.34 praktikum5/dip/bad/main.dart



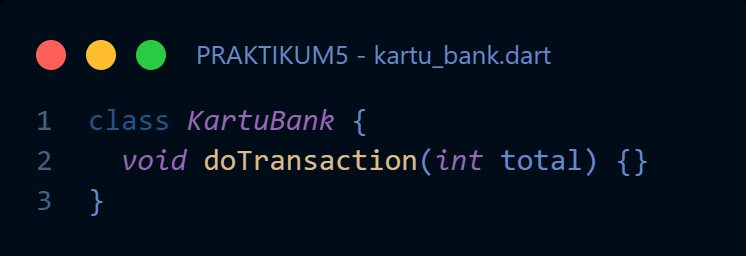
Hasil kode program 5.35



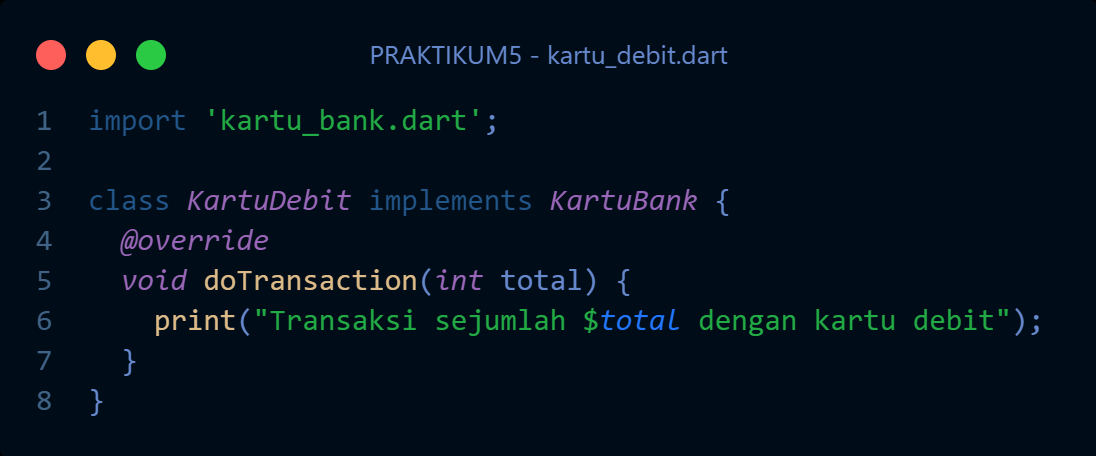
Buatlah gambar dan kode program berikut untuk menunjukkan penyesuaian studi kasus terhadap *Dependency Inversion Principle*, kemudian tampilkan hasilnya!



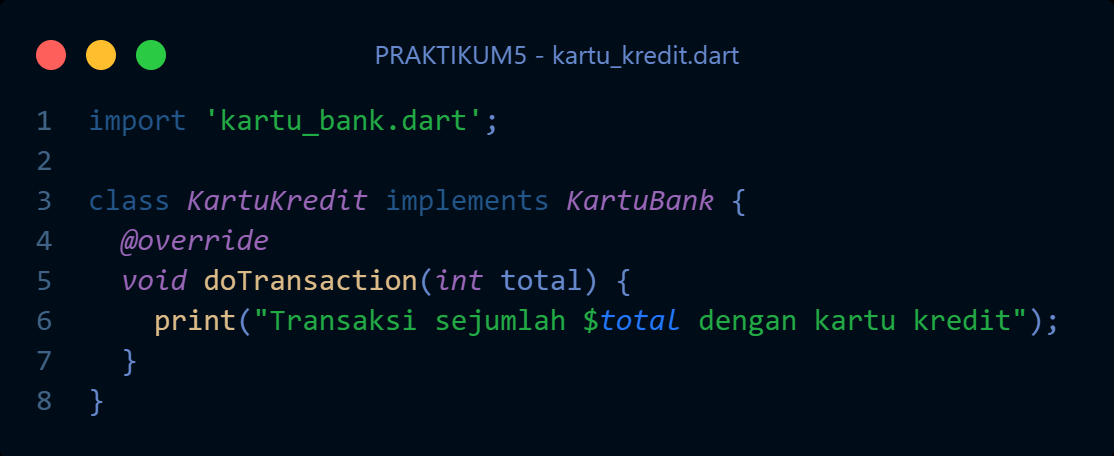
Kode program 5.36 praktikum5/dip/good/kartu\_bank.dart



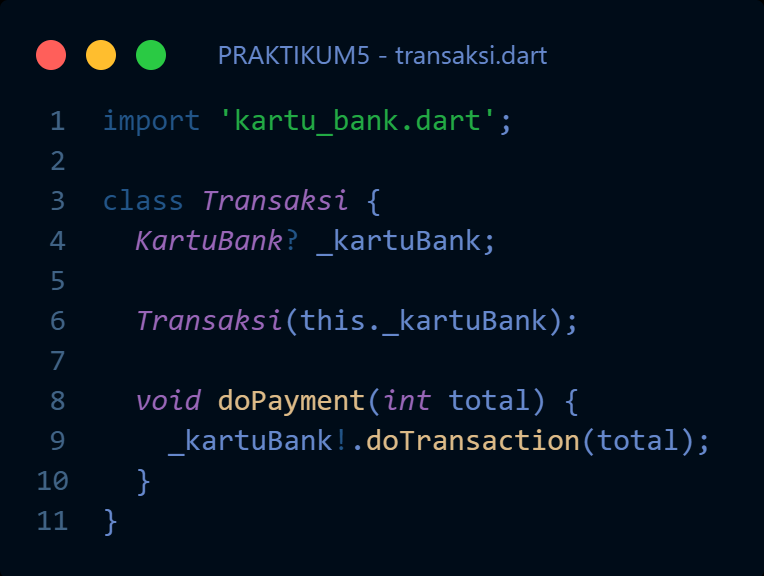
Kode program 5.37 praktikum5/dip/good/kartu\_debit.dart



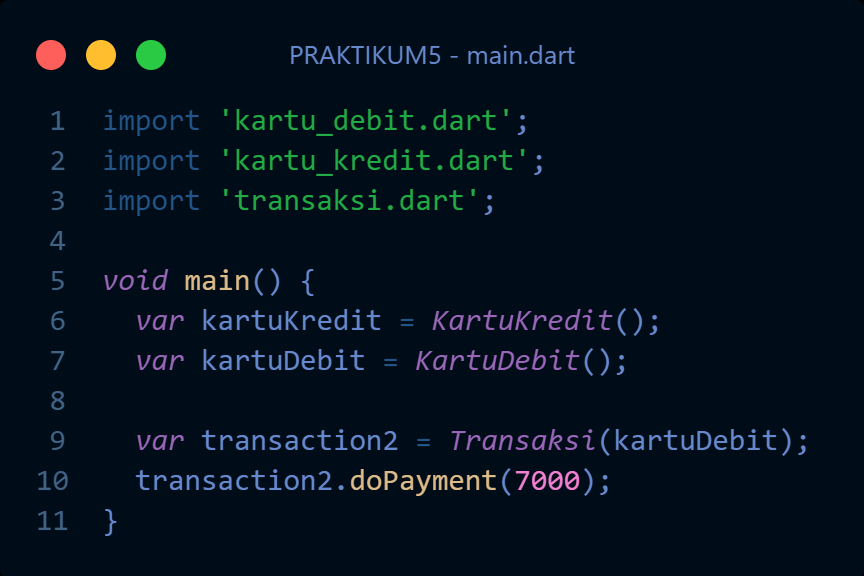
Kode program 5.38 praktikum5/dip/good/kartu\_kredit.dart



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



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

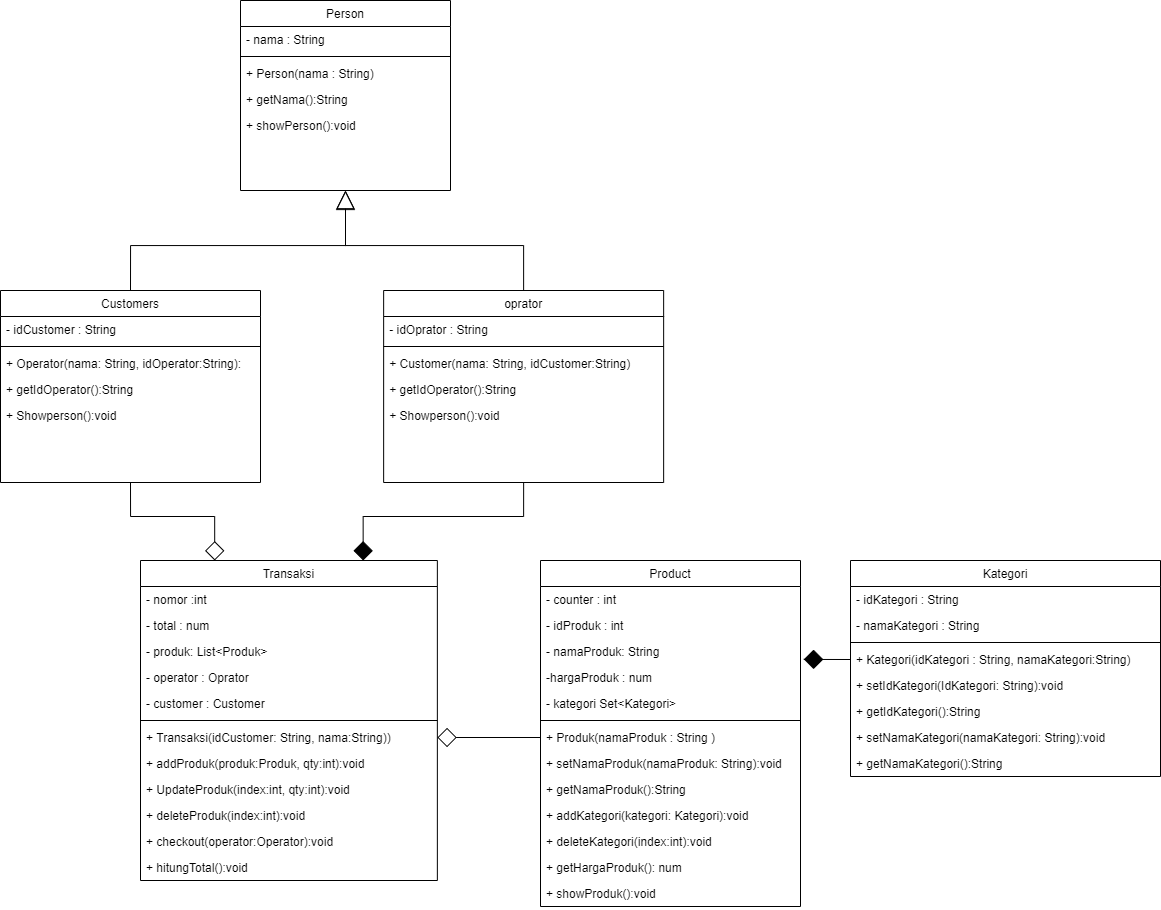


Hasil kode program 5.39



## TUGAS INDIVIDU

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



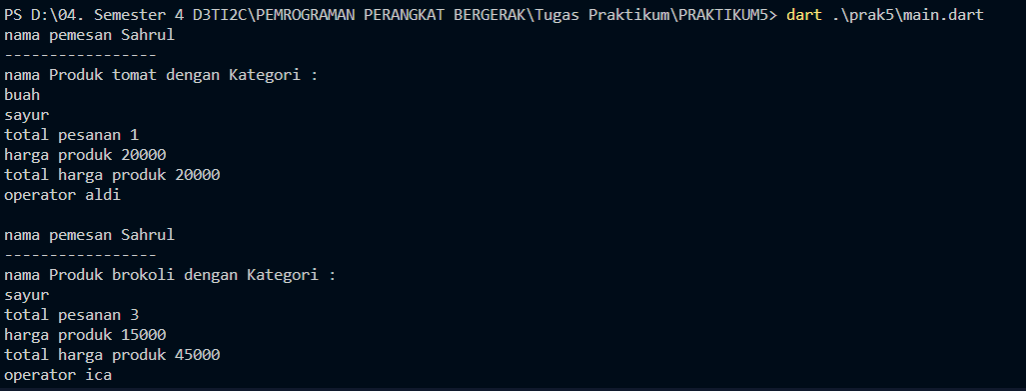
* + 1. Buatlah kode program dart berdasarkan studi kasus class diagram soal nomor 1!

|  |  |
| --- | --- |
|  |  |

Main.dart



Hasil Kode Program



## 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-05550](https://www.alpharithms.com/solid-guidelines-for-better-software-development-055500/) [0/](https://www.alpharithms.com/solid-guidelines-for-better-software-development-055500/). Diakses tanggal 14 Maret 2022.