**LAPORAN PRAKTIKUM**



|  |  |  |
| --- | --- | --- |
| **NIM** | **:** | 2003073 |
| **Nama** | **:** | Ica Natasya |
| **Kelas** | **:** | D3TI.2C |
| **Mata Kuliah** | **:** | **Pemrograman Perangkat Bergerak (TIU3403)** |
| **Praktikum ke / Judul** | **:** | 4/ Pemrograman Dart Berorientasi Objek |
| **Tanggal Praktikum** | **:** | 10 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 4**

PEMROGRAMAN DART BERORIENTASI OBJEK

1. **TUJUAN PRAKTIKUM**

**Tujuan Umum**

Mahasiswa memahami pembuatan kode program *Dart* dengan paradigma (pola pikir) berorientasi objek

**Tujuan Khusus**

Mahasiswa dapat

1. Menjelaskan konsep *class* dan *object*
2. Menjelaskan 4 pilar pemrograman berorientasi objek pada bahasa *Dart*
3. Menjelaskan relasi *class*
4. Menerapkan *exception handling* pada kode program *Dart*
5. Membuat kode program *Dart* berdasarkan pilar PBO, relasi *class*, dan mengimplementasikan *exception handling*
6. **TEORI SINGKAT**

***Class & Object***

Bahasa pemrograman *Dart* mendukung konsep dan paradigma berorientasi objek. Pada konsep tersebut dibutuhkan pemahaman dasar terkait *Class* dan *Object*. Adapun *class* adalah suatu rancangan atau *blueprint* dari suatu *object*. Sedangkan *object* adalah hasil instansiasi dari suatu *class*. Suatu class dapat diinstansiasi menjadi beberapa *object*. Dalam rancangan suatu *class* terdiri atas 3 bagian, yaitu nama *class*, properti atau atribut atau variabel, serta *method* atau operasi atau tingkah laku *(behaviour)*. Ilustrasi rancangan *class (class diagram)* disajikan pada Gambar 1.



**Gambar 1.** Ilustrasi Rancangan *Class (Class Diagram)* Mobil

Pada Gambar 1 dijelaskan bahwa terdapat suatu rancangan *Object* bernama Mobil. *Class* Mobil ini dapat diinstansiasi menjadi beberapa objek Mobil, misalnya mobil1, mobil2, dst. Pada *class* diagram ditunjukkan juga bahwa *class* Mobil memiliki ciri, yaitu merk dan cc. Hal ini berarti setiap objek yang akan diinstansiasi pasti memiliki kedua ciri (properti atau variabel) tersebut. Semua objek yang diinstansiasi dapat melakukan operasi setMerk(), setCc(), getJenis(), getCc() dan showInfoMobil(). Pada rancangan *class* tersebut berisi juga suatu *constructor* Mobil, artinya saat *class* Mobil diinstansiasi, dibutuhkan satu atau beberapa parameter berdasarkan rancangan yang disajikan pada *class* diagram. Pembuatan kode program berdasarkan rancangan *class* diagram, dapat dilihat pada gambar berikut (cara 1 dan cara 2):

| Cara 1 | Cara 2 |
| --- | --- |

Pembuatan kode program Mobil.dart dapat dilakukan dengan dua cara. Cara pertama, pembuatan *setter* dan *getter* dilakukan dengan tanpa penyingkatan kode program. Sedangkan cara kedua, jika diperhatikan terdapat penulisan kode program yang lebih singkat, khususnya pada *setter* dan *getter*. Keduanya dapat menghasilkan *output* yang sama, tetapi tentunya dengan cara pemanggilan yang berbeda. Adapun cara pemanggilan kode properti dan *method* dari *class* Mobil.dart dijelaskan pada contoh kode program MobilTest.dart.

Dilakukan pemanggilan *method* untuk mengubah nilai merk dan cc melalui cara yang berbeda. Hal tersebut sejalan dengan cara pendeklarasian *method setter* dan *getter*. Pada cara 1 berikut, dijelaskan pemanggilan *method* berdasarkan cara 1 pada Mobil.dart. Sebaliknya, cara 2 berikut berisi kode program yang memanggil *method* pada Mobil.dart dengan cara 2 sebelumnya.

|  |  |
| --- | --- |
| Cara 1 | Cara 2 |

Kedua cara tersebut dapat menghasilkan *output* yang sama. Berikut adalah potongan gambar *output* yang dihasilkan:



**PBO Pada Bahasa Pemrograman Dart dan Java**

Jika dibandingkan dengan bahasa pemrograman *Java*, ditemukan beberapa perbedaan pada bahasa pemrograman *Dart*, antara lain pendeklarasian *access modifier* dan *method overloading*. Tidak seperti bahasa pemrograman *Java*, pada bahasa pemrograman *Dart* hanya dikenal 2 (dua) *access modifier*, yaitu *public* dan *private*. Secara *default*, *access modifier* yang diberikan pada suatu variabel atau *method* bahasa pemrograman *Dart* adalah *public*. Selain itu, bahasa pemrograman *Dart* tidak mendukung konsep *method overloading*, di mana suatu *method* dapat dideklarasikan 2 kali dengan parameter yang berbeda, baik dalam penulisan urutan parameter, tipe data parameter, maupun jumlah parameter.

**Tabel 1.** Perbandingan Bahasa Pemrograman *Dart* dan *Java*

|  |  |  |  |
| --- | --- | --- | --- |
| No | Bahasa Pemrograman | *Access Modifier* | *Method Overloading* |
| 1 | *Java* | public  private  protected  default (tanpa *keyword*) | didukung |
| 2 | *Dart* | *private* (notasi *underscore*)  *public* (tanpa notasi) | tidak didukung |

**Pilar Pemrograman Berorientasi Objek**

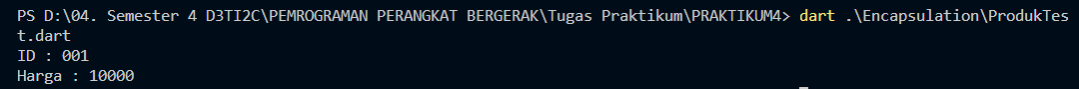
Pada bagian ini dijelaskan mengenai 4 (empat) pilar penyusun konsep pemrograman berorientasi objek, yaitu *encapsulation*, *inheritance*, *polymorphism,* dan *abstraction*. Selain dipaparkan tentang penjelasan konsep keempat pilar tersebut, diberikan juga contoh kode program yang mendukung penerapannya.

1. *Encapsulation*

Konsep ini berkaitan dengan pembatasan akses suatu objek terhadap suatu properti maupun *method*. Penjelasan lain yang lebih singkat mengenai *encapsulation* secara harfiah adalah pembungkusan atau pengkapsulan. Mengingat tidak adanya *access modifier* selain *public* dan *private* pada bahasa pemrograman *Dart*, maka konsep *encapsulation* dapat didukung dengan kedua *access modifier* tersebut.

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

**Hasil Program**



1. *Inheritance*

Konsep pewarisan suatu *property* dan *method* dari suatu *parent class* *(superclass)* ke *child class (subclass)*. Terdapat 5 (lima) jenis *inheritance* yaitu:

* *single inheritance* (1 *superclass* & 1 *subclass*)
* *multiple inheritance* (2 atau lebih *superclass* & 1 *subclass*)
* *multilevel inheritance* (1 *superclass*, 1 *subclass* atau *superclass*, & 1 subclass)
* *hierarchical inheritance* (1 *superclass* & 2 atau lebih *subclass*), dan
* *hybrid inheritance* (gabungan beberapa jenis *inheritance*).

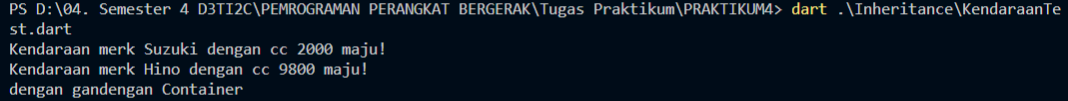
Berikut adalah contoh kode program *single inheritance* studi kasus Kendaraan dan Truk.



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

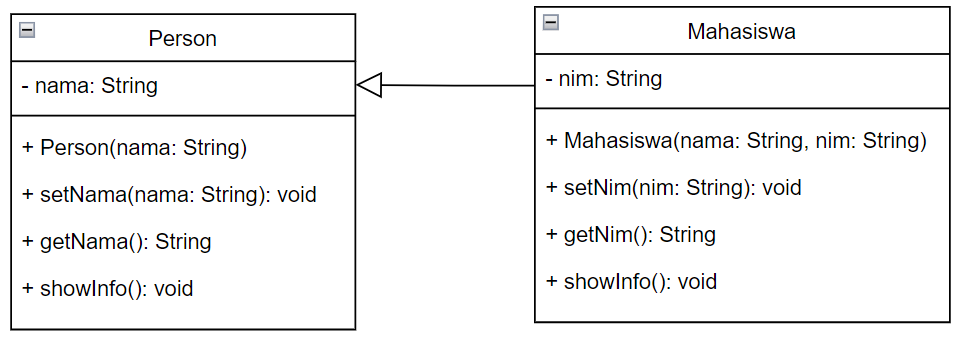


**Hasil Program**



1. *Polymorphism*

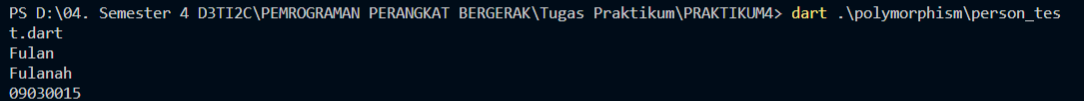
Konsep pewarisan dengan modifikasi suatu sifat atau perilaku dari *parent* ke *child*. Artinya *child class* dimungkinkan terdapat suatu *method* yang namanya sama dengan *parent class* tetapi *body method*-nya (deklarasinya) berbeda. Konsep ini memiliki kesamaan pada bahasa pemrograman *Java*, dimana terdapat konsep *overriding*.



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



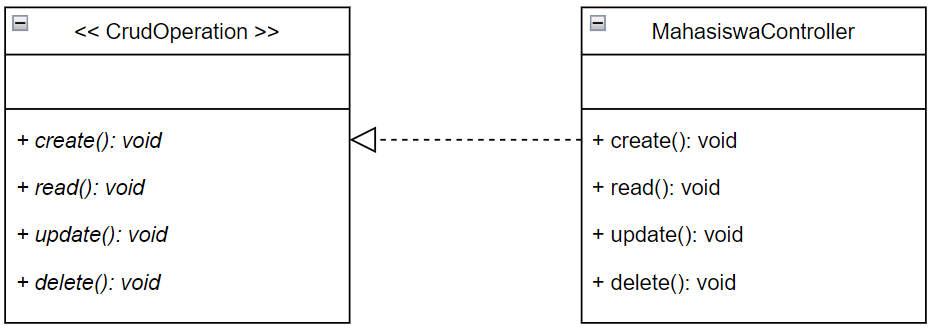
**Hasil Program**



1. *Abstraction*

Konsep deklarasi suatu *class* yang tidak dapat diinstansiasi menjadi objek, artinya *class* tersebut menjadi suatu abstraksi dari *class* yang lain (turunannya). Bahasa pemrograman *Dart* memiliki kesamaan konsep *abstraction* dengan *Java*, di mana terdapat suatu *abstract* dan *interface*. *Interface* mengizinkan pembuatan struktur *class* yang hanya berisi *method* abstrak (tidak ada deklarasinya). Sebaliknya, *abstract class* dapat berisi satu atau beberapa deklarasi *method*, selama ada minimal satu *method* abstrak. Berikut adalah contoh implementasi kode program *interface* dan *abstract*.

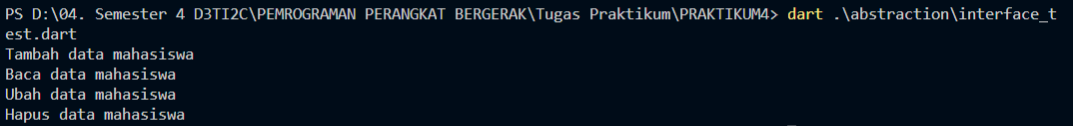
* *Interface*



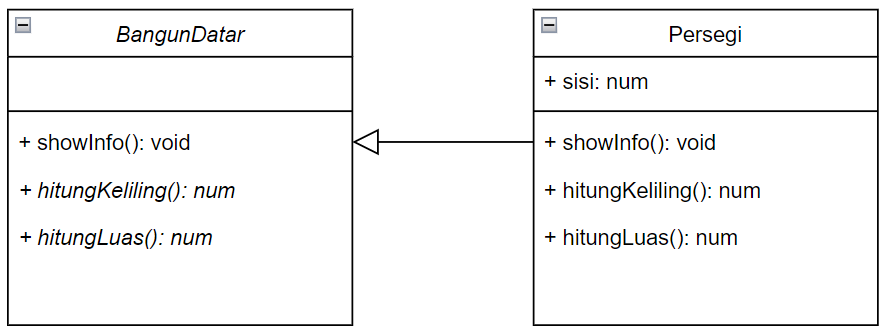
|  |  |
| --- | --- |
|  |  |



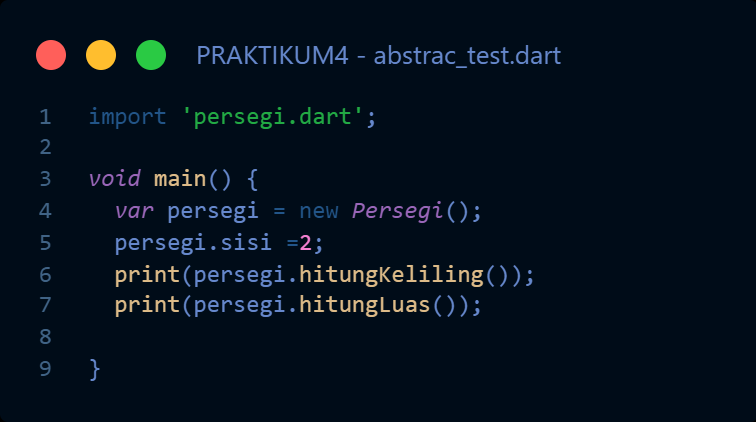
**Hasil Program**



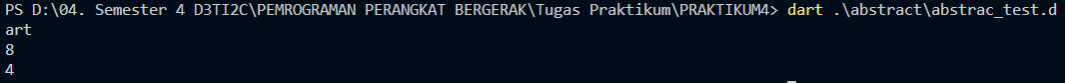
* *Abstract*



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



**Hasil Program**

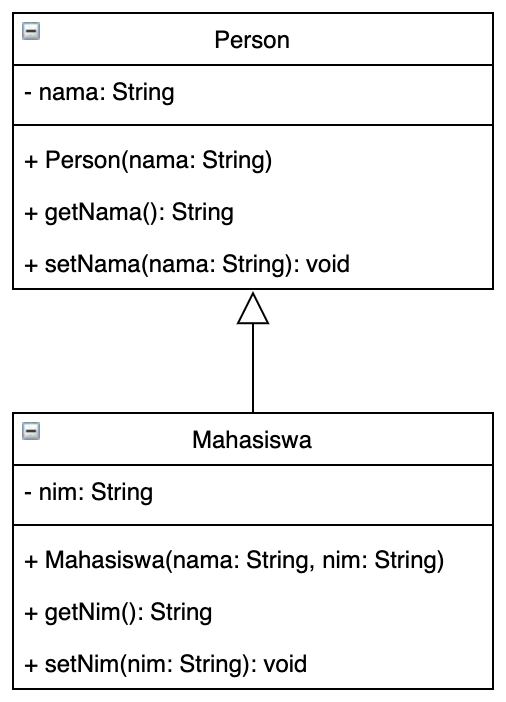


**Relasi Class**

Dalam implementasi penyelesaian masalah, suatu objek akan bekerja sama (interaksi) dengan objek yang lain, sehingga diperlukan rancangan relasi antar class. Adapun jenis relasi antar class dibagi menjadi 4 (empat) yaitu generalization, composition, aggregation, dan association.

1. Generalization / Inheritance

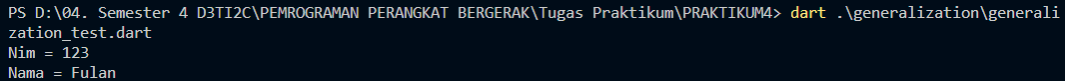
Relasi class yang menjelaskan suatu pewarisan (IS-A) property dan method dari suatu parent class ke child class. Sebagai contoh diberikan gambaran relasi generalization antara Person dan Mahasiswa.



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

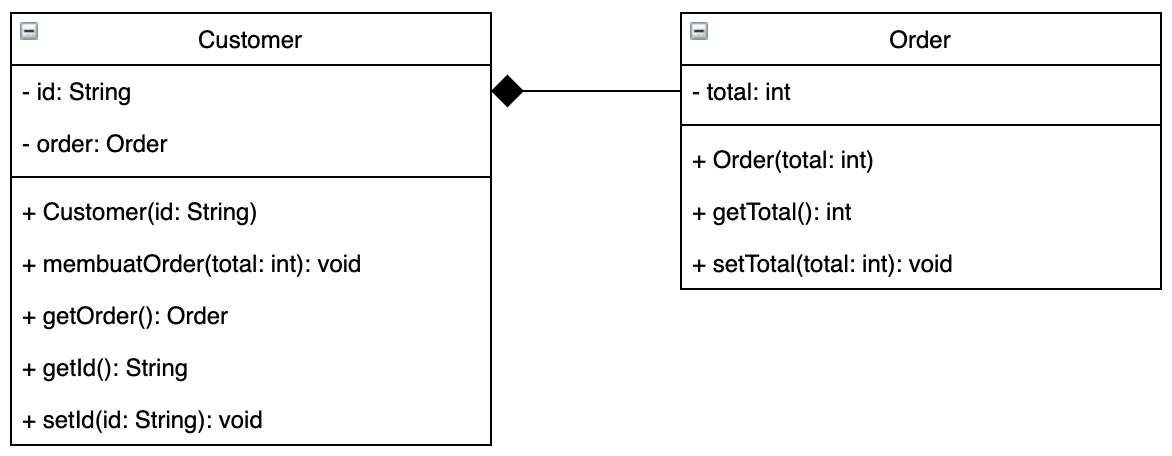


**Hasil Program**

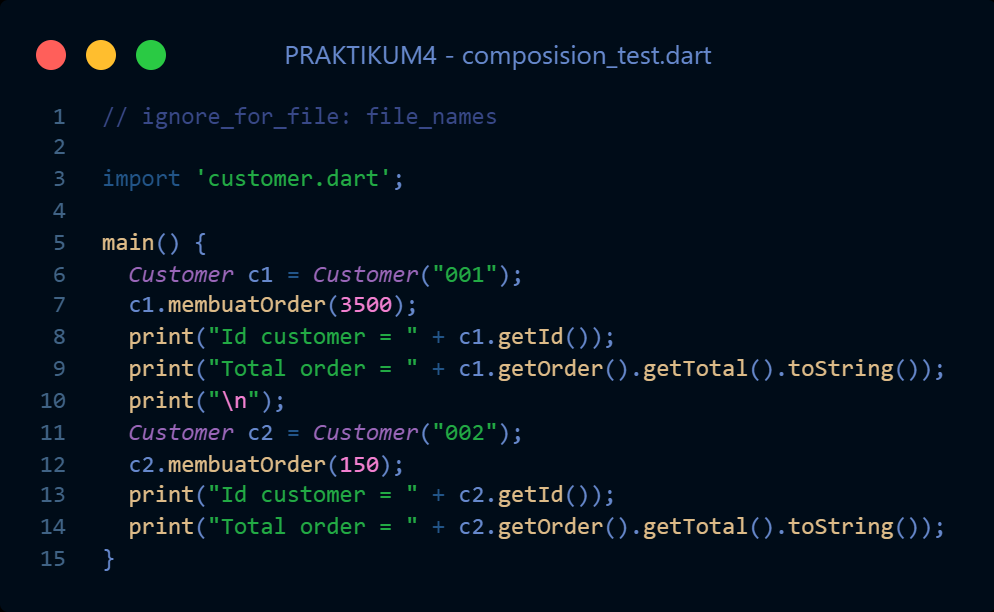


1. Composition

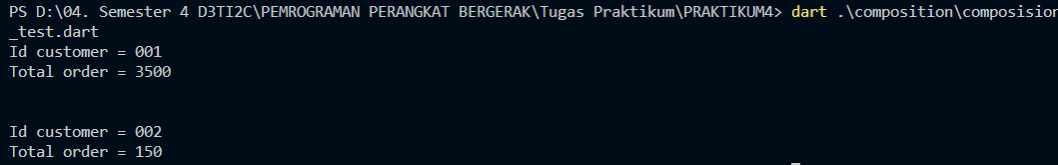
Relasi class yang menjelaskan bahwa suatu class dapat menjadi bagian dari (PART-OF) class yang lain. Hal ini dapat diartikan bahwa dimungkinkan suatu objek dari suatu class tidak akan ada apabila class lain tidak ada. Berikut adalah contoh relasi composition antara Customer dengan Order. Order tidak akan ada jika Customer tidak ada.



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

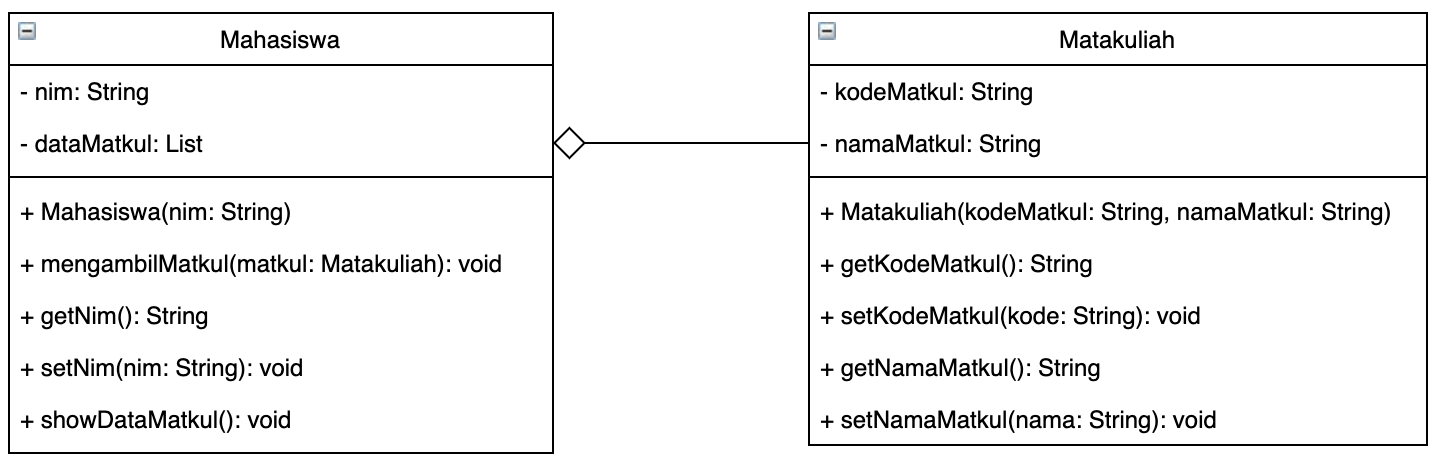


**Hasil Program**



1. Aggregation

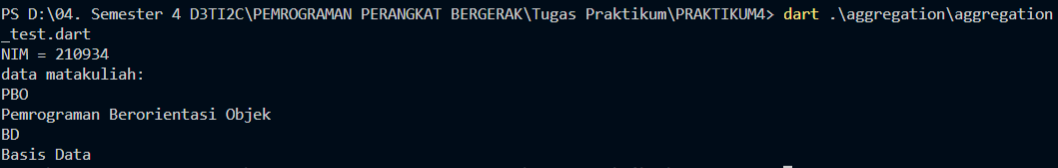
Relasi pada class diagram yang digunakan untuk menunjukkan bahwa suatu class memiliki referensi class lain (HAS-A).



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

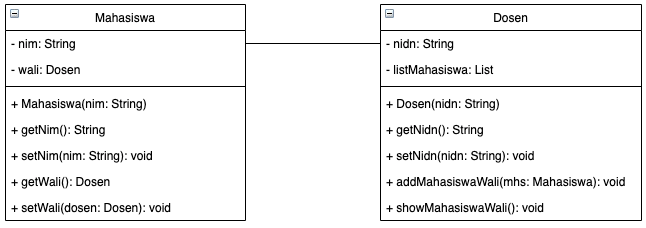


**Hasil Program**



1. Association

Relasi pada class diagram yang digunakan untuk menunjukkan dua class yang menyimpan referensi class satu sama lain.



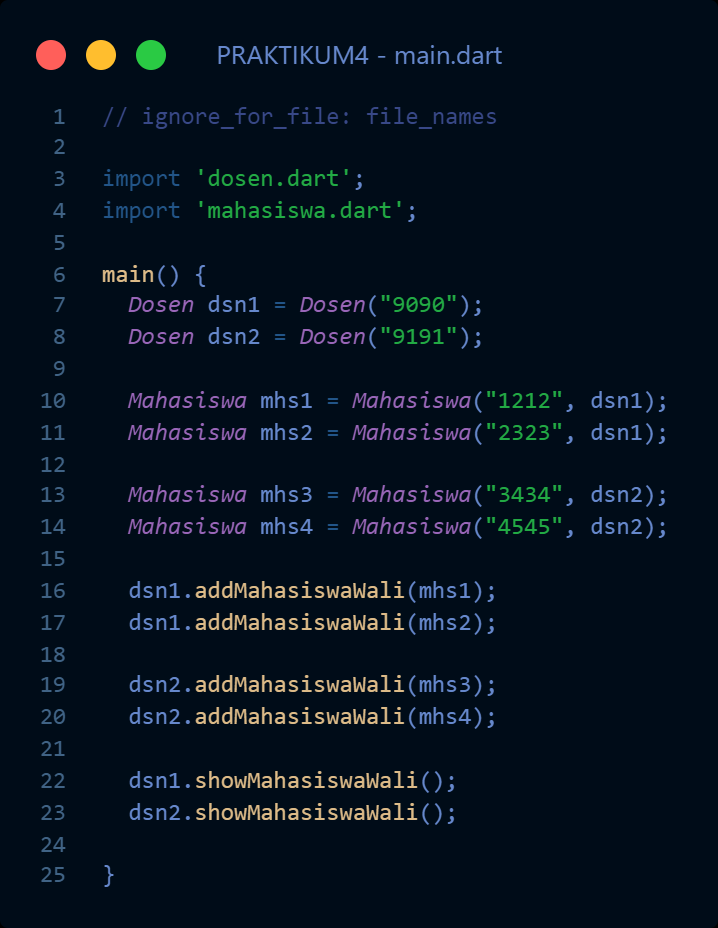
Mahasiswa.dart



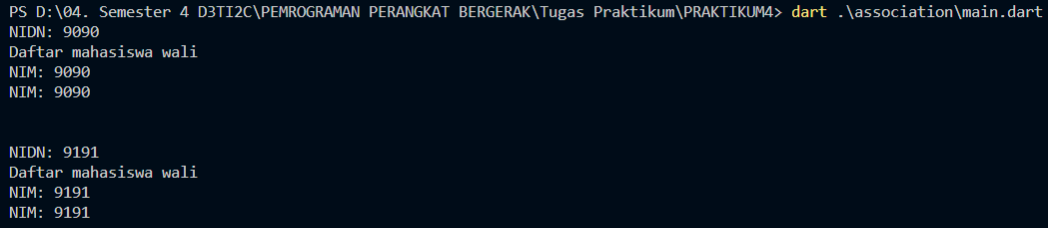
Dosen.dart



Main.dart

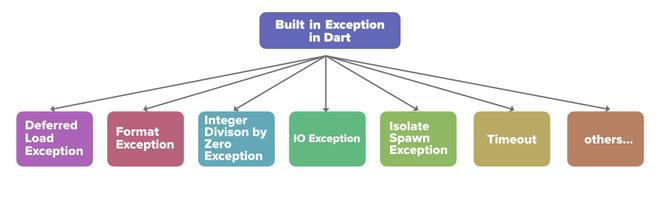


**Hasil Program**



**Exception Handling**

Bahasa pemrograman Dart dapat memberikan suatu exception untuk menandakan suatu perilaku kode program yang tidak diharapkan (unexpected) atau yang salah telah terjadi selama eksekusi. Saat hal tersebut terjadi, kode program melempar (throw) suatu exception, sehingga kita perlu menangkapnya (catch). Jika tidak, maka program akan dihentikan secara paksa. Ilustrasi jenis exception pada Dart dapat dilihat pada Gambar 4.1.

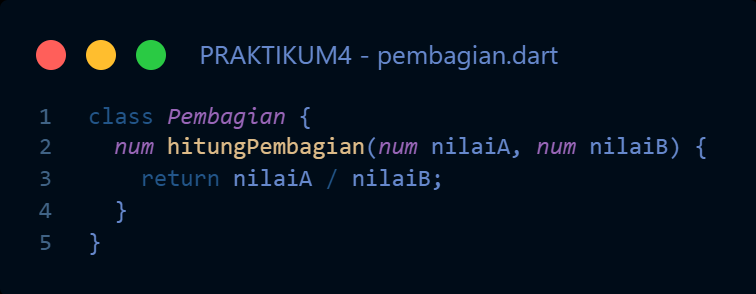


**Gambar 4.1** *Exception* Pada Bahasa Pemrograman *Dart*

Untuk dapat menangani *exception* yang mungkin muncul, maka diperlukan suatu struktur program yang tersusun atas *block* try dan *block* on atau *block* catch.

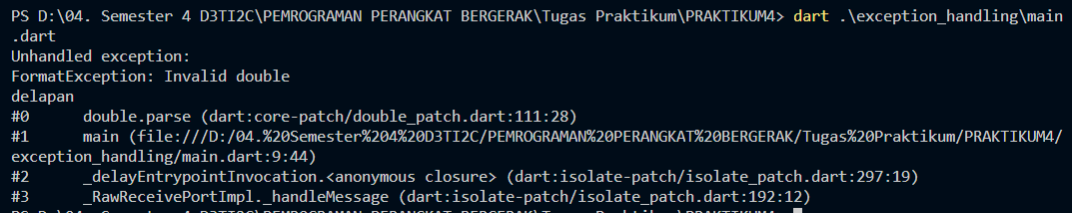
|  |
| --- |
| try {  // kode yang mungkin melempar (throw) exception  }  on Exception1 {  // kode untuk handling exception  }  catch Exception2 {  // kode untuk handling exception  } |

Sebagai contoh diberikan potongan kode program yang berujung pada pelemparan (*throw*) suatu *exception* yang berasal dari pembagian suatu nilai (num) berdasarkan hasil konversi suatu *string*.





**Hasil Program**

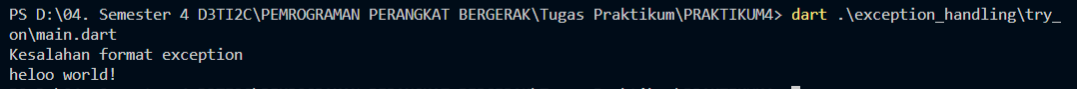


Untuk menangani hal tersebut, maka diperlukan implementasi block try..on atau try..catch pada Main.dart.

* try..on



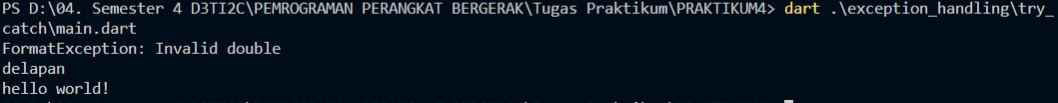
**Hasil Program**



* try..catch



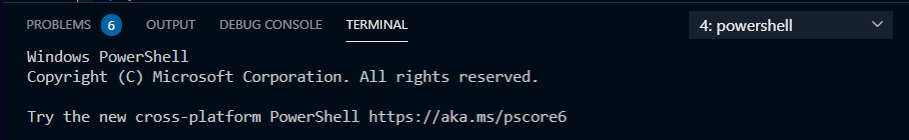
**Hasil Program**



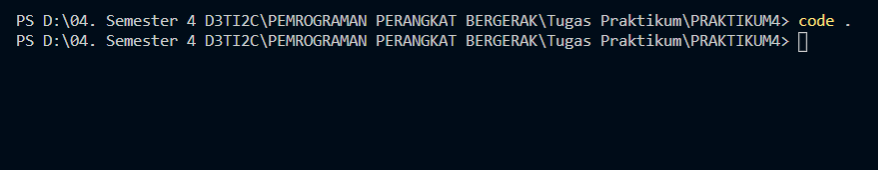
1. **LANGKAH DAN HASIL PELAKSANAAN PRAKTIKUM**

Langkah-langkah praktikum

1. Buatlah folder bernama “praktikum4” pada *File Explorer*
2. Buka *command line* pada *folder* “praktikum4” dengan cara tekan *Shift* + klik kanan pada area di dalam *folder*, kemudian pilih *Open PowerShell window here* atau *Git Bash Here*



1. Ketik perintah “code .” lalu tekan Enter



1. Buatlah folder baru di dalam “praktikum4” bernama “classobject” dengan cara klik *icon New Folder*, lalu beri nama classobject

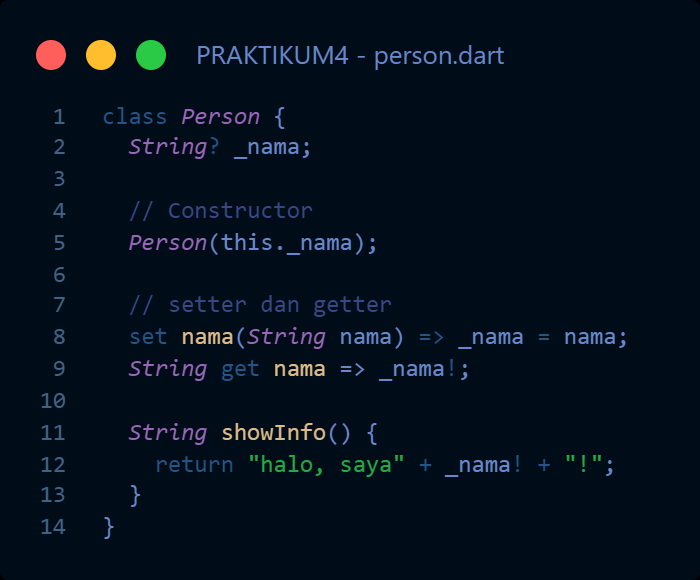
|  |  |
| --- | --- |
|  |  |

1. Buatlah *file* baru di dalam *folder* “classobject” bernama “Person.dart” dengan cara klik kanan pada folder classobject, lalu pilih New File

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

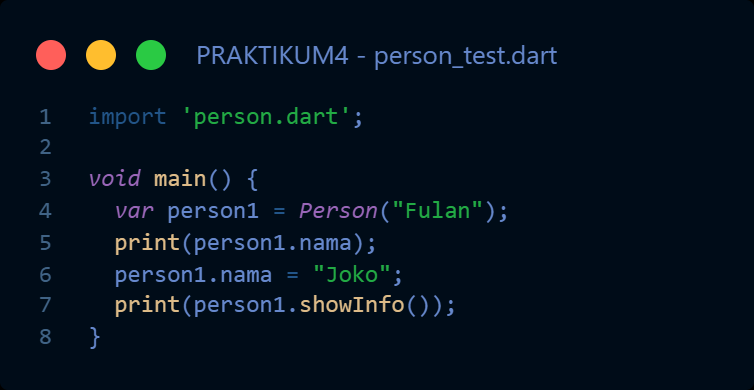
1. Tulis dan simpan kode program 4.1 berikut

Kode program 4.1. Person.dart

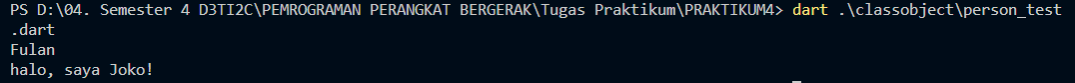


1. Buatlah file baru di dalam folder “classobject” bernama “PersonTest.dart”
2. Tulis dan simpan kode program 4.2 berikut

Kode program 4.2 PersonTest.dart



1. Jalankan program dengan cara klik menu Terminal > New Terminal, lalu ketik perintah berikut



1. Selesai
2. **LATIHAN**

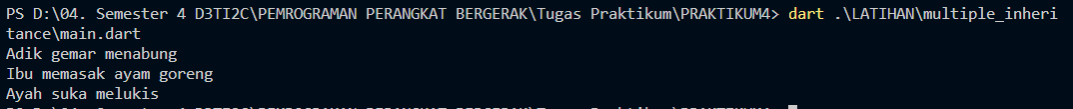
**Latihan *Multiple* *Inheritance***

1. Buatlah folder baru bernama “multipleinheritance” di dalam folder praktikum4.
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut:



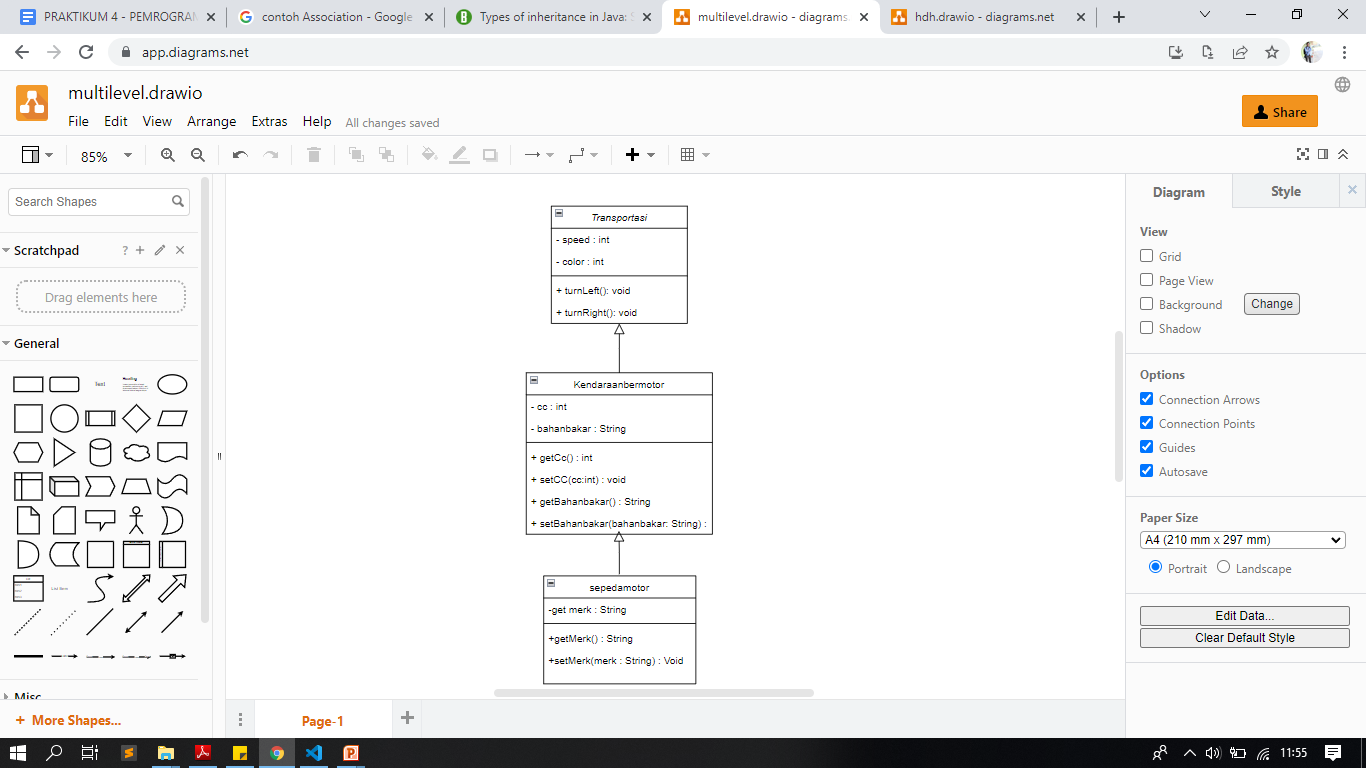
|  |  |
| --- | --- |
|  |  |

**Hasil Program**



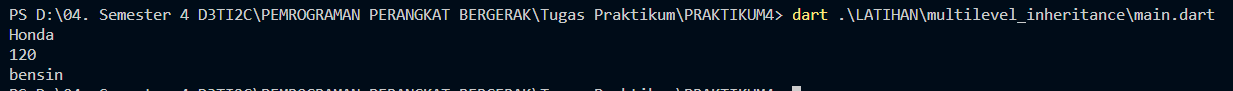
**Latihan *Multilevel Inheritance***

1. Buatlah folder baru bernama “multilevelinherintance” di dalam folder praktikum4
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut



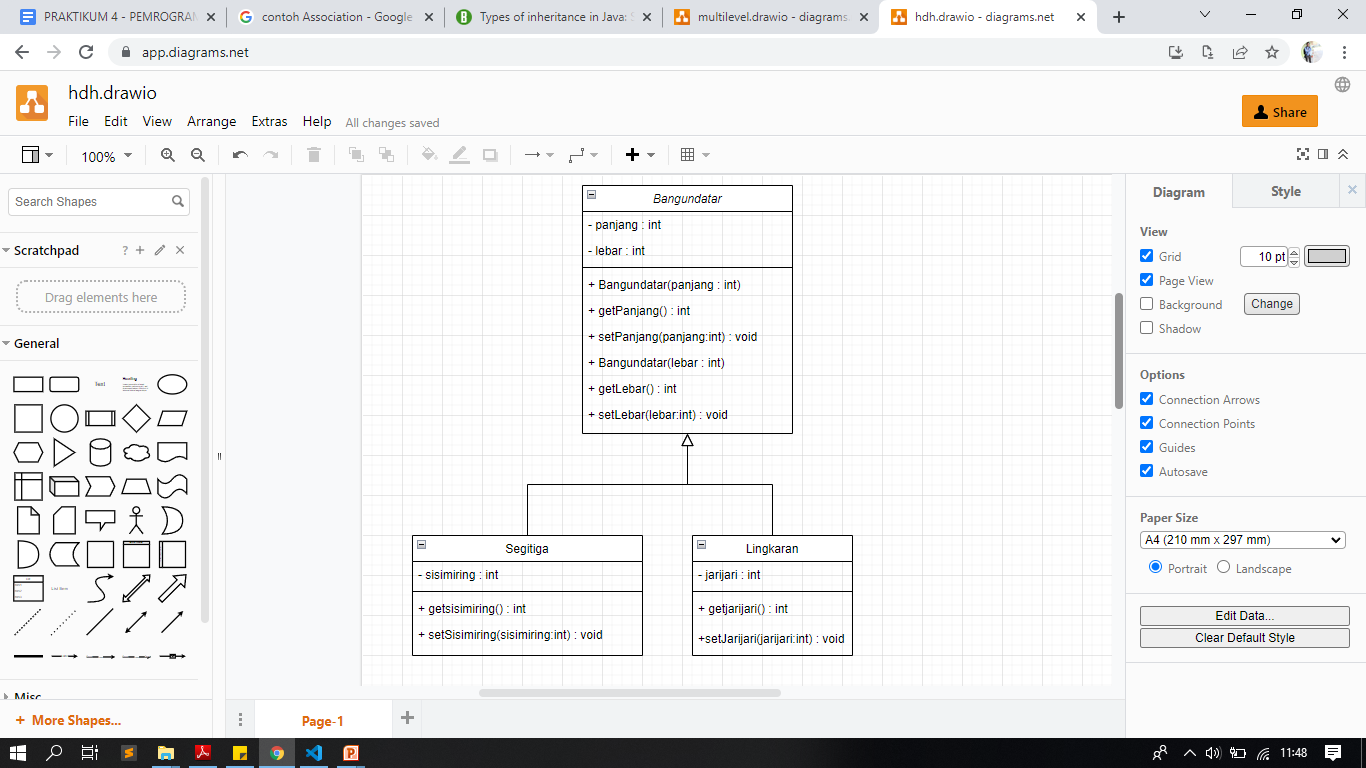
|  |  |
| --- | --- |
|  | main.dart |

**Hasil Program**

****

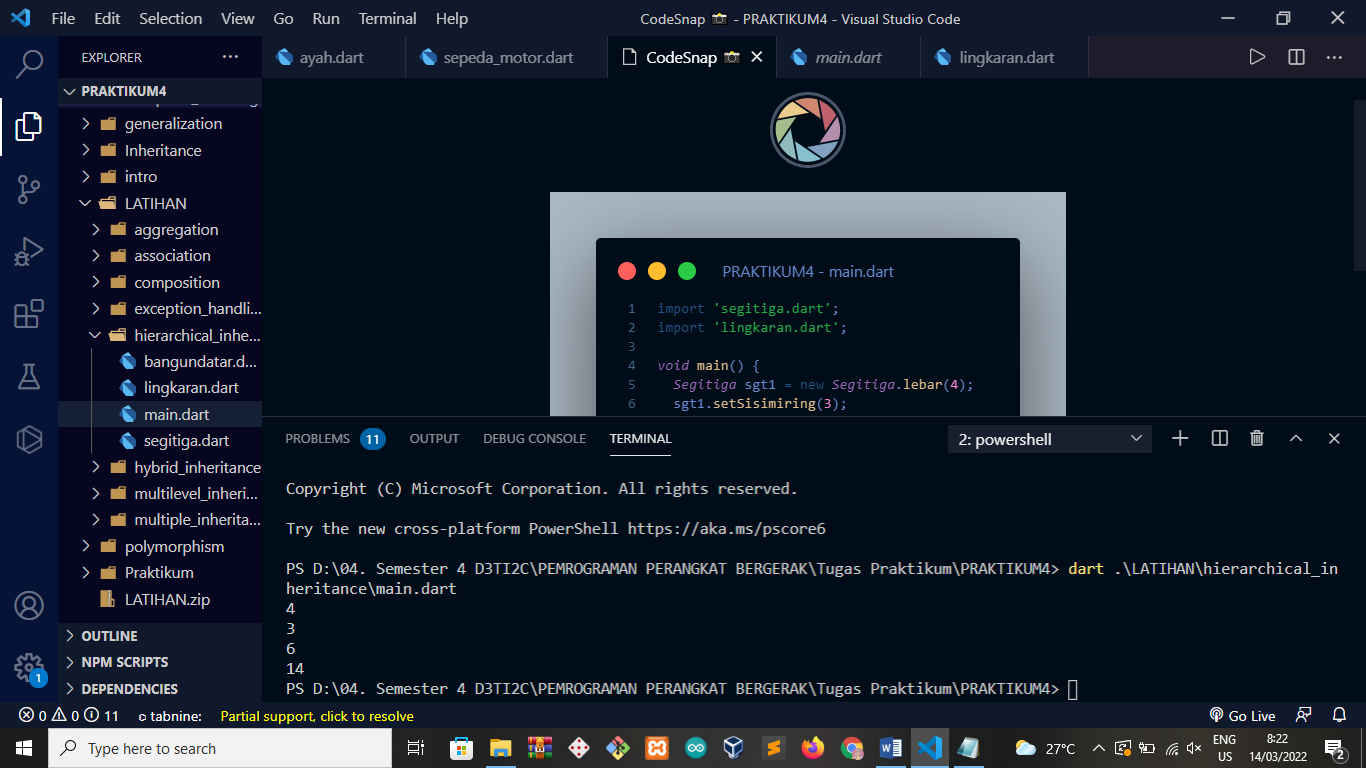
**Latihan *Hierarchical Inheritance***

1. Buatlah folder baru bernama “hierarchical\_inherintance” di dalam folder praktikum4
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut:



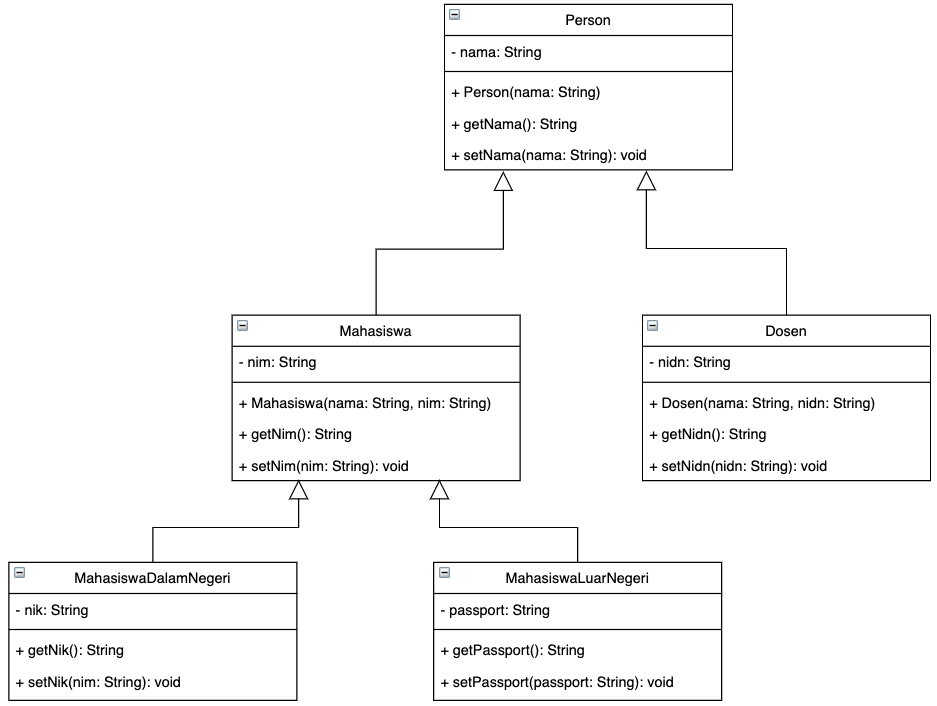
|  |  |
| --- | --- |
|  |  |

**Hasil Program**



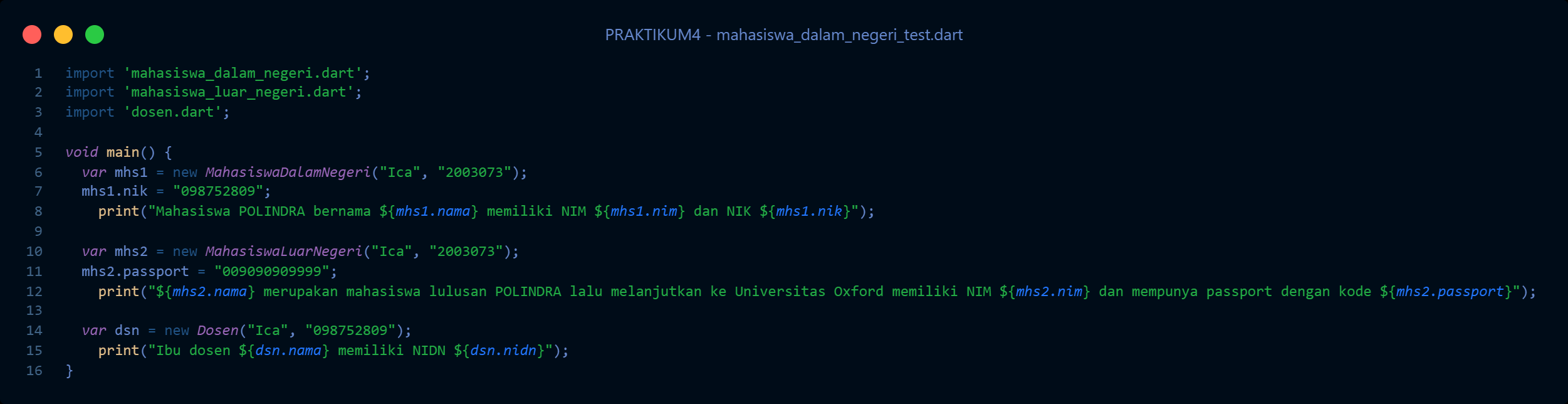
**Latihan *Hybrid Inheritance***

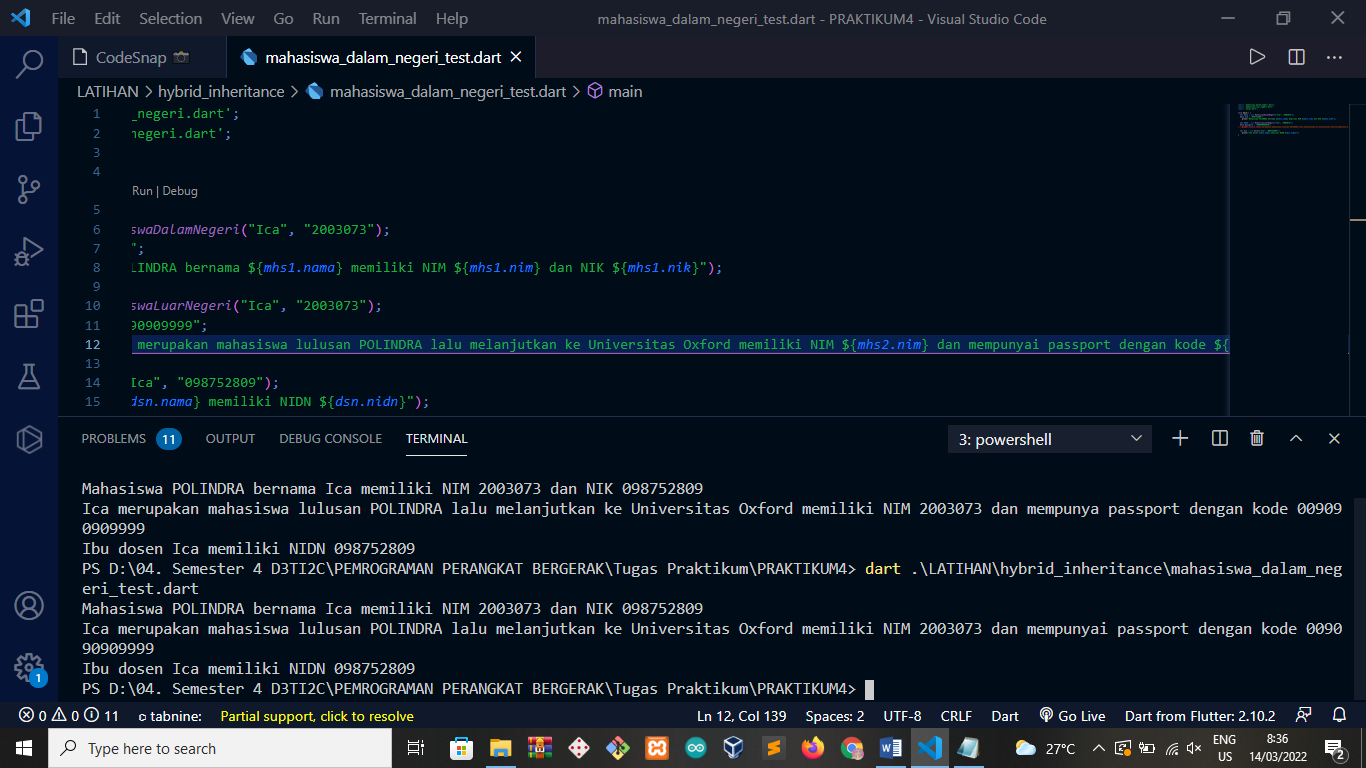
1. Buatlah folder baru bernama “hybridinheritance” di dalam folder praktikum4.
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut:



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

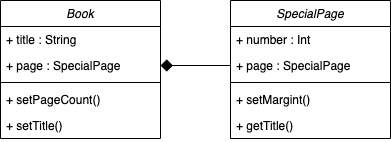
Main.dart



**Hasil Program**

**Latihan *Composition***

1. Buatlah folder baru bernama “composition” di dalam folder praktikum4.
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut:

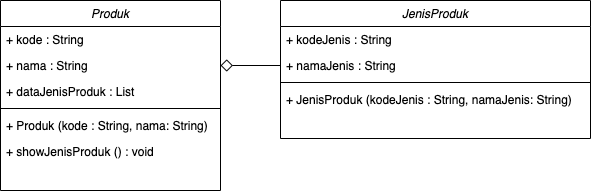


|  |  |
| --- | --- |
|  | main.dart |

**Hasil Program**

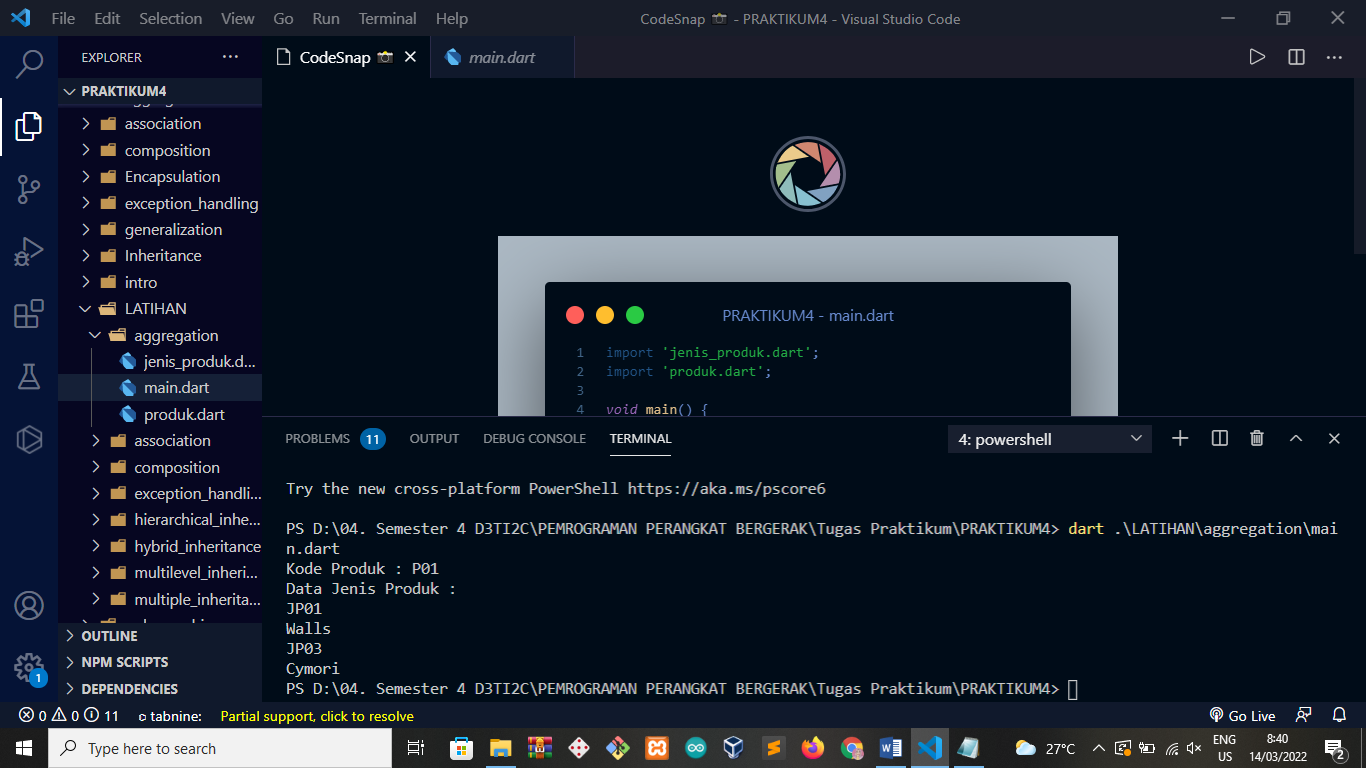
**Latihan *Aggregation***

1. Buatlah folder baru bernama “aggregation” di dalam folder praktikum4.
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut:



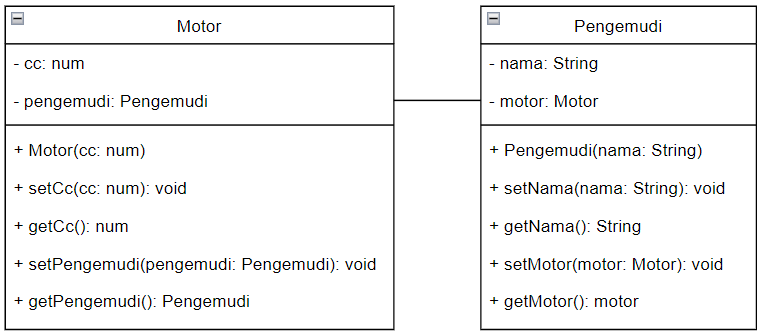
|  |  |
| --- | --- |
|  | main.dart |

**Hasil Program**



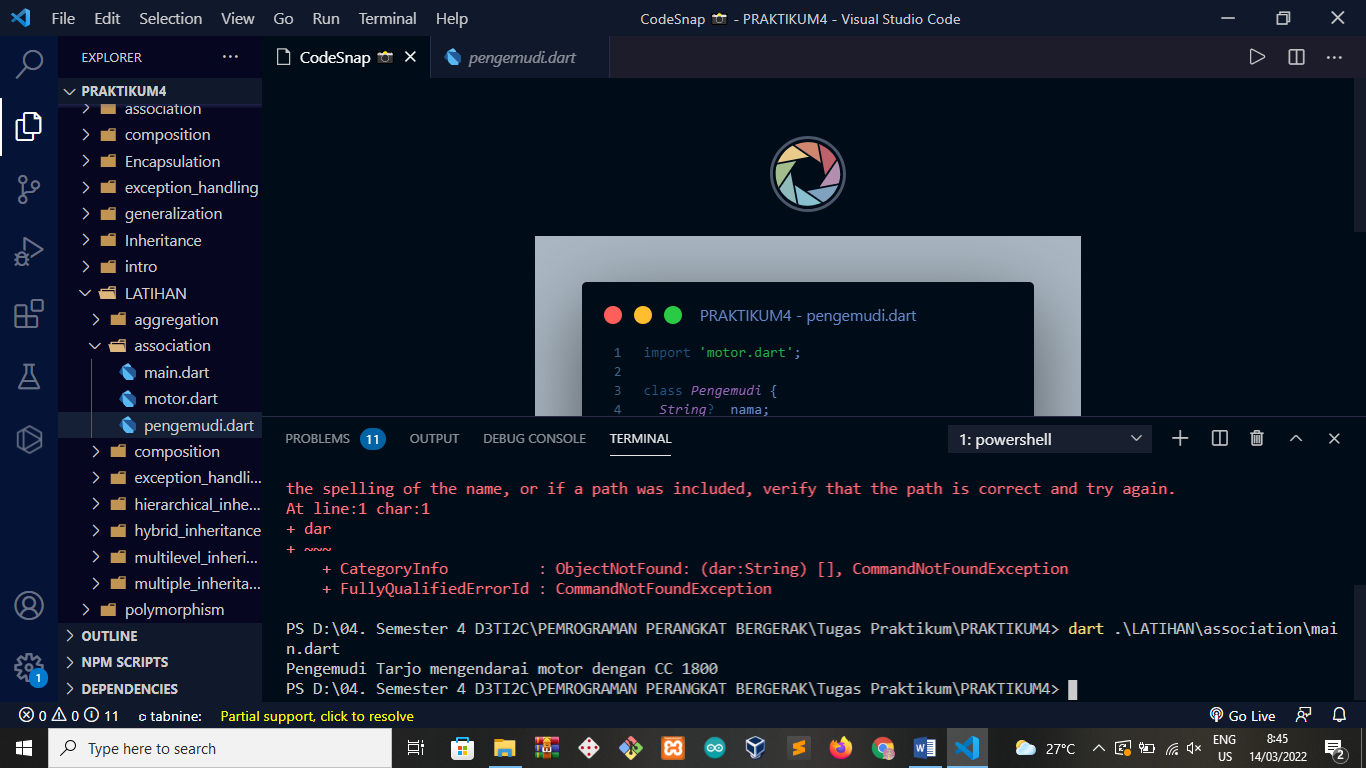
**Latihan *Association***

1. Buatlah folder baru bernama “association” di dalam folder praktikum4.
2. Buatlah implementasi kode program dart berdasarkan diagram UML berikut:



|  |  |
| --- | --- |
|  | main.dart |

**Hasil Program**



**Latihan *Exception Handling***

Buatlah sebuah file dengan ekstensi \*.dart di dalam folder praktikum4, lalu ketik

dan simpanlah beberapa kode program berikut, serta tampilkan *(screenshot)*

hasilnya!

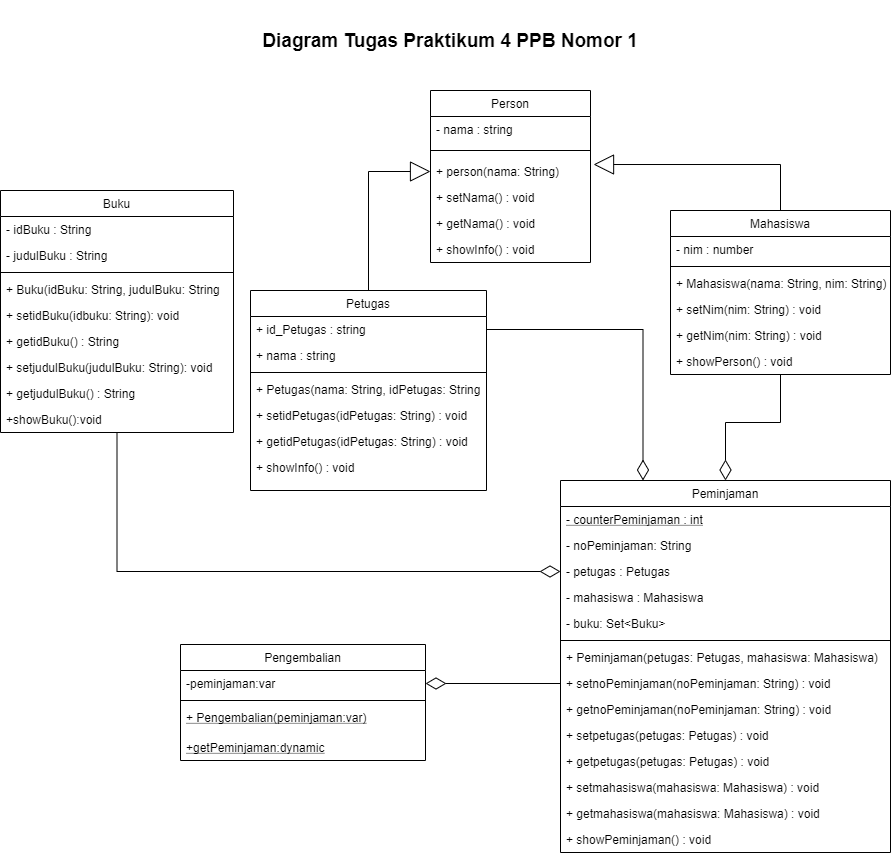
Kode *scripting* 4.3 praktikum4/exceptionhandling/ContohException.dart



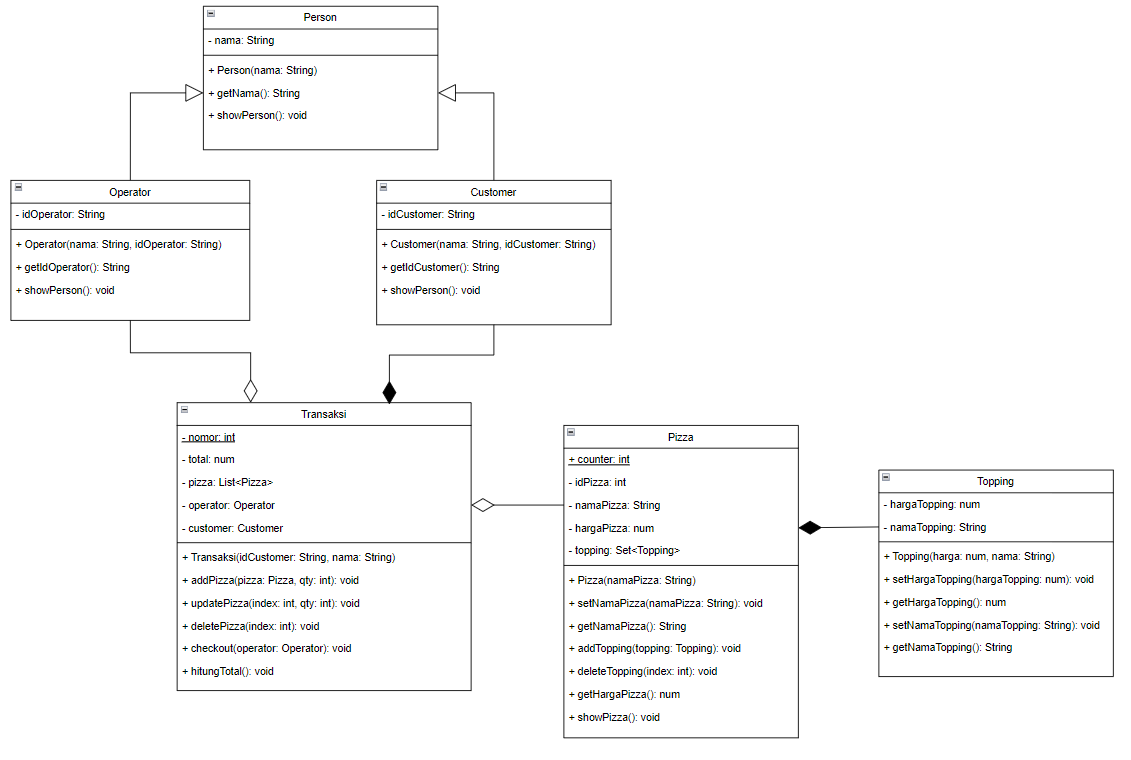
**Hasil kode program 4.3.**

|  |
| --- |
|  |

1. **TUGAS INDIVIDU**
2. Buatlah *class diagram* beserta relasinya berdasarkan studi kasus Perpustakaan yang terdiri dari class Buku, Mahasiswa, Petugas, Peminjaman, dan Pengembalian

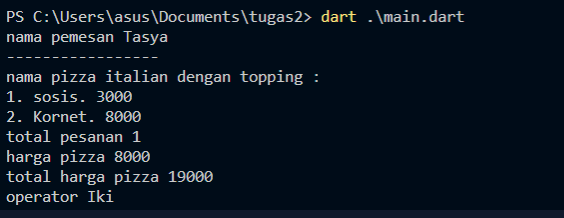


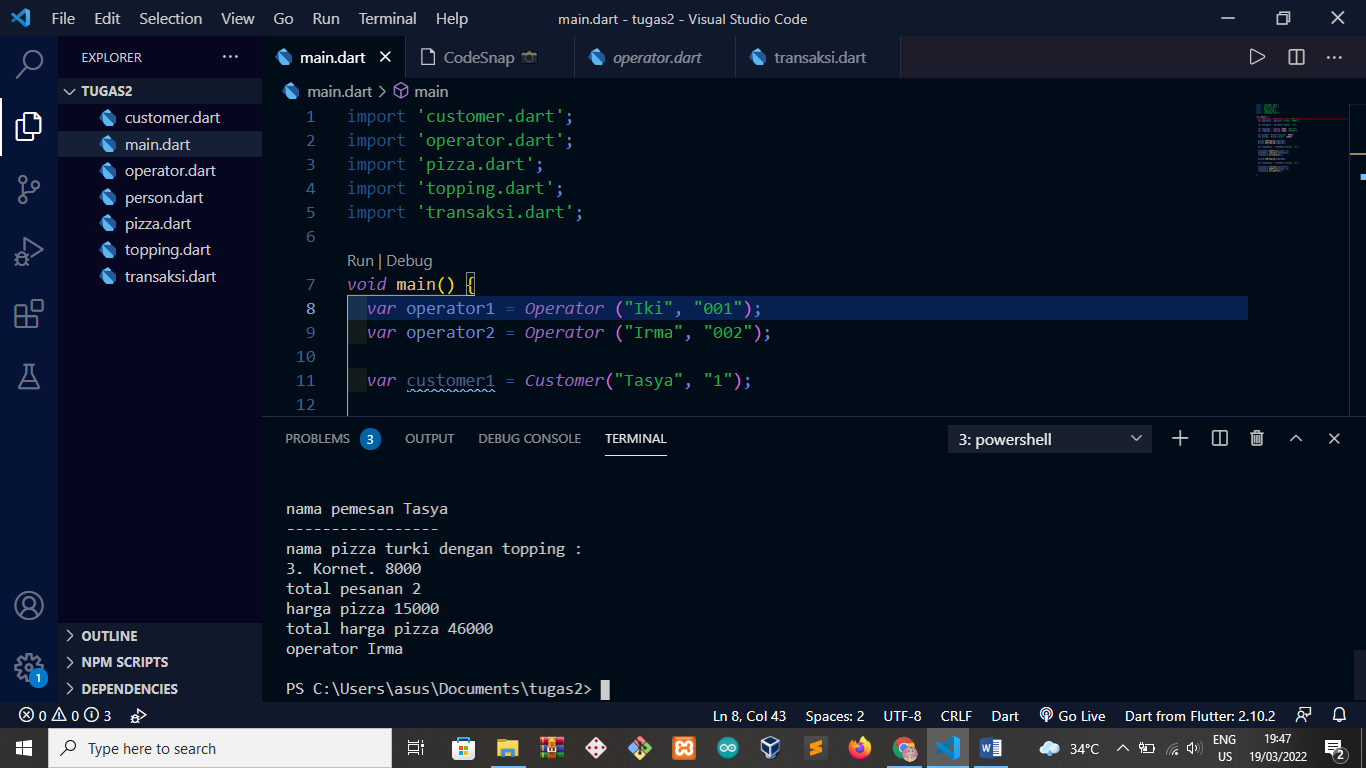
1. Buatlah kode program berdasarkan relasi *class diagram* berikut



|  |  |
| --- | --- |
|  | Main.dart |

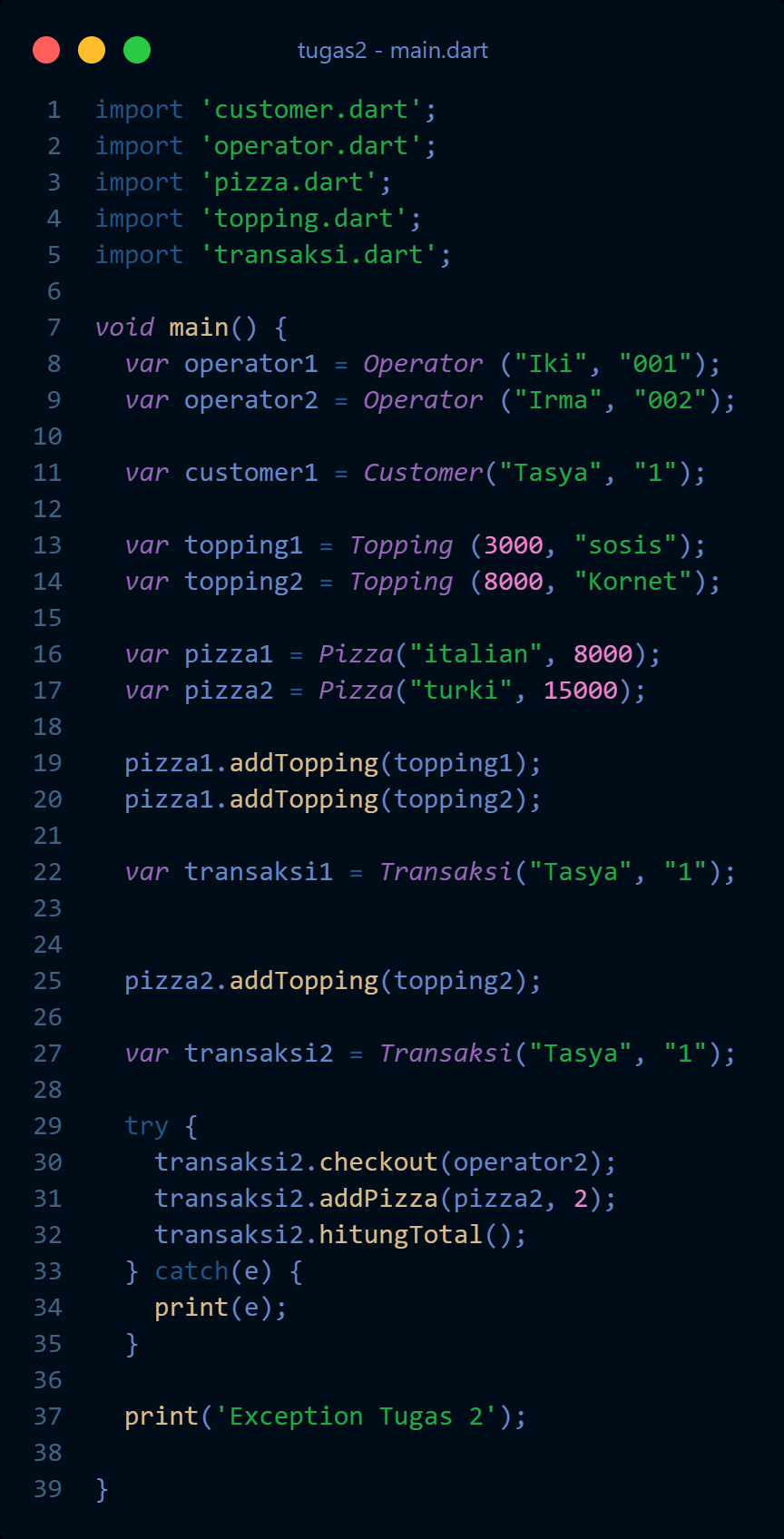
**Hasil Kode Program**

****

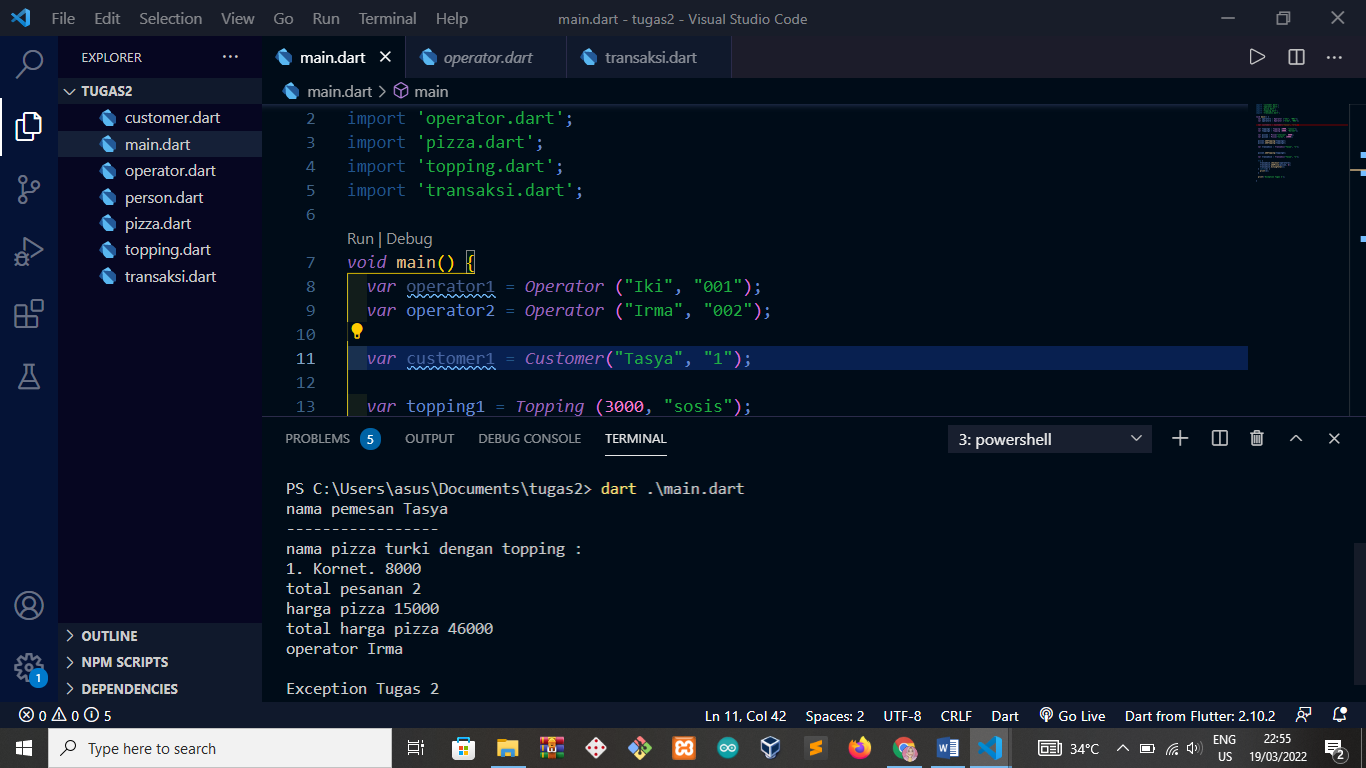


1. Terapkanlah exception handling untuk mencegah kesalahan yang mungkin dilakukan oleh user berdasarkan studi kasus nomor 1 atau 2!

Kode Program main.dart nomor 2 yang sudah menerapkan exception handling



**Hasil Kode Program nomor 2**



1. **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.