

# 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

#### A. 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*

#### B. 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

```
PRAKTIKUM4 - Mobil.dart

1 class Mobil {
2
3 // property atau variabel atau attribute
4 String? _merk;
5 int? _cc;
6
7 // Constructor
8 Mobil(this._merk, this._cc);
9
10 // method setter merk
11 void setMerk(String merk) {
12     this._merk;
13 }
14
15 // method setter cc
16 void setCc(int cc) {
17     this._cc;
18 }
19
20 // method getter merk
21 String getMerk() {
22     return this._merk!;
23 }
24
25 // method getter cc
26 int getcc() {
27     return this._cc!;
28 }
29
30 void showInfoMobil() {
31     print (_merk);
32     print (_cc);
33 }
34 }
35
36 // set merk(String merk) => _merk = merk;
37 // set cc(int cc) => _cc = cc;
```

#### Cara 2

```
PRAKTIKUM4 - Mobil2.dart

1 class Mobil {
2 // property atau variabel atau attribute
3 String? _merk;
4 int? _cc;
5
6 // constructor
7 Mobil(this._merk, this._cc);
8
9 // setter
10 set merk(String merk) => _merk = merk;
11 set cc(int cc) => _cc = cc;
12
13 // getter
14 String get merk => _merk!;
15 int get cc => _cc!;
16
17 void showInfoMobil() {
18     print(_merk);
19     print(_cc);
20 }
21 }
```

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

```
PRAKTIKUM4 - MobilTest.dart

1 import 'Mobil.dart';
2
3 void main() {
4   Mobil mobil1 = new Mobil("Toyota", 1500);
5   mobil1.setMerk("Honda");
6   mobil1.setCc(2000);
7   mobil1.showInfoMobil();
8 }
9
```

Cara 2

```
PRAKTIKUM4 - MobilTest2.dart

1 import 'Mobil2.dart';
2
3 void main() {
4   Mobil mobil1 = new Mobil("Toyota", 1500);
5   mobil1.merk = "Honda";
6   mobil1.cc = 2000;
7   mobil1.showInfoMobil();
8 }
9
```

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

```
PS D:\04, Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\MobilTest2.dart
Honda
2000
```

### 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	<i>Java</i>	public private protected default (tanpa <i>keyword</i> )	didukung
2	<i>Dart</i>	<i>private</i> (notasi <i>underscore</i> ) <i>public</i> (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.

<pre>PRAKTIKUM4 - Produk.dart  1 class Produk { 2   // Property menjadi private 3   // karena diawali underscore ( _ ) 4   String? _id; 5   num? _harga; 6 7   // constructor 8   Produk(this._id, this._harga); 9 10  // setter 11  set id(String id) =&gt; _id; 12  set harga(num harga) =&gt; _harga; 13 14  // getter 15  String get id =&gt; _id; 16  num get harga =&gt; _harga; 17 }</pre>	<pre>PRAKTIKUM4 - ProdukTest.dart  1 import 'Produk.dart'; 2 3 void main() { 4   Produk produk1 = new Produk("001", 10000); 5   print("ID : " + produk1.id); 6   print("Harga" + produk1.harga.toString()); 7 }</pre>
---	---

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\tugas Praktikum\PRAKTIKUM4> dart .\Encapsulation\ProdukTest.dart
ID : 001
Harga : 10000
```

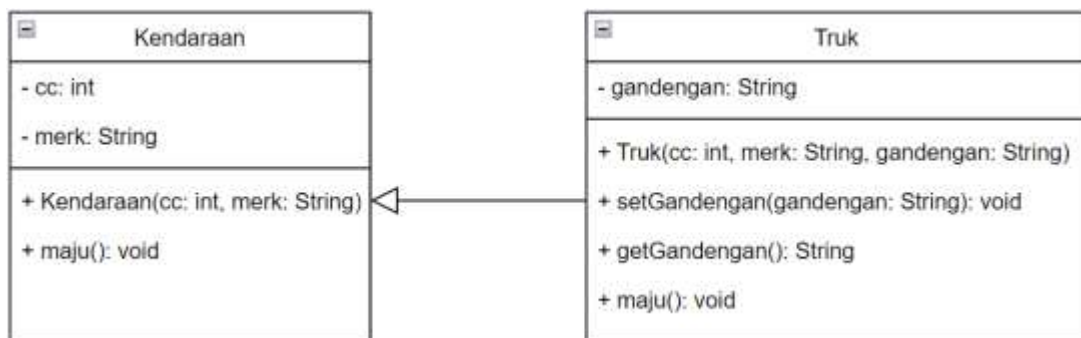


## 2. Inheritance

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.



```
PRAKTIKUM4 - Kendaraan.dart

1 import 'dart:io';
2 class Kendaraan {
3   // Property
4   int? _cc;
5   String? _merk;
6
7   // Constructor
8   Kendaraan(this._cc, this._merk);
9
10  void maju(){
11    print("Kendaraan merk " +
12      this._merk! +
13      "dengan cc " +
14      this._cc!.toString() +
15      " maju!");
16  }
17 }
18 }
```

```
PRAKTIKUM4 - Truk.dart

1 import 'dart:io';
2 import 'Kendaraan.dart';
3
4 class Truk extends Kendaraan {
5   // Property
6   String? _gandengan;
7
8   // Constructor
9   Truk(int? cc, String? merk, this._gandengan):
10     super(cc, merk);
11
12   // Getter & Setter
13   set gandengan(String gandengan) => _gandengan;
14   String get gandengan => _gandengan!;
15
16   void maju() {
17     super.maju();
18     stdout.write("dengan gandengan " + this._gandengan!);
19   }
20 }
```



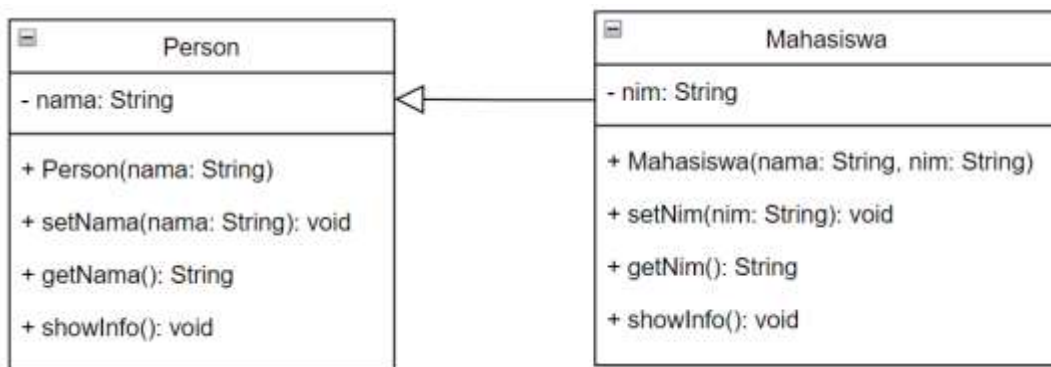
```
PRAKTIKUM4 - KendaraanTest.dart
1 import 'dart:io';
2 import 'Kendaraan.dart';
3 import 'Truk.dart';
4
5 void main() {
6   Kendaraan kendaraan1 = new Kendaraan(2000, "Suzuki");
7   kendaraan1.maju();
8
9   stdout.write("\n");
10
11   Truk truk1 = new Truk(9800, "Hino", "Container");
12   truk1.maju();
13 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\tugas Praktikum\PRAKTIKUM4> dart .\Inheritance\KendaraanTe
st.dart
Kendaraan merk Suzuki dengan cc 2000 maju!
Kendaraan merk Hino dengan cc 9800 maju!
dengan gandengan Container
```

### 3. 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*.







```
PRAKTIKUM4 - person.dart
1 import 'dart:ffi';
2 import 'dart:io';
3
4 class Person {
5   String? _nama;
6
7   Person(this._nama);
8
9   set nama(String nama) => _nama;
10  String get nama => _nama;
11
12  void showInfo() {
13    stdout.write(nama);
14  }
15 }

PRAKTIKUM4 - mahasiswa.dart
1 import 'dart:io';
2 import 'person.dart';
3
4 class Mahasiswa extends Person {
5   String? _nim;
6
7   Mahasiswa(String? nama, this._nim) : super(nama);
8
9   set nim(String nim) => _nim;
10  String get nim => _nim;
11
12  @override
13  void showInfo() {
14    super.showInfo();
15    stdout.write("\n" + this._nim!);
16  }
17 }
```

```
PRAKTIKUM4 - person_test.dart
1 import 'mahasiswa.dart';
2 import 'person.dart';
3
4 void main() {
5   final person = new Person("Fulan\n");
6   person.showInfo();
7
8   final mahasiswa = new Mahasiswa("Fulanah", "09030015");
9   mahasiswa.showInfo();
10 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\polymorphism\person_test.dart
Fulan
Fulanah
09030015
```

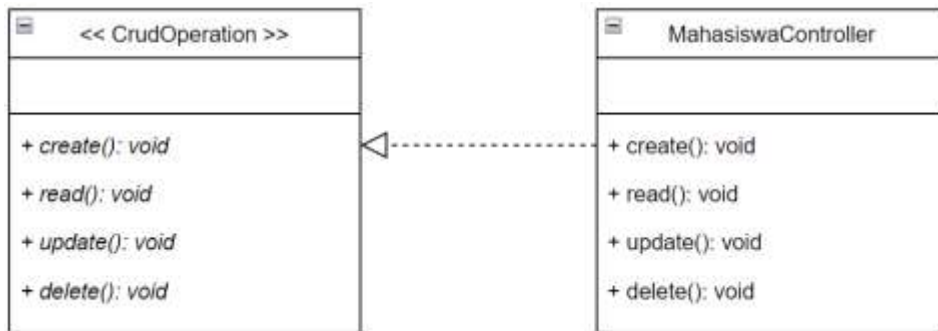
#### 4. Abstraction

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



```
1 class CrudOperation {
2     void create() {}
3     void read() {}
4     void update() {}
5     void delete() {}
6 }
```

```
1 import 'crud_operation.dart';
2
3 class MahasiswaController implements CrudOperation {
4     @override
5     void create() {
6         print("Tambah data mahasiswa");
7     }
8     @override
9     void read() {
10        print("Baca data mahasiswa");
11    }
12    @override
13    void update() {
14        print("Ubah data mahasiswa");
15    }
16    @override
17    void delete() {
18        print("Hapus data mahasiswa");
19    }
20 }
21 }
```

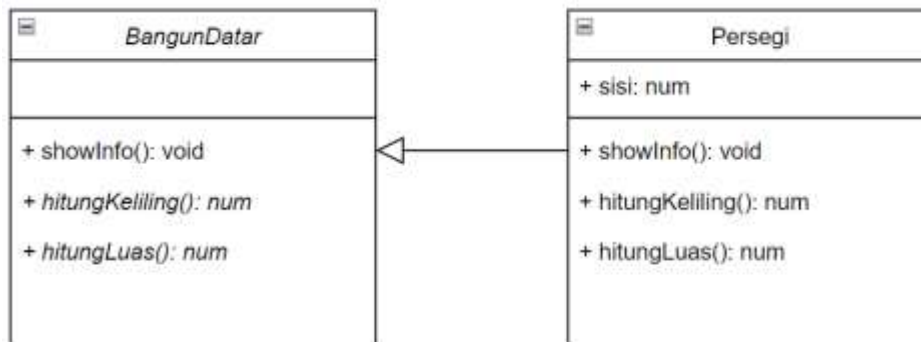
```
1 import 'mahasiswa_controller.dart';
2
3 void main() {
4     final mhs = new MahasiswaController();
5     mhs.create();
6     mhs.read();
7     mhs.update();
8     mhs.delete();
9 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\abstraction\interface_test.dart
Tambah data mahasiswa
Baca data mahasiswa
Ubah data mahasiswa
Hapus data mahasiswa
```



- **Abstract**



```
PRAKTIKUM4 - bangun_datar.dart
1 abstract class BangunDatar {
2
3   void showInfo() {
4     print("Bangun datar");
5   }
6
7   num hitungKeliling();
8
9   num hitungLuas();
10 }

PRAKTIKUM4 - persegi.dart
1 import 'bangun_datar.dart';
2
3 class Persegi extends BangunDatar {
4
5   num? sisi;
6   void showInfo() {
7     print("Persegi");
8   }
9
10  @override
11  num hitungKeliling() {
12    return this.sisi! * 4;
13  }
14
15  @override
16  num hitungLuas() {
17    return this.sisi! * this.sisi;
18  }
19
20 }
```

```
PRAKTIKUM4 - abstrac_test.dart
1 import 'persegi.dart';
2
3 void main() {
4   var persegi = new Persegi();
5   persegi.sisi = 2;
6   print(persegi.hitungKeliling());
7   print(persegi.hitungLuas());
8
9 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\abstract\abstrac_test.d
art
8
4
```

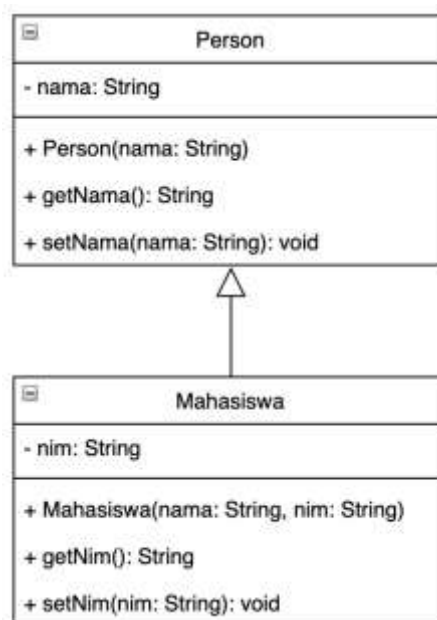


## Relasi Class

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`.



<pre>7 // ignore_for_file: file_names 8 9 class Person { 10   String _nama; 11 12   Person(this._nama); 13 14   String getNama() { 15     return _nama; 16   } 17 18   void setNama(String nama) { 19     _nama = nama; 20   } 21 }</pre>	<pre>1 // ignore_for_file: file_names 2 3 import 'person.dart'; 4 5 class Mahasiswa extends Person { 6   String _nim; 7 8   Mahasiswa(String nama, this._nim) : super(nama); 9 10  String getNim() { 11    return _nim; 12  } 13 14  void setNim(String nim) { 15    _nim = nim; 16  } 17 }</pre>
---	---



```
PRAKTIKUM4 - generalization_test.dart

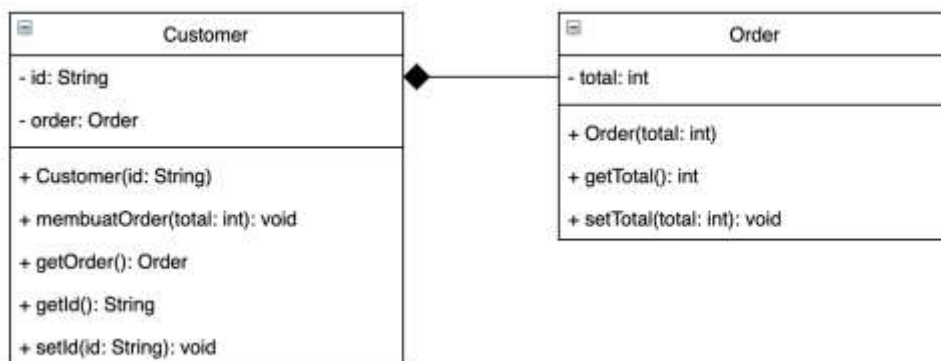
1 // ignore_for_file: file_names
2
3 import 'mahasiswa.dart';
4
5 main() {
6   Mahasiswa mhs = Mahasiswa("Fulan", "123");
7   print("Nim = " + mhs.getNim());
8   print("Nama = " + mhs.getNama());
9 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\generalization\generalization_test.dart
Nim = 123
Nama = Fulan
```

## 2. 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.





```
PRAKTIKUM4 - order.dart

1 // ignore_for_file: file_names
2
3 class Order {
4   int _total;
5
6   Order(this._total);
7
8   int getTotal() {
9     return _total;
10  }
11
12  void setTotal(int total) {
13    _total = total;
14  }
15 }
```

```
PRAKTIKUM4 - customer.dart

1 // ignore_for_file: file_names
2
3 import 'order.dart';
4
5 class Customer {
6   String? _id;
7   Order? _order;
8
9   Customer(this._id);
10
11  void membuatOrder(int total) {
12    _order = Order(total);
13  }
14
15  Order getOrder() {
16    return _order!;
17  }
18
19  String getId() {
20    return _id!;
21  }
22
23  void setId(String id) {
24    _id = id;
25  }
26 }
```

```
PRAKTIKUM4 - composition_test.dart

1 // ignore_for_file: file_names
2
3 import 'customer.dart';
4
5 main() {
6   Customer c1 = Customer("001");
7   c1.membuatOrder(3500);
8   print("Id customer = " + c1.getId());
9   print("Total order = " + c1.getOrder().getTotal().toString());
10  print("\n");
11  Customer c2 = Customer("002");
12  c2.membuatOrder(150);
13  print("Id customer = " + c2.getId());
14  print("Total order = " + c2.getOrder().getTotal().toString());
15 }
```

### Hasil Program

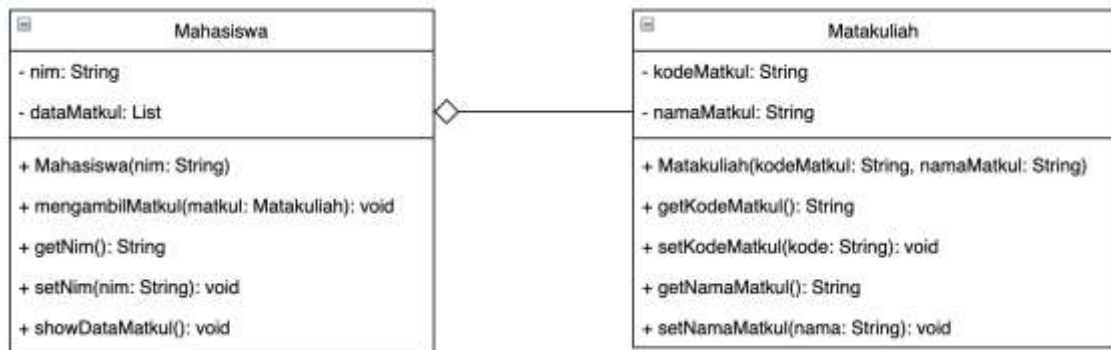
```
PS D:\04, Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\composition\composition_test.dart
Id customer = 001
Total order = 3500

Id customer = 002
Total order = 150
```



### 3. Aggregation

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



```
PRAKTIKUM4 - matakuliah.dart
1 // ignore_for_file: file_names, unnecessary_getters_setters
2
3 class Matakuliah {
4   String _kodeMatkul;
5   String _namaMatkul;
6
7   Matakuliah(this._kodeMatkul, this._namaMatkul);
8
9   String get kodeMatkul => _kodeMatkul;
10
11   set kodeMatkul(String value) {
12     _kodeMatkul = value;
13   }
14
15   String get namaMatkul => _namaMatkul;
16
17   set namaMatkul(String value) {
18     _namaMatkul = value;
19   }
20 }

PRAKTIKUM4 - mahasiswa.dart
1 // ignore_for_file: file_names, prefer_final_fields, deprecated_member_use,
2 // prefer_collection_literals, unnecessary_getters_setters
3
4 import 'matakuliah.dart';
5
6 class Mahasiswa {
7   String _nim;
8   List<Matakuliah> _dataMatkul = [];
9
10   Mahasiswa(this._nim);
11
12   void mengambilMatkul(Matakuliah matkul) {
13     _dataMatkul.add(matkul);
14   }
15
16   void showDataMatkul() {
17     print("NMN = " + _nim);
18     print("data matakuliah: ");
19
20     for (var element in _dataMatkul) {
21       print(element.kodeMatkul);
22       print(element.namaMatkul);
23     }
24   }
25 }
```

```
PRAKTIKUM4 - aggregation_test.dart
1 // ignore_for_file: file_names, unused_local_variable
2
3 import 'mahasiswa.dart';
4 import 'matakuliah.dart';
5
6 main() {
7   Matakuliah mk1 = Matakuliah("PBO", "Pemrograman Berorientasi Objek");
8   Matakuliah mk2 = Matakuliah("PM", "Pemrograman Mobile");
9   Matakuliah mk3 = Matakuliah("PWI", "Pemrograman Web 1");
10  Matakuliah mk4 = Matakuliah("BD", "Basis Data");
11
12  Mahasiswa mhs1 = Mahasiswa("210934");
13  mhs1.mengambilMatkul(mk1);
14  mhs1.mengambilMatkul(mk4);
15
16  mhs1.showDataMatkul();
17 }
```



## Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\aggregation\aggregation_test.dart
NIM = 210934
data matakuliah:
PBO
Pemrograman Berorientasi Objek
BD
Basis Data
```

### 4. Association

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



Mahasiswa.dart

```
PRAKTIKUM4 - mahasiswa.dart

1 // ignore_for-file: file_names, unused_field, prefer_final_fields,
2 // unnecessary_getters_setters
3
4 import 'dosen.dart';
5
6 class Mahasiswa {
7   String _nim;
8   Dosen _wali;
9
10  Mahasiswa(this._nim, this._wali);
11
12  void setNim(String nim) {
13    _nim = nim;
14  }
15
16  String getNim() {
17    return _nim;
18  }
19
20  void setWali(Dosen wali) {
21    _wali = wali;
22  }
23
24  Dosen getWali() {
25    return _wali;
26  }
27 }
```





## Dosen.dart

```
PRAKTIKUM4 - dosen.dart

1 // ignore_for_file: file_names, unused_field, prefer_final_fields,
2 // unnecessary_getters_setters
3
4 import 'mahasiswa.dart';
5
6 class Dosen {
7   String _nIdn;
8   List<Mahasiswa> _listMahasiswa = [];
9
10  Dosen(this._nIdn);
11
12  String getNIdn() {
13    return _nIdn;
14  }
15
16  void setNIdn(String nIdn) {
17    _nIdn = nIdn;
18  }
19
20  void addMahasiswaWali(Mahasiswa mhs) {
21    _listMahasiswa.add(mhs);
22  }
23
24  void showMahasiswaWali() {
25    print("NIDN: " + _nIdn);
26    print("Daftar mahasiswa wali");
27    for (var data in _listMahasiswa) {
28      print("NIM: " + _nIdn);
29    }
30    print("\n");
31  }
32
33 }
```

## Main.dart

```
PRAKTIKUM4 - main.dart

1 // ignore_for_file: file_names
2
3 import 'dosen.dart';
4 import 'mahasiswa.dart';
5
6 main() {
7   Dosen dsn1 = Dosen("9090");
8   Dosen dsn2 = Dosen("9191");
9
10  Mahasiswa mhs1 = Mahasiswa("1212", dsn1);
11  Mahasiswa mhs2 = Mahasiswa("2323", dsn1);
12
13  Mahasiswa mhs3 = Mahasiswa("3434", dsn2);
14  Mahasiswa mhs4 = Mahasiswa("4545", dsn2);
15
16  dsn1.addMahasiswaWali(mhs1);
17  dsn1.addMahasiswaWali(mhs2);
18
19  dsn2.addMahasiswaWali(mhs3);
20  dsn2.addMahasiswaWali(mhs4);
21
22  dsn1.showMahasiswaWali();
23  dsn2.showMahasiswaWali();
24
25 }
```



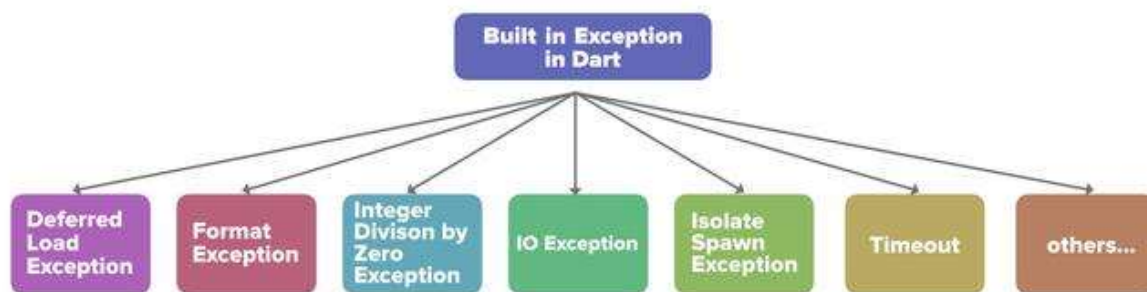
## Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\association\main.dart
NIDN: 9090
Daftar mahasiswa wali
NIM: 9090
NIM: 9090

NIDN: 9191
Daftar mahasiswa wali
NIM: 9191
NIM: 9191
```

## 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*.



```
PRAKTIKUM4 - pembagian.dart

1 class Pembagian {
2   num hitungPembagian(num nilaiA, num nilaiB) {
3     return nilaiA / nilaiB;
4   }
5 }
```

```
PRAKTIKUM4 - main.dart

1 import 'pembagian.dart';
2
3 main() {
4   String x = "delapan";
5   double y = 0;
6   num hasil;
7
8   final pembagian = new Pembagian();
9   hasil = pembagian.hitungPembagian(double.parse(x), y);
10  print(hasil);
11
12  print("hello world!");
13 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\exception_handling\main
.dart
Unhandled exception:
FormatException: Invalid double
delapan
#0      double.parse (dart:core-patch/double_patch.dart:111:28)
#1      main (file:///D:/04.%20Semester%204%20D3TI2C/PEMROGRAMAN%20PERANGKAT%20BERGERAK/Tugas%20Praktikum/PRAKTIKUM4/
exception_handling/main.dart:9:44)
#2      _delayEntrypointInvocation.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:297:19)
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:192:12)
```

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



- try..on

```
PRAKTIKUM4 - main.dart

1  import 'pembagian.dart';
2
3  main() {
4    String x = "delapan";
5    double y = 0;
6    num hasil;
7
8    final pembagian = new Pembagian();
9
10   try {
11     hasil = pembagian.hitungPembagian(double.parse(x), y);
12     print(hasil);
13   } on FormatException {
14     print('Kesalahan format exception');
15   }
16
17   print("heloo world!");
18 }
```

#### Hasil Program

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\exception_handling\try_
on\main.dart
Kesalahan format exception
heloo world!
```

- try..catch

```
PRAKTIKUM4 - main.dart

1  import 'pembagian.dart';
2
3  main() {
4    String x = "delapan";
5    double y = 0;
6    num hasil;
7
8    final pembagian = new Pembagian();
9
10   try {
11     hasil = pembagian.hitungPembagian(double.parse(x), y);
12     print(hasil);
13   } catch(e) {
14     print(e);
15   }
16
17   print("hello world!");
18 }
```

#### Hasil Program

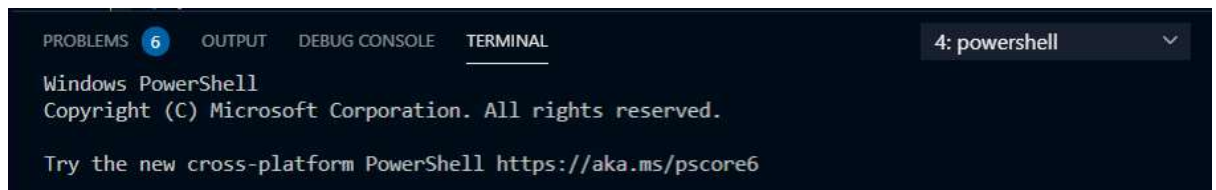
```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\exception_handling\try_
catch\main.dart
FormatException: Invalid double
delapan
hello world!
```



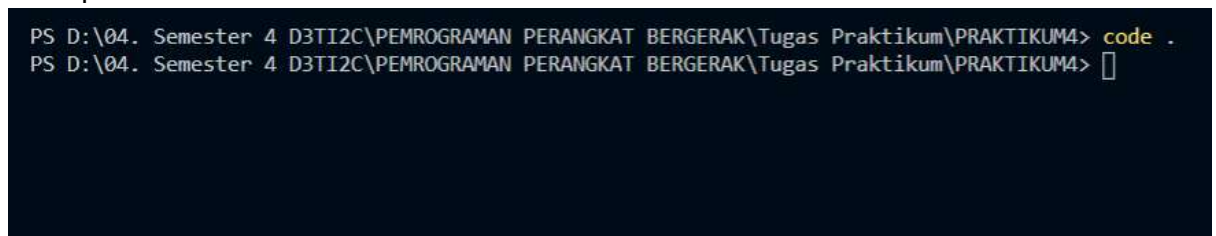
### C. LANGKAH DAN HASIL PELAKSANAAN PRAKTIKUM

Langkah-langkah praktikum

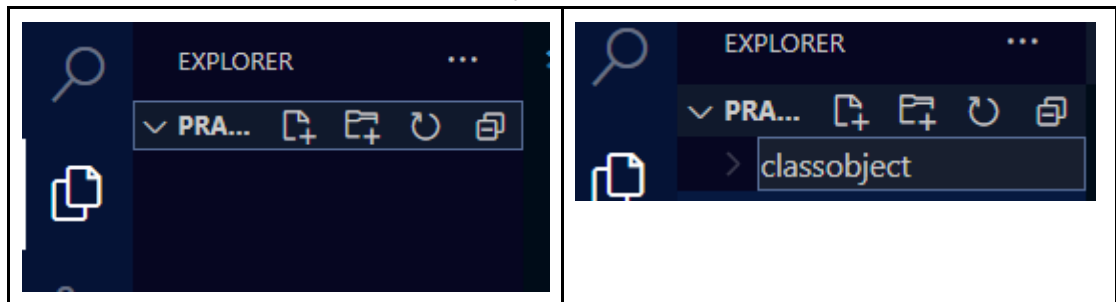
1. Buatlah folder bernama “praktikum4” pada *File Explorer*
2. Buka *command line* pada folder “praktikum4” dengan cara tekan *Shift* + klik kanan pada area di dalam folder, kemudian pilih *Open PowerShell window here* atau *Git Bash Here*



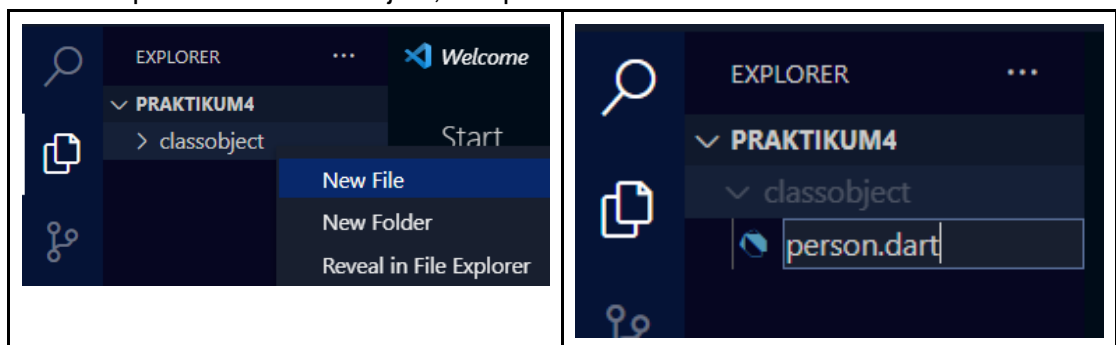
3. Ketik perintah “code .” lalu tekan Enter



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



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





6. Tulis dan simpan kode program 4.1 berikut  
Kode program 4.1. `Person.dart`

```
PRAKTIKUM4 - person.dart

1 class Person {
2   String? _nama;
3
4   // Constructor
5   Person(this._nama);
6
7   // setter dan getter
8   set nama(String nama) => _nama = nama;
9   String get nama => _nama!;
10
11   String showInfo() {
12     return "halo, saya" + _nama! + "!";
13   }
14 }
```

7. Buatlah file baru di dalam folder “classobject” bernama “PersonTest.dart”  
8. Tulis dan simpan kode program 4.2 berikut  
Kode program 4.2 `PersonTest.dart`

```
PRAKTIKUM4 - person_test.dart

1 import 'person.dart';
2
3 void main() {
4   var person1 = Person("Fulan");
5   print(person1.nama);
6   person1.nama = "Joko";
7   print(person1.showInfo());
8 }
```

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

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\classobject\person_test
.dart
Fulan
halo, saya Joko!
```

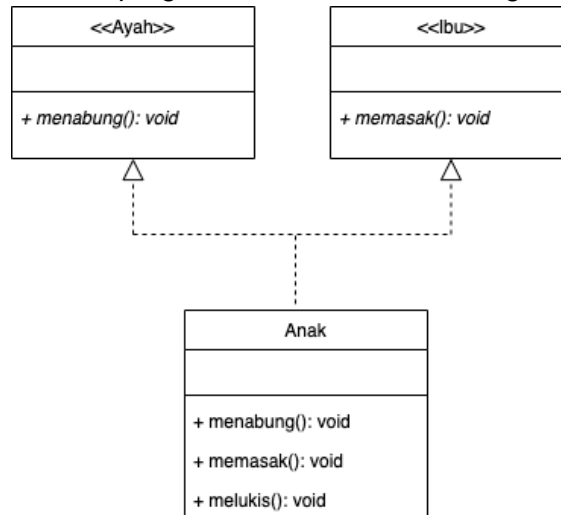
10. Selesai



## D. LATIHAN

### Latihan *Multiple Inheritance*

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



```
PRAKTIKUM4 - ayah.dart
1 class Ayah {
2   void menabung() {}
3 }
4

PRAKTIKUM4 - ibu.dart
1 class Ibu {
2   void memasak() {}
3 }

PRAKTIKUM4 - main.dart
1 import 'anak.dart';
2
3 void main() {
4   final ank = new Anak();
5   ank.menabung();
6   ank.masak();
7   ank.melukis();
8 }

PRAKTIKUM4 - anak.dart
1 import 'ayah.dart';
2 import 'ibu.dart';
3
4 class Anak implements Ayah, Ibu {
5   @override
6   void memasak() {
7     print("Ibu memasak ayam goreng");
8   }
9
10  @override
11  void menabung() {
12    print("Adik gemar menabung");
13  }
14
15  void melukis() {
16    print("Ayah suka melukis");
17  }
18 }
```

### Hasil Program

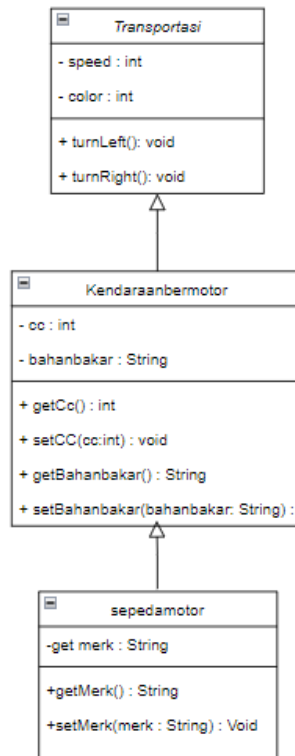
```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\multiple_inheritance\main.dart
Adik gemar menabung
Ibu memasak ayam goreng
Ayah suka melukis
```





### Latihan *Multilevel Inheritance*

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



PRAKTIKUM4 - transportasi.dart

```
1 class Transportasi {
2   int? _speed;
3   int? _color;
4
5   void turnLeft() {}
6   void turnRight() {}
7 }
8
```

PRAKTIKUM4 - sepeda\_motor.dart

```
1 import 'kendaran_bermotor.dart';
2
3 class SepedaMotor extends KendaranBermotor {
4   String? _merk;
5
6   set merk(String merk) => _merk = merk;
7   String get merk => _merk;
8 }
9
```

PRAKTIKUM4 - kendaran\_bermotor.dart

```
1 import 'transportasi.dart';
2
3 class KendaranBermotor extends Transportasi {
4   int? _cc;
5   String? _bahanbakar;
6
7   KendaranBermotor(this._cc, this._bahanbakar);
8
9   set cc(int cc) => _cc = cc;
10  int get cc => _cc;
11
12  set bahanbakar(String bahanbakar)
13  => _bahanbakar = bahanbakar;
14  String get bahanbakar
15  => _bahanbakar;
16 }
17
```

main.dart

```
1 import 'sepeda_motor.dart';
2
3 void main() {
4   SepedaMotor spd1 = new SepedaMotor();
5   spd1.merk = "Honda";
6   spd1.cc = 120;
7   spd1.bahanBakar = ("bensin");
8
9   print(spd1.merk);
10  print(spd1.cc);
11  print(spd1.bahanBakar);
12
13 }
```

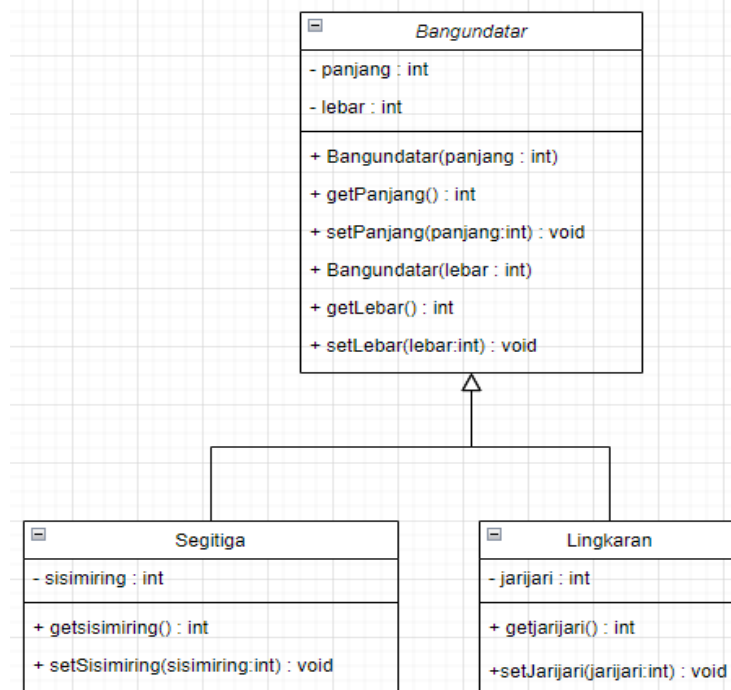


## Hasil Program

```
PS D:\04, Semester 4 D3TI2C\PENROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\multilevel_inheritance\main.dart  
Honda  
120  
bensin
```

### Latihan *Hierarchical Inheritance*

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



```
PRAKTIKUM4 - bangundatar.dart

1 class Bangundatar {
2   int? _panjang;
3   int? _lebar;
4
5   Bangundatar.panjang(this, _panjang);
6   int getPanjang() {
7     return _panjang;
8   }
9   void setPanjang(int panjang) {
10    _panjang = panjang;
11  }
12  Bangundatar.lebar(this, _lebar);
13  int getLebar() {
14    return _lebar;
15  }
16  void setLebar(int lebar) {
17    _lebar = lebar;
18  }
19 }
20 }
```

```
PRAKTIKUM4 - lingkaran.dart

1 import 'bangundatar.dart';
2
3 class Lingkaran extends Bangundatar {
4   int? _jarijari;
5
6   Lingkaran.lebar(int? lebar) = super.lebar(lebar);
7
8   int getjarijari(){
9     return _jarijari;
10  }
11  void setJarijari(int jarijari) {
12    _jarijari = jarijari;
13  }
14 }
```



```
PRAKTIKUM4 - segitiga.dart
1 import 'bangundatar.dart';
2
3 class Segitiga extends BangunDatar {
4   int _sisimiring;
5
6   Segitiga.lebar(int lebar) : super.lebar(lebar);
7
8   int getSisimiring(){
9     return _sisimiring;
10  }
11  void setSisimiring(int sisimiring) {
12    _sisimiring = sisimiring;
13  }
14 }

PRAKTIKUM4 - main.dart
1 import 'segitiga.dart';
2 import 'lingkaran.dart';
3
4 void main() {
5   Segitiga sgt1 = new Segitiga.lebar(4);
6   sgt1.setSisimiring(3);
7
8   print(sgt1.getLebar());
9   print(sgt1.getSisimiring());
10
11   Lingkaran lkr1 = new Lingkaran.lebar(6);
12   lkr1.setJarijari(14);
13
14   print(lkr1.getLebar());
15   print(lkr1.getJarijari());
16
17 }
```

Hasil Program

```
PS D:\04. Semester 4 DITIZC\PEMROGRAMAN PERANGKAT BERGERAK\tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\hierarchical_inheritance\main.dart
4
3
6
14
```

### Latihan Hybrid Inheritance

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





```
PRAKTIKUM4 - person.dart
1 class Person {
2   String? _nama;
3
4   Person(this._nama);
5
6   set nama(String nama) => _nama = nama;
7   String get nama => _nama;
8 }
9

PRAKTIKUM4 - mahasiswa.dart
1 import 'person.dart';
2
3 class Mahasiswa extends Person {
4   String? _nim;
5
6   Mahasiswa(String? nama, this._nim) : super(nama);
7
8   set nim(String nim) => _nim = nim;
9   String get nim => _nim;
10 }
11

PRAKTIKUM4 - dosen.dart
1 import 'person.dart';
2
3 class Dosen extends Person {
4   String? _nidn;
5
6   Dosen(String? nama, this._nidn) : super(nama);
7
8   set nidn(String nidn) => _nidn = nidn;
9   String get nidn => _nidn;
10 }

PRAKTIKUM4 - mahasiswa_dalam_negeri.dart
1 import 'mahasiswa.dart';
2
3 class MahasiswaDalamNegeri {
4   String? _nik;
5
6   MahasiswaDalamNegeri(this._nik);
7
8   set nik(String nik) => _nik = nik;
9   String get nik => _nik;
10 }

PRAKTIKUM4 - mahasiswa_luar_negeri.dart
1 import 'mahasiswa.dart';
2
3 class MahasiswaLuarNegeri {
4   String? _passport;
5
6   MahasiswaLuarNegeri(this._passport);
7
8   set passport(String passport) => _passport = passport;
9   String get passport => _passport;
10 }
```

Main.dart

```
PRAKTIKUM4 - mahasiswa_dalam_negeri_test.dart
1 import 'mahasiswa_dalam_negeri.dart';
2 import 'mahasiswa_luar_negeri.dart';
3 import 'dosen.dart';
4
5 void main() {
6   var mhs1 = new MahasiswaDalamNegeri("Ica", "0908752809");
7   mhs1.nik = "0908752809";
8   print("Mahasiswa POLIBERA bernama ${mhs1.nama} memiliki NIM ${mhs1.nim} dan NIK ${mhs1.nik}");
9
10  var mhs2 = new MahasiswaLuarNegeri("Ica", "2003073");
11  mhs2.passport = "090999999";
12  print("Mahasiswa POLIBERA bernama ${mhs2.nama} melanjutkan ke Universitas Oxford memiliki NIM ${mhs2.nim} dan mempunyai passport dengan kode ${mhs2.passport}");
13
14  var doc = new Dosen("Ica", "0908752809");
15  print("Dosen Ica memiliki NIDN ${doc.nidn}");
16 }
```

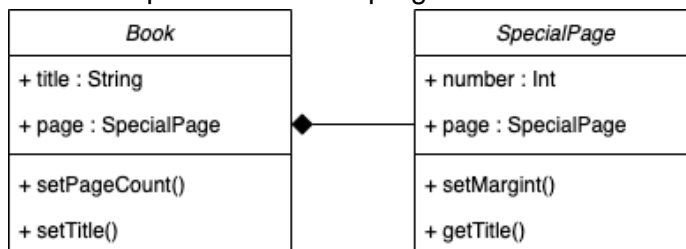
## Hasil Program

```
P5 D:\04_Semester 4 D3 TI\PEMROGRAMAN PERANGKAT BERGERAK\tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\hybrid_inheritance\mahasiswa_dalam_negeri_test.dart
Mahasiswa POLIBERA bernama Ica memiliki NIM 2003073 dan NIK 0908752809
Ica merupakan mahasiswa lulusan POLIBERA lalu melanjutkan ke Universitas Oxford memiliki NIM 2003073 dan mempunyai passport dengan kode 090999999
Dua dosen Ica memiliki NIDN 0908752809
```



### Latihan *Composition*

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

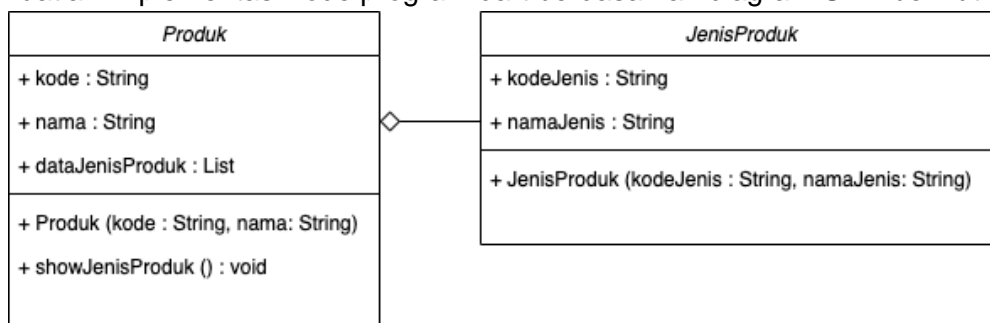


<pre>PRAKTIKUM4 - book.dart  1 import 'special_page.dart'; 2 3 abstract class Book { 4   String? title; 5   SpecialPage? page; 6 7   setPageCount() { 8 9   } 10 11  setTitle() { 12 13  } 14 }</pre>	<pre>PRAKTIKUM4 - special_page.dart  1 abstract class SpecialPage { 2   int? number; 3   SpecialPage? page; 4 5   setMargint() { 6 7   } 8   getTitle() { 9 10  } 11 }</pre> <p>main.dart</p>
---	---

### Hasil Program

### Latihan *Aggregation*

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





```
PRAKTIKUM4 - produk.dart
1 import 'jenis_produk.dart';
2
3 class Produk {
4   String? kode;
5   String? nama;
6
7   List<Jenisproduk> dataJenisProduk = [];
8
9   // Produk(this.kode, this.nama);
10
11   // void namaP() {
12
13   // }
14
15   void showJenis() {
16     print("Kode Produk : "+ kode);
17     print("Data Jenis Produk :");
18
19     for (var element in dataJenisProduk) {
20       print(element.kodeJenis);
21       print(element.namaJenis);
22     }
23   }
24
25   void memilikiJp (Jenisproduk jp) {
26     dataJenisProduk.add(jp);
27   }
28 }
```

```
PRAKTIKUM4 - jenis_produk.dart
1 class Jenisproduk {
2   String? kodeJenis;
3   String? namaJenis;
4
5   Jenisproduk(this.kodeJenis, this.namaJenis);
6
7 }
```

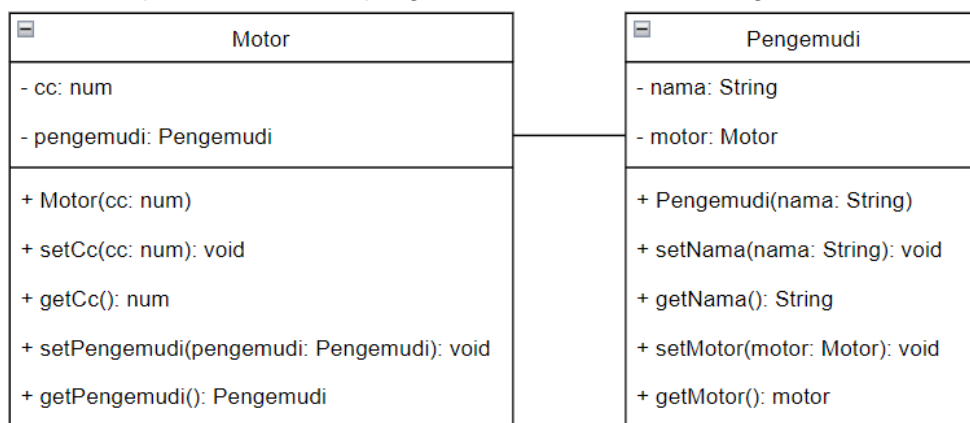
```
main.dart
1 import 'jenis_produk.dart';
2 import 'produk.dart';
3
4 void main() {
5   Jenisproduk jp1 = new Jenisproduk("JP01", "Malls");
6   Jenisproduk jp2 = new Jenisproduk("JP02", "Doritos");
7   Jenisproduk jp3 = new Jenisproduk("JP03", "Cyrus");
8
9   Produk p1 = new Produk();
10  p1.kode = "P01";
11
12  p1.milikiJp(jp1);
13  p1.milikiJp(jp3);
14
15  p1.showJenis();
16 }
```

### Hasil Program

```
PS D:\04. Semester 4 D3TIK\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\aggregation\mai
n.dart
Kode Produk : P01
Data Jenis Produk :
JP01
Malls
JP03
Cyrus
```

### Latihan Association

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







```
PRAKTIKUM4 - motor.dart

1 import 'pengemudi.dart';
2
3 class Motor {
4   num? _cc;
5   Pengemudi? _pengemudi;
6
7   Motor(this._cc);
8
9   void setCc(num cc) {
10     _cc = cc;
11   }
12   num getCc() {
13     return _cc!;
14   }
15   void setPengemudi(Pengemudi pengemudi) {
16     _pengemudi = pengemudi;
17   }
18   Pengemudi getPengemudi() {
19     return _pengemudi;
20   }
21 }
```

```
PRAKTIKUM4 - pengemudi.dart

1 import 'motor.dart';
2
3 class Pengemudi {
4   String? _nama;
5   Motor? _motor;
6
7   Pengemudi(this._nama);
8
9   void setName(String nama) {
10     _nama = nama;
11   }
12   String getNama() {
13     return _nama!;
14   }
15   void setMotor(Motor motor) {
16     _motor = motor;
17   }
18   Motor getMotor() {
19     return _motor!;
20   }
21 }
```

main.dart

```
PRAKTIKUM4 - main.dart

1 import 'motor.dart';
2 import 'pengemudi.dart';
3
4 main() {
5   Pengemudi p1 = new Pengemudi("Tarjo");
6   Motor m1 = new Motor(1800);
7
8   print("Pengemudi: " + p1.getNama()
9     + " mengendarai motor dengan CC "
10     + m1.getCc().toString());
11 }
12 }
```

### Hasil Program

```
PS D:\B4. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\association\mai
n.dart
Pengemudi Tarjo mengendarai motor dengan CC 1800
PS D:\B4. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> 
```





### 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

```
PRAKTIKUM4 - contoh_exception.dart

1  import 'dart:async';
2  import 'dart:io';
3
4  void main() {
5    try {
6      print("-- Contoh throw DeferredLoadException --");
7      throw DeferredLoadException('Ini adalah deferred load exception');
8    } on IOException catch (e) {
9      print("Muncul hanya ketika ada IOException, $e");
10   } catch (e) {
11     print(e);
12   } finally {
13     print("Akhir dari blok try deferred load exception\n");
14   }
15
16   try {
17     print("-- Contoh throw HttpException --");
18     throw HttpException('ini adalah http exception');
19   } on HttpException catch (e) {
20     print(e);
21   } catch (e) {
22     print("Pesan ini muncul apabila ada exception selain HttpException");
23   } finally {
24     print("Akhir dari blok try http exception");
25   }
26 }
```

Hasil kode program 4.3.

```
PS D:\04. Semester 4 D3TI2C\PEMROGRAMAN PERANGKAT BERGERAK\Tugas Praktikum\PRAKTIKUM4> dart .\LATIHAN\exception_handling\contoh_exception.dart
-- Contoh throw DeferredLoadException --
DeferredLoadException: 'Ini adalah deferred load exception'
Akhir dari blok try deferred load exception

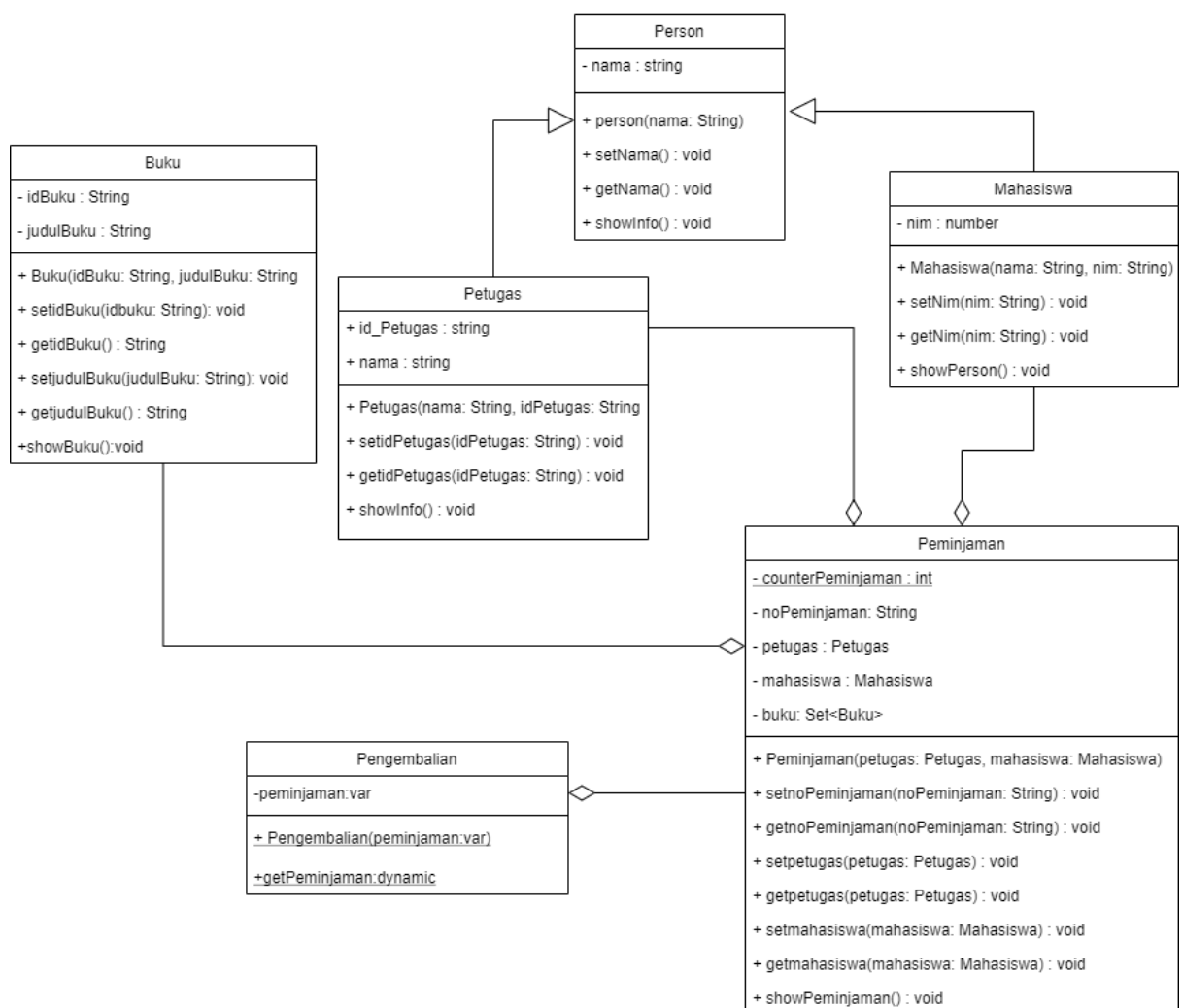
-- Contoh throw HttpException --
HttpException: ini adalah http exception
Akhir dari blok try http exception
```



## E. TUGAS INDIVIDU

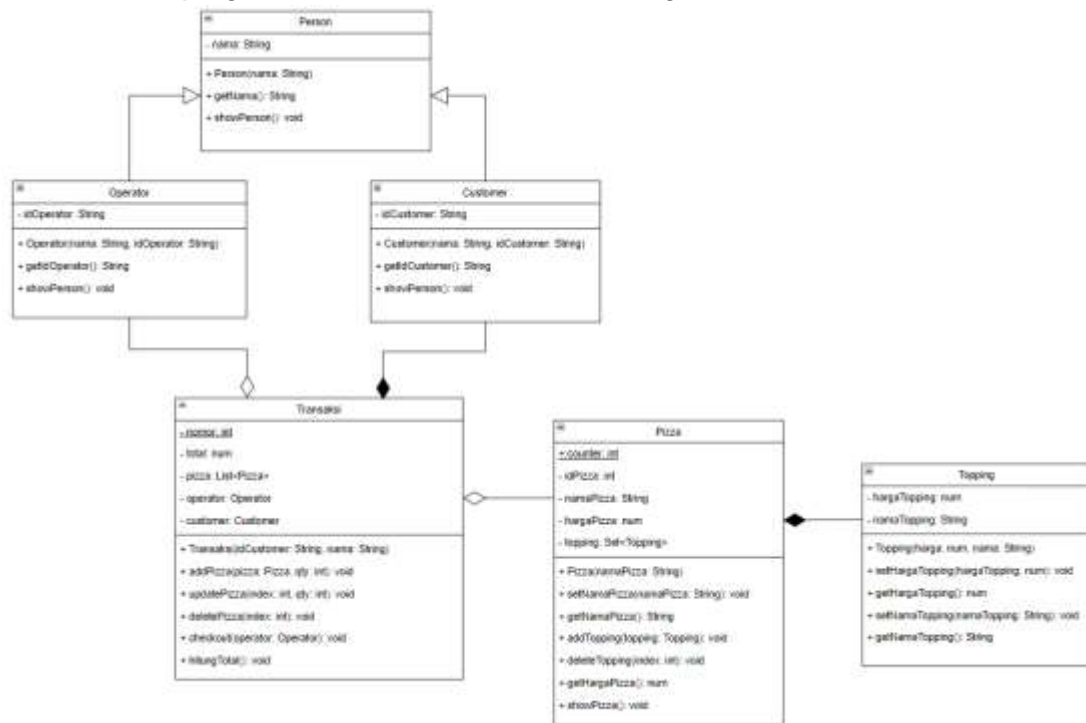
1. Buatlah *class diagram* beserta relasinya berdasarkan studi kasus Perpustakaan yang terdiri dari class Buku, Mahasiswa, Petugas, Peminjaman, dan Pengembalian

Diagram Tugas Praktikum 4 PPB Nomor 1





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



```
PRAKTIKUM4 - person.dart

1 class Person {
2   String? _nama;
3
4   Person(this._nama);
5   set nama(String? nama) => _nama;
6   String getNama() {
7     return _nama!;
8   }
9   void showPerson() {
10    print(_nama!);
11  }
12 }
```

```
PRAKTIKUM4 - operator.dart

1 import 'person.dart';
2
3 class Operator extends Person{
4   String? _idOperator;
5
6   Operator(String nama, this._idOperator): super(nama);
7   String getIdOperator() {
8     return _idOperator!;
9   }
10  void showPerson() {
11    print(_idOperator!);
12  }
13 }
14
15 }
```

```
PRAKTIKUM4 - customer.dart

1 import 'person.dart';
2 class Customer extends Person{
3   String? _idCustomer;
4
5   Customer(String nama, this._idCustomer): super(nama);
6   String getIdCustomer() {
7     return _idCustomer!;
8   }
9   void showPerson() {
10    print(_idCustomer!);
11  }
12 }
13
14 }
```

```
PRAKTIKUM4 - topping.dart

1 class Topping {
2   num? _hargaTopping;
3   String? _namaTopping;
4
5   Topping(this._hargaTopping, this._namaTopping);
6
7   void set hargaTopping (num hargaTopping)
8     => _hargaTopping;
9   num get hargaTopping => _hargaTopping!;
10
11   void set namaTopping (String namaTopping)
12     => _namaTopping;
13   String get namaTopping => _namaTopping!;
14 }
```



```
tugas2 - transaksi.dart
1 import 'operator.dart';
2 import 'customer.dart';
3 import 'pizza.dart';
4
5 class Transaksi {
6   static int _jumlah;
7   int _total;
8   Operator? _operator;
9   Customer? _customer;
10
11   static Pizza? _pizza = null;
12
13   Transaksi(String? idCustomer, String? nama) {
14     _customer = new Customer("Tasya", "1");
15   }
16
17   void addPizza(Pizza pizza, int qty) {
18     _pizza.add(pizza);
19     _total = qty;
20   }
21
22   void updatePizza(int index) {
23     print(_pizza[index]);
24   }
25
26   void checkout(Operator operator) {
27     _operator = operator;
28   }
29
30   void hitungTotal() {
31     print("nama pemesan ${_customer?.nama}");
32     print("-----");
33     for (var pizza in _pizza) {
34       pizza.showPizza();
35     }
36     print("total pesanan & total");
37     print("harga pizza & pizza.hargaPizza");
38     print("total harga pizza & total * pizza.getHargaPizza()");
39   }
40
41   print("operator ${_operator?.nama}");
42   print("");
43 }
```

```
tugas2 - pizza.dart
1 import 'topping.dart';
2
3 class Pizza {
4   static int _counter = 1;
5   int? _idPizza;
6   String? _namaPizza;
7   num? _hargaPizza;
8   Set<Topping> _topping = new Set<Topping>();
9
10   Pizza(this._namaPizza, this._hargaPizza) {
11     _counter++;
12   }
13
14   void setNamaPizza(String? namaPizza) => _namaPizza;
15   String get namaPizza => _namaPizza;
16
17   void addTopping(Topping topping) {
18     _topping.add(topping);
19   }
20
21   void deleteTopping(int index) {
22     _topping.remove(index);
23   }
24
25   num get hargaPizza => _hargaPizza;
26   num getHargaPizza() {
27     for (var topping in _topping) {
28       _hargaPizza = _hargaPizza + topping.hargaTopping;
29     }
30   }
31
32   return _hargaPizza;
33 }
34
35 void showPizza() {
36   print("nama pizza & harga pizza & topping : ");
37   for (var topping in _topping) {
38     print("${_counter++} : $topping.namaTopping, $topping.hargaTopping");
39   }
40 }
41 }
```

## Main.dart

```
tugas2 - main.dart
1 import 'customer.dart';
2 import 'operator.dart';
3 import 'pizza.dart';
4 import 'topping.dart';
5 import 'transaksi.dart';
6
7 void main() {
8   var operator1 = Operator("Iki", "001");
9   var operator2 = Operator("Irma", "002");
10
11   var customer1 = Customer("Tasya", "1");
12
13   var topping1 = Topping(3000, "sosis");
14   var topping2 = Topping(8000, "Kornet");
15
16   var pizza1 = Pizza("italian", 8000);
17   var pizza2 = Pizza("turki", 15000);
18
19   pizza1.addTopping(topping1);
20   pizza1.addTopping(topping2);
21
22   var transaksi1 = Transaksi("Tasya", "1");
23
24   transaksi1.addPizza(pizza1, 1);
25   transaksi1.checkout(operator1);
26   transaksi1.hitungTotal();
27
28   pizza2.addTopping(topping2);
29
30   var transaksi2 = Transaksi("Tasya", "1");
31
32   transaksi2.checkout(operator2);
33   transaksi2.addPizza(pizza2, 2);
34   transaksi2.hitungTotal();
35
36 }
```



### Hasil Kode Program

```
PS C:\Users\asus\Documents\tugas2> dart .\main.dart
nama pemesan Tasya
-----
nama pizza italian dengan topping :
1. sosis. 3000
2. Kornet. 8000
total pesanan 1
harga pizza 8000
total harga pizza 19000
operator Iki
```

```
nama pemesan Tasya
-----
nama pizza turki dengan topping :
3. Kornet. 8000
total pesanan 2
harga pizza 15000
total harga pizza 46000
operator Irma
```

3. 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

```
tugas2 - main.dart
1 import 'customer.dart';
2 import 'operator.dart';
3 import 'pizza.dart';
4 import 'topping.dart';
5 import 'transaksi.dart';
6
7 void main() {
8   var operator1 = Operator("Iki", "001");
9   var operator2 = Operator("Irma", "002");
10
11   var customer1 = Customer("Tasya", "1");
12
13   var topping1 = Topping(3000, "sosis");
14   var topping2 = Topping(8000, "Kornet");
15
16   var pizza1 = Pizza("Italian", 8000);
17   var pizza2 = Pizza("turki", 15000);
18
19   pizza1.addTopping(topping1);
20   pizza1.addTopping(topping2);
21
22   var transaksi1 = Transaksi("Tasya", "1");
23
24
25   pizza1.addTopping(topping1);
26
27   var transaksi2 = Transaksi("Tasya", "1");
28
29   try {
30     transaksi2.checkout(operator2);
31     transaksi2.addPizza(pizza2, 2);
32     transaksi2.hitungTotal();
33   } catch (e) {
34     print(e);
35   }
36
37   print('Exception Tugas 2');
38
39 }
```



### Hasil Kode Program nomor 2

```
PS C:\Users\asus\Documents\tugas2> dart .\main.dart
nama pemesan Tasya
-----
nama pizza turki dengan topping :
1. Kornet. 8000
total pesanan 2
harga pizza 15000
total harga pizza 46000
operator Irma

Exception Tugas 2
```

### F. 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.