Mobile Programming Advanced

FLUTTER

Jeremia Tarigan

Dart OOP

Topik

Pengenalan Object Oriented Programming

Object Oriented Programming

- Object Oriented Programming adalah sudut pandang bahasa pemrograman yang berkonsep “objek”
- Ada banyak sudut pandang bahasa pemrograman, namun OOP adalah yang sangat populer saat ini.
- Ada beberapa istilah yang perlu dimengerti dalam OOP, yaitu: Object dan Class

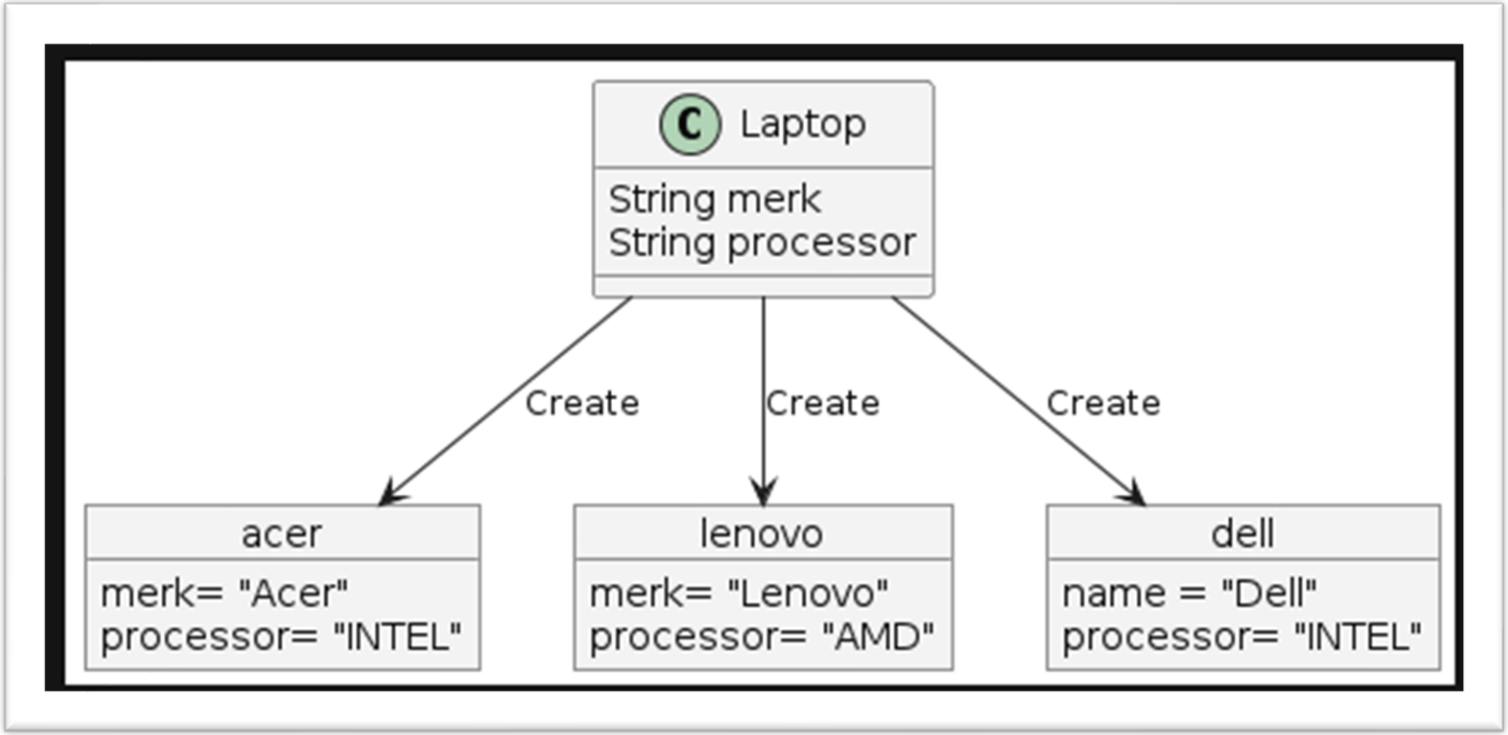
Object

- Object adalah data yang berisi field / properties / attributes dan method / function / behavior
- Semua data di Dart adalah Object

Class

- Class adalah blueprint, prototype atau cetakan untuk membuat Object
- Class berisikan deklarasi semua properties dan functions yang dimiliki oleh Object
- Setiap Object selalu dibuat dari Class
- Dan sebuah Class bisa membuat Object tanpa batas

Contoh Class dan Object



Contoh Program Class

```
class Laptop{
    String? merk;
    String? processor;

  void myLaptop(String merk,String processor){
       print('Merk Laptop saya $ merk dan menggunakan processor $processor');
    }
}
```

Contoh Program Membuat Object

```
void main(){
  var laptopObject = Laptop();
  laptopObject.merk = "Asus";
  laptopObject.processor = "Intel";

  print(laptopObject.merk);
  print(laptopObject.processor);

  laptopObject.myLaptop("Acer","AMD");
}
```

Topik

Properties

- Dart create dart-oop

Properties

- Fields / Properties / Attributes adalah data yang bisa kita sisipkan di dalam Object
- Namun sebelum kita bisa memasukkan data di fields, kita harus mendeklarasikan data apa saja yang dimiliki object tersebut di dalam deklarasi class-nya
- Membuat field sama seperti membuat variable, namun ditempatkan di block class
- Field wajib dimasukkan nilai nya, kecuali field yang nullable

Contoh FIELD

```
class Laptop{

    String? merk;

    String? processor;

}
```

Manipulasi Field

- Fields yang ada di object, bisa kita manipulasi. Tergantung final atau bukan.
- Jika final, berarti kita tidak bisa mengubah data field nya, namun jika tidak, kita bisa mengubah field nya
- Untuk memanipulasi data field, sama seperti cara pada variable
- Untuk mengakses field, kita butuh kata kunci . (titik) setelah nama object dan diikuti nama field nya

Topik

METHOD

Method

- Selain menambahkan field, kita juga bisa menambahkan method ke object
- Method adalah function yang terdapat di dalam class
- Cara dengan mendeklarasikan method tersebut di dalam block class
- Sama seperti function biasanya, kita juga bisa menambahkan return value, parameter di method yang ada di dalam block class
- Untuk mengakses method tersebut, kita bisa menggunakan tanda titik (.) dan diikuti dengan nama method nya. Sama seperti mengakses field

Topik

Constructor

Constructor

- Saat kita membuat Object, maka kita seperti memanggil sebuah method, karena kita menggunakan kurung ()
- Di dalam class, kita bisa membuat constructor, constructor adalah method yang akan dipanggil saat pertama kali Object dibuat.
- Mirip seperti di method, kita bisa memberi parameter pada constructor
- Nama constructor harus sama dengan nama class, dan tidak membutuhkan kata kunci void atau return value
- Ketika kita menambahkan Constructor pada class, maka saat membuat Object baru, kita wajib mengikuti parameter yang ada di Constructor

Contoh Constructor

```
class Laptop {

  String? merk;
  String? processor;

  Laptop(String paramMerk, String
paramMerk){
     merk = paramMerk;
     processor = paramMerk;
   }

}
```

```
void main(){

        var laptopObject = Laptop("Asus", "Intel");
         print(laptopObject.merk);
         print(laptopObject.processor);

}
```

Topik

This Keyword

This Keyword

- Saat kita membuat kode di dalam block constructor atau method di dalam class, kita bisa menggunakan kata kunci this untuk mengakses object saat ini
- Misal kadang kita butuh mengakses sebuah field yang namanya sama dengan parameter method, hal ini tidak bisa dilakukan jika langsung menyebut nama field, kita bisa mengakses nama field tersebut dengan kata kunci this
- This juga tidak hanya digunakan untuk mengakses field milik object saat ini, namun juga bisa digunakan untuk mengakses method

Contoh This Keyword

```
class Laptop {

  String? merk;
  String? processor;

  Laptop(merk, processor){
    this.merk = merk;
    this.processor = processor;
  }

}
```

```
void main(){

  var laptopObject = Laptop("Asus", "Intel");
  print(laptopObject.merk);
  print(laptopObject.processor);

}
```

Topik

Inheritance

Inheritance

- Inheritance atau pewarisan adalah kemampuan untuk menurunkan sebuah class ke class lain
- Dalam artian, kita bisa membuat class Parent dan class Child
- Class Child, hanya bisa punya satu class Parent, namun satu class Parent bisa punya banyak class Child
- Saat sebuah class diturunkan, maka semua field dan method yang ada di class Parent, secara otomatis akan dimiliki oleh class Child
- Untuk melakukan pewarisan, di class child, kita harus menggunakan kata kunci extends lalu diikuti dengan nama class parent nya.

Contoh Inheritance

```
class Manager {
      String? name;
      void sayHello(String name){
     print('Hello $name, my name is ${this.name}');
   }
}
```

Contoh Memanggil Inheritance

```
class VicePresident extends Manager {

}

void main(){
  var manager = Manager();
  manager.name = “Sembiring";
  manager.sayHello(‘Tarigan’);

  var vp = VicePresident();
  vp.name = " Sembiring ";
  vp.sayHello(‘Tarigan');
}
```

Topik

Polymorphism

Polymorphism

- Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk.
- Dalam OOP, Polymorphism adalah kemampuan sebuah object berubah bentuk menjadi bentuk lain
- Polymorphism erat hubungannya dengan Inheritance

Contoh Inheritance

```
class Employee {
  String name;
  Employee(this.name);
}

class Manager extends Employee{
  Manager(String name) : super(name);
}

class VicePresident extends Manager{
  VicePresident(String name) : super(name);
}
```

```
void main(){
  Employee employee = Employee('Eko');
  print(employee);

  employee = Manager('Eko');
  print(employee);

  employee = VicePresident('Eko');
  print(employee);
}
```

Topik

Import

Import

- Import adalah kemampuan untuk menggunakan class, function atau variable yang berada di file yang berbeda
- Import sangat cocok digunakan ketika kode program kita sudah banyak dan ditempatkan di bayak file

Contoh Import

```dart
class Category {
  String id;
  String name;


 Category(this.id, this.name);

}
```

```dart
import 'data/category.dart';

   void main(){
  var category = Category('1',
   'Laptop');
  print(category.id);
  print(category.name);
}
```

Topik

Abstract Class

Abstract Class

- Saat kita membuat class, kita bisa menjadikan sebuah class sebagai abstract class.
- Abstract class artinya, class tersebut tidak bisa dibuat sebagai object secara langsung, hanya bisa diturunkan
- Untuk membuat sebuah class menjadi abstract, kita bisa menggunakan kata kunci abstract sebelum kata kunci class
- Dengan demikian abstract class bisa kita gunakan sebagai kontrak untuk class child

Contoh Abstract Class

```dart
abstract class Location{
  String? name;
}

  class City extends Location {
  City(String name){
    this.name = name;
  }
}
```

```dart
import 'data/location.dart';

void main(){
var city = City('Medan');
}
```

Topik

Access Modifier

Access Modifier

- Secara default, saat kita membuat Field atau Method dalam sebuah class, maka semua Field dan Method tersebut bisa diakses oleh siapa saja
- Kadang-kadang kita ingin menyembunyikan Field atau Method
- Untuk menyembunyikan akses Field atau Method, kita bisa menambahkan _ (underscore) di awal Field atau Method nya
- Secara otomatis Field atau Method tersebut, hanya bisa di akses dari file tersebut, dan artinya tidak bisa diakses dari luar file

Contoh Access Modifier

```dart
class Product {

    String? id;
  String? name;
  int? _quantity;

    String toString(){
     return "Product{id=$id,
    name=$name, quantity=$_quantity}";
  }
}
```

```dart
import 'data/product.dart';

void main(){
var product = Product();
product.id = '1';
product.name = 'Laptop';
// product._quantity = 100; erorr
}
```

Topik

Static

Static

- Static merupakan sebuah kata kunci yang digunakan untuk membuat field dan method yang bisa diakses tanpa harus membuat object nya
- Saat kita membuat field atau method yang static, artinya dia secara otomatis bisa diakses oleh object manapun, oleh karena itu perlu hati-hati ketika membuat field atau method status
- Field atau Method static bisa diakses oleh Method yang tidak static
- Namun Field atau Method static tidak bisa mengakses Field atau Method yang tidak static
- Biasanya static digunakan untuk membuat utility
- Untuk mengakses static Field atau Method, kita bisa langsung menggunakan nama Class diikuti dengan titik dan static Field atau Method nya

Contoh Access Modifier

```
class Application {

  static final String kampus = ‘Politeknik Wilmar Bisnis Indonesia';
  static final String prodi = ‘TRPL';

}

void main(){
  print(Application.name);
  print(Application.author);
}
```

Topik

Interface

Access Modifier

- Sebelumnya kita sudah tahu bahwa abstract class bisa kita gunakan sebagai kontrak untuk class child nya.
- Namun sebenarnya yang lebih tepat untuk kontrak adalah Interface
- Jangan salah sangka bahwa Interface disini bukanlah User Interface
- Interface mirip seperti abstract class, yang membedakan adalah di Interface, semua method otomatis abstract
- Untuk mewariskan interface, kita tidak menggunakan kata kunci extends, melainkan implements

Contoh Access Modifier

```dart
class Product {

    String? id;
  String? name;
  int? _quantity;

    String toString(){
     return "Product{id=$id,
    name=$name, quantity=$_quantity}";
  }
}
```

```dart
import 'data/product.dart';

void main(){
var product = Product();
product.id = '1';
product.name = 'Laptop';
// product._quantity = 100; erorr
}
```

Topik

Callable Class

Callable Class

- Callable Class merupakan class yang bisa dipanggil seperti function
- Untuk membuat Callable Class, kita perlu menambahkan sebuah method bernama call() di class tersebut
- Parameter dan Return Value dari Method tersebut bisa disesuaikan sesuai keinginan kita
- Setelah membuat objectnya, kita bisa langsung memanggil method call() tersebut menggunakan nama objectnya saja

Contoh Callable Class

```
class Sum {

  int first;
  int second;

  Sum(this.first, this.second);

  int call(){
    return first + second;
  }

}

void main(){

  var sum = Sum(10, 10);
  print(sum());

}
```

Topik

Exception

Exception

- Saat kita membuat aplikasi, kita tidak akan terhindar dengan yang namanya error
- Error direpresentasikan dengan istilah exception, dan semua direpresentasikan dalam bentuk class exception
- Kita bisa menggunakan class exception sendiri, atau menggunakan yang sudah disediakan
- Untuk membuat sebuah exception, kita bisa menggunakan kata kunci throw, diikuti dengan object exception nya
- https://api.dart.dev/stable/2.14.4/dart-core/Exception-class.html

Contoh Exception

```
class Validation {
  static void validate(String username, String password) {
    if (username == "") {
      throw ValidationException("Username is blank");
    } else if (password == "") {
      throw ValidationException("Password is blank");
    } else if (username != ‘politeknik' || password != ‘wbi') {
      throw Exception('Login failed');
    }
    // valid
  }
}
```

Topik

Try Catch

Try Catch

- Saat kita memanggil sebuah method yang bisa menyebabkan exception, maka secara otomatis program akan berhenti.
- Jika kita tidak ingin program berhenti, kita perlu menangkap exception tersebut, dan melakukan sesuatu ketika terjadi exception
- Untuk menangkap exception, kita bisa menggunakan try-catch
- Cara menggunakan try-catch sangat mudah, di block try, kita tinggal panggil method yang bisa menyebabkan exception, dan di block catch, kita bisa melakukan sesuatu jika terjadi exception

Contoh Exception

```
try {
    Validation.validate(“politeknik", "salah");
  } on ValidationException catch (exception, stackTrace) {
    print('Validation Error : ${exception.message}');
    print('Stack Trace : ${stackTrace.toString()}');
  } on Exception catch (exception, stackTrace) {
    print('Error : ${exception.toString()}');
    print('Stack Trace : ${stackTrace.toString()}');
  } finally {
    print('Finally');
  }
```

TERIMA KASIH....