Pemrograman Mobile Pertemuan 4

OLEH : NANDANG HERMANTO

Functions

Object-oriented programming (OOP)

Functions

- Sekumpulan kode program yang bersama-sama melakukan tugas tertentu, dapat dipanggil dari bagian program yang lain secara berulang ulang sesuai dengan kebutuhan.
- digunakan untuk memecah kode besar menjadi modul yang lebih kecil dan menggunakannya kembali saat dibutuhkan.
- Fungsi membuat program lebih mudah dibaca dan mudah di-debug.
- meningkatkan pendekatan modular dan meningkatkan penggunaan kembali kode.
- Function bisa mengebalikan nilai dan bisa tidak mengambelikan nilai
- Function yang tidak menghasilkan/mengembailkan nilai diawali dengan kata **void**
- Function yang menghasilkan/mengembalikan nilai diawali dengan **type data**

Membuat Function

```
void func_name (daftar_parameter):
{
   isi program
   isi program
   isi program
}
```

```
type_data func_name (daftar_parameter):
{
   isi program
   isi program
   isi program
   return <nilaibalik>
}
```

Func_nama ➔nama fungsi

Daftar_parameter➔digunakan untuk mengirim data kedalam function, sama seperti deklarasi variable,boleh ada dan boleh tidak ada

Isi program ➔ berisi kode program yang akan dieksekusi

return <nilaibalik> ➔ nilai yang dikembalikan/yang dihasilan dari fungsi

Function tanpa nilai balik

Function Tanpa parameter

```
void infoMahasiswa() {
  print("Nim : SA2100001");
  print("Nama : Nandang");
  print("Email : nanadang007@gmail.com");
}

Run | Debug
void main(List<String> args) {
  infoMahasiswa();
}
```

Function dengan parameter

```
void infoMahasiswa(String nim, nama, email) {
  print("Nim : $nim");
  print("Nama : $nama");
  print("Email : $email");
}

Run | Debug
void main(List<String> args) {
  infoMahasiswa("SA2100001", "Nandang", "nanadang007@gmail.com");
}
```

Function dengan parameter optional

```dart
void infoMahasiswa(String nim, String nama, [String? email]) {
  print("Nim : $nim");
  print("Nama : $nama");
  print("Email : $email");
}

Run | Debug
void main(List<String> args) {
  infoMahasiswa("SA2100001", "Nandang");
  infoMahasiswa("SA2100001", "Nandang", "nanadang007@gmail.com");
}
```

Function dengan Named parameter

```dart
void infoMahasiswa({String? nim, String? nama, String? email}) {
  print("Nim : $nim");
  print("Nama : $nama");
  print("Email : $email");
}

Run | Debug
void main(List<String> args) {
  infoMahasiswa(nim: "SA210001");
  infoMahasiswa(nim: "SA210001",nama: "Nandang Hermanto");
  infoMahasiswa(nim: "SA210001",nama: "Nandang Hermanto",email: "nandang007@gmail.com");
}
```

```dart
void infoMahasiswa({required String? nim, required String? nama, String? email}) {
  print("Nim : $nim");
  print("Nama : $nama");
  print("Email : $email");
}

Run | Debug
void main(List<String> args) {
  infoMahasiswa(nim: "SA210001");
  infoMahasiswa(nim: "SA210001",nama: "Nandang Hermanto");
  infoMahasiswa(nim: "SA210001",nama: "Nandang Hermanto",email: "nandang007@gmail.com");
}
```

Function dengan nilai balik

```dart
double tambah(double bil1, double bil2) {
  return bil1 + bil2;
}

Run | Debug
void main(List<String> args) {
  print(tambah(20, 30));
}
```

```dart
double operasiAritmatika(double bil1, double bil2, String jenisOperasi) {
  double hasil = 0;
  if (jenisOperasi == "+") {
    hasil = bil1 + bil2;
  } else if (jenisOperasi == "-") {
    hasil = bil1 - bil2;
  } else if (jenisOperasi == "*") {
    hasil = bil1 * bil2;
  } else if (jenisOperasi == "/") {
    hasil = bil1 / bil2;
  }

  return hasil;
}

Run | Debug
void main(List<String> args) {
  print(operasiAritmatika(6, 2, "+"));
  print(operasiAritmatika(6, 2, "-"));
  print(operasiAritmatika(6, 2, "*"));
  print(operasiAritmatika(6, 2, "/"));
}
```

Function Short Expression

```dart
double terkecil(double bil1, double bil2) => bil1 < bil2 ? bil1 : bil2;

Run | Debug
void main(List<String> args) {
  print(terkecil(20, 10));
}
```

Contoh Penggunaan dalam Flutter

```
void operasiAritmatika(String jenisOperasi) {
  double bil1 = double.parse(conBilangan1.text);
  double bil2 = double.parse(conBilangan2.text);
  double hasil = 0;
  if (jenisOperasi == "+") {
    hasil = bil1 + bil2;
  } else if (jenisOperasi == "-") {
    hasil = bil1 - bil2;
  } else if (jenisOperasi == "*") {
    hasil = bil1 * bil2;
  } else if (jenisOperasi == "/") {
    hasil = bil1 / bil2;
  }

  conBilanganHasil.text = hasil.toString();
}
```

```
child: ElevatedButton(
  onPressed: () {
    setState(() {
      operasiAritmatika("+");
    });
  },
  child: Text("Tambah"),
), // ElevatedButton
```

```
child: ElevatedButton(
  onPressed: () {
    setState(() {
      operasiAritmatika("-");
    });
  },
  child: Text("Kurang"),
), // ElevatedButton
```

```
child: ElevatedButton(
  onPressed: () {
    setState(() {
      operasiAritmatika("*");
    });
  },
  child: Text("Kali"),
), // ElevatedButton
```

```
child: ElevatedButton(
  onPressed: () {
    setState(() {
      operasiAritmatika("/");
    });
  },
  child: Text("Bagi"),
), // ElevatedButton
```

OOP DART

Paradigma OOP berdasarkan pada konsep objek yang memiliki atribut serta dapat melakukan operasi atau prosedur tertentu

Contoh :

bayangkan kucing sebagai sebuah objek.

Objek kucing ini memiliki karakteristik seperti warna bulu, usia kucing, dan berat badan. Ciri-ciri ini disebut dengan attributes atau properties.

Selain itu kucing juga bisa melakukan beberapa hal seperti makan, tidur, dan bermain. Perilaku pada objek kucing ini adalah sebuah method

4 pilar pemrograman berorientasi objek
Encapsulation

Enkapsulasi adalah kondisi di mana status atau kondisi di dalam class, dibungkus dan bersifat privat. Artinya objek lain tidak bisa mengakses atau mengubah nilai dari property secara langsung.

Pada contoh kasus kucing kita tidak bisa langsung mengubah berat badan dari kucing, namun kita bisa menambahkannya melalui fungsi atau method makan.

4 pilar pemrograman berorientasi objek
Abstraction

Abstraksi bisa dibilang merupakan penerapan alami dari enkapsulasi. Abstraksi berarti sebuah objek hanya menunjukkan operasinya secara high-level.

Misalnya kita cukup tahu bagaimana seekor kucing makan, namun kita tidak perlu tahu seperti apa metabolisme biologis dalam tubuh kucing yang membuat berat badannya bertambah.

4 pilar pemrograman berorientasi objek **Inheritance**

Beberapa objek bisa memiliki beberapa karakteristik atau perilaku yang sama, namun mereka bukanlah objek yang sama.

Kucing memiliki sifat dan perilaku yang umum dengan hewan lain, seperti memiliki warna, berat, dsb. Maka dari itu kucing sebagai objek turunan (subclass) mewarisi semua sifat dan perilaku dari objek induknya (superclass).

ikan juga mewarisi sifat dan perilaku yang sama, namun ikan bisa berenang sementara kucing tidak.

4 pilar pemrograman berorientasi objek
Polymorphism

Polymorphism dalam bahasa Yunani berarti “banyak bentuk.” Sederhananya objek dapat memiliki bentuk atau implementasi yang berbeda-beda pada satu metode yang sama.

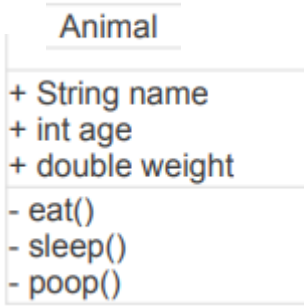
Semua hewan bernafas, namun tentu kucing dan ikan memiliki cara bernafas yang berbeda. Perbedaan bentuk atau cara pernafasan tersebut merupakan contoh dari polymorphism.

Class

Class merupakan sebuah blueprint untuk membuat objek. Di dalam kelas ini kita mendefinisikan sifat (attribute) dan perilaku (behaviour) dari objek yang akan dibuat. Sebagai contoh kelas Animal memiliki atribut berupa nama, berat, dan umur, dll. Kemudian perilakunya adalah makan, tidur, dsb

Class

Class merupakan sebuah blueprint untuk membuat objek. Di dalam kelas ini kita mendefinisikan sifat (attribute) dan perilaku (behaviour) dari objek yang akan dibuat. Sebagai contoh kelas Animal memiliki atribut berupa nama, berat, dan umur, dll. Kemudian perilakunya adalah makan, tidur, dsb




```
1.  class Animal {
2.    String name;
3.    int age;
4.    double weight;
5.
6.    Animal(this.name, this.age, this.weight);
7.
8.    void eat() {
9.      print('$name is eating.');
10.     weight = weight + 0.2;
11.   }
12.
13.   void sleep() {
14.     print('$name is sleeping.');
15.   }
16.
17.
18.   void poop() {
19.     print('$name is pooping.');
20.     weight = weight - 0.1;
21.   }
22. }
```

Menggunakan Class

Class yang sudah dibuat bisa digunakan untuk menciptakan Object, setiap object yang dibuat memiliki semua attribute dan method yang dimiliki oleh class nya

```
void main(List<String> arguments) {
  Animal kucing=Animal();

  kucing.eat();
  kucing.sleep();
  kucing.poop();
}
```

Properties & Methods

Variabel dan fungsi di dalam class dikenal dengan property dan method

Untuk membuat property bersifat privat tambahkan underscore (_) sebelum nama property, kemudian untuk mengakses property tersebut gunakan getter dan setter

Jika class diletakan di file lain maka tambahkan import pada file yang menggunakan class tersebut

librarysaya.dart

```dart
class Animal{
  String _neme;
  int _age;
  double _weight;

  Animal(this._neme, this._age, this._weight);

  double get weight => _weight;

  set weight(double value) {
    _weight = value;
  }

  int get age => _age;

  set age(int value) {
    _age = value;
  }

  String get neme => _neme;

  set neme(String value) {
    _neme = value;
  }
}
```

latihanoop.dart

```dart
import 'librarysaya.dart';
void main(List<String> arguments) {

  Animal kucing=Animal("Kucing",10,30);

  kucing.age=20;
  print(kucing.age);

  kucing.neme="Simeong";
  print(kucing.neme);

  kucing.weight=3;
  print(kucing.weight);

}
```

Constructor

Constructor adalah fungsi spesial dari sebuah kelas yang digunakan untuk membuat objek.

Constructor memiliki nama yang sama dengan nama kelas.

Constructor tidak memiliki nilai kembalian (return type).

Constructor akan secara otomatis dipanggil ketika sebuah objek dibuat.

Jika kita tidak mendefinisikan constructor, default constructor tanpa argumen akan dibuat.

Pada beberapa kasus kita mungkin akan membutuhkan beberapa constructor untuk skenario yang berbeda-beda. Pada situasi ini kita bisa memanfaatkan named constructor.

Contoh Constructor

```dart
class Animal{
  String _neme="";
  int _age=0;
  double _weight=0;

  Animal(this._neme, this._age, this._weight);
  Animal.name(this._weight);
  Animal.weight(this._weight);
  Animal.age(this._age);
  Animal.nw(this._neme, this._weight);
```

```dart
Animal kucing=Animal("Kucing",10,30);
Animal kucing1=Animal.age(10);
Animal kucing2=Animal.nw("Simeong", 3);
```

Inheritance

Inheritance adalah kemampuan suatu program untuk membuat kelas baru dari kelas yang ada.

Konsep inheritance ini bisa dibayangkan layaknya seorang anak mewarisi sifat dari orang tuanya.

Di dalam OOP kelas yang menurunkan sifat disebut sebagai kelas induk (parent class/superclass) sementara kelas yang mewarisi kelas induknya disebut sebagai kelas anak (child class/subclass).

Inheritance

Cat	Fish	Bird
+ name **+ weight** **+ age** + furColor	**+ name** **+ weight** **+ age** + skinColor	**+ name** **+ weight** **+ age** + featherColor
- eat() **- sleep()** **- poop()** - walk()	**- eat()** **- sleep()** **- poop()** - swim()	**- eat()** **- sleep()** **- poop()** - fly()

Bisa kita lihat pada tabel di atas bahwa objek Cat, Fish, dan Bird memiliki beberapa *property* dan *method* yang sama seperti name, weight, age, eat(), dan sleep().

Inheritance

```dart
class Animal {
  String _name = '';
  int _age;
  double _weight = 0;

  Animal(this._name, this._age, this._weight);

  String get name => _name;

  double get weight => _weight;

  void eat() {
    print('$_name is eating.');
    _weight = _weight + 0.2;
  }

  void sleep() {
    print('$_name is sleeping.');
  }
}
```

```dart
class Cat extends Animal {
  String furColor = "";

  Cat(String name, int age, double weight, String furColor): super(name, age, weight) {
    this.furColor = furColor;
  }

  void walk() {
    print('$name is walking');
  }
}
```

Abstract Class

Sesuai namanya, abstract merupakan gambaran umum dari sebuah kelas.

Ia tidak dapat direalisasikan dalam sebuah objek.

Secara harfiah hewan merupakan sebuah sifat. Kita tidak tahu bagaimana objek hewan tersebut. Kita bisa melihat bentuk kucing, ikan, dan burung namun tidak untuk hewan.

Maka dari itu konsep abstract class perlu diterapkan agar kelas Animal tidak dapat direalisasikan dalam bentuk objek namun tetap dapat menurunkan sifatnya kepada kelas turunannya.

```
1.  abstract class Animal {
2.      String name;
3.      int age;
4.      double weight;
5.
6.      // ...
7.  }
```

Implicit Interface

Selain abstract class, cara lain yang bisa kita gunakan untuk menerapkan abstraksi dalam OOP adalah dengan interface. Interface atau antarmuka merupakan set instruksi yang bisa diimplementasi oleh objek. Secara umum, interface berfungsi sebagai penghubung antara sesuatu yang abstrak dengan sesuatu yang nyata

Dart tidak memiliki keyword atau syntax untuk mendeklarasikan interface seperti bahasa pemrograman OOP lainnya.

Setiap class di dalam Dart dapat bertindak sebagai interface. Maka dari itu interface pada Dart dikenal sebagai implicit interface.

Untuk mengimplementasikan interface, gunakan keyword implements.

Kita bisa mengimplementasikan beberapa interface sekaligus pada satu kelas

Implicit Interface

```dart
abstract class Animal {
  String name;
  int age;
  double weight;

  // ...
}
```

```dart
class Flyable {
  void fly() { }
}
```

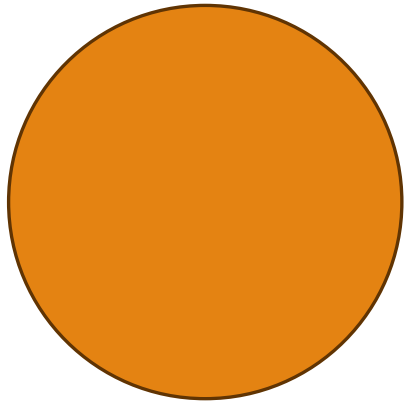
```dart
class Bird extends Animal implements Flyable {
  String featherColor;

  Bird(String name, int age, double weight, this.featherColor) : super(name, age, weight);

  @override
  void fly() {
    print('$name is flying');
  }

}
```

Contoh Penerapan OOP (Pewarisan)

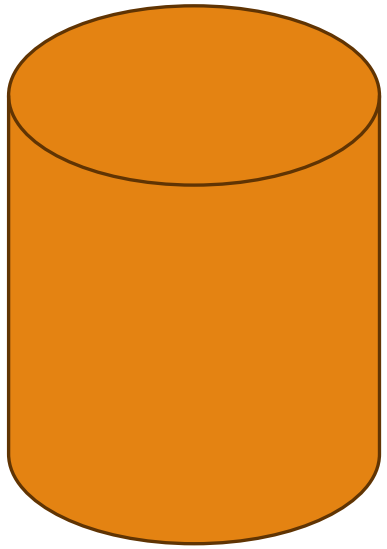


Luas Lingkaran : $\pi \times r^2$

Keliling Lingkaran : $2 \times \pi \times r$

Maka dapat diambil kesimpulan

Class Lingkaran dijadikan sebagi Class induk
Class Tabung dijadikan Sebagai Class Perluasan `extends`



Volume Tabung = $\pi r^2 t$ **= Luas Lingkaran X tinggi**

Luas Selimut Tabung = $2\pi r t$ **= Keliling Lingkaran X tinggi**

Luas Permukaan Tabung = $2\pi r^2 + 2\pi rt$ **= 2 X luas lingkarang + Keliling lingkaran X tinggi**

Contoh Penerapan OOP (Pewarisan)

```dart
class Lingkaran {
  double? _jariJari;

  Lingkaran([this._jariJari]);

  double? get getJariJari {
    return _jariJari;
  }

  set setJariJari(jariJari) {
    _jariJari = jariJari;
  }

  double getLuasLingkaran() {
    return 22 / 7 * _jariJari! * _jariJari!;
  }

  double getKelilingLingkaran() {
    return 2 * 22 / 7 * _jariJari!;
  }
}
```

```dart
class Tabung extends Lingkaran {
  double? _tinggi;

  Tabung([super._jariJari, this._tinggi]);

  double? get getTinggi {
    return _tinggi;
  }

  set setTinggi(double tinggi) {
    _tinggi = tinggi;
  }

  double getVolumeTabung() {
    return getLuasLingkaran() * _tinggi!;
  }

  double getLuasSelimutTabung() {
    return getKelilingLingkaran() * _tinggi!;
  }

  double getLuasPermukaanTabung() {
    return getLuasSelimutTabung() + 2 * getLuasLingkaran();
  }
}
```

Terimakasih