**By: Eng. Elias Al-Showaiter**

# [Dart Programming Language] - Lecture [8]

# [Classes]

## 1. Introduction

Understanding classes in Dart is essential for mastering object-oriented programming (OOP) in the language. Although we have studied multiple concepts so far in this course, creating complex full stack programs is yet not possible with what we have got, the missing peace is to understand classes. Classes allow us to be free in creating all kinds of attributes and behaviors we want.

Learning Objectives

1. Understand the basics of classes.
2. Be able to create classes.
3. Understand the advanced topics in classes.

## 2. Context

Technical Background

When we advance in this course, we will find ourselves needing to describe objects that are not easily defined using the default environment tools, classes provide a fast and clean method to do that. As a developer, in the end of this course, you will realize that everything is built from classes.

Key Terms and Definitions:

1. Class: it is a group of attributes and methods used to describe an object, and it is used to create instance from this object.
2. Attributes: variables that are used to describe the object.
3. Methods: functions that are used to describe the object.
4. Object: an instance of a class.
5. Constructor: a special method that is used to create the object from the class.

## 3. Main Content

### 1. Defining a class:

A class is defined using the class keyword followed by the class name. here is the syntax:

```
class ClassName {
 // Instance variables
 // Constructor
 // Methods
}
```

Let’s take a look at a simple example:

```
class Car {

//attributes

  String brand;

  int year;

  // Constructor

  Car(this.brand, this.year);

  // Method

  void displayInfo() {

    print('Brand: $brand, Year: $year');

  }

}
```

### 2. Attributes:

Attributes also known as (fields or properties) are variables defined in the class, they are used to describe the class we are creating, the more attributes you have the more you describe the class in detail.

### 3. Constructor:

A **constructor** is a special method used to create and initialize an object. Dart allows for several types of constructors, including default, named, and factory constructors.

**Default Constructor:** If you don't provide a constructor, Dart automatically provides a default constructor.

### 4. Methods:

Methods are functions defined inside a class that can operate on attributes and perform specific actions.

### 5. Objects:

Like functions, class definition does not execute the code, we need to call the class in the main function, like this:

```
void main() {

  Car myCar = Car('Toyota', 2015);

  myCar.displayInfo(); // Output: Brand: Toyota, Year: 2015

}
```

## 6. Inheritance:

Inheritance is the process of moving all the attributes and methods from a class into another class, the class who gives the attributes and methods is called the parent class or the super class, and the class that receives the attributes and methods is called the child class or the subclass.

```
class Animal {
  String type;
  Animal(this.type);
  void move() {
    print('Animal is moving');
  }}
class Dog extends Animal {
  String family;
  String color;
  Dog(this.family, this.color, String type) : super(type);
  void description() {
    print('this dog belongs to the $family family');
  }}
void main() {
  Dog anter = Dog('local', "dog", "black");
  anter.move(); // Animal is moving
  anter.description(); // this dog belongs to local family
}}
```

We created two classes, the first called animal, with 1 attributes called type, and 1 method called move, and used a constructor. The second class is called dog and inherits the attribute type and method move, and they are as if they belong to the class dog. So in the constructor we are required to include the attributes in the parent class.

## 7. Polymorphism:

Polymorphism allows objects to be treated as instances of their parent class. The most common use of polymorphism in Dart is through method overriding, where a subclass provides a specific implementation of a method that is already defined in its superclass.

```
class Animal {
  String type;
  Animal(this.type);
  void move() {
    print('Animal is moving');
  }}
class Dog extends Animal {
  String family;
  String color;
  Dog(this.family, this.color, String type, : super(type);
  @override
   void move() {
    print('the dog is moving');
  }}
void main() {
  Animal anter = Dog('local', "dog", "black");
  anter.move(); // the dog is moving
}
```

## 8. Encapsulation:

Encapsulation refers to the practice of bundling data (attributes) and methods (functions) that operate on that data within a single unit or class. Encapsulation also involves restricting direct access to some of an object's components, which is why it is often described as "data hiding."

```
class BankAccount {

  // Private attribute

  double _balance = 0.0;

  void deposit(double amount) {

    if (amount > 0) {

      _balance += amount;

    }

  }

  // Public method to get the balance

  double get balance => _balance;

}

void main() {

  BankAccount account = BankAccount();

  account.deposit(100);

  print('Balance: ${account.balance}'); // Output: Balance: 100.0

 account.balance = 100; // ERROR

}
```

## 9. Static Attributes and methods:

**static members** are attributes and methods that belong to the class itself rather than to any instance of the class. Static members are shared among all instances of the class.

```
class MathStatic {
  static double pi = 3.14159;
  static double circleArea(double radius) {
    return pi * radius * radius;
  }}
void main() {
  print('Area of circle: ${ MathStatic.circleArea(5)}');
  print('Area of circle: ${ MathStatic.pi }');
}
```