

Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 1
Dart OOP Part 2

Lecture 6



Table Of Content

- Introduction
- Encapsulation
- Inheritance
- Super Constructors



Introduction

OOP

- ◆ Object-oriented Programming, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.



OOP Concepts

01

Encapsulation

02

Inheritanc
e

03

Polymorphism

04

Abstraction

Encapsulation

Encapsulation

- ◆ In Dart, **Encapsulation** means **hiding data** within a library, preventing it from outside factors. It helps you control your program and prevent it from becoming too complicated.



Encapsulation

The ingredients and preparation details are hidden within the "Pizza" class, ensuring that you only need to know what kind of pizza you want, not how it's made.



Library In Dart

- ◆ By default, every **.dart** file is a library. A library is a collection of functions and classes. A library can be imported into another library using the **import** keyword.



Encapsulation

Encapsulation can be achieved by:

- Declaring the class properties as **private** by using **underscore(_)**.
- Providing public **getter** and **setter** methods to access and update the value of private property.

Note: Dart doesn't support keywords like **public**, **private**, and **protected**. Dart uses **_** (underscore) to make a property or method private. The encapsulation happens at library level, not at class level.



Getter and Setter

Getter and **setter** methods are used to access and update the value of private property. **Getter** methods are used to access the value of private property. **Setter** methods are used to update the value of private property.



```
class Employee {  
    // Private properties  
    int? _id;  
    String? _name;  
  
    // Getter method to access private property _id  
    int getId() {  
        return _id!;  
    }  
    // Getter method to access private property _name  
    String getName() {  
        return _name!;  
    }  
    // Setter method to update private property _id  
    void setId(int id) {  
        this._id = id;  
    }  
    // Setter method to update private property _name  
    void setName(String name) {  
        this._name = name;  
    }  
}  
  
void main() {  
    // Create an object of Employee class  
    Employee emp = new Employee();  
    // setting values to the object using setter  
    emp.setId(1);  
    emp.setName("John");  
  
    // Retrieve the values of the object using getter  
    print("Id: ${emp.getId()}");  
    print("Name: ${emp.getName()}");  
}
```

Id: 1
Name: John

Private properties

Private property is a property that can only be accessed from same **library**. Dart does not have any keywords like **private** to define a private property. You can define it by prefixing an **underscore (_) to its name.**



```
class Employee {  
    // Private property  
    var _name;  
  
    // Getter method to access private property _name  
    String getName() {  
        return _name;  
    }  
  
    // Setter method to update private property _name  
    void setName(String name) {  
        this._name = name;  
    }  
}  
  
void main() {  
    var employee = Employee();  
    employee.setName("Jack");  
    print(employee.getName());  
}
```

Jack

Private properties

Why Aren't Private Properties Private?

In the main method, if you write the following code, it will compile and run without any error. Let's see why it is happening.

```
class Employee {  
    // Private property  
    var _name;  
  
    // Getter method to access private property _name  
    String getName() {  
        return _name;  
    }  
  
    // Setter method to update private property _name  
    void setName(String name) {  
        this._name = name;  
    }  
}  
  
void main() {  
    var employee = Employee();  
    employee._name = "John"; // It is working, but why?  
    print(employee.getName());  
}
```



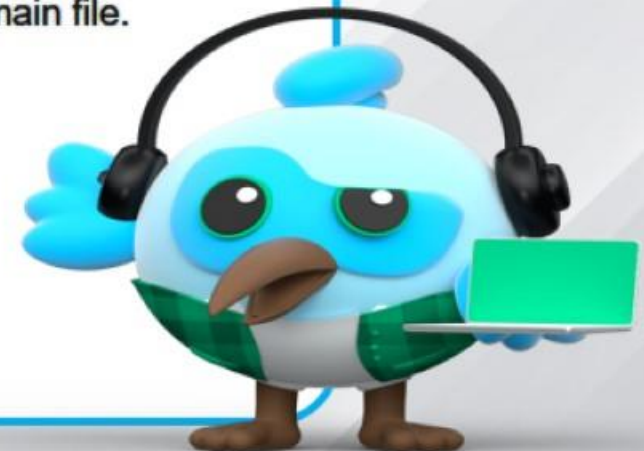
Private properties

Reason:

The reason is that using **underscore** (`_`) before a variable or method name makes it **library private** not **class private**. It means that the variable or method is only visible to the library in which it is declared. It is not visible to any other library. In simple words, library is one file. If you write the main method in a separate file, this will not work.

Solution:

To see private properties in action, you must create a separate file for the class and import it into the main file.



Read only properties

You can control the properties access and implement the encapsulation in the dart by using the read-only properties. You can do that by adding the **final** keyword before the properties declaration. Hence, you can only access its value, but you cannot change it.

Note: Properties declared with the **final** keyword must be initialized at the time of declaration. You can also initialize them in the constructor.

Note: In dart, any identifier like (class, class properties, top-level function, or variable) that starts with an underscore _ it is private to its library.



```
class Student {  
    final _schoolname = "ABC School";  
  
    String getSchoolName() {  
        return _schoolname;  
    }  
}  
  
void main() {  
    var student = Student();  
    print(student.getSchoolName());  
    // This is not possible  
    //student._schoolname = "XYZ School";  
}
```

ABC School

Encapsulation

Why Encapsulation Is Important?

- **Data Hiding:** Encapsulation hides the data from the outside world. It prevents the data from being accessed by the code outside the class. This is known as data hiding.
- **Testability:** Encapsulation allows you to test the class in isolation. It will enable you to test the class without testing the code outside the class.
- **Flexibility:** Encapsulation allows you to change the implementation of the class without affecting the code outside the class.
- **Security:** Encapsulation allows you to restrict access to the class members. It will enable you to limit access to the class members from the code outside the library.



Inheritance

Inheritance

Inheritance is a sharing of behaviour between two classes. It allows you to define a class that extends the functionality of another class. The **extend** keyword is used for inheriting from parent class.

Note: Whenever you use inheritance, it always create a **is-a** relation between the parent and child class like **Student is a Person, Truck is a Vehicle, Cow is a Animal** etc.



Inheritance

Dart supports single inheritance, which means that a class can only inherit from a single class. Dart does not support multiple inheritance which means that a class cannot inherit from multiple classes.

In this syntax, **Parent Class** is the superclass and **Child Class** is the subclass. The **Child Class** inherits the properties and methods of the **Parent Class**.

```
class ParentClass {  
    // Parent class code  
}  
  
class ChildClass extends ParentClass {  
    // Child class code  
}
```



Inheritance

In this syntax, **Parent Class** is the superclass and **Child Class** is the subclass. The **Child Class** inherits the properties and methods of the **Parent Class**.

```
class ParentClass {  
    // Parent class code  
}  
  
class ChildClass extends ParentClass {  
    // Child class code  
}
```



Inheritance

Parent Class: The class whose properties and methods are inherited by another class is called parent class. It is also known as base class or super class.

Child Class: The class that inherits the properties and methods of another class is called child class. It is also known as derived class or subclass.



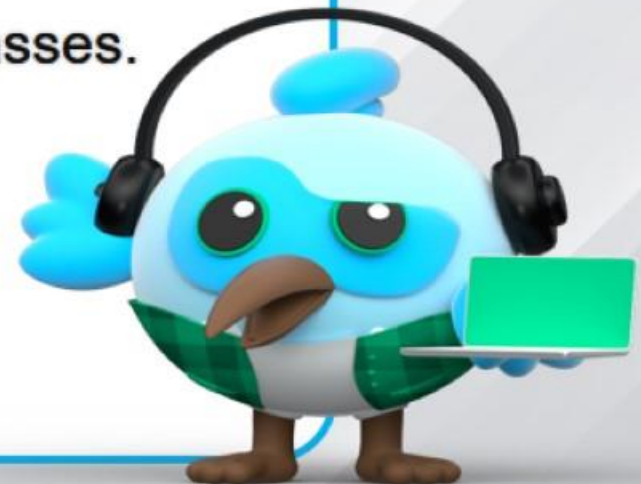
```
class Person {  
    // Properties  
    String? name;  
    int? age;  
  
    // Method  
    void display() {  
        print("Name: $name");  
        print("Age: $age");  
    }  
}  
  
// Here In student class, we are extending the  
// properties and methods of the Person class  
class Student extends Person {  
    // Fields  
    String? schoolName;  
    String? schoolAddress;  
  
    // Method  
    void displaySchoolInfo() {  
        print("School Name: $schoolName");  
        print("School Address: $schoolAddress");  
    }  
}  
  
void main() {  
    // Creating an object of the Student class  
    var student = Student();  
    student.name = "John";  
    student.age = 20;  
    student.schoolName = "ABC School";  
    student.schoolAddress = "New York";  
    student.display();  
    student.displaySchoolInfo();  
}
```

```
Name: John  
Age: 20  
School Name: ABC School  
School Address: New York
```

Inheritance

Advantages Of Inheritance In Dart

- It promotes reusability of the code and reduces redundant code.
- It helps to design a program in a better way.
- It makes code simpler, cleaner and saves time and money on maintenance.
- It facilitates the creation of class libraries.
- It can be used to enforce standard interface to all children classes.



```
class Car{
    String color;
    int year;

    void start(){
        print("Car started");
    }
}

class Toyota extends Car{
    String model;
    int price;

    void showDetails(){
        print("Model: $model");
        print("Price: $price");
    }
}

void main(){
    var toyota = Toyota();
    toyota.color = "Red";
    toyota.year = 2020;
    toyota.model = "Camry";
    toyota.price = 20000;
    toyota.start();
    toyota.showDetails();
}
```

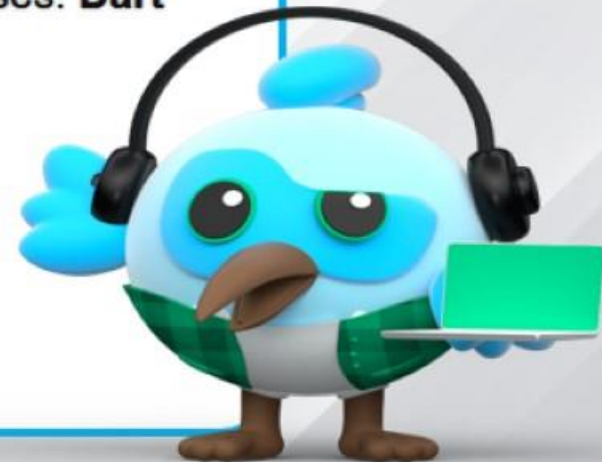
```
Car started
Model: Camry
Price: 20000
```

Inheritance

Types Of Inheritance In Dart

1. **Single Inheritance** - In this type of inheritance, a class can inherit from only one class. In Dart, we can only extend one class at a time.
2. **Multilevel Inheritance** - In this type of inheritance, a class can inherit from another class and that class can also inherit from another class. In Dart, we can extend a class from another class which is already extended from another class.
3. **Hierarchical Inheritance** - In this type of inheritance, parent class is inherited by multiple subclasses. For example, the **Car** class can be inherited by the **Toyota** class and **Honda** class.
4. **Multiple Inheritance** - In this type of inheritance, a class can inherit from multiple classes. **Dart does not support multiple inheritance.**

For e.g. **Class Toyota extends Car, Vehicle {}** is not allowed in Dart.



Single Inheritance

In this example below, there is super class named **Car** with two properties **name** and **price**. There is sub class named **Tesla** which inherits the properties of the super class. The sub class has a method **display** to display the values of the properties.

```
class Car {  
    // Properties  
    String? name;  
    double? price;  
}  
  
class Tesla extends Car {  
    // Method to display the values of the properties  
    void display() {  
        print("Name: ${name}");  
        print("Price: ${price}");  
    }  
}  
  
void main() {  
    // Create an object of Tesla class  
    Tesla t = new Tesla();  
    // setting values to the object  
    t.name = "Tesla Model 3";  
    t.price = 50000.00;  
    // Display the values of the object  
    t.display();  
}
```

```
Name: Tesla Model 3  
Price: 50000.0
```



Multilevel Inheritance

In this example below, there is super class named **Car** with two properties **name** and **price**. There is sub class named **Tesla** which inherits the properties of the super class. The sub class has a method **display** to display the values of the properties. There is another subclass named **Model3** which inherits the properties of the sub class **Tesla**. The sub class **Model3** has a method **display** to display the values of the properties.

```
class Car {  
    // Properties  
    String name;  
    double price;  
}  
  
class Tesla extends Car {  
    // Method to display the values of the properties  
    void display() {  
        print("Name: ${name}");  
        print("Price: ${price}");  
    }  
}  
  
class Model3 extends Tesla {  
    // Properties  
    String color;  
  
    // Method to display the values of the properties  
    void display() {  
        super.display();  
        print("Color: ${color}");  
    }  
}  
  
void main() {  
    // Create an object of Model3 class  
    Model3 m = new Model3();  
    // setting values to the object  
    m.name = "Tesla Model 3";  
    m.price = 50000.00;  
    m.color = "Red";  
    // Display the values of the object  
    m.display();  
}
```

```
Name: Tesla Model 3  
Price: 50000.0  
Color: Red
```



Hierarchical Inheritance

In this example below, there is class named **Shape** with two properties **diameter 1** and **diameter 2**. There is sub class named **Rectangle** with method **area** to calculate the area of the rectangle. There is another subclass named **Triangle** with method **area** to calculate the area of the triangle.

```
class Shape {  
    // Properties  
    double? diameter1;  
    double? diameter2;  
}  
  
class Rectangle extends Shape {  
    // Method to calculate the area of the rectangle  
    double area() {  
        return diameter1! * diameter2!;  
    }  
}  
  
class Triangle extends Shape {  
    // Method to calculate the area of the triangle  
    double area() {  
        return 0.5 * diameter1! * diameter2!;  
    }  
}  
  
void main() {  
    // Create an object of Rectangle class  
    Rectangle r = new Rectangle();  
    // setting values to the object  
    r.diameter1 = 10.0;  
    r.diameter2 = 20.0;  
    // Display the area of the rectangle  
    print("Area of the rectangle: ${r.area()}");  
  
    // Create an object of Triangle class  
    Triangle t = new Triangle();  
    // setting values to the object  
    t.diameter1 = 10.0;  
    t.diameter2 = 20.0;  
    // Display the area of the triangle  
    print("Area of the triangle: ${t.area()}");  
}
```

Area of the rectangle: 200.0
Area of the triangle: 100.0



Inheritance

Key Points

- Inheritance is used to reuse the code.
- Inheritance is a concept which is achieved by using the **extends** keyword.
- Properties and methods of the super class can be accessed by the sub class.
- Class **Dog** extends class **Animal**{ } means Dog is sub class and Animal is super class.
- The sub class can have its own properties and methods.



Inheritance

Why Dart Does Not Support Multiple Inheritance?

Dart does not support multiple inheritance because it can lead to ambiguity. For example, if class **Apple** inherits class **Fruit** and class **Vegetable**, then there may be two methods with the same name **eat**. If the method is called, then which method should be called? This is the reason why Dart does not support multiple inheritance.



Super Constructor

Super

What Is Super In Dart?

Super is used to refer to the parent class. It is used to call the parent class's properties and methods.

```
class Laptop {  
  // Method  
  void show() {  
    print("Laptop show method");  
  }  
}  
  
class MacBook extends Laptop {  
  void show() {  
    super.show(); // Calling the show method of the parent class  
    print("MacBook show method");  
  }  
}  
  
void main() {  
  // Creating an object of the MacBook class  
  MacBook macbook = MacBook();  
  macbook.show();  
}
```

Laptop show method
MacBook show method



Super

Accessing Super Properties In Dart

In this example below, the **display()** method of the **Tesla** class calls the **noOf Seats** property of the parent class using the **super** keyword.

```
class Car {  
  int noOfSeats = 4;  
}  
  
class Tesla extends Car {  
  int noOfSeats = 6;  
  
  void display() {  
    print("No of seats in Tesla: $noOfSeats");  
    print("No of seats in Car: ${super.noOfSeats}");  
  }  
}  
  
void main() {  
  var tesla = Tesla();  
  tesla.display();  
}
```

```
No of seats in Tesla: 6  
No of seats in Car: 4
```



Super

Super With Constructor In Dart

In this example below, the **Manager** class constructor calls the **Employee** class constructor using the **super** keyword.

```
class Employee {  
  // Constructor  
  Employee(String name, double salary) {  
    print("Employee constructor");  
    print("Name: $name");  
    print("Salary: $salary");  
  }  
}  
  
class Manager extends Employee {  
  // Constructor  
  Manager(String name, double salary) : super(name, salary) {  
    print("Manager constructor");  
  }  
}  
  
void main() {  
  Manager manager = Manager("John", 25000.0);  
}
```

```
Employee constructor  
Name: John  
Salary: 25000.0  
Manager constructor
```



Super

Super With Named Constructor In Dart

In this example below, the **Manager** class named constructor calls the **Employee** class named constructor using the **super** keyword.

```
class Employee {  
  // Named constructor  
  Employee.manager() {  
    print("Employee named constructor");  
  }  
}  
  
class Manager extends Employee {  
  // Named constructor  
  Manager.manager() : super.manager() {  
    print("Manager named constructor");  
  }  
}  
  
void main() {  
  Manager manager = Manager.manager();  
}
```

Employee named constructor
Manager named constructor



Super

Super With Multilevel Inheritance In Dart

In this example below, the **MacBook Pro** class method **display** calls the **display** method of the parent class **MacBook** using the **super** keyword. The **MacBook** class method **display** calls the **display** method of the parent class **Laptop** using the **super** keyword.

```
class Laptop {  
  // Method  
  void display() {  
    print("Laptop display");  
  }  
}  
  
class MacBook extends Laptop {  
  // Method  
  void display() {  
    print("MacBook display");  
    super.display();  
  }  
}  
  
class MacBookPro extends MacBook {  
  // Method  
  void display() {  
    print("MacBookPro display");  
    super.display();  
  }  
}  
  
void main() {  
  var macbookpro = MacBookPro();  
  macbookpro.display();  
}
```

```
MacBookPro display  
MacBook display  
Laptop display
```



Super

Key Points To Remember

- The **super** keyword is used to access the parent class members.
- The **super** keyword is used to call the method of the parent class.



What is Next ?



Thank You