### Definitions of OOP Concepts

**"Abstract classes, static,**

**Encapsulation, polymorphism"**

Object Oriented Programming is a programming paradigm that believes in grouping data (properties) and methods (actions) together inside a box. It demonstrates the pattern of real-world objects.

**Dart OOP features**

Dart supports all the features for Object-oriented programing paradigm like Classes, Inheritance, Interfaces, Polymorphism, etc. Inheritance in Dart might seem a little weird but apart from that, everything is alright.

Various OOP features can be implemented in dart.

They are:

* Class
* Objects
* Inheritance
* Polymorphism
* Encapsulation

**Classes**

Class is a user defined data type and it contains its own data members(Constructors , fields, getters and setters) and member functions. A class encapsulates data for the object.

A class in Dart can be declared by using the keyword class followed by the class name and the body of the class should be enclosed with a pair of curly braces {}

Dart classes do not support constructor overloading, but you can use the flexible function argument specifications from the language (optional, positional, and named) to provide different ways to instantiate a class. Also, you can have named constructors to define alternatives.

Declaring class in Dart:



### Static variables

Static variables (class variables) are useful for class-wide state and constants:

class Queue {

static const initialCapacity = 16;

// ···

}

void main() {

assert(Queue.initialCapacity == 16);

}

Static variables aren’t initialized until they’re used.

### Static methods

Static methods (class methods) don’t operate on an instance, and thus don’t have access to this. They do, however, have access to static variables. As the following example shows, you invoke static methods directly on a class:

import 'dart:math';

class Point {

double x, y;

Point(this.x, this.y);

static double distanceBetween(Point a, Point b) {

var dx = a.x - b.x;

var dy = a.y - b.y;

return sqrt(dx \* dx + dy \* dy);

}

}

void main() {

var a = Point(2, 2);

var b = Point(4, 4);

var distance = Point.distanceBetween(a, b);

assert(2.8 < distance && distance < 2.9);

print(distance);

}

## Abstract

To define a class that doesn’t require a full, concrete implementation of its entire interface, use the abstract modifier.

Abstract classes cannot be constructed from any library, whether its own or an outside library. Abstract classes often have [abstract methods](https://dart.dev/language/methods#abstract-methods).

// Library a.dart

abstract class Vehicle {

void moveForward(int meters);

}

// Library b.dart

import 'a.dart';

// Error: Cannot be constructed

Vehicle myVehicle = Vehicle();

// Can be extended

class Car extends Vehicle {

int passengers = 4;

// ···

}

// Can be implemented

class MockVehicle implements Vehicle {

@override

void moveForward(int meters) {

// ...

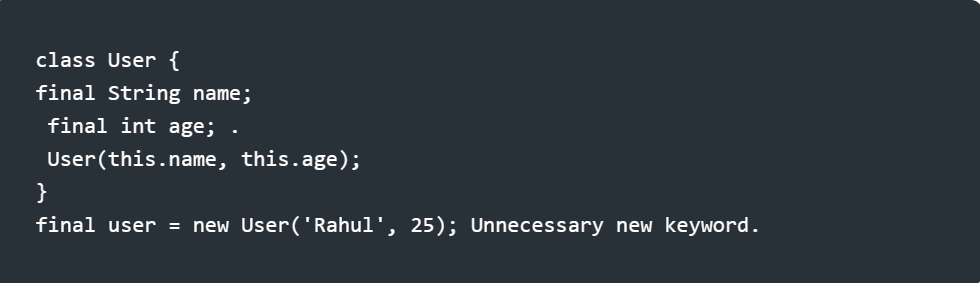
}

}

### ****Creating objects****

Like most OOP languages, Dart supports the keyword new for creating instances of classes. Here is an example of a traditional object instantiation, using the new keyword:

As you can see, flutter SDK gives us hints that we can omit the new keyword. Which means you can create object instances without using the new keyword.



### ****Objects****

Objects are basic building blocks of a Dart program. An object is a combination of data and methods. and everything is treated as an object in Dart. An object is a variable or instance of the class used to access the class's properties. Objects have two features - state and behavior.

All created objects implicitly inherit from the base objects: the Object.

### ****Some Benefit of Objects****

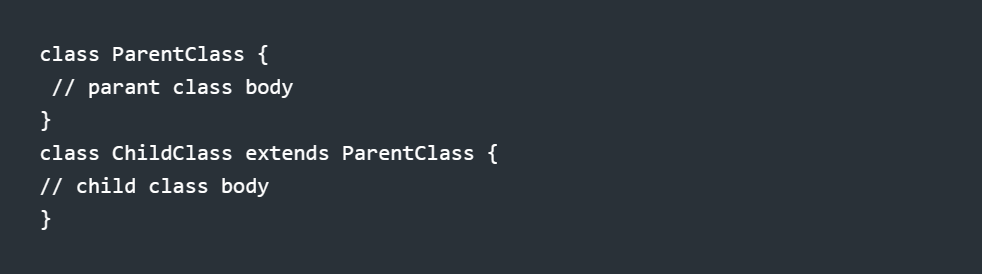
**Modularity:** The source code of an object can be maintained individually and can hide from the other object's source code.

**Data - hiding:** Using oops programming, the details of the internal functionality of code are hidden from the others. For example - Users only interact with the application, but they aren't familiar with the internal implementation.

**Reusability** - We don't need to write the same code again and again. We can use the object of class multiple times in our program.

### ****Inheritance****

Inheritance allows you to define a class that extends the functionality of another class.With the help of Inheritance, one class can make use of all the properties and characteristics of parent class i.e base class. Dart supports single inheritance. It means that a class can inherit from a single class. Dart doesn’t support multiple inheritances. In other words, a class cannot inherit from two or more classes. To define a class that inherits from another class, you use the extends keyword as follows:



The child class will have all properties and methods of the parent class. Also, it can extend the parent class by either overriding the methods from the parent class or having more methods in the child class.

**Example:**

class Sports {

final String name;

const Sports({this.name = ''});

void score(int count){

print("${count} point scored!");

}

void win(){

print("${name}: Game Won!");

}

}

class Cricket extends Sports {

Cricket() : super(name: 'Cricket Format');

void gameFormat(String format) {

print("${format} Cricket");

}

}

class Badminton extends Sports {

Badminton() : super(name: 'Badminton');

startService() {

print("Service started!");

}

}

void main() {

var cricket = Cricket();

cricket.gameFormat("T20");

cricket.score(4);

cricket.win();

print(“--------------------------------”);

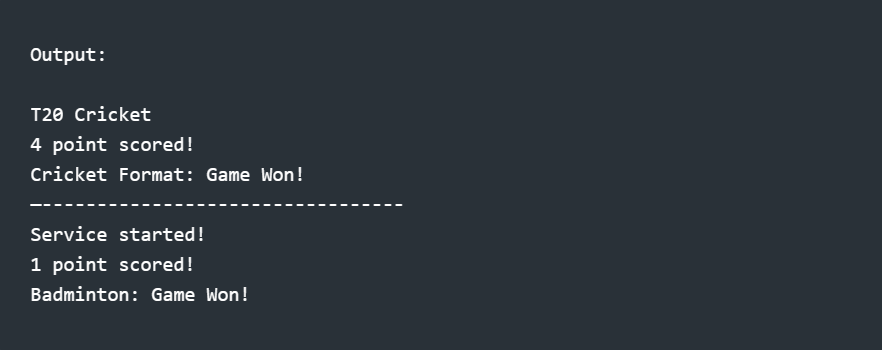
var badminton = Badminton();

badminton.startService();

badminton.score(1);

badminton.win();

}



**Polymorphism**

Polymorphism is an essential concept in an object-oriented programming language. In polymorphism, an object can take multiple forms. As the word suggests, **poly** means **many** and **morph** means **forms**, hence, polymorphism means having multiple forms. Polymorphism is generally used to achieve the inheritance mechanism.

Polymorphism in Dart is supported only in the form of runtime polymorphism (For example, method overriding). Subclasses usually override instance methods, getters, and setters. We can use the @override annotation to indicate that we’re overriding a member.

**Features of Polymorphism**

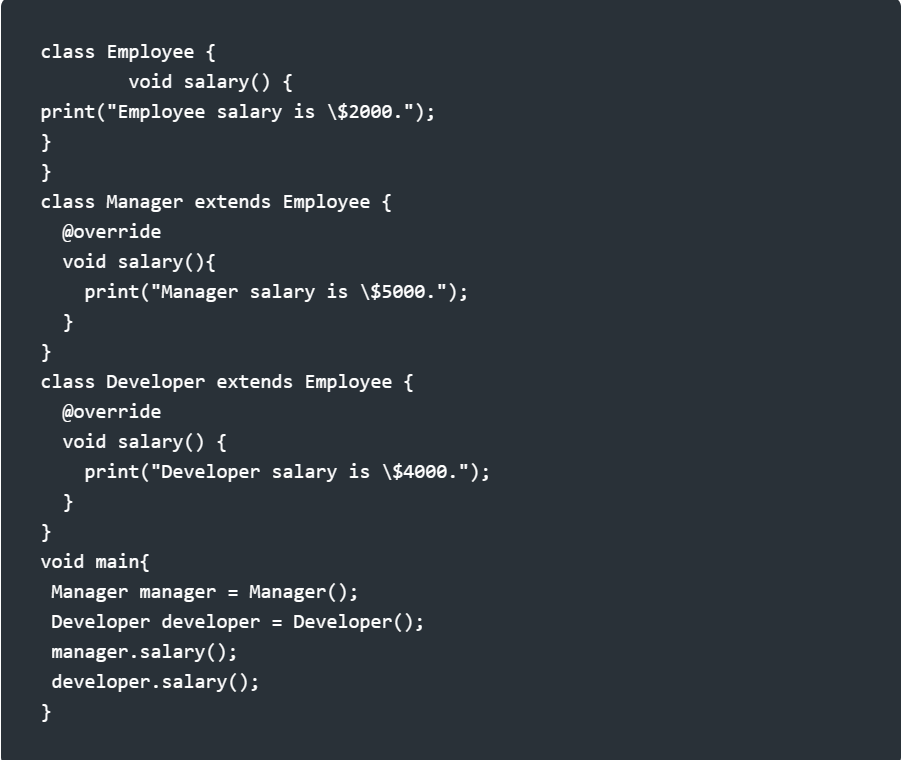
* The functionality of a method behaves differently in different scenarios.
* It allows the same name for a member or method in a class with different data types.
* The behavior of a method depends on the data provided.
* Polymorphism supports implicit type conversion..

**Advantage Of Polymorphism In Dart**

* Subclasses can override the behavior of the parent class.
* It allows us to write code that is more flexible and reusable.
* Reusable code
* Reduces coupling between different functionalities.

**Polymorphism By Method Overriding In Dart**

In this example below, there is a class named Employee with a method named salary(). The salary() method is overridden in two child classes named Manager and Developer.



In the above example, the @override annotation means the method is overridden from the parent class. Here the method salary() in the Manage and Developer class is overridden by the parent class Employee.

**Encapsulation**

Encapsulation is the principle that limits access to an object's state; and the bundling of methods and operations that do work on a set of data.

The whole idea behind encapsulation is to hide the implementation details from users.

If a data member is private it means it can only be accessed within the same class. No outside class can access private data members of other classes.

However if we set up public getter and setter methods to update the private data fields then the outside class can access those private data fields via public methods.

**Benefits of Encapsulation**

* **Data Hiding:** The users will have no idea about the implementation details of the class. It will not be visible to the user how the class is storing the values. The user will only know that values are being passed and initialized.
* **More Flexibility:** We can make the variables of the class read-only or write-only according to our requirements. If we want to make the variables read-only then we have to use the setter methods like setName(), SetAge(), etc, and if we want to make the variables write only then we need to use the getter methods like getName(), getValue(), etc from the program.
* **Reusability:** Encapsulation also provides the feature of code reusability and is easy to change with newer requirements.
* **Easy Testing:** The encapsulated code is easy to test for unit testing.

In conclusion, Dart, a robust and flexible programming language, offers strong support for object-oriented programming. It allows developers to create structured, scalable, and readable code by leveraging core concepts such as classes, objects, inheritance, polymorphism, and encapsulation. These features, widely used in real-world applications, offer several benefits such as code reusability, modularity, data hiding, and improved testing capabilities, we discussed the concepts like Classes, Objects, Inheritance, Polymorphism, and Encapsulation, with examples and outlining the benefits of each.