Object Oriented Programming (OOPs)

Object-Oriented Programming (OOP) is a method where you design software using objects that combine data and methods. Classes define the structure and behavior of these objects. It helps in organizing code, making it more manageable and reusable.

### **Features of OOPs**

### Class

### Object

### Encapsulation

### Inheritance

### Polymorphism

### Abstraction

### **Advantages**

1. It is easy to understand and use.
2. It increases reusability and decreases complexity.
3. The productivity of programmers increases.
4. It makes the code easier to maintain, modify and debug.
5. It promotes teamwork and collaboration.
6. It reduces the repetition of code.

What is Class?

In OOP, a **class** is a blueprint for creating objects. A class defines a type of object by specifying the properties (data) and methods (functions) that the objects created from the class will have. Essentially, a class acts as a template for objects, determining what data they will hold and what operations can be performed on that data.

What is an Object?

An object in OOPs is a specific instance of a class that holds data (attributes) and methods (functions) to manipulate that data. It represents a real-world entity or concept in the program.

An object is an instance of a class. You can create multiple objects of the same class.

### **Instantiation**

In object-oriented programming, instantiation is the process of creating an instance of a class. In other words, you can say that instantiation is the process of creating an object of a class.

Types of methods in dart:

1. Instance Method
2. Class Method (Static Method).

Instance Method:

A method that can be accessed by using the instance of class is called instance methods. The instance methods can be no arguments or with arguments. The instance method can access by instance variable **this** keyword.

**Class Method:**

The class method is declared with the **static** keyword. It can be accessed by using the **class name** instead of the class object. These methods are common to all instances of that individual class. The static methods only can access the static variables.

### **Constructor in Dart**

A constructor is a special method used to initialize a new instance or object of a class. The name of the constructor is the same name as the class name and the constructor it is called when an object is created. Constructors can be used to set initial values for instance variables.

**With Constructor:** Person person = Person("John", 30);

**Without Constructor:** Person person = Person();

person.name = "John";

person.age = 30;

Note: In Dart, the String?, int?, and int? with a question mark (?) indicate that these variables can hold null values. This means they are nullable.

What is this keyword?

The this keyword refers to the current instance of the class. It is used to access the properties and methods of the current object.

### **Constructor with Default Values**

### A constructor with default values in Dart is defined by assigning default values to optional parameters within the constructor. Dart allows default values for optional positional parameters and named parameters.

### **Constructor with Optional Parameters**

A constructor with optional parameters allows you to create objects with varying numbers of arguments, making your constructors more flexible.

Dart provides two ways to handle optional parameters:

1. **Optional positional parameters**.
2. **Optional named parameters**.

### Optional Positional Parameters

Optional positional parameters are parameters that are not required when creating an instance. They are enclosed in square brackets [] in the constructor's parameter list.

### Optional Named Parameters (**Constructor with Named Parameters**)

Optional named parameters allow you to specify parameters with names, which can be provided in any order. Named parameters are enclosed in curly braces {}.

Types of Constructor in Dart Programming

1. Default Constructor.
2. Parameterized Constructor.
3. Name Constructor.
4. Constant Constructor.

### **Default Constructor**

The constructor which is automatically created by the dart compiler if you don’t create a constructor is called a default constructor. A default constructor has no parameters. A default constructor is declared using the class name followed by parentheses ().

### **Parameterized Constructor**

Parameterized constructor is used to initialize the instance variables of the class. Parameterized constructor is the constructor that takes parameters. It is used to pass the values to the constructor at the time of object creation.

### **Named Constructor in Dart**

### In most programming languages like java, c++, c#, etc., we can create multiple constructors with the same name. But in Dart, this is not possible. Well, there is a way. We can create multiple constructors with the same name using **named constructors**.

**Note**: Named constructors improves code readability. It is useful when you want to create multiple constructors with the same class name.

### **Constant Constructor in Dart**

**Constant constructor** is a constructor that creates a constant object. A constant object is an object whose value cannot be changed. A constant constructor is declared using the keyword **const**.

**Note**: **Constant Constructor** is used to create a object whose value cannot be changed. It improves the performance of the program.

### **Benefits Of Constant Constructor In Dart**

* Improves the performance of the program.

**Immutable Objects**: When you use a const constructor, Dart creates compile-time constants. This means that the objects are immutable and their values are set once and for all. Immutable objects are often more efficient because they can be reused rather than recreated.

**Reduced Memory Usage**: Objects created with const constructors are canonicalized. This means that the same object instance is reused wherever the same constant value is used. This reduces memory overhead because instead of creating multiple instances of the same value, a single instance is used.

**Faster Instantiation**: Since const objects are created at compile time, the instantiation of these objects is faster at runtime. This is because the object creation and initialization have already been done during compilation, avoiding the need for runtime computation.

**Optimized Comparisons**: Immutable and canonicalized objects can be compared more efficiently. Since their values are fixed and their instances are reused, comparing two const objects is essentially the same as comparing their references, which is very fast.

**Reduced Garbage Collection Pressure**: By reusing instances of const objects, you reduce the number of objects that the garbage collector needs to manage. This can lead to less frequent garbage collection cycles and improved performance.

**Predictable Behavior**: const objects provide predictable behavior because they are immutable. This can lead to fewer bugs and easier reasoning about the state and behavior of your program.

### **Encapsulation in Dart**

### 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.

### **What Is 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.

### **How To Achieve Encapsulation In Dart?**

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.

### **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.

### **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.

### **Getter and Setter Methods**

### **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.

### **Getter in Dart**

**Getter** is used to get the value of a property. It is mostly used to access a **private property’s** value. Getter provide explicit read access to an object properties.

Syntax: return\_type get property\_name {// Getter body}

Note: Instead of writing { } after the property name, you can also write **=>** (fat arrow) after the property name.

### **Setter In Dart**

### **Setter** is used to set the value of a property. It is mostly used to update a **private property’s** value. Setter provide explicit write access to an object properties.

### **Syntax**

set property\_name (value) {

// Setter body

}

**Note:** Instead of writing { } after the property name, you can also write **=>** (fat arrow) after the property name.

### **Why Is Setter Important?**

* It is used to set the value of a private property.
* It is also used for data validation.
* It gives you better control over the data.

### **Inheritance In Dart**

### 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.

### **Syntax**

class ParentClass {

// Parent class code

}

class ChildClass extends ParentClass {

// Child class code

}

### **Terminology**

**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 sub class.

### **Inheritance of Constructor**

Inheritance of constructor in Dart is a process of inheriting the constructor of the parent class to the child class. It is a way of reusing the code of the parent class.

### **Polymorphism in Dart**

Poly means **many** and morph means **forms**. Polymorphism is the ability of an object to take on many forms. As humans, we have the ability to take on many forms. We can be a student, a teacher, a parent, a friend, and so on. Similarly, in object-oriented programming, polymorphism is the ability of an object to take on many forms.

**Note**: In the real world, polymorphism is updating or modifying the feature, function, or implementation that already exists in the parent class.

### **Syntax**

class ParentClass{

void functionName(){

}

}

class ChildClass extends ParentClass{

@override

void functionName(){

}

}

In dart there mainly two forms of polymorphism:

1. Static Polymorphism (Compile-Time Polymorphism).
2. Dynamic Polymorphism (Runtime Polymorphism).

**Static Polymorphism (Compile-time)**

Method Overloading in Dart

Since Dart does not support traditional method overloading, the language provides alternative ways to achieve similar outcomes:

1. Named Parameters.
2. Optional Positional Parameters.
3. Factory Constructors.
4. Different Method Names.
5. Default Parameters.

**Dynamic Polymorphism (Run-time)**

1. Method Overriding.
2. Interfaces (Abstract Classes).
3. Dynamic Type.
4. Duck Typing.
5. Abstract Classes.

Method Overriding:

Method overriding in Dart means redefining a method in a subclass that was already defined in its superclass, providing a specific implementation.

Method Overloading:

Method overloading, which does not support directly, involves defining multiple methods with the same name but different parameters in the same class, achieved in Dart through optional or named parameters instead.

### **Static in Dart**

If you want to define a variable or method that is shared by all instances of a class, you can use the **static** keyword. Static members are accessed using the class name. It is used for **memory management**.

### **Dart Static Variable**

A static variable is a variable when you declare this variable via static keyword in Class for accessing the variable you don’t need to create an object of a class with just by class name it can access everywhere (class\_name.variable\_name).

A static variable is a variable that is shared by all instances of a class. It is declared using the static keyword. It is initialized only once when the class is loaded. It is used to store the **class-level data**.

### **Dart Static Method**

A static method is shared by all instances of a class. It is declared using the static keyword. You can access a static method without creating an object of the class.

### **Syntax**

class ClassName{

static returnType methodName(){

//statements

}

}

### **Key Points to Remember**

1. Static members are accessed using the class name.
2. All instances of a class share static members.

Dart Inheritance

Dart inheritance is defined as the process of deriving the properties and characteristics of another class. It provides the ability to create a new class from an existing class. It is the most essential concept of the oops (Object-Oriented programming approach). We can reuse the all the behavior and characteristics of the previous class in the new class.

* **Parent Class -** A class which is inherited by the other class is called **superclass** or **parent class**. It is also known as a **base class**.
* **Child Class -** A class which inherits properties from other class is called the child class. It is also known as the **derived class** or **subclass**.

Accessing Properties and Methods:

Overriding Methods:

Accessing Superclass Members:

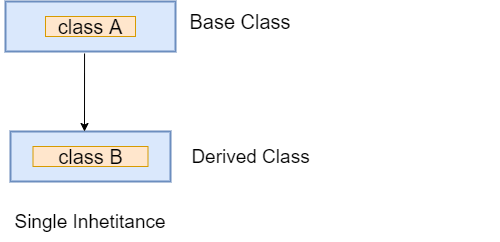
## **Types of Inheritance**

The inheritance can be mainly four types. These are given below.

* Single Inheritance
* Multiple Inheritance
* Multilevel Inheritance
* Hierarchical Inheritance

### **Single Level Inheritance**

In the single inheritance, a class is inherited by a single class or subclass is inherited by one parent class.



### **Multilevel Inheritance**

In the multiple inheritance, a subclass is inherited by another subclass or creates the chaining of inheritance.

### **Hierarchical Inheritance**

In the hierarchical inherence, two or more classes inherit a single class.

# Dart Super Constructor

The child class can inherit all properties (methods, variables) and behavior of parent expect parent class constructor.& The superclass constructor can be invoke in sub class by using the **super()** constructor. We can access both non-parameterized and parameterized constructor of superclass. Accessing the constructor of superclass is slightly different in the Dart.

### **Syntax:**

SubClassConstructor():**super**() {

}

## **Implicit (Indirectly) super**

As we know that the constructor is automatically called when we instantiate a class. When we create the object of sub class, it invokes the constructor of sub class which implicitly invokes the parent class's default (non-parameterized) constructor. We can use **super()** constructor in our subclass to invoke superclass constructor.

## **Explicit super**

If the superclass constructor consists of parameters then we require to call super() constructor with argument in to invoke superclass constructor in subclass explicitly.