**Abstract Class**

**If we declare abstract functions inside the class then this class must be abstract.**

**What is abstract function?**

**It is only to declare function without implementation.**

ex: Void fun\_name ();

**If all functions in class are abstract then the class must have previous keyword abstract.**

ex: abstract class Class\_name{

void fun1();

void fun2();}

**If there is a combination of abstract and non-abstract functions then the class mustn't have abstract keyword.**

ex: class Class\_name{

void fun1(); // abstract function

int fun2(){ // non-abstract function

return 2\*5 ;}}

Generally, abstract classes are used to implement the abstract methods in the extended subclasses.

ex:

// Creating Abstract Class

abstract class Gfg {

    // Creating Abstract Methods

    void say();

    void write();}

class Geeksforgeeks extends Gfg{

    @override

    void say()

    { print("Yo Geek!!");  }

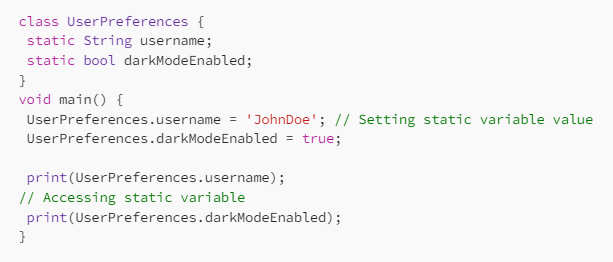
    @override

    void write()

    { print("Geeks For Geeks"); }}

Static

In Dart, “static” is a keyword that can be used to define class members (variables, methods and constructor) that belong to the class itself rather than to instances of the class.

* Static Variables  
  Static variables are declared using the “static” keyword within a class. They are shared among all instances of the class. Static variables are initialized only once and retain their values across multiple instances. They can be accessed using the class name itself, without creating an instance.
* Static Methods  
  Static methods belong to the class itself, rather than the instances of the class. They can be called without creating an instance of the class. Static methods are useful for performing operations or computations that are not specific to any particular instance.
* [Utility Classes](https://medium.com/@yetesfadev/the-power-of-utility-classes-in-dart-a-comprehensive-guide-f1dcdab5edd7):  
  Utility classes are classes that provide helper functions or constants that can be accessed globally without the need for creating instances. They often have private constructors to prevent instantiation.
* Sharing Data:  
  Static variables can be used to share data between different parts of an application, such as screens or widgets. They provide a convenient way to store and access data that needs to be shared without passing it through constructors or function parameters.
* 
* Performance Considerations:  
  While static variables and methods can be beneficial in certain scenarios, their excessive use can lead to tight coupling and make code harder to maintain and test. Additionally, static methods cannot be overridden in subclasses, which may limit flexibility in some cases. It’s important to use static members judiciously.
* Class-level State:  
  Static variables can be used to maintain class-level state, meaning that the variable retains its value across multiple instances of the class. This can be useful when you want to keep track of information that is relevant to the entire class rather than individual instances.
* Initialization Order:  
  Static variables are initialized before any instance of the class is created. This means that you can access and use static variables even without creating an instance of the class.

**Encapsulation:**

This means that the internal representation of an object is hidden from the outside. In Dart, you can create private fields by prefixing an underscore to the name (\_variable), and you can use getters and setters to control access to those fields. This way, you can control how data within an object is accessed or modified.

**Polymorphism:**

Polymorphism allows objects of different classes to be treated as objects of a common superclass. There are two types of polymorphism: compile-time (or static) and runtime (or dynamic). In Dart, you can make use of polymorphism through method overriding, where a method in a subclass can provide a different implementation for a method that is already defined in its superclass.