Dart Tools & Testing

By: Laams Innovation Lab

Revision

- Parts of a class
- Static Overrides
- What are immutable classes?
- What are abstract classes?
- Create class named Paisa, and add addition, subtraction, multiplication, division and equality operators to it.

Content

- **extends** vs **implements**
- Understanding Interfaces and Mixins
- Interfaces and implementation
- Mixins
- Extension Methods
- Generics

All Parts of a Class

- Instance Related Parts:
  - Instance Variables => Object Properties
  - Instance Methods => Object Behaviors
  - Getters & Setters => Read and Write Access to Object Properties
  - Constructors => instantiates Objects.
  - Method Overrides
  - Operators
- Class Related Parts:
  - Static Variables
  - Static Methods

Abstract Classes

Abstract classes are those classes which cannot be **instantiated**, meaning they do not create **objects** directly, unless it has **factory** or **static** **constructor(s)**, and **concrete** **sub-type(s)** to be **instantiated** from those constructors.

Abstract classes are created by adding an **abstract** keyword before a class **definition**.

Abstract classes are **useful** for creating **interfaces**, and a generalized **super class**, whose instantiation is not semantically logical.

Abstract classes often have **abstract methods**, which are **function declarations** with no **concrete implementation**. To create an abstract method, instead of the **function block**, a **semicolon** is added.

Interfaces and Implementation

An Interface, when **defined**, declares a **contract** to be **implemented** by its **sub-classes**, and when used, ***provides*** a clear **API** to the user.

In Dart every **concrete** or **abstract** **class** is implicitly an **interface**. Hence, implement any class as an interface use the keyword **implement**.

The difference between **implements** and **extends** keywords are that when **implementing**, the **sub-class** re-implements all **instance variables**, and **concrete** or **abstract methods**. While, by extending those are inherited.

Also, opposed to extends keyword, the class can implement multiple classes.

Mixins

To **reuse** codes in **multiple** **class** **hierarchies** we can use **Mixins**.

A class with no **constructors**, which **extends** **Object**, can be used as a **Mixin**.

To **create** a Mixin, use the keyword **mixin** with an **identifier** and a **block**, inside which variables and methods are defined.

To **use** a **Mixin** with a class use the **with** keyword after the **class** identifier followed by the **Mixin's** identifier.

To **limit** a Mixin's use to a specific **type**, use the **on** keyword after its identifier followed by the target **type's** identifier.

Extension Methods

You can add **functionality** to **class** without **extending** or **knowing** what functionalities the **class** has using **extension** methods.

Generics

All ready explained.

Module & Components

To create a software **module** or **component** you use a **combination** of **abstract**, **concrete**, **generic classes**, **class hierarchies**, **interfaces**, **static**, **concrete** and **asynchronous methods** and **functions**, and **static** or instance **variables** of different **types**.

Hence, Review the Dart Language Tour.

Dart Test

- The Test Function:
  - test(String description, (){});
- The group function:
  - group(String description, (){});
- The expect function:
  - expect(test, expectation);

Dart Commands

- dart analyze
- dart compile
- dart create
- dart fix
- dart format
- dart migrate
- dart pub
- dart run
- dart test

- dartdoc
- dartaotruntime
- webdev
- build_runner
- build_web_compiler
- Dart Dev Tools