**TITLES**

Flutter:

Widget- stateless and statefull widget

State- emphemeral and app state

Advantages and disadvantages of flutter

Flutter architecture

Dart

Dart is an open-source, general-purpose, object-oriented programming language with C-style syntax developed by **Google in 2011**. The purpose of Dart programming is to create a frontend user interfaces for the web and mobile apps. It is under active development, compiled to native machine code for building mobile apps, inspired by other programming languages such as Java, JavaScript, C#, and is Strongly Typed. Since Dart is a compiled language so you cannot execute your code directly; instead, the compiler parses it and transfer it into machine code.

## Comments

Comments are the lines of **non-executable** code.

* **Make format comments:** It is a single line comment (//)
* **Block Comments:** It is a multi-line comment (/\*...\*/)
* **Doc Comments:** It is a document comment that used for member and types (///).

**Continue**

The continue statement allows you to skip the remaining code inside the loop and immediately jump to the next iteration of the loop.

**Break**

The **break** statement allows you to terminate or stops the current flow of a program and continues execution after the body of the loop.

## Object-Oriented Programming

What is oops?

Object-Oriented Programming, or OOP, is a way of writing computer programs that revolves around the concept of "objects." These objects represent real-world things and ideas, and the program's structure is built around these objects.

Dart is an object-oriented programming language, which means every value in a Dart is an object. A number is also an object in Dart language. Dart programming supports the concept of OOPs features like objects, classes, interfaces, etc.

Class

Template or blueprint to create an object.

Object

Properties and behavior of a class.

an object is an instance of a class and is used to access the properties and methods defined within that class from outside the class. In object-oriented programming, classes serve as blueprints for creating objects. When you create an object of a class, you are creating a specific instance of that class with its own set of data (properties) and behavior (methods).

An object is like a digital representation of a thing or idea in your program. It has characteristics and can do things. For instance, a "dog" object might have attributes like "color" and "breed" and can perform actions like "bark" or "fetch."

Top of Form

Spread operators

Represented by 3 dots(…) and is used to combine a map or list to another one.

## Using the Spread Operator with Lists

var list1 = [1, 2, 3];

var list2 = [4, 5, 6];

var combinedList = [...list1, ...list2];

print(combinedList); // Output: [1, 2, 3, 4, 5, 6]

### Adding Elements to a List

var numbers = [2, 3, 4];

var newNumbers = [1, ...numbers];

print(newNumbers); // Output: [1, 2, 3, 4]

## Using the Spread Operator with Maps

var map1 = {'a': 'apple', 'b': 'banana'};

var map2 = {'c': 'cherry', 'd': 'date'};

var combinedMap = {...map1, ...map2};

print(combinedMap); // Output: {a: apple, b: banana, c: cherry, d: date}

### Adding Elements to a Map

var countries = {'USA': 'Washington D.C.', 'France': 'Paris'};

var newCountries = {'Germany': 'Berlin', ...countries};

print(newCountries); // Output: {Germany: Berlin, USA: Washington D.C., France: Paris}

Cascade operators

Cascades (..) allow you to perform a sequence of operations on the same object.

In other words we can make operations like multiple function call or multiple value assignment sequentially over objects in a single statement

Compile time

Compile time is the period when the programming code is converted to the machine code

Runtime

Runtime is the period of time when a program is running and generally occurs after compile time

Future

Future in Flutter refers to an object that represents a value that is not yet available but will be at some point in the future. A Future can be used to represent an asynchronous operation that is being performed, such as fetching data from a web API, reading from a file, or performing a computation.

Flutter

Widget

Widget is the fundamental building blocks to construct a user interface. Everything in Flutter is a widget, from the simplest elements like text and buttons to more complex structures like entire screens and layouts. Widgets are used to define the structure and appearance of an app.

There are two types of widgets in Flutter:

* **StatelessWidget:** A widget that does not depend on any mutable state. Once a StatelessWidget is created, its properties cannot change. These widgets are used for parts of the UI that don't change dynamically. Indicates that once a StatelessWidget is created, the properties (data) of that widget cannot be modified.

* StatefulWidget: These are widgets that can change dynamically. They have mutable state, and when the state changes, the widget is rebuilt to reflect those changes.

State

**State is a Data. It is information based on which Flutter will build its UI.**

In Flutter conceptually State can be divided into two categories:

* Ephemeral State (Local State)
* App State (Global State)

**Ephemeral State**

An ephemeral State is a Local State. It works on a Single Widget or Some of the Widgets. We can neatly organize this state in a specific widget. Usually, people avoid using State Management Techniques to handle such local states. We can easily use *setState()* for this.

For example, if you have a button that changes color when clicked, the color of the button is a piece of local state. Global state is data that needs to be accessed by multiple widgets within the app.

**App State**

App State is a Global State. It is also called as App level state. Compared to Ephemeral State, App State is complex. It is widely spread and used on multiple pages or sometimes in an entire app. We should not organize this State’s code in a specific widget or page. Many State management approaches exist in Flutter to handle the same like GetX, Provider, Riverpod, etc.

Example of App state is the User’s login status, User payment status, Purchased items by the user, etc.

If you organize global state data in specific code instead of using state management solutions like GetX or Provider, there are a few implications:

**Lack of Centralized Management:**

Without a dedicated state management solution, you might find yourself manually passing state data between different parts of your app. This can lead to less organized and more error-prone code.

**Increased Boilerplate Code:**

You would need to write more boilerplate code for managing state changes, notifying listeners, and handling updates across different widgets.

**Potential Code Duplication:**

Global state data might end up being duplicated or inconsistently managed across your app, leading to maintenance challenges.

**Limited Reactivity:**

Reactivity, which is the ability of the UI to automatically update when the underlying data changes, may be more challenging to achieve without a structured state management solution.

**Difficulty in Scaling:**

As your app grows, managing global state without a dedicated solution might become increasingly complex, making it harder to scale your application.

**Limited DevTools and Features:**

State management solutions like GetX and Provider often come with developer tools and additional features that can make debugging and development more efficient. Not using such tools might result in a less streamlined development process.

class AppState {

String loggedInUser;

bool isPaymentComplete;

List<String> purchasedItems;

// Constructor

AppState({

required this.loggedInUser,

required this.isPaymentComplete,

required this.purchasedItems,

});

}

Advantages of flutter

1. Single Codebase:
   * What it means: You can use one set of code for building apps on different platforms like iOS, Android, web, and desktop.
   * Why it's cool: It saves time and effort because you don't have to write separate code for each platform. Plus, it makes it easier to maintain and keep things consistent.
2. Fast Development (Hot Reload):
   * What it is: A feature called hot reload lets developers instantly see changes in the app without restarting it.
   * Why it's cool: Developers can quickly try out new ideas, fix issues, and see the results right away. No need to wait for the whole app to restart.

Hot reload

A hot reload is a great functionality present in a flutter. It is the easiest and the fastest function which helps you to apply changes, fix bugs, creating UIs, and add features. It takes approximately one second to perform its functionality. In hot reload it does not destroy the preserved state.  But you cannot use a hot reload once the app gets killed.

Hot restart

A hot restart has a slightly different functionality as compared to a hot reload. It is faster as compared to the full restart function. It destroys the preserved states of our app, and the code gets fully compiled again and starts from the default state. It takes more time as compared to hot reload but takes less time than the full restart function.

|  |  |
| --- | --- |
| **Hot Reload** | **Hot Restart** |
| Hot Reload allows us to see the reflected change after bug fixes, building User interfaces and even adding certain features to the app without running your application afresh over and over again. | Hot restart destroys the preserved State value and set them to their default. |
| When Hot Reload is invoked, the host machine checks the edited code since the last compilation and recompiles that again. | Hot Restart takes much higher time than Hot reload. |
| Hot Reload does not work when Enumerated types are changed to regular Classes and when classes are changed to enumerated types. | Hot reload is also know as ‘stateful hot reload’ |
| Hot Reload does not work when generic types are modified | Hot Reload is useful because it saves time by just implementing the functionality based on the closest build class in less than 10secs. |

1. Performance:
   * What it means: Flutter apps run fast because they are turned into native machine code.
2. -Why it's cool: This leads to smooth animations and responsive interfaces, providing a great user experience.
3. Rich UI Experience (Widgets):
   * What it is: Flutter has lots of ready-to-use components (widgets) for creating beautiful and engaging user interfaces.
   * Why it's cool: Developers can easily make apps look good and follow the design styles of both Android and iOS.
4. Access to Native Features:
   * What it means: Flutter allows developers to use features specific to each platform, like camera or GPS.
   * Why it's cool: Apps can take advantage of device capabilities without sacrificing the cross-platform nature of Flutter.
5. Ecosystem and Community:
   * What it is: A growing community of developers sharing tools, tutorials, and support for Flutter.
   * Why it's cool: It provides a wealth of resources, making it easier to learn Flutter and solve problems along the way.
6. Cross-Platform Development:
   * What it means: You can build apps for iOS, Android, web, and desktops all at once using the same code.
   * Why it's cool:Saves time and money because you only need to write and maintain one codebase for multiple platforms. Gets your app to users faster.

Disadvantages of flutter

1. Relatively New Framework:

* What it means: Flutter is newer compared to some other frameworks.
* Considerations: Smaller talent pool and fewer resources; might face challenges finding work in big IT organizations focused on older technologies.

1. App Size:

* What it is: Flutter apps have larger file sizes due to including the Flutter engine and framework.
* Why it matters: Initial download size is bigger, but ongoing efforts are made to optimize app size.

1. Platform Specific Limitation:

* What it means: Some platform-specific features may not be fully supported or might need extra workarounds.
* Considerations: While Flutter provides access to native capabilities, some advanced platform-specific features may require additional effort.

1. Learning Curve:

* What it is: Learning Flutter might take some time due to understanding its concepts, architecture, and widget system.
* Considerations: Despite the learning curve, thorough documentation, tutorials, and community support make it manageable for newcomers.

1. Limited Libraries and Packages:

* What it means: Flutter's ecosystem is growing, but it may have fewer libraries and packages compared to more established frameworks.
* Considerations: The flexibility of Flutter allows developers to use existing native libraries when needed.

1. Dependency on Flutter Update:

* What it means: Updates to Flutter may occasionally introduce breaking changes.
* Considerations: Developers need to keep the app up-to-date with the latest Flutter versions, and major updates might require some adjustments.

Flutter architecture

the flutter framework’s architecture has three main parts or layers:

* Embedder
* Engine
* Framework

Embedder layer

The**embedder layer** has platform-specific embedders, many of which provide an entry point and coordinate with their respective operating system to access services like rendering, storage, and more. It has many embedders for possible targeted platforms.

Java and C++ are used in embedders for android, objective-C/ C++ for IOS and macOS, and C++ for Linux and windows.

The Flutter Engine:

1. What it is:
   * 1. The Flutter Engine is like the powerhouse of Flutter, the framework for building apps.
2. What it does:
   * 1. It's responsible for making sure that the essential parts of Flutter work smoothly together.
     2. This includes handling animations, graphics, file operations, networking, making apps accessible, and providing a way for plugins to extend functionality.
3. Why it's important:
   * 1. The Flutter Engine is the behind-the-scenes magic that makes Flutter apps run on different devices. It's like the engine in a car that powers everything and ensures your app works well across platforms.

Framework layer

In the **framework layer**, the developer interacts with and writes flutter applications. It is written in dart language and has predefined libraries, layouts, and more.

The framework layer has three main layer components, which are the following:

* Foundation layer
* Rendering layer
* Widget layer

1. Foundation layer
   1. To Flutter, some foundational classes and some building block services provide abstraction. Some of the main building block services are animations and gestures.
   2. Flutter can support animations like tween, hero, silver, transform, fade in the widget, animation builder, animated opacity, and any animation related to physics.
   3. The gesture is a widget used to detect gestures like tapping, dragging, and scaling. It has an invisible widget name gesture detector.
2. Rendering layer
   1. This layer is responsible for converting widgets in a flutter to pixels and showing them on the screen. It takes a tree of renderable objects called the widgets tree. Whenever any animation, input, or state of the widget changes, this layer is called, which updates the layout and shows them on the screen.
3. Widget layer
   1. A widget is a like component in ReactJS. Each renderable object has its widget, which the developer uses to make a widget tree. There are many predefined widgets. We can also write code and create a new widget that can be used in an application, just like we make components in ReactJS.

Top of Form

**Database**

**Hive** is a lightweight and fast NoSQL database that can be used to store data in Flutter applications.