**BWT TASK 06**

**Stateless vs. Stateful Widgets in Flutter**

**Stateless Widgets**

* **Definition:** Stateless widgets are immutable, meaning their properties cannot change once they are created. They are used when the user interface does not change dynamically.
* **Usage:** Stateless widgets are ideal for static content where the UI does not need to rebuild.
* **Example:** Text, icon etc

**Stateful Widgets**

* **Definition:** Stateful widgets are mutable and can change their state during the lifetime of the widget. They are used when the user interface needs to change dynamically.
* **Usage:** Stateful widgets are ideal for dynamic content where the UI needs to rebuild based on user interaction or other events.
* **Example:** Checkbox, radio, slider etc

**Differences Between Stateful and Stateless Widgets**

1. **State Management:**
   * **Stateless Widgets:** No internal state to manage; all properties are final.
   * **Stateful Widgets:** Maintains state that can change over time and causes UI to re-render.
2. **Rebuilds:**
   * **Stateless Widgets:** Rebuilds only when the widget tree is rebuilt.
   * **Stateful Widgets:** Rebuilds every time the state changes using setState().
3. **Use Cases:**
   * **Stateless Widgets:** Static content, headers, buttons, etc.
   * **Stateful Widgets:** Forms, sliders, text fields, dynamic lists, etc.

**Lifecycle of Stateful Widgets**

1. **createState():** This method is called when a StatefulWidget is created. Creates an instance of the State object.

@override

State<MyStatefulWidget> createState() => \_MyStatefulWidgetState();

1. **initState():** Called once when the state object is inserted into the widget tree meaning state object is created. initState is called once and only once. Used to initialize variable, streams subscription, change modifier, set up animations, or start data fetching etc.

@override

void initState() {

super.initState();

// Initialize state variables or start animations here

}

1. **didChangeDependencies():** Called immediately after initState() and whenever the dependencies of the State object change e.g., when an InheritedWidget it depends on changes, to appear and disappear keyboard etc.

@override

void didChangeDependencies() {

super.didChangeDependencies();

// Handle changes in dependencies here

}

1. **build():** Describes the part of the UI represented by the widget. This can be after a call to initState(), didChangeDependencies(), didUpdateWidget() or when the state is changed via call to setState().

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('My Stateful Widget'),

),

body: Center(

child: Text('Hello, World!'),

),

);

}

1. **didUpdateWidget():** Called whenever the widget configuration changes (e.g., if the parent widget rebuilds and passes new properties). This is useful for updating state based on new widget properties.

@override

void didUpdateWidget(covariant MyStatefulWidget oldWidget) {

super.didUpdateWidget(oldWidget);

// Update state based on new widget properties here

}

1. **setState():** Called to trigger a rebuild of the widget.

void \_incrementCounter() {

setState(() {

\_counter++;

});

}

1. **deactivate():** Called when the state object is removed from the widget tree but might be reinserted before the current frame ends.

@override

void deactivate() {

super.deactivate();

// Perform cleanup actions here

}

1. **dispose():** Called when the state object is permanently removed from the widget tree. Use this method to dispose of resources (e.g., animation controllers, streams) and perform cleanup.

@override

void dispose() {

// Dispose of resources here

super.dispose();

}

**Hot Reload vs. Hot Restart**

* **Hot Reload:**
  + **Definition:** Injects updated code into the Dart Virtual Machine (VM) and allows you to see changes without losing the app state.
  + **Usage:** Ideal for minor changes such as UI tweaks or bug fixes.
  + **Example:** Changing a text or color in the widget.
* **Hot Restart:**
  + **Definition:** Restarts the app from the beginning and recreates the app state from scratch.
  + **Usage:** Necessary for structural changes in the app, such as adding new dependencies or changing the app’s main method.
  + **Example:** Adding a new dependency or changing app initialization code.

**Code Examples**

**Stateless Widget Example**

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('Stateless Widget Example')),

body: Center(

child: MyStatelessWidget(),

),

),

);

}

}

class MyStatelessWidget extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Text('I am a Stateless Widget');

}

}

**Stateful Widget Example**

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('Stateful Widget Example')),

body: Center(

child: MyStatefulWidget(),

),

),

);

}

}

class MyStatefulWidget extends StatefulWidget {

@override

\_MyStatefulWidgetState createState() => \_MyStatefulWidgetState();

}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

int \_counter = 0;

void \_incrementCounter() {

setState(() {

\_counter++;

});

}

@override

Widget build(BuildContext context) {

return Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Text('You have pushed the button this many times:'),

Text(

'$\_counter',

style: Theme.of(context).textTheme.headline4,

),

ElevatedButton(

onPressed: \_incrementCounter,

child: Text('Increment'),

),

],

);

}

}