**Module 3: Introduction to Flutter Widgets and UI Components**

**Theory Assignments:**

1. **Explain the difference between Stateless and Stateful widgets with examples.**

Understanding the difference between Stateless and Stateful widgets is fundamental in Flutter development. Here's a clear breakdown:

* Stateless vs Stateful Widgets in Flutter

1. Stateless Widget

A Stateless widget is immutable — once it's built, it cannot change its state during its lifetime. It’s ideal for UI elements that don’t need to update dynamically.

Use When:

* UI doesn’t change based on user interaction or data updates.
* You just need to display static content.

Example:

import 'package:flutter/material.dart';

class MyStatelessWidget extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

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

body: Center(

child: Text('I am a Stateless Widget'),

),

);

}

}

2. Stateful Widget

A Stateful widget is mutable — it can change its state during its lifetime. It’s perfect for UI elements that respond to user input, animations, or data changes.

Use When:

* You need to update the UI dynamically.
* You’re handling user interaction (e.g., button clicks, form input).
* You’re working with animations or timers.

Example:

import 'package:flutter/material.dart';

class MyStatefulWidget extends StatefulWidget {

@override

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

}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

int counter = 0;

void \_incrementCounter() {

setState(() {

counter++; // Triggers UI rebuild

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

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

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text('Counter Value: $counter'),

ElevatedButton(

onPressed: \_incrementCounter,

child: Text('Increment'),

),

],

),

),

);

}

}

Summary Table

|  |  |  |
| --- | --- | --- |
| Feature | Stateless Widget | Stateful Widget |
| Mutability | Immutable | Mutable |
| UI Update | No | Yes (via setState) |
| Use Case | Static UI | Dynamic UI |
| Lifecycle Management | Simple | More complex (has state object) |

If you want, I can help you convert one of your existing apps from stateless to stateful or vice versa to see the impact in action. Want to try that?

1. **Describe the widget lifecycle and how state is managed in Stateful widgets.**

understanding the **widget lifecycle** and **state management** in **Stateful widgets** will help you build more responsive and efficient apps. Let’s break it down:

**Widget Lifecycle in Flutter**

Flutter widgets go through a lifecycle, especially **Stateful widgets**, which have two parts:

* The **widget class** (MyWidget) — immutable configuration.
* The **state class** (\_MyWidgetState) — mutable state and logic.

🧬 Key Lifecycle Methods in Stateful Widgets

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| createState() | Called once when the widget is inserted into the widget tree. |
| initState() | Called once when the state object is created. Ideal for initialization. |
| didChangeDependencies() | Called when dependencies change (e.g., inherited widgets). |
| build() | Called every time setState() is triggered. Builds the UI. |
| setState() | Triggers a rebuild by updating the state. |
| didUpdateWidget() | Called when the parent widget changes and needs to update this widget. |
| deactivate() | Called when the widget is removed from the tree temporarily. |
| dispose() | Called when the widget is permanently removed. Ideal for cleanup. |

**Example with Lifecycle Methods**

**class MyStatefulWidget extends StatefulWidget {**

**@override**

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

**}**

**class \_MyStatefulWidgetState extends State<MyStatefulWidget> {**

**@override**

**void initState() {**

**super.initState();**

**print('initState called');**

**}**

**@override**

**void didChangeDependencies() {**

**super.didChangeDependencies();**

**print('didChangeDependencies called');**

**}**

**@override**

**void didUpdateWidget(MyStatefulWidget oldWidget) {**

**super.didUpdateWidget(oldWidget);**

**print('didUpdateWidget called');**

**}**

**@override**

**void dispose() {**

**print('dispose called');**

**super.dispose();**

**}**

**int counter = 0;**

**void \_incrementCounter() {**

**setState(() {**

**counter++;**

**});**

**}**

**@override**

**Widget build(BuildContext context) {**

**print('build called');**

**return Scaffold(**

**appBar: AppBar(title: Text('Lifecycle Demo')),**

**body: Center(**

**child: Column(**

**mainAxisAlignment: MainAxisAlignment.center,**

**children: [**

**Text('Counter: $counter'),**

**ElevatedButton(**

**onPressed: \_incrementCounter,**

**child: Text('Increment'),**

**),**

**],**

**),**

**),**

**);**

**}**

**}**

**How State Is Managed**

**State Object**

* Holds the **mutable data**.
* Lives as long as the widget is in the widget tree.
* Rebuilds the UI when data changes via setState().

**setState() Function**

* Notifies Flutter that the internal state has changed.
* Triggers the build() method to re-render the widget.

**setState(() {**

**counter++; // Updates the state**

**});**

* **Visual Lifecycle Flow**

**createState()**

**↓**

**initState()**

**↓**

**didChangeDependencies()**

**↓**

**build()**

**↓**

**(setState → build again)**

**↓**

**didUpdateWidget()**

**↓**

**deactivate()**

**↓**

**dispose()**

1. **List and describe five common Flutter layout widgets (e.g., Container, Column, Row).**

Layout widgets are the backbone of Flutter UI design. They help you position, size, and align elements on the screen. Here are **five essential Flutter layout widgets** every developer should master:

1. **Container**

A versatile widget used for **styling, positioning, and sizing** child widgets.

* **Features:**
* Add padding, margin, borders, background color, and constraints.
* Can contain a single child.

🧪 Example:

**Container(**

**padding: EdgeInsets.all(16),**

**margin: EdgeInsets.symmetric(horizontal: 20),**

**decoration: BoxDecoration(**

**color: Colors.blue,**

**borderRadius: BorderRadius.circular(10),**

**),**

**child: Text('Hello Container'),**

**)**

2. **Column**

Arranges children **vertically** from top to bottom.

* Features:
* Supports alignment via mainAxisAlignment and crossAxisAlignment.
* Useful for stacking widgets vertically.

Example:

**Column(**

**mainAxisAlignment: MainAxisAlignment.center,**

**crossAxisAlignment: CrossAxisAlignment.start,**

**children: [**

**Text('First'),**

**Text('Second'),**

**Text('Third'),**

**],**

**)**

3. **Row**

Arranges children **horizontally** from left to right.

* **Features:**
* Similar to Column, but for horizontal layout.
* Great for buttons, icons, or horizontal lists.

Example:

**Row(**

**mainAxisAlignment: MainAxisAlignment.spaceAround,**

**children: [**

**Icon(Icons.home),**

**Icon(Icons.search),**

**Icon(Icons.settings),**

**],**

**)**

4. **Expanded**

Takes up **remaining space** in a Row or Column.

* Features:
* Must be used inside Flex widgets like Row or Column.
* Helps distribute space evenly or proportionally.

Example:

**Row(**

**children: [**

**Expanded(child: Container(color: Colors.red)),**

**Expanded(child: Container(color: Colors.green)),**

**Expanded(child: Container(color: Colors.blue)),**

**],**

**)**

5. **Stack**

Overlays widgets on top of each other — like layers.

* Features:
* Children are positioned relative to the edges.
* Useful for badges, overlays, or custom designs.

Example:

**Stack(**

**children: [**

**Container(width: 100, height: 100, color: Colors.yellow),**

**Positioned(**

**top: 10,**

**left: 10,**

**child: Icon(Icons.star, color: Colors.red),**

**),**

**],**

**)**

* **Summary Table**

|  |  |  |
| --- | --- | --- |
| **Widget** | **Layout Direction** | **Key Use Case** |
| Container | N/A | Styling, padding, decoration |
| Column | Vertical | Stack widgets top to bottom |
| Row | Horizontal | Arrange widgets left to right |
| Expanded | N/A | Fill available space in Row/Column |
| Stack | Overlay | Layer widgets on top of each other |