

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

Theory Assignments

Name: Chandvaniya Abhay

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

1. Stateless Widget

Definition

A **StatelessWidget** is a widget whose UI **does not change** after it is built. It does **not store any state**, and once rendered, it remains the same unless the parent widget rebuilds it with new data.

Characteristics

- Immutable (cannot change internally)
- UI depends only on constructor parameters
- No `setState()` method
- Lightweight and faster

When to Use

- Static text or images
- Icons
- UI elements that never change (e.g., labels, headings)

2. Stateful Widget

Definition

A **StatefulWidget** is a widget whose UI **can change dynamically** during runtime due to user interaction, data updates, or other events.

Characteristics

- Has a mutable state
- Uses a separate State class
- Can rebuild UI using `setState()`
- Suitable for interactive elements

When to Use

- Buttons that update UI
- Forms and text fields
- Animations
- Counters, toggles, switches

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

1. Stateful Widget Lifecycle

A **StatefulWidget** consists of **two classes**:

- StatefulWidget → immutable configuration
- State<T> → mutable state and lifecycle methods

Lifecycle Flow (in order)

createState() → initState() → didChangeDependencies()
→ build() → (setState() → build())*
→ deactivate() → dispose()

2. Lifecycle Methods Explained

1 createState()

- Called **once** when the widget is inserted into the widget tree
- Creates the State object

@override

```
_CounterState createState() => _CounterState();
```

2 initState()

- Called **once** after createState()
- Used for initialization logic

Common Uses

- Initialize variables
- Start animations
- API calls

- Add listeners

```
@override  
void initState() {  
  super.initState();  
  count = 0;  
}
```

3 didChangeDependencies()

- Called after initState()
- Called again if inherited widgets change

Common Uses

- Access InheritedWidget
- Theme or localization updates

```
@override  
void didChangeDependencies() {  
  super.didChangeDependencies();  
}
```

4 build()

- Called whenever the UI needs to be rendered
- Should be **pure and fast**
- Depends only on the current state and widget configuration

```
@override  
Widget build(BuildContext context) {  
  return Text('Count: $count');  
}
```

5 setState()

- Triggers a rebuild of the widget
- Notifies Flutter that state has changed

```
setState(() {  
  count++;  
});
```

Important

- setState() does **not** rebuild the entire app
- Only rebuilds the affected widget subtree

6 didUpdateWidget()

- Called when the parent widget rebuilds with new configuration
- Used to respond to widget property changes

```
@override  
void didUpdateWidget(covariant MyWidget oldWidget) {  
  super.didUpdateWidget(oldWidget);  
}
```

7 deactivate()

- Called when the widget is temporarily removed from the tree
- Might be reinserted later

```
@override  
void deactivate() {  
  super.deactivate();  
}
```

8 dispose()

- Called **once** when the widget is permanently removed
- Used to clean up resources

Common Uses

- Cancel timers
- Dispose animation controllers
- Remove listeners

```
@override
void dispose() {
  controller.dispose();
  super.dispose();
}
```

3. State Management in Stateful Widgets

What is State?

State represents **data that can change over time** and affects the UI.

How State is Managed

1. State is stored in the State class
2. UI reads values from the state
3. Changes to state are wrapped in setState()
4. Flutter rebuilds the widget with updated values

Example: State Management

```
class ToggleExample extends StatefulWidget {
  @override
  _ToggleExampleState createState() => _ToggleExampleState();
}
```

```
class _ToggleExampleState extends State<ToggleExample> {
```

```
bool isOn = false;
```

```
@override
```

```
Widget build(BuildContext context) {
```

```
  return Switch(
```

```
    value: isOn,
```

```
    onChanged: (value) {
```

```
      setState(() {
```

```
        isOn = value;
```

```
      });
```

```
    },
```

```
  );
```

```
}
```

```
}
```

4. Key Points to Remember

- StatefulWidget itself is **immutable**
- The **State object** holds mutable data
- setState() tells Flutter to rebuild UI
- Lifecycle methods help manage initialization and cleanup
- Proper lifecycle handling improves performance and avoids memory leaks

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

1. Container

Description

Container is a versatile widget used for **styling, positioning, and sizing** a child widget.

Common Uses

- Padding and margin
- Background color
- Borders and shadows
- Width and height constraints

Example

```
Container(  
  padding: EdgeInsets.all(16),  
  margin: EdgeInsets.all(8),  
  color: Colors.blue,  
  child: Text('Hello'),  
)
```

2. Row

Description

Row arranges its children **horizontally** from left to right.

Common Properties

- mainAxisAlignment
- crossAxisAlignment

Example

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceBetween,  
  children: [  
    Icon(Icons.star),  
    Icon(Icons.star),  
    Icon(Icons.star),  
  ],  
)
```

3. Column

Description

Column arranges its children **vertically** from top to bottom.

Common Properties

- mainAxisAlignment
- crossAxisAlignment

Example

```
Column(  
  mainAxisAlignment: MainAxisAlignment.center,  
  children: [  
    Text('Item 1'),  
    Text('Item 2'),  
  ],  
)
```

4. Expanded

Description

Expanded is used inside Row, Column, or Flex to **fill available space** proportionally.

Common Uses

- Responsive layouts
- Sharing available space

Example

```
Row(  
  children: [  
    Expanded(child: Container(color: Colors.red)),  
    Expanded(child: Container(color: Colors.green)),  
  ],  
)
```

5. Padding

Description

Padding adds **space around a widget** without affecting its layout logic.

Example

```
Padding(  
  padding: EdgeInsets.symmetric(horizontal: 16),  
  child: Text('Padded Text'),  
)
```