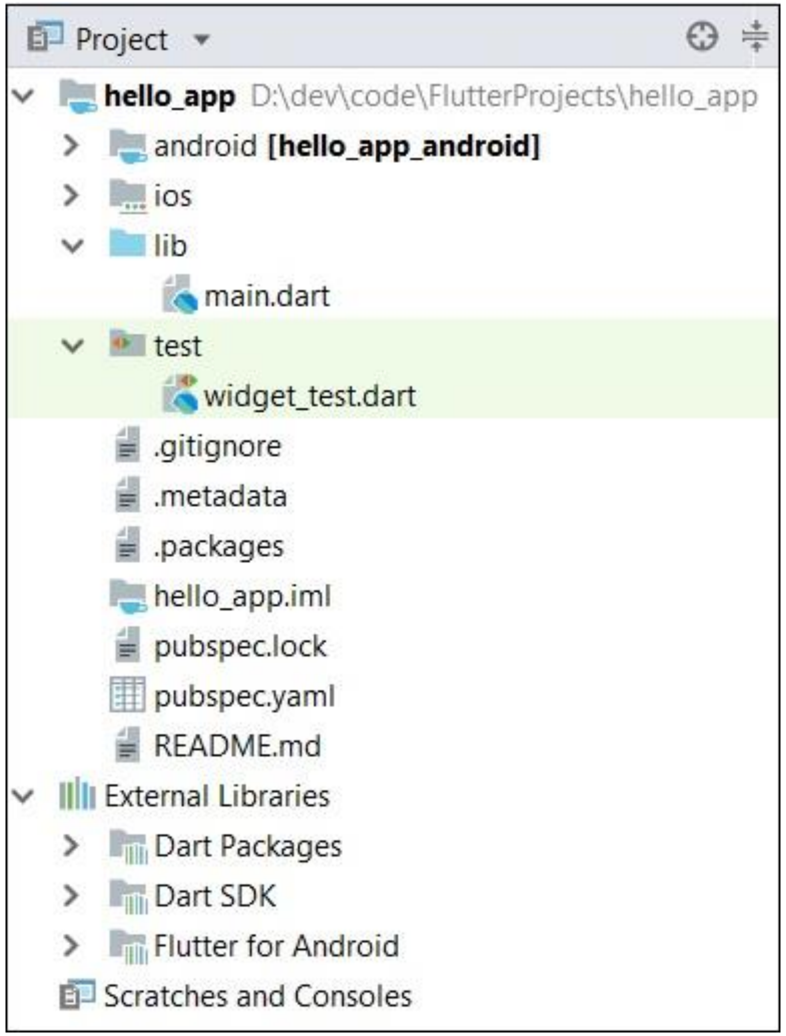
Mobile Application Development

Instructor: Dr. Hammoudeh Alamri

Building a flutter app

Various components of the structure of the application are:



- **android** – Auto generated Android native code
- **ios** – Auto generated iOS native code
- **lib** – Main folder containing Dart code written using flutter framework
- **ib/main.dart** – Entry point of the Flutter application
- **test** – Folder containing Dart code to test the flutter application
- **test/widget_test.dart** – Sample code
- **.gitignore** – Git version control file
- **.metadata** – auto generated by the flutter tools
- **.packages** – auto generated to track the flutter packages
- **.iml** – project file used by Android studio
- **pubspec.yaml** – Used by **Pub**, Flutter package manager
- **pubspec.lock** – Auto generated by the Flutter package manager **Pub.** which contains **(**Metadata and dependencies**)**
- **README.md** – Project description file written in Markdown format

• Creating a Simple App:

- Start by creating a new Flutter project using the command
    - flutter create my_first_app

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
.
.
.
}

class MyHomePage extends StatelessWidget {
.
.
.
}
```

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(title: 'College of
Computers, MAD-lesson1'),
    );
  }
}
```

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
.
.
.
}

class MyHomePage extends StatelessWidget {
.
.
.
}
```

```
class MyHomePage extends StatelessWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'Salam, Guys!, this is Dr. Mazin Alkathiti',
              style: TextStyle(fontSize: 24),
            ),
          ],
        ),
      ),
    );
  }
}
```

Key Components and Their Functions:

1. **main() function:** The entry point of the Flutter application.
    - Calls runApp() to start the app.
2. **MyApp class:** The root widget of the app.
    - Contains the build method that returns the app's widget tree.
3. **MaterialApp widget:** Defines the overall appearance and behavior of the app.

- Properties: title, theme, and home
    - title: Sets the app's title.
    - theme: Defines the app's theme, including colors, fonts, and styles.
    - home: Sets the initial widget to display when the app starts.

4. **MyHomePage class:** The main widget that displays the greeting message.
    - Contains the build method that returns the widget tree for the home page.

5.Scaffold widget: Provides a basic layout structure for the app.

•Properties: appBar, and body

- •appBar: Defines the app bar at the top.
- •body: Sets the main content area of the app.

6.AppBar widget: Creates the app bar at the top of the screen.

- •Contains a Text widget to display the app's title.

7.Center widget: Centers its child widget within its available space.

8.Column widget: Arranges its children vertically.

- •Property: mainAxisAlignment: Controls how the children are aligned within the column.

9.Text widget: Displays text content.

- •Properties: style: Defines the appearance of the text, including font size, color, etc.

How the App Works:

The following breakdown explains how each component contributes to the overall structure and functionality of the simple greeting message app:-

1. The main() function starts the app by calling runApp(const MyApp()).
2. The MyApp widget builds the app's structure using the MaterialApp widget.
3. The MaterialApp's home property sets the MyHomePage as the initial widget.
4. The MyHomePage widget builds the home screen using a Scaffold widget.
5. The Scaffold's appBar displays the app's title.
6. The Scaffold's body contains a Center widget to center the greeting message.
7. The Center widget contains a Column to vertically arrange the greeting text.
8. The Text widget displays the greeting message "Hello, Flutter!" with a font size of 24.

Widgets and the Widget Tree

Widget Tree: Flutter applications are built using a nested tree of widgets, where the root widget is passed to the runApp() function.

•The widget tree determines the structure and layout of the UI.

•This example demonstrates how widgets can be nested within each other.

•The Column widget contains multiple children widgets such as Text and ElevatedButton.

•Container adds padding and background color, enhancing the appearance.

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Nested Widget Tree'),
      ),
      body: Center(
        child: Container(
          padding: EdgeInsets.all(16.0),
          color: Colors.blue[50],
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              Text('This is a nested widget.'),
              Text('It has multiple levels of widgets.'),
              ElevatedButton(
                onPressed: () {},
                child: Text('Click Me'),
              ),
            ],
          ),
        ),
      ),
    ),
  );
}
```

Building Interactive UI with Hot Reload

Hot Reload vs. Hot Restart:

Hot Reload: Instantly applies code changes to the app without a full reload. Ideal for UI adjustments and debugging during development.

Hot Restart: Restarts the entire app, clearing the current state but incorporating all code updates.

Example: Adding Interactivity:

- •Modify the earlier MyFirstApp to make the button interactive:
- •This app demonstrates a state change. When the button is pressed, the updateMessage function updates the message variable, triggering a UI rebuild.
- •setState() is used to notify Flutter of the state change.

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyInteractiveApp());
}

class MyInteractiveApp extends StatefulWidget {
  @override
  _MyInteractiveAppState createState() =>
_MyInteractiveAppState();
}

class _MyInteractiveAppState extends State<MyInteractiveApp> {
  String message = 'Welcome to Flutter!';
  int counter = 0;

  void updateMessage() {
    setState(() {
      message = 'Button Pressed!';
      counter++;
    });
  }
```

```dart
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Interactive UI'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(message),
              Text("$counter"),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: updateMessage,
                child: Text('Press Me'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Widgets in Flutter

- In Flutter, a widget is the fundamental building block of the user interface (UI).
- Everything displayed on the screen, including buttons, padding, rows, and complex layouts, is a widget.

Types of Widgets:

- **StatelessWidget:** A widget that does not require any mutable state. Its configuration and appearance are immutable after being created.
- **StatefulWidget:** A widget that can hold and change its internal state over time. It is used for interactive elements that can change dynamically.

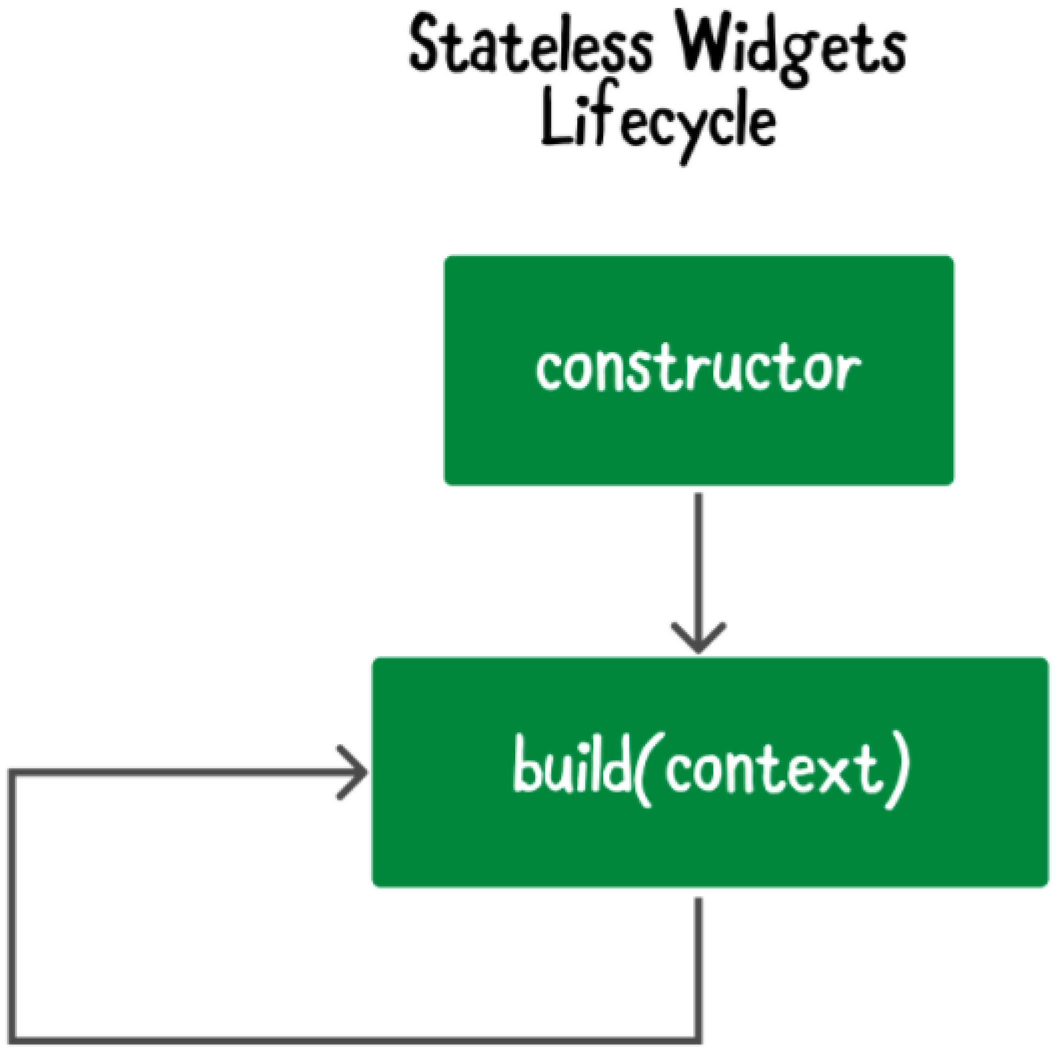
StatelessWidget

StatelessWidget is a widget that describes part of the UI by building a tree of other widgets that describes the user interface more concretely. It does not change over time and cannot hold any state that affects its appearance.

•**Structure:** StatelessWidget must override the build method to describe the part of the user interface represented by this widget.

Use Cases for StatelessWidget:

- Displaying static content like labels, icons, or decorative elements.
- Layout containers that don't require user interaction, such as Container, Padding, and Column.


Stateless Widgets Lifecycle
constructor
build(context)

Example

MyStatelessApp and MyCustomWidget are stateless widgets.

•The Text widget inside MyCustomWidget is immutable, and its state does not change during the lifecycle of the widget.

```
class MyStatelessApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
title: Text('Stateless Widget Example'),   ),
        body: Center(
          child: MyCustomWidget(),
        ),
),    );  }}

class MyCustomWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text(  'Hello, I am a Stateless Widget!',
      style: TextStyle(fontSize: 24),
);  }}
```

StatefulWidget in Detail

- **Definition:** A StatefulWidget is a widget that has a mutable state. It can change its internal state and re-render parts of its UI when state changes occur.
- **Structure:** A StatefulWidget consists of two classes:
  1. **StatefulWidget Class:** This is immutable and can be recreated if the parent widget tree changes.
  2. **State Class:** Contains the mutable state for the widget and the build method that describes the part of the user interface.

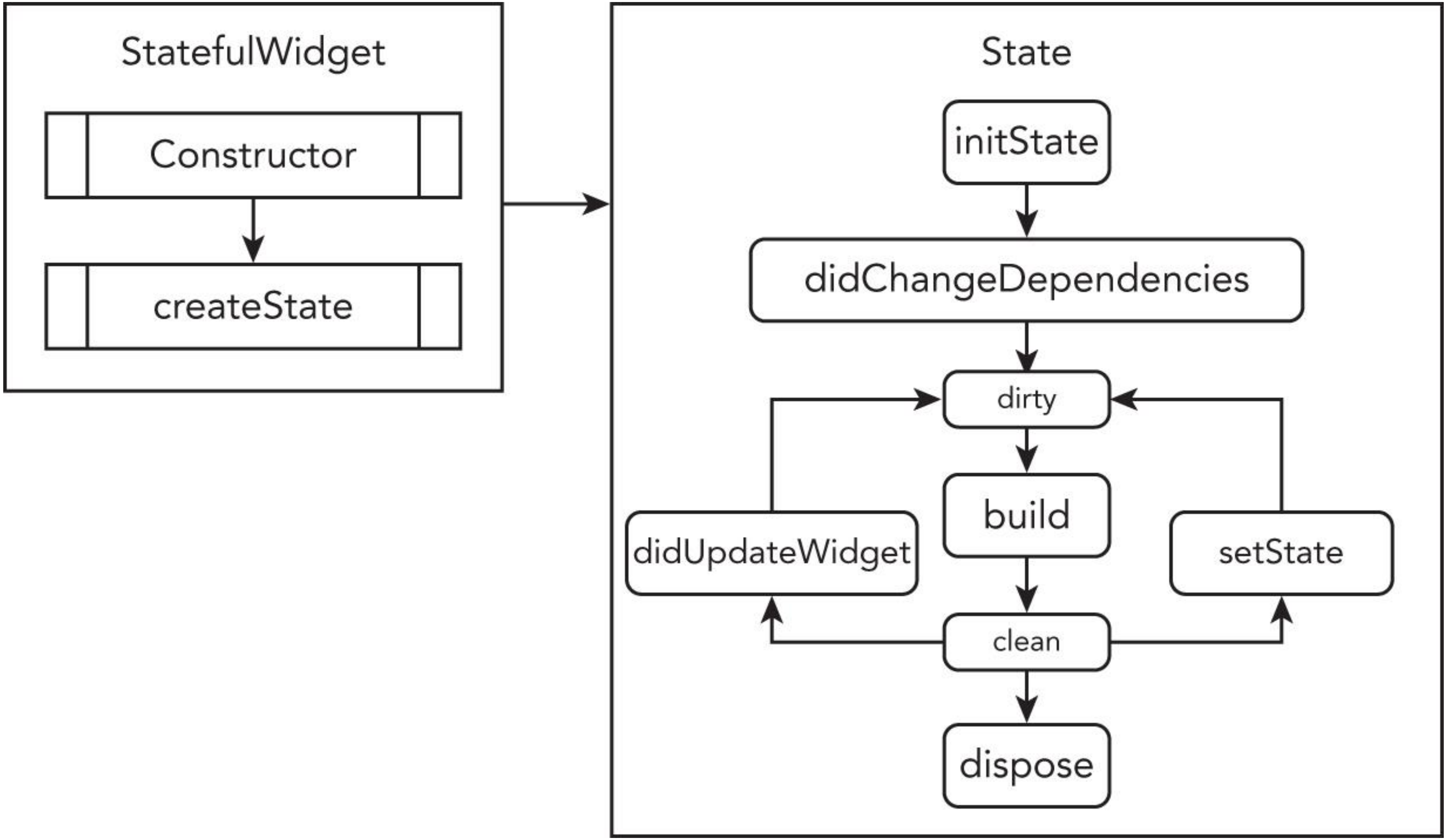
- **Use Cases for StatefulWidget:**
  - Forms where user input needs to be stored and updated dynamically.
  - Apps that require real-time updates, such as counters, timers, and interactive elements.

Key Differences Between StatelessWidget and StatefulWidget

Feature	StatelessWidget	StatefulWidget
State	Immutable; cannot change.	Mutable; can be changed during widget's lifecycle.
UI Updates	Does not change; static UI.	Can re-render when state changes occur.
Lifecycle Methods	Only build method.	Multiple lifecycle methods: initState, build, dispose, etc.
Use Cases	Static UI elements like text, icons, images.	Dynamic elements like forms, animations, and interactive widgets.

Lifecycle of StatefulWidget

- **Key Lifecycle Methods:**
    - **initState():** Called when the widget is created for the first time. Used for initializing data.
    - **build():** Called whenever the state changes or the widget is first built.
    - **setState():** Used to update the state and trigger a UI rebuild.
    - **dispose():** Called when the widget is removed from the widget tree. Used for cleanup tasks.


StatefulWidget
Constructor
createState
State
initState
didChangeDependencies
dirty
build
didUpdateWidget
setState
clean
dispose

```
class _MyLifecycleAppState extends State<MyLifecycleApp> {
  @override
  void initState() {
    super.initState();
    print("initState called");
  }

  @override
  void dispose() {
    print("dispose called");
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    print("build called");
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(          title: Text('Stateful Widget Lifecycle'),        ),
        body: Center(          child: Text('Check the console for lifecycle methods.'),        ),
      ),
    );
  }
}
```

Practical Implementation: Building a Simple To-Do App

- Create a StatefulWidget for Managing To-Do List

```
class _TodoAppState extends State<TodoApp> {
  final List<String> _todos = [];

  void _addTodoItem(String task) {
    if (task.isNotEmpty) {
      setState(() {
        _todos.add(task);
      });
    }
  }
```

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Simple To-Do App'),
      ),
      body: Column(
        children: <Widget>[
          TextField(
            onSubmitted: _addTodoItem,
            decoration: InputDecoration(
              labelText: 'Enter a new task',
            ),
          ),
          Expanded(
            child: ListView.builder(
              itemCount: _todos.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(_todos[index]),
                );
              },
            ),          ),          ],          ),       ),    ); }}
```

Add Functionality to Remove To-Do Items

•Added a delete button to each to-do item.

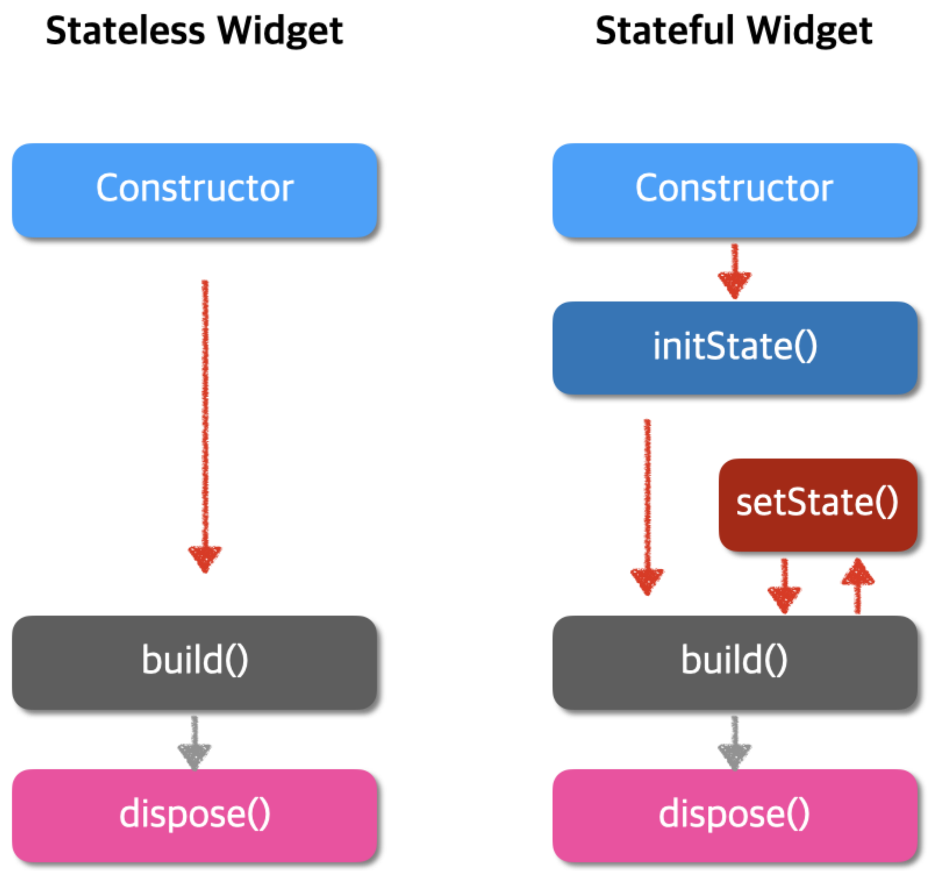
•The _removeTodoItem function removes the selected item from the list.

```
class _TodoAppState extends State<TodoApp> {
  final List<String> _todos = [];

  void _addTodoItem(String task) {
    if (task.isNotEmpty) {
      setState(() {
        _todos.add(task);
      });
    }
  }

  void _removeTodoItem(int index) {
    setState(() {
      _todos.removeAt(index);
    });
  }
```

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
            .
            .
            .
          Expanded(
            child: ListView.builder(
              itemCount: _todos.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(_todos[index]),
                  //subtitle: Text("$index"),
                  trailing: IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () => _removeTodoItem(index),
                  ),
                );
              },
            ),
          ),          ],        ),      ),    );  }}
```


Stateless Widget
Stateful Widget
Constructor
build()
dispose()
Constructor
initState()
setState()
build()
dispose()