Reg No.: 223018552
Module : Mobile application systems & Design
ASSIGNMENT I | Topic: State Management in Flutter

① Explanation of Flutter State Managements Methods

A. Provider: Uses Flutter's InheritedWidget internally to share and manage state efficiently across the widget tree.

It allows widgets to listen to state changes and rebuild automatically when the state changes.

B. Riverpod: Is an improved and safer version of Provider. It was created by the same developer but fixes many limitations of Provider.

Riverpod does not depend on the widget tree, which makes it more flexible, testable, and reliable.

C. Bloc (Business logic Component): is more advanced state management solution that separates busic logic from UI using streams and events.

D. GetX: It is a powerful and lightweight state management solution. It also provides routing, dependency injection, and state management in one package.

GetX is known for fast development and minimal boilerplate code.

Key features:
- Very fast development
- Less code required
- High performance
- Easy navigation and dependency management

② Table : When Each State Management is Applicable

Situation	Provider	Riverpod	Bloc	GetX
* small Applications	Excellent choic	Good choice	Not recommended	Excellent choice
* Medium Applications	Very good	Excellent	Good	Excellent
* Large / Enterprise applications	Limited	Excellent	Best choice	Good
* Team projects	Good	Excellent	Best choice	~~Y~~Moderate
* Fast development	Good	Good	slow	Best choice
* Strict Architecture requirement	Moderate	Excellent	Best choice	Weak

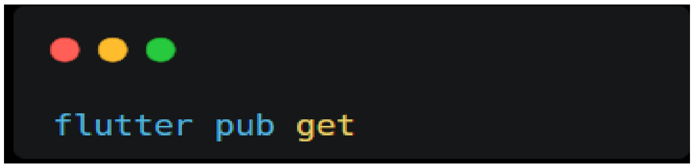
Question 3

1. Adding Dependency

Add Provider in pubspec.yaml:

Dependencies:
 provider: ^6.1.2

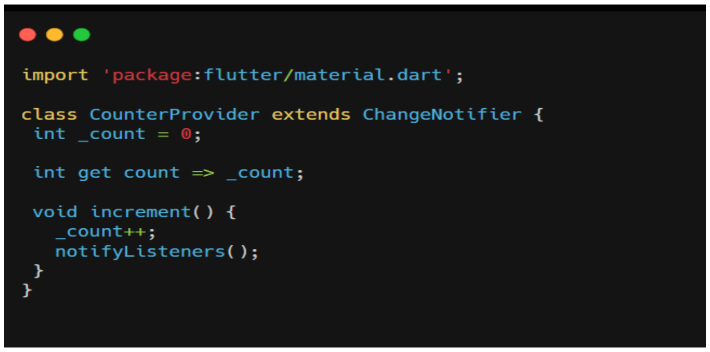
Run:



```
flutter pub get
```

Explanation:
 This adds the Provider package to the project. Provider is a state management solution that helps manage and share data across the application efficiently.

2. Creating a State Class



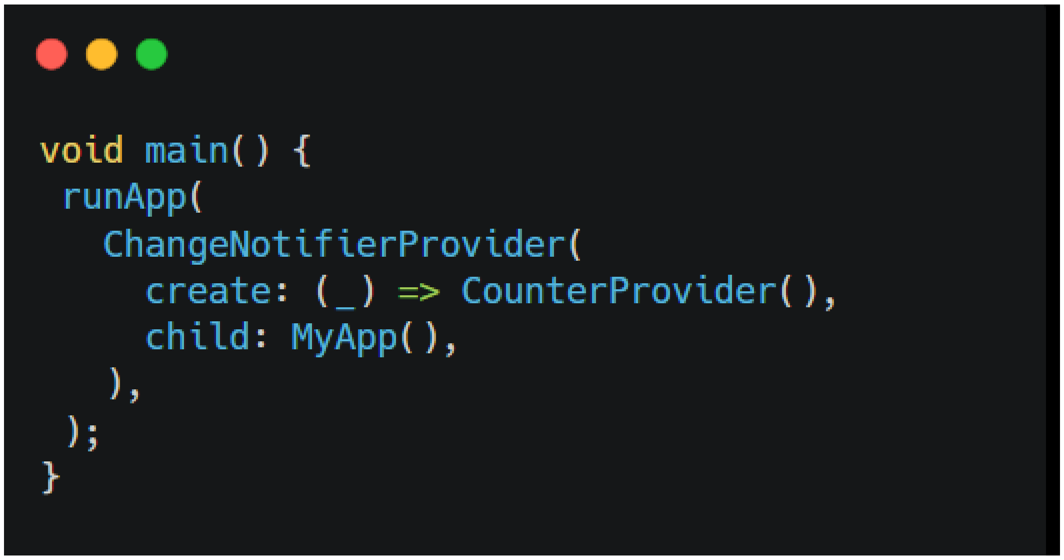
```
import 'package:flutter/material.dart';

class CounterProvider extends ChangeNotifier {
 int _count = 0;

 int get count => _count;

 void increment() {
   _count++;
   notifyListeners();
 }
}
```

Explanation:

- The class extends ChangeNotifier, which allows it to notify widgets when data changes.
- _count is a private variable that stores the state.
- count is a getter to access the value safely.
- increment() modifies the state.
- notifyListeners() informs all listening widgets to rebuild.

This class separates business logic from UI.

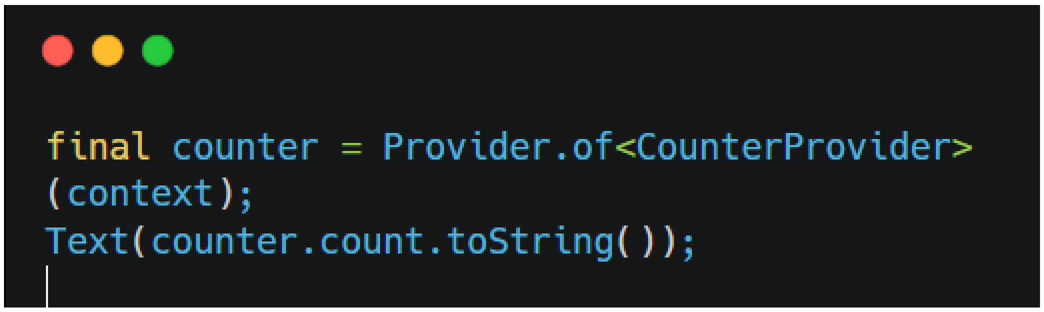
3. Providing the State



```dart
void main() {
 runApp(
   ChangeNotifierProvider(
     create: (_) => CounterProvider(),
     child: MyApp(),
   ),
 );
}
```

Explanation:
ChangeNotifierProvider provides the state to the widget tree.
By wrapping the app, all child widgets can access the state without passing data manually.

4. Accessing the State

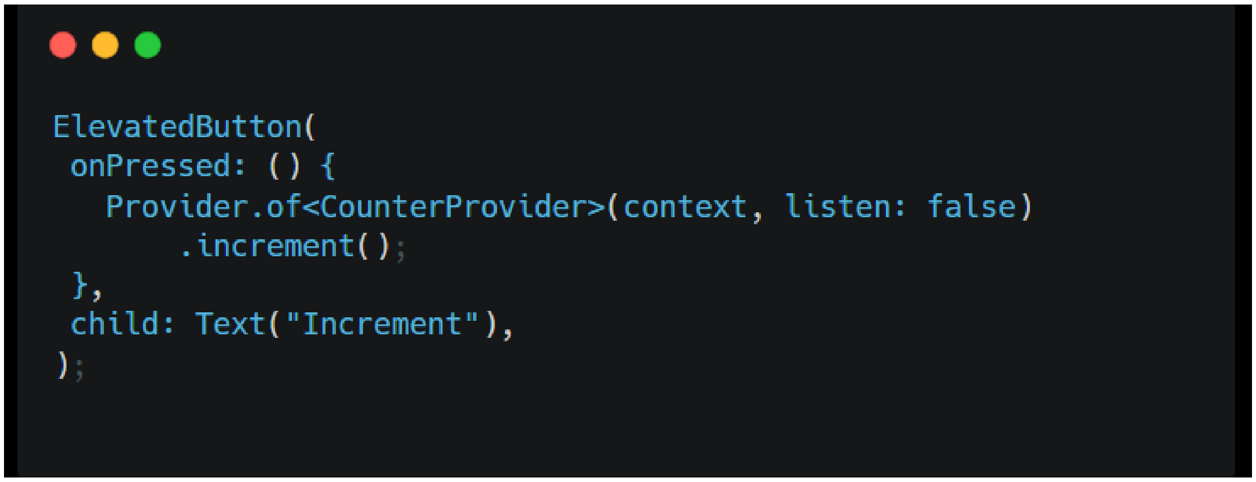


```dart
final counter = Provider.of<CounterProvider>
(context);
Text(counter.count.toString());
```

Explanation:
Provider.of retrieves the current state from the provider.
When the state changes, the widget automatically rebuilds if it is listening.

You can also use Consumer<CounterProvider> for better performance control.

5. Updating the State



```
ElevatedButton(
 onPressed: () {
   Provider.of<CounterProvider>(context, listen: false)
       .increment();
 },
 child: Text("Increment"),
);
```

Explanation:
Calling increment() updates the state variable.
listen: false prevents unnecessary rebuild when only calling a method.

6. How UI Rebuild Happens

Explanation:
When increment() changes _count, notifyListeners() is triggered.
All widgets that are listening to CounterProvider rebuild automatically.
This makes the Provider reactive and ensures the UI always reflects the latest data.

Conclusion

Provider simplifies state management in Flutter by:

- Separating logic from UI
- Reducing boilerplate code
- Automatically rebuilding widgets when data changes

It is suitable for small to medium applications and easy to maintain.