StateMangment Bloc & Cubit Part 2

- What is `BlocListener`?

It's **not** for building widgets — it's for **reacting** to state changes.

You use `BlocListener` when:

- You want to do **something once** when a state changes
  (like a toast, dialog, or navigation)=> **Side effect**
- You don't want the UI to rebuild for this action

```
BlocListener<MyCubit, MyState>(

  listener: (context, state) {
    // Side effect here
    if (state is MySpecialState) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Special state reached!')),
      );
    }
  },
  child: MyUIWidget(),
)
```

What is BlocConsumer?

It's a **combination** of `BlocBuilder` and `BlocListener` in a single widget.
That means it can **build UI** and **react to state changes** (side effects) in the same place.

Instead of writing `BlocBuilder` **+** `BlocListener` separately, you can just use `BlocConsumer`.

```
BlocConsumer<MyCubit, MyState>(

  listener: (context, state) {
    // Side effect here
    if (state is MySpecialState) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Special state reached!')),
      );
    }
  },
  builder: (context, state) {
    // Build the UI

    return CircularProgressIndicator();
  },
)
```

What is MultiBlocProvider?

When you have **more than one Cubit** or Bloc in the same part of your app, `MultiBlocProvider` allows you to provide multiple instances to the widget tree efficiently. This prevents deeply nested `BlocProvider` widgets.

```
MultiBlocProvider(
  providers: [
    BlocProvider<CounterCubit>(
      create: (context) => CounterCubit(),
    ),
    BlocProvider<ThemeCubit>(
      create: (context) => ThemeCubit(),
    ),
    // Add more BlocProviders as needed
  ],
  child: MyApp(), // Your application's root widget
)
```

a synchronous programming

what? :=> functions can run **without waiting** for other tasks to finish.

ex) making a cup of coffee ☕

→ so the app can keep working while waiting for **slow operations** (like network requests, file reads, or database queries).

Why use it?

1. **Better performance** – The app can do other work instead of sitting idle.
2. **Efficient resource usage** – No CPU time wasted waiting for slow tasks.
3. **Essential for I/O-heavy tasks** – Perfect for APIs, databases, and files.

What is the Event Loop in Dart?

1)runs synchronous code **Normal Code** (fast operations)

2)if Async ? => add to queue (wating list untill its done)

```
print("A");
Future.delayed(Duration(seconds: 2),
() {
  print("B");
});
print("C");
```

Asynchronous Programming Concepts

Future

Represents a value that will be available at some point in the future.

async

A keyword used before a function to mark it as asynchronous, allowing it to contain `await` expressions.

await

Used inside an `async` function to pause execution until a `Future` completes and returns its value.

Bad example

```
void main()
{
  print("start");
  getData();
  print("end");
}

Future<void>getData()
{
  return Future.delayed(Duration(seconds: 2),() {
    print('data loaded');
  },);
}
```

Good example

```
void main()async
{
  print("start");
 await getData();
  print("end");
}

Future<void>getData ()async
{
  await Future.delayed(Duration(seconds: 2),() {
    print('data loaded');
  },);
}
```

Task:

"برنامج عمل شاي" ☕

- اطبع: `"بدينا التحضير"`
- انتظر 2 ثواني (تحضير المياه) → اطبع `"سخّنا الميه"`.
- انتظر 1 ثانية (إضافة الشاي) → اطبع `"حطينا الشاي في الكوب"`.
- انتظر 3 ثواني (الشاي) → اطبع `"الشاي جاهز"`.
- اطبع `"خلصنا"`.

output:

```
بدينا التحضير
سخّنا الميه
حطينا الشاي
الشاي جاهز
خلصنا
```

Hint:

```
Future boilWater() async { // TODO: حط تأخير 2 ثانية وطباعة }

Future addTea() async { // TODO: حط تأخير 1 ثانية وطباعة }

Future waitForTea() async { // TODO: حط تأخير 3 ثواني وطباعة }

void main() async { print("بدينا التحضير");

// TODO: استدعي الدوال بالترتيب باستخدام await

print("خلصنا"); }
```

solution:

```
Future boilWater() async
{
await Future.delayed(Duration(seconds: 2)); print("سخّنا الميه");
}
Future addTea() async
{
await Future.delayed(Duration(seconds: 1)); print("حطينا الشاي في الكوب");
}
Future waitForTea() async
{
await Future.delayed(Duration(seconds: 3)); print("الشاي جاهز");
}
void main() async
{
print("بدينا التحضير");
await boilWater(); await addTea(); await waitForTea();
print("خلصنا");
}
```