Block Pattern (Business Logic Component)

→ Popüler bir state management mimarisidir.

→ reactive programming ve stream'ler (akışlar) ile çalışır.

Event → eylem veya olaylar event olarak adlandırılır. Event'ler kullanıcı etkileşimleri (butona tıklama, form doldurma gibi) veya uygulamanın içsel durum değişiklikleri olabilir. Her bir event, Bloc tarafından dinlenir ve bu event'e göre aksiyon gerçekleştirilir.

→ Soyut bir türev olay sınıfı tüm counter olayları bu sınıftan türeyecek. Bu sayede tüm olaylar artık bir tür altında toplanabilir.

```
abstract class CounterEvent {}
class IncrementEvent extends CounterEvent {}
class DecrementEvent extends CounterEvent {}
```

extend'da anlıyoruz ki bu olayın sayıcı ile ilgili bir olay olduğunu

Bu örnekte CounterEvent sınıfı bir soyut sınıf olup artırım (IncrementEvent) ve azaltım (DecrementEvent) olaylarını temsil eden iki event türetilmiştir.

State → State, uygulamanın belli bir anda sahip olduğu bilgiyi ifade eder.

→ UI bileşenleri, Bloc tarafından sağlanan state bilgilerini kullanarak ekranı günceller. Her bir state, Bloc tarafından yönetilir ve Streamler aracılığıyla UI'ya gönderilir.

```
abstract class CounterState {}
class CounterInitial extends CounterState {
  final int counterValue;
  CounterInitial(this.counterValue);
}
class CounterUpdated extends CounterState {
  final int counterValue;
  CounterUpdated(this.counterValue);
}
```

→ sayacın başlangıç durumunu temsil eden sınıf

→ sayacın mevcut değerini tutar

→ constructer

→ sayacın güncellenmiş halini temsil eden sınıf

→ kullanıcı arttırma/azaltma yaptığında bu duruma geçilir

→ güncel sayaç değerini tutar

→ constructer

CounterState sınıfı, sayacın ilk durumunu (CounterInitial) ve güncellenmiş sayacı (CounterUpdated) temsil eder

Bloc Class (İş Mantığı Bileşeni) → Bloc sınıfı, eventleri alıp, iş mantığını uygulayarak bunlara göre durumu değiştiren temel bileşendir.

→ Event'ler streamler üzerinden dinlenir ve işlenir, sonuçta yeni bir state üretilir ve bu yeni state de stream üzerinden UI'e iletilir.

```
import 'package:bloc/bloc.dart';
class CounterBloc extends Bloc<CounterEvent, CounterState> {
  CounterBloc() : super(CounterInitial(0));
  @override
  Stream<CounterState> mapEventToState(CounterEvent event) async* {
    if (event is IncrementEvent) {
      yield CounterUpdated(state.counterValue + 1);
    } else if (event is DecrementEvent) {
      yield CounterUpdated(state.counterValue - 1);
    }
  }
}
```

→ Bloc kütüphanesini import ediyoruz

→ CounterBloc sınıfı, bundan türer

constructer

CounterEvent → Bloc'un işleyeceği olay türü

CounterState → Bloc'un üreteceği durum türü

→ gelen olayları (events) state'lere dönüştürür.

→ yeni bir CounterUpdated durumu oluşturuyoruz

→ yield keyword'ü ile bu yeni durumu yayınlıyoruz (emit)

• async* ve yield kullanımı Bloc'un durumları bir stream üzerinden yayınlayabilmesini sağlar

CounterBloc sınıfı, eventleri alır ve her bir event'e göre durumları günceller. Örneğin

Stream ve Sink → Bloc Pattern'de veri akışı Stream ve Sink kavramları ile

Stream → Statelerin UI'ya iletildiği akıştır. UI bileşenleri bu stream'leri dinler yönetilir ve gelen state'e göre kendini günceller.

Sink → Eventlerin Bloc'a gönderildiği veri girişidir. Kullanıcı etkileşimleri sonucu oluşan event'ler, Sink'e gönderilerek işlenir.

Bloc, bu iki yapı arasında köprü görevi görerek veri akışını sağlar.

StreamBuilder → UI kısmında, Bloc tarafından yayınlanan state'leri dinlemek için StreamBuilder kullanılır. Bu widget bir stream'i dinleyerek gelen state'e göre UI'ı yeniden oluşturur.

```
class CounterScreen extends StatelessWidget {
@override
Widget build(BuildContext context) {
  final counterBloc = BlocProvider.of<CounterBloc>(context);
  return Scaffold(
    appBar: AppBar(title: Text('counter')),
    body: Center(
      child: StreamBuilder<CounterState>(
        stream: counterBloc.stream,
        initialData: counterBloc.state,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            final counterValue = snapshot.data.counterValue;
            return Text('$counterValue', style: TextStyle(fontSize: 24));
          } else {
            return CircularProgressIndicator();
          }
        },
      ),
    ),
    floatingActionButton: Column(
      mainAxisAlignment: MainAxisAlignment.end
      children<widget>[
        FloatingActionButton(
          onPressed: () => counterBloc.add(IncrementEvent()),
          child: Icon(Icons.add),
        ),
        FloatingActionButton(
          onPressed: () => counterBloc.add(DecrementEvent()),
          child: Icon(Icons.remove)
        ),
```

BlocProvider üzerinden counterBloc örneğini alıyoruz. Bu, widget ağacının üstünde bir BlocProvider olmasını gerektirir

ile mevcut Bloc'a erişim

StreamBuilder widget'ı Bloc'tan gelen durum değişikliklerini dinler

dinlenecek stream

Başlangıç durumu

veri geldiyse (durum güncellendiyse)

Anlık counter değerini al

Büyük fontta counter değerini göster

Veri yoksa yükleme göstergesi göster

Sağ altta yer alacak butonlar

Bu örnekte, StreamBuilder aracılığıyla Bloc tarafından yayınlanan state dinlenir ve UI buna göre güncellenir. Kullanıcı artı butonuna tıkladığında IncrementEvent, eksi butonuna tıkladığında ise DecrementEvent tetiklenir

IncrementEvent gönder

DecrementEvent gönder