07

Bloc

상태관리

Bloc 패턴

- Blocbusiness logic component는 UI와 비지니스 로직을 분리하는 디자인 패턴이자 상태 관리 프레임워크
- Bloc 패턴은 화면을 제공하는 프런트엔드 애플리케이션을 개발할 때 주로 사용

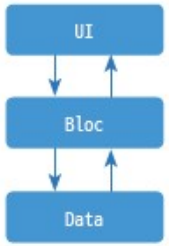


그림 20-1 Bloc 패턴

Bloc 구성 요소

• 패키지 등록하기

```
dependencies:
  bloc: ^8.0.2
  flutter_bloc: ^8.0.1
```

이벤트 주입

- 이벤트event는 Bloc의 입력 요소
- 벤트는 Bloc를 동작하게 하는 입력 요소

Bloc 구성 요소

이벤트 주입

- Bloc에서 이벤트를 작성할 때는 특별한 규칙이 없습니다.
- 열거형 상수로 표현하거나 개발자가 만드는 클래스로 작성

• 이벤트를 열거형 상수로 선언

```
enum CounterEvent { increment, decrement }
```

• 이벤트를 클래스로 선언

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

• 값을 포함하는 클래스 이벤트 선언

```
abstract class CounterEvent {
  int no;
  CounterEvent(this.no);
}
class IncrementEvent extends CounterEvent {
  IncrementEvent(int no): super(no);
}
class DecrementEvent extends CounterEvent {
  DecrementEvent(int no): super(no);
}
```

Bloc 구성 요소

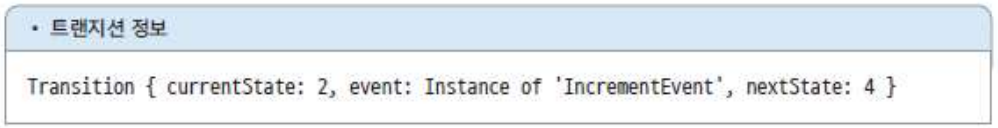
상태 출력

- 상태state는 Bloc의 출력 요소
- 이벤트에 의해 실행된 업무 로직의 결과이며 앱 내의 여러 위젯에서 이용하는 상태 데이터

Bloc 구성 요소

트랜지션 정보

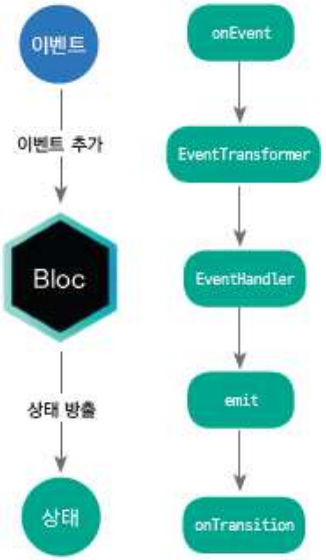
- 트랜지션transitions은 정보 요소
- Bloc에 이벤트가 발생하고 업무 로직이 실행되어 상태 데이터가 발생하거나 변경됩니다. 이런 일련의 흐름에 관한 정보를 트랜지션이라 합니다.
- 트랜지션 정보는 이벤트에 따라 Bloc에서 자동으로 발생하는 정보



Bloc 구성 요소

Bloc 클래스

- Bloc를 상속받아 작성하는 클래스
  - •위젯에서 발생하는 이벤트를 받는 클래스
  - •이벤트에 따른 적절한 업무 로직을 실행하는 클래스
  - •업무 로직 실행 결과를 앱의 상태로 유지하고 위젯에서 이용할 수 있게 제공하는 클래스



Bloc 구성 요소

Bloc 클래스

- Bloc 클래스 선언하기
  - Bloc는 Bloc를 상속받아 작성하는 개발자 클래스
  - 제네릭 정보로 Bloc에 발생하는 이벤트 타입과 Bloc에서 유지하는 상태의 타입을 명시

• Bloc 클래스 선언하기

```
class BlocCounter extends Bloc<CounterEvent, int> {
  BlocCounter() : super(0) {
    ... (생략) ...
  }
}
```

Bloc 구성 요소

Bloc 클래스

- 이벤트 등록하기
  - 감지할 이벤트는 생성자에서 on() 함수로 등록

• 감지할 이벤트 등록하기

```
BlocCounter() : super(0) {

  on<IncrementEvent>((event, emit) {
    ... (생략) ...
    emit(state + event.no);
  });
  on<DecrementEvent>((event, emit) {
    ... (생략) ...
    emit(state - event.no);
  });
}
```

Bloc 구성 요소

Bloc 클래스

- 이벤트가 발생할 때 적절한 업무 로직을 실행하고 그 결과 데이터를 상탯값으로 이용하려면, 두 번째 매개변수로 전달된 함수의 매개변수로 상태 데이터를 전달

• 열거형 상수로 선언한 이벤트 등록하기

```
BlocCounter() : super(0) {
  on<CounterEvent>((event, emit) {
    switch(event) {
      case CounterEvent.increment:
        emit(state + 1);
        break;
      case CounterEvent.decrement:
        emit(state - 1);
        break;
    }
  });
}
```

Bloc 구성 요소

Bloc 클래스

- onEvent() 함수 재정의하기
  - Bloc 클래스에 onError(), onTransition(), onEvent() 함수를 재정의할 수 있습니다.
  - onEvent() 함수는 이벤트가 발생할 때마다 자동으로 호출
  - onEvent()의 매개변수로 발생한 이벤트 정보가 전달

• onEvent() 함수 재정의하기

```
@override
void onEvent(CounterEvent event) {
  super.onEvent(event);
  ... (생략) ...
}
```

Bloc 구성 요소

Bloc 클래스

- onTransition() 함수 재정의하기
  - 이벤트 발생으로 상탯값이 어떻게 변경되었는지는 Bloc 내부에서 트랜지션 정보로 발생
  - 트랜지션 정보를 활용하고 싶다면 onTransition() 함수를 재정의

• onTransition() 함수 재정의하기

```
@override
void onTransition(Transition<CounterEvent, int> transition) {
  super.onTransition(transition);
  print('transition.... $transition');
}
```

▶ 실행 결과

```
transition.... Transition { currentState: 0, event: Instance of 'IncrementEvent',
nextState: 2 }
```

Bloc 구성 요소

Bloc 클래스

- onError() 함수 재정의하기
  - onError() 함수는 이벤트 발생으로 특정 업무를 처리하다 오류가 발생할 때 자동으로 호출

• onError() 함수 재정의하기

```
@override
void onError(Object error, StackTrace stackTrace) {
  print('error..... $error, $stackTrace');
  super.onError(error, stackTrace);
}
```

Bloc 구성 요소

Bloc 프로바이더

- Bloc 클래스를 정의했으면 위젯에서 이용하도록 등록
- BlocProvider는 위젯이므로 Bloc를 이용할 위젯의 상위 위젯으로 선언
- create 속성의 함수에서 반환한 Bloc 객체를 child에 명시한 위젯부터 그 하위 위젯에서 이용

• Bloc 프로바이더로 등록하기

```
BlocProvider<BlocCounter>(
  create: (context) => BlocCounter(),
  child: MyWidget(),
)
```

Bloc 구성 요소

Bloc 프로바이더

- Bloc의 상탯값을 이용하려면 state 속성을 이용
- 이벤트를 주입할 때는 add() 함수를 이용

• Bloc 객체 얻기

```
final BlocCounter counterBloc = BlocProvider.of<BlocCounter>(context);
```

• 상탯값 사용하기

```
Text('${counterBloc.state}'),
```

• 이벤트 주입하기

```
ElevatedButton(
  child: Text('increment'),
  onPressed: () {
    counterBloc.add(IncrementEvent(2));
  },
),
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare