Jetpack Compose에서 State 이해하기

State의 개념 / 상태 관리 API / Recomposition과의 관계

다이스

State란?

UI에서 동적으로 변하는 데이터를 의미

Compose에서는 상태 변경 시 UI를 자동으로 업데이트 (Recomposition)

사용 예시

- 사용자의 입력
- 오류 메시지
- 애니메이션 상태
- 등등

State와 Composition

Composition 생명주기

1. 초기 Composition -> 컴포저블 최초 렌더링
2. Recomposition -> 상태 변경 시 발생, 변경된 UI만 렌더링

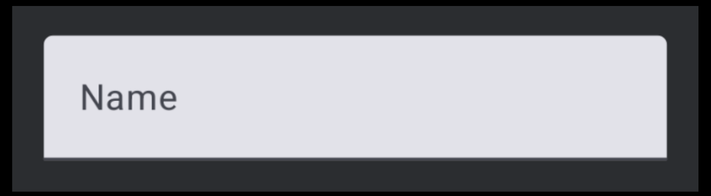
Compose Runtime에 자동으로 상태를 추적

ViewSystem의 View.invalidate()와 유사 동작

상태 예제 코드

MainActivity.kt

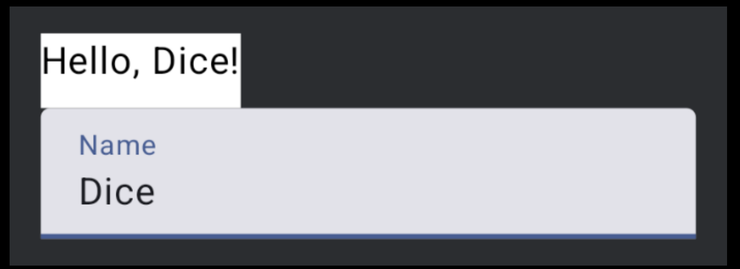
```kotlin
@Composable
fun HelloContent(modifier: Modifier = Modifier) {
    Column(modifier = modifier.padding(16.dp)) {
        var name by rememberSaveable { mutableStateOf( value: "")}

        if (name.isNotEmpty()) {
            Text(
                text = "Hello, $name!",
                modifier = Modifier.padding(bottom = 8.dp),
            )
        }

        TextField(
            value = name,
            onValueChange = { name = it },
            label = { Text( text: "Name") }
        )
    }
}
```



사용자가 “Dice” 입력

state가 “Dice”로 변경

Recomposition



상태 관리 API

mutableStateOf

- 상태값 변경 시 Recomposition 트리거

- 관찰 가능한 상태 객체 생성

MainActivity.kt

```
val count = mutableStateOf( value: 0)
```

상태 관리 API

remember

- 초기 Composition에서 상태를 메모리에 저장
- Recomposition 시 기존 값 사용

MainActivity.kt

```
var count by remember { mutableStateOf( value: 0) }
```

상태 관리 API

rememberSaveable

- 화면 회전 등 구성 변경 시에도 상태 유지

- Bundle 저장 가능한 타입 / Saver 사용

MainActivity.kt

```
var count by rememberSaveable { mutableStateOf( value: 0) }
```

remember + mutableStateOf

MainActivity.kt

```kotlin
// State<String> 타입
val mutableState: MutableState<String> = remember { mutableStateOf( value: "") }

// String 타입 (delegate 속성 사용)
val value: String by remember { mutableStateOf( value: "") }

// 구조 분해 선언 (value: String, setValue: (String) -> Unit)
val (value, setValue) = remember { mutableStateOf( value: "") }
```

왜 remember와 mutableStateOf가 함께 사용되야 할까?

MainActivity.kt

```kotlin
@Composable
fun CounterWithoutRemember() {
    var count = mutableStateOf( value: 0)

    Column(modifier = Modifier.padding(16.dp)) {
        Text(text = "Count: ${count.value}")

        Button(onClick = { count.value++ }) {
            Text( text: "Increment")
        }
    }
}
```

MainActivity.kt

```kotlin
@Composable
fun CounterWithoutRemember() {
    var count by remember { mutableIntStateOf( value: 0) }

    Column(modifier = Modifier.padding(16.dp)) {
        Text(text = "Count: $count")

        Button(onClick = { count++ }) {
            Text( text: "Increment")
        }
    }
}
```

Recomposition

MainActivity.kt

```kotlin
@Composable
fun MultipleStateExample(modifier: Modifier = Modifier) {
    var countA by remember { mutableIntStateOf( value: 0) }
    var countB by remember { mutableIntStateOf( value: 0) }

    Column(modifier = modifier) {
        CountAComposable(countA)
        CountBComposable(countB)

        Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
            Button(onClick = {
                countA++
                println("\nClick A Button")
            }) { Text( text: "Increment A") }

            Button(onClick = {
                countB++
                println("\nClick B Button")
            }) { Text( text: "Increment B") }
        }
    }
}
```

MainActivity.kt

```kotlin
@Composable
fun CountAComposable(count: Int) {
    println("[CountA] Composable recomposed with count=$count")
    Text(text = "Count A: $count")
}

@Composable
fun CountBComposable(count: Int) {
    println("[CountB] Composable recomposed with count=$count")
    Text(text = "Count B: $count")
}
```

```
[CountA] Composable recomposed with count=0
[CountB] Composable recomposed with count=0

Click A Button
[CountA] Composable recomposed with count=1

Click B Button
[CountB] Composable recomposed with count=1

Click A Button
[CountA] Composable recomposed with count=2
```