Jetpack Compose의 Recomposition Process 및 Key의 역할

Jetpack Compose?

- 선언적 UI 프레임 워크
- Kotlin을 사용하여 UI 구축
- 코드는 간결하고, 가독성이 높으며, 컴파일 타임에 오류를 잡아냄

Recomposition

- UI를 구성하는 데이터나 상태가 변경되었을 때 발생
  - 데이터나 상태가 변경되면 변경된 부분만 다시 그려 화면 업데이트

    → 기존 방식의 명령형 UI 프레임워크와 대조적으로 선언적인 방식으로 상태 변경에 대응하는 것을 의미

Recomposition Process

- *Recomposition의 과정*

(1) 상태(State) 변경

- UI를 구성하는 데이터나 상태가 변경됨

(2) Recomposition 트리 구축

- 변경된 상태에 기반하여 Compose는 Recomposition 트리 구축
  - 이 트리는 UI의 현재 상태와 이전 상태를 비교하는 데 사용됨

- 또한 Compose 내부에서 구현 되며, 상태가 변경된 부분을 식별하는데 사용됨

(3) Recomposition 실행

- 구축된 Recomposition 트리를 기반으로 Compose는 변경된 부분만을 다시 그려 UI 업데이트
  - 이때 효율성을 높이기 위해 Compose는 **key**를 활용하여 정확한 변경 부분 식별

(4) 자동화된 UI 업데이트

- Compose는 Recomposition을 수행함으로써 상태 변경에 따라 자동으로 UI 업데이트

  → 명시적으로 UI를 업데이트할 필요가 없음

▼ Recomposition Process

- 상태 변경이 감지되면 Compose가 해당 변경에 대응하여 UI를 효율적으로 업데이트 하는 과정을 의미

Key

- *Recomposition process를 최적화하고 UI 업데이트를 효율적으로 수행하기 위한 요소 중 하나*

특징

- **고유성 보장**
  - Key는 각 UI 요소를 식별하는데 사용되는 고유한 아이디를 가짐
  - 각 요소에는 유일한 Key가 있어야 함

    → 이는 Compose가 이전 상태와 현재 상태를 구분하고 변경된 부분을 정확하게 식별할 수 있도록 도와줌
- **재사용 가능한 상태 저장**
  - Key를 사용하면 Compose가 UI 요소의 이전 상태를 추적하고, 변경된 부분만을 업데이트하는데 사용됨

    → 상태를 재사용하여 효율적으로 UI를 그릴 수 있음
- **효율적인 업데이트**

- Key를 사용하면 Compose는 이전 상태와 현재 상태를 효율적으로 비교하여 변경된 부분만을 업데이트

  → 이를 통해 불필요한 UI 재생성을 방지하고 성능을 최적화할 수 있음

- **동적 데이터와의 효과적인 작업**
  - 동적으로 변하는 데이터에 대응하기 위해 Key를 사용하면 해당 데이터의 변경에 따라 정확하게 UI를 업데이트할 수 있음