03-3. Lifecycle Aware Components

Android Architecture Component

Lifecycle Aware Components

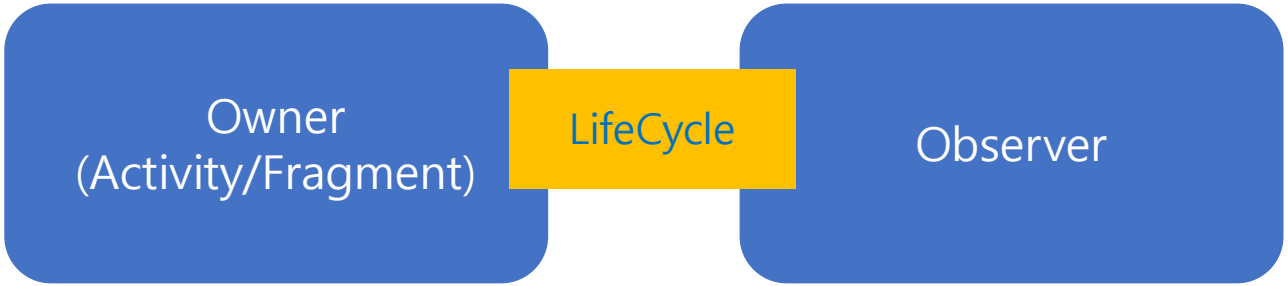
•Activity/ Fragment 의 Lifecycle 처리를 따로 구성
•Activity는 Lifecycle Owner 가 되며 이를 처리하는 곳이 Lifecycle Observer

•Activity 의 모든 라이프사이클 코드를 분리하겠다는 개념이 아님
•라이프사이클과 관련된 의미단위의 작업을 추상화 시켜서 작성

•Switching between coarse and fine-grained location updates
•Stopping and starting video buffering
•Starting and stopping network connectivity
•Pausing and resuming animated drawables

Lifecycle Aware Components

•Lifecycle : Activity나 Fragment의 라이프사이클 변경을 감지하였다가 Observer 에게 알려주는 역할의 클래스
•Observer : LifecycleObserver를 구현한 클래스, 라이프 사이클 변경 시 실행될 함수를 어노테이션으로 등록




```
implementation "androidx.lifecycle:lifecycle-runtime-ktx:2.3.1"
// optional - ProcessLifecycleOwner
implementation "androidx.lifecycle:lifecycle-process:2.3.1"
```

Lifecycle Observer

- Activity/ Fragment Lifecycle 변경 시 실행될 코드 등록
- DefaultLifecycleObserver 구현한 클래스
- Fragment 의 경우 onCreateView, onAttach 등의 Fragment 만을 위한 라이프사이클 감지는 안됨

```
class MyActivityLifecycleObserver: DefaultLifecycleObserver {
    override fun onStart(owner: LifecycleOwner) {
        super.onStart(owner)
    }

    override fun onStop(owner: LifecycleOwner) {
        super.onStop(owner)
    }
}
```

Lifecycle Owner

- Lifecycle 이 변경되는 클래스 자체를 지칭하며 일반적으로 Activity, Fragment
- LifecycleOwner 는 getLifecycle() 이라는 추상함수를 가지는 LifecycleOwner 인터페이스를 구현한 클래스

```
class MyActivity : Activity(), LifecycleOwner {
    lateinit var mLifecycleRegistry: LifecycleRegistry

    override fun getLifecycle(): Lifecycle {
        return mLifecycleRegistry
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        mLifecycleRegistry = LifecycleRegistry(this)
        mLifecycleRegistry.currentState = Lifecycle.State.CREATED

        lifecycle.addObserver(MyActivityLifecycleObserver())
    }

    public override fun onStart() {
        super.onStart()
        mLifecycleRegistry.currentState = Lifecycle.State.STARTED
    }
}
```

Lifecycle Owner

- Fragment, AppCompatActivity 는 이미 이 인터페이스가 구현된 클래스임
- lifecycle property에 의해 Lifecycle 객체가 리턴되며 이곳에 addObserver() 함수를 이용해 Observer 등록

```
lifecycle.addObserver(lifecycleObserver)
```

Process Lifecycle Owner

- 전체 어플리케이션이 resume 되거나 pause 되는 상황을 감지해야 하는 경우ProcessLifecycleOwner 를 이용
- ON_CREATE는 최초에 한번 호출되며 ON_DESTROY 는 호출되지 않음
- 어플리케이션의 모든 ACTIVITY가 화면을 점유하지 못하게 되면 ON_PAUSE, ON_STOP 이 호출되며 다시 화면에 나오게 될 때 ON_START, ON_RESUME 이 호출됨
- 일반적으로 Application 클래스에서 Observer를 등록하여 사용함

```
ProcessLifecycleOwner.get().lifecycle.addObserver(MyProcessLifecycleObserver())
```

LiveData

•LifecycleObserver 의 작업에 의한 데이터를 Activity 등에서 이용해야 하는 경우 Observer를 만들 때 LiveData를 상속받아 작성함

```
class MyActivityLifecycleObserver: LiveData<Int>(), DefaultLifecycleObserver {

}
```

```
val liveData = MyLifecycleObserver4()
liveData.observe(this, { result ->
    Log.d("kkang", "activity....$result")
})
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare