2022/2/26

mvvm에 대한개요

아키텍쳐는 왜씀?

협업을 위해서 어느 위치에 어떤 로직이 들어있나를 미리 정의해 놓아서

-> 비슷한 코드를 보고 알아보기 쉬워짐(가독성이 좋아짐)

-> 유지보수가 수월해짐

-> 코드들을 역할분리가 명확해짐(객체간의 관심사 분리)

-> 의존관계가 분리됨(변화에 유연함 새로운기능이나 코드가 좀바뀌어도 코드가 유지됨)

단점

특정 아키텍쳐로 개발을 시작하면 다른 아키텍쳐로 변환 거의 불가

->프로젝트를 다시만드는 수준임(모든 프로젝트에 같은 개념을 가지기에)

프로젝트에 아키텍쳐별 상성있어서 나중에 프로젝트가 안맞다고 깨닳아도 답이없음

->비효율적인 코드 생성

생산속도 엄청나게 느려짐

-> 기능 하나 만들때마다 파일 엄청 만들어야함 레어어마다 파일들을 만들어야해서 보일러플레이트 코드 양산됨

MVVM이 무엇인가?

우선 mvp가 선행되고 -> 단점을 경험하고 발전함을 느껴봐야함

태초에 액티비티에 모든걸 때려박는 코드가 있었다->스파게티코드

-> mvc나옴 근데 이거 어짜피 안드로이드에서는 스파게티와는 다를 바가없음

분리가 완벽하게 안됨 뷰에 모델이 박혀있어서

-> mvp가 그래서 나옴

뷰는 프리젠터에 일을 시키고 프리젠터는 모델에서 정보가져옴

프리젠터가 요청할 뿐이고 뷰는 멍청함

프리젠터가 가공해서 뷰에 넘겨줌

-> 뷰랑 모델을 완전히 분리하게됨

단점 프리젠터가 혼자 미쳐날뜀 그냥 프리젠터만 비대해짐

뷰랑 프리젠터에 양방향 의존성이생김 뷰랑 프리젠터랑 서로 알아야해서

1:1 관계됨 그래서 재활용이안됨

-> 그래서 MVVM이 나옴

뷰는 뷰모델을 알지만 뷰모델은 뷰가 뭔지 모름 어디서 쓰이는지

그걸 느슨하게 데이터 바인딩으로 묶어놓음

뷰에대한 상태같은것들을 뷰모델이 박아놓음

뷰는 뷰모델을 관찰하고있고 변경점이 생기면 뷰가 알아채서 스스로 갱신하는것이

MVVM이다

-> 사실 MVVM은 레이어가 하나 숨어져있다

바인더라는 레이어가 뷰랑 뷰모델 사이에 바인더가 숨어있다->안드로이드에서는 데이터바인딩으로 지원한다

-> 데이터바인딩을 안쓰면 MVVM이 아니다 그건 그냥 MVP다

ViewModel은 AAC의 뷰모델이 아니다 -> 당연한소리

AAC 없이도 뷰모델 구현가능해야함

-> 사실 MVVM은 마소에서 만든건데 어딜 감히 구글이

AAC뷰모델은 안드로이드 라이프사이클에 종속됨

-> configuration change 라는 안드로이드에 용어있음 -> 회전할때 파괴되었다가 다시 호출되는것

가로뷰 세로뷰 사실 다른것

뷰모델 애초에 액티비티 죽었다고 죽는건 좀 이상함

원래라면 액티비티가 죽었다고 뷰모델이 죽으면 안됨

MVVM의 최대 목적은 view 와 viewmodel을 분리시키는게 최대목표

그랩아져씨 말이 맞단다→ 결론 xml로 죄다 옮기자

이런코드 잘못된 코드라고 한다

액티비티 자체에서 뷰모델을 부름 이런 코드자체가 뷰모델이 박살남

->xml로 죄다 옮겨라

사실 좀 극단적임 이런코드 줄이는 방식으로 사용하자

-> 이걸 의식해서인지 MVVM 이라는 단어안씀

권장 아키텍쳐라 적혀있음

우리는 구글의 권장 아키텍쳐 를 사용해 볼것임

구글 아키텍쳐는 용어가 좀다름

뷰모델이랑 뷰는 presentation layer

modeld 을 나눠서 domain layer data layer 로 나눔

모델 레이어 나뉘는것

use case repo | data source 레트로핏,sql,room,relem

domain data

도메인: 프로젝트를 관통하는 핵심 로직들 (로또라면 로또번호 생성기같은거 )

-> 도메인 거의 다 서버에 들어있음

핵심 거의다 서버에서 함

클라 애초에 가져와서 뿌리는거밖에없음

-> 안드로이드 로컬 로또라면 그런 로직이 도메인에 들어가는거임 서버에서 하는 핵심 로직들

그리고 이거 나눈거 관심사 분리 때문임

유즈 케이스 -> 레파지토리한테 요청

레파지토리 -> 일기 받아오는데 인터넷이 연결되어있는지 서버에서 온건지 로컬에서 온건지 이런거

레파지토리에서 한층 가리는거임-> 데이터 소스를 소유함(local data source,remote data source)

순서 일단 로컬뒤지고 난후 없네 하면 리모트 뒤져서보내주는거

데이터 소스 -> 레파지토리는 어떤 원천에서 뽑아온건지 룸인지 뭐 쉐어드 프리퍼런스인지

이런거 몰라야하기 때문에 데이터 소스를 소유하는거고 데이터 소스가 이걸 가져오는 역할

다음 마지막 구현체 레트로핏 같은거

말리빈님 -> 데이터소스 impl안만듬 왜만드는지 모르겠다함

그냥 인터페이스만 만들면된다함 -> 그냥 인터페이스 일뿐임

상속받아서 다형성을 활용하면됨 구현체를 쓰는게 아니라

이거 어려움

어쨋든 딱히 impl 구현할 필요없단다.

개론끝!!

데이터 바인딩 시작

findviewbyid 부터 시작

이거 왜 안쓰는데?

-> 아이디의 중복,nullpoint,아이디 정신머리없음

그리고 이상한거 넣어도 컴파일 에러가 아니라 런타임 에러가남

더빡쳐 런타임 !!!! -> npe남 -> 코틀린이 감히? -> null안정적이지 못함

자료형이 안맞아도 컴파일됨 -> 런타임이 남 이거 빨간줄뜨는거 컴파일됨 근데 그냥 안스에서 경고해주는거임 -> 뷰타입에 대해서도 안전하지않음 type unsafe

-> 결론 type unsafe, null unsafe 함

이걸 보완한것

view binding, data binding 애초에 뷰 덩어리 객체에 접근하기에 다른거에 접근못함

null safe

그리고 애초에 타입이 정해져서옴 -> type safe함 이런걸 강제 캐스팅하는놈은 이상한 놈이고

databinding이 사실 더 먼저나옴

근데 왜 두개 분리했을까?

-> 데이터 바인딩 최초에 나왔는데 왜 뷰바인딩 또나옴?

레이아웃으로 감싸줘야해서 귀찮

또차이 xml에 코드 못넣는거 뷰바인딩은 그냥 코드단으로 가져오는 뷰번들일 뿐임

데이터 바인딩 안에 뷰바인딩 기능이있는거 근데 역순이니까 단순한 라이트 버전이 나온거임

무거우니까

구글 공식문서에 그래서 둘다 적절히 사용하란다

데이터바인딩 실제 구현체

코틀린 어노테이션 프로세서가 열심히만듬

여기에 구현체있음

xml 만 단순 만들어도 바인딩 객체 바로 만들어져있음

-> 그니까 그냥 item 같은것도 바로바로 객체가 만들어지는것임

바인딩을 만들어내는 방법이 많다

-> 바인딩 객체 만드는방법

layoutinflater는 AppCompatActivity()

에 들어있음

inflater는 자바코드가아님 xml을 뷰로 만들어야하니까 핸드폰은 jvm이니까

xml을 파싱해서 쓰는거임 -> 그게 inflate하는거임

inlater안에 xml파서가 있음 ㄹㅇ 파서가 죄다 만들어버림

데이터 바인딩 variable 넣는방법 두가지임

binding.변수명 = 변수

binding.

BR 써서 하는것-> 다시보자 강의

데이터바인딩 주의점 int타입은 바인딩 해줄때 다 string로 바꿔서 넣어줘야함

이유 -> text 세팅할때 @string/어쩌구 이런거 넣을수있음 이거 int 값임 이런 리소스 아이디

그래서 int넣으면 리소스 값으로 인식해서 리소스가서 엄한거 찾고 없다고 난리임

그래서 숫자 값넣고 싶으면 어캐하는데? ->

1.문자열로 변환해서 바꿈

gmt로 데이터 나오는거 바인딩 어댑터로 깍아서 하겠다함

xml 에서는 이상한값 예를 들어 내가 이름지정해놓고 바인딩어댑터 같은거 안해놓은거 이런경우

그냥 무시하고 파싱자체를 안함

앞에 app써주는게 정석임

그리고

바인딩 어댑터 이름에 “app:이름" 이렇게 설정해도됨 app붙일꺼면 이렇게하지뭐

nullable하게 변수받아줌 데이터 바인딩에서

clean project가 kotlin kapt가만든 java 데이터 바인딩 같은거 만든거 다 삭제해버리는거였음

-> 그래서 어짜피 리빌드 했어야함

바인딩 어댑더

애초에 그냥 파일로 만들어야함 static해야해서

-> 그래서 object나 companion object에 해도 사실 이건 진짜 자바에서 static가 아님

그래서 @jvmstatic 어노테이션을 붙여주는거임