

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

06차 안드로이드 세미나 More! WriteActivity.kt

00 WriteActivity.kt 파헤치기

WriteActivity를 파헤치기 전, 왼쪽에는 PPT, 오른쪽에는 전체 코드를 켜주세요!

The image displays a side-by-side comparison of a presentation slide and its corresponding Android code. On the left, a PowerPoint slide titled '00 WriteActivity.kt 파헤치기' (Dissecting WriteActivity.kt) explains the purpose of the code: to demonstrate how to request permissions and use callbacks. It includes a list of bullet points and a code snippet for requesting storage permissions. On the right, the Android Studio interface shows the full Kotlin code for WriteActivity.kt, which implements the logic described in the slide, including permission requests and network calls.

00 WriteActivity.kt 파헤치기

게시를 쓰기 위해 통신을 좀 더 알아보고, 권한 요청 로직도 이해해 봅시다!

- “앨범에서 찾기” 버튼을 눌렀을 때, 세미나 때에는 곧바로 앨범이 열리는 로직이었지만, 지금은 권한 요청에 대한 메시지를 띄우고, 권한을 OK한다면 앨범이 띄워지는 로직으로 바뀌볼 것 입니다!
- 그 첫번째로 “앨범에서 찾기” 버튼 클릭 리스너 블록을 아래와 같은 함수(requestReadExternalStoragePermission()) 호출로 바꿔줍시다!

```
private fun setBtnClickListener() {  
    //앨범 열기 버튼 리스너  
    btn_write_act_show_album.setOnClickListener { it: View! -> {  
        //바로 앨범을 열지 않고,  
        //권한 허용을 확인 한 뒤 앨범을 열도록 하는 메소드를 호출합니다.  
        requestReadExternalStoragePermission()  
    }  
    //글쓰기 완료 버튼 리스너  
    btn_write_act_complete.setOnClickListener { it: View! -> {  
        //게시글 쓰기 서버 통신 관련 메소드를 호출합니다!  
        getWriteBoard@response()  
    }  
}
```

```
class WriteActivity : AppCompatActivity() {  
    val REQUEST_CODE_SELECT_IMAGE: Int = 1004  
    val My_READ_STORAGE_REQUEST_CODE = 7777  
  
    var imageURI : String? = null  
  
    // lazy를 안쓰고 인스턴스 상수를 생성한다면 아래와 같은 상수 선언과 같습니다!  
    // val networkService : NetworkService = ApplicationController.instance.networkService  
    val networkService: NetworkService by lazy {  
        ApplicationController.instance.networkService  
    }  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_write)  
        setOnBtnClickListener()  
    }  
  
    private fun setOnBtnClickListener() {  
        //앨범 열기 버튼 리스너  
        btn_write_act_show_album.setOnClickListener { it: View! -> {  
            //바로 앨범을 열지 않고,  
            //권한 허용을 확인 한 뒤 앨범을 열도록 하는 메소드를 호출합니다.  
            requestReadExternalStoragePermission()  
        }  
        //글쓰기 완료 버튼 리스너  
        btn_write_act_complete.setOnClickListener { it: View! -> {  
            //게시글 쓰기 서버 통신 관련 메소드를 호출합니다!  
            getWriteBoardResponse()  
        }  
    }  
  
    //앨범을 여는 메소드입니다!  
    //앨범에서 사진을 선택한 결과를 받기 위해 startActivityForResult를 통해 앨범 액티비티를  
    startActivity(Intent(this, AlbumActivity::class.java))  
}
```

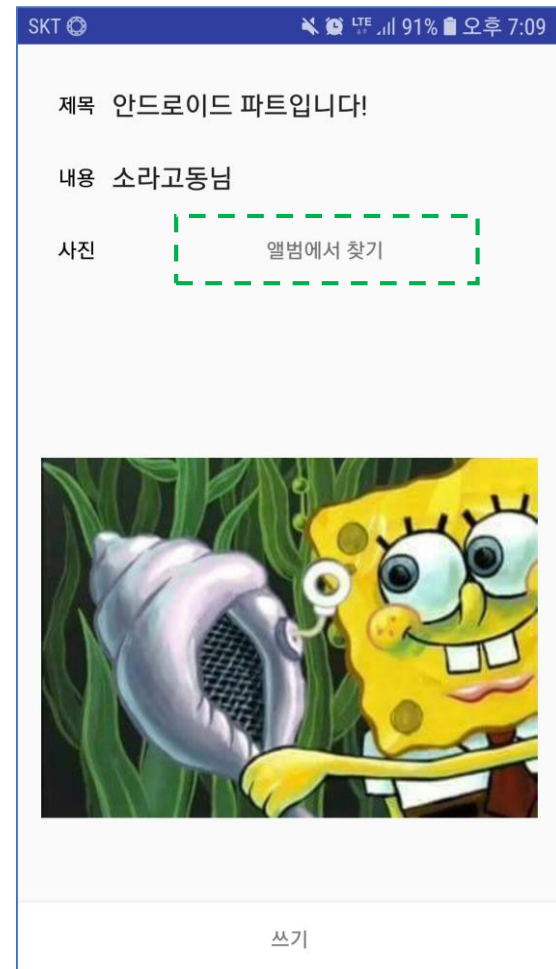
왼쪽은 PPT 오른쪽이 안드로이드 스튜디오에서 전체 코드 입니다!!

함수들도 많고 callback 함수도 많기 때문에 둘다 켜 놓고 이해하는게 좋아요!! 왔다 갔다 이동하기 때문에!!!

게시물 쓰기를 통해 통신을 좀 더 알아보고, 권한 요청 로직도 이해해 봅시다!

- “앨범에서 찾기” 버튼을 눌렀을 때, 세미나 때에는 곧바로 앨범이 열리는 로직이었지만, 지금은 권한 요청에 대한 메시지를 띄우고, 권한을 OK한다면 앨범이 띄워지는 로직으로 바꿔볼 것 입니다!
- 그 첫번째로 “앨범에서 찾기” 버튼 클릭 리스터 블록을 아래와 같은 함수(`requestReadExternalStoragePermission()`) 호출로 바꿔줍니다!

```
private fun setOnBtnClickListener() {  
    //앨범 열기 버튼 리스너  
    btn_write_act_show_album.setOnClickListener {  
        //바로 앨범을 열지 않고,  
        //권한 허용을 확인 한 뒤 앨범을 열도록 하는 메소드를 호출합니다.  
        requestReadExternalStoragePermission()  
    }  
    //글쓰기 완료 버튼 리스너  
    btn_write_act_complete.setOnClickListener {  
        //게시물 쓰기 서버 통신 관련 메소드를 호출합니다!  
        getWriteBoardResponse()  
    }  
}
```



requestReadExternalStoragePermission() 메소드 내부 로직 알아보기

- 해당 메소드는 곧바로 앨범을 열지 않고, 과거에 권한 요청 메시지에 대해 허용을 했는지 안 했는지를 체크합니다.
 - 1 경우 1) 허용을 했더라면 곧바로 showAlbum()을 통해 앨범을 열어줍니다.
 - 2 경우 2) 허용 하지 않았더라면, 권한에 대한 허용을 요청하는 메시지를 띄웁니다.

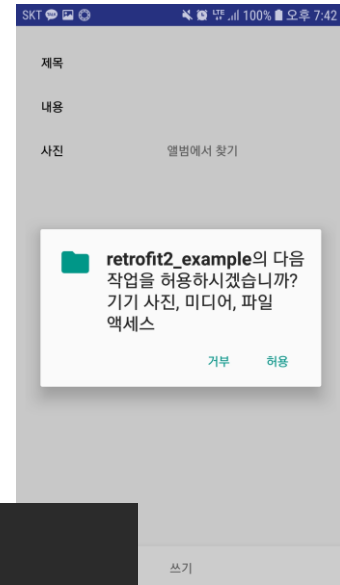
```
//이 메소드는 외부저장소(앨범과 같은)에 접근 관련해 권한 요청을 하는 로직을 메소드로 만든 것입니다!
private fun requestReadExternalStoragePermission(){
    //첫번째 if문을 통해 과거에 이미 권한 메시지에 대한 OK를 했는지 아닌지에 대해 물어봅니다!
    if(ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED){
        if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.READ_EXTERNAL_STORAGE)){
            //사용자에게 권한을 왜 허용해야되는지에 메시지를 주기 위한 대한 로직을 추가하려면 이 블록에서 하됩니다!!
            //하지만 우리는 그냥 비워놓습니다!! 딱히 줄말 없으면 비워놔도 무관해요!!! 굳이 뭐 안넣어도됩니다!
        } else {
            2 //아래 코드는 권한을 요청하는 메시지를 띄우는 기능을 합니다! 요청에 대한 결과는 callback으로 onRequestPermissionsResult 메소드에서 받습니다!!!
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE), My_READ_STORAGE_REQUEST_CODE)
        }
    } else {
        1 //첫번째 if 문의 else 로써, 기존에 이미 권한 메시지를 통해 권한을 허용했다면 아래와 같은 곧바로 앨범을 여는 메소드를 호출 해주면됩니다!!
        showAlbum()
    }
}
```

requestReadExternalStoragePermission()의 ② 경우에 대한 상황을 알아보시다.

- 우리는 아직 메시지를 통한 권한 요청에 허용한 적이 없기 때문에, 오른쪽과 같은 권한 허용 요청 메시지가 뜰 것 입니다! 우리가 요청한 권한은

`Manifest.permission.READ_EXTERNAL_STORAGE` 으로, 외부 저장소 읽기 관련 권한이므로 “**기기 사진, 미디어, 파일 액세스**”라는 메시지가 뜨는 것이고, 요청 권한에 따라 메시지 내용이 달라진답니다!!

- 그 다음은 “**허용/거부**” 버튼을 눌렀을 때 상황에 대해 코드를 보면서 알아보도록 하죠!



```
//외부저장소(앨범과 같은)에 접근 관련 요청에 대해 OK를 했는지 거부했는지를 callback으로 받는 메소드입니다!
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {
    //onActivityResult와 같은 개념입니다. requestCode로 어떤 권한에 대한 callback인지를 체크합니다.
    if (requestCode == My_READ_STORAGE_REQUEST_CODE){
        //결과에 대해 허용을 눌렀는지 체크하는 조건문이구요!
        if(grantResults.size > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED){
            2-1 //이곳은 외부저장소 접근을 허용했을 때에 대한 로직을 쓰시면됩니다. 우리는 앨범을 여는 메소드를 호출해주면되겠죠?
            showAlbum()
        } else {
            2-2 //이곳은 외부저장소 접근 거부를 했을때에 대한 로직을 넣어주시면 됩니다.
            finish()
        }
    }
}
```

알아볼 것도 없네요! if(조건문)을 통해 허용을 했더라면 2-1 블록이 수행되고,
거부를 했다면 2-2 블록이 수행됩니다! 허용을 했으면 showAlbum() 호출하면 끝!

이어서은 showAlbum()을 통해 앨범을 열고, 사진을 선택했을 때 상황을 알아봅시다!

```
//앨범을 여는 메소드입니다!  
//앨범에서 사진을 선택한 결과를 받기위해 startActivityForResult를 통해 앨범 액티비티를 열어요!  
private fun showAlbum(){  
    val intent = Intent(Intent.ACTION_PICK)  
    intent.type = android.provider.MediaStore.Images.Media.CONTENT_TYPE  
    intent.data = android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI  
    startActivityForResult(intent, REQUEST_CODE_SELECT_IMAGE)  
}
```

- showAlbum() 메소드에 대한 내용은 아래와 같습니다. 그냥 앨범을 띄워주는 역할이예요!!

중요하게 체크할 점은 startActivityForResult(intent, REQUEST_CODE_SELECT_IMAGE) 으로 띄워줬기 때문에

사진 선택에 대한 결과를 callback 받기 위해 override fun onActivityResult() 를 override 해줘야 한다는 점!!

```
//startActivityForResult를 통해 실행한 액티비티에 대한 callback을 처리하는 메소드입니다!
```

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
```

```
    super.onActivityResult(requestCode, resultCode, data)
```

```
    //REQUEST_CODE_SELECT_IMAGE를 통해 앨범에서 보낸 요청에 대한 Callback인지를 체크!!!
```

```
    if (requestCode == REQUEST_CODE_SELECT_IMAGE) {
```

```
        if (resultCode == Activity.RESULT_OK) {
```

```
            //data.data에는 앨범에서 선택한 사진의 Uri가 들어있습니다!! 그러니까 제대로 선택됐는지 null인지 아닌지를 체크!!!
```

```
            if(data != null){
```

```
                val selectedImageUri : Uri = data.data
```

```
                //Uri를 getRealPathFromURI라는 메소드를 통해 절대 경로를 알아내고, 인스턴스 변수 imageURI에 넣어줍니다!
```

```
                imageURI = getRealPathFromURI(selectedImageUri)
```

```
                Glide.with(this@WriteActivity)
```

```
                    .load(selectedImageUri)
```

```
                    .thumbnail(0.1f)
```

```
                    .into(iv_write_act_choice_image)
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

이 메소드에 대해서는 너무 많이 언급했기 때문에!!
이제는 감이 좀 잡혀야 해요!!! 엉엉...

사진 선택에 대한 결과로 사진에 대한 Uri를 얻습니다!
그 Uri를 이용해서 2가지 작업을 해줍니다.

첫번째, 인스턴스 변수 imageURI에 절대 경로 담기
두번째, Glide를 통해 ImageView에 띄워주기

00 WriteActivity.kt 파헤치기

앞장 피피티 이어서 설명..

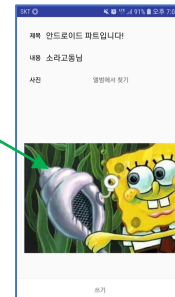
```
//startActivityResult를 통해 실행한 액티비티에 대한 callback을 처리하는 메소드입니다!
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    //REQUEST_CODE_SELECT_IMAGE를 통해 앨범에서 보낸 요청에 대한 Callback인지를 체크!!!
    if (requestCode == REQUEST_CODE_SELECT_IMAGE) {
        if (resultCode == Activity.RESULT_OK) {
            //data.data에는 앨범에서 선택한 사진의 Uri가 들어있습니다!! 그러니까 제대로 선택했는지 null인지 아닌지를 체크!!!
            if (data != null) {
                val selectedImageUri : Uri = data.data
                //Uri를 getRealPathFromURI라는 메소드를 통해 절대 경로를 알아내고, 인스턴스 변수 imageURI에 넣어줍니다!
                imageURI = getRealPathFromURI(selectedImageUri)

                Glide.with(this@WriteActivity)
                    .load(selectedImageUri)
                    .thumbnail(0.1f)
                    .into(iv_write_act_choice_image)
            }
        }
    }
}
```

첫번째, 인스턴스 변수 imageURI에 절대 경로 담기
두번째, Glide를 통해 ImageView에 띄워주기

- 첫번째에 대한 작업을 왜 하나 면! 다른 함수에서 이 변수를 참조할 것이기 때문에 인스턴스 변수로 담아준 것 입니다!
- 두번째에 대한 작업을 왜 하나 면! 우리가 무슨 사진을 선택했는지 눈으로 보기 위해서!!!
아참! 그리고 ImageView에 대한 옵션 좀 바꿔줄게요!! scaleType을 fitCenter로 해주세요!

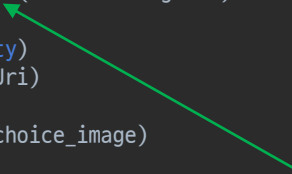
```
<ImageView
    android:id="@+id/iv_write_act_choice_image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitCenter" />
```



앞장 피피티 또 이어서 설명..

```
//startActivityResult를 통해 실행한 액티비티에 대한 callback을 처리하는 메소드입니다!
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    //REQUEST_CODE_SELECT_IMAGE를 통해 앨범에서 보낸 요청에 대한 Callback인지를 체크!!!
    if (requestCode == REQUEST_CODE_SELECT_IMAGE) {
        if (resultCode == Activity.RESULT_OK) {
            //data.data에는 앨범에서 선택한 사진의 Uri가 들어있습니다!! 그러니까 제대로 선택됐는지 null인지 아닌지를 체크!!!
            if(data != null){
                val selectedImageUri : Uri = data.data
                //Uri를 getRealPathFromURI라는 메소드를 통해 절대 경로를 알아내고, 인스턴스 변수 imageURI에 넣어줍니다!
                imageURI = getRealPathFromURI(selectedImageUri)

                Glide.with(this@WriteActivity)
                    .load(selectedImageUri)
                    .thumbnail(0.1f)
                    .into(iv_write_act_choice_image)
            }
        }
    }
}
```



getRealPathFromURI(Uri넣기)

- getRealPathFromURI()메소드의 경우는 이미지 절대경로를 파악하기 위한 메소드예요!!
코드에 대한 이해를 하지 마시고, 이번 예제 뿐만 아니라 절대 경로를 알아야하는 상황이 생기면 이 코드를 복붙해서 쓰시면 됩니다!! 이것을 이해할 필요 없어요!! 그냥 Uri를 메개변수로 넣으면 String으로 절대경로가 return 된다는 함수!!

```
//Uri에 대한 절대 경로를 리턴하는 메소드입니다! 굳이 코드를 해석하려고 하지 말고,  
//앱잼때 코드를 복붙을 통해 재사용해주세요!!  
fun getRealPathFromURI(content : Uri) : String {  
    val proj = arrayOf(MediaStore.Images.Media.DATA)  
    val loader : CursorLoader = CursorLoader(this, content, proj, null, null, null)  
    val cursor : Cursor = loader.loadInBackground()  
    val column_idx = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA)  
    cursor.moveToFirst()  
    val result = cursor.getString(column_idx)  
    cursor.close()  
    return result  
}
```


그 다음은!!! 드디어 통신을 하는 부분을 볼 것입니다!

- 앨범에서 **사진**을 선택, **제목**, **내용**에 문자열을 모두 입력했다면 이제 데이터들을 서버로 보낼 일만 남았어요!!
- WriteActivity.kt 코드 외 다른 통신 코드는 여기서 다루지 않을 거예요!
6차 세미나 PPT에 충분히 설명됐기 때문에!! 여기서는 비동기로 이뤄지는 통신에 대해 통신을 담당하는 메소드에 넣어야 될 코드 위주로 설명할게요!
- 양식을 다 채웠다면 “**쓰기**” 버튼을 눌렀을 때 상황을 알아보시다!!

```
private fun setOnBtnClickListener() {  
    //앨범 열기 버튼 리스너  
    btn_write_act_show_album.setOnClickListener {  
        //바로 앨범을 열지 않고,  
        //권한 허용을 확인 한 뒤 앨범을 열도록 하는 메소드를 호출합니다.  
        requestReadExternalStoragePermission()  
    }  
    //글쓰기 완료 버튼 리스너  
    btn_write_act_complete.setOnClickListener {  
        //게시물 쓰기 서버 통신 관련 메소드를 호출합니다!  
        getWriteBoardResponse()  
    }  
}
```



getWriteBoardResponse() : 통신 관련 처리를 모아둔 함수를 알아보자!

- 1~4번으로 구분 지어서 설명하겠습니다!

```
private fun getWriteBoardResponse() {  
    val input_title = et_write_act_title.text.toString()  
    val input_contents = et_write_act_content.text.toString()  
    if (input_title.isNotEmpty() && input_contents.isNotEmpty()) {  
        //Multipart 형식은 String을 RequestBody 타입으로 바꿔줘야 합니다!  
        val token = SharedPreferencesController.getAuthorization(this)  
        val title = RequestBody.create(MediaType.parse("text/plain"), input_title)  
        val contents = RequestBody.create(MediaType.parse("text/plain"), input_contents)  
  
        //아래 3줄의 코드가 이미지 파일을 서버로 보내기 위해 MultipartBody.Part 형식으로 만드는 로직입니다.  
        // imageURI는 앨범에서 선택한 이미지에 대한 절대 경로가 담겨있는 인스턴스 변수입니다.  
        val file : File = File(imageURI)  
        val requestfile : RequestBody = RequestBody.create(MediaType.parse("multipart/form-data"), file)  
        val data : MultipartBody.Part = MultipartBody.Part.createFormData("photo", file.name, requestfile)  
  
        val postWriteBoardResponse =  
            networkService.postWriteBoardResponse(token, title, contents, data)  
  
        postWriteBoardResponse.enqueue(object : Callback<PostWriteBoardResponse> {  
            override fun onFailure(call: Call<PostWriteBoardResponse>, t: Throwable) {  
                Log.e("write fail", t.toString())  
            }  
            override fun onResponse(call: Call<PostWriteBoardResponse>, response: Response<PostWriteBoardResponse>) {  
                if (response.isSuccessful) {  
                    toast(response.body()!!.message)  
                    finish()  
                }  
            }  
        })  
    }  
}
```

00 WriteActivity.kt 파헤치기

- 일단 @Multipart 가 무엇이나?!

서버가 multipart/form-data 으로 통신을 해달라고 했기 때문에 쓰는 겁니다!!! 우리가 마음대로 보내면 안되잖아요~!

- 일단 RequestBody가 무엇이나?! HTTP 통신 시 **자바 객체를 담을 수 있는 형식**으로 생각해주세요!!

서버로 전달하거나 전달 받기 위해 데이터를 가공하는 방식으로 생각해보시면 됩니다!! **너무 깊게 생각하지 마세요!**

```
1 val token = SharedPreferencesController.getAuthorization(this)
  var title = RequestBody.create(MediaType.parse("text/plain"), input_title)
  var contents = RequestBody.create(MediaType.parse("text/plain"), input_contents)
```

token은 header에 붙일 것이므로 String입니다.
body가 아닙니다!!

- 1 그냥 String으로 보내면 좋겠지만 **파일 전송이 들어있을 경우 @Multipart로 통신해야 하므로,**

RequestBody라는 형식으로 String을 가공해야 합니다! 이런 작업이 1 부분에서 이루어지는 것이구요!

(그냥 단순히 파일이 첨부되는 통신은 @Multipart로 한다고 생각하시고, 만약 다른 파일을 첨부하는 통신이 있다면 현재 이 코드들을 참조하여 다른 통신에 알맞게 바꿔서 사용하시면 됩니다!!! Int나 Double은 그냥 타입 그대로!)

```
2 //아래 3줄의 코드가 이미지 파일을 서버로 보내기 위해 MultipartBody.Part 형식으로 만드는 로직입니다.
// imageURI는 앨범에서 선택한 이미지에 대한 절대 경로가 담겨있는 인스턴스 변수입니다.
val file : File = File(imageURI)
val requestfile : RequestBody = RequestBody.create(MediaType.parse("multipart/form-data"), file)
val data : MultipartBody.Part = MultipartBody.Part.createFormData("photo", file.name, requestfile)
```

- 2 마찬가지로 @Mutipart 형식으로 통신하기 때문에 이런 과정을 거쳐야합니다! 하지만 여기서는

MultipartBody.Part라는 형식으로 한번 더 가공하는 것 뿐이에요! 파일은 이렇게 보낸다고 생각하고 사용하시면 됩니다!!! **data**라는 상수는 우리가 선택한 사진이라고 생각하시면 됩니다!!!!!!!!!!

앞선 피피티에 대한 코멘트

- 이것은 NetworkService에 명시한 게시글 글쓰기 추상 메소드로, 통신에 대한 형식(틀)을 갖춰 놓은 곳입니다.

```
//게시판 글쓰기
@Multipart
@POST("/contents")
fun postWriteBoardResponse(
    @Header("Authorization") token : String,
    @Part("title") title : RequestBody,
    @Part("contents") contents : RequestBody,
    @Part photo: MultipartBody.Part?
) : Call<PostWriteBoardResponse>
```

- 앞에 설명했듯 @Multipart, RequestBody, MultipartBody.Part 등의 낯선 것들을 사용하는 이유는 서버에서 이렇게 통신 해달라고 API에 명시했기 때문에 사용한 것입니다. 그렇기 때문에 앞으로 파일 전송이 들어간 통신 API이거나, multipart/form-data 라는 키워드가 보인다면 이런 방식으로 통신하면 됩니다!

이어서 ③, ④ 분석!

- 3,4는 앞서 가공한 데이터를 비동기식으로 서버로 보내고 Request에 대한 Response를 받는 로직입니다!
이부분이 통신을 이루는 코드예요!

```
③ val postWriteBoardResponse =  
    networkService.postWriteBoardResponse(token, title, contents, data)  
  
    postWriteBoardResponse.enqueue(object : Callback<PostWriteBoardResponse> {  
        ④ override fun onFailure(call: Call<PostWriteBoardResponse>, t: Throwable) {  
            Log.e("write fail", t.toString())  
        }  
        override fun onResponse(call: Call<PostWriteBoardResponse>, response: Response<PostWriteBoardResponse>) {  
            if (response.isSuccessful) {  
                toast(response.body()!!.message)  
                finish()  
            }  
        }  
    })
```

- 3,4은 GET방식이던 DELETE방식이던 POST던 PUT이던 **Retrofit2**라는 라이브러리를 쓴다면 무조건 공통적으로 반복되는 코드로 통신을 많이 하다 보면 눈에 매우 익혀질 겁니다!
- 3번의 경우 NetworkService에 명시한 추상메소드의 매개변수 순서대로 데이터를 알맞게 넣어주면 되구요!
- 4번의 경우 통신에 대한 Response를 받는 곳으로, if(response.isSuccessful)을 통해 통신이 성공적으로 이뤄졌는지 확인 후 내부 블록에 전달받은 데이터 바디(response.body())에서 데이터를 꺼내서 쓰면 됩니다!!!
위의 코드에서는 딱히 전달 받을 데이터가 없었기 때문에 message라는 데이터를 굳이 꺼내서 toast로 띄워준거예요!
- 다시한번 강조하는 것이지만!! 우리는 통신 자체보다 데이터를 어떻게 View에 뿌려주는지가 중요한 것입니다!! 잊지마요!

이어서 4 분석! more

```
postWriteBoardResponse.enqueue(object : Callback<PostWriteBoardResponse> {  
    override fun onFailure(call: Call<PostWriteBoardResponse>, t: Throwable) {  
        Log.e("write fail", t.toString())  
    }  
    override fun onResponse(call: Call<PostWriteBoardResponse>, response: Response<PostWriteBoardResponse>) {  
        if (response.isSuccessful) {  
            toast(response.body()!!.message)  
            finish()  
        }  
    }  
})
```

- 위의 초록색 박스는 서버로 보낸 요청에 대한 응답이 이루어지는 부분입니다. 즉, callback을 처리하는 부분이죠.
통신 후 이어질 로직들은 저 초록 박스 내 if(response.isSuccessful) 조건문 안에 다 넣으면 됩니다!!! 쉽죠?!
- 비교적 게시판 글쓰기와 같은 통신에서는 전달받을 데이터가 딱히 없으므로 위와 같은 단순한 로직일 겁니다!
하지만 GET 방식과 같은, 다양한 데이터를 요청하는 대한 통신의 경우는 우리가 View에 뿌려줘야 할 데이터들이 매우 많아요!
즉, 통신 후 이어질 작업이 많다는 것이죠! 그럼 초록색 블록에 들어갈 코드도 많아 질 것이고.. 그러니까 여기서 드리고 싶은 팁이 있는데, 최대한 코드를 함수로 묶어서 보기 좋게 관리하라는 점입니다!!! 여러분 파이팅!!! 보충 자료 끝!

S H O U T · O U R · P A S S I O N · T O G E T H E R

ODo IT SOPT O

THANK U

세미나 자료 남윤환

SHOUT OUR PASSION TOGETHER
SOPT